Model Interface Package for Simulink

Modeling Guide

For Model Interface Package for Simulink 4.5

Release 2021-A – May 2021

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

This publication and the contents hereof are subject to change without notice.

Contents

Generating Simulink Implementation Containers 261

Information for Former RTI Users 275

Limitations 281

Model Interface Package for Simulink Glossary 293

Index 299

About This Guide

Content

This guide shows you how to work with the Model Interface Package for Simulink. The guide focuses on the following main topics:

- Introduction to the Model Interface Package for Simulink
- Using behavior models with ConfigurationDesk
- Specifying the interface of behavior models
- Generating code container files for ConfigurationDesk and VEOS Player
- Information on the build process and the simulation behavior of your models in ConfigurationDesk

Required knowledge

Knowledge in handling MATLAB/Simulink is assumed.

Target group

This guide is primarily targeted at engineers who implement Simulink models and build code using the Simulink® Coder™.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
⚠ CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.

Symbol	Description
?	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
📖	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the 📄 icon in dSPACE Help. The PDF opens on the first page.

Introduction to the Model Interface Package for Simulink

Use scenarios

The Model Interface Package for Simulink lets you:

- Specify the interface of a Simulink behavior model for the simulation in ConfigurationDesk and VEOS Player.
- Generate a container file containing the behavior model code. You can use the container file in ConfigurationDesk and in VEOS Player.

> **Note**
>
> To generate a Simulink implementation container file, or to start model code generation in ConfigurationDesk, you need a Simulink® Coder™ license.

Features of the Model Interface Package for Simulink

Purpose of the Model Interface Package for Simulink

The Model Interface Package for Simulink lets you specify the interface of Simulink models that you can use in ConfigurationDesk or VEOS Player.

There are two ways to use Simulink models in ConfigurationDesk or VEOS Player:

1. Adding the Simulink model directly to a ConfigurationDesk application

   This approach lets you generate model port blocks directly into the Simulink model or make changes in the Simulink model known to ConfigurationDesk via model analysis. This simplifies the simultaneous development of the Simulink behavior model and the I/O functionality in ConfigurationDesk.
2. Generating a Simulink implementation container (SIC file)

   In Simulink, you can start model code generation for your Simulink model and thus, generate an SIC file. You can add SIC files to ConfigurationDesk or

VEOS Player. A MATLAB installation is not required for the build process in ConfigurationDesk.

Note

To generate SIC files, you have to specify a suitable dSPACE Run-Time Target as the system target file. Refer to the following table:

Use Scenario	Required System Target File
SIC files for VEOS on Windows and for ConfigurationDesk	`dsrt.tlc`
SIC files for VEOS on Linux	`dsrt64.tlc`

In ConfigurationDesk and VEOS Player, you can connect the model ports of the behavior model to ports of other models to implement communication between them. In ConfigurationDesk, you can also map the model ports to function ports.

Benefits of Simulink implementation containers

With a Simulink implementation container file (SIC file), you can separate the model code generation from the build process. This approach has the following benefits:

- You can use the same system target file for Simulink behavior models in VEOS Player (running on Windows) and in ConfigurationDesk.
- A MATLAB installation is necessary only for building SIC files. For using of SIC files in ConfigurationDesk or VEOS Player, a MATLAB installation is not required.
- Executable applications can contain Simulink implementation containers from different MathWorks Releases and different dSPACE Releases.
- In ConfigurationDesk, you can precompile an SIC file to save time in the build process. Optionally, you can create precompiled SIC files without readable source files. This can be useful for IP protection.

Specifying the interface of a behavior model

The Model Interface Blockset provides blocks for defining the interface of a Simulink behavior model. The interface includes input and output signals as well as runnable functions. Runnable functions are exported from function-call subsystems, and can be used for modeling asynchronous tasks (ConfigurationDesk only). Runnable functions exported in predefined tasks can be used in ConfigurationDesk and VEOS Player.

For details on specifying the model interface of a behavior model, refer to Specifying the Interface of Behavior Models on page 49.

Separating models from an overall model

The Model Interface Blockset contains the **Model Separation** tool, which separates individual models from an overall model in MATLAB/Simulink. In ConfigurationDesk you can use the output of the model separation to build a multicore real-time application or a multi-processing-unit application and download it to the dSPACE real-time hardware, where the separated models are executed in parallel on the single cores of a single processing-unit or a multi-processing-unit system. Model separation also creates a model communication description file (MCD file), which describes the signal flow between the separated models.

Note

With the Model Interface Package for Simulink, you cannot generate Simulink implementation containers (SIC files) for the overall model and for the separated models, and use them in ConfigurationDesk or VEOS Player. If you work with SIC files, you are not recommended to use the model communication description file (MCD file) with ConfigurationDesk, because the MCD file references the separated Simulink models, and not the generated SIC files. The MCD file cannot be used in VEOS Player.

Generating a Simulink implementation container

The dSPACE Run-Time Targets are the system target files for which you have to generate a Simulink implementation container file (SIC file). You can add SIC files to ConfigurationDesk or import them into VEOS Player. For details, refer to Generating Simulink Implementation Containers on page 261. The `dsrt.tlc` system target file is also used when you start model code generation in ConfigurationDesk.

Display of messages

When you work with the Model Interface Package for Simulink, messages inform you about possible problems or upcoming actions. The messages are displayed in the MATLAB Command Window or in dialogs. They provide links or buttons for you to open the related topic in the Model Interface Package for Simulink Message Reference, which contains more information. The messages can also contain links to blocks or models that triggered the message.

Using Simulink Models With ConfigurationDesk

Introduction

To familiarize you with the concept of using Simulink behavior models with ConfigurationDesk.

Where to go from here

Information in this section

Modeling Concept

Introduction

If you create behavior models that you want to use with ConfigurationDesk, the following modeling concept will help you to build real-time applications more flexibly and efficiently.

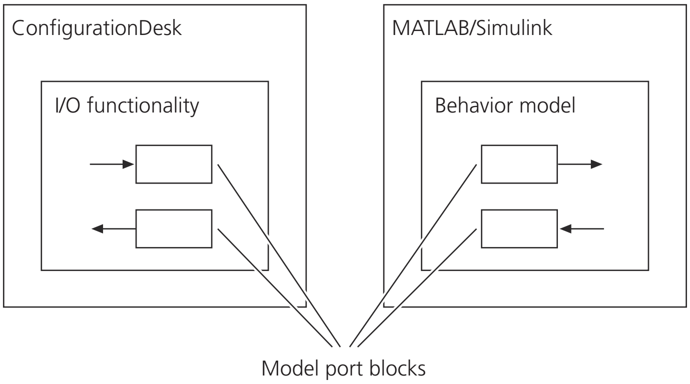
Where to go from here

Information in this section

Introduction to the Modeling Concept

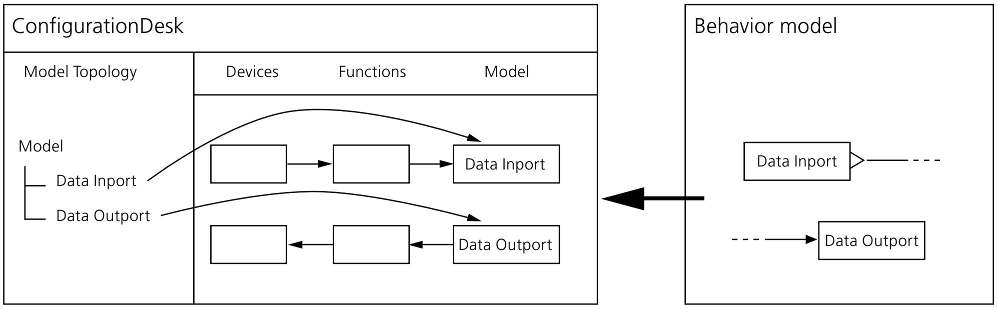
Creating models for ConfigurationDesk

ConfigurationDesk works with MATLAB®/Simulink® as the modeling tool. The real-time model on which your real-time application is based is strictly subdivided into the I/O functionality in ConfigurationDesk and the behavior model in MATLAB/Simulink. The behavior model does not contain I/O functionality. The functions for measuring and generating I/O signals are specified in ConfigurationDesk. The interface between the behavior model in Simulink and the I/O functionality in ConfigurationDesk is implemented by model port blocks. The Model Interface Blockset provides blocks for data inputs, data outputs and runnable functions for creating the model interface in your Simulink behavior model. The data port blocks (**Data Inport** blocks and **Data Outport** blocks) implement the data exchange between ConfigurationDesk and MATLAB/Simulink. The runnable functions provided by a behavior model are functions that must be called within tasks for computing results. They are available in ConfigurationDesk after a model analysis. You can then assign them to periodic or asynchronous tasks. As an alternative, the runnable functions provided by the behavior model can be exported as predefined tasks.

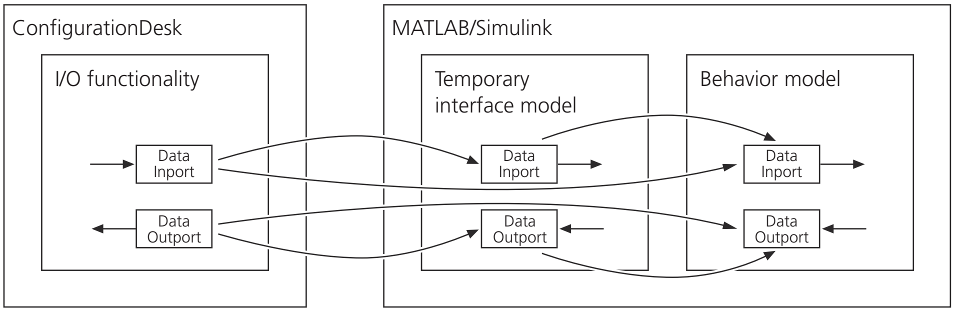


The model port blocks can be created in ConfigurationDesk or in the Simulink model.

Model port blocks created in the Simulink model If the new model port blocks come from your behavior model, they are imported to the model topology in ConfigurationDesk, where you can map them to function blocks. The model topology in ConfigurationDesk displays the structure of the referenced behavior model that is relevant for the model interface including the root model with its subsystems, and all available model port blocks.



Model port blocks created in ConfigurationDesk If the new model port blocks come from ConfigurationDesk's I/O functionality, they can be exported to an interface model in MATLAB/Simulink, where you can copy them to your behavior model. They can also be generated directly into the behavior model.



The behavior model that you want to use with ConfigurationDesk must be added to the ConfigurationDesk application. Because the behavior model and the I/O functionality are separate, you can replace one behavior model with another without modifications to the I/O functionality, provided that the behavior model uses the same model interface.

Block identification

Data port blocks and their ports provide unique identifiers (IDs). The IDs are defined when you copy a data port block from the Model Interface Blockset to a behavior model. If you want to take a data port block from one model (such as the interface model) and use it in another model (such as the behavior model), you must use a special paste command to keep its IDs. For example, you can create different behavior models with an identical model interface in this way.

Note

- If you use the standard copy & paste commands, the inserted blocks are given new IDs. They are treated as new model port blocks.
- Hardware-Triggered Runnable Function blocks and Software-Triggered Runnable Function blocks must have unique function names in the behavior model.
- Model Port blocks that are part of a library block have unique block and signal IDs. If you copy the library block and the contained model port blocks, the model port blocks in the copy receive new unique block and signal IDs.

Supported blocksets

Behavior models used with ConfigurationDesk can contain the following dSPACE blocksets:

- *dSPACE RTI CAN MultiMessage Blockset* to include CAN message handling in your model.
- *dSPACE RTI LIN MultiMessage Blockset* to include LIN message handling in your model.
- *dSPACE FlexRay Configuration Blockset* to include FlexRay message handling in your model.
- *dSPACE Automotive Simulation Models* to simulate of automotive engines (gasoline and Diesel) and vehicle dynamics.
- *dSPACE MotionDesk Blockset* to animate your simulation results in 3-D.

Note

If your behavior model contains blocks from the dSPACE RTI CAN MultiMessage Blockset, the dSPACE LIN MultiMessage Blockset, or the FlexRay Configuration Blockset, you cannot generate a Simulink implementation container for it. However, you can use such models in ConfigurationDesk by adding them directly to a ConfigurationDesk application.

Task creation

The behavior model that you create can include several tasks, runnable functions, and events depending on your model architecture. The number of periodic tasks in your behavior model depends on the number of different sample times and the tasking mode used (**Treat each discrete rate as a separate task** checkbox in Simulink's **Configuration Parameters** dialog). Model port blocks for data inports and data outports can be used in an existing task, but they can also be responsible for new periodic tasks, if you configure them with different sample times.

Build process

When you have completed the implementation, you can start the build process directly in ConfigurationDesk, or via the **ConfigurationDesk** menu in the Simulink behavior model. ConfigurationDesk controls the process and starts Simulink® Coder^{TM} to generate the code for the behavior model. After the code for all the components belonging to the executable application has been generated, ConfigurationDesk builds the real-time application and downloads it to the hardware.

Simulation and execution

The real-time application can be executed in real time on the connected dSPACE hardware. In Simulink, the behavior model can be executed in Simulink Normal simulation mode, Simulink Accelerator simulation mode, and Simulink Rapid Accelerator simulation mode. In Simulink simulation mode, the simulation is independent of the connected ConfigurationDesk application. For this reason, the model ports are without functionality, and do not reflect the actual signals of the application. However, they must match the signals' data types and widths that are specified in the behavior model.

Migration

If you previously used your model with an RTI platform, or if you want to switch your Simulink model between an RTI and a ConfigurationDesk environment, you must know the main differences between their modeling concepts, and how to migrate your models. Refer to Information for Former RTI Users on page 275.

Further information

For further information on the above features, you can follow the links given in Related Topics. For general information on ConfigurationDesk, refer to ConfigurationDesk Real-Time Implementation Guide .

Related topics

Basics

SCALEXIO – Hardware and Software Overview
Working with Runnable Functions and Function Triggers..82

Modeling Workflow

Introduction

There are three ways of implementing a Simulink behavior model for ConfigurationDesk:

- Starting with a behavior model in Simulink without model port blocks.
- Starting with the I/O functionality in ConfigurationDesk.
- Starting with the creation of a new Simulink behavior model including model port blocks.

Starting with a behavior model in Simulink without model port blocks

To start with a behavior model in Simulink without model port blocks, you have to perform the following steps:

1. Create a new ConfigurationDesk project from the Simulink behavior model.
2. In the **Model-Function Mapping Browser** in ConfigurationDesk, add I/O functionality and suitable model port blocks to the ConfigurationDesk application.
3. In ConfigurationDesk, propagate the model port blocks to the Simulink behavior model.
4. In the Simulink behavior model, connect the model port blocks to the respective signals.

Starting with the I/O functionality in ConfigurationDesk

To start with the I/O functionality in ConfigurationDesk, you have to perform the following steps:

1. Add a Simulink behavior model to the ConfigurationDesk application.
2. Specify an external device interface and create a hardware topology, if required.
3. Add I/O functionality to the signal chain.
4. Extend the signal chain with suitable model port blocks.
5. Generate a temporary interface model containing blocks from the Model Interface Blockset.
6. Use the standard Copy command together with the **Paste and Keep IDs** command to copy the model port blocks from the interface model to the Simulink behavior model.

> **Note**
>
> Instead of steps 3 - 6, you can also add I/O functionality including suitable model port blocks to the signal chain via the Model-Function Mapping Browser. Subsequently, you can propagate the model port blocks directly to the Simulink behavior model.

7. In the Simulink behavior model, connect the model port blocks to the respective signals.

Starting with the creation of a new Simulink behavior model including model port blocks

To start with a new behavior model in Simulink, you have to perform the following steps:

1. Open a new Simulink model and implement your algorithms using blocks from the supported Simulink and dSPACE blocksets.

> **Note**
>
> The Model Interface Package for Simulink provides the DSRT template for Simulink models that are preconfigured for dSPACE Run-Time Target. You can create new Simulink models on the basis of the template that is accessible via the Simulink start page.

2. Add blocks from the Model Interface Blockset to your behavior model to provide the model interface to ConfigurationDesk.
3. Add the behavior model to the ConfigurationDesk application. By default, the model interface and the task information of the behavior model is analyzed automatically. ConfigurationDesk then contains a model interface representing the structure of the Simulink behavior model.
4. In ConfigurationDesk, add the model port blocks to the signal chain and map them to function blocks.

> **Note**
>
> ConfigurationDesk provides propagate operations that configure the model port blocks according to the configuration of the function blocks to which they are mapped. To make sure that model port blocks are not accidentally changed by a propagate operation, you can place them in subsystems in the Simulink model that are set to read-only. When analyzed, these subsystems are marked with a lock symbol in ConfigurationDesk.

5. If necessary, configure tasks and model further aspects of your application.

Using Simulink models containing RTI blocks

If you want to work with Simulink models that contain RTI blocks, you have to migrate these models before you can use them in ConfigurationDesk. Refer to Migrating Models on page 278. After you migrate these models, you can use them in a ConfigurationDesk application according to one of the workflows described above.

Keeping the interfaces synchronized

To keep the model interfaces in Simulink and in ConfigurationDesk synchronized, you must propagate the following modifications that you made in one tool to the other tool:

- If you add model port blocks to your Simulink behavior model or modify the structure of the model, you must analyze the model in ConfigurationDesk to make the changes available in the ConfigurationDesk model interface.
- If you delete model ports or model port blocks in the behavior model that are used in the signal chain in ConfigurationDesk, they are displayed as unresolved after a model analysis. You can then delete them from ConfigurationDesk.

> **Tip**
>
> The Model Interface Package for Simulink provides the **Delete Selection and Connected Blocks** to delete a model port block in the Simulink behavior model, the related model port block in ConfigurationDesk, and the connected function blocks in one step.

- If you add model port blocks to the ConfigurationDesk application by extending the signal chain, you can propagate the new model port blocks from ConfigurationDesk to the Simulink behavior model. Alternatively, you can use the **Generate New Simulink Model Interface** command to create a new interface model containing the new model port blocks. The new model port blocks must then be copied to your behavior model.

> **Note**
>
> If you want to copy a model port block from the interface model to the behavior model, you must use the standard Copy command together with the **Paste and Keep IDs** command provided by the Model Interface Blockset to keep the unique identifiers of the model port block and its signals. For further information, refer to Basics on Model Port Block IDs and Signal IDs on page 63.

If you want to see all the modifications in your behavior model's interface also in the model topology, you must start the model analysis feature in ConfigurationDesk. The model analysis is automatically done when you start a build process.

Related topics

Basics

Migrating Models..278

Information on the Floating-Point Support on dSPACE Products

Introduction

Floating-point arithmetic is an integral part of almost any behavior model. While common implementations typically aim for optimal precision, real-time scenarios must provide maximum computing performance and deterministic execution time. Therefore, certain limitations take place which might slightly reduce precision. The standard implementation presented in dSPACE products offers a good compromise that is suitable for the majority of applications. However, there might be exceptional cases where some tweaking might be necessary.

Scope

This information applies to the following dSPACE products:

- SCALEXIO
- MicroAutoBox III
- MicroLabBox
- DS1007

For information on other products, contact dSPACE Support.

Prerequisites

When you read this information, note the following prerequisites:

- For the sake of readability, this information focuses on non-negative numbers. The handling of negative numbers is mostly symmetrical to the handling of positive numbers.
- By default, the dSPACE floating-point implementation generally conforms to IEEE 754 using 64-bit double precision arithmetic with the following limitations.

Range

The absolute range of numbers that can be represented in IEEE 754 at full precision of 15-17 decimal digits is between 10^{-308} and 10^{308}. These numbers are also called *normal* numbers. Any number smaller than 10^{-308} is interpreted as zero whereas any number bigger than 10^{308} is interpreted as infinity (*inf*). The dSPACE implementation supports the full range of IEEE 754 normal numbers.

Subnormal/denormal numbers

It is possible to extend the range of representable small numbers from 10^{-308} up to 10^{-324} by using so-called *subnormal* or *denormal* numbers. These numbers have a reduced precision compared to normal numbers and may introduce run-time penalties in form of a reduced floating-point performance. The standard dSPACE implementation supports denormal numbers but the behavior depends on the actual dSPACE product. For SCALEXIO, denormal numbers are supported by default. For all other dSPACE products, denormal numbers are treated as zero.

Changing the handling of denormal numbers in SCALEXIO You can change the handling of denormal numbers in SCALEXIO to *treat as zero* by defining the preprocessor macro `DS_ENABLE_DENORMALS_ZERO_HANDLING` in the related build configuration in ConfigurationDesk. To do so, perform these steps:

1. Open the Build Configuration table.
2. Select the relevant build configuration set.
3. In the Properties Browser, enter `DS_ENABLE_DENORMALS_ZERO_HANDLING` in the Macros to be defined edit field.

For more information, refer to Build Configuration Table (ConfigurationDesk User Interface Reference 🕮).

Finite mathematics

By default, the dSPACE default implementation uses the following compiler optimization flag:

```
-ffast-math
```

This assumes what is known as *finite mathematics* which means that all arguments to functions are expected to be well-defined numbers. This is a limitation compared to full IEEE 754 support. Performing calculations using the special values *infinity* and *not a number* (see below) may have undefined results and must therefore be avoided.

Special case: Infinity and division by zero As a result of a computation, a number might become infinite. Typical examples for this are numbers that become larger than the normal floating-point range, or the result of a division by zero. In the dSPACE implementation, a division by zero never leads to an exception but in a result of infinity. Using infinity as an argument to subsequent mathematical operations should be avoided.

Special case: Not a number As a result of a computation, a number might become not a number (NaN). Typical examples for this are the attempt to calculate the logarithm of a negative number or to divide zero by zero. In the dSPACE implementation, an undefined mathematical operation never leads to an exception but results in NaN. Using NaN as an argument to subsequent mathematical operations must be avoided.

Note

Comparing NaN to NaN in the dSPACE implementation always yields true. This differs from standard IEEE 754 implementations where NaN ≠ NaN.

Comparing floating-point numbers

In general, when working with floating-point numbers it is difficult to predict where and how rounding errors will occur. Therefore, comparing two values for exact equality must be avoided except for the special cases infinity, NaN, and 0.

Mathematical functions

In some exceptional cases, calculating a mathematical function can result in significantly reduced computation performance, for example, for very large numbers or very small numbers. The loss of performance can be about 100 microseconds in contrast to just a few microseconds. To provide optimal real-time performance, the standard mathematical functions have been replaced by dSPACE internal implementations that have a more deterministic execution time. Refer to the following table:

Standard Mathematical Function	dSPACE Internal Implementation
Sine	sin(x)
Cosine	cos(x)
Tangent	tan(x)
Arc tangent	atan(x), atan2(x,y)
Euler exponent	exp(x)
General exponent	pow(x,y)

Compared to the standard mathematical functions, the dSPACE internal implementations have a maximum loss of precision of *ULP(1)* (ULP = unit of least precision). This means, that the result might differ in at most one digit of the mantissa.

Re-enabling the standard mathematical functions If additional precision is required, the standard mathematical functions can be re-enabled. To re-enable the standard math functions, perform the following steps:

1. Connect to your board using a browser such as Chrome or Firefox.
2. Open the Configuration ⇒ Custom Configuration tab.
3. In the free text form, enter the following lines:

```
[MATH]
DS_FAST_SIN=0
DS_FAST_COS=0
DS_FAST_TAN=0
DS_FAST_EXP=0
DS_FAST_POW=0
DS_FAST_ATAN=0
```

4. Click Change to finish the operation.

Note

You can omit any of the lines above in order to re-configure some of the math functions.

Impact of hardware on calculation results

In addition to the reasons described above, the overall precision of floating-point operations also depends on the actual hardware floating-point unit (FPU) that is used. For example, the FPU of an x86-based CPU can calculate with up to 80 bits internally, whereas the FPU of an ARM-based CPU only uses 64 bits. As a result, it might happen that two operations provide slightly different results when they are executed on a SCALEXIO system (x86) in contrast to the MicroAutoBox III (ARM).

Note

Comparing floating-point values for exact equality must be avoided.

Further readings

By default, the dSPACE implementation sets the `-ffast-math` compiler flag. For more information, refer to the GNU Compiler Collection specification for GCC version 5.2.0, chapter 3.

Basic Information on Handling MATLAB

Introduction

ConfigurationDesk can access MATLAB to execute several operations.

Where to go from here

Information in this section

Using ConfigurationDesk with MATLAB

Introduction

ConfigurationDesk can access MATLAB for basic operations such as starting and exiting MATLAB, or opening a model.

Starting MATLAB

If you execute an operation in ConfigurationDesk that requires access to MATLAB, and MATLAB is not running, ConfigurationDesk starts it.

The working directory of MATLAB depends on the following points:

- MATLAB is already started
  - The working directory of the running MATLAB session is used.
- MATLAB is started by ConfigurationDesk
  - If there is no reference to a Simulink model specified in the ConfigurationDesk application, or the reference is not valid, the MATLAB working directory is set to `<ConfigurationDesk Application>`.
  - If the ConfigurationDesk application contains a reference to a Simulink model, the MATLAB working directory is set to the model path.

Note

If you have connected several MATLAB installations to your dSPACE installation, you must select one of them as the preferred connection. If only one connectable MATLAB installation is available on your host PC, this automatically becomes the preferred MATLAB installation.

Exiting MATLAB

The MATLAB session started by ConfigurationDesk runs independently of the ConfigurationDesk process. You cannot close MATLAB from ConfigurationDesk, you must do so manually. You cannot close the MATLAB session while a ConfigurationDesk operation has access to it.

Opening the behavior model with initial values

In ConfigurationDesk, you can specify an initialization command that is executed before the behavior model referenced to the ConfigurationDesk application is opened.

The command is not executed if the behavior model is already open. To activate the initialization command, you must close the model and reopen it from ConfigurationDesk, or you can type the command in the MATLAB Command Window and execute it.

Note the following points when using an initialization command:

- The command is executed in the MATLAB workspace.
- The working directory is set to the model's path during execution.
- The command can consist of several MATLAB commands.
- You can change the working directory and enlarge the MATLAB search path.
- The model will not be opened if an error occurs.

For details on the initialization command, refer to Model Implementation Properties (ConfigurationDesk User Interface Reference 📖)

Saving the behavior model

If you save the ConfigurationDesk application, and you have opened and modified the referenced behavior model, you are asked to save the behavior model too. The state of a generated interface model is not checked, because it is not contained in the ConfigurationDesk application.

Tip

You can specify not to be prompted every time, if you want to save the model. To do so, you must type the following command in the MATLAB Command Window:

```
dsmpb_pref('Set', 'ModelSaveMode',
'SaveWithoutConfirmation');
```

For more information, refer to Handling Model Interface Package for Simulink Preferences on page 30.

Handling configuration sets at version change

If you change the MATLAB version and/or the dSPACE Release, configuration sets stored in a MAT file of an earlier version might cause problems. Therefore, you are recommended to create these configuration sets again when you change the Release version.

How to Customize the MATLAB Start-Up

Objective

To customize the MATLAB start-up, you must use custom scripts.

Execution order of the initialization phase

The execution order is:

- dSPACE initialization
- `dsstartup.m`
- `dspoststartup.m`
- `startup.m`

Method

To customize the MATLAB start-up

1 Add the necessary commands in the `dsstartup.m` file, for example, to automatically open a library or model whenever MATLAB is started. The `dsstartup.m` file has to be placed in the MATLAB search path.

For more information, refer to dsstartup.m (Model Interface Package for Simulink Reference 📖).

> **Tip**
>
> To specify further commands to be executed after the dSPACE initialization, you can use **`dspoststartup.m`**.

Example

Suppose you want MATLAB to automatically change to your working folder `d:\work` and open the Simulink model `my_model.slx`. Create the dsstartup.m file and write the following commands to it:

```
cd d:\work
my_model
```

Customizing the MATLAB shutdown

You can customize the MATLAB shutdown by using the `dsfinish.m` file to carry out user-specific commands before MATLAB is shut down. Place this M file in the MATLAB search path, for example, in MATLAB's working folder. You can specify several `dsfinish.m` files. All `dsfinish.m` files located in the MATLAB search path are executed before MATLAB is shut down.

Related topics

References

dsfinish.m (Model Interface Package for Simulink Reference 📖)
dspoststartup.m (Model Interface Package for Simulink Reference 📖)
dsstartup.m (Model Interface Package for Simulink Reference 📖)
startup.m (Model Interface Package for Simulink Reference 📖)

Handling Model Interface Package for Simulink Preferences

Introduction

The Model Interface Package for Simulink provides methods for you to read, write, and save preferences, such as the model save mode.

Saving preferences via API command

You can read, write, and save the preferences of the Model Interface Package for Simulink via the `dsmpb_pref()` API command. Saving preferences means that the settings you made are available in the next MATLAB session and are valid for the associated dSPACE installation.

For more information, refer to dsmpb_pref (Model Interface Package for Simulink API Reference 📖).

> **Note**
>
> You cannot transfer preferences between different dSPACE installations, but you can read the settings of one installation and write them to another installation via the `dsmpb_pref()` API command.

Related topics

References

dsmpb_pref (Model Interface Package for Simulink API Reference 📖)

User-Friendly Connection Between Simulink and ConfigurationDesk

Introduction

The Model Interface Package for Simulink provides methods to simplify working with Simulink models in ConfigurationDesk.

Where to go from here

Information in this section

Remote Access to ConfigurationDesk

Introduction

The Model Interface Package for Simulink lets you easily execute actions in ConfigurationDesk via remote access. For this purpose, the Model Interface Package for Simulink provides specific commands for the following actions:

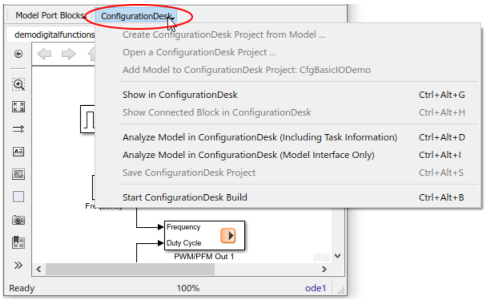
- Handling the ConfigurationDesk project and application.
- Displaying ConfigurationDesk project information.
- Displaying elements in ConfigurationDesk.
- Deleting elements in the Simulink behavior model and their representatives in ConfigurationDesk in one step.

- Analyzing the Simulink behavior model.
- Starting the build process in ConfigurationDesk.

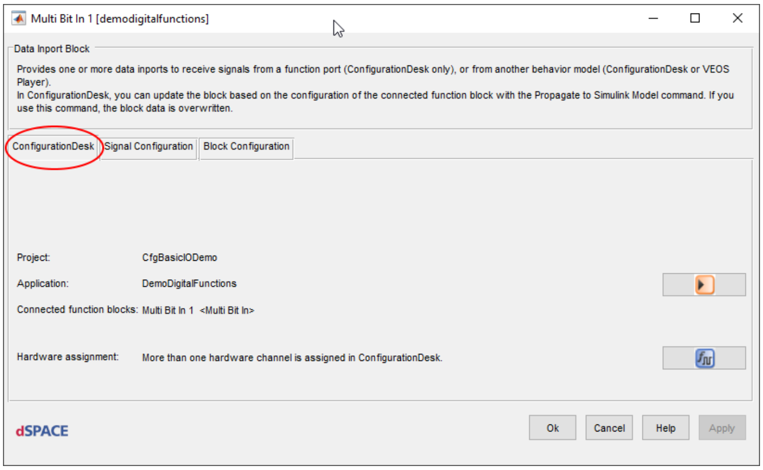
Accessing the commands

You can access the commands as follows:

- Via the **ConfigurationDesk** menu on the menu bar of your Simulink behavior model.


Model Port Blocks
ConfigurationDesk
demodigitalfunctions
Create ConfigurationDesk Project from Model ...
Open a ConfigurationDesk Project ...
Add Model to ConfigurationDesk Project: CfgBasicIODemo
Show in ConfigurationDesk Ctrl+Alt+G
Show Connected Block in ConfigurationDesk Ctrl+Alt+H
Analyze Model in ConfigurationDesk (Including Task Information) Ctrl+Alt+D
Analyze Model in ConfigurationDesk (Model Interface Only) Ctrl+Alt+I
Save ConfigurationDesk Project Ctrl+Alt+S
Start ConfigurationDesk Build Ctrl+Alt+B
Frequency
Duty Cycle
PWM/PFM Out 1
Ready
100%
ode1

- Via the **ConfigurationDesk** page in the model port block dialogs.



Provides one or more data inports to receive signals from a function port (ConfigurationDesk only), or from another behavior model (ConfigurationDesk or VEOS Player).
In ConfigurationDesk, you can update the block based on the configuration of the connected function block with the Propagate to Simulink Model command. If you use this command, the block data is overwritten.

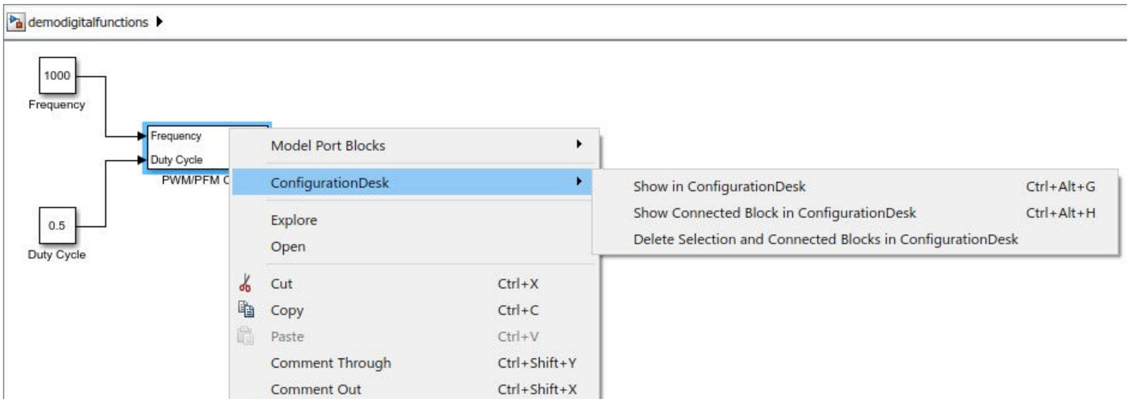
Project: CfgBasicIODemo

Application: DemoDigitalFunctions

Connected function blocks: Multi Bit In 1 <Multi Bit In>

Hardware assignment: More than one hardware channel is assigned in ConfigurationDesk.

- Via the context menu of selected model port blocks.

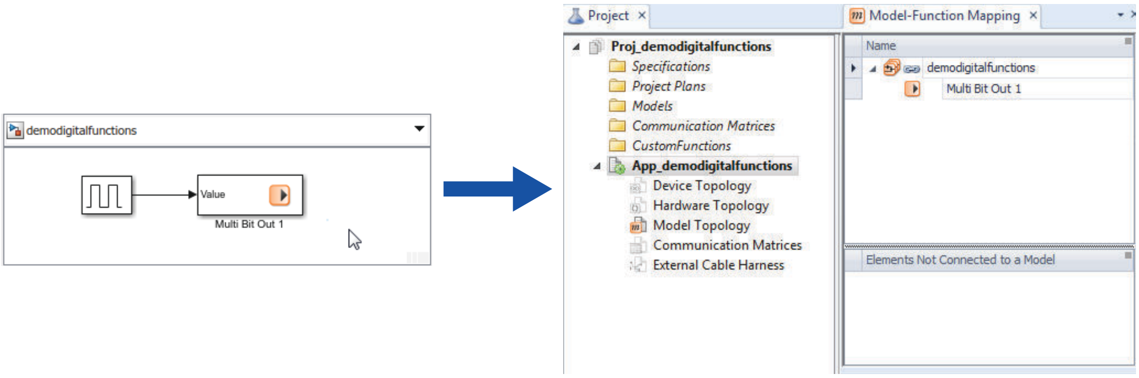


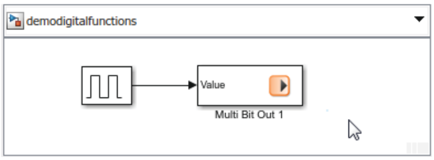
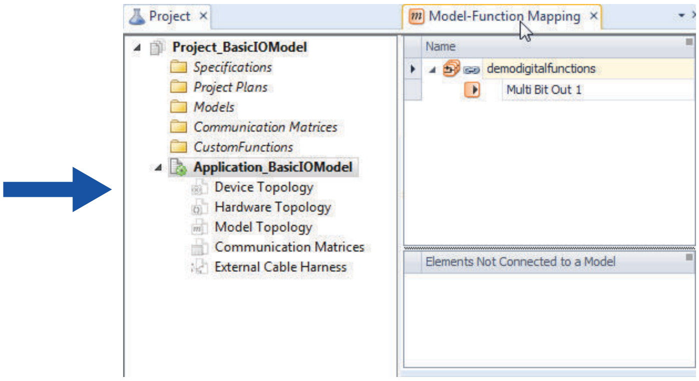
Note

Depending on whether a ConfigurationDesk project is already open or not, some commands may be unavailable.

Handling the ConfigurationDesk project and application

The following commands for project and application handling are available via the ConfigurationDesk menu in the Simulink model.

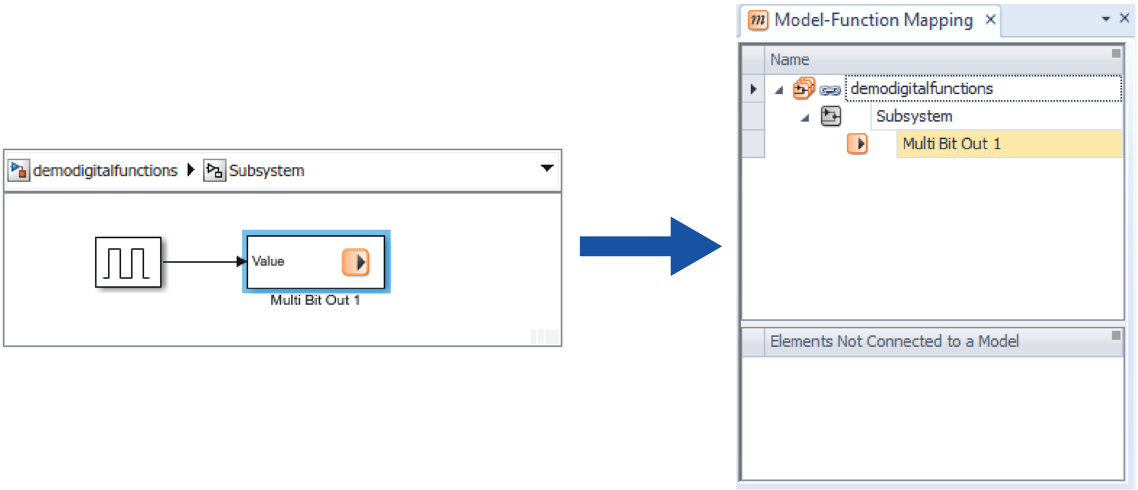
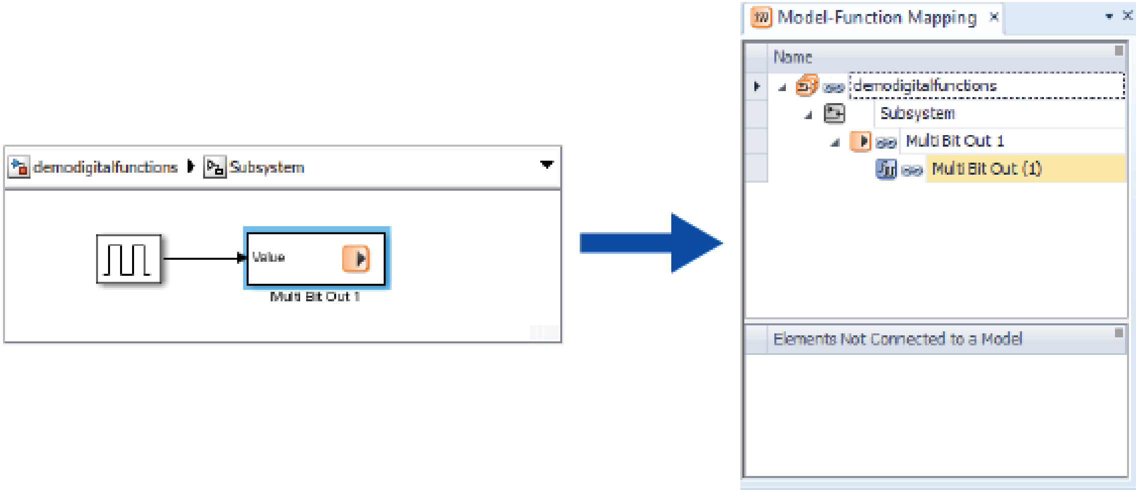
Command	Result in ConfigurationDesk
Create ConfigurationDesk Project from Model	If you select this command, a dialog opens for you to confirm that the Simulink behavior model is prepared for use in ConfigurationDesk. If you click **OK**, the following actions are performed: ▪ The Simulink behavior model is configured automatically for dSPACE Run-Time Target (dsrt.tlc). ▪ The Simulink behavior model is saved. You are prompted to specify a file name and a location for the model. After this, the following actions are performed: ▪ A new ConfigurationDesk project with a new ConfigurationDesk application is created. ▪ The current Simulink behavior model is added to the new ConfigurationDesk application. ▪ The model interface and the task information of the behavior model is analyzed.  demodigitalfunctions Value Multi Bit Out 1 Project Proj_demodigitalfunctions Specifications Project Plans Models Communication Matrices CustomFunctions App_demodigitalfunctions Device Topology Hardware Topology Model Topology Communication Matrices External Cable Harness Model-Function Mapping Name demodigitalfunctions Multi Bit Out 1 Elements Not Connected to a Model
Open a ConfigurationDesk Project	The **Open Project** dialog is opened in ConfigurationDesk. To avoid conflicts, this command is available only if no project is currently open in ConfigurationDesk.
Add This Model to ConfigurationDesk Project	If you select this command, a dialog opens for you to confirm that the Simulink behavior model is prepared for

Command	Result in ConfigurationDesk
	use in ConfigurationDesk. If you click OK, the following actions are performed: ▪ The Simulink behavior model is configured automatically for dSPACE Run-Time Target (dsrt.tlc). ▪ The Simulink behavior model is saved. You are prompted to specify a file name and a location for the model. Then, the current Simulink behavior model is added to the active ConfigurationDesk application of the open ConfigurationDesk project.  Project Model-Function Mapping Project_BasicIOModel Specifications Project Plans Models Communication Matrices CustomFunctions Application_BasicIOModel Device Topology Hardware Topology Model Topology Communication Matrices External Cable Harness Name demodigitalfunctions Multi Bit Out 1 Elements Not Connected to a Model
Save ConfigurationDesk Project	The ConfigurationDesk project and the active application is saved. If there are any unsaved changes in the Simulink behavior model, a dialog is displayed in which you are prompted if you want to save the Simulink model as well.

Displaying and deleting elements in ConfigurationDesk

The following commands for displaying elements in ConfigurationDesk are available via the ConfigurationDesk menu in the Simulink model:

Command	Result in ConfigurationDesk
Show in ConfigurationDesk	The related model port block or subsystem is selected in the Model-Function Mapping Browser. This command is also available on the ConfigurationDesk page in the model port block dialogs (Show Block in ConfigurationDesk), and in the ConfigurationDesk context menu of a model port block.

Command	Result in ConfigurationDesk
 demodigitalfunctions Subsystem Value Multi Bit Out 1 Model-Function Mapping Name demodigitalfunctions Subsystem Multi Bit Out 1 Elements Not Connected to a Model	
Show Connected Block in ConfigurationDesk	The function block to which the related model port block is mapped in ConfigurationDesk is selected in the Model-Function Mapping Browser. This command is also available on the ConfigurationDesk page in the model port block dialogs, and in the ConfigurationDesk context menu of a model port block.
 demodigitalfunctions Subsystem Value Multi Bit Out 1 Model-Function Mapping Name demodigitalfunctions Subsystem Multi Bit Out 1 Multi Bit Out (1) Elements Not Connected to a Model	
Delete Selection and Connected Blocks in ConfigurationDesk	The following elements are deleted in one step: ▪ In Simulink: The selected Data Inport and Data Outport blocks in the Simulink model. ▪ In ConfigurationDesk: ▪ The model port blocks that correspond to the selected Data Inport and Data Outport blocks in ConfigurationDesk. ▪ The function blocks to which the corresponding model port blocks are mapped in ConfigurationDesk.

Analyzing the Simulink behavior model

If you modify the model interface or the task information in the Simulink behavior model, you must make these modifications known to ConfigurationDesk. This can be done via the following commands in the ConfigurationDesk menu:

- Analyze Model in ConfigurationDesk (Including Task Information)

  If you select this command, the model interface and the task information provided by the Simulink behavior model are analyzed by ConfigurationDesk.
- Analyze Model in ConfigurationDesk (Model Interface Only)

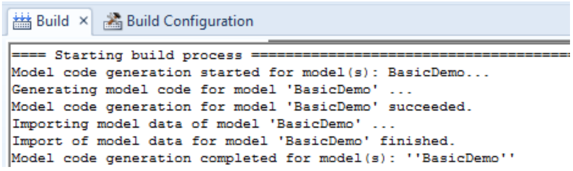
  If you select this command, the model interface provided by the Simulink behavior model is analyzed by ConfigurationDesk.

> **Note**
>
> These commands have the same functionality as the Analyze Simulink Model (Including Task Information) command and the Analyze Simulink Model (Model Interface Only) command in ConfigurationDesk.

Starting the build process

The command for starting the build process is available via the ConfigurationDesk menu in the Simulink model.

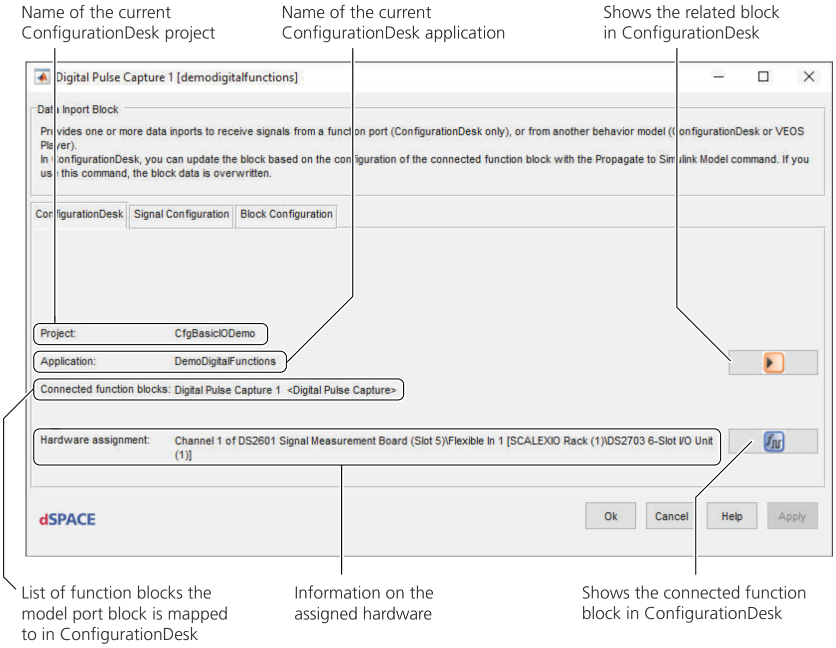
Command	Result in ConfigurationDesk
Start ConfigurationDesk Build	The build process is started in ConfigurationDesk. ConfigurationDesk is brought to the front and shows the Build Log Viewer, which displays information on the build process. **Note** This command has the same functionality as the Start Build command in ConfigurationDesk.



```
==== Starting build process =====================================
Model code generation started for model(s): BasicDemo...
Generating model code for model 'BasicDemo' ...
Model code generation for model 'BasicDemo' succeeded.
Importing model data of model 'BasicDemo' ...
Import of model data for model 'BasicDemo' finished.
Model code generation completed for model(s): ''BasicDemo''
```

Displaying ConfigurationDesk project information

The following illustration shows the information that is displayed on the ConfigurationDesk page of the model port block dialogs:



Name of the current ConfigurationDesk project

Name of the current ConfigurationDesk application

Shows the related block in ConfigurationDesk

Digital Pulse Capture 1 [demodigitalfunctions]

Data Inport Block

Provides one or more data inports to receive signals from a function port (ConfigurationDesk only), or from another behavior model (ConfigurationDesk or VEOS Player).
In ConfigurationDesk, you can update the block based on the configuration of the connected function block with the Propagate to Simulink Model command. If you use this command, the block data is overwritten.

Project: CfgBasicIODemo

Application: DemoDigitalFunctions

Connected function blocks: Digital Pulse Capture 1 <Digital Pulse Capture>

Hardware assignment: Channel 1 of DS2601 Signal Measurement Board (Slot 5)\Flexible In 1 [SCALEXIO Rack (1)\DS2703 6-Slot I/O Unit (1)]

List of function blocks the model port block is mapped to in ConfigurationDesk

Information on the assigned hardware

Shows the connected function block in ConfigurationDesk

Related topics

References

Menu Commands for the Remote Access of ConfigurationDesk (Model Interface Package for Simulink Reference 🕮)

Connecting Simulink Models to ConfigurationDesk

Simplified integration of Simulink models in ConfigurationDesk

To be able to use a Simulink model as a behavior model in ConfigurationDesk, it must be a part of a ConfigurationDesk project. The Model Interface Package for Simulink provides commands that let you create a ConfigurationDesk project directly from a Simulink model, or to add a Simulink model to an active ConfigurationDesk application. If you do this, the following actions are performed:

1. The Simulink model is preconfigured for the use in ConfigurationDesk.
2. The Simulink model is added to the active ConfigurationDesk application.
3. The Simulink model is analyzed by ConfigurationDesk.

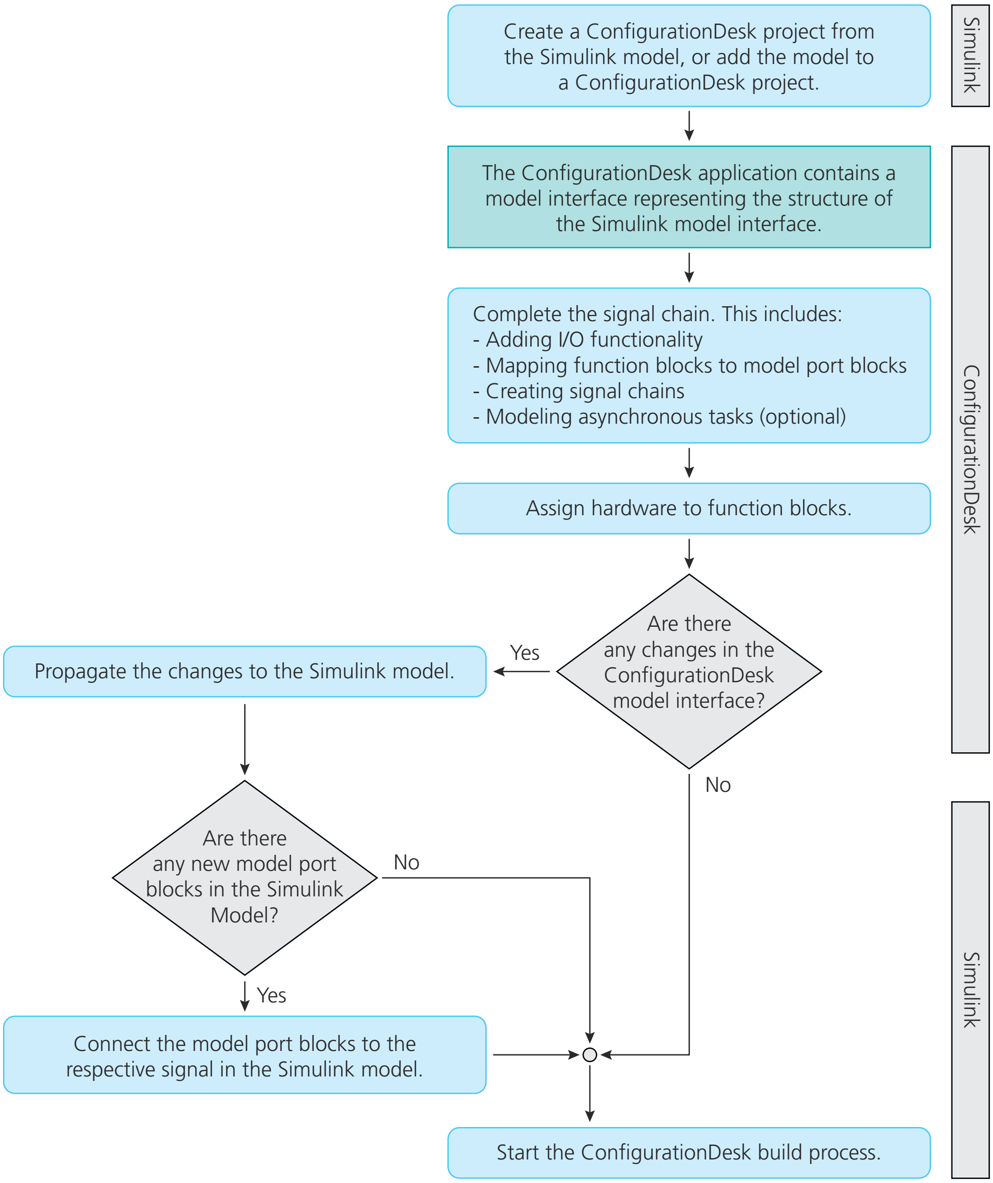
The active ConfigurationDesk application then contains a model interface representing the structure of the Simulink behavior model. This is called ConfigurationDesk model interface. You can use the ConfigurationDesk model interface to map it to function blocks, for creating new signal chains, and for modeling asynchronous tasks.

Note

When the model is imported into ConfigurationDesk, the Simulink data types are converted to corresponding data types used in ConfigurationDesk. For more information, refer to Adding the Generated Model Interface to Your Behavior Model via an Interface Model on page 46 and Data Interchange Between ConfigurationDesk and Behavior Model (ConfigurationDesk Real-Time Implementation Guide 📖).

Workflow of using Simulink models in ConfigurationDesk

The following illustration shows the workflow of using Simulink behavior models in ConfigurationDesk:

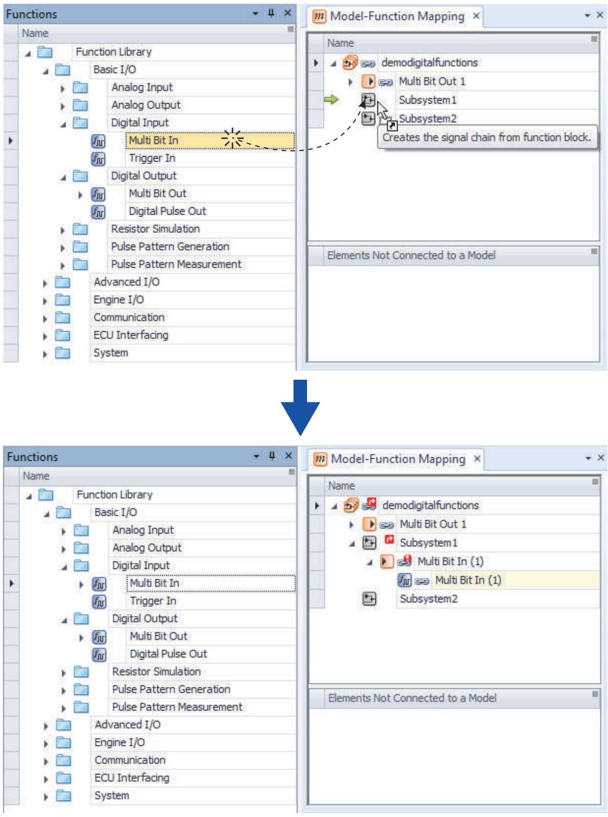

Create a ConfigurationDesk project from the Simulink model, or add the model to a ConfigurationDesk project.
Simulink
The ConfigurationDesk application contains a model interface representing the structure of the Simulink model interface.
Complete the signal chain. This includes:
- Adding I/O functionality
- Mapping function blocks to model port blocks
- Creating signal chains
- Modeling asynchronous tasks (optional)
ConfigurationDesk
Assign hardware to function blocks.
Are there any changes in the ConfigurationDesk model interface?
Yes
Propagate the changes to the Simulink model.
No
Are there any new model port blocks in the Simulink Model?
No
Yes
Connect the model port blocks to the respective signal in the Simulink model.
Start the ConfigurationDesk build process.
Simulink

Working with the Model-Function Mapping Browser

ConfigurationDesk provides the Model-Function Mapping Browser to complete the signal chain in an easy way. For example, you can perform the following actions via drag & drop:

- Adding I/O functionality and creating signal chains in one step by dragging function blocks to the model root or to the subsystem.

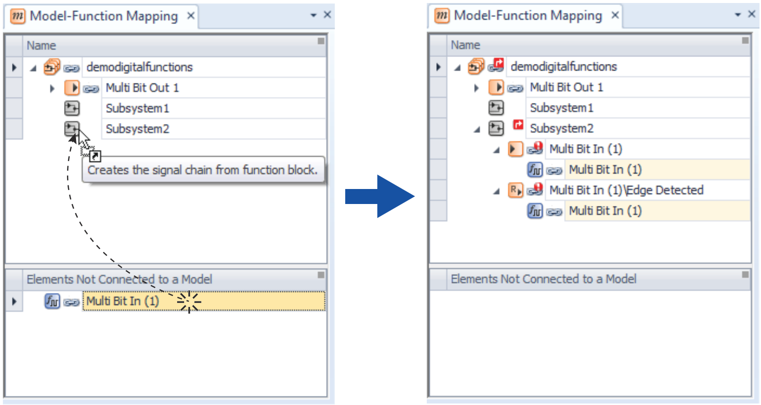
See the following illustration.



Creates the signal chain from function block.

- Modeling asynchronous tasks by dragging function blocks with enabled event generation to a subsystem of the Simulink model or to an unconnected Runnable Function block.

  See the following illustration (for an animated graphic, refer to dSPACE Help).



Creates the signal chain from function block.

Model port blocks that are newly created via the drag & drop action are unresolved which is indicated by a ❗ symbol. Unresolved model port blocks must be propagated to the Simulink model to make them resolved in ConfigurationDesk. Existing model port blocks that have been changed are marked with a 🟥 symbol if the changes have not yet been propagated to the Simulink behavior model.

> **Note**
>
> Mapping function blocks and model port blocks via drag & drop is not supported for function blocks with configuration ports, such as CAN blocks or LIN blocks. These blocks must be mapped manually in the Signal Chain Browser.

Working with default hardware in ConfigurationDesk If you specified a default hardware in ConfigurationDesk and opened a new empty model in Simulink, you can create a ConfigurationDesk project and application via the Create ConfigurationDesk Project from Model command. The new project has then the following additional characteristics:

- ConfigurationDesk shows only the function blocks that are available for the specified hardware.
- Conflicts concerning missing hardware resources are avoided.

Related topics

Basics

Working with Simulink Behavior Models (ConfigurationDesk Real-Time Implementation Guide 🕮)

References

Menu Commands for the Remote Access of ConfigurationDesk (Model Interface Package for Simulink Reference 🕮)

Synchronizing the Simulink Model Interface and the ConfigurationDesk Model Interface

Synchronizing model interfaces

For a correct data exchange between a Simulink model and ConfigurationDesk, both model interfaces must be synchronous. If you change either the Simulink model interface or the ConfigurationDesk model interface, the model interfaces are not synchronous any longer.

Changes in the Simulink model interface To make modifications in the Simulink model known to ConfigurationDesk, you can start a model analysis via one of the following commands from the ConfigurationDesk menu in the Simulink model:

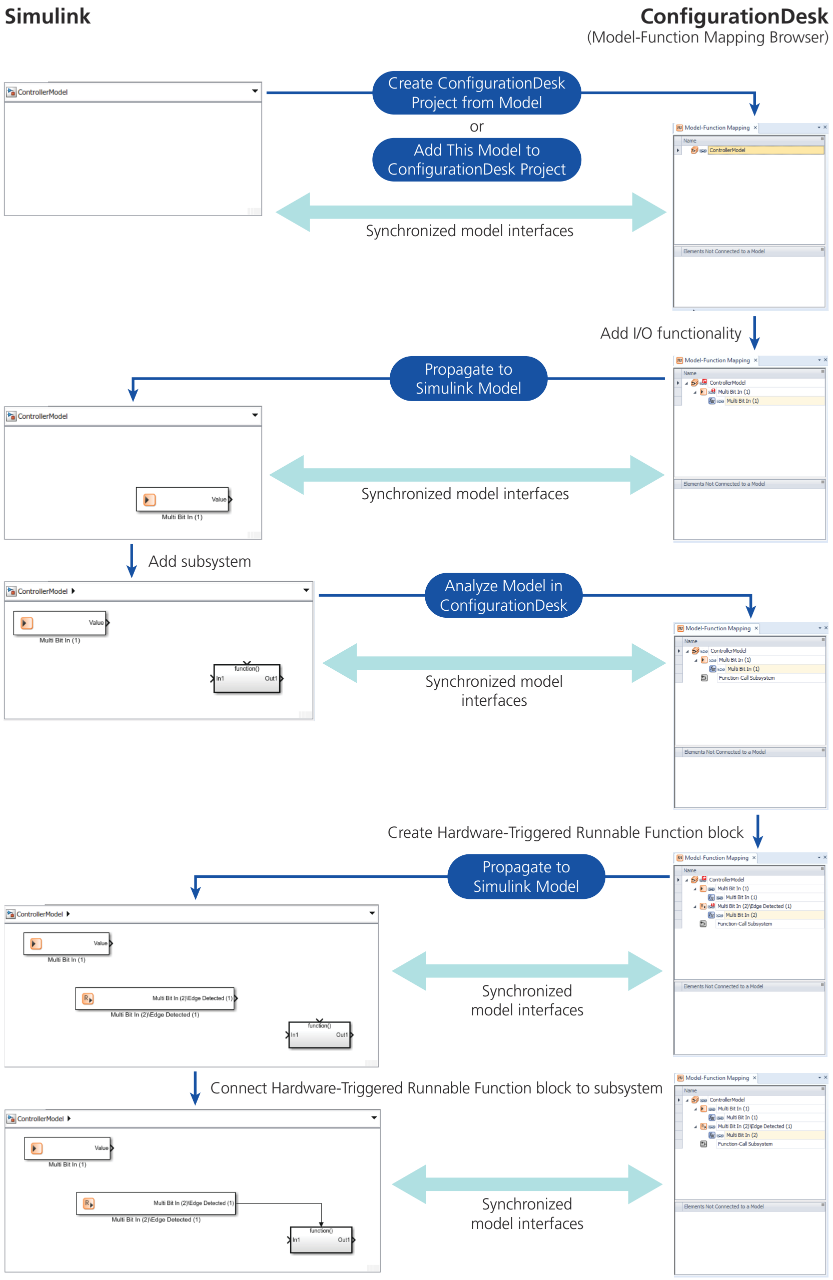
- Analyze Model in ConfigurationDesk (Model Interface Only)
- Analyze Model in ConfigurationDesk (Including Task Information)

Note

Commands for analyzing the Simulink model are also available in ConfigurationDesk.

Changes in the ConfigurationDesk model interface If you modified the ConfigurationDesk model interface, you can propagate the changes to the Simulink model via the **Propagate to Simulink Model** command.

The following illustration shows in which cases the model interface in Simulink or in ConfigurationDesk must be updated:


Simulink
ConfigurationDesk
(Model-Function Mapping Browser)
ControllerModel
Create ConfigurationDesk Project from Model
or
Add This Model to ConfigurationDesk Project
Model-Function Mapping
Name
ControllerModel
Elements Not Connected to a Model
Synchronized model interfaces
Add I/O functionality
Propagate to Simulink Model
Multi Bit In (1)
Value
Synchronized model interfaces
Add subsystem
Analyze Model in ConfigurationDesk
function()
In1
Out1
Function-Call Subsystem
Synchronized model interfaces
Create Hardware-Triggered Runnable Function block
Propagate to Simulink Model
Multi Bit In (2)\Edge Detected (1)
Multi Bit In (2)
Synchronized model interfaces
Connect Hardware-Triggered Runnable Function block to subsystem
Synchronized model interfaces

Note

The following applies if you use the **Propagate to Simulink Model** command:

- To prevent parts of a Simulink model from being accidentally changed by a propagate operation, you can protect specific subsystems by setting them to read-only in Simulink. When analyzed, these subsystems are marked with a lock symbol in ConfigurationDesk.
- This command is available only for model port blocks and function blocks. It cannot be used for configuration port blocks.
- Possible formatting of the model port blocks in Simulink, e.g., the color, is lost.

Modifying model port blocks

You have to consider the following modifications in the Simulink behavior model that affect the model interface:

- You modified the block and port settings.

  Some modifications, for example, the port width, affect on the signal chain in ConfigurationDesk. The new settings are available in ConfigurationDesk after a model analysis.
- You added a model port or a model port block.

  The new model port or model port block is available in ConfigurationDesk after a model analysis.
- You deleted a model port or a model port block.

  ConfigurationDesk detects the deleted model port or model port block after a model analysis. If the model port block or model port is used by your application, it is not automatically deleted from the signal chain. A deleted model port block is marked as unresolved in the signal chain and in the **Model-Function Mapping Browser**. A deleted model port that is mapped to a function port is marked as unresolved in the signal chain.. Unresolved model port blocks are not considered for code generation. Unresolved model ports in ConfigurationDesk have no functionality.

Switching via keyboard shortcuts

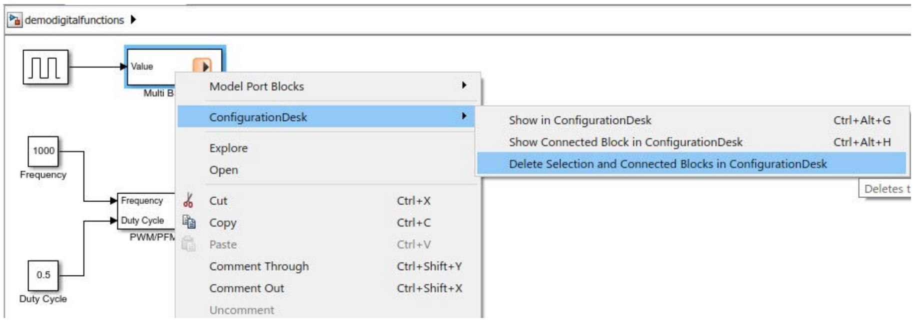
Both, the Model Interface Package for Simulink and ConfigurationDesk provide the same keyboard shortcuts that let you switch between a Simulink model port block and its related blocks in ConfigurationDesk.

Shortcut Key	Description
`Ctrl` + `Alt` + `G`	Lets you switch between a model port block and the related model port block in Simulink or in ConfigurationDesk.
`Ctrl` + `Alt` + `H`	Lets you switch between a Simulink model port block and the connected function block in ConfigurationDesk.

Switching between the blocks lets you directly view and change the configuration of the function block, for example. If necessary, you can propagate any changes in ConfigurationDesk back to the Simulink behavior model.

Deleting blocks in Simulink and in ConfigurationDesk in one step

The Model Interface Package for Simulink provides the Delete Selection and Connected Blocks in ConfigurationDesk command. This command is accessible via the ConfigurationDesk context menu of the selected Data Inport blocks, Data Outport blocks, or Hardware-Triggered Runnable Function blocks:



If you execute this command, the following blocks are deleted in one step:

- In Simulink: The selected model port blocks in the Simulink model.
- In ConfigurationDesk:
  - The model port blocks that correspond to the model port blocks selected in Simulink.
  - The function blocks to which the corresponding model port blocks are mapped in ConfigurationDesk.

Because the model port blocks are deleted in the Simulink model and in ConfigurationDesk, both model interfaces are still synchronous. In this case, it is not necessary to analyze the Simulink model in ConfigurationDesk or to propagate the changes from ConfigurationDesk to the Simulink model.

Note

- You cannot undo this command in Simulink.
- You can execute this command even if the model port block is not mapped to a function block in ConfigurationDesk.

Related topics

Basics

Connecting Simulink Models to ConfigurationDesk...37

References

Analyze Model in ConfigurationDesk (Including Task Information) (Model Interface Package for Simulink Reference 🕮)
Analyze Model in ConfigurationDesk (Model Interface Only) (Model Interface Package for Simulink Reference 🕮)

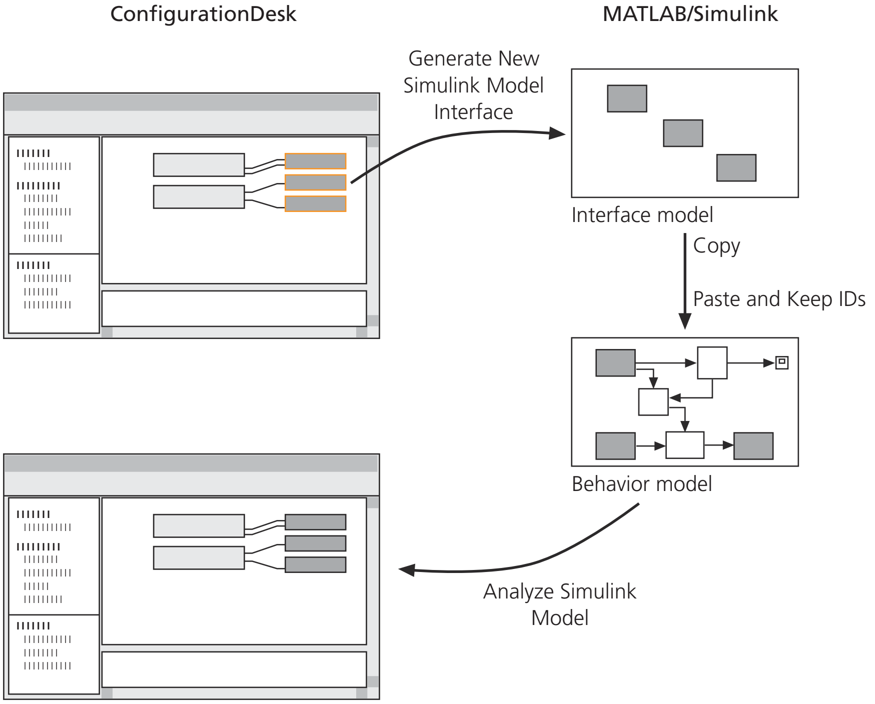
Adding the Generated Model Interface to Your Behavior Model via an Interface Model

Introduction

ConfigurationDesk lets you export the ConfigurationDesk model interface as a new Simulink interface model. You can then add the generated model interface to your Simulink behavior model manually.

Generating the model interface

If you have extended the signal chain in ConfigurationDesk with suitable model port blocks, you can let ConfigurationDesk generate the model interface, i.e., the model port blocks of your ConfigurationDesk application are generated as data port blocks in a new interface model. These blocks contain their identities right from the start.



Some function blocks in ConfigurationDesk provide event generation. If you want to model asynchronous tasks in your ConfigurationDesk application, you must enable event generation for the relevant function block. You can then let ConfigurationDesk generate an interface model containing **Hardware-Triggered Runnable Function** blocks suitable for modeling asynchronous tasks.

Adding the generated model interfaces to your behavior model manually

There are two methods to combine your behavior model with the generated interface models:

- You can copy your behavior model to the interface model and use the data port blocks as its inports and outports. Then you can save the enlarged interface model as the new behavior model to be used as the reference in the ConfigurationDesk application. This will be accessed for analyzing the model interfaces and generating the real-time application.

- You can copy the generated data port blocks in the interface model and paste them to your behavior model via the standard Copy command together with the **Paste and Keep IDs** command from the **Model Port Blocks** menu in the Simulink Editor and use them as its inports and outports.

Note

If you use the standard Copy and Paste commands to copy the model port blocks from the generated interface model to your behavior model, the block identities are reset. The model port blocks are then treated as new model port blocks in ConfigurationDesk that have no mapping in the signal chain. If you want to use variants of the behavior model in your real-time application by replacing the referenced working model in the ConfigurationDesk application, you must ensure that the model interfaces are identical. You must use model port blocks with the same identities in each model variant. For further information, refer to Basics on Model Port Block IDs and Signal IDs on page 63.

See also General Information on the Model Interface Blockset on page 51.

Data type mapping

The data types that you specified in your behavior model for the data ports are automatically mapped to the data types used in the data ports in ConfigurationDesk. In the other direction, the data types specified in ConfigurationDesk are automatically converted to the Simulink data types in the generated interface model.

Simulink Data Type	ConfigurationDesk Data Type
int8	Int8 (-128 ...+127)
uint8	UInt8 (0 ... +255)
int16	Int16 (-32768 ... +32767)
uint16	UInt16 (0 ... 65535)
int32	Int32 (-2147483648 ... +2147483647)
uint32	UInt32 (0 ... +4294967295)
int64	Int64 (-9223372036854775808 ... 9223372036854775807)
uint64	Uint64 (0 ... 18446744073709551615)
boolean	Bool (0/1)
single	Float32 (-3.402823E+38 ... +3.402823E+38)
double	Float64 (-1.79769e+308 ... 1.79769e+308)

For further information, refer to Data Interchange Between ConfigurationDesk and Behavior Model (ConfigurationDesk Real-Time Implementation Guide 🕮).

Tip

ConfigurationDesk lets you exchange Simulink bus signals between Simulink models and ConfigurationDesk via structured data ports. For details, refer to Exchanging Simulink Bus Signals Between Simulink and ConfigurationDesk or VEOS Player on page 71.

Modifying the block and port settings

If you want to modify the settings of the model port blocks and ports, you can do this only in the behavior model. Except for the block and port names of generated model port blocks, all the settings of the model port blocks are read-only in ConfigurationDesk.

Related topics

Basics

Specifying the Model Interface (ConfigurationDesk Real-Time Implementation Guide 🕮)

Specifying the Interface of Behavior Models

Introduction

To familiarize you with the methods for modeling the interfaces of behavior models. These are:

- Implementing the model interface using the blocks from the Model Interface Blockset.
- Separating models from an overall model using the **Model Separation Setup** tool.

Note

The Model Interface Package for Simulink lets you generate Simulink implementation container files (SIC files) for the separated models, and use them in ConfigurationDesk or VEOS Player. If you work with SIC files, you are not recommended to use the MCD file with ConfigurationDesk, because the MCD file references the separated Simulink models, not the SIC files you generated for them. The MCD file cannot be used in VEOS Player.

- Working with function triggers and handling runnable functions.
- Modeling CAN, LIN, and FlexRay bus communication in the behavior model.

Note

The Model Interface Package for Simulink does not support the generation of SIC files for behavior models that contain CAN, LIN, or FlexRay bus communication. However, you can use such models in ConfigurationDesk by adding them directly to a ConfigurationDesk application.

Where to go from here

Information in this section

Creating the Interface of Behavior Models

Introduction

You can implement the interface of a behavior model by using the blocks from the Model Interface Blockset.

Where to go from here

Information in this section

General Information on the Model Interface Blockset

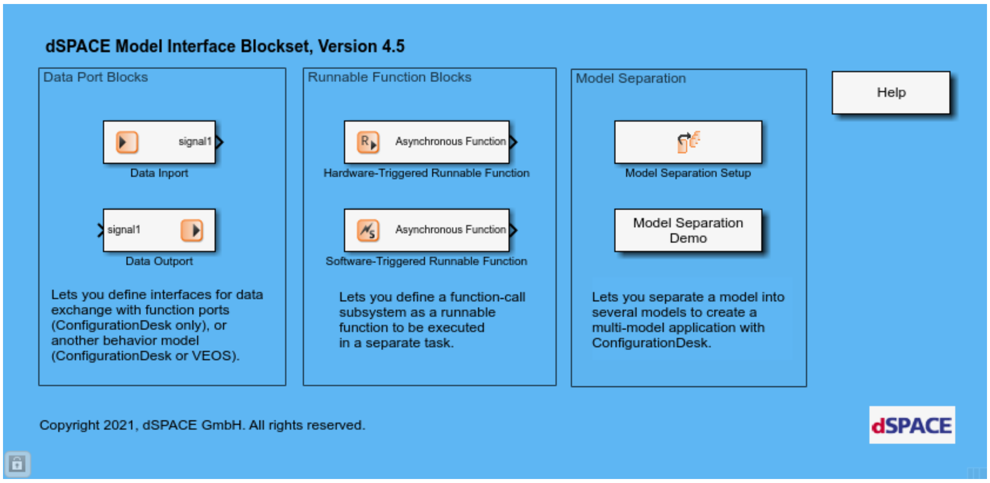
Introduction

The Model Interface Blockset contains blocks for data inputs and data outputs (= data port blocks) and runnable functions. These form:

- The interface between the behavior model and the I/O functionality in the real-time application (RTA file).
- The interface between two or more behavior models in your real-time application (multicore or multi-PU application) or in the offline simulation application (OSA file).

Model Interface Blockset

The Model Interface Blockset is installed together with the Model Interface Package for Simulink. To open the blockset, type `mips` in the MATLAB Command Window.



dSPACE Model Interface Blockset, Version 4.5

Lets you define interfaces for data exchange with function ports (ConfigurationDesk only), or another behavior model (ConfigurationDesk or VEOS).

Lets you define a function-call subsystem as a runnable function to be executed in a separate task.

Lets you separate a model into several models to create a multi-model application with ConfigurationDesk.

Copyright 2021, dSPACE GmbH. All rights reserved.

The Model Interface Blockset provides the following blocks:

- Data port blocks:
  - A **Data Inport** block in the behavior model provides one or more data inports to retrieve signals from another model (in ConfigurationDesk and VEOS Player) or from I/O functionality (only in ConfigurationDesk).
  - A **Data Outport** block in the behavior model provides one or more data outports to apply signals to other models (in ConfigurationDesk and VEOS Player) or to I/O functionality (only in ConfigurationDesk).
- Runnable function blocks
  - A **Hardware-Triggered Runnable Function** block exports a connected function-call subsystem as a runnable function. The runnable function is available in ConfigurationDesk after a complete model analysis. You can then assign it to a periodic or asynchronous task.
  - A **Software-Triggered Runnable Function** block exports connected function-call subsystems as runnable functions in predefined tasks for use in ConfigurationDesk or VEOS Player.
- Model separation blocks
  - The **Model Separation Setup** block opens a GUI that lets you separate individual models from an overall behavior model. Refer to Separating Models from an Overall Model to Build Multimodel Applications on page 98.
  - Model Separation Demo provides access to a demo model for model separation.

A **Help** button that provides access to the user documentation is also provided.

Data port blocks and signals

The Data Inport block and the Data Outport block provide one signal by default. You can add further signals to a data port block, for example, to combine signals of the same model functionality.

Block identification

To ensure that model port blocks can be clearly identified, they must have unique identifiers. You must specify a unique name for each runnable function provided by a Hardware-Triggered Runnable Function block or a Software-Triggered Runnable Function block. Refer to the following topics:.

- Hardware-Triggered Runnable Function Block (Model Interface Package for Simulink Reference 🕮)
- Software-Triggered Runnable Function Block (Model Interface Package for Simulink Reference 🕮)

Using model port blocks in Simulink libraries

You can build Simulink libraries with model port blocks. You can modify the configuration of the model port blocks in the library. If the blocks that you copy to the behavior model are stored on the root level of the library, you can edit their settings in the behavior model. If the blocks that you copy to the model are stored in other library blocks, for example, a subsystem, the settings of the blocks are read-only in the behavior model.

Model port blocks that reside in libraries or in library block instances have a unique set of block and signal IDs.

Note

It is not possible to use a Simulink library as a behavior model in ConfigurationDesk.

Configuring model port blocks via variables

You can use variables from the MATLAB workspace to configure model port blocks more flexibly.

The following settings can be set by workspace variables:

- Sample time
- Width
- Initial value (for Data Inport blocks)
- Required priority (Hardware-Triggered Runnable Function blocks and Software-Triggered Runnable Function blocks)

All the other block settings cannot be specified by MATLAB workspace variables.

Model referencing

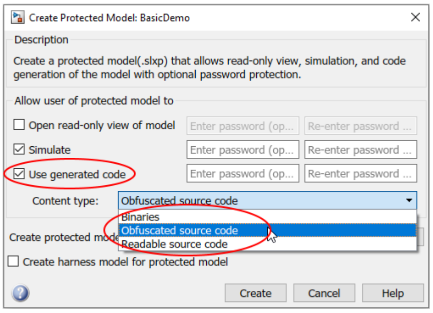
If you are using the model referencing feature of Simulink, note that the interface to ConfigurationDesk must be within the top-level model and not in one of the referenced models.

There are important limitations on the use of model referencing. For a detailed list of model referencing limitations, refer to the Simulink® Coder™ documentation.

Support of protected Simulink models

The Model Interface Package for Simulink supports Simulink behavior models and/or Simulink implementation containers that contain protected referenced models. To create a protected referenced model, you have to select the **Use generated code** checkbox in the Simulink model's **Create protected models** dialog. The Model Interface Package for Simulink supports the following options from the **Content type** list (including password-protection):

- Obfuscated source code
- Readable source code



Create Protected Model: BasicDemo

Description

Create a protected model(.slxp) that allows read-only view, simulation, and code generation of the model with optional password protection.

Allow user of protected model to

□ Open read-only view of model | Enter password (op... | Re-enter password ...

☑ Simulate | Enter password (op... | Re-enter password ...

☑ Use generated code | Enter password (op... | Re-enter password ...

Content type: Obfuscated source code

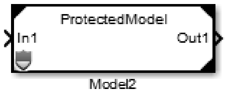
Binaries
Obfuscated source code
Readable source code

Create protected mode

□ Create harness model for protected model

A protected model can be additionally protected by password.

The following figure shows a **Model** block that represents a reference to a protected model.



Note

Note the following limitations when you use protected models:

- A model that contains a protected model can only be built if the protected model contains its generated code.
- The build options of the protected model, especially the system target file, must be suitable for the parent model.
- The Model Interface Package for Simulink does not support the **Binaries** option for protected models.
- The Model Interface Package for Simulink does not support the protection of a model if this model contains other **Model** blocks.
- When generating model code for a given model referencing hierarchy with a certain version of the Model Interface Package for Simulink, all protected models used in this hierarchy must have been generated with the Model Interface Package for Simulink 4.1 and later.

Handling model data

The Model Interface Package for Simulink provides API commands for handling data of the behavior model and model port blocks.

Reading/writing model port block data You can read and write data from or to model port blocks via API commands. Refer to the following topics:

- dsmpb_get (Model Interface Package for Simulink API Reference)
- dsmpb_set (Model Interface Package for Simulink API Reference)
- dsmpb_addsignal (Model Interface Package for Simulink API Reference)
- dsmpb_rmsignal (Model Interface Package for Simulink API Reference)
- dsmpb_assignbusobjects (Model Interface Package for Simulink API Reference)

Removing data from a model If required, you can remove all data resulting from the Model Interface Blockset from a Simulink model. This is useful, for example, if you want to use a Simulink model in an environment without a Model Interface Package for Simulink installation. In this case, you have to delete all model port blocks from the model and execute the `dsmpb_mdlcleanup()` API command. Refer to dsmpb_mdlcleanup (Model Interface Package for Simulink API Reference).

Related topics

References

Data Inport Block (Model Interface Package for Simulink Reference)
Data Outport Block (Model Interface Package for Simulink Reference)
Hardware-Triggered Runnable Function Block (Model Interface Package for Simulink Reference)
Software-Triggered Runnable Function Block (Model Interface Package for Simulink Reference)

Creating the Model Interface to ConfigurationDesk by Using Model Port Blocks

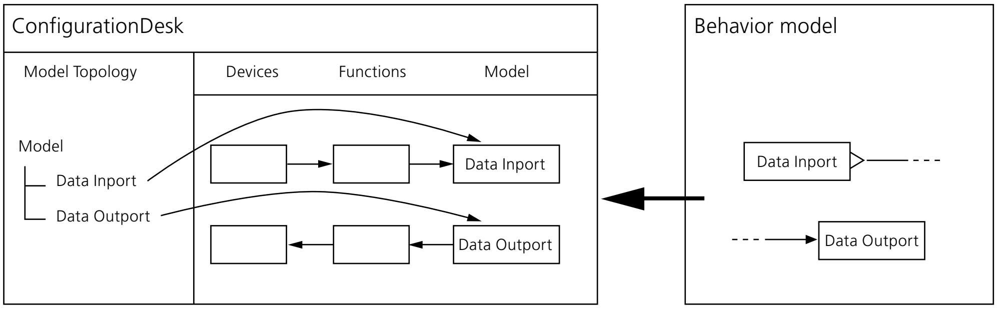
Introduction

You should familiarize yourself with creating the model interface by using model port blocks. The description below focuses on the **Data Inport** and **Data Outport** block. For information on how to work with the **Hardware-Triggered Runnable Function** block and the **Software-Triggered Runnable Function** block, refer to Working with Runnable Functions and Function Triggers on page 82.

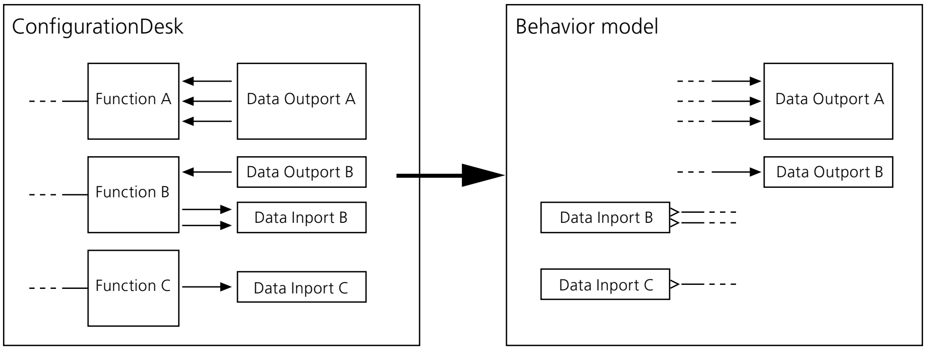
Basics on the model interface

Model implementing for ConfigurationDesk means that the behavior model does not contain I/O functionality. The functions for measuring and generating I/O signals are specified in the ConfigurationDesk application. The interface between your behavior model and the ConfigurationDesk application is implemented using the model port blocks. You must configure a model port in a model port block for each signal that you want to connect to a function block.

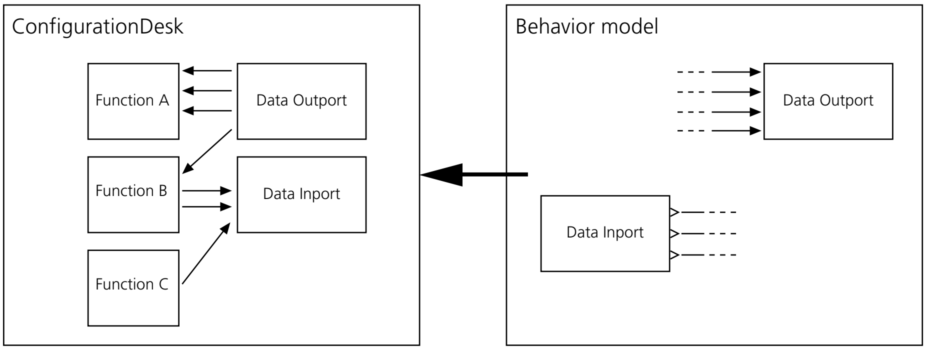
After analyzing the model interface in ConfigurationDesk, you get a model port block in the model topology of your ConfigurationDesk application for each model port block in your behavior model. You can use the model port blocks from the model topology to complete the signal chain in ConfigurationDesk that consists of the external devices, the function blocks and the model port blocks.



If you use the functionalities from ConfigurationDesk to build the model interface, one generated model port block refers to one single function block. Each data direction and port type requires a new model port block.



If you create the model interface using the Model Interface Blockset in MATLAB/Simulink, you can design a more flexible interface.



A further important aspect of adding ports and model port blocks to your model is the timing behavior in the real-time application of the function mapped to the data ports. When you build the real-time application, function modules are created according to the added model port blocks and assigned to tasks that are responsible for the timing behavior. For details on function modules, refer to Basics on Function Triggers on page 86.

Dependencies between data ports and function modules

The architecture of the behavior model defines the function modules in the real-time application. The data port blocks in the behavior model are assigned to these function modules specifying the timing behavior of the mapped function blocks. For further information on function modules, refer to Basics on Function Triggers on page 86.

Adding model port blocks to the behavior model

If you have opened the Model Interface Blockset in MATLAB, you can select a block from the blockset and drag it to an opened model. This creates a copy of the blockset element in the model. You can also use the **Copy** and **Paste** commands to copy already configured model port blocks from any library or model to your behavior model. The copied model port block is treated as a new block in the model interface and it has new unique IDs.

Limitations for using model port blocks

You have to consider limitations concerning the use of model port blocks. Refer to Limitations of Model Port Blocks on page 282.

Using Simulink Inports and Outports

You can also use Simulink Inports and Outports as interface blocks for your model instead of the Data Inport and Data Outport blocks of the Model Interface Blockset, with the following limitations:

- The inports and outports must be on the root level of your model.
- The signal passed by the Inport or Outport block can be one of the following:
  - scalar or vector
  - Data type = double, single, int8, uint8, int16, uint16, int32, uint32, int64, uint64, boolean, or Simulink.Bus objects
- The signal passed by the Inport or Outport block must be real (non-complex) and sample based.
- Signals to or from another behavior model that are connected to Simulink Inports and Outports are always read at the beginning of a simulation step or written at the end of a simulation step. In contrast to this, the signals of I/O functions or from other behavior models that are connected to signal ports are read and written when the data port block is executed within a simulation step.
- For Simulink Inports, the Include virtual blocks checkbox on the DSRT variable description file options page of the Configuration Parameters dialog must be selected. Otherwise, the build process returns an error message. The ports will be correctly displayed in ConfigurationDesk's model topology even if the checkbox is cleared.
- Outport blocks which retrieve a vectorized signal from a virtual block are not supported.

- The signals of the Simulink Inport and Outport blocks must be represented by global variables in the generated model code. Some build options prevent the generation of global variables, for example, Signal storage reuse. By enabling the Test point option, however, you can force the generation of global variables.

Tip

To configure the Inport and Outport blocks according to the requirements of ConfigurationDesk, dSPACE provides the `dsmpb_configurerootslports()` API command. Refer to dsmpb_configurerootslports (Model Interface Package for Simulink API Reference 📖).

Related topics

Basics

Adding the Generated Model Interface to Your Behavior Model via an Interface Model.......... 46

How to Configure Data Port Blocks

Objective

By adding a **Data Inport** block or **Data Outport** block from the Model Interface Blockset to your behavior model, you get an instance of the block with a default configuration. This configuration can be adapted to your needs.

The default configuration of a data port block

A Data Inport block with its default configuration is named *Data Inport* and a Data Outport block is named *Data Outport*. A data port block provides one data port named *signal1*. The data port is configured as scalar value of double data type, and the sample time of the block will be inherited.

signal1
Data Inport
signal1
Data Outport

Dependencies to ConfigurationDesk

The block and port configurations take effect in ConfigurationDesk after you update the model topology by executing the Analyze Simulink Model (Model Interface Only) command in ConfigurationDesk. The settings are only displayed in ConfigurationDesk and cannot be edited there.

Method

To configure a data port block

1 In the behavior model, you can edit the name according to the naming conventions for Simulink blocks.

Note

Depending on the tool chain you are working with (operating system, experiment software, modeling tool, etc.), the full range of UTF-16 characters might not be supported. To avoid problems, use only ASCII characters for names of model port blocks and model ports.

In addition, to avoid conflicts, do not use the following characters in block names and signals:

- "
- /
- .

2 Double-click the block to open the Data Inport or Data Outport block dialog.

3 On the Signal Configuration page, you can add a signal by clicking the ⬚ button. The default signal name is *signal* plus a consecutive number.

To delete a signal, you must select it and click the ☒ button.

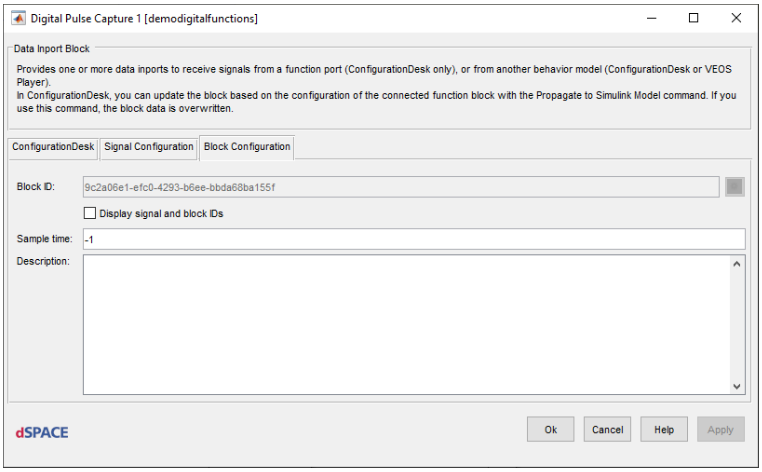
4 You can sort the signal list by selecting a signal and clicking the ↑ button to move the signal up one item, or the ↓ button to move the signal down one item.

5 Select a signal from the list and configure the signal properties, for example, the signal name. Note that the Initial value setting can only be specified for signals of Data Inport blocks.

Note

The port name must not be empty and must not begin or end with a "/".

6 On the Block Configuration page, you can specify the sample time and a description.



Digital Pulse Capture 1 [demodigitalfunctions]

Data Inport Block

Provides one or more data inports to receive signals from a function port (ConfigurationDesk only), or from another behavior model (ConfigurationDesk or VEOS Player).
In ConfigurationDesk, you can update the block based on the configuration of the connected function block with the Propagate to Simulink Model command. If you use this command, the block data is overwritten.

Block ID: 9c2a06e1-efc0-4293-b6ee-bbda68ba155f

☐ Display signal and block IDs

Sample time: -1

Description:

Note

If required, you can assign a new block ID to the data port block. For this purpose, you must enable the ⚙ button via the `dsmpb_pref()` API command. To do so, you must type the following command in the MATLAB Command Window:

```
dsmpb_pref('Set', 'EnableEditIds', true);
```

7 Click OK to apply the settings and close the dialog.

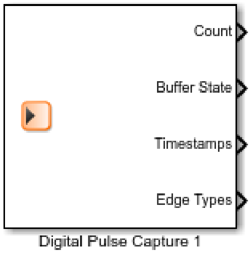
Result

You have added signals to the block and specified their settings. There are some settings, such as the sample time of the block or the data specification of the signal via type, width and initial value, that are used for the code generation. Other settings, such as the block and signal descriptions, are useful for passing information to the ConfigurationDesk or VEOS Player user.

Tip

You can use MATLAB workspace variables to configure the Initial value, Width, and Sample time settings.

In the illustration below, you can see a Data Inport block with four data inports as an example.



Note

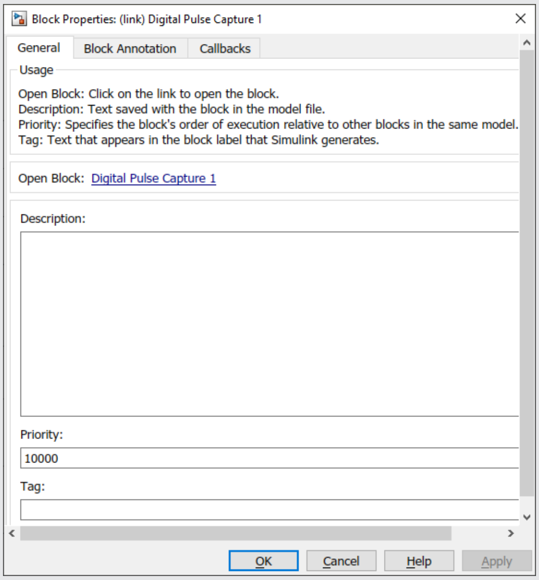
Data Inport blocks and Data Outport blocks have the following default block priorities:

- Data Inport block: 10000
- Data Outport block: -10000

The default block priorities ensure that Data Outport blocks are calculated as early as possible, and Data Inport blocks are calculated as late as possible. This is especially important to avoid model communication deadlocks in multicore real-time applications or multi-processing-unit applications. If a Data Inport block is connected to a Data Outport block via direct feed-through, the following Simulink warning is issued in the MATLAB Command Window during the build process:

```
Warning: Unable to honor user-specified priorities.
'<DataInportBlockPath>(pri=[10000, Inf]) has to
execute before '<DataOutportBlockPath>' (pri=
[-10000, Inf]) to satisfy data dependencies
```

In this case, Simulink ignores the default block priorities. The Data Inport block is calculated before the Data Outport block. To avoid this warning, you can change or delete the priorities of the relevant blocks. Select Block Properties from the context menu of the block, and change the Priority in the General tab:



Usage

Open Block: Click on the link to open the block.
Description: Text saved with the block in the model file.
Priority: Specifies the block's order of execution relative to other blocks in the same model.
Tag: Text that appears in the block label that Simulink generates.

Open Block: Digital Pulse Capture 1

Description:

Priority: 10000

Tag:

Related topics

HowTos

How to Configure Runnable Function Blocks.. 84

Basics on Model Port Block IDs and Signal IDs

Introduction

If model port blocks are created in a Simulink behavior model for the first time, the model port blocks and their signals are given unique identifiers (IDs). This is the case, for example, if you copy a model port block from the Model Interface Blockset to a behavior model, or if you create a Simulink model containing model port blocks with ConfigurationDesk.

Unique IDs ensure that model port blocks and signals can be clearly identified by ConfigurationDesk or VEOS Player. A model port block in ConfigurationDesk and the related model port block in the Simulink behavior model are a pair if they have the same ID in both tools.

Behavior of IDs during work

The following overview lists operations that maintain or change block and signal IDs.

Operations that do not change IDs Model port blocks keep their IDs when you perform the following actions:

- Move them in the same Simulink model or library using drag & drop, or cut & paste.
- Duplicate them using the standard Copy command together with the **Paste and Keep IDs** command within the same or different models or libraries.
- Rename signal ports and model port blocks.
- Rename subsystems that contain model port blocks.
- Rename the model or library.
- Change the order of the ports of a model port block.
- Undo the deletion of a model port block.

Operations that change IDs The IDs in the modified model port block change when you perform the following actions:

- Copy a model port block using copy & paste.
- Copy a model port block using drag & drop. This includes using drag & drop from a library, the Simulink Library Browser, or from a model to another model or library.
- Move a model port block between models and/or libraries using cut & paste.

Note

If you add or modify a model port block in a Simulink library or a Subsystem model, you must then open, initialize (e.g., with `Ctrl+D`), and save the Simulink models that reference those subsystems. This ensures that IDs are assigned to the added model port blocks and model ports in the Simulink models.

Assigning new IDs

You can assign new unique IDs to a model port block. To enable this feature, you must enter the `dsmpb_pref('Set','EnableEditIds', true);` API command in the MATLAB Command Window. You can then assign a new block ID or signal ID via the ▣ button on the **Block Configuration** page or the **Signal Configuration** page of the block dialog.

> **Tip**
>
> As an alternative, you can assign new IDs via the `dsmpb_set()` API command. Refer to dsmpb_set (Model Interface Package for Simulink API Reference 📖).

Displaying IDs in the behavior model

You can display the signal and block IDs of a data port block by selecting the **Display signal and block IDs** checkbox on the **Block Configuration** page of the block dialog. However, the number of displayed IDs is limited to 15 for each block.

Checking uniqueness of model port blocks

During a model analysis or before code generation in ConfigurationDesk, the model port block IDs are checked for uniqueness. However, the **Model Port Blocks** menu of a Simulink model provides the **Check Model for DSRT Code Generation** command, that lets you perform this check manually. If the behavior model contains duplicate model port block IDs, error messages inform you of the affected model port blocks.

Using model port blocks with identical IDs

The Model Interface Package for Simulink provides methods for you to create model port blocks with identical IDs. This is useful in the following use cases:

- Creating different variants of behavior models with identical model interfaces.
- Creating different variants of specific parts in a behavior model. Only one variant is active in each case.

For more information, refer to Using Model Port Blocks with Identical IDs on page 65.

For more information on Variant Subsystems, refer to the Simulink documentation.

Related topics

Basics

Adding the Generated Model Interface to Your Behavior Model via an Interface Model............ 46

Using Model Port Blocks with Identical IDs

Introduction

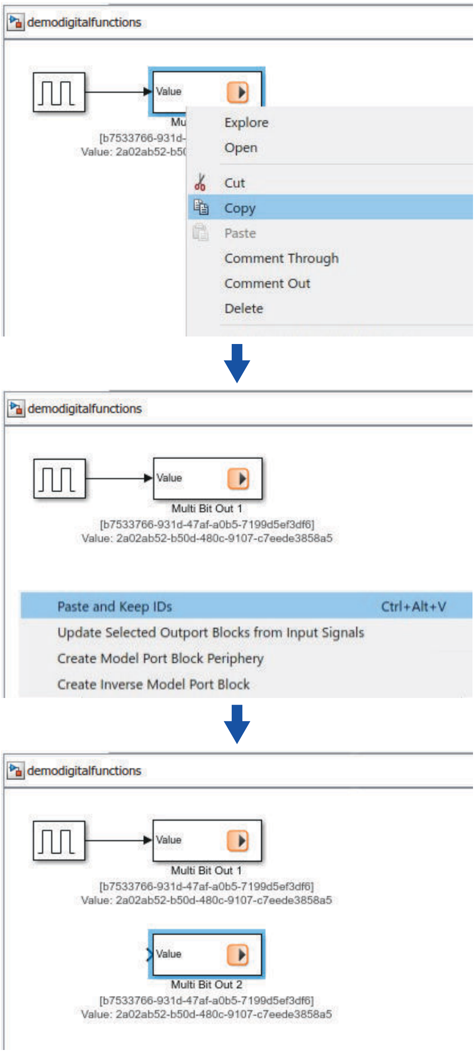
The Model Interface Package for Simulink lets you create model port blocks that have the same block and signal IDs. Model port blocks with the same IDs are useful in the following use cases:

- Creating different variants of behavior models with identical model interfaces.
- Creating different variants of specific parts in a behavior model. Only one variant is active in each case.

Creating model port blocks with identical IDs

To create model port blocks with identical IDs, you must use the standard Copy command together with the Paste and Keep IDs command from the context menu of the relevant block, for example. The newly created block has the same IDs as the original block.

Refer to the following illustration:


demodigitalfunctions
Value
Mu
[b7533766-931d-
Value: 2a02ab52-b50
Explore
Open
Cut
Copy
Paste
Comment Through
Comment Out
Delete
demodigitalfunctions
Value
Multi Bit Out 1
[b7533766-931d-47af-a0b5-7199d5ef3df6]
Value: 2a02ab52-b50d-480c-9107-c7eede3858a5
Paste and Keep IDs
Ctrl+Alt+V
Update Selected Outport Blocks from Input Signals
Create Model Port Block Periphery
Create Inverse Model Port Block
demodigitalfunctions
Value
Multi Bit Out 1
[b7533766-931d-47af-a0b5-7199d5ef3df6]
Value: 2a02ab52-b50d-480c-9107-c7eede3858a5
Value
Multi Bit Out 2
[b7533766-931d-47af-a0b5-7199d5ef3df6]
Value: 2a02ab52-b50d-480c-9107-c7eede3858a5

Tip

To display IDs of a model port block in the behavior model, select the **Display signal and block IDs** checkbox on the **Block Configuration** page of the model port block's dialog.

The **Paste and Keep IDs** command can be executed as follows:

- Via the context menu of a Simulink behavior model canvas.
- Via the **Model Port Blocks** menu.
- Via the **`Ctrl+Alt+V`** shortcut.
- Via the `dsmpb_copyblockwithids()` API command to copy model port blocks with their IDs . Refer to dsmpb_copyblockwithids (Model Interface Package for Simulink API Reference 📖).

Note

- If you use the standard Copy and Paste commands, the inserted blocks are given new IDs. The model port blocks are then treated as new model port blocks in ConfigurationDesk without a mapping in the signal chain.
- If new IDs have been assigned to model port blocks or model ports, the model's dirty flag is set. You must save the model before you close it to keep the connections between ConfigurationDesk and the behavior model.

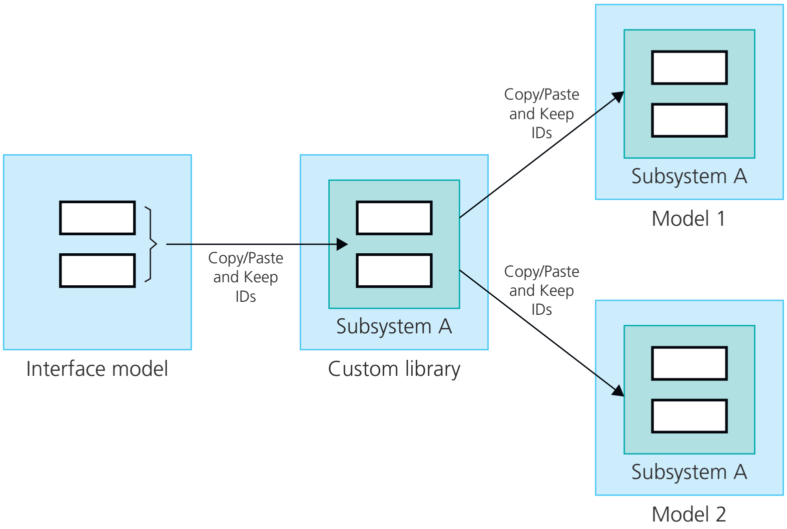
Creating behavior model variants for ConfigurationDesk

Creating different variants of behavior models for use in ConfigurationDesk comprises the following steps:

- In ConfigurationDesk, generate a temporary Simulink interface model.
- Copy the model port blocks from the temporary Simulink interface model to each variant of the behavior model using the standard Copy command together with the **Paste and Keep IDs** command.
- If required, update the mapping of the related model ports in ConfigurationDesk after a model analysis.

Creating behavior model variants with custom libraries

To make it easier to create variants of behavior models, you can copy model port blocks including their IDs to a subsystem in a custom library. You can then copy the subsystem with the model port blocks including their IDs from the custom library to different behavior models. This allows you to create different behavior models with an identical model interface. Refer to the following illustration:

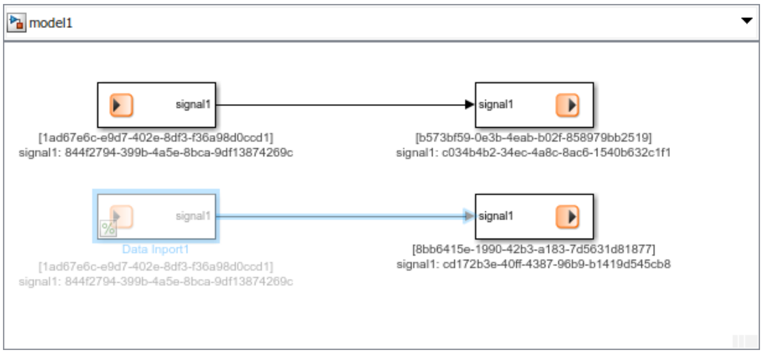


Note

Changed model port block IDs or signal IDs in custom libraries are not inherited to the model port blocks in behavior model variants. You must update the model port blocks and their IDs in the behavior model variants via the `dsmpb_copyblockwithids()` API command, if required.

Creating variants of behavior model parts

To create variants of behavior model parts, you can connect the respective model parts to model port blocks with identical IDs, and then comment out the model parts that must be deactivated. The deactivated parts of the behavior model are not available in ConfigurationDesk or VEOS Player. Refer to the following illustration:


model1
signal1
[1ad67e6c-e9d7-402e-8df3-f36a98d0ccd1]
signal1: 844f2794-399b-4a5e-8bca-9df13874269c
signal1
[b573bf59-0e3b-4eab-b02f-858979bb2519]
signal1: c034b4b2-34ec-4a8c-8ac6-1540b632c1f1
signal1
Data Inport1
[1ad67e6c-e9d7-402e-8df3-f36a98d0ccd1]
signal1: 844f2794-399b-4a5e-8bca-9df13874269c
signal1
[8bb6415e-1990-42b3-a183-7d5631d81877]
signal1: cd172b3e-40ff-4387-96b9-b1419d545cb8

As an alternative, you can work with Variant Subsystem blocks, or Variant Sink and Variant Source blocks. You have to place model port blocks in the variants using the standard Copy command together with the **Paste and Keep IDs** commands. Only one variant is active in each case. For more information on Variant Subsystems, or Variant Sink and Variant Source blocks, refer to the Simulink documentation.

Note

If the model interface is modified, all instances of model port blocks with the same IDs must be updated. To do so, update the model port blocks in the behavior model via the **Propagate to Simulink Model** command in ConfigurationDesk. This ensures that both the active and inactive instances of all model port blocks with the same IDs in the behavior model are updated.

Related topics

Basics

Basics on Model Port Block IDs and Signal IDs.. 63

Simplified Preparation of Model Interfaces for Model Communication

Introduction

To simplify your work, the Model Interface Package for Simulink provides methods for you to easily create communication interfaces for behavior models. For this purpose, the Model Interface Package for Simulink lets you create model port blocks that have the same configuration but the inverse data direction as the selected model port blocks. Creating inverse model port blocks is especially useful if you need a counterpart for a structured model port block.

Note

ConfigurationDesk also provides this feature, so you can create preconfigured inverse model port blocks in ConfigurationDesk as well as in your Simulink model. Refer to Simplified Preparation of Model Interfaces for Model Communication (ConfigurationDesk Real-Time Implementation Guide 🕮).

Characteristics of inverse model port blocks

The inverse model port blocks have the following characteristics:

- They have the same structure and port names as the original model port blocks.
- They have the same port configuration as the original model port blocks.
- They have the inverse data direction of the original model port block.
- They have the same names as the original model port blocks, but with an additional postfix. The postfix indicates the data direction of the inverse model port block. The following table shows some examples:

Name of the Original Model Port Block	Name of the Preconfigured Inverse Model Port Block
MyDataInportBlock	MyDataInportBlock_Out[1]
MyDataOutportBlock	MyDataOutportBlock_In[1]

[1] If there is already a model port block with the name of the new model port block in the Simulink model, the new model port block gets an incrementing number as an additional postfix.

- Each new model port block gets a unique block ID and unique signal IDs.

Workflow

Creating inverse model port blocks and preparing them for model communication comprises the following steps:

1. In the Simulink model, select the model port blocks you want to create inverse model port blocks for.
2. Create inverse model port blocks for the selected blocks via the **Create Inverse Model Port Blocks** command from the **Model Port Blocks** menu.
3. Use **Cut** and **Paste** to transfer the inverse model port blocks to the desired Simulink model.

> **Note**
>
> You can create inverse model port blocks via the `dsmpb_createinversedmodelportblock()` API command. Refer to dsmpb_createinversedmodelportblock (Model Interface Package for Simulink API Reference 🕮).

Next steps

To use the original and the inverse model port blocks for model communication, you have to perform the following steps:

1. In Simulink, add the inverse model port blocks to the desired Simulink models via **Cut** and **Paste**.
2. Go on working with the inverse model port blocks as usual, e.g., add the related Simulink models to a ConfigurationDesk application or generate an SIC file to use in ConfigurationDesk or VEOS Player. Note that you have to set up model communication between the Simulink models in ConfigurationDesk manually. Refer to Setting Up Model Communication (ConfigurationDesk Real-Time Implementation Guide 🕮).

Related topics

References

Create Inverse Model Port Block (Model Interface Package for Simulink Reference 🕮)
dsmpb_createinversedmodelportblock (Model Interface Package for Simulink API Reference 🕮)

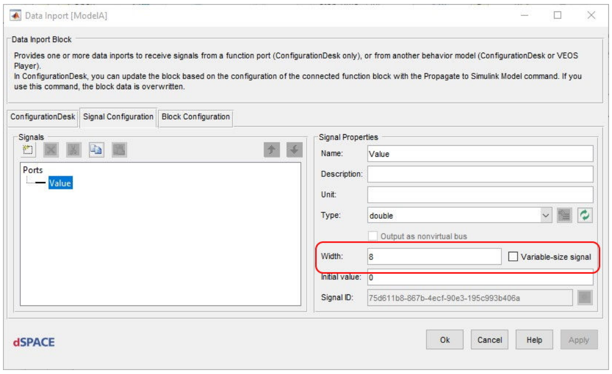
Specifying Variable-Size Signals for Model Port Blocks

Introduction

The Model Interface Package for Simulink lets you specify that a signal is a variable-size signal. This means that the number of elements of variable-size signals varies during run time.

Specifying variable-size signals

To specify that a signal is a variable-size signal, you must select the Variable-size signal checkbox on the Signal Configuration page of the Data Inport block dialog. The value specified in the Width edit field then indicates the maximum number of elements. Refer to the following illustration:



Data Inport [ModelA]

Data Inport Block

Provides one or more data inports to receive signals from a function port (ConfigurationDesk only), or from another behavior model (ConfigurationDesk or VEOS Player).
In ConfigurationDesk, you can update the block based on the configuration of the connected function block with the Propagate to Simulink Model command. If you use this command, the block data is overwritten.

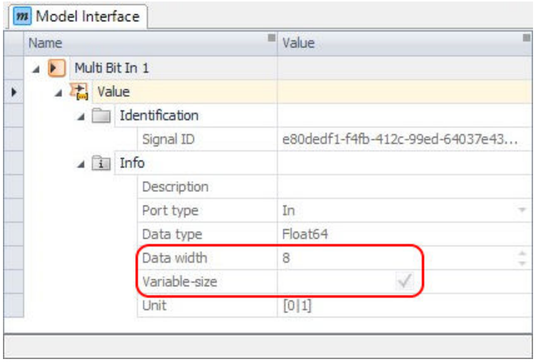
Signals

Ports
Value

Signal Properties

Name: Value
Description:
Unit:
Type: double
□ Output as nonvirtual bus
Width: 8 □ Variable-size signal
Initial value: 0
Signal ID: 75d611b8-867b-4ecf-90e3-195c993b406a

In ConfigurationDesk, variable-size signals can be identified using the Variable-size and the Data width properties:



Model Interface

Name	Value
Multi Bit In 1	
Value	
Identification	
Signal ID	e80dedf1-f4fb-412c-99ed-64037e43...
Info	
Description	
Port type	In
Data type	Float64
Data width	8
Variable-size	✓
Unit	[0|1]

Note

Signals of Data Outport blocks inherit the variable-size property from the block's input signals.

Specifying variable-size signals via Simulink.Bus objects

For variable-size signals of Simulink.Bus objects, the DimensionsMode property is set to *Variable*. When you assign such a Simulink.Bus object to a to a block port via the Type property, the signal specified in this way is variable-size.

Exchanging Simulink Bus Signals Between Simulink and ConfigurationDesk or VEOS Player

Introduction

You can exchange Simulink bus signals between a Simulink model and ConfigurationDesk or VEOS Player. To do so, you need suitable data port blocks. The **Model Interface Blockset** provides methods for you to create and update data port blocks with structured data ports for bus signals.

Where to go from here

Information in this section

Basics of Data Port Blocks with Structured Data Ports

Introduction

If you want to pass a Simulink bus signal to ConfigurationDesk or VEOS Player to map it to the ports of suitable function blocks or other models, you must know how to create data port blocks with structured data ports in your Simulink behavior model.

Structured ports of data port blocks

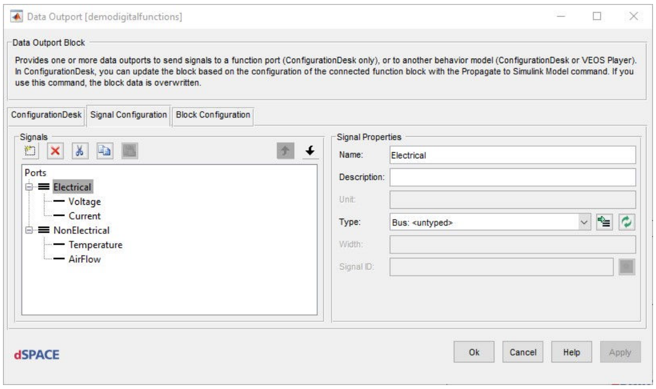
A structured data port is a hierarchically structured port of a **Data Inport** block or a **Data Outport** block. Each structured data port consists of further structured and/or unstructured data ports. The structured data ports can consist of signals with different data types (double, single, int8, int16, int32, int64, uint8, uint16, uint32, uint64, Boolean). You can connect only Simulink bus signals to structured data ports.

Note

- If you connect a Simulink bus signal that contains unsupported signals (for example, complex signals) to a structured data port or use a bus in the context of the automation API commands and update the structured data port from its input signals, the update is aborted with an error message.
- As a general rule, all signals connected to a data port block must have the same sample time as the data port block. This holds also true for the individual signals of a bus signal connected to a data port block. An exception to this rule is the case of signals connected to a data port block running at the fastest sample time of the model. In this case, the signals can also be continuous or constant signals.

Block dialog with structured data ports

In the data port block dialogs, structured data ports are shown in a hierarchical tree view. The following illustration shows an example of a **Data Outport** block with two structured data ports:



Data Outport Block

Provides one or more data outports to send signals to a function port (ConfigurationDesk only), or to another behavior model (ConfigurationDesk or VEOS Player). In ConfigurationDesk, you can update the block based on the configuration of the connected function block with the Propagate to Simulink Model command. If you use this command, the block data is overwritten.

Name: Electrical

Type: Bus: <untyped>

Methods of creating data port blocks with structured data ports

In a Simulink model, you can create data port blocks with structured data ports for bus signals as follows:

- Creating structured data ports via the **Signal Configuration** page of a **Data Inport** or **Data Outport** block dialog. You have the following options:
  - Creating ports for typed bus signals by specifying a Simulink.Bus object as the data type.
  - Creating ports for untyped bus signals by using the button.
- Creating data port blocks from **Bus Creator** blocks.
- Creating data port blocks from Simulink.Bus objects via API command.

Tip

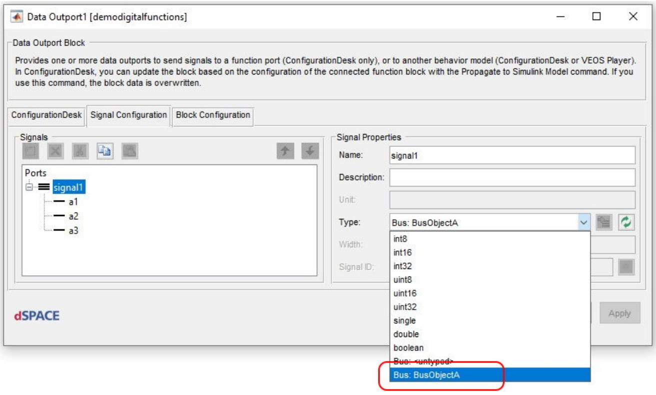
Alternatively, you can add a **Data Outport** block from the **Model Interface Blockset** to the model, connect a bus signal to its data port and then select the **Update Data Outport Block from Input Signals** command from the **Model Port Blocks** menu.

The Model Interface Package for Simulink also provides an API command for this purpose. Refer to dsmpb_updatemodelportblock (Model Interface Package for Simulink API Reference 🕮)

Creating ports for typed bus signals via block dialog

To create structured data ports from a Simulink.Bus object, you have to perform the following steps:

1. Create a Simulink.Bus object in the MATLAB base workspace or the model's Data Dictionary, for example, by using the Simulink `buseditor` command.
2. Select the Simulink.Bus object from the **Type** list on the **Signal Configuration** page of the data port's block dialog:



Data Outport1 [demodigitalfunctions]

Data Outport Block

Provides one or more data outports to send signals to a function port (ConfigurationDesk only), or to another behavior model (ConfigurationDesk or VEOS Player). In ConfigurationDesk, you can update the block based on the configuration of the connected function block with the Propagate to Simulink Model command. If you use this command, the block data is overwritten.

Signal Properties

Name: signal1

Description:

Type: Bus: BusObjectA

Note

- If you use Simulink.Bus objects with the Imported data scope, you must make sure that the data type definition for the Simulink.Bus object matches the one in your custom header file exactly. Otherwise, accessing variables on the real-time hardware can lead to incorrect results.
- **Data Inport** and **Data Outport** blocks support Simulink.Bus objects with up to 100,000 leaf signals. If any signal in a Simulink.Bus object is variable-size, the maximum number of signals is limited to 3000.
- You can update the listed Simulink.Bus objects in the dialog with the Simulink.Bus objects in the MATLAB base workspace and in the model's Data Dictionary via the ↻ button.
- You can also assign a Simulink.Bus object to the port of a data port block via the `dsmpb_set()` API command. Refer to dsmpb_set (Model Interface Package for Simulink API Reference 🕮).
- For an untyped bus port, you can create a Simulink.Bus object and assign it to the port with the ☰ button

Creating/updating model port blocks with structured data ports via ConfigurationDesk commands

Selecting Propagate to Simulink Model or Generate New Simulink Model Interface for a model port block with structured data ports in ConfigurationDesk creates or updates related model port blocks in the Simulink model. If a suitable Simulink.Bus object already exists in the MATLAB workspace, it is used as the data type for the Simulink model port block. If no suitable Simulink.Bus object exists, the Model Interface Package for Simulink can create one and assign it to the Simulink model port block. The automatic creation of a suitable Simulink.Bus object must be enabled using the `dsmpb_pref()` API command. To do this, enter the following command in the MATLAB Command Window:

```
dsmpb_pref('Set', 'CreateBusObjectsDuringPropagation', 'True');
```

For more information, refer to dsmpb_pref (Model Interface Package for Simulink API Reference 📖).

Tip

Simulink.Bus objects created this way can have more than 3.000 leaf signals. Additionally, code generation performance is improved.

Note

- Simulink.Bus objects can only be created if the names of all signals and all levels of the bus correspond to C identifiers. Otherwise, a message is displayed and no Simulink.Bus object is created. The data type of the port is then still Bus: <untyped>.
- Simulink.Bus objects that are created via Propagate to Simulink Model or Generate New Simulink Model Interface are not saved automatically with the Simulink model. You must save them, for example, in a MAT file.

Creating ports for untyped bus signals via block dialog

To create bus signals via the Signal Configuration page of a Data Inport or Data Outport block, you have to perform the following steps:

1. On the Signal Configuration page of a selected Data Inport or Data Outport block, select a signal.
2. Add new signals to the selected signal via the button.

   The data type of the selected signal automatically changes to Bus: <untyped>.

Creating data port blocks from Bus Creator blocks

To create a data port block from a Bus Creator block, you have to perform the following steps:

1. In the Simulink model, create a hierarchical bus structure using Bus Creator blocks.
2. Select the Bus Creator block.

3. Create a data port block using one of the following commands from the Model Port Blocks menu:
   - Create Data Outport Block from Bus Creator Block
   - Create Data Inport Block from Bus Creator Block

Creating data port blocks from Simulink.Bus objects via API command

To create a data port block from a Simulink.Bus object via an API command, you have to perform the following steps:

1. Create a Simulink.Bus object in the MATLAB base workspace or the model's Data Dictionary, for example, using the Simulink `buseditor` command.
2. Create a data port block for the Simulink.Bus object by using the `dsmpb_generatemodelportblock()` API command. For more information, refer to dsmpb_generatemodelportblock (Model Interface Package for Simulink API Reference 🕮).

Data port blocks with structured data ports in ConfigurationDesk

You can transfer data port blocks with structured data ports to ConfigurationDesk just like other data port blocks by analyzing the Simulink model. Refer to Data Interchange Between ConfigurationDesk and Behavior Model (ConfigurationDesk Real-Time Implementation Guide 🕮).

In ConfigurationDesk you can create model port blocks with structured data ports for function blocks with a structured function port interface. Refer to Creating Model Port Blocks With Structured Data Ports for Bus Signals (ConfigurationDesk Real-Time Implementation Guide 🕮).

Data port blocks with structured data ports in VEOS Player

In VEOS Player, the structured data ports are displayed in a hierarchical structure. Refer to Create Data Inport Block from Bus Creator Block (Model Interface Package for Simulink Reference 🕮).

Related topics

References

Create Data Inport Block from Bus Creator Block (Model Interface Package for Simulink Reference 🕮)
Create Data Outport Block from Bus Creator Block (Model Interface Package for Simulink Reference 🕮)
dsmpb_generatemodelportblock (Model Interface Package for Simulink API Reference 🕮)
dsmpb_updatemodelportblock (Model Interface Package for Simulink API Reference 🕮)

Basics of Updating Data Port Blocks with Structured Data Ports

Introduction

The Model Interface Package for Simulink lets you update data port blocks with structured data ports.

Methods of updating data port blocks

The **Model Interface Blockset** provides the following methods:

- Updating data port blocks from Simulink.Bus objects in the MATLAB workspace or the model's data dictionary.
- Updating **Data Outport** blocks from their input signals.

Updating data port blocks from Simulink.Bus objects

To update a data port block to match a specified Simulink.Bus object, you must use the `dsmpb_updatemodelportblock()` API command. Refer to dsmpb_updatemodelportblock (Model Interface Package for Simulink API Reference).

Updating Data Outport blocks from input signals

To update an existing **Data Outport** block from its input signals, you have to perform the following steps:

1. Connect the input signal to the **Data Outport** block.
2. Update the **Data Outport** block using the **Update Selected Outport Blocks from Input Signals** command of the **Model Port Blocks** menu:

> **Note**
>
> - This command is not available in Simulink libraries.
> - You can also use the `dsmpb_updatemodelportblock()` API command to update **Data Outport** blocks from their input signals. Refer to dsmpb_updatemodelportblock (Model Interface Package for Simulink API Reference).

Update results

Updating a data port block provides the following results:

- During the update, each data port is completely restructured to match the hierarchical structure of the connected block.
- All data ports contained in the new structure are analyzed with regard to their full paths within the new structure. If a path matches the path of a data port in the old structure, the signal ID is reused. If no match is found, the existing signal IDs are assigned to the updated signals according to the order of the old signals (for information on signal IDs, refer to Basics on Model Port Block IDs and Signal IDs on page 63).
- Each structured data port which is connected to a block that no longer outputs a Simulink bus signal becomes an unstructured data port and vice versa. In this case, the ID of the root node of the structured signal is assigned to the unstructured data port or vice versa.

Note

There are some cases that may cause errors during the update process. No update is performed if one of the following cases applies:

- One of the data ports is not connected to a block.
- Not all properties of the incoming signals can be determined.
- One of the signals is unsupported.
- The model initialization results in an error.

Related topics

References

Update Selected Outport Blocks from Input Signals (Model Interface Package for Simulink Reference 🕮)

Creating a Suitable Simulink Block Periphery for Unconnected Data Ports

Motivation

The Model Interface Package for Simulink provides methods for creating suitable peripheral Simulink blocks for unconnected model ports ⧉ automatically.

This is especially useful for data port blocks with structured data ports. With a valid bus signal, the Simulink model can be initialized. The **Create Model Port Block Periphery** command creates the suitable bus hierarchy. Afterwards you can replace the created peripheral Simulink blocks with the suitable signals.

Command access

You can execute the **Create Model Port Block Periphery** command for a selected model port block via the **Model Port Blocks** menu.

Note

Alternatively, you can use an API command to create peripheral Simulink blocks for a model port block. Refer to dsmpb_createmodelportblockperiphery (Model Interface Package for Simulink API Reference 🕮).

Overview of created blocks

If you execute the **Create Model Port Block Periphery** command, a peripheral Simulink block is generated for each unconnected data port of the selected model port blocks. Each newly created block is connected to the related data port. The following table gives an overview of the blocks that are created for unconnected model ports, depending on the model port type:

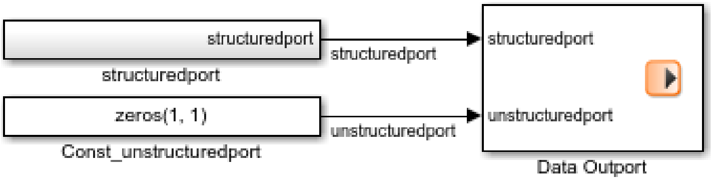
Model Port Type	Created Model Port Block Periphery
Unstructured data outport	Simulink Constant block with the following properties: ▪ Value = zeros(1, <Width>). ▪ Type is derived from the connected data port.
Structured data outport	Simulink Subsystem block that contains a representation of the hierarchical bus structure. This includes Constant blocks, Bus Creator blocks, and Outport blocks as well as suitably named signals.
Unstructured data inport	Simulink Terminator block
Structured data inport	Simulink Subsystem block that contains a representation of the hierarchical bus structure. This includes Terminator blocks, Bus Selector blocks, and Outport blocks as well as suitably named signals.

Naming scheme of the created blocks

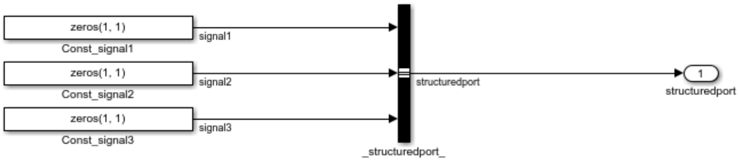
The names of the new peripheral blocks are derived from the data ports for which they are created. In addition, new Constant blocks and Terminator blocks get an appropriate `Const_` or `Term_` prefix. In the case of a naming conflict, the new block gets a number as a suffix, which is increased if necessary.

Example

The following illustration shows a Data Outport block with a structured and an unstructured data port as well as the block periphery created for it:



The structured Port subsystem contains the blocks and signals as shown in the following illustration:



Related topics

References

dsmpb_createmodelportblockperiphery (Model Interface Package for Simulink API Reference 🕮)

Points to Note When Using Simulink In Bus Element and Out Bus Element Blocks

Benefits of In Bus Element and Out Bus Element blocks

You can use Simulink In Bus Element and Out Bus Element blocks on the root level of a Simulink model. An In Bus Element block combines an Inport block with a Bus Selector block. An Out Bus Element block combines an Outport block with a Bus Creator block. This lets you simplify the interfaces of Simulink behavior models and subsystems.

For more information on In Bus Element and Out Bus Element blocks, refer to the Simulink documentation.

Placement in the Simulink model

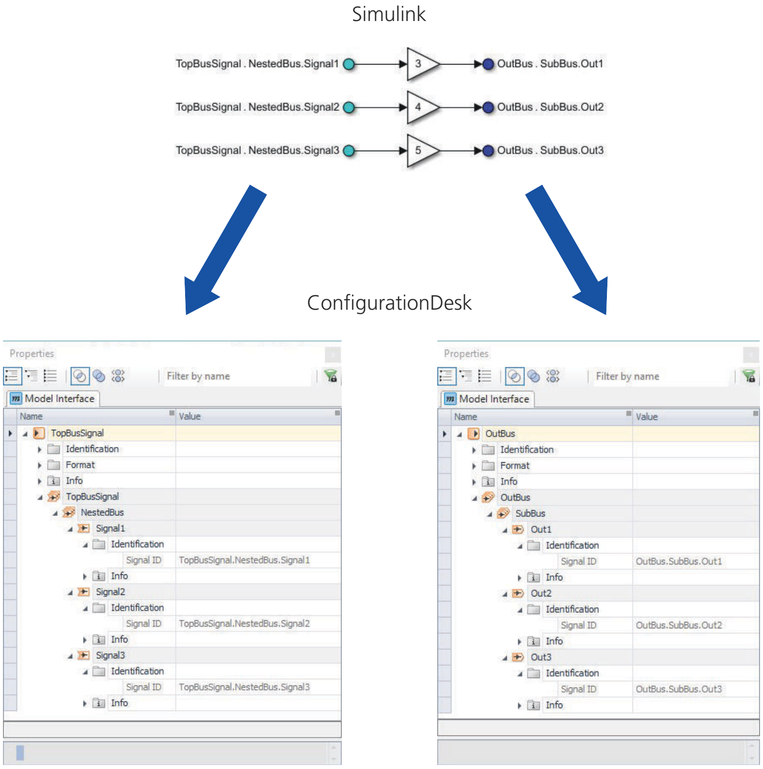
To specify a model interface, In Bus Element and Out Bus Element blocks can be placed in a Simulink model as follows:

- On the root level of the model.
- At the top level of a subsystem you want to separate from the overall model to make it a new model.

Simulink models that contain these blocks can be used in ConfigurationDesk. Simulink implementation containers (SIC files) generated from the models can be used in ConfigurationDesk or the VEOS Player.

Representation in ConfigurationDesk

If you add a Simulink model that has In Bus Element or Out Bus Element blocks on its root level to a ConfigurationDesk application, ConfigurationDesk creates related structured or unstructured model port blocks, depending on the hierarchies of the actual signals. The hierarchy of a resulting model port block is displayed in the **Properties Browser**. Refer to the following illustration:



Name	Value
TopBusSignal	
Identification	
Format	
Info	
TopBusSignal	
NestedBus	
Signal1	
Identification	
Signal ID	TopBusSignal.NestedBus.Signal1
Info	
Signal2	
Identification	
Signal ID	TopBusSignal.NestedBus.Signal2
Info	
Signal3	
Identification	
Signal ID	TopBusSignal.NestedBus.Signal3
Info	

Name	Value
OutBus	
Identification	
Format	
Info	
OutBus	
SubBus	
Out1	
Identification	
Signal ID	OutBus.SubBus.Out1
Info	
Out2	
Identification	
Signal ID	OutBus.SubBus.Out2
Info	
Out3	
Identification	
Signal ID	OutBus.SubBus.Out3
Info	

The **Show in Simulink Model** command in ConfigurationDesk highlights all the In Bus Element or Out Bus Element blocks that belong to the related bus signal.

Note

The signal ID of a bus element corresponds to its signal path. If you modify the block path of an In Bus Element or Out Bus Element block that has a related model port block in a ConfigurationDesk application, the model port block in ConfigurationDesk is unresolved after a model analysis. After you modified the block path of an In Bus Element or Out Bus Element block in Simulink, you must delete the related model port block in ConfigurationDesk and analyze the model again.

Restrictions

When you work with In Bus Element and Out Bus Element blocks in a behavior model, the following restrictions apply:

- The analysis of a model that contains root-level In Bus Element and Out Bus Element blocks requires an initialization of the model. Therefore, analyzing the model interface via the **Analyze Simulink Model (Model Interface Only)** command is not possible.
- In the following case, the results of the model analysis and the code generation differ:
  - The signal of an In Bus Element block is specified by a Simulink.Bus object.

- The In Bus Element block is connected to a Bus Selector block that selects a subset of the bus signals.
- No other signal is used.

In the above case, a model port block with the complete bus hierarchy is displayed in ConfigurationDesk after a model analysis. However, after code generation, a model port block with only those bus signals that are selected by the Bus Selector block is displayed in ConfigurationDesk.

Related topics

Basics

Basics on Simulink Implementation Containers..262

HowTos

How to Generate a Simulink Implementation Container..264

Working with Runnable Functions and Function Triggers

Introduction

When implementing the behavior model that you want to use with ConfigurationDesk, you should be familiar with function triggers and the handling of runnable functions.

Where to go from here

Information in this section

Introduction to Runnable Functions and Function Triggers

Purpose

Describes the basics of runnable functions and function triggers, and provides information on the effects of different Simulink tasking modes in ConfigurationDesk.

Where to go from here

Information in this section

Basics on Runnable Functions

Definition of runnable functions

A behavior model provides runnable functions. A runnable function is a function that must be called within a task to compute specific results.

Runnable functions provided by a behavior model

The runnable functions provided by the behavior model result either from different sample rates specified in the behavior model, or from the following blocks of the Model Interface Blockset:

- Hardware-Triggered Runnable Function block

  The Hardware-Triggered Runnable Function block exports a function-call subsystem as a runnable function. The runnable function can be used in ConfigurationDesk for modeling asynchronous tasks.

- Software-Triggered Runnable Function block

  The Software-Triggered Runnable Function block exports function-call subsystems as runnable functions in predefined tasks for use in ConfigurationDesk or VEOS Player.

  If you use this block, the runnable function is called each time the block is executed in the generated model code. That means the runnable function is assigned to a predefined task that is triggered by a software event provided by the model. In this case, the runnable function cannot be assigned to an arbitrary task triggered by an I/O event in ConfigurationDesk, for example.

For information on the how to configure runnable function blocks, refer to How to Configure Runnable Function Blocks on page 84.

> **Note**
>
> - The names of the runnable functions must be unique in the behavior model.
> - The Priority value specifies the task priority restriction. When you configure the tasks to which the runnable functions are assigned in ConfigurationDesk, you must make sure that the relationship between the task priorities matches the relationship between the predefined priorities, even if the absolute priority values differ from the predefined values. For more information, refer to Basics on Modeling Tasks in ConfigurationDesk (ConfigurationDesk Real-Time Implementation Guide 📖))
> - The priority order is always *Higher priority value indicates lower task priority*. You cannot switch the priority order. The *Higher priority value indicates higher task priority* option in the Solver dialog of the Simulink® Coder™ is automatically cleared when you start the build process.

Working with runnable functions in ConfigurationDesk

The runnable functions provided by a behavior model are available in ConfigurationDesk after a complete model analysis. For details, refer to Analyze Simulink Model (Including Task Information) (ConfigurationDesk User Interface Reference 📖). You can then assign them to periodic or asynchronous tasks. For basics on modeling tasks in ConfigurationDesk, refer to Basics on Tasks, Events, and Runnable Functions (ConfigurationDesk Real-Time Implementation Guide 📖).

How to Configure Runnable Function Blocks

Objective

By adding a Hardware-Triggered Runnable Function block or a Software-Triggered Runnable Function block from the Model Interface Blockset ↗ to your model, you get an instance of the block with a default configuration that you can adapt to your needs.

Default configuration of runnable function blocks

A runnable function block with its default configuration provides one runnable function port named *Asynchronous Function*. The task priority is set to 30, and the sample time of the block is inherited.

Dependencies to ConfigurationDesk

The block and runnable function configurations take effect in ConfigurationDesk after you execute the Analyze Simulink Model (Including Task Information) command in ConfigurationDesk. The settings are only displayed in ConfigurationDesk and cannot be edited there.

Method

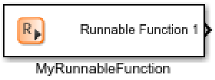
To configure a runnable function block

1. In the behavior model, you can edit the name according to the naming conventions for Simulink blocks.
2. Double-click the block to open the block dialog for the runnable function block.
3. On the Runnable function page, specify the name of the runnable function and the priority value in the range 0 … 127.
4. On the Block Configuration page, specify the sample time and a description. In addition, you can select to display the signal and block IDs in the behavior model.
5. Click OK to apply the settings and close the dialog.

Result

You have configured the runnable function block. There are some settings, such as the sample time of the block or the priority, that are used for code generation. Other settings are useful for passing information to the ConfigurationDesk user.

In the figure below, you can see a Hardware-Triggered Runnable Function block with modified names for the block and the runnable function as an example.



Related topics

HowTos

How to Configure Data Port Blocks..58

Different Simulink Tasking Modes and Their Effects in ConfigurationDesk

Introduction

Simulink lets you select either the single-tasking mode or the multitasking mode for the behavior model. You can switch between the modes via the Treat each discrete rate as a separate task checkbox on the Solver page of the Configuration Parameters dialog. You should know which elements the behavior model provides for modeling executable applications depending on the selected tasking mode.

Components resulting from different Simulink tasking modes

Regardless of the tasking mode, the behavior model provides the following elements for modeling an executable application:

- A runnable function for the Simulink base rate task (no predefined task).
- A runnable function without predefined task for each Hardware-Triggered Runnable Function block in the behavior model.
- A predefined task including a runnable function and with an assigned software event for each Software-Triggered Runnable Function block in the behavior model.

Single-tasking mode In single-tasking mode, no further components are provided. The calculation of all periodic parts of the model is performed in the runnable function for the Simulink base rate.

Multitasking mode In multitasking mode, the different sample rates of the behavior model which do not equal the base rate are available as predefined tasks in ConfigurationDesk, because Simulink triggers periodical software events to execute these tasks.

To make predefined elements available in ConfigurationDesk, you have to execute the `Analyze Simulink Model (Including Task Information)` command for the behavior model.

Related topics

References

Analyze Simulink Model (Including Task Information) (ConfigurationDesk User Interface Reference 🕮)

Basics on Function Triggers

Introduction

If you are familiar with function triggers, you know the timing of the I/O access and you can avoid trigger problems caused by multiple trigger sources.

I/O access bound to function modules

I/O access is bound to the execution of so-called *function modules*. Function modules are always part of a task. The following parts of a behavior model define a function module:

Data port blocks (from the Model Interface Blockset) with the same sample time A function module consists of all the blocks in the model that have the same sample time, if at least one data port block has the following characteristics:

- The block's sample time matches the sample time.
- The block is not contained in an atomic subsystem.

If these conditions are fulfilled, all the data port blocks with the same characteristics are assigned to the function module.

Data port blocks (from the Model Interface Blockset) in atomic subsystems with the same sample time A function module consists of all the blocks in an atomic subsystem that have the same sample time, if at least one data port block has the following characteristics:

- The block's sample time matches the sample time.
- The block is contained in the atomic subsystem. The relevant atomic subsystem is the one directly above the data port block. For example, in the hierarchy

  `Model/Atomic1/Virtual1/Atomic2/Virtual2/MyBlock`

  the `MyBlock` data port block defines a function module for the `Atomic2` subsystem, not for `Atomic1`.

If these conditions are fulfilled, all the data port blocks with the same characteristics are assigned to this function module.

Simulink Inport/Outport blocks with the same sample time A function module can consist of all the blocks in the model that have the same sample time, if at least one Simulink Inport/Outport has the following characteristics:

- The block's sample time matches the sample time.
- The block resides on the root level of the model.

If these conditions are fulfilled, all the Simulink Inports/Outports with the same characteristics are assigned to the function module.

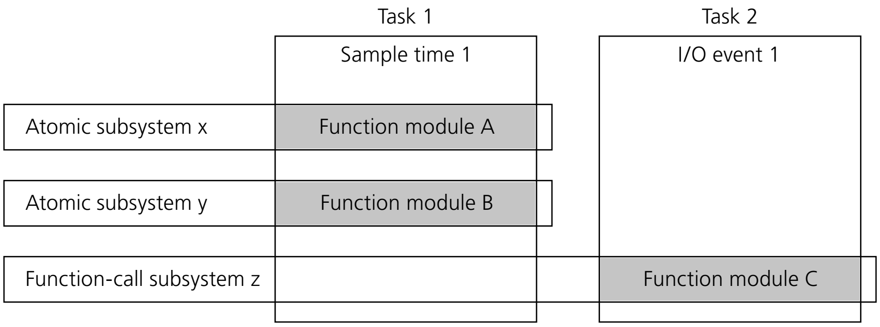
Note

> Data port blocks not contained in atomic subsystems and Simulink Inports/Outports on the root level of the behavior model define separate function modules, even if the blocks' sample times are identical.

Example of function modules

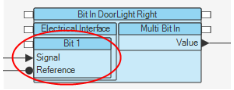
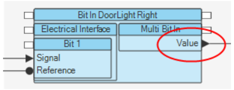
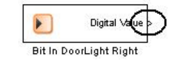
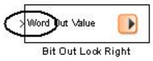
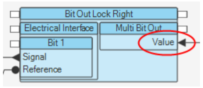
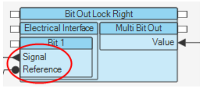
Suppose your behavior model contains three atomic subsystems, and each subsystem contains at least one data port block. Two of these atomic subsystems are calculated with the same Sample time 1. The third atomic subsystem is a function-call subsystem connected to a Hardware-Triggered Runnable Function block. In ConfigurationDesk, the associated runnable function is assigned to a task that is triggered by an I/O event.

Code generation would result in Function module A and Function module B, both of which are part of Task 1. Function module C is part of asynchronous Task 2.



Timing of the I/O access

The table below describes the relationship between the task triggers, the function triggers, and the port status, that is, the temporal characteristics of I/O access. It is assumed that the task consists of one function module.

Execution Step During Real-Time Simulation		Resulting Port Status	
1.	Task is triggered. The functions are triggered to update the values of the signal inports, i.e., the input signals are sampled.	New values are available at signal inports in ConfigurationDesk.	 Bit In DoorLight Right Electrical Interface Multi Bit In Bit 1 Value Signal Reference
2.	Execution of the task begins. Execution of the function module begins. The functions that are mapped to all the data inports assigned to the function module are triggered to update the values of the function outports.	New values are available at function outports in ConfigurationDesk.	 Bit In DoorLight Right Electrical Interface Multi Bit In Bit 1 Value Signal Reference
3.	Data read[1)]	New values are available at data inports in the behavior model.	
	Execution of model code[1)]	New values are available at data outports in the behavior model.	
	Data write[1)]	New values are available at function inports in ConfigurationDesk.	 Bit Out Lock Right Electrical Interface Multi Bit Out Bit 1 Value Signal Reference
4.	Execution of the function module finishes. The functions that are mapped to all the data outports assigned to the function module are triggered to	New values are available at signal outports in ConfigurationDesk.	 Bit Out Lock Right Electrical Interface Multi Bit Out Bit 1 Value Signal Reference

Execution Step During Real-Time Simulation		Resulting Port Status	
	update the values of the signal outports.		

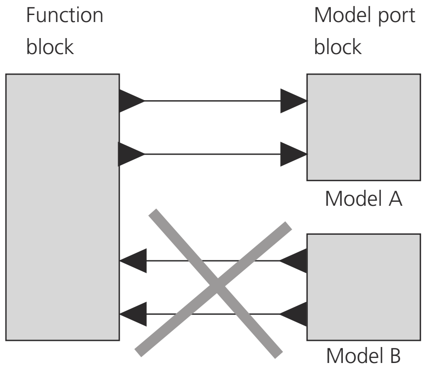
[1] For details on the execution order of blocks, refer to the Simulink Coder documentation from MathWorks.

Note

I/O access is performed only during real-time simulation. During Simulink simulation, Data Inports output the Default value and Data Outports do not output any signal at all.

Multi-model restriction

In general, model port blocks that reside in different behavior models must not be mapped to the same function block.



However, the following function blocks are an exception to this rule if the mapped model port blocks belong to the same application process:

- Bus Configuration block
- CAN block
- LIN block
- Gigalink block
- Power On Signal In block
- Non-Volatile Memory Access block

For an overview of the mapping rules, refer to Rules for Model Port Mapping (ConfigurationDesk Real-Time Implementation Guide 📖).

Malfunction due to multiple trigger sources

When a function block for which multiple trigger sources are not allowed is triggered by multiple function modules, a conflict is displayed in the Conflicts Viewer.

You have to resolve this conflict before you start the build process to get proper build results. You should avoid multiple function triggers. Refer to How to Avoid Multiple Function Triggers on page 90.

Note

The assignment of a data port to its associated function module can be determined only during the model code generation and not during the model analysis. If you move data port blocks in the behavior model from one subsystem to another, this action might trigger the "Function block: Multiple triggering" conflict even if the reason for this conflict was eliminated and the model was re-analyzed in ConfigurationDesk. Generating the model code (either by starting the build process or by generating it explicitly) updates this information and resets the conflict.

Related topics

Basics

Implementing I/O Functionality (ConfigurationDesk Real-Time Implementation Guide 🕮)
Modeling Concept.. 16
Specifying the Model Interface (ConfigurationDesk Real-Time Implementation Guide 🕮)

How to Avoid Multiple Function Triggers

Objective

To avoid invalid data transfer to or from the behavior model.

Invalid data transfer

If a function has multiple trigger sources, default code is generated during the build process instead of task-based read/write access. In consequence, only initial values are transferred to or from the behavior model.

Method

To avoid multiple function triggers

1 Make sure that a function is triggered by only one function module. You can use one of the following methods:

- ConfigurationDesk:

  Change the mapping of functions and model port blocks in ConfigurationDesk.

- Behavior model:

  Move the model port blocks into appropriate subsystems in the behavior model.

Result

The function has only one valid trigger source.

Related topics

Basics

Assigning Functions of ConfigurationDesk to Tasks

Introduction

To relate function execution to specific sample rates or trigger events.

Where to go from here

Information in this section

How to Assign Functions to Periodic Tasks

Objective

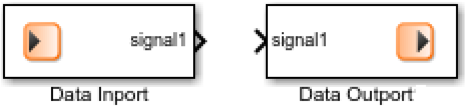
If you want the behavior model to retrieve data from or send data to the ConfigurationDesk function blocks at specific periodic sample rates, you must use **Data Inport** and **Data Outport** blocks with those sample rates.

Tip

You can use all Simulink and Simulink® Coder™ implementation options for periodic tasks without any restrictions.

Preconditions

An interface model has been generated from the model port blocks that are connected to the function block. It contains the required Data Inport and Data Outport blocks. For information on generating interface models, refer to How to Transfer Unresolved Model Port Blocks to a Simulink Behavior Model via an Interface Model (ConfigurationDesk Real-Time Implementation Guide).



Possible methods

There are two ways to assign functions to periodic tasks:

- Specify a sample rate for the Data Inport and Data Outport blocks, refer to Method 1.
- Let the Data Inport and Data Outport blocks inherit the sample time from the model or an atomic subsystem, refer to Method 2.

Method 1

To assign functions to periodic tasks by specifying explicit sample times

1 Copy and paste the data port blocks from the interface model into the behavior model (root level) by using Simulink's standard Copy operation together with the Paste and Keep IDs command from the Model Port Blocks menu.

> **Note**
>
> If multiple data port blocks are connected to the same function in ConfigurationDesk, they must have the same sample time and reside in the same atomic subsystem.

2 Double-click the data port block to open its dialog.

3 Select the Block configuration page.

4 Specify the Sample time as an integer multiple of the fixed step size (base sample time) in seconds.

In addition, you can specify a sample time offset. The syntax is as follows:

`[Sample time, offset]`, e.g., `[0.01,0.002]`

The offset must be an integer multiple of the base sample time, and must be smaller than the sample time value.

5 Click OK.

Simulink associates the data port block with a task that executes at its sample rate.

Method 2

To assign functions to periodic tasks by sample time inheritance

1 Copy and paste the data port blocks from the interface model into the behavior model, i.e., into the atomic subsystem that provides the right sample time. Use Simulink's standard Copy operation together with the Paste and Keep IDs command from the Model Port Blocks menu.

> **Note**
>
> If multiple data port blocks are connected to the *same* function in ConfigurationDesk, they must have the same sample time and reside in the same atomic subsystem.

2. Double-click the data port block to open its dialog.
3. Select the Block configuration page.
4. Specify the Sample time as `-1`.
5. Click OK.

   Simulink associates the data port block with a task that executes at its sample rate.

Result

You have assigned functions to periodic tasks.

Next step

If blocks with different sample times are connected via signal lines in the behavior model, they need an appropriate coupling mechanism. You can use all Simulink and Simulink Coder options to model rate transitions:

- Rate Transition block
- Unit Delay block
- Zero-Order-Holder block
- Automatically handle rate transition for data transfer option on the Solver page of the Configuration parameters dialog (`Ctrl` + `E`).

Related topics

Basics

Basics on Function Triggers.. 86

HowTos

How to Configure Periodic Tasks in the Behavior Model... 96
How to Transfer Unresolved Model Port Blocks to a Simulink Behavior Model via an Interface Model (ConfigurationDesk Real-Time Implementation Guide 🕮)

References

Data Inport Block (Model Interface Package for Simulink Reference 🕮)
Data Outport Block (Model Interface Package for Simulink Reference 🕮)

How to Trigger Function-Call Subsystems by Asynchronous I/O Events

Objective

To trigger function-call subsystems in the behavior model by using I/O events generated in ConfigurationDesk, you must connect a **Hardware-Triggered Runnable Function** block to the function-call subsystem. After a complete model analysis, the function-call subsystem is available as a runnable function in ConfigurationDesk. You must then assign the runnable function and the I/O event to a task.

Preconditions

- You must have created a Simulink model containing a function-call subsystem.
- The Model Interface Blockset must be open.

Method

To trigger function-call subsystems by asynchronous I/O events

1. Copy a Hardware-Triggered Runnable Function block from the Model Interface Blockset to your Simulink model containing the function-call subsystem.
2. Connect the outport of the Hardware-Triggered Runnable Function block to the function() inport of the function-call subsystem.

> **Note**
>
> The following limitations apply:
> - The function() inport of the function-call subsystem must be scalar.
> - You cannot connect multiple Hardware-Triggered Runnable Function blocks to the same function-call subsystem via a Simulink Mux block.
> - You cannot connect a Hardware-Triggered Runnable Function block to multiple function-call subsystems.

3. Double-click the Hardware-Triggered Runnable Function block to open its dialog.
4. Specify the block parameters, for example, the priority. For details on the block parameters, refer to Hardware-Triggered Runnable Function Block (Model Interface Package for Simulink Reference).

   The next steps are performed in ConfigurationDesk. Refer to How to Model Asynchronous Tasks for Models with Runnable Function Blocks (Manually) (ConfigurationDesk Real-Time Implementation Guide) and perform the steps described there.

Result

The code that is generated for the blocks in the function-call subsystem is executed during real-time simulation when the I/O event occurs that is assigned to the task that executes the runnable function.

Note

For simulations in Simulink, the function-call subsystem is executed at the sample time of the Hardware-Triggered Runnable Function block.

Configuring Tasks in MATLAB/Simulink®

Purpose

To configure periodic tasks.

How to Configure Periodic Tasks in the Behavior Model

Objective

To specify the sample time, offset, and priority values of periodic tasks.

Preconditions

The behavior model is open and contains blocks with different sample rates, and multitasking is enabled.

Possible methods

There are two ways you can prioritize such model tasks:

- Prioritize tasks manually, refer to Method 1.
- Let Simulink automatically prioritize tasks, refer to Method 2.

Method 1

To configure periodic tasks in the behavior model by specifying priority values

1. Open the Configuration Parameters dialog (**`Ctrl`** + **`E`**) and select the Solver page.
2. Set the Periodic sample time constraint to Specified.
3. Specify the Sample time properties parameter, which comprises the sample time, offset, and priority values of all periodic tasks in the behavior model, for example:

   ```
   [[0.1, 0, 10]; [0.2, 0, 11]; [0.3, 0, 12]]
   ```

   > **Note**
   >
   > Faster sample times must have higher priorities, i.e., lower priority values.

   For details, refer to the documentation from MathWorks®.
4. Do not select Higher task priority value indicates higher task priority because the Model Interface Package for Simulink ignores and automatically clears it when building the real-time application.

Method 2

To configure periodic tasks in the behavior model by applying a default algorithm

1. Open the Configuration Parameters dialog (**`Ctrl+E`**) and select the Solver page.

2 Set the Periodic sample time constraint to Unconstrained or Ensure sample time independent.

   Simulink automatically prioritizes all the tasks in the behavior model according to a default algorithm. By default, Simulink® Coder^{TM} assigns a priority value of 39 or 40, depending on whether the model contains blocks with continuous sample times, to the fastest periodic task. All the other periodic tasks are assigned lower priorities.

3 Do not select Higher task priority value indicates higher task priority,, because the Model Interface Package for Simulink ignores and automatically clears it when building the real-time application.

Result

You have configured all the periodic tasks in the behavior model.

Separating Models from an Overall Model to Build Multimodel Applications

Introduction

Model separation enables you to separate models from an overall model in MATLAB/Simulink so that you can execute them on separate cores or, in the case of multimodel application processes, on one core of dSPACE real-time hardware (MicroAutoBox III or SCALEXIO systems). To define and create the models to be separated, dSPACE provides the **Model Separation Setup** tool. You need to know the workflow for separating models and how to handle the **Model Separation Setup** tool.

Note

The Model Interface Package for Simulink lets you generate Simulink implementation container files (SIC files) for the separated models, and use them in ConfigurationDesk or VEOS Player. If you work with SIC files, you are not recommended to use the MCD file with ConfigurationDesk, because the MCD file references the separated Simulink models, not the SIC files you generated for them. The MCD file cannot be used in VEOS Player.

Where to go from here

Information in this section

Basics on Separating Models from an Overall Model

Use scenario

Suppose you want to separate models from an overall model to build a multimodel application that you can run on the single cores of a single-processing-unit system or a multi-processing-unit-system. dSPACE provides the **Model Separation** tool for you to separate top-level subsystems as models from an overall model. The overall model can then still be used for offline simulations.

Note

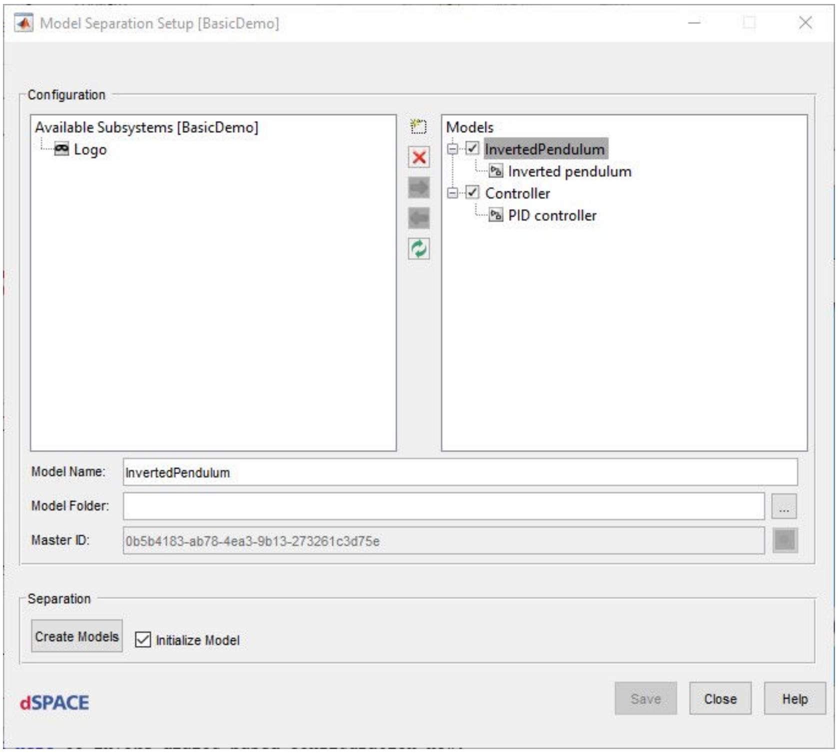
- In addition to top-level subsystems, you can also separate Stateflow charts, MATLAB functions, or truth tables on the model's top-level for model separation..
- Since ConfigurationDesk supports multi-processing-unit applications that can be executed on the single cores of a multi-processing-unit system, you can generally separate an unlimited number of models. However, if you want to run your real-time application on the cores of a single-processing-unit system, the maximum number of models to be separated is (number of cores)-1, since one core is reserved for the execution of system services.

  For multi-processing-unit systems, the maximum number of models to be separated is (number of cores)-(number of processing units), since each processing unit needs one core for the execution of system services.

  If you separate more models than the maximum allowed number, the download of the resulting multimodel application in ConfigurationDesk is aborted with an error message.
- The workflow described in this chapter is not suitable for very large overall models. It is recommended to build the single models from scratch and add them to your ConfigurationDesk application one by one instead. Refer to Workflow for Creating a Multicore Real-Time Application Using Multiple Behavior Models (ConfigurationDesk Real-Time Implementation Guide 📖) and Workflow for Creating a Multi-PU Application Using Multiple Behavior Models (ConfigurationDesk Real-Time Implementation Guide 📖).

Model Separation Setup

If you click the Model Separation Setup block, the Model Separation Setup opens:



Model Separation Setup [BasicDemo]
Configuration
Available Subsystems [BasicDemo]
Models
Separation
Create Models
Save
Close
Help
dSPACE

Available Subsystems [BasicDemo]
- Logo

Models
- ☑ InvertedPendulum
  - Inverted pendulum
- ☑ Controller
  - PID controller

Model Name: InvertedPendulum

Model Folder:

Master ID: 0b5b4183-ab78-4ea3-9b13-273261c3d75e

☑ Initialize Model

The Model Separation Setup lets you separate top-level subsystems as models from an overall model. It lists all the top-level subsystems that can be separated as models. You can define model entries in the Models tree and assign one or more top-level subsystems to them. Each direct connection (Simulink signal line) between two top-level subsystems that are assigned to different models is regarded as model communication. You can create the separated models by clicking Create Models.

Note

To open the Model Separation Setup, you can also select the Open Model Separation Setup command from the Model Port Blocks menu.

Create models

The Create Models command creates the models you selected in the Models tree and saves them as MDL or SLX files, depending on the Simulink preferences. The models contain the assigned top-level subsystems. For each model interface, appropriate data port blocks from the Model Interface Blockset are created and connected to the subsystems' inports and outports. In addition to the separated models, a model communication description file (MCD file) is generated. The MCD file contains information on the separated models and their connections. You can import the MCD file into ConfigurationDesk to add the separated models to a ConfigurationDesk application. Refer to How to Import a Model Topology (ConfigurationDesk Real-Time Implementation Guide 📖).

If separated models are connected via Simulink bus signals, the separated models contain **Data Inport**/**Data Outport** blocks with structured data ports. Suitable information on these interconnections is written to the model communication description file (MCD file). After you import the MCD file to ConfigurationDesk, suitable model port blocks with structured data ports and their interconnections are created in ConfigurationDesk. The block IDs and signal IDs of newly created **Data Inport**/**Data Outport** blocks are derived from the master ID that is displayed in the **Model Separation Setup**.

Separating models with Goto/From block connections If you perform model separation for top-level subsystems that include **Goto** or **From** blocks, the **Model Separation Setup** tool creates new blocks on the top level of the separated models. The following blocks are created during model separation:

- For each **Goto** block in a subsystem that must be separated, a **Data Outport** block and a suitable **From** block are created on the top level of the separated model. The **From** block is connected to the **Data Outport** block.
- For each **From** block in a subsystem, a **Data Inport** block and a suitable **Goto** block are created on the top level of the separated model. The **Goto** block is connected to the **Data Inport** block.

Note

- If you separate only one model from the overall model, no MCD file is created.
- As a general rule, all signals connected to a data port block must have the same sample time as the data port block. This also applies to the individual signals of a bus signal connected to a data port block. An exception to this rule is signals connected to a data port block running at the fastest sample time of the model. In this case, the signals can also be continuous or constant signals.

Standard workflow

The standard workflow for separating models from an overall model in MATLAB/Simulink and importing the new model topology in ConfigurationDesk consists of the following steps:

1. Preparing the overall model to fulfill the preconditions for model separation, if needed. For details, refer to Preparing the Overall Model for Model Separation on page 103.
2. Defining the models to be separated. Refer to Defining the Models to be Separated on page 105.
3. Creating the models and a model communication description file (MCD file). Refer to Creating Models and Generating a Model Communication Description File on page 107.
4. Importing the MCD file in ConfigurationDesk to add the new model topology to a ConfigurationDesk application. Refer to Adding the Separated Models to a ConfigurationDesk Application on page 110.

Model separation via API command

The Model Interface Package for Simulink provides the `dsmsb_separate()` API command for separating models. Refer to dsmsb_separate (Model Interface Package for Simulink API Reference 🕮).

Related topics

Basics

Creating Models and Generating a Model Communication Description File.............................. 107
Defining the Models to be Separated.. 105

HowTos

How to Define Models to be Separated from an Overall Model.. 106
How to Separate Models from an Overall Model... 109

References

Model Separation Setup Block (Model Interface Package for Simulink Reference 🕮)

Preparing the Overall Model for Model Separation

Introduction

The structure of the overall model and the model communication must fulfill several preconditions for model separation.

Preconditions for the overall model structure

The following preconditions must be fulfilled in the overall model:

- All the model parts you want to separate as models must reside in top-level subsystems.
- You must have saved the overall model at least once.

Note

The overall model can contain model parts that are not intended to be executed on single cores of the dSPACE real-time hardware.

Preconditions for subsystems to be assigned

The subsystems you want to separate as models must fulfill the following preconditions:

- The Read/Write permissions of the subsystems must be set to ReadWrite or ReadOnly.
- The subsystems must not be configurable.
- The subsystems must not be commented out or commented through.

- The subsystems must not be instances of Simulink library blocks or Model Interface Package for Simulink library blocks.
- The subsystems must not have a have a function-call, action, state, reset, or connection port.
- A subsystem must be assigned to exactly one target model.

Note

If the one or more of the above preconditions are not fulfilled, model separation is not performed and an error message is displayed.

Preconditions for model communication

You must specify the model communication via the **Inport** and **Outport** blocks of the top-level subsystems and their connections. Each direct connection (Simulink signal line) between inport and outport of two top-level subsystems that are assigned to different models is regarded as a model communication. The inports and outports of the top-level subsystems must fulfill the following preconditions:

- To avoid model initialization (for example, because it takes too long), all inports and outports of the subsystems must have an explicitly specified scalar or vectorial port width and a supported data type..

  The following block parameter settings must be made:

  - Sampling mode = Sample based
  - Signal type = real

  The following data types are supported:

  - double
  - single
  - uint64
  - int64
  - uint32
  - int32
  - uint16
  - int16
  - uint8
  - int8
  - Boolean
  - A Simulink.Bus object defined in the base workspace or in the model's data dictionary

Note

You can specify I/O interfaces via the inports and outports of the top-level subsystems. All inports and outports of a top-level subsystem that are not part of a model communication are interpreted as I/O interfaces. However, there is one exception to this rule: If two top-level subsystems are assigned to the same model entry, and the Inport of one subsystem is directly connected to the Outport of the other subsystem, these ports are not considered as I/O interfaces, i.e., no **Data Inport** and **Data Outport** blocks are created for these ports. The direct connection is transferred to the separated model.

Configuring model communication related sample times

The model port blocks which are generated in the separated models in order to perform model communication are configured with the same sample times as their related subsystem inports/outports. Thus, to configure the sample time for model communication, you have to configure the sample times of the top-level subsystem inports/outports.

Defining the Models to be Separated

Model properties

The **Model Separation Setup** lets you specify names and folders for the models to be separated. You can assign one or more top-level subsystems to each model to be separated.

For detailed instructions, refer to How to Define Models to be Separated from an Overall Model on page 106.

Default values

If you do not specify the properties of a model to be separated, the model entry has the following default configuration:

- A name following the conventions for Simulink names is assigned to the model entry.
- No model folder is specified (i.e., the model is created in the folder of the overall model).
- No top-level subsystem is assigned to the model entry.
- The model is selected for model separation.

Related topics

Basics

Basics on Separating Models from an Overall Model........ 99
Preparing the Overall Model for Model Separation........ 103

How to Define Models to be Separated from an Overall Model

Objective

Before model separation, you must define the models to be separated from the overall model via the Model Separation Setup. You must specify names and folders, and assign one or more top-level subsystems to the models.

Preconditions

- The overall model must be open.
- The preconditions for model separation must be fulfilled in the overall model. Refer to Basics on Separating Models from an Overall Model on page 99.

Method

To define models to be separated from an overall model

1 Double-click the Model Separation Setup block in the overall model to open the Model Separation Setup.

> **Tip**
>
> - Alternatively, you can open the Model Separation Setup via the Open Model Separation Setup command in the Model Port Blocks menu.
> - It is not necessary to copy the Model Separation Setup block to the model to run model separation.

The overall model is analyzed and all top-level subsystems that can be separated as models are displayed in the Model Separation Setup.

2 Click ⬚ to add a new model entry in the Models tree.

In the Model name edit field, enter a name for the model to be separated.

3 In the Model folder edit field, enter a folder name or path for the model to be saved in.

4 From the list of unassigned subsystems, select a subsystem and click ➡ to assign the subsystem to the model entry.

5 Repeat the above steps for all models that you want to separate from the overall model.

Result

You have defined the models to be separated from the overall model.

Next steps

After you define the models to be separated, you can create the models and generate a model communication description file (MCD file). Refer to How to Separate Models from an Overall Model on page 109.

Related topics

Basics

Creating Models and Generating a Model Communication Description File

Introduction

The **Model Separation Setup** provides the **Create Models** command to separate subsystems into individual models from an overall behavior model. You need to know the steps that are performed if you use the command.

> **Note**
>
> Before separating the overall model, it is checked whether the specified subsystems and their signals comply with the preconditions for model separation. For each block or signal that does not comply, a message with a link to the related block or signal source is issued in the MATLAB Command Window. This can help you modify the model separation configuration.

Creating models

After defining the models, you can select the models to be separated in the **Model Separation Setup**. This setup provides the **Create Models** command, which performs the following actions:

Creating the selected models Creating the selected models comprises the following steps:

- The selected models are created and the assigned top-level subsystems are copied to them. When copying the top-level subsystems to the separated models, the **Paste and Keep IDs** command is used for the subsystems' model port blocks to keep the model port blocks' identity. The models are saved as MDL or SLX files. Existing MDL or SLX files with the same name are overwritten.
- For each model interface, a **Data Inport** block or **Data Outport** block from the Model Interface Blockset is created. The model port blocks are connected to the subsystems' inports and outports according to the following rules.

  In the separated model, an outport of a top-level subsystem is connected to a **Data Outport** block if one of the following preconditions is fulfilled in the overall model:

  - The outport is connected to the inport of a block that is not assigned to the same separated model.
  - The outport is connected to a signal line. At least one branch of the signal line is not connected to another port.
  - The outport is not connected to a signal line.

In the separated model, an inport of a top-level subsystem is connected to a **Data Inport** block if one of the following preconditions is fulfilled in the overall model:

- The inport is connected to the outport of a block that is not assigned to the same separated model.
- The inport is connected to a signal line that is not connected to an outport as a signal source.
- The inport is not connected to a signal line.

> **Note**
>
> If you perform model separation for an overall model multiple times, the IDs of the model port blocks are preserved.

- Direct connections between two top-level subsystems that are assigned to the same model entry, are adopted by the separated models without being changed.
- A copy of the overall model's configuration set is assigned to each separated model.
- Model separation considers signal lines in the overall model. For example, when two subsystems are fed by the same signal and copied into one separated model, model separation inserts one **Data Inport** block and connects it to both subsystems.

Generating a model communication description file If you separate two or more models from an overall model, a model communication description file (MCD file) is generated in the folder the overall model resides in. The MCD file contains a list of the separated models and a list of the **Data Inport** and **Data Outport** blocks created in the models including their model communication. The MCD file has the name of the overall model. Only direct connections between top-level subsystems and From/Goto blocks that are assigned to different models are taken into account.

For detailed instructions, refer to How to Separate Models from an Overall Model on page 109.

> **Note**
>
> The separated models and the MCD file represent a snapshot of the overall model, and do not reflect changes of the overall model. If you performed one or more of the following changes, you must use the **Create Models** command again to make them available in ConfigurationDesk:
>
> - Functional changes in a part of the overall model which is assigned to a separated model.
> - Changes to the overall model's configuration set.
> - Changes to the definitions of the models entries (name, folder, or assigned top-level subsystems) specified in the **Model Separation Setup**.
> - Changes to the connections at Inports and Outports of top-level subsystems that are assigned to model entries in the **Model Separation Setup**.

Related topics

HowTos

How to Separate Models from an Overall Model.. 109

How to Separate Models from an Overall Model

Objective

After you define the models to be separated, you can separate them and generate a model communication description file (MCD file).

Preconditions

You must have defined the models to be separated. Refer to How to Define Models to be Separated from an Overall Model on page 106.

Method

To separate models from an overall model

1. In the Models tree of the Model Separation Setup tool, select the checkboxes of the models that you want to be created.
2. Click Create Models.

Result

The selected models containing the assigned top-level subsystems are created and saved as MDL or SLX files. An MCD file is generated and saved in the folder of the overall model. For details, refer to Creating Models and Generating a Model Communication Description File on page 107.

Next steps

You can import the MCD file into ConfigurationDesk to add the separated models to a ConfigurationDesk application. Refer to Adding the Separated Models to a ConfigurationDesk Application on page 110.

Related topics

Basics

Creating Models and Generating a Model Communication Description File.............................. 107

References

Model Separation Setup (Model Interface Package for Simulink Reference 📖)

Adding the Separated Models to a ConfigurationDesk Application

Introduction

You can import the model communication description file (MCD file) into ConfigurationDesk to add the separated models to a ConfigurationDesk application.

Importing the MCD file

ConfigurationDesk provides a command to import the generated MCD file to add it to your active ConfigurationDesk application. When you import the MCD file into your ConfigurationDesk application, the separated models are displayed in the **Model Browser**. The model port blocks that are involved in model communication including their connections are displayed in the working view. A reference to the MCD file is added to the model topology of the active ConfigurationDesk application. Refer to How to Import a Model Topology (ConfigurationDesk Real-Time Implementation Guide 🕮).

Reloading the MCD file

If you have updated your MCD file, you have to reload it to make the new model communication description available in ConfigurationDesk. Refer to Working With MCD Files (ConfigurationDesk Real-Time Implementation Guide 🕮).

Clearing the reference to the MCD file

ConfigurationDesk provides a command for clearing the reference to the MCD file. Once the reference to an MCD file is cleared, you can work with your multimodel application independently of the model topology in MATLAB/Simulink. For more information, refer to Working With MCD Files (ConfigurationDesk Real-Time Implementation Guide 🕮).

Related topics

Basics

Working With MCD Files (ConfigurationDesk Real-Time Implementation Guide 🕮)

References

Clear Model Communication Description File (ConfigurationDesk User Interface Reference 🕮)
Reload Model Communication Description File (ConfigurationDesk User Interface Reference 🕮)

Separation of Models Containing Goto and From Block Connections

Introduction

The **Model Separation Setup** tool lets you separate top-level subsystems that are connected via **Goto** and **From** blocks as models from an overall model.

Creating model port blocks for Goto and From blocks

If you perform model separation for top-level subsystems that include Goto or From blocks, the Model Separation Setup block creates new blocks on the top level of the separated models. The following blocks are created during model separation:

- For each Goto block in a subsystem that must be separated, a Data Outport block and a suitable From block are created on the top level of the separated model. The From block is automatically connected to the Data Outport block.
- For each From block in a subsystem, a Data Inport block and a suitable Goto block are created on the top level of the separated model. The Goto block is automatically connected to the Data Inport block.

In this way, the signals from the Goto and From blocks in the subsystems' hierarchy are passed to the model port blocks on the top level of the separated model.

Note

For creating model port blocks, the following preconditions must be fulfilled:

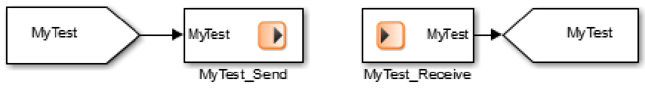
- The preconditions described in Preparing the Overall Model for Model Separation on page 103.
- The Initialize Model checkbox must be selected.
- The Tag Visibility of the Goto blocks must be set to global.

Characteristics of the created model port blocks

The model port blocks on the top level of the separated models have the following characteristics:

Property	Value
Block name	▪ Data Outport blocks: <gotoTag>_Send ▪ Data Inport blocks: <gotoTag>_Receive
Port name	<gotoTag>
Port data type	▪ Specified by the CompiledDataType property of the outport of the Goto block or the inport of the From block during a successful model initialization ▪ Limited to the supported data types: double, single, int8, int16, int32, int64, uint8, uint16, uint32, uint64, boolean, and bus signals whose signals have the supported data types
Port width	Limited to scalar and vectorial port dimensions

Example The illustration below shows examples for blocks created on the top level of the separated models for Goto and From blocks:



Preconditions for model port block creation

Certain preconditions must be fulfilled for the creation of model port blocks.

Preconditions for creating Data Outport blocks If one of the following preconditions is fulfilled, a Data Outport block is created for a Goto block in the separated model:

- In the overall model, there is no matching From block.
- In the overall model, there is at least one matching From block in a subsystem that is not assigned to the separated model containing the Goto block.
- There is at least one matching From block on the top level of the overall model.

Note

The Tag Visibility property of Goto blocks must be set to `global`.

Preconditions for creating Data Inport blocks If one of the following preconditions is fulfilled, a Data Inport block is created for a From block in the separated model:

- In the overall model, there is no matching Goto block.
- In the overall model, there is at least one matching Goto block in a subsystem that is not assigned to the separated model containing the From block.
- There is at least one matching Goto block on the top level of the overall model.

Note

- If a Goto block is connected to multiple From blocks that are assigned to the same model to be separated, only one Data Inport block is created in the separated model. The Data Inport block is then connected to all the related From blocks.
- If your model contains blocks from the MotionDesk Blockset, it might be necessary to initialize the model before you start model separation. This enables these blocks to establish their internal connections, and avoids the creation of unnecessary model port blocks.

MCD file entries

During model separation, a connection entry is created in the MCD file for each Goto and From block connection between subsystems that are assigned to different models to be separated.

Related topics

Basics

Basics on Separating Models from an Overall Model.. 99
Preparing the Overall Model for Model Separation.. 103

Separating Models That Contain CAN or LIN Bus Communication

Introduction

If your overall model contains blocks from the RTI CAN MultiMessage Blockset or the RTI LIN MultiMessage Blockset, you must note the following points when performing model separation.

Modeling instructions

Suppose you want to separate models containing blocks from the RTI CAN MultiMessage Blockset or the RTI LIN MultiMessage Blockset from the overall model. In this case, all blocks from the RTI CAN MultiMessage Blockset or the RTI LIN MultiMessage Blockset must be assigned to the same model. You can mix the blocks from the RTI CAN MultiMessage Blockset and the RTI LIN MultiMessage Blockset in one separated model.

Limitation concerning the model folder

If you assign top-level subsystems containing blocks from the RTI CAN MultiMessage Blockset or the RTI LIN MultiMessage Blockset to a model, you must leave the Model folder edit field in the Model Separation Setup empty for that model. Otherwise, model separation is aborted with an error message.

Related topics

Basics

Basics on Separating Models from an Overall Model..99
Preparing the Overall Model for Model Separation..103

Adding Custom Functionality to Model Separation

Pre-model separation and post-model separation hooks

You can use hook functions to add custom functionality to the model separation process. The Model Separation Setup tool ensures that hook functions that are on the MATLAB path and that comply with the following naming conventions are called at an appropriate time:

`*_dsmsbhook.m` with the following syntax:

```
[errCode, errMsg] = *_dsmsbhook(method, varargin)
```

The hook function must provide the following methods:

- PreSeparationCheck
- PreSeparation
- PostSeparation

PreSeparationCheck

The `PreSeparationCheck` hook is called once before model separation. It has the following syntax:

```
[errCode, errMsg] = *_dsmsbhook(‘PreSeparationCheck‘, modelHdl,
modelCfg)
```

The `PreSeparationCheck` hook has the following parameters:

errCode Numeric error code:

0: No error occurred.

1: An error occurred.

Note

If errCode != 0, model separation does not start.

errMsg Error message describing the error that occurred.

Empty string: No error.

modelHdl Handle of the overall model

modelCfg The `modelCfg` parameter is a 1 x n struct with the data fields described in the following table. n is the number of models to be separated from the overall model.

Data Field Name	Data Type	Description
Name	String	Name of the model to be created
Modelfolder	String	Name of the folder the created model is saved in
AssignedSubsystems	Cell array of strings	List of the names of the assigned subsystems
Create	Boolean	0: The model is not created. 1: The model is created.

PreSeparation

The `PreSeparation` hook is called for each model immediately after it is created, and before the subsystems and model port blocks are added. It has the following syntax:

```
[errCode, errMsg] = *_dsmsbhook(‘PreSeparation‘, modelHdl,
modelIdx, modelCfg)
```

The `PreSeparation` hook has the following parameters:

errCode Numeric error code:

0: No error occurred.

1: An error occurred.

errMsg Error message describing the error that occurred.

Empty string: No error.

modelHdl The `modelHdl` parameter specifies the handle of the overall model.

modelIdx The `modelIdx` parameter specifies the index of the model to be created. It can be used to access the `modelCfg` data fields. Refer to the following table.

modelCfg The `modelCfg` parameter is a struct with the data fields described in the following table.

Data Field Name	Data Type	Description
Name	String	Name of the model to be created
Modelfolder	String	Name of the folder the created model is saved in
AssignedSubsystems	Cell array of strings	List of the names of the assigned subsystems
Create	Boolean	0: The model is not created. 1: The model is created.

PostSeparation

The `PostSeparation` hook is called after model separation and before the model is saved. It has the following syntax:

```
[errCode, errMsg] = *_dsmsbhook(‘PostSeparation‘, modelHdl,
modelIdx, modelCfg)
```

The `PostSeparation` hook has the following parameters:

errCode Numeric error code:

0: No error occurred.

1: An error occurred.

errMsg Error message describing the error that occurred.

Empty string: No error.

modelHdl The `modelHdl` parameter specifies the handle of the overall model.

modelIdx The `modelIdx` parameter specifies the index of the model to be created. It can be used to access the `modelCfg` data fields. Refer to the following table.

modelCfg The `modelCfg` parameter is a struct with the data fields described in the following table.

Data Field Name	Data Type	Description
Name	String	Name of the model to be created.
Modelfolder	String	Name of the folder the created model is saved in.
AssignedSubsystems	Cell array of strings	List of the names of the assigned subsystems.
Create	Boolean	0: The model is not created. 1: The model is created.

Related topics

Basics

Basics on Separating Models from an Overall Model 99

Modeling Communication Interfaces in Your Model

Kinds of communication

CAN and LIN bus communication You can implement CAN and LIN bus communication for your dSPACE platform in Simulink. dSPACE provides RTI blocksets for the implementation.

FlexRay bus communication You can implement FlexRay bus communication for your dSPACE platform in Simulink. dSPACE provides the dSPACE FlexRay Configuration Package for the implementation.

Note

- If your behavior model contains CAN, LIN, or FlexRay communication, you cannot generate a Simulink implementation container for it. However, you can use such models in ConfigurationDesk by adding them to a ConfigurationDesk application.
- VEOS Player does not support Simulink implementation containers generated for models that contain blocks from the RTI CAN MultiMessage Blockset or the RTI LIN MultiMessage Blockset.
- For ConfigurationDesk, the following applies:

  As an alternative to the RTI CAN MultiMessage Blockset and the RTI LIN MultiMessage Blockset, you can use the dSPACE Bus Manager to model CAN or LIN communication. Refer to ConfigurationDesk Bus Manager Implementation Guide 📖 .

Where to go from here

Information in this section

Information in other sections

Adding Bus and Gigalink Communication to the Signal Chain (ConfigurationDesk Real-Time Implementation Guide 📖)

Modeling a CAN Bus Interface

Introduction

You can model a CAN bus interface in Simulink by using the RTI CAN MultiMessage Blockset. The following topics provide information on using the RTI CAN MultiMessage Blockset.

Note

VEOS Player does not support Simulink implementation containers generated for models that contain blocks from the RTI CAN MultiMessage Blockset.

Where to go from here

Information in this section

General Information on Modeling CAN Communication

Introduction

The SCALEXIO system can simulate a CAN bus. You can connect your ECU to the simulated CAN bus to perform restbus simulation for communication testing purposes, and to simulate CAN message errors and errors on the physical layer of the CAN bus.

An ECU is connected to a CAN bus simulated by the SCALEXIO system in several steps. One of them is to model the CAN communication in Simulink. For information on the whole workflow, refer to CAN Bus Connection (SCALEXIO – Hardware and Software Overview 🕮).

Modeling CAN communication in Simulink

Several steps have to be performed to make communication via the CAN bus available. One of them is to model the CAN communication in Simulink using blocks from the RTI CAN MultiMessage Blockset. The RTI CAN MultiMessage Blockset lets you configure and handle a large number of CAN messages. All the incoming and outgoing messages of an entire CAN controller can be controlled by a single Simulink block. For detailed information on the RTI CAN MultiMessage Blockset, refer to the RTI CAN MultiMessage Blockset Reference 🕮.

If you model the CAN communication for a SCALEXIO system and want to perform model separation later on in ConfigurationDesk to execute the separated models on single cores of the SCALEXIO system, you need to know some specifics. Refer to Separating Models That Contain CAN or LIN Bus Communication on page 113.

After CAN communication is modeled in Simulink, the Simulink model must be analyzed in ConfigurationDesk. During model analysis, one Configuration Port block is generated in ConfigurationDesk for every identified **RTICANMM ControllerSetup** block contained in the Simulink model. For further information, refer to Building the Signal Chain for CAN Bus Communication (ConfigurationDesk Real-Time Implementation Guide 🕮).

Demo model

ConfigurationDesk comes with a CANMM demo giving you an example of how to implement CAN communication using the RTICANMM blocks in the behavior model. The demo is available in the project root folder after you first start ConfigurationDesk.

> **Note**
>
> To use the CANMM demo model, you must create the S-functions for all the RTICANMM blocks it contains and save it before analyzing it in ConfigurationDesk and building a real-time application. Use the **Create S-Function for all CAN Blocks** command from the **Options** menu of the **RTICANMM GeneralSetup** block for this. For further information, refer to Options Menu (RTICANMM GeneralSetup) (RTI CAN MultiMessage Blockset Reference 🕮).

Basics on the RTI CAN MultiMessage Blockset

Introduction

The RTI CAN MultiMessage Blockset is a Simulink blockset for efficient and dynamic handling of complex CAN setups in hardware-in-the-loop (HIL) applications. All the incoming RX messages and outgoing TX messages of an entire CAN controller can be controlled by a single Simulink block. CAN communication is configured via database files (DBC file format, FIBEX file format, MAT file format, or AUTOSAR XML file format).

Supported dSPACE platforms

The RTI CAN MultiMessage Blockset is supported by the following dSPACE platforms:

- SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board
- MicroAutoBox II
- MicroLabBox
- PHS-bus-based systems (DS1006 or DS1007 modular systems) containing one of the following I/O boards:
  - DS2202 HIL I/O Board
  - DS2210 HIL I/O Board
  - DS2211 HIL I/O Board
  - DS4302 CAN Interface Board
  - DS4505 Interface Board, if the DS4505 is equipped with DS4342 CAN FD Interface Modules

The dSPACE platforms provide 1 ... 4 CAN controllers (exception: DS6342 provides 1 ... 8 CAN controllers). A CAN controller performs serial communication according to the CAN protocol. To use a dSPACE board with CAN bus interface, you must configure the CAN controller – called the CAN channel – according to the application.

> **Note**
>
> The RTI CAN MultiMessage Blockset is not supported by the MicroAutoBox III. If you work with the MicroAutoBox III, you must use the Bus Manager to implement CAN and CAN FD bus simulation.

Managing large CAN message bundles

With the RTI CAN MultiMessage Blockset, you can configure and control a large number of CAN messages (more than 200) from a single Simulink block.

Support of CAN FD protocol

The RTI CAN MultiMessage Blockset supports the classic CAN protocol, the non-ISO CAN FD protocol (which is the original CAN FD protocol from Bosch) and the ISO CAN FD protocol (according to the ISO 11898-1:2015 standard). The CAN FD protocols allow data rates higher than 1 Mbit/s and payloads of up to 64 bytes per message.

Keep in mind that the CAN FD protocols are supported only by dSPACE platforms equipped with a CAN FD-capable CAN controller.

For more information, refer to Basics on Working with CAN FD on page 124.

Support of AUTOSAR features

The RTI CAN MultiMessage Blockset supports miscellaneous AUTOSAR features, such as secure onboard communication and global time synchronization. For more information, refer to Aspects of Miscellaneous Supported AUTOSAR Features (RTI CAN MultiMessage Blockset Reference 🕮).

Manipulating signals to be transmitted

The RTI CAN MultiMessage Blockset provides several options to manipulate the values of signals before they are transmitted. You can switch between them with the **Bus Navigator** in ControlDesk via entries in the generated TRC file. In addition, you can calculate checksum, parity, toggle, counter, and mode counter values.

Updating a model

The RTI CAN MultiMessage Blockset creates an S-function for the specified database file. You can easily update the CAN configuration of a model by replacing the database file and updating the S-function.

> **Tip**
>
> You do not have to recreate the S-function for the RTI CAN MultiMessage Blockset if you switch from one processor board to another, e.g., from a DS1006 to a DS1007, and vice versa.
> When you switch to or from a MicroAutoBox II or MicroLabBox, you only have to recreate the ControllerSetup block. You also have to recreate the ControllerSetup block if you change the controller name.

Modifying model parameters during run time

Model parameters such as messages or signal values can be modified during run time either via model input or via the **Bus Navigator** in ControlDesk. For modifying model parameters via ControlDesk, a variable description file (TRC) is automatically generated each time you create an S-function for the RTICANMM MainBlock. The entries of the TRC file let you analyze received signals, change the values of signals to be transmitted, etc. In ControlDesk, you can access the settings specified in the TRC file via the model's system description file (SDF). The SDF file bundles all the TRC files and additional information for the application.

(Relevant only for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board) For information on where to find the signals of the CAN bus in the TRC file, refer to Available TRC File Variable Entries and Their Locations in the TRC File on page 235.

User-defined variables

You can include user-defined variables in a TRC file in addition to the parameters of the RTI CAN MultiMessage Blockset.

Working with variants of CAN communication

You can work with different CAN communication variants on one CAN controller, and switch between them during run time. Only one variant for each CAN controller can be active at a time. In the **Bus Navigator**, the active variant is labeled when an application is running on the real-time hardware. An inactive variant is labeled . If you open layouts of an inactive variant, the headers of all the RX and TX layouts are red.

Gatewaying messages between CAN buses

You can easily exchange messages between different CAN buses. In addition, you can use the gatewaying feature to modify messages in gateway mode during run time.

Online modification of gateway exclude list

You can modify the exclude list of RTICANMM Gateways during run time, i.e., specify messages not to be gatewayed.

Dynamic message triggering

You can modify the cycle times and initiate a spontaneous transmission (interactive or model-based) during run time.

Defining free raw messages

You can define free raw messages as additional messages that are independent of the database file. Once they are defined, you can use them like standard database messages and specify various options, for example:

- Trigger options
- Ports and displays
- Message ID and length adjustable during run time

The following features are not supported:

- Checksum generation
- Custom signal manipulation

Capturing messages

You can process the raw data of messages whose IDs match a given filter via the capture messages feature. This can be a specific ID, a range of IDs, or even all IDs. Optionally, you can exclude the messages contained in the database file.

The captured messages can be made available as outports of the MainBlock or in the TRC file.

CAN partial networking

With CAN partial networking, you can set selected ECUs in a network to sleep mode or shut them down if they do not have to run continuously. Wake-up messages then activate specific ECUs as and when required, and for as long as required.

Note

Partial networking is possible for the following dSPACE real-time hardware:

- MicroAutoBox II equipped with the DS1513 I/O Board
- MicroLabBox
- dSPACE hardware that supports working with CAN FD messages and that is equipped with DS4342 CAN FD Interface Modules, such as:
  - PHS-bus-based systems (DS1006 or DS1007 modular systems) with DS4505 Interface Board
  - MicroAutoBox II variants with DS1507
  - MicroAutoBox II variants with DS1514

The RTI CAN MultiMessage Blockset lets you specify the CAN partial networking wake-up messages by filtering message IDs and message data.

The CAN transceiver of the dSPACE real-time hardware is switched to sleep mode via the real-time application. You can use partial networking messages to wake up dSPACE real-time hardware after its CAN transceiver is switched to sleep mode.

(Relevant for MicroAutoBox II only) If a MicroAutoBox II's CAN transceiver is woken up via a partial networking message, the MicroAutoBox II behaves as if it was powered up manually. Depending on where the real-time application is loaded (flash memory or RAM), the MicroAutoBox II starts the application or waits for further input.

For more information, refer to Partial Networking Page (RTICANMM ControllerSetup) (RTI CAN MultiMessage Blockset Reference 🕮).

TRC file entries with initial data

TRC/SDF files generated for Simulink models including blocks from the RTI CAN MultiMessage Blockset contain initial data. The RTI CAN MultiMessage Blockset supplies all variables with initial values when they are included in the TRC file. TRC files with initial data lets you to perform offline calibration with ControlDesk.

Visualization with the Bus Navigator

The RTI CAN MultiMessage Blockset supports visualization with the Bus Navigator. Layouts/instruments are generated on demand and provide access to all CAN signals and all switches required to configure CAN communication during run time. You do not have to preconfigure layouts by hand.

RTI CAN Blockset and RTI CAN MultiMessage Blockset

(Not relevant for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board) You can use the RTI CAN Blockset and the RTI CAN MultiMessage Blockset in parallel for different CAN controllers. However, you cannot use the RTI CAN MultiMessage Blockset and the RTI CAN Blockset for the same CAN controller.

Further information on the RTI CAN MultiMessage Blockset

The following documents provide further information on the RTI CAN MultiMessage Blockset:

- RTI CAN MultiMessage Blockset Reference 📖

  This RTI reference provides a full description of the RTI CAN MultiMessage Blockset.
- RTI CAN MultiMessage Blockset Tutorial 📖

  This tutorial guides you through your first steps with the RTI CAN MultiMessage Blockset.

Basics on Working with CAN FD

Introduction

Using the CAN FD protocol allows data rates higher than 1 Mbit/s and payloads longer than 8 bytes per message.

Basics on CAN FD

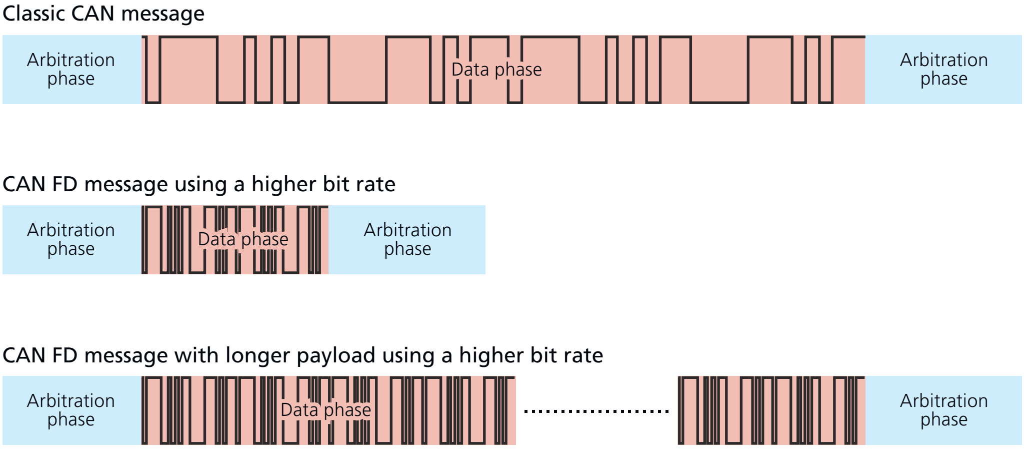
CAN FD stands for *CAN with Flexible Data Rate*. The CAN FD protocol is based on the CAN protocol as specified in ISO 11898-1. Compared with the classic CAN protocol, CAN FD comes with an increased bandwidth for the serial communication. The improvement is based on two factors:

- The CAN FD protocol allows you to use CAN messages with longer data fields (up to 64 bytes).
- The CAN FD protocol allows you to use a higher bit rate (typically higher by a factor of 8). It is possible to switch inside the message to the faster bit rate.

Arbitration phase and data phase CAN FD messages consist of two phases: a data phase and an arbitration phase. The data phase spans the phase where the data bits, CRC and length information are transferred. The rest of the frame, outside the data phase, is the arbitration phase. The data phase can be configured to have a higher bit rate than the arbitration phase.

CAN FD still uses the CAN bus arbitration method. During the arbitration process, the standard data rate is used. After CAN bus arbitration is decided, the data rate can be increased. The data bits are transferred with the preconfigured higher bit rate. At the end of the data phase, CAN FD returns to the standard data rate.

The following illustration shows a classic CAN message, a CAN FD message using a higher bit rate during the data phase, and a CAN FD message with longer payload using a higher bit rate. You can see the implications of the CAN FD features: The arbitration phases are identical in all cases, because the standard bit rate is always used. The lengths of the data phases differ depending on the payload length and bit rate used.



CAN FD protocols

Currently, there are two CAN FD protocols on the market, which are not compatible with each other.

- The *non-ISO CAN FD protocol* represents the original CAN FD protocol from Bosch.
- The *ISO CAN FD protocol* represents the CAN FD protocol according to the ISO 11898-1:2015 standard.

Compared to the non-ISO CAN FD protocol, the ISO CAN FD protocol comes with an improved failure detection capability.

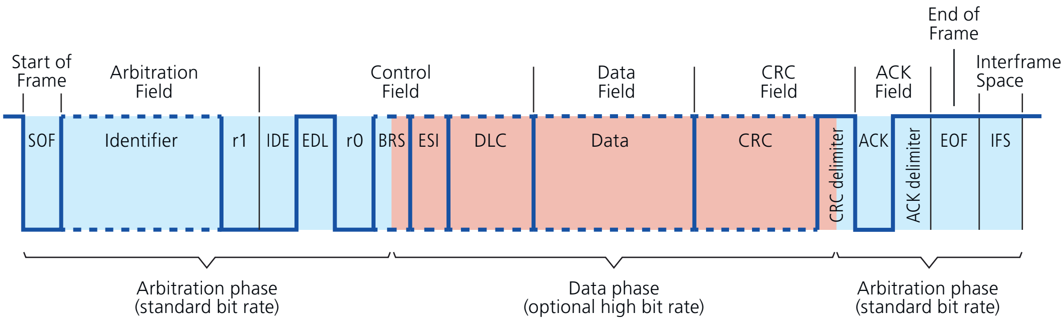
The RTI CAN MultiMessage Blockset supports both CAN FD protocols.

CAN FD message characteristics

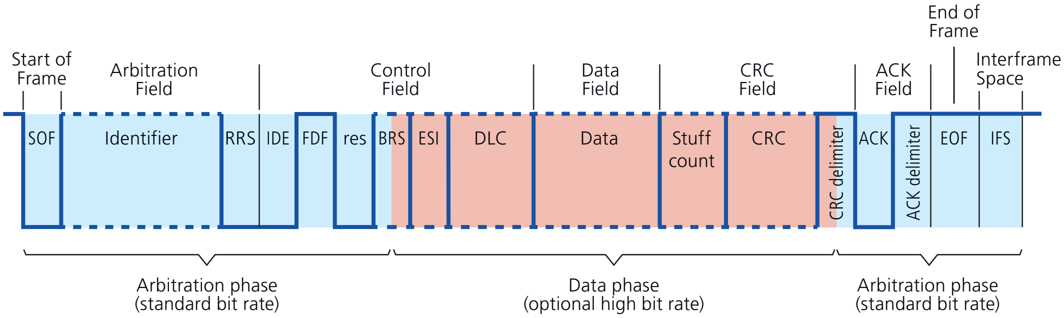
In principle, CAN FD messages and CAN messages consist of the same elements and have the same message structure.

For an overview of the fields of a CAN FD message and the message components for each of the two CAN FD protocols, refer to the following illustration:

- Non-ISO CAN FD protocol:



- ISO CAN FD protocol:



There are some differences between CAN FD messages and CAN messages:

- The Data field and the CRC field of CAN FD messages can be longer. The maximum payload length of a CAN FD message is 64 bytes.
- The Control fields are different:
  - A classic CAN message always has a dominant (= 0) reserved bit immediately before the data length code.

    In a CAN FD message, this bit is always transmitted with a recessive level (= 1) and is called *EDL* (Extended Data Length) or *FDF* (FD Format), depending on the CAN FD protocol you are working with. So CAN messages and CAN FD messages are always distinguishable by looking at the EDL/FDF bit. A recessive EDL/FDF bit indicates a CAN FD message, a dominant EDL/FDF bit indicates a CAN message.

    In CAN FD messages, the EDL/FDF bit is always followed by a dominant reserved bit (*r0*/*res*), which is reserved for future use.
  - A CAN FD message has the additional *BRS* (Bit Rate Switching) bit, which allows you to switch the bit rate for the data phase. A recessive BRS bit switches from the standard bit rate to the preconfigured alternate bit rate. A dominant BRS bit means that the bit rate is not switched and the standard bit rate is used.
  - A CAN FD message has the additional *ESI* (Error State Indicator) bit. The ESI bit is used to identify the error status of a CAN FD node. A recessive ESI bit indicates a transmitting node in the 'error active' state. A dominant ESI bit indicates a transmitting node in the 'error passive' state.
- Since CAN FD messages can contain up to 64 data bytes, the coding of the DLC (data length code) has been expanded. The following table shows the possible data field lengths of CAN FD messages and the corresponding DLC values.

DLC	Number of Data Bytes
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6

DLC	Number of Data Bytes
0111	7
1000	8
1001	12
1010	16
1011	20
1100	24
1101	32
1110	48
1111	64

If necessary, padding bytes are used to fill the data field of a CAN FD message to the next greater possible DLC value.

For classic CAN messages, the DLC values 1000 ... 1111 are interpreted as 8 data bytes.

- (Valid for the ISO CAN FD protocol only) The CRC fields are different:
  - The CRC field of a CAN FD message was extended by a stuff count before the actual CRC sequence. The stuff count consists of a 3-bit stuff bit counter (reflects the number of the data-dependent dynamic stuff bits (modulo 8)), and an additional parity bit to secure the counter.
  - The start value for the CRC calculation was changed from '0...0' to '10...0'.

Activating CAN FD mode in the RTI CAN MultiMessage Blockset

To ensure that CAN FD messages are properly transmitted and received during run time, the CAN FD mode must be enabled at two places in the RTI CAN MultiMessage Blockset: in the MainBlock and at the CAN controller selected in the MainBlock. To do so, perform the following steps:

- In the ControllerSetup block, select the CAN FD protocol to be used. Refer to Setup Page (RTICANMM ControllerSetup) (RTI CAN MultiMessage Blockset Reference 🕮).
- In the MainBlock, select the **CAN FD support** checkbox. Refer to General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

To monitor CAN FD messages, it is sufficient to enable CAN FD support in the ControllerSetup block.

Supported database file types

To work with CAN FD messages, you need a suitable database file containing descriptions in the CAN FD format. The RTI CAN MultiMessage Blockset supports CAN FD for the following database file types:

- DBC file
- AUTOSAR system description file
- FIBEX file (FIBEX 4.1.2 only)

CanFrameTxBehavior and CanFrameRxBehavior attributes In AUTOSAR and FIBEX files, the `CanFrameTxBehavior` and/or `CanFrameRxBehavior` attributes of a message can be defined to specify whether the message is to be treated as a CAN FD message or classic CAN 2.0 message. The RTI CAN MultiMessage Blockset evaluates the attribute as follows:

- If the `CanFrameTxBehavior` attribute is defined for a message in the database file, RTICANMM uses this setting for the message on the CAN bus for both directions, i.e., for sending and receiving the message.
- If the `CanFrameTxBehavior` attribute is not defined in the database for a message, RTICANMM uses the `CanFrameRxBehavior` setting of the message for sending and receiving the message.

Supported dSPACE platforms

The RTI CAN MultiMessage Blockset supports working with CAN FD messages for the following dSPACE hardware:

- SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, or DS6342 CAN Board
- The following dSPACE platforms if they are equipped with DS4342 CAN FD Interface Modules:
  - DS1006 modular system with DS4505 Interface Board
  - DS1007 modular system with DS4505 Interface Board
  - MicroAutoBox II in the following variants:
    - 1401/1507
    - 1401/1511/1514
    - 1401/1513/1514

When you connect a DS4342 CAN FD Module to a CAN bus, you must note some specific aspects (such as bus termination and using feed-through bus lines). For more information, refer to:

- PHS-bus-based system with DS4505: DS4342 Connections in Different Topologies (PHS Bus System Hardware Reference 🕮)
- MicroAutoBox II: DS4342 Connections in Different Topologies (MicroAutoBox II Hardware Installation and Configuration Guide 🕮)

Working with CAN messages and CAN FD messages

Both messages in CAN format and in CAN FD format can be transmitted and received via the same network.

Related topics

References

Setup Page (RTICANMM ControllerSetup) (RTI CAN MultiMessage Blockset Reference 🕮)

Basics on Working with a J1939-Compliant DBC File

Introduction

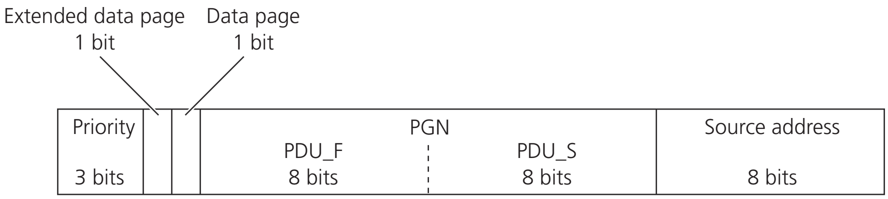
J1939 is a vehicle CAN bus standard defined by the Society of Automotive Engineers (SAE). It is used for communication in heavy-duty vehicles, for example, for communication between a tractor and its trailer.

The RTI CAN MultiMessage Blockset supports you in working with J1939-compliant DBC files.

Broadcast and peer-to-peer communication

CAN message identifier for J1939 Standard CAN messages use an 11-bit message identifier (CAN 2.0 A). J1939 messages use an extended 29-bit message identifier (CAN 2.0 B).

The 29-bit message identifier is split into different parts (according to SAE J1939/21 Data Link Layer):



29-bit CAN message ID

- The 3-bit *priority* is used during the arbitration process. A value of 0 represents the highest priority, a value of 7 represents the lowest priority.
- The 1-bit *extended data page* can be used as an extension of the PGN. The RTI CAN MultiMessage Blockset lets you specify whether to use the extended data page bit this way. Refer to Code Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖).
- The 1-bit *data page* is a selector for the PDU_F in the PGN. It can be taken as an extension of the PGN. The RTI CAN MultiMessage Blockset uses the data page bit in this way.
- The 16-bit *PGN* is the parameter group number. It is described in this section.
- The 8-bit *source address* is the address of the transmitting network node.

Parameter group number (PGN) A 16-bit number in the 29-bit message identifier of a CAN message defined in a J1939-compliant DBC file. Each PGN references a *parameter group* that groups parameters and assigns them to the 8-byte data field of the message. A parameter group can be the engine temperature including the engine coolant temperature, the fuel temperature, etc. PGNs and parameter groups are defined by the SAE (see SAE J1939/71 Vehicle Application Layer).

The first 8 bits of the PGN represent the *PDU_F* (Protocol Data Unit format). The PDU_F value specifies the communication mode of the message (peer-to-peer or broadcast). The interpretation of the *PDU_S* value (PDU-specific) depends on the PDU_F value. For messages with a PDU_F < 240 (peer-to-peer communication, also called PDU1 messages), PDU_S is not relevant for the PGN, but contains the destination address of the network node that receives the message. For messages with a PDU_F ≥ 240 (broadcast messages, also called PDU2 messages), PDU_S specifies the second 8 bits of the PGN and represents the group extension. A group extension is used to increase the number of messages that can be broadcast in the network.

PDU_F (first 8 bits)	PDU_S (second 8 bits)	Communication Mode
< 240	Destination address	Peer-to-peer (message is transmitted to one destination network node)
≥ 240	Group extension	Broadcast (message is transmitted to any network node connected to the network)

Message attributes in J1939-compliant DBC files

A message described in a J1939-compliant DBC file is described by the ID attribute and others.

DBC files created with CANalyzer 5.1 or earlier In a DBC file created with CANalyzer 5.1 or earlier, the ID attribute describing a message actually is the *message PGN*. Thus, the ID provides no information on the source and destination of the message. The source and destination can be specified by the *J1939PGSrc* and *J1939PGDest* attributes.

DBC files created with CANalyzer 5.2 or later In a DBC file created with CANalyzer 5.2 or later, the ID attribute describing a message actually is the *CAN message ID, which consists of the priority, PGN, and source address*. Thus, the ID provides information on the source and destination of the message. Further senders can be specified for a message in Vector Informatik's CANdb++ Editor (*_BO_TX_BU* attribute). When a DBC file is read in, RTICANMM automatically creates new instances of the message for its other senders.

Source/destination mapping

Messages in a J1939-compliant DBC file that are described only by the PGN have no *source/destination mapping*. Messages that are described by the CAN message ID (consisting of the priority, PGN, and source address) have source/destination mapping.

Tip

The RTI CAN MultiMessage Blockset lets you specify source/destination mapping for messages that are described only by the PGN. The mapping can be specified in the RTICANMM MainBlock or in the DBC file using the *J1939Mapping* attribute.

Container and instance messages

The RTI CAN MultiMessage Blockset distinguishes between *container* and *instance messages* when you work with a J1939-compliant DBC file:

Container message A J1939 message defined by its PGN (and Data Page bit). A container can receive all the messages with its PGN in general. If several messages are received in one sampling step, only the last received message is held in the container. Container messages can be useful, for example, when you configure a gateway with message manipulation.

Instance message A J1939 message defined by its PGN (and Data Page bit), for which the source (transmitting) network node and the destination (receiving) network node (only for peer-to-peer communication) are defined.

Note

The RTI CAN MultiMessage Blockset only imports instances for which valid source node and destination node specifications are defined in the DBC file. In contrast to instances, containers are imported regardless of whether or not valid source node and destination node specifications are known for them during the import. This lets you configure instances in the RTI CAN MultiMessage Blockset.

There is one container for each PGN (except for proprietary PGNs). If you work with DBC files created with CANalyzer 5.1 or earlier, the container can be clearly derived from the DBC file according to its name. With DBC files created with CANalyzer 5.2 or later, several messages with the same PGN might be defined. In this case, either the message with the shortest name or an additionally created message (named `CONT_<shortest_message_name>`) is used as the container for the PGN. The RTI CAN MultiMessage Blockset lets you specify the container type in the RTICANMM MainBlock. If several messages fulfill the condition of the shortest name, the one that is listed first in the DBC file is used. For messages with proprietary PGNs, each message is its own container (because proprietary PGNs can have different contents according to their source and destination nodes).

Network management

The J1939 CAN standard defines a multimaster communication system with decentralized network management. J1939 network management defines the automatic assignment of network node addresses via address claiming, and provides identification for each network node and its primary function.

Each network node must hold exactly one 64-bit name and one associated address for identification.

Address The 8-bit network node *address* defines the source or destination for J1939 messages in the network. The address of a network node must be unique. If there is an address conflict, the network nodes try to perform dynamic network node addressing (address claiming) to ensure unique addresses, if this is enabled for the network nodes.

The J1939 standard reserves the following addresses:

- Address 0xFE (254) is reserved as the 'null address' that is used as the source address by network nodes that have not yet claimed an address or have failed to claim an address.
- Address 0xFF (255) is reserved as the 'global address' and is exclusively used as a destination address in order to support message broadcasting (for example, for address claims).

The RTI CAN MultiMessage Blockset does not allow J1939 messages to be sent from the null or global addresses.

> **Note**
>
> The RTI CAN MultiMessage Blockset interprets attributes in the DBC file like this:
>
> - In a DBC file created with CANalyzer 5.1 or earlier, the *name* network node attributes and the *J1939PGSrc* and *J1939PGDest* message attributes are read in. The J1939PGSrc attribute is interpreted as the address of the node that sends the message, the J1939PGDest attribute is interpreted as the address of the node that receives the message.
> - In a DBC file created with CANalyzer 5.2 or later, the *name* and *NMStationAddress* network node attributes are read in. The NMStationAddress attribute is interpreted as the network node address.

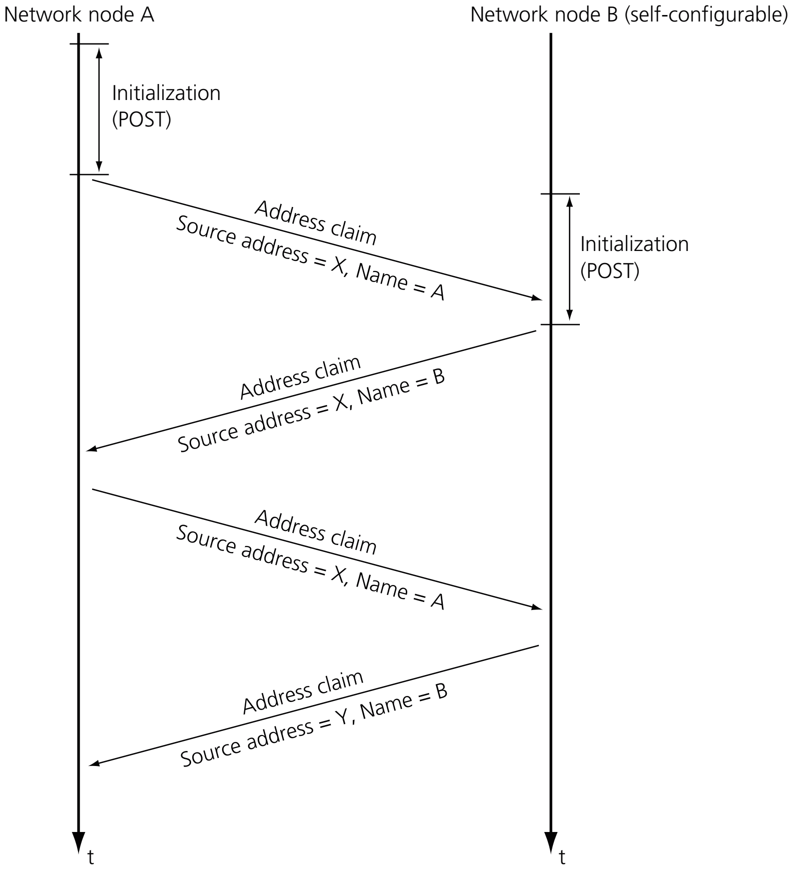
Name The J1939 standard defines a 64-bit *name* to identify each network node. The name indicates the main function of the network node with the associated address and provides information on the manufacturer.

Arbitrary Address Capable	Industry Group	Vehicle System Instance	Vehicle System	Reserved	Function	Function Instance	ECU Instance	Manufacturer Code	Identity Number
1 bit	3 bit	4 bit	7 bit	1 bit	8 bit	5 bit	3 bit	11 bit	21 bit

Address claiming The J1939 standard defines an address claiming process in which addresses are autonomously assigned to network nodes during network initialization. The process ensures that each address is unique.

Each network node sends an address claim to the CAN bus. The nodes receiving an address claim verify the claimed address. If there is an address conflict, the network node with the lowest 64-bit name value (highest priority) gets the claimed address. The other network nodes must claim different addresses.

The following illustration shows the address claiming process with two network nodes claiming the same address. Network node A has a 64-bit name of higher priority.


Network node A
Network node B (self-configurable)
Initialization
(POST)
Address claim
Source address = X, Name = A
Initialization
(POST)
Address claim
Source address = X, Name = B
Address claim
Source address = X, Name = A
Address claim
Source address = Y, Name = B
t
t

The following steps are performed in the address claiming procedure:

- Node A starts initialization and the power-on self-test (POST).
- While node B performs initialization and POST, node A sends its address claim message.
- After performing initialization and POST, node B sends its address claim message, trying to claim the same source address as node A.
- In response to the address claim message of node B, the 64-bit names of the network nodes are compared. Because the name of network node A has a higher priority, network node A succeeds and can use the claimed address. Node A sends its address claim message again.
- Network node B receives the address claim message and determines that node A's name has higher priority. Node B claims a different address by sending another address claim message.

The RTI CAN MultiMessage Blockset supports J1939 network management including address claiming for self-configurable address network nodes. This allows network nodes simulated by the RTI CAN MultiMessage Blockset to change their addresses, if necessary, and to update their internal address assignments if addresses of external network nodes are changed.

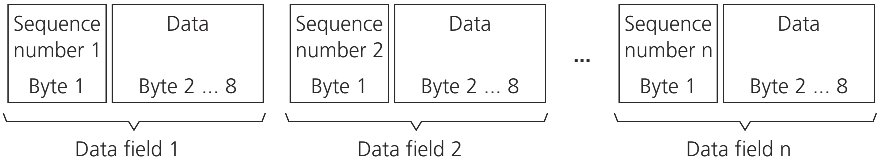
Note

The RTI CAN MultiMessage Blockset supports network management only for network nodes for which network addresses are contained in the DBC file and that have unique 64-bit name identifiers. The node configuration is taken directly from the DBC file and can be adapted on the RTI CAN MultiMessage Blockset dialog pages.

Messages > 8 bytes (message packaging)

Standard CAN messages have a data field of variable length (0 ... 8 bytes). The J1939 protocol defines transport protocol functions which allow the transfer of up to 1785 bytes in one message.

A multipacket message contains up to 255 data fields, each of which has a length of 8 bytes. Each data field is addressed by the 1-byte sequence number, and contains 7 bytes of data. This yields a maximum of 1785 (= 255 · 7) bytes in one message.



The RTI CAN MultiMessage Blockset supports J1939 message packaging via BAM and RTS/CTS:

Broadcasting multipacket messages via BAM To broadcast a multipacket message, the sending network node first sends a *Broadcast Announce Message* (BAM). It contains the PGN and size of the multipacket message, and the number of packages. The BAM allows all receiving network nodes to prepare for the reception. The sender then starts the actual data transfer.

Peer-to-peer communication of multipacket messages via RTS/CTS To transfer a multipacket message to a specific destination in the network, the sending network node sends a *request to send* (RTS). The receiving network node responds with either a *clear to send* (CTS) message or a connection abort message if the connection cannot be established. When the sending network node receives the CTS message, it starts the actual data transfer.

By default, the number of CTS packets is set to 1. To allow peer-to-peer communication for multipacket J1939 messages longer than 8 bytes via RTS/CTS, you can change the default number of CTS packets. The RTI CAN MultiMessage Blockset provides the special MATLAB preference `set_j1939_cts_packet_number`. To change the default number of CTS packets, you must type the following command in the MATLAB Command Window:

```
rtimmsu_private('fcnlib', 'set_j1939_cts_packet_number', 'can',
<n>);
```

The argument <n> describes the number of CTS packets. The value must be in the range [1, 255].

Related topics

Basics

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File (RTI CAN MultiMessage Blockset Tutorial 🕮)

Transmitting and Receiving CAN Messages

Introduction

Large CAN message bundles (> 200 messages) can be managed by a single Simulink block provided by the RTI CAN MultiMessage Blockset.

Defining CAN communication

To define the CAN communication of a CAN controller, you can specify a DBC, MAT, FIBEX, or AUTOSAR system description file as the database file on the General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮). You can also define CAN communication without using a database file.

DBC file as the database The Data Base Container (DBC) file format was developed by Vector Informatik GmbH, Stuttgart, Germany. For the RTI CAN MultiMessage Blockset, you can use all the DBC files that pass the consistency check of Vector Informatik's CANdb++ Editor.

FIBEX file as the database The Field Bus Exchange (FIBEX) format is an XML exchange file format. It is used for data exchange between different tools that work with message-oriented bus communication. A FIBEX file usually describes more than one bus system. You therefore have to select one of the available bus systems if you work with a FIBEX file as the database.

AUTOSAR system description file as the database You can use an AUTOSAR system description file as the database for CAN communication. AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership that aims to develop and establish an open standard for automotive electric/electronic (E/E) architectures.

AUTOSAR system description files are files of AUTOSAR XML file type that describe a system according to AUTOSAR. A system is a combination of a hardware topology, a software architecture, a network communication, and information on the mappings between these elements. AUTOSAR system description files are instances of the AUTOSAR System Template.

An AUTOSAR system description file usually describes more than one bus system. You therefore have to select one of the available bus systems if you work with an AUTOSAR XML file as the database.

MAT file as the database You can also use the MAT file format as the database for CAN communication, or specify other database file formats as the database. You must convert your specific database files into the MAT file format for this purpose. Because the MAT file requires a particular structure, it must be generated by M-script.

Working without a database file If you want to work without a database file, you can use free raw messages and/or capture messages. These messages are independent of DBC and MAT files.

Changing database defaults When you work with a database file, you can change its default settings via the following dialog pages of the RTICANMM MainBlock:

- Cycle Time Defaults Page (RTICANMM MainBlock)
- Base/Update Time Page (RTICANMM MainBlock)
- TX Message Length Page (RTICANMM MainBlock)
- Message Defaults Page (RTICANMM MainBlock)
- Signal Defaults Page (RTICANMM MainBlock)
- Signal Ranges Page (RTICANMM MainBlock)
- Signal Errors Page (RTICANMM MainBlock)

Defining RX messages and TX messages

You can receive and/or transmit each message defined in the database file that you specify for CAN communication.

Defining RX messages You can define RX messages on the RX Messages Page (RTICANMM MainBlock).

Defining TX messages You can define TX messages on the TX Messages Page (RTICANMM MainBlock).

Triggering TX message transmission

You can specify different options to trigger the transmission of TX messages. For example, message transmission can be triggered cyclically or by an event.

For details, refer to Triggering Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Triggering reactions to the reception of RX messages

You can specify the reactions to receiving a specific RX message. One example of a reaction is the immediate transmission of a TX message.

For details, refer to Raw Data Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Working with raw data

The RTI CAN MultiMessage Blockset lets you to work with the raw data of messages. You have to select the messages for this purpose. The RTICANMM

MainBlock then provides the raw data of these messages to the model byte-wise for further manipulation. You can easily access the raw data of RX messages via a Simulink Bus Selector block.

For details, refer to Raw Data Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖).

Implementing checksum algorithms

You can implement checksum algorithms for the checksum calculation of TX messages and checksum verification of RX messages.

Checksum header file You have to specify the checksum algorithms in a checksum header file. This needs to have a C-coded switch-case directive to switch between algorithms.

Checksum calculation for TX messages You can assign a checksum algorithm to each TX message. A checksum is calculated according to the algorithm and assigned to the TX message. Then the message is transmitted together with the calculated checksum.

Checksum check for RX messages You can assign a checksum algorithm to each RX message. A checksum is calculated for the message and compared to the checksum in the received message. If they differ, this is indicated at the error ports for RX messages if these ports are enabled.

Checksum algorithms based on end-to-end communication protection The RTI CAN MultiMessage Blockset supports run-time access to E2E protection parameters from AUTOSAR communication matrices and DBC files. This means that you can implement checksum algorithms based on end-to-end communication (E2E protection) parameters. E2E protection checksum algorithms are implemented in the same checksum header file as the checksum algorithms without E2E protection data.

For details, refer to Checksum Definition Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖).

Gatewaying messages

Gatewaying means exchanging CAN messages between two CAN buses. Gatewaying also applies to messages that are not specified in the database file. You can also exclude individual messages specified in the database file from being exchanged.

You can gateway CAN messages in two ways:

Controller gateway This is a gateway between two CAN controllers. The gateway is between two RTICANMM ControllerSetup blocks and is independent of the active CAN controller variant.

MainBlocks gateway This is a gateway between different variants of two CAN controllers. The gateway is between two RTICANMM MainBlocks. The MainBlocks gateway is active only if the variants of both CAN controllers are active at the same time.

For details, refer to RTICANMM Gateway (RTI CAN MultiMessage Blockset Reference 📖).

Related topics

References

General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖)

Manipulating Signals to be Transmitted

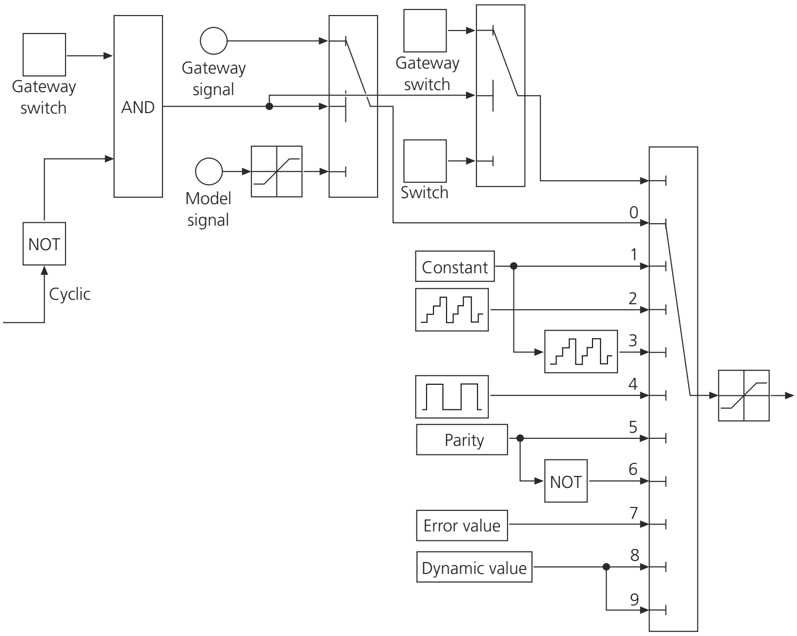
Introduction

All the signals of all the RX and TX messages (see Defining RX messages and TX messages on page 136) automatically get corresponding entries in the generated TRC file. This allows you to analyze them (signals of RX messages) or change their values (signals of TX messages) with the Bus Navigator in ControlDesk.

Manipulating signals to be transmitted

The RTI CAN MultiMessage Blockset provides several options to manipulate the values of signals before they are transmitted. You can switch between the options you have specified via entries in the generated TRC file.

The illustration below visualizes the options.



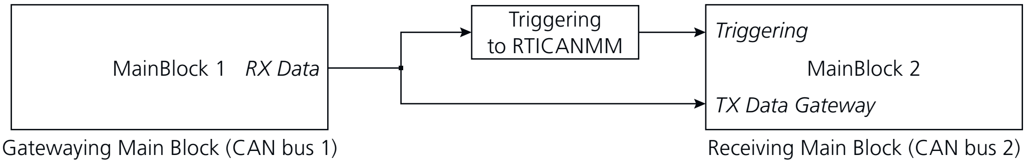
You can switch between these options in ControlDesk.

TX model signal A signal of a TX message whose value can be changed from within the model. By default, the values of TX model signals cannot be changed in ControlDesk. If you also want to manipulate TX model signals from ControlDesk, you have to select them on the Input Manipulation Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖).

Because additional code has to be generated, TX model signals reduce performance. For optimum performance, you should specify as few TX model signals as possible.

You can specify TX model signals on the Model Signals (TX) Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Gateway signal A signal to be manipulated before it is exchanged between two CAN buses. Gateway signals have to be gatewayed via two RTICANMM MainBlocks. You have to specify gateway signals for the receiving MainBlock.



MainBlock1 gateways messages and their signals to MainBlock2. Specifying gateway signals for MainBlock2 adds a **TX Data Gateway** inport to it. The specified gateway signals are transmitted via CAN bus 2 with the signal values received from MainBlock1 when triggered by the messages received from MainBlock1. You therefore have to specify triggered message transmission for the messages of the gateway signals on the Message Cyclic Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮). In addition, you can enable signal value switching for gateway signals during run time, for example, to transmit the signal constant value instead of the gateway value.

Note

Implementing gateway signals at least doubles the number of block inports and therefore reduces performance. If you want to exchange messages between CAN controllers and do not want to perform signal manipulation, you should use the RTICANMM Gateway block instead.

You can specify gateway signals on the pages located in the Gateway Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Toggle signal A 1-bit signal that can be used, for example, to indicate whether CAN messages are transmitted. If a CAN message is transmitted, the toggle signal value alternates between 0 and 1. Otherwise, the toggle signal value remains constant.

You can specify toggle signals on the Toggle Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Parity signal A signal that a parity bit is appended to. You can specify one or more signals of a TX message as parity signals. A parity bit is calculated for the specified signals according to whether even or odd parity is selected. The bit is appended to the signal and the TX message is transmitted with the parity signal.

You can specify parity signals on the Parity Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Counter signal A signal of a TX message that is used to check for correct message transmission or to trigger the transmission of signals in a message.

- Behavior of counter signals

  The value of a counter signal changes with every message transmission. You can specify the counter start, step, and stop values, etc. Each time the counter reaches the stop value, it turns around to the counter start value.

- Use of counter signals

  You can specify counter signals to check for correct transmission of a message or trigger the transmission of signals in a message.

  - *Checking correct message transmission*: The receiver of a message expects a certain counter signal value. For example, if the transmission of a message was stopped for two transmissions, the expected counter signal value and the real counter signal value can differ, which indicates an error. Counter signals used in this way are often called alive counters.
  - *Triggering the transmission of signals*: You can trigger the transmission of signals if you specify a mode signal as a counter signal. The signal value of the mode signal triggers the transmission of mode-dependent signals. Because the counter signal value changes with every message transmission, this triggers the transmission of the mode-dependent signals. By using counter signals in this way, you can work with signals which use the same bytes of a message. Counter signals used in this way are often called mode counters.

- Using counter signals in ControlDesk

  In ControlDesk, you can transmit the signal's constant, counter, or increment counter value. The increment counter value is the counter value incremented by the signal's constant value.

You can specify counter signals on the Counter Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Error value A static signal value that indicates an error. You can specify an error value for each signal to be transmitted. Alternatively, error values can be defined in a database file. In ControlDesk, you can switch to transmit the signal's error value, constant value, etc. However, you cannot change the error value during run time. In ControlDesk, you can use a Variable Array (MultiState LED value cell type) to indicate if the error value is transmitted.

You can specify error values on the Signal Errors Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Dynamic value A signal value that is transmitted for a defined number of times.

- *Behavior of dynamic values*: You can specify to use a dynamic value for each signal to be transmitted. In ControlDesk, you can specify to transmit the signal's constant value or dynamic value. If you switch to the dynamic value, it is transmitted for a defined number of times (countdown value). Then signal manipulation automatically switches back to the signal manipulation option used before the dynamic value.

- *Example of using dynamic values*: Suppose you specify a dynamic value of 8 and a countdown value of 3. If you switch to the dynamic value of the signal, the signal value 8 is sent the next 3 times the TX message is transmitted. Then the signal manipulation option is reset.

You can specify dynamic values on the pages located in the Dynamic Signal Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Modeling a FlexRay Bus Interface

Introduction

The SCALEXIO system or MicroAutoBox III can simulate a FlexRay bus. You can connect your ECU to the simulated FlexRay bus to perform restbus simulation for communication testing purposes, and to simulate FlexRay message errors.

Required knowledge

It is assumed that you have good knowledge of the basics of FlexRay networks and FIBEX (ASAM MCD-2 NET) files and/or AUTOSAR System Templates.

Where to go from here

Information in this section

General Information on Modeling FlexRay Communication

Introduction

A FlexRay bus interface must be modeled in Simulink using the dSPACE FlexRay Configuration Package. The following topics provide information on how to work with the **FlexRay Configuration Blockset** and the **FlexRay** function block.

Where to go from here

Information in this section

Features of the dSPACE FlexRay Configuration Package

Introduction

This topic lists the features of the dSPACE FlexRay Configuration Package.

Components of the dSPACE FlexRay Configuration Package

dSPACE provides the dSPACE FlexRay Configuration Package for you to connect an ECU to a FlexRay bus simulated by the SCALEXIO system or the MicroAutoBox III. The dSPACE FlexRay Configuration Package consists of the following components:

- FlexRay Configuration Tool
- FlexRay Configuration Blockset
- **FlexRay** function block type in ConfigurationDesk

FlexRay Configuration Tool

With the FlexRay Configuration Tool, you can configure a SCALEXIO system or a MicroAutoBox III as a simulation node in a FlexRay network. You can generate all the code files that are necessary for modeling with the FlexRay Configuration Blockset and simulating with ControlDesk.

For information on how to work with the FlexRay Configuration Tool, refer to the FlexRay Configuration Tool Guide 🕮.

FlexRay Configuration Blockset

The main features of the FlexRay Configuration Blockset are:

- Generating FlexRay communication blocks configured for your FlexRay network. The necessary configuration data is created by the FlexRay Configuration Tool according to a FIBEX file or AUTOSAR System Template.
- Supporting single-channel and dual-channel FlexRay systems
- Providing configured Simulink blocks for
  - Time-triggered task execution
  - Sending and receiving PDUs
  - Updating the configured Simulink blocks if the configuration was changed
- Simulating the FlexRay node on a SCALEXIO system or a MicroAutoBox III
- Supporting PDU-based modeling
- Simulating several FlexRay buses on one dSPACE SCALEXIO Processing Unit or a MicroAutoBox III, for example, to simulate a gateway

For detailed information on the FlexRay Configuration Blockset, refer to FlexRay Configuration Blockset Reference 🕮.

FlexRay function block type

The **FlexRay** function block type in ConfigurationDesk lets you specify functions for:

- Configuring the controllers
- Starting and stopping the controllers
- Reading the statuses of the controllers
- Configuring the deadline violation handling
- Enabling or disabling the communication of an ECU
- Synchronizing the FlexRay cluster and the SCALEXIO or MicroAutoBox III system

For detailed information on the **FlexRay** function block type, refer to FlexRay (ConfigurationDesk Function Block Properties 🕮).

Related topics

Basics

Overview of the Workflow.. 144

Supported Channel Types

Introduction

The FlexRay Configuration Package supports various channel types for configuring a SCALEXIO system or a MicroAutoBox III as a simulation node in a FlexRay network.

Supported SCALEXIO channel types

The FlexRay Configuration Tool and the FlexRay Configuration Blockset support the following SCALEXIO channel types:

- Bus 1: Channel type on a DS2671 Bus Board.
- FlexRay 1: Channel type on a DS2672 Bus Module.
- FlexRay 2: Channel type on a DS6311 FlexRay Board.

Supported MicroAutoBox III channel types

The FlexRay Configuration Tool and the FlexRay Configuration Blockset support the following MicroAutoBox III channel types:

- FlexRay 3: Channel type on a DS4340 FlexRay Interface Module.
- FlexRay 4: Channel type on a DS1521 Bus Board.

More hardware data

DS2671 Bus Board For more board-specific data, refer to DS2671 Bus Board (SCALEXIO Hardware Installation and Configuration 🕮).

DS2672 Bus Module For more board-specific data, refer to DS2672 Bus Module (SCALEXIO Hardware Installation and Configuration 🕮).

DS6311 FlexRay Board For more board-specific data, refer to DS6311 FlexRay Board (SCALEXIO Hardware Installation and Configuration 🕮).

DS4340 FlexRay Interface Module For more module-specific data, refer to DS4340 FlexRay Interface Module Data Sheet (MicroAutoBox III Hardware Installation and Configuration 🕮).

Related topics

Basics

Introduction to the Function Block (FlexRay) (ConfigurationDesk I/O Function Implementation Guide 🕮)

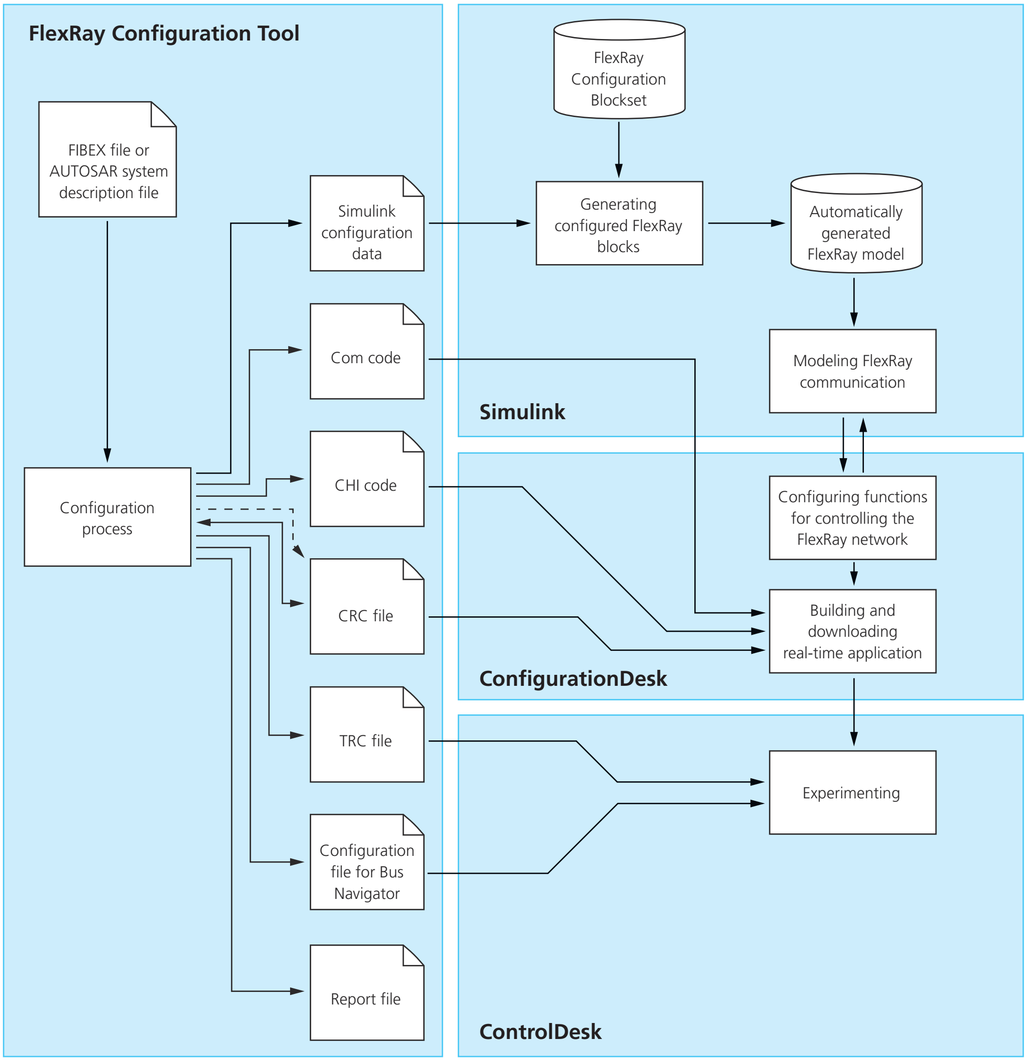
Overview of the Workflow

Introduction

The workflow for setting up a FlexRay network for real-time simulation with a SCALEXIO or MicroAutoBox III system is divided into several process steps.

Workflow

The following illustration gives an overview of the workflow.


FlexRay Configuration Tool
FIBEX file or AUTOSAR system description file
Configuration process
Simulink configuration data
Com code
CHI code
CRC file
TRC file
Configuration file for Bus Navigator
Report file
FlexRay Configuration Blockset
Generating configured FlexRay blocks
Automatically generated FlexRay model
Modeling FlexRay communication
Simulink
Configuring functions for controlling the FlexRay network
Building and downloading real-time application
ConfigurationDesk
Experimenting
ControlDesk

Tip

Instead of ConfigurationDesk, you can also use ControlDesk to download the real-time application.

The workflow is divided into several process steps:

1. You have to create FlexRay configuration data using the FlexRay Configuration Tool. This creates Simulink configuration data, communication code, CHI code, a TRC file, and a configuration file for the Bus Navigator according to a FIBEX or AUTOSAR system description file. The Simulink configuration data holds all the parameters for building the Simulink blocks needed to model the FlexRay communication. For detailed information, refer to FlexRay Configuration with the FlexRay Configuration Tool (FlexRay Configuration Tool Guide 📖).

Tip

You can perform the configuration process and the model design on different PCs. To do so, you must copy the generated files (Simulink configuration data, Com, CHI code, and TRC file) to the PC where the FlexRay Configuration Blockset is installed.

2. To model the FlexRay communication, you need Simulink blocks which are configured for FlexRay. The FlexRay Configuration Blockset contains a MATLAB command which generates a model based on the Simulink configuration data (this model is called *automatically generated FlexRay model* below). The blocks in the automatically generated FlexRay model can be used for PDU-based modeling and for updating your model if the FlexRay configuration has changed. Refer to Generating Blocks Configured for a FlexRay Network on page 148. In addition, the automatically generated FlexRay model contains blocks for task-based modeling. Refer to Handling FlexRay Communication Tasks on page 147.
3. You have to design the FlexRay communication in your Simulink model using the blocks from the automatically generated FlexRay model. Refer to Modeling FlexRay Communication on page 152.
4. You have to analyze the Simulink model to make the FlexRay configuration of your model available in ConfigurationDesk. Refer to Synchronizing the Simulink Model Interface and the ConfigurationDesk Model Interface on page 41.
5. You have to configure your FlexRay network using the **FlexRay** function block type in ConfigurationDesk. For details, refer to Configuring a FlexRay Network on page 174.
6. You have to generate the model interface, copy the model port blocks from the generated interface model to your Simulink model, and connect them to the blocks in your Simulink model. The model port blocks are named automatically. Refer to Creating the Model Interface to ConfigurationDesk by Using Model Port Blocks on page 55.

Tip

It is also possible to create the entire interface in your Simulink model using the blocks from the Model Interface Blockset, analyze the model afterwards in ConfigurationDesk, and map the model port blocks to the **FlexRay** function block. However, in this case the model port blocks are not named automatically.

7. When the design of your FlexRay network is finished, you can build the real-time application, download it to the SCALEXIO system or the MicroAutoBox III and execute it. The TRC file generated by the FlexRay Configuration Tool contains the PDUs' variables, signals and raw data which are selected to be controlled via a TRC file. To visualize them during the execution of the real-time application, you can build a data connection with ControlDesk instruments. Refer to Simulating FlexRay Networks on page 187.

Related topics

Basics

Introduction to the FlexRay Configuration Tool (FlexRay Configuration Tool Guide 🕮)

References

General Information on the FlexRay Configuration Blockset (FlexRay Configuration Blockset Reference 🕮)

Handling FlexRay Communication Tasks

Introduction

The dSPACE FlexRay Configuration Package lets you configure several options for task handling.

Configuring tasks with the FlexRay Configuration Tool

The FlexRay Configuration Tool provides methods for you to configure and create communication tasks, application tasks, and to specify some options for the synchronization task. For details, refer to Creating Tasks (FlexRay Configuration Tool Guide 🕮).

Communication and application task priority

The priority of communication and application tasks is specified in ConfigurationDesk. The specified priority applies to all the Hardware-Triggered Runnable Function blocks in your Simulink model that are assigned to the tasks configured in the FlexRay Configuration Tool, and to the communication tasks specified in the FlexRay Configuration Tool. For details, refer to Building the Signal Chain for FlexRay Communication (ConfigurationDesk Real-Time Implementation Guide 🕮).

Task handling in ConfigurationDesk

During the model analysis, ConfigurationDesk creates a Configuration Port block in the model topology for each identified FLEXRAYCONFIG UPDATE block in your Simulink model. The Configuration Port block is used to configure a FlexRay function block. After mapping the Configuration Port block to the Configuration function port of the FlexRay function block, you can view information on the task configuration in the Properties Browser and the Executable Application view in the Table window.

Refer to Functions for Configuring a FlexRay Network on page 175.

In addition, the FlexRay function block provides functions for you to handle the tasks of your real-time application. Refer to Common FlexRay Functions on page 178.

Generating Blocks Configured for a FlexRay Network

Introduction

Before you can design the FlexRay communication in Simulink, you have to generate Simulink blocks which are configured for modeling FlexRay communication. The automatically generated FlexRay model contains a FLEXRAYCONFIG UPDATE block configured for your FlexRay configuration. If you change the FlexRay configuration afterwards, you can use this block to update all the FlexRay blocks used in your Simulink model.

Where to go from here

Information in this section

How to Generate Blocks for Modeling a FlexRay Communication

Objective

You need an automatically generated FlexRay model containing Simulink blocks configured for modeling the FlexRay communication.

Automatically generated FlexRay model

The FlexRay Configuration Tool writes a file containing all the parameters for designing the communication of your FlexRay application (Simulink configuration data). Before you can start to model the FlexRay communication, you must start a generation process via a MATLAB command. This command generates an automatically generated FlexRay model which contains blocks on the basis of the Simulink configuration data. You can use the generated model as a library for designing the FlexRay communication.

Preconditions

You must have generated the Simulink configuration data using the FlexRay Configuration Tool (see Basics of Code Generation (FlexRay Configuration Tool Guide 📖)).

Method

To generate blocks for modeling a FlexRay communication

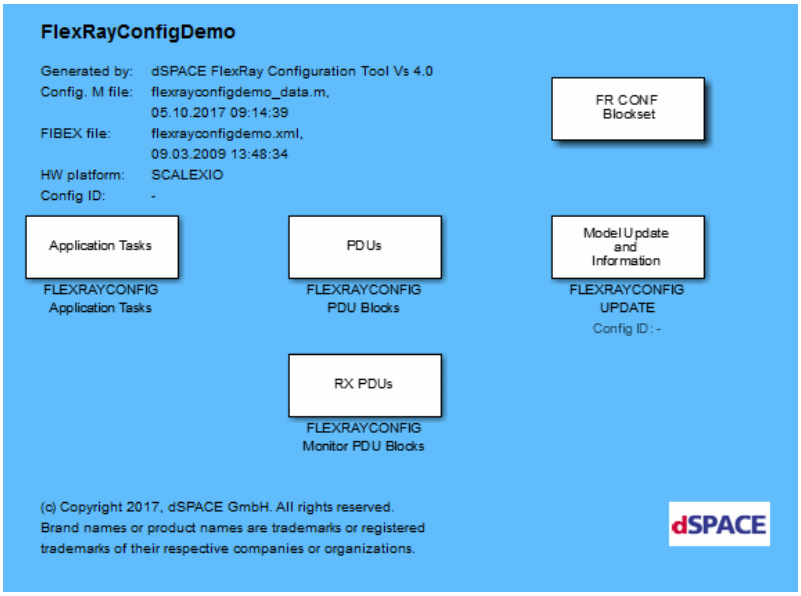
1 Open MATLAB.

2 In the MATLAB Command Window, enter `dsfr_modelgenerate('FileName')`.

`FileName` is the file name of an M file containing the Simulink configuration data. It is generated by the FlexRay Configuration Tool. If the M file is not located in the current folder, you must additionally specify the whole path, for example, `c:\mydir\filename.m`. You can specify further optional parameters, see dsfr_modelgenerate (FlexRay Configuration Blockset Reference 🕮).

Result

The automatically generated FlexRay model is generated with the name specified in the FlexRay Configuration Tool. After the generation process, the model is opened.



FlexRayConfigDemo

Generated by: dSPACE FlexRay Configuration Tool Vs 4.0
Config. M file: flexrayconfigdemo_data.m, 05.10.2017 09:14:39
FIBEX file: flexrayconfigdemo.xml, 09.03.2009 13:48:34
HW platform: SCALEXIO
Config ID: -

(c) Copyright 2017, dSPACE GmbH. All rights reserved.
Brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

The automatically generated FlexRay model contains the following blocks which are configured for FlexRay with the Simulink configuration data:

FLEXRAYCONFIG UPDATE A configured FLEXRAYCONFIG UPDATE block is in the root level of the generated model. It contains information on the configuration and the FIBEX file or AUTOSAR system description file. This block is mandatory in your Simulink model. During the model analysis, ConfigurationDesk creates a Configuration Port block in the model topology for each identified FLEXRAYCONFIG UPDATE block. The Configuration Port block is used to configure a FlexRay function block. The FLEXRAYCONFIG UPDATE block can also be used for updating your Simulink model if the FlexRay configuration changes.

Note

> There must be exactly one FLEXRAYCONFIG UPDATE block for each FlexRay configuration in the Simulink model. Because a Simulink model can contain up to four FlexRay configurations, there can be up to four FLEXRAYCONFIG UPDATE blocks in a Simulink model.

FLEXRAYCONFIG PDU Blocks The FLEXRAYCONFIG PDU Blocks subsystem contains all the PDUs which can be used for modeling FlexRay communication.

FLEXRAYCONFIG Application Tasks The FLEXRAYCONFIG Application Tasks subsystem contains all the created application tasks.

FR CONF Blockset The FR CONF Blockset library contains the FlexRay configuration blocks which are the basis for the automatically generated FlexRay model. You cannot use these blocks directly in your Simulink model.

The blocks in the model are clustered in subsystems according to their types. For information on the structure of the automatically generated FlexRay model, refer to Structure of the Automatically Generated FlexRay Model (FlexRay Configuration Blockset Reference 📖).

Related topics

References

dsfr_modelgenerate (FlexRay Configuration Blockset Reference 📖)

How to Update the FlexRay Blocks in Simulink Models

Objective

If you used the blocks of the automatically generated FlexRay model in your Simulink model and afterwards changed the FlexRay configuration, you must update your Simulink model. You can use the FLEXRAYCONFIG UPDATE block for updating your model automatically. This is useful when you have a lot of changes in a large model.

Basics

The automatically generated FlexRay model contains a FLEXRAYCONFIG UPDATE block configured for your FlexRay configuration. If you change the FlexRay configuration afterwards, you can use this block to update all the FlexRay blocks used in your Simulink model.

Note

You must use the FLEXRAYCONFIG UPDATE block which was configured first for your configuration. Do not replace this block by the new generated FLEXRAYCONFIG UPDATE block, which cannot update the blocks used before.

The block updates all the FlexRay blocks used in your Simulink model. In addition, it generates a reduced automatically generated FlexRay model named `<ModelName>_diff.mdl`. This model contains only the Simulink blocks of the FlexRay configuration which are not used in your FlexRay model. All the blocks which are used in your Simulink model are updated. You can delete obsolete blocks from your Simulink model.

Note

The mapping subsystems which belong to the FLEXRAYCONFIG PDU RX and FLEXRAYCONFIG PDU TX blocks are not updated. You must update them manually.

Tip

It is recommended to use the update function of the FLEXRAYCONFIG UPDATE block even if the FlexRay configuration has not changed. The reduced automatically generated FlexRay model is especially useful for keeping track of large Simulink models.

Preconditions

- You must have generated the automatically generated FlexRay model.
- The FLEXRAYCONFIG UPDATE block must reside in your Simulink model.

Method

To update the FlexRay blocks in Simulink models

1. Open the FLEXRAYCONFIG UPDATE block in your Simulink model.
2. On the Model Update page, use the Browse button to choose a file as the basis for the FlexRay configuration.
3. Select or clear the Delete obsolete block(s) option.

   If you select the option, all the obsolete FlexRay blocks are deleted in your Simulink model. Obsolete blocks are blocks which were dragged from the previous automatically generated FlexRay configuration and do not exist in the updated FlexRay configuration.
4. Click Update.

Result

All the configured FlexRay blocks in your Simulink model are updated. A reduced FlexRay model is generated automatically, containing only blocks which are not used in your Simulink model. Two log files are created in the working directory.

When the update process is completed, you can find links to these two log files in the MATLAB workspace:

- `<ModelName>_diff.mdl` containing all the updated blocks.
- `<ModelName>_UpdateSummary.log` lists the unused blocks from the library and old blocks in the model. The log file also displays the number of updated blocks.

Related topics

References

FLEXRAYCONFIG UPDATE (FlexRay Configuration Blockset Reference 🕮)

Modeling FlexRay Communication

Introduction

The FlexRay configuration contains PDUs (protocol data units) which comprise several signals. Using PDU-based modeling, you can handle several signals with one Simulink block.

FLEXRAYCONFIG PDU TX and FLEXRAYCONFIG PDU RX blocks can be used for PDUs. PDUs are defined in FIBEX+ and FIBEX 3.x versions and AUTOSAR System Templates. There are no PDU elements available in FIBEX 2.0 and lower versions, only frames. The FLEXRAYCONFIG PDU TX and FLEXRAYCONFIG PDU RX blocks can also be used for frames, with one PDU in each frame. The block dialog shows whether real PDUs or frames are used.

Where to go from here

Information in this section

Information in other sections

FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 🕮)
To send a PDU via a FlexRay network.

FLEXRAYCONFIG PDU RX (FlexRay Configuration Blockset Reference 🕮)
To read a PDU from the FlexRay bus.

How to Send or Receive Signals of PDUs

Objective

The configuration process generates a block and mapping subsystems for each PDU which is sent or received. You can use these in the Simulink model.

Blocks for sending or receiving PDUs

PDUs and their signals are sent using the FLEXRAYCONFIG PDU TX block, and received using the FLEXRAYCONFIG PDU RX block. During the configuration process, these blocks are generated for each PDU which is sent or received. To simplify the handling of the blocks, several mapping subsystems are also generated and connected to them. The blocks and the mapping subsystems are stored in the automatically generated FlexRay model.

The names of the generated blocks have information on the PDU:

`<PDU-/Frame-Name>_ID_BC_CR_PDU_TX/RX` (for single-channel FIBEX or AUTOSAR system description files)
`<PDU-/Frame-Name>_ID_BC_CR_PDU_TX/RX_Ch` (for dual-channel FIBEX or AUTOSAR system description files)

- ID: Slot ID
- BC: Base cycle
- CR: Cycle repetition
- PDU_TX: Send PDU
  PDU_RX: Receive PDU
- Ch: Used channel (A, B, AB)

Method

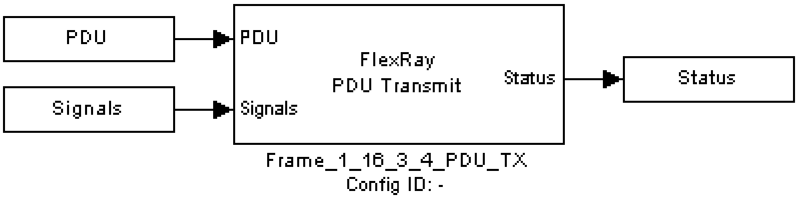
To send or receive signals of PDUs

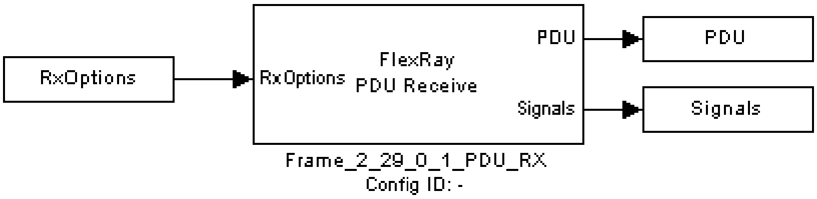
1 Open the automatically generated FlexRay model.

2 Open the subsystems to access the FLEXRAYCONFIG PDU RX or FLEXRAYCONFIG PDU TX blocks. The blocks are nested in several subsystems. Open the subsystems in the following order:

   1. FLEXRAYCONFIG PDU Blocks subsystem
   2. <ECU_Name> subsystem (<ECU_Name> is the short name of the ECU which sends or receives the signal)
   3. Static Transmission, Dynamic Transmission, Network Management or <Application_Name> subsystem (<Application_Name> is the name of an application task that PDUs and their signals are assigned to)

   The subsystems contain all the PDUs and their signals which are configured for sending or receiving. Several mapping subsystems are used to structure the inports and outports of the PDU blocks, see the following examples.





For details on the structure of the subsystems, refer to FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 📖) and FLEXRAYCONFIG PDU RX (FlexRay Configuration Blockset Reference 📖).

3 To send a PDU, drag its **FLEXRAYCONFIG PDU TX** block and the connected mapping subsystems (**PDU**, **Signals**, and **Status**), if available, to the Simulink model.

To receive a PDU, drag its **FLEXRAYCONFIG PDU RX** block and the connected mapping subsystems (**RXOptions**, **PDU**, and **Signals**), if available, to the Simulink model.

> **Tip**
>
> Instead of the mapping subsystems, you can also use a Bus Creator block and connect it to the FLEXRAYCONFIG PDU RX block. If you do so, all the signals the FLEXRAYCONFIG PDU block expects must be connected to the Bus Creator block.

Result

The Simulink model is prepared for sending or receiving the signals of the PDU. The signals are available in the **Signals** mapping subsystem. You can connect each signal to further blocks in the Simulink model.

Related topics

HowTos

How to Generate Blocks for Modeling a FlexRay Communication.. 148

References

FLEXRAYCONFIG PDU RX (FlexRay Configuration Blockset Reference 📖)
FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 📖)
Structure of the Automatically Generated FlexRay Model (RTI FlexRay Configuration Blockset Reference 📖)

Sending Static PDUs and Sub-PDUs

Introduction

You can enable or disable the sending of static PDUs and sub-PDUs.

Enabling or disabling static PDUs

You can enable or disable the sending of static PDUs using the **FLEXRAYCONFIG PDU TX** block. The block has two inports in the PDU mapping subsystem which you can use:

HWEnable The **HWEnable** inport enables or disables the sending of static TX PDUs via hardware: that is, by enabling or disabling the controller TX buffer which is reserved for the relevant bus slot. As a consequence, all the PDUs which share the same bus slot and therefore are assigned to the same controller TX buffer are enabled or disabled together. Some PDUs cannot be enabled or disabled in this way, for example, startup and sync PDUs.

Note

- If the Simulink model contains several PDU TX blocks whose PDUs share the same bus slot, their HWEnable inports manipulate the same slot. In this case, the setting of the most recently calculated HWEnable port automatically becomes the valid setting. To safely enable or disable the sending of static PDUs which share the same slot, you must either enable or disable all the HWEnable inputs consistently at the same time, or use only one PDU TX block in your model, if possible.
- Enabling or disabling the sending of static TX PDUs via hardware has a higher priority than via software. As a consequence, all TX PDUs which share the same bus slot are automatically enabled by hardware *and* software, if one of them is enabled by its HWEnable inport.

SWEnable The SWEnable inport enables or disables the sending of each static TX PDU via software. This is possible for each static PDU.

Sending null frames or regular data

The FlexRay Configuration Tool allows explicit sending of null frames for static PDUs. When a static PDU is disabled or enabled via software, either regular data or a null frame is sent, according to the database version and settings made in the FlexRay Configuration Tool. The following describes which data is sent in each case.

Static PDU is disabled via software If a static PDU is *disabled via software* (`SWEnable` = 0), the data that is sent depends on the following factors:

- Database version
- Setting of the SW Enable Configuration property (see General Page (FlexRay Configuration Tool Reference 📖))
- Settings of the CHI Code Generator (see Generators Page (FlexRay Configuration Tool Reference 📖))

The following table shows which data is sent in a specific case.

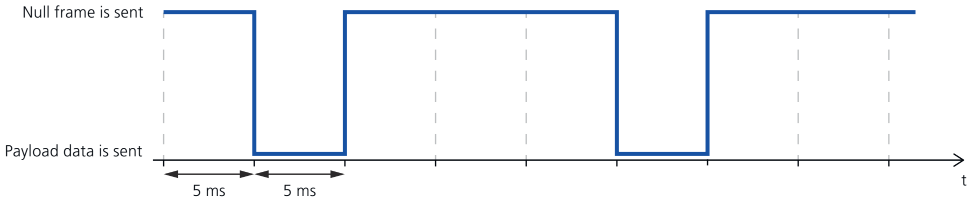
Database Version	SWEnable Configuration (General Properties)	Static TX Buffer Transmission Mode (CHI Code Generator)	Behavior
FIBEX version ≤ 2.0	Control of L-PDU commit to FlexRay buffer[1)]	Event (null frame used)	Null frames are sent.
		State (old value used)	Old data is sent.
FIBEX+, FIBEX 3.x, FIBEX 4.1.x, or	Control of L-PDU commit to FlexRay buffer	Event (null frame used)	Null frames are sent.
		State (old value used)	Old data is sent. The update bit has the value that was set before SWEnable was set to 0.

Database Version	SWEnable Configuration (General Properties)	Static TX Buffer Transmission Mode (CHI Code Generator)	Behavior
AUTOSAR System Template	Control of I-PDU payload data update	Event (null frame used)	The update bit is set to 0 and old data is sent. **Note** An update bit value of 0 due to the SWEnable setting can be overruled. This can happen when the update bit of a PDU is manipulable. If the UpdateBitEnable variable is 1 (which means that automatic calculation of the update bit is disabled), the update bit of the PDU is set to the value specified by the UpdateBitValue variable. If SWEnable is 0 and the update bit value fed to the UpdateBitValue inport is 1, the PDU still sends old data. If UpdateBitEnable is 0, the SWEnable setting is used as specified to enable/disable the sending of static TX PDUs.
		State (old value used)	The update bit is set to 0 and old data is sent.

[1] This setting cannot be changed.

Static PDU is enabled via software If a static PDU is *enabled via software* (`SWEnable` = 1), usually the PDU payload data is sent. However, there is one exception to this rule: An LPDU that contains exactly one PDU that neither has been updated nor has a PDU update bit, has a null frame sent instead of the LPDU with old payload data. This transmission behavior enables you to detect whether a received PDU contains updated payload data, even if no PDU update bit exists.

The following illustration shows an example for this null frame transmission behavior.



In the example, the LPDU is sent cyclically every 5 ms. Its cycle repetition (defined in absolutely scheduled timing) is 1. The only PDU contained in the LPDU has a cyclic timing of 20 ms. The PDU is updated every 20 ms, which means in every fourth transmission cycle of the LPDU. When the PDU is updated, no null frame is sent. Instead, the new PDU data is packed into the LPDU and the new LPDU data is committed to the FlexRay controller. In the cycles in which the PDU is not updated but has old data, a null frame is sent.

> **Note**
>
> Timings of PDUs can change during run time. For example, this can happen when you use a static PDU with sub-PDUs that have their own timings or if you work with different transmission modes for a PDU.

If a static PDU is *enabled via software*, the data which is sent depends on the following factors:

- Database version
- Setting of the **SW Enable Configuration** property (see General Page (FlexRay Configuration Tool Reference 📖))
- Settings of the CHI Code Generator (see Generators Page (FlexRay Configuration Tool Reference 📖))

Sending null frames instead of old payload data is possible if the following conditions are met:

- Database version = FIBEX+, FIBEX 3.x, FIBEX 4.1.x, or AUTOSAR System Template
- **SW Enable Configuration** = 'Control of L-PDU commit to FlexRay buffer'
- **Static TX buffer transmission mode** = 'Event (null frame used)'

In all other cases, old payload data is sent as a rule.

Triggering sub-PDUs

A sub-PDU is subordinate to a PDU. It contains signals only. Sub-PDUs of a static PDU can have their own timing (CT parameter), but use the same slot of the superordinate PDU (specified by ID, BC, and CR parameters). A sub-PDU of a static PDU can be triggered by triggering the static PDU with a switch code. The trigger type SSC is used for this.

The **FLEXRAYCONFIG PDU TX** block provides the **EnableMultiplexerSwitchCode** inport in the PDU system, which lets you specify the switch code for static sub-PDUs.

Related topics

HowTos

How to Manipulate the Update Bit of a PDU..166

References

FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 📖)
FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 📖)

Sending Dynamic PDUs or Sub-PDUs

Introduction

Dynamic PDUs or sub-PDUs must be sent by using the **FLEXRAYCONFIG PDU TX** block, see Triggering of Dynamic Frames and Subframes (FlexRay Configuration Tool Guide 📖).

> **Tip**
>
> The following section describes how you can trigger the dynamic PDUs or sub-PDUs via the Simulink model. You can also trigger them using the TRC file, see How to Prepare the Manipulation or Monitoring of Frames/PDUs and Signals via TRC File (FlexRay Configuration Tool Guide 📖) and Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

Enabling PDUs

Cyclic PDUs (DC and DEC types) are enabled by default if they contain no sub-PDUs. You can use the **TxEnable** inport in the **PDU** mapping subsystem which is connected to the **FLEXRAYCONFIG PDU TX** block to enable or disable the sending of a dynamic PDU.

Triggering PDUs

The timing of a PDU is specified by several parameters (ID, BC, CR, and CT). Whether a dynamic PDU can be sent cyclically or event-triggered depends on the PDU trigger type DE, DC, or DEC (see Triggering of Dynamic Frames and Subframes (FlexRay Configuration Tool Guide 📖)). A PDU can contain signals and/or sub-PDUs.

Triggering sub-PDUs

A sub-PDU is subordinate to a PDU. Sub-PDUs of a PDU can have their own timing (CT parameter) but use the same slot of the superordinate PDU (specified by ID, BC, and CR parameters). A switch code specifies the sub-PDU to trigger. Whether a dynamic sub-PDU can be sent cyclically or event-triggered depends on the trigger type SDE, SDC, or SDEC (see Triggering of Dynamic Frames and Subframes (FlexRay Configuration Tool Guide 📖)). A sub-PDU contains signals only.

If a sub-PDU has no timing, it inherits the timing of the (parent) PDU in FIBEX versions lower than 3.0. Since FIBEX version 3.0 and for AUTOSAR System Templates, sub-PDUs cannot inherit the timing of the parent PDU. These sub-PDUs are interpreted as dynamic event sub-PDUs. Sub-PDUs cannot be sent cyclically, they must be triggered.

Related topics

References

FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 🕮)
FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮)

How to Handle Checksum Calculation for a PDU

Objective

If you have assigned a CRC algorithm to a PDU in the FlexRay Configuration Tool, you can enable or disable checksum calculation or select another CRC algorithm.

Basics

You can implement your own checksum algorithm (CRC algorithm) in a C-coded source file (CRC C file) using a special template. You can assign the CRC algorithm to TX or RX PDUs in the FlexRay Configuration Tool. The **FLEXRAYCONFIG PDU RX** and **FLEXRAYCONFIG PDU TX** blocks have inports that enable or disable checksum calculation and inports that select another CRC algorithm. For details, refer to Basics on Implementing Checksum Algorithms (FlexRay Configuration Tool Guide 🕮).

The method described below must be executed for each PDU whose checksum calculation you want to change.

Changing CRC C file

You can use the **FLEXRAYCONFIG UPDATE** block to select a CRC C file other than that specified with the FlexRay Configuration Tool, refer to CRC Settings Page (FLEXRAYCONFIG UPDATE) (FlexRay Configuration Blockset Reference 🕮).

CRC manipulation via TRC file

If CRC calculation is enabled and selected for the TRC file in the FlexRay Configuration Tool (see Basics on Implementing Checksum Algorithms (FlexRay Configuration Tool Guide 🕮)), appropriate variables are also written to the TRC file. Thus, you can manipulate CRC calculation via ControlDesk, refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

Preconditions

- You must have assigned a CRC algorithm to the PDUs. For details, refer to How to Assign a Checksum Algorithm to Frames (FlexRay Configuration Tool Guide 🕮).
- The **FLEXRAYCONFIG PDU RX** or **FLEXRAYCONFIG PDU TX** block and the connected mapping subsystems of the PDU must reside in the Simulink model (see How to Send or Receive Signals of PDUs on page 153).

Method

To handle checksum calculation for a PDU

1 If the PDU is transmitted, open the PDU mapping subsystem connected to its FLEXRAYCONFIG PDU TX block.

If the PDU is received, open the RXOptions mapping subsystem connected to its FLEXRAYCONFIG PDU RX block.

2 Connect the inports of the block to appropriate Simulink blocks.

Port	Description
CRCEnable	Enables or disables checksum calculation.
CRCType	Selects the CRC algorithm. The value must be one of the ID values that were specified in the FlexRay Configuration Tool.

3 Repeat the steps above for all the PDUs whose CRC algorithm you want to change.

Result

You have changed the checksum calculation for PDUs.

Related topics

References

FLEXRAYCONFIG PDU RX (FlexRay Configuration Blockset Reference 🕮)
FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 🕮)

How to Send PDUs in Raw Format

Objective

When a PDU is configured for raw data access, the FLEXRAYCONFIG PDU TX block has additional ports. These ports can be used to access each bit or byte of a PDU, ignoring the specified signals.

Raw data in TRC file

If raw data access is enabled for PDUs and selected for the TRC file in the FlexRay Configuration Tool (see How to Configure a Frame for Raw Data Access (FlexRay Configuration Tool Guide 🕮)), variables for access are also written to the TRC file. Thus, you can manipulate raw data via ControlDesk, refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

Preconditions

- The PDU must be configured for raw data access in the FlexRay Configuration Tool, refer to How to Configure a Frame for Raw Data Access (FlexRay Configuration Tool Guide 🕮).
- The FLEXRAYCONFIG PDU TX block of the PDU must be in the Simulink model (see How to Send or Receive Signals of PDUs on page 153).

Method

To send PDUs in raw format

1. Open the PDU mapping subsystem that is connected to the FLEXRAYCONFIG PDU TX block.
2. Connect the inports of the subsystem to appropriate Simulink blocks:

Port	Description
RawDataEnable	Enables raw data access
RawDataTxBytes	Provides a vector of bytes containing data to be written to the TX PDU. The number of bytes is specified by the Max number of raw data bytes property, which can be specified in the FlexRay Configuration Tool.
RawDataStartPosition	Specifies the start position within the TX PDU in bits. The value must be specified in bits and is therefore not limited to bytes. A value of 0 is at the first bit position. The maximum value depends on the payload length of the TX PDU, which is specified in bytes.
RawDataLength	Specifies the number of bits which are accessed. If the value is 0, no data is written. The maximum value depends on the Max number of raw data bytes property which can be specified in the FlexRay Configuration Tool. It is specified in bytes.

   For more details on the ports, refer to FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮).

3. Open the Status mapping subsystem that is connected to the FLEXRAYCONFIG PDU TX block.
4. Connect the outports of the subsystem to appropriate Simulink blocks:

Port	Description
RawDataAccessStatus	Displays status information of raw data access.

   For more details on the ports, refer to FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮).

Result

Your model is prepared for raw data access. You can change each bit of the PDU to be sent.

Related topics

HowTos

How to Generate Blocks for Modeling a FlexRay Communication.. 148

How to Prepare the Manipulation or Monitoring of Frames/PDUs and Signals via TRC File (FlexRay Configuration Tool Guide 🕮)
How to Receive PDUs in Raw Format... 163

References

FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 🕮)
Structure of the Automatically Generated FlexRay Model (FlexRay Configuration Blockset Reference 🕮)

How to Receive PDUs in Raw Format

Objective

When a PDU is configured for raw data access, the FLEXRAYCONFIG PDU RX block has additional ports. The ports can be used to access each bit or byte of a PDU, ignoring the specified signals.

Raw data in TRC file

If raw data access is enabled for PDUs and selected for the TRC file in the FlexRay Configuration Tool (see How to Configure a Frame for Raw Data Access (FlexRay Configuration Tool Guide 🕮)), variables for access are also written to the TRC file. Thus, you can monitor raw data in ControlDesk, refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

Preconditions

- The PDU must be configured for raw data access in the FlexRay Configuration Tool, refer to How to Configure a Frame for Raw Data Access (FlexRay Configuration Tool Guide 🕮).
- The FLEXRAYCONFIG PDU RX block of the PDU must be in the Simulink model (see How to Send or Receive Signals of PDUs on page 153).

Method

To receive PDUs in raw format

1 Open the RXOptions mapping subsystem that is connected to the FLEXRAYCONFIG PDU RX block.

2 Connect the inports of the block to appropriate Simulink blocks.

Port	Description
RawDataStartPosition	Specifies the start position within the RX PDU in bits. The value must be specified in bits and is therefore not limited to bytes. A value of 0 is at the first bit position. The maximum value depends on the payload length of the RX PDU, which is specified in bytes.
RawDataLength	Specifies the number of bits which are accessed. If the value is 0, no data is read and the RawDataRxBytes outport contains the previously read data.

Port	Description
	The maximum value depends on the **Max number of raw data bytes** property which can be specified in the FlexRay Configuration Tool. It is specified in bytes.

For more details on the ports, refer to FLEXRAYCONFIG PDU RX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮).

3 Connect the outports of the block to appropriate Simulink blocks.

Port	Description
RawDataRxBytes	Provides a vector of bytes which contains the data which are monitored by the RX PDU.
RawDataAccessStatus	Displays status information on raw data access.

For more details on the ports, refer to FLEXRAYCONFIG PDU RX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮).

Result

Your model is prepared for raw data access. You can read the bits of the received PDU.

Related topics

HowTos

How to Generate Blocks for Modeling a FlexRay Communication.. 148
How to Prepare the Manipulation or Monitoring of Frames/PDUs and Signals via TRC File (FlexRay Configuration Tool Guide 🕮)
How to Send PDUs in Raw Format... 161

References

FLEXRAYCONFIG PDU RX (FlexRay Configuration Blockset Reference 🕮)
Structure of the Automatically Generated FlexRay Model (FlexRay Configuration Blockset Reference 🕮)

How to Manipulate the Payload Length of a PDU

Objective

You can manipulate the payload length of the TX PDUs or read the payload length of RX PDUs when you use raw data access.

Preconditions

- The PDU must be configured for raw data access, and payload length manipulation or reading must be enabled in the FlexRay Configuration Tool, refer to How to Configure a Frame for Raw Data Access (FlexRay Configuration Tool Guide 🕮).

> **Note**
>
> A PDU can be configured for payload length manipulation only if it is mapped 1:1 with a FlexRay frame.

- The FLEXRAYCONFIG PDU TX block of the PDU must be in the Simulink model (see How to Send or Receive Signals of PDUs on page 153).

Method

To manipulate the payload length of a PDU

1 Open the PDU mapping subsystem connected to the FLEXRAYCONFIG PDU TX block.

2 Connect the inports of the block to appropriate Simulink blocks.

Port	Description
PayloadLengthEnable	Enables or disables the manipulation of the payload length.
PayloadLengthValue	Sets the payload length value in bytes of the PDU.

For more details on the ports, refer to FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮).

3 Open the Status mapping subsystem that is connected to the FLEXRAYCONFIG PDU TX block.

4 Connect the outports of the subsystem to appropriate Simulink blocks:

Port	Description
PayloadLengthStatus	Displays status information with payload length manipulation.

For more details on the ports, refer to FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮).

Result

You can manipulate the payload length of the TX PDU.

> **Tip**
>
> You can read the payload length of RX PDUs. The PDU mapping subsystem connected to the FLEXRAYCONFIG PDU RX block is given a PayloadLengthValue outport if the PDU is configured to read the payload length.

Related topics

References

FLEXRAYCONFIG PDU RX (FlexRay Configuration Blockset Reference 🕮)
FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 🕮)

How to Manipulate the Update Bit of a PDU

Objective

If an update bit of a PDU is specified in a FIBEX file or AUTOSAR system description file, you can manipulate its values. An update bit is a Boolean value which is true each time that the PDU is sent. It is false if the PDU is not sent but the frame containing it is sent.

Update bit manipulation via TRC file

You can also manipulate the update bit via the TRC file if the bit is configured accordingly. Refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

Preconditions

- The update bit of a PDU must be specified in the FIBEX file or AUTOSAR system description file.
- The FLEXRAYCONFIG PDU TX block of the PDU must be in the Simulink model (see How to Send or Receive Signals of PDUs on page 153).

Method

To manipulate the update bit of a PDU

1 Open the PDU mapping subsystem connected to the FLEXRAYCONFIG PDU TX block.

2 Connect the inports of the block to appropriate Simulink blocks.

Port	Description
UpdateBitEnable	Enables or disables the manual manipulation of the update bit.
UpdateBitValue	Sets the value of the update bit.

For more details on the ports, refer to FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮).

Result

You can set the update bit manually.

Related topics

References

FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 🕮)

Sending and Receiving Signal- and Signal-Group-Specific Update Bits

Introduction

Communication cluster files can contain signal-specific and signal-group-specific update bits. The update bits of signals/signal groups are set when the associated TX PDU is sent. You can evaluate the received signal/signal group update bits for RX PDUs.

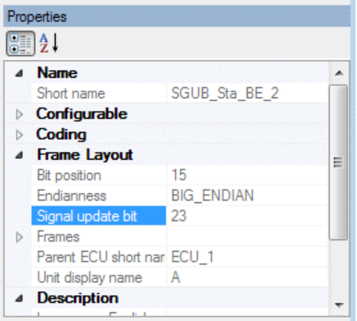
Sending update bits of signals/signal groups

The update bit of a signal/signal group is set to 1 each time that the TX PDU that the group belongs to is sent. The update bit value is 0 if the TX PDU is not sent.

Receiving update bits of signals/signal groups

The update bit of a signal/signal group is set to 1 each time that the TX PDU that the signal/signal group belongs to is sent. The update bit value is 0 if the TX PDU is not sent. You can evaluate the received signal/signal group update bits for RX PDUs. The value of a signal group update bit is assigned to the signals belonging to that signal group.

The FlexRay Configuration Tool displays the signal update bit positions for signals of TX or RX PDUs in the **Properties** view. The bit positions are specified in the communication cluster file.



Properties

Name	
Short name	SGUB_Sta_BE_2
Configurable	
Coding	
Frame Layout	
Bit position	15
Endianness	BIG_ENDIAN
Signal update bit	23
Frames	
Parent ECU short nar	ECU_1
Unit display name	A
Description	

The signal update bit values of signals of an RX PDU are part of the **Signals** bus of the **FLEXRAYCONFIG PDU RX** block. Refer to FLEXRAYCONFIG PDU RX Mapping Subsystems (FlexRay Configuration Blockset Reference 📖).

Signal update bit values can also be made available in the TRC file. If they are selected for the TRC file in the FlexRay Configuration Tool, the `<Signal_x>` groups within the `Receiving` and `Monitoring` groups contain the `Update Bit Value` variable. For further information, refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

Preconditions

Whether update bits of signals and/or signal groups can be received by RX PDUs depends on the database your FlexRay configuration is based on. The following table shows which signal- or signal-group-specific update bits the dSPACE FlexRay Configuration Package supports for the different database types:

Configuration Based On ...	Reception of Signal-Specific Update Bits	Reception of Signal-Group-Specific Update Bits
FIBEX 2.0 or earlier	–	–
FIBEX 3.0 or 3.1	✓	–
FIBEX 4.1, 4.1.1, or 4.1.2	✓	–
FIBEX+	✓	–
AUTOSAR 3.1.4, 3.2.1, 3.2.2, 4.0.3, 4.1.1, 4.1.2, 4.2.1, 4.2.2, 4.3.0, 4.3.1, 4.4.0, or AUTOSAR Classic Platform Release R19-11 or R20-11 system description file	✓	✓

Related topics

Basics

Using the Generated TRC File of PDU-Based Modeling in ControlDesk.................................... 189

References

FLEXRAYCONFIG PDU RX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮)

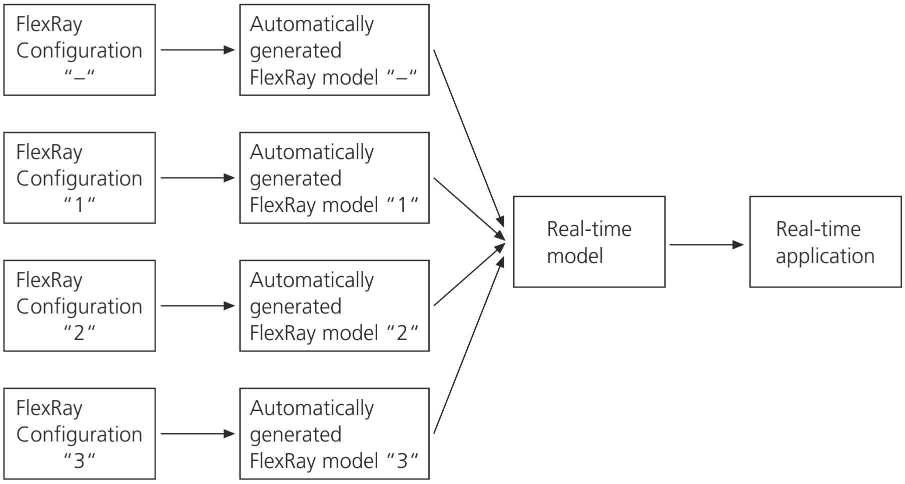
Modeling Several FlexRay Buses on One dSPACE Real-Time System

Introduction

You can use FLEXRAYCONFIG blocks from different FlexRay configurations in one real-time model, for example, to model a gateway.

Overview

The following illustration shows the principle of generating a real-time model with several FlexRay configurations. You can use up to four FlexRay configurations in one real-time model.


FlexRay Configuration "–"
FlexRay Configuration "1"
FlexRay Configuration "2"
FlexRay Configuration "3"
Automatically generated FlexRay model "–"
Automatically generated FlexRay model "1"
Automatically generated FlexRay model "2"
Automatically generated FlexRay model "3"
Real-time model
Real-time application

Creating configurations

You must create a FlexRay configuration for each FlexRay bus which you want to implement in your real-time model, using the FlexRay Configuration Tool. The FlexRay configurations are the basis for the automatically generated FlexRay models. A *configuration ID* is used to identify blocks from different FlexRay configurations. It must therefore be unique for each FlexRay bus. The configuration ID also specifies the priority of the FlexRay configurations. For details, refer to How to Create Configurations for Multiple Buses (FlexRay Configuration Tool Guide 🕮).

Modeling and configuring

Before you can start modeling and configuring, you must generate the FlexRay model for each FlexRay configuration. For details, refer to How to Generate Blocks for Modeling a FlexRay Communication on page 148.

After generating the FlexRay models (one for each configuration), you can use them to implement the FlexRay communication in your real-time model. You can configure parameters for controlling the FlexRay network via the **FlexRay** function block type in ConfigurationDesk. Refer to Building the Signal Chain for FlexRay Communication (ConfigurationDesk Real-Time Implementation Guide 🕮).

Note the following points:

- The configuration ID is added as read-only information beneath the block name of the configured blocks.
- It is recommended to always implement the FlexRay bus with configuration ID '-' in the real-time model.
- FlexRay configurations 1, 2, and 3 have a different synchronization task.
- The synchronization task of FlexRay configurations 1, 2, and 3 are driven by the cycle count start interrupt of the FlexRay controller CTR0.
- One **FLEXRAYCONFIG UPDATE** block is generated for each configuration.
- Only one CRC C file is allowed for all the FlexRay configurations used. However, you can implement different checksum algorithms in the file.

- The task belonging to FlexRay configuration '-' must have a higher priority than the task of the other configurations.The default task priorities are as follows:

Configuration ID	Task	Default Task Priority
'-'	Synchronization task	0
	Application/Com tasks	1
1	Cycle Start Interrupt (Sync-Task)	2
	Application/Com tasks	3
2	Cycle Start Interrupt (Sync-Task)	4
	Application/Com tasks	5
3	Cycle Start Interrupt (Sync-Task)	6
	Application/Com tasks	7

The default task priorities are sufficient in most use scenarios. However, you can configure the tasks and their priorities in ConfigurationDesk if required, for example, in the **Task Configurations** view. Refer to Modeling Executable Applications and Tasks (ConfigurationDesk Real-Time Implementation Guide 🕮).

Real-time simulation

In the real-time simulation, you want to observe the communication between the FlexRay buses. This can be done in ControlDesk via the Bus Navigator or via variables which are available in the TRC file.

Bus Navigator ControlDesk's Bus Navigator is a graphical environment for handling bus configurations, for example, FlexRay configurations. It has a Bus Navigator tree which lists all the elements of the FlexRay configurations. The tree has a FlexRay node for configuration '-' and FlexRay_*cn* nodes (*n* is the configuration ID) for configurations 1, 2, 3. Under these nodes, the FlexRay variables are sorted by ECU and channel. For details, refer to Introduction to the Bus Navigator (ControlDesk Bus Navigator 🕮).

TRC file A TRC file is generated for the simulation. The generated TRC file contains variables for controlling the PDUs, their signals and raw data. To visualize the variables, you can build a data connection with instruments in ControlDesk. If you have enabled the multiple bus option in the FlexRay Configuration Tool, configuration IDs are added to the structure of the TRC file:

```
FlexRay
   ConfigId -
      ...
   ConfigId 1
      ...
```

Below the `ConfigId` nodes, the structure is the same as for a single bus. For details, refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

Limitations of multiple bus support

The following limitations apply to multiple bus support (working with several configurations for one model):

- The FlexRay Configuration Tool supports a maximum of four configurations for one SCALEXIO Processing Unit. Only two configurations are supported for the MicroAutoBox III. The maximum number of possible configurations might be lower due to limited processing power.
- It cannot be guaranteed that the tasks of the lower-priority FlexRay configurations are executed at the scheduled times. This can lead to the following problems:
  - Temporary buffer lock error
  - Dynamic TX PDUs might be omitted.
  - Static TX PDUs contain old data or are sent as null frames.
- It is not possible to send static TX PDUs which share the same communication slot on the same channel via different FlexRay nodes or different bus controllers, even if their cycle counter filtering (defined by base cycle and cycle repetition in absolute scheduled timing) is different. If the controller of a FlexRay node is configured to send a static TX PDU within a specific communication slot with the given cycle counter filtering of the TX PDU, it automatically sends null frames in the slot each time the PDU is not sent. Any attempt by another bus node to send a static TX PDU in the slot results in invalid frames on the FlexRay bus.

  For further information, refer to *FlexRay Communications System Protocol Specification Version 2.1*.
- Only one XCP configuration is allowed for a real-time system.
- Only one CRC C file is allowed for all the FlexRay configurations used.

Related topics

References

FLEXRAYCONFIG UPDATE (FlexRay Configuration Blockset Reference 📖)

How to Switch the Transmission Mode of a PDU

Objective

If different transmission modes or timings are specified for PDUs, you can switch between them. Only one transmission mode can be active for a PDU at a time.

> **Tip**
>
> You can also switch the PDU transmission mode via the TRC file if it is configured accordingly. Refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

PDU transmission modes

The timing of PDUs is specified in the FIBEX or AUTOSAR system description file. A FIBEX or AUTOSAR system description file can contain different timings for one PDU. To create FlexRay configurations that use different timings for each PDU, the FlexRay Configuration Package provides transmission modes. The transmission modes are assigned to each PDU. Each transmission mode is assigned to a specific timing defined in the FIBEX or AUTOSAR system description file.

Besides the transmission modes with timings from the underlying FIBEX or AUTOSAR system description file, the FlexRay Configuration Package also provides transmission modes that allow you to create additional timings based on the corresponding LPDU timing.

For more information on configuring transmission modes, refer to How to Configure PDU Transmission Modes (FlexRay Configuration Tool Guide 🕮).

Preconditions

- Different transmission modes or timings of a PDU must be specified in the FIBEX or AUTOSAR system description file.
- The PDU must be configured for switching between its transmission modes during run time in the FlexRay Configuration Tool. Refer to How to Configure PDU Transmission Modes (FlexRay Configuration Tool Guide 🕮).
- The FLEXRAYCONFIG PDU TX block of the PDU must be in the Simulink model (see How to Send or Receive Signals of PDUs on page 153).

Method

To switch the transmission mode of a PDU

1 Open the PDU mapping subsystem connected to the FLEXRAYCONFIG PDU TX block.

2 Connect the inports of the block to appropriate Simulink blocks.

Port	Description
TransmissionModeSelector	Selects the transmission mode.
TxTrigger	Triggers event-based sending of the PDU. This port is available only in connection with event-based and mixed timings. If the value is equal to or greater than 1, the event-based timing is activated. Otherwise, cyclic timing is used (in the case of mixed timings).

For more details on the ports, refer to FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 🕮).

Result

You can switch the transmission mode of a PDU.

Related topics

HowTos

How to Configure PDU Transmission Modes (FlexRay Configuration Tool Guide 🕮)

References

FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 🕮)

How to Switch the Minimum Delay Time Support of a PDU

Objective

If minimum delay time information is specified for PDUs in an AUTOSAR system description file, you can enable or disable minimum delay time support for them.

> **Tip**
>
> You can also enable or disable minimum delay time support via the TRC file if it is configured accordingly. Refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

Minimum delay time support

Minimum delay time information is defined in the AUTOSAR system description file. This includes the definition of minimum delay time values for individual IPDUs and the ECU-specific definition on whether the defined minimum delay times are to be applied to event-based PDU transmissions only or also to cyclic PDU transmissions.

The minimum delay time of an IPDU specifies the minimum delay time between successive transmissions of this IPDU. It determines the time span (in seconds) that must elapse before new IPDU data can be packed into the LPDU, i.e., before a new transmission of this IPDU is possible.

For more information on configuring PDUs for minimum delay time support, refer to How to Configure PDUs for Minimum Delay Time Support (FlexRay Configuration Tool Guide 🕮).

Preconditions

- Minimum delay time information must be specified for the PDU in the AUTOSAR system description file.
- The minimum delay time feature must be enabled for the PDU in the FlexRay Configuration Tool. Refer to How to Configure PDUs for Minimum Delay Time Support (FlexRay Configuration Tool Guide 🕮).
- The **FLEXRAYCONFIG PDU TX** block of the PDU must be in the Simulink model (see How to Send or Receive Signals of PDUs on page 153).

Method

To switch the minimum delay time support of a PDU

1 Open the PDU mapping subsystem connected to the FLEXRAYCONFIG PDU TX block.

2 Connect the inports of the block to appropriate Simulink blocks.

Port	Description
MDTEnable	Enables or disables minimum delay time support.

For more information on the port, refer to FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 📖).

3 Open the Status mapping subsystem connected to the FLEXRAYCONFIG PDU TX block.

4 Connect the outports of the block to appropriate Simulink blocks.

Port	Description
MDTStatus	Displays information on the last captured trigger and on whether and how it was considered for triggering.
MDTTime	Displays the remaining minimum delay time.

For more information on the ports, refer to FLEXRAYCONFIG PDU TX Mapping Subsystems (FlexRay Configuration Blockset Reference 📖).

Result

You can enable or disable the minimum delay time support of a PDU and get status information on the minimum delay time feature.

Related topics

HowTos

How to Configure PDUs for Minimum Delay Time Support (FlexRay Configuration Tool Guide 📖)

References

FLEXRAYCONFIG PDU TX (FlexRay Configuration Blockset Reference 📖)

Configuring a FlexRay Network

Introduction

You can configure the FlexRay network using the FlexRay function block type in ConfigurationDesk.

Where to go from here

Information in this section

Functions for Configuring a FlexRay Network

Introduction

The **FlexRay** function block type in ConfigurationDesk lets you specify function ports and signal ports for configuring a FlexRay network.

FlexRay function block ports

The model interface of the **FlexRay** function block in ConfigurationDesk lets you enable different function ports for configuring a FlexRay network. The functions are divided into common functions and controller-specific functions.

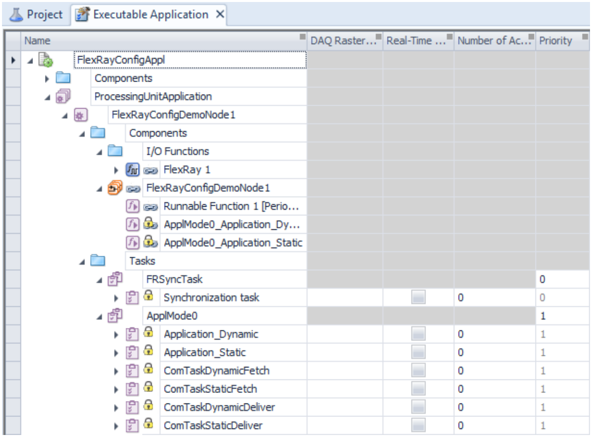
> **Note**
>
> After enabling ports and configuring parameters for controlling your FlexRay network, you have to generate suitable model port blocks, copy them to your Simulink model and connect them with your Simulink blocks. Refer to Generate Simulink Model Interface – All Unresolved Blocks – New Model (ConfigurationDesk User Interface Reference 🕮).

The electrical interface of the **FlexRay** function block contains signal ports which provide the interface to external devices. In addition, it lets you select the type of hardware resource to be used with the function block.

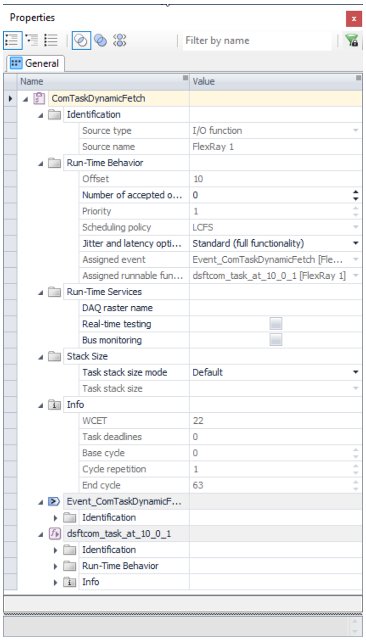
The available functions and ports of the FlexRay function block are listed below.

Task information

During the model analysis, ConfigurationDesk creates a Configuration Port block in the model topology for each identified **FLEXRAYCONFIG UPDATE** block in your Simulink model. The Configuration Port block is used to configure a **FlexRay** function block. After mapping the Configuration Port block to the **Configuration** function port of a **FlexRay** function block, you can view information on the task configuration in ConfigurationDesk's Executable Application table view and in the Properties Browser.



Name	DAQ Raster…	Real-Time …	Number of Ac…	Priority
FlexRayConfigAppl				
Components				
ProcessingUnitApplication				
FlexRayConfigDemoNode1				
Components				
I/O Functions				
FlexRay 1				
FlexRayConfigDemoNode1				
Runnable Function 1 [Perio…				
ApplMode0_Application_Dy…				
ApplMode0_Application_Static				
Tasks				
FRSyncTask				0
Synchronization task		☐	0	0
ApplMode0				1
Application_Dynamic		☐	0	1
Application_Static		☐	0	1
ComTaskDynamicFetch		☐	0	1
ComTaskStaticFetch		☐	0	1
ComTaskDynamicDeliver		☐	0	1
ComTaskStaticDeliver		☐	0	1



Name	Value
ComTaskDynamicFetch	
Identification	
Source type	I/O function
Source name	FlexRay 1
Run-Time Behavior	
Offset	10
Number of accepted o…	0
Priority	1
Scheduling policy	LCFS
Jitter and latency opti…	Standard (full functionality)
Assigned event	Event_ComTaskDynamicFetch [Fle…
Assigned runnable fun…	dsftcom_task_at_10_0_1 [FlexRay 1]
Run-Time Services	
DAQ raster name	
Real-time testing	☐
Bus monitoring	☐
Stack Size	
Task stack size mode	Default
Task stack size	
Info	
WCET	22
Task deadlines	0
Base cycle	0
Cycle repetition	1
End cycle	63
Event_ComTaskDynamicF…	
Identification	
dsftcom_task_at_10_0_1	
Identification	
Run-Time Behavior	
Info	

In addition, a FlexRay function block provides functions for you to handle the tasks of your FlexRay application. Refer to Common FlexRay Functions on page 178.

FlexRay functions

The model interface of the FlexRay function block type provides common and controller-specific FlexRay functions. For details, refer to the following topics:

- Common FlexRay Functions on page 178
- Controller-Specific FlexRay Functions on page 182

Global configuration parameters

The FlexRay function block provides global FlexRay parameters to configure your FlexRay network, such as Number of controllers used, Channels used, Start up synchronization mode, and configuration parameters for deadline violation handling. For details, refer to Common Function Block Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮).

Electrical interface

The electrical interface of the FlexRay function block contains signal ports which provide the interface to the external devices (e.g., ECU) and to the dSPACE real-time hardware. The function block offers parameters to configure the properties of these interfaces. The number of signal ports depends on the number of controllers, the specified Feedthrough mode, and the number of FlexRay channels to be used (A, B, A&B). The following signal ports are available for a controller:

Name	Description
FlexRay CTR<n>, channel A - terminated / not terminated	
FR CTR<n> A+	Channel A Bus Plus
FR CTR<n> A-	Channel A Bus Minus
FR CTR<n> GndA	Ground
FlexRay CTR<n>, channel A - feed-through	
FR CTR<n> A+	Channel A Bus Plus
FR CTR<n> A-	Channel A Bus Minus
FR CTR<n> A+ FT	Channel A Bus Plus fed through
FR CTR<n> A- FT	Channel A Bus Minus fed through
FlexRay CTR<n>, channel B (if activated) - terminated / not terminated	
FR CTR<n> B+	Channel B Bus Plus
FR CTR<n> B-	Channel B Bus Minus
FR CTR<n> GndB	Ground
FlexRay CTR<n>, channel B (if activated) - feed-through	
FR CTR<n> B+	Channel B Bus Plus
FR CTR<n> B-	Channel B Bus Minus
FR CTR<n> B+ FT	Channel B Bus Plus fed through
FR CTR<n> B- FT	Channel B Bus Minus fed through

You can configure the signal ports for failure simulation. For details, refer to Electrical Interface Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮).

Reference information

For details on the FlexRay function block type, refer to FlexRay (ConfigurationDesk Function Block Properties 🕮).

Related topics

Basics

FlexRay (ConfigurationDesk I/O Function Implementation Guide 📖)

Common FlexRay Functions

Introduction

The **FlexRay** function block type provides common FlexRay functions for configuring your FlexRay network.

Available functions

- Com Cyclic Control Chx on page 178
- Com Event Control Chx on page 179
- Status Chx on page 180
- Error Hook Status on page 180
- Memberships on page 181
- Synchronization Task on page 181
- Timetable Control on page 182

Com Cyclic Control Chx

Note

> The **FlexRay** function block type provides three **Com Cyclic Control** functions:
>
> - Com Cyclic Control ChA
> - Com Cyclic Control ChB
> - Com Cyclic Control ChAB
>
> The availability of the functions depends on the FlexRay channel(s) used for communication, which are either taken from the FIBEX or AUTOSAR system description file the project is based on or specified via the **Channels used** property (see Common Function Block Properties (FlexRay) (ConfigurationDesk Function Block Properties 📖)).

The **Com Cyclic Control Chx** function lets you control the execution of cyclic (static and dynamic) frames of the communication layer. The following function ports are available:

- **Enable operations** lets you enable static and dynamic cyclic transmissions and receptions.
- **Enable reception** lets you enable static and dynamic cyclic receptions.
- **Enable transmission** lets you enable static and dynamic cyclic transmissions.
- **Com state (Chx)**

  The **Com state** function port lets you display the state of the cyclic part of the communication layer for the channel (A or B). The following values are possible:

Value	Description
1 (On)	The cyclic part of the communication layer is currently being executed.
0 (Off)	The cyclic part of the communication layer is currently not being executed.

For more information, refer to:

- Ports and Properties of the Com Cyclic Control Chx Functions (FlexRay) (ConfigurationDesk I/O Function Implementation Guide 📖)
- Common Functions Properties (FlexRay) (ConfigurationDesk Function Block Properties 📖)

Com Event Control Chx

Note

The FlexRay function block type provides three common Com Event Control functions:

- Com Event Control ChA
- Com Event Control ChB
- Com Event Control ChAB

The availability of the functions depends on the FlexRay channel(s) used for communication, which are either taken from the FIBEX or AUTOSAR system description file the project is based on or specified via the Channels used property (see Common Function Block Properties (FlexRay) (ConfigurationDesk Function Block Properties 📖)).

The Com event control function lets you control the execution of dynamic event frames of the communication layer. The following function ports are available:

- Enable operations lets you enable dynamic event transmissions and receptions.
- Enable reception lets you enable dynamic event receptions.
- Enable transmission lets you enable dynamic event transmissions.
- Com state (Chx)

  The Com state function port lets you display the state of the event part of the communication layer for the channel (A or B). The following values are possible:

Value	Description
1 (On)	The event part of the communication layer is currently being executed.
0 (Off)	The event part of the communication layer is currently not being executed.

For more information, refer to:

- Ports and Properties of the Com Event Control Chx Port Functions (FlexRay) (ConfigurationDesk I/O Function Implementation Guide 📖)

- Common Functions Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮)

Status Chx

> **Note**
>
> The **FlexRay** function block type provides two common **Status** functions:
> - Status ChA
> - Status ChB
>
> The availability of the functions depends on the FlexRay channel(s) used for communication which are either taken from the FIBEX or AUTOSAR system description file the project is based on or specified via the **Used channels** property (see Common Function Block Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮)).

The **Status** function lets you enable the **Null frame counter** function port for you to display the number of received null frames that occurred during the communication for channel A or channel B. Received null frames are counted only if their ID is valid for the simulated FlexRay node. The value is incremented until the counter overflows. After an overflow, the counter starts at **0**.

For more information, refer to:

- Ports and Properties of the Common Functions Function Group (FlexRay) (ConfigurationDesk I/O Function Implementation Guide 🕮)
- Common Functions Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮)

Error Hook Status

The **Error Hook Status** function lets you track information on the FlexRay errors that occurred. The following function ports are available:

Last error type The **Last error type** provides information on the type of the last FlexRay error that occurred. The following values are possible:

Value	Description
0	No error hook detected
1	Buffer locked error
2	Buffer full error
3	Local buffer corrupted
4	Invalid frame error

Error counter The **Error counter** tracks the number of FlexRay errors that occurred.

For more information, refer to:

- Ports and Properties of the Common Functions Function Group (FlexRay) (ConfigurationDesk I/O Function Implementation Guide 🕮)
- Common Functions Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮)

Memberships

The Memberships function lets you enable or disable controllers/buffers of a specific membership. You can use the function to disable the communication of simulated ECUs if you want to simulate an ECU malfunction or replace them by the real ECUs. The number of enabled Membership n and Membership n State function ports depends on the number of memberships you configured in the FlexRay Configuration Tool. The following configuration parameters are displayed:

Related ECUs Displays the ECUs that are assigned to the membership.

Related controllers Displays the FlexRay controllers that are assigned to the membership.

> **Note**
>
> The Memberships function is only available if the following preconditions are fulfilled:
>
> - You must have configured memberships for your ECUs in the FlexRay Configuration Tool (see Disabling the Communication of Simulated ECUs (FlexRay Configuration Tool Guide 🕮)).
> - You must have mapped the Configuration function port of the FlexRay function block to the associated Configuration Port block (see FlexRay Bus Connection (SCALEXIO – Hardware and Software Overview 🕮) and Recommended Workflow for Implementing FlexRay Communication (ConfigurationDesk Real-Time Implementation Guide 🕮)).

For further information, refer to Membership <n> Function Properties (Memberships - FlexRay) (ConfigurationDesk Function Block Properties 🕮).

Synchronization Task

The Synchronization Task function lets you configure the behavior of the synchronization task. The following configuration parameters are available:

Synchronization limit Displays the value of the Synchronization limit parameter. It is calculated as follows:

```
Synchronization limit = x * Macrotick length
```

Synchronization threshold Displays the value of the Synchronization threshold parameter. It is calculated as follows:

```
Synchronization threshold = y * Macrotick length
```

If the difference between the local and the global time is less than the threshold value, the FlexRay host is set to the synchronized state. If the difference between local and global time is greater than the Synchronization limit parameter, the synchronization state is reset to unsynchronized.

In addition, the following function ports are available:

- The Enable sync service function port lets you enable the synchronization service.
- The Execute tasks with synchronization only function port lets you specify to execute tasks only if the host and the FlexRay cluster are synchronized.

- The **Enable task execution** function port allows you to start and stop tasks without stopping the simulation.
- The **Synchronization status** function port indicates the status of the synchronization.

For further information, refer to Synchronization Task Function Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮).

Timetable Control

The **Timetable Control** function lets you control the execution of a timetable task. The following function ports are available:

Timetable stop and restart This function port lets you stop the execution of a specific timetable for the current communication cycle. The timetable is restarted automatically at the beginning of the next communication cycle.

> **Note**
>
> Timetable stop and restart are triggered once when the value written to the port changes from 0 to 1.

Enable timetable This function port lets you enable or disable the execution of a specific timetable permanently.

For further information, refer to Timetable Control Function Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮).

Related topics

Basics

FlexRay (ConfigurationDesk I/O Function Implementation Guide 🕮)

Controller-Specific FlexRay Functions

Introduction

The **FlexRay** function block provides controller-specific FlexRay parameters and functions for configuring your FlexRay network.

Available parameters and functions

- Controller-specific global configuration parameters on page 183
- Reset on page 183
- Status on page 183
- Stop and Restart on page 185
- Wakeup Pattern Chx on page 186

Controller-specific global configuration parameters

The FlexRay function block provides the following global configuration parameters for a controller:

- Termination Chx and Feedthrough Chx let you specify to terminate, not terminate, or feed through bus lines of channel A or B.
- Inhibit cold start lets you switch off the cold start mode of a FlexRay controller.

In addition, the following read-only parameters are displayed:

- Membership ID
- Buffer count
- Used buffer
- Is used for synchronization
- Is used for startup

For further information, refer to CTR<n> Controller Functions (FlexRay) (ConfigurationDesk Function Block Properties 🕮).

Reset

The Reset function lets you reset a FlexRay communication controller. The following function ports are available:

Reset This function port lets you set the FlexRay controller to the configuration mode and restarts it afterwards. It allows you to restart or reintegrate a FlexRay controller which already entered the configuration mode or to start or reintegrate a running FlexRay controller.

> **Note**
>
> The reset is triggered once when the value written to the port changes from 0 to 1.

Cold start mode This function lets you set the *cold start inhibit* flag of the FlexRay controller.

For more information, refer to:

- Ports and Properties of the CTR<n> Function Groups (FlexRay) (ConfigurationDesk I/O Function Implementation Guide 🕮)
- CTR<n> Controller Functions (FlexRay) (ConfigurationDesk Function Block Properties 🕮)

Status

The Status function lets you monitor controller-specific information. The following function ports are available:

Protocol state This function port lets you monitor the current Protocol state. The following values are possible:

Value	Description
0	Configuration
1	Initialize schedule
2	Normal active operation
3	Normal passive operation
4	Integration consistency check
5	Integration listen
11	Wake up
21	Cold start listen
22	Integration cold start listen
23	Join cold start
24	Cold start collision resolution
25	Cold start consistency check
26	Coldstart gap
50	Default configuration
51	Ready
52	Halt
60	Wakeup standby
61	Wakeup listen
62	Wakeup send
63	Wakeup detect
70	Monitor mode
80	Startup prepare
81	Abort startup

Cycle counter (app. cycle) This function port lets you monitor the value of the FlexRay communication counter according to the application cycle. The communication counter starts with 0. The maximum value is calculated as follows:

```
maximum value =
APPLICATION_CYCLE_DURATION/COMMUNICATION_CYCLE_DURATION-1
```

Cycle counter (absolute) This function port lets you monitor the value of the FlexRay communication counter value according to the value of the CCCVR (current cycle counter value register) of the FlexRay controller. The communication counter starts with 0. It runs independently of the application cycle.

Error level This function port lets you monitor the error level of the FlexRay controller. The following values are possible:

Value	Description
0	Everything is OK (green).
1	Some minor problems have occurred (yellow).
2	There is a serious problem (red).

Wakeup state This function port lets you monitor the wakeup state of the related controller. The following values are possible:

Value	Description
0	Undefined
1	Received header
2	Received wakeup
3	Collision header
4	Collision wakeup
5	Unknown
6	Transmitted

Channel active state Chx This function port lets you monitor the state of channel A or B. The following values are possible:

Value	Description
0	Channel is inactive. No valid frame/PDU was received during the current and the previous execution of the status block.
1	Channel is active.

For more information, refer to:

- Ports and Properties of the CTR<n> Function Groups (FlexRay) (ConfigurationDesk I/O Function Implementation Guide 📖)
- CTR<n> Controller Functions (FlexRay) (ConfigurationDesk Function Block Properties 📖)

Stop and Restart

The Stop and Restart function lets you stop and restart a FlexRay communication controller. The following function ports are available:

Stop This function port lets you set the FlexRay controller to the configuration mode.

Restart This function port restarts or reintegrates the FlexRay controller.

> **Note**
>
> The stop and the restart are triggered once when the value written to the port changes from **0** to **1**.

Cold start mode This function port lets you set the *cold start inhibit* flag of the FlexRay controller.

For more information, refer to:

- Ports and Properties of the CTR<n> Function Groups (FlexRay) (ConfigurationDesk I/O Function Implementation Guide 🕮)
- CTR<n> Controller Functions (FlexRay) (ConfigurationDesk Function Block Properties 🕮)

Wakeup Pattern Chx

> **Note**
>
> The **FlexRay** function block type provides two **Wakeup Pattern** functions for each controller:
>
> - Wakeup Pattern ChA
> - Wakeup Pattern ChB
>
> The availability of the functions depends on the FlexRay channel(s) used for communication, which are either taken from the FIBEX or AUTOSAR system description file the project is based on or specified via the **Channels used** property (see Common Function Block Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮)).

The **Wakeup Pattern Chx** function lets you transmit the wakeup pattern via the related communication controller. It provides the following configuration parameters:

Name	Description
Use parameter from FIBEX file	Switch between the wakeup parameters as defined in the FIBEX or AUTOSAR system description file and manual settings
Wakeup count	Number of wakeup symbols
TX idle time	Wakeup symbol idle time
TX low time	Wakeup symbol low time

The following function ports are available:

Send This function port lets you initiate the transmission of the wakeup pattern on channel A or B.

For further information, refer to Wakeup Pattern Chx Function Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮).

Related topics

Basics

FlexRay (ConfigurationDesk I/O Function Implementation Guide 🕮)

Simulating FlexRay Networks

Introduction

When the design of your FlexRay network is finished, you can build the real-time application, download it to the SCALEXIO system or the MicroAutoBox III and execute it. The TRC file generated by the FlexRay Configuration Tool contains the PDU's variables, signals and raw data which are selected for control via a TRC file. To visualize them during the execution of the real-time application, you can build a data connection with ControlDesk instruments.

Where to go from here

Information in this section

Information in other sections

Building Real-Time Applications (ConfigurationDesk Real-Time Implementation Guide 🕮)

Preventing a Deadline Violation

Deadline violation (DLV)

If the worst-case execution time of the application task or of the communication task directly before the synchronization task code is exceeded, a deadline violation (DLV) occurs. This also occurs if the synchronization task starts and the application or communication task is still running. A deadline violation leads to a warning or to program termination depending on the configuration of the deadline violation handling in the **FlexRay** function block (refer to Common Function Block Properties (FlexRay) (ConfigurationDesk Function Block Properties 🕮)).

Preventing a DLV

To prevent a deadline violation, you can specify a new worst-case execution time in the FlexRay Configuration Tool.

- For manually created tasks, you can specify larger values for the WCET.
- For automatic generated tasks, you can increase the **WCET adjustment factor**.

Related topics

Basics

Tunable Properties of Tasks (FlexRay Configuration Tool Guide 🕮)

Using the Generated TRC File of PDU-Based Modeling in ControlDesk

Introduction

The TRC file generated by the FlexRay Configuration Tool contains the PDU variables, signals, and raw data that are selected for control via a TRC file. To visualize the variables, you can establish a data connection with ControlDesk instruments. After you built and downloaded the real-time application, all signals from the TRC file are displayed in the **Variables** controlbar in ControlDesk.

Structure of the TRC file

The variables are structured in groups as follows ('<>' marks placeholders):

```
FlexRay
   ConfigId <Config Id>   (optional)
      Monitoring
         <ECU_Name>
            <Channel>
               <PDU_Name>
                 Signals
                    <Signal_1>
                       ...
                    <Signal_n>
                 CRC Data
                 Update Bit
                 Status
                 Raw Data
                 Update Contained PDU
                 Secure Onboard Communication
                 Global Time Synchronization
      Receiving
         <ECU_Name>
            <Channel>
               <PDU_Name>
                 Signals
                    <Signal_1>
                    ...
                    <Signal_n>
                 CRC Data
                 Update Bit
                 Status
                 Raw Data
                 Update Contained PDU
                 Secure Onboard Communication
                 Global Time Synchronization
      Sending
         <ECU_Name>
            <Channel>
               <PDU_Name>
                 Signals
                    <Signal_1>
                    ...
                    <Signal_n>
                 Enable
                 Trigger
                 CRC Data
                 Update Bit
                 Raw Data
                 Transmission Mode
                 Minimum Delay Time
                 Contained PDU Send Status
                 Secure Onboard Communication
                 Global Time Synchronization
```

The group names are derived from the short names of the elements which are specified in the database (FIBEX file or AUTOSAR system description file). In some cases, the group names differ, see TRC File Containing Several PDUs with the Same Short Name on page 207.

The `ConfigId <Config Id>` group is optional (`<ConfigId>` is '-', 1, 2, or 3). The group is generated if you use the multiple bus option. For details, refer to Modeling Several FlexRay Buses on One dSPACE Real-Time System on page 168.

The variables that are included in the groups are described below.

Sending group

The `Sending` group contains all the variables to control the TX PDUs and their signals. PDUs that are included in the TRC file must be selected in the FlexRay Configuration Tool, see How to Prepare the Manipulation or Monitoring of Frames/PDUs and Signals via TRC File (FlexRay Configuration Tool Guide 🕮).

<Signal_x> group A `<Signal_x>` group contains the variables to control one signal of the PDU.

Variable	Description
Value	Value for the signal to be sent. This variable is optional. To generate it, you must have selected the Signal Value Access feature.
Dynamic Value	Value to be sent during dynamic signal manipulation[1)]. The value must be specified as the coded data type. This variable is optional. To generate it, you must have selected the Signal Value Access feature.
Physical Value	Physical value for the signal to be sent. This variable is optional. To generate it, you must have selected the Signal Value Access feature and enabled the Use physical data type of the TRC File Generator, and conversion must be defined in the FIBEX file or AUTOSAR system description file.
Dynamic Physical Value	Value to be sent during dynamic signal manipulation[1)], specified as the physical data type. This variable is optional. To generate it, you must have selected the Signal Value Access feature and enabled the Use physical data type of the TRC File Generator, and conversion must be defined in the FIBEX file or AUTOSAR system description file.
Coded Physical Value Source Switch	To specify whether the coded or physical signal value is to be sent. The following values are possible: ▪ 0: Coded signal value is sent: i.e., a text table value is sent via the `Value` variable. ▪ 1: Physical signal value is sent: i.e., the `Physical Value` variable is used to sent the signal value.. This variable is optional. It is available for signals with the SCALE_LINEAR_TEXTTABLE computation method only. To generate it, the following conditions must be met: ▪ You must have selected the Signal Value Access feature. ▪ You must have enabled the Use physical data type and Activate support for text tables options of the Trace File Generator. ▪ The Physical data type is usable property must be set to True. ▪ The Physical data type conversion layer option must be set to COMMUNICATION on the General page of the General Properties dialog.
Validity	To specify the validity status for the signal to be sent over the bus: ▪ 0: NOT VALID ▪ 1: VALID ▪ 2: ERROR ▪ 4: NOT AVAILABLE

Variable	Description
	▪ 8: NOT DEFINED ▪ 16: OTHER The variable is essential only if data to be sent is specified in ControlDesk (`Source Switch = 1`). This variable is optional. To generate it, you must have selected the Signal Validity Control feature. This is possible only if a non-valid value (coded and/or physical) exists for the respective signal: i.e., if at least one value with validity ≠ 'VALID' is defined in the database file for the signal.
Dynamic Validity	To specify the validity status for the signal to be sent during dynamic signal manipulation[1)]: ▪ 0: NOT VALID ▪ 1: VALID ▪ 2: ERROR ▪ 4: NOT AVAILABLE ▪ 8: NOT DEFINED ▪ 16: OTHER This variable is optional. To generate it, you must have selected the Signal Validity Control feature. This is possible only if a value (coded and/or physical) with validity status ≠ 'VALID' exists for the respective signal.
Countdown Value	Time span for the dynamic signal manipulation[1)]. It specifies how often the dynamic values must be sent.
Source Switch	To switch the source for the send signal. ▪ 0: Data of the Simulink model is sent ▪ 1: Data which is specified via the TRC file is sent ▪ 2: The current value of the alive counter is sent ▪ 12: The alive counter is stopped for the time span specified for dynamic signal manipulation (`Countdown Value`). The last value of the alive counter is sent. When `Countdown Value` reaches 0, the source is switched back to the last `Source Switch` value.
AliveCounter	The `AliveCounter` group is added if you specified a signal as an alive counter in the FlexRay Configuration Tool and activated the Signal Alive Counter Control feature in the Element Selection dialog. The group contains the following variables: ▪ `Value`: Current value of the alive counter. ▪ `Runtime Behavior`: To specify the behavior of the alive counter at run time. ▪ 0: The alive counter runs independently of the selected source for the signal. **Note** There is one exception: If `Source Switch` is 12, the alive counter stops independently of the `Runtime Behavior`. ▪ 1: The alive counter runs only if its value is really sent. If the source for the signal is switched to another source than the alive counter, for example, to SL (a value from the Simulink model), the alive counter stops. ▪ 2: The alive counter stops at the current value. ▪ `Offset`: Offset value that is added to the alive counter during run time.

Variable	Description
Tx inspect	The `Tx inspect` group is added if the **Physical data type conversion layer** option is set to **COMMUNICATION** on the **General** page of the **General Properties** dialog in the FlexRay Configuration Tool. The group can contain the following variables: ▪ `Tx Value` or `Tx Physical Value` The value shows the transmitted coded or physical value of a signal via TRC file. A signal is sent as the physical data type if it has a usable physical data type and the **Use physical data type** option is set to True. The option is set for the Trace File Generator on the **Generators** page of the **General Properties** dialog in the FlexRay Configuration Tool. ▪ `Available SL Value` or `Available SL Physical Value` The value shows the coded or physical value that is set in the Simulink model for the respective signal. The **Port data type** property of the signal specifies the data type which is used in the Simulink model.

[1] For details on dynamic signal manipulation, refer to Dynamic Signal Manipulation on page 210.

Enable group The variables of the `Enable` group allow you to control the sending of cyclic PDUs.

The following variables are available for *dynamic* PDUs if you selected them for the **Frame Dynamic Control** feature:

Variable	Description
Tx Enable	To enable the sending of dynamic cyclic PDUs.
Multiplexer Switch Code	To select the switch code of a sub-PDU. It is available only for dynamic PDUs that contain sub-PDUs. You can only enter valid values which are documented in the variable's description. Switch codes are available only for sub-PDUs with cyclic timing. If a sub-PDU does not have its own cyclic timing, it is handled as a dynamic event sub-PDU.
Source Switch	To switch the source for the TX PDU. ▪ 0: Data of the Simulink model is sent ▪ 1: Data which is specified via the TRC file is sent

The following variables are available for *static* PDUs if you selected them for the **Frame Static Control** feature:

Variable	Description
SW Enable	To enable the sending of static PDUs. It is only available for static PDUs. If a static PDU is *enabled* via software (SWEnable = 1), payload data or null frames can be sent. The data which is sent depends on the configuration of SWEnable (see General Page (FlexRay Configuration Tool Reference 🕮)) and on the settings of the CHI Code Generator (see Generators Page (FlexRay Configuration Tool Reference 🕮)): ▪ The **SW Enable Configuration** property in the **General Properties** dialog is set to **Control of L-PDU commit to FlexRay buffer** and the **Static TX buffer transmission mode** property of the CHI Code Generator is set to **Event (null frame used)**: ▪ A null frames is sent if the LPDU to be sent contains only PDUs that have not been updated and that do not contain a PDU update bit.

Variable	Description
	▪ Payload data is sent for all other LPDUs. ▪ Other settings: ▪ Payload data is sent. If a static PDU is *disabled* via software (SWEnable = 0), old data or null frames can be sent. The data which is sent depends on the database version, on the configuration of SWEnable (see General Page (FlexRay Configuration Tool Reference 🕮), and on the settings of the CHI Code Generator (see Generators Page (FlexRay Configuration Tool Reference 🕮)): ▪ FIBEX version ≤ 2.0: The **SW Enable Configuration** property in the **General Properties** dialog is always set to **Control of L-PDU commit to FlexRay buffer**. You cannot change this setting. Null frames are sent if the **Static TX buffer transmission mode** property of the CHI Code Generator is **Event (null frame used)**. Old data is sent if the **Static TX buffer transmission mode** property is set to **State (old value used)**. ▪ FIBEX+, FIBEX 3.0, FIBEX 3.1, FIBEX 4.1.x, or AUTOSAR System Template: If the **SW Enable Configuration** property in the **General Properties** dialog is set to **Control of L-PDU commit to FlexRay buffer**: ▪ Null frames are sent if the **Static TX buffer transmission mode** property of the CHI Code Generator is set to **Event (null frame used)**. ▪ Old data is sent if the **Static TX buffer transmission mode** property is set to **State (old value used)**. The update bit has the value that was set before SWEnable was set to 0. If the **SW Enable Configuration** property in the **General Properties** dialog is set to **Control of I-PDU payload data update**: ▪ The update bit is set to 0 and old data is sent, independently of the settings of the **Static TX buffer transmission mode** property. If the **Static TX buffer transmission mode** property is set to **Event (null frame used)**, note the following information: **Note** In some cases, the `SW Enable` variable is overruled. This can happen if the update bit of a PDU is manipulable (see `Update Bit` group below). If the `Update Bit Enable` variable is 1, automatic update bit calculation is disabled. The update bit of the PDU is set to the value specified by the `Update Bit Value` variable. If `SW Enable` is 0 and the update bit value is 1, the PDU still sends old data. If the `Update Bit Enable` variable is not used, `SW Enable` is used to enable or disable the sending of static TX PDUs.
Multiplexer Switch Code	To select the switch code of a sub-PDU. It is available only for static PDUs that contain sub-PDUs. You can only enter valid values which are documented in the variable's description.
Source Switch	To switch the source for the TX PDU. ▪ 0: Data of the Simulink model is sent ▪ 1: Data which is specified via the TRC file is sent

Trigger group The variables of the `Trigger` group control the sending of dynamic event PDUs. The following variables are available if you selected a PDU for the Frame Dynamic Control feature:

Variable	Description
Tx Trigger	To trigger the dynamic PDU.
Multiplexer Switch Code	To select the switch code of a sub-PDU. It is available only for dynamic PDUs that contains sub-PDUs. You can only enter valid values which are documented in the variable's description. The switch codes in the description are different for event and cyclic sub-PDUs.
Source Switch	To switch the source for the TX PDU. ▪ 0: Data of the Simulink model is sent ▪ 1: Data which is specified via the TRC file is sent

CRC Data group The variables of the `CRC Data` group control the CRC calculation of TX PDUs. The following variables are available if you selected a PDU for the Frame CRC Control feature:

Variable	Description
CRC Enable	To enable or disable the CRC calculation of the PDU. ▪ 0: CRC calculation is disabled ▪ 1: CRC calculation is enabled
Dynamic CRC Enable	To enable or disable the CRC calculation of the PDU during dynamic checksum calculation manipulation.[1]
Type	To select the algorithm that is used for CRC calculation. The CRC algorithms which are specified in the FlexRay Configuration Tool are documented in the variable's description. For details, refer to Using User-Defined Checksum Algorithms (FlexRay Configuration Tool Guide 🕮).
Dynamic Type	To select the algorithm that is used for CRC calculation during dynamic checksum calculation manipulation.[1]
Countdown Value	Number of transmission cycles for which dynamic checksum calculation manipulation is performed.[1]
Source Switch	To switch the source for the TX PDU. ▪ 0: Data of the Simulink model is used for checksum calculation manipulation ▪ 1: Data which is specified via the TRC file is used for checksum calculation manipulation ▪ 8: For the specified number of transmission cycles (`Countdown Value`), the dynamic CRC manipulation data specified via the TRC file is used for checksum calculation manipulation. When `Countdown Value` reaches 0, the source is automatically switched back to the switch state that was set before.

[1] For further information, refer to Dynamic Checksum Calculation Manipulation on page 212.

Update Bit group The variables of the `Update Bit` group manipulate the update bit of a PDU. The following variables are available if you selected a PDU for the Frame Update Bit Control feature:

Variable	Description
Value	Value of the update bit
Dynamic Value	Update bit value to be sent during dynamic update bit manipulation[1)]
Update Bit Enable	To enable or disable automatic calculation of the update bit in the FTCom task when the `Source Switch` variable is 1: ▪ 0: Automatic calculation is enabled. This overwrites the value specified by the `Value` variable. ▪ 1: Automatic calculation is disabled. The value specified by the `Value` variable is used.
Dynamic Update Bit Enable	To enable or disable automatic update bit calculation during dynamic update bit manipulation.[1)]
Countdown Value	Number of transmission cycles for which dynamic update bit manipulation is performed.[1)]
Source Switch	To switch the source for the TX PDU. ▪ 0: Data of the Simulink model is used for update bit manipulation ▪ 1: Data which is specified via the TRC file (`Value` variable) is used for update bit manipulation ▪ 8: For the specified number of transmission cycles (`Countdown Value`), the dynamic update bit manipulation data specified via the TRC file is used for update bit manipulation. When `Countdown Value` reaches 0, the source is automatically switched back to the switch state that was set before.

[1)] For further information, refer to Dynamic Update Bit Manipulation on page 211.

Raw Data group The variables of the `Raw Data` group manipulate raw data of a PDU. The following variables are available if you selected a PDU for the Frame Raw Data Access feature:

Variable	Description
Tx Bytes	Raw data to be sent.
Raw Data Enable	To select raw data which is sent. ▪ 0: Raw data specified in the Simulink model is sent[1)]. If it is not available, signal data specified using the TRC file (if the `Source Switch` variable is not 0) or signal data specified in the Simulink model (if the `Source Switch` variable is 0) is sent. If signal data is not available, the last sent data or the initial values are sent. ▪ 1: Raw data specified by the `Tx Bytes` variable is sent The variable is effective only if the `Source Switch` variable is not 0.
Source Switch	To switch the source for the TX PDU. ▪ 0: Data of the Simulink model is sent ▪ 1: Data which is specified via the TRC file (`Tx Bytes` variable) is sent

[1)] As a precondition, raw data transmission must be activated in Simulink.

Transmission Mode group The variables of the `Transmission Mode` group control the transmission mode of a PDU. The following variables are available if you have selected a PDU for the Frame Dynamic Control or Frame Static Control (only PDU concept) feature:

Variable	Description
Value	Transmission mode to be used. ▪ 0: Transmission mode 'False'

Variable	Description
	▪ 1: Transmission mode 'True' ▪ 98: Transmission Mode 'LPDU timing triggered' ▪ 99: Transmission Mode 'User-Defined' For more information on PDU transmission modes, refer to How to Configure PDU Transmission Modes (FlexRay Configuration Tool Guide 🕮).
Source Switch	To switch the source for the TX PDU ▪ 0: Data of the Simulink model is sent ▪ 1: Data which is specified via the TRC file (`Value` variable) is sent

Minimum Delay Time group The variables of the `Minimum Delay Time` group allow you to control the minimum delay time feature of TX PDUs. The following variables are available if you have selected a PDU for the **Frame Minimum Delay Time Control** feature:

Variable	Description
Enable MDT	To enable or disable minimum delay time support for the PDU: ▪ 0: The minimum delay time feature is disabled. There is no delay time between successive transmissions of the PDU. ▪ 1: The minimum delay time feature is enabled. The minimum delay time specified for the PDU in the AUTOSAR system description file must be complied with between two transmissions of the PDU. For more information on minimum delay time support, refer to How to Configure PDUs for Minimum Delay Time Support (FlexRay Configuration Tool Guide 🕮).
Source Switch	To switch the source for the TX PDU: ▪ 0: Data of the Simulink model is used to enable or disable the minimum delay time feature. ▪ 1: Data which is specified via the TRC file (`Enable MDT` variable) is used to enable or disable the minimum delay time feature.
MDT Triggering Status	Information on the last captured trigger and on whether and how the trigger was considered for triggering (if the minimum delay time feature is enabled for the PDU). ▪ 0: No recent trigger. ▪ 1: Accepted event trigger. The event-based triggering occurred outside of an active minimum delay time. ▪ 2: Accepted cyclic trigger. The cyclic triggering occurred outside of an active minimum delay time. ▪ 3: Accepted event trigger, discarded cyclic trigger. The simultaneous event-based and cyclic triggering occurred outside of an active minimum delay time. ▪ 4: Discarded event trigger. The event-based triggering occurred during the active minimum delay time. ▪ 8: Discarded cyclic trigger. The cyclic triggering occurred during the active minimum delay time.

Variable	Description
	▪ 12: Discarded event trigger, discarded cyclic trigger. The simultaneous event-based and cyclic triggering occurred during the active minimum delay time.
Time	Remaining minimum delay time.

Contained PDU Send Status The variables of the `Contained PDU Send Status` group control the sending of contained IPDUs. The following variables are available if you have selected a contained IPDU for the Frame Container Control feature:

Variable	Description
Contained PDU Send Status	To display the send status of the contained IPDU and the associated container: ▪ 0: Contained IPDU was not triggered. ▪ 1: Contained IPDU was triggered but not added to the container IPDU yet. ▪ 2: Contained IPDU was triggered and packed into the container IPDU. ▪ 4: Container IPDU (and thus the contained IPDU) was sent.
Container PDU Trigger Status	To display the trigger status of the container IPDU: ▪ 0: Container IPDU was not triggered. ▪ 1: Container IPDU was triggered because the threshold was exceeded. ▪ 2: Container IPDU was triggered because it was full. ▪ 4: Container IPDU was triggered by the first added contained IPDU (`FirstContainedTrigger`). ▪ 8: Container IPDU was triggered because this contained IPDU was added to the container IPDU (`TRIGGER_ALWAYS`) ▪ 16: Container IPDU was triggered by the container IPDU's timeout. For container IPDUs with static container layout, only 0 and 8 are valid trigger status values. This is because a static container IPDU can only be triggered by adding a contained IPDU (`TRIGGER_ALWAYS`).

For more information on container IPDUs and contained IPDUs, refer to Aspects of Miscellaneous Supported AUTOSAR Features (FlexRay Configuration Tool Guide 📖).

Secure Onboard Communication group The variables of the `Secure Onboard Communication` group control the sending of secured IPDUs. The following variables are available if you have selected an authentic IPDU for the Frame Authentication Control feature:

Variable	Description
Authenticator Value	The `Authenticator Value` group contains the following groups and variables: ▪ The `Control` group contains the following variables: ▪ `Type`: To invalidate the authenticator of a secured IPDU. The meaning of this value is user-code-dependent. Proposed values are: ▪ 0: Do not invalidate the authenticator ▪ 1: Invalidate the authenticator This variable corresponds to the `AuthenticationType` user code variable.
Tx Inspect	The `Tx Inspect` group contains the following variables: ▪ `Authenticator Value`: The value shows the actually transmitted (truncated) authenticator value. This variable does not correspond to any user code variable.

Variable	Description
	▪ `Freshness Value`: The value shows the actually transmitted (truncated) freshness value. This variable does not correspond to any user code variable.

For more information on secure onboard communication, refer to Aspects of Miscellaneous Supported AUTOSAR Features (FlexRay Configuration Tool Guide 🕮).

Global Time Synchronization group The variables of the `Global Time Synchronization` group let you control the sending of time synchronization messages. The following variables are available if you have selected frames for the Frame Global Time Sync Access feature:

Variable	Description
CRC	The `CRC` group contains the following variables: ▪ `Value`: CRC value for the time synchronization message to be sent. ▪ `Dynamic Value`: CRC value to be sent during dynamic global time synchronization manipulation.[1] ▪ `Countdown Value`: Number of transmission cycles for which dynamic CRC manipulation is performed.[1] ▪ `Source Switch`: To switch the source for the TX PDU: ▪ 0: Data of a fixed E2E_P02 algorithm is sent. ▪ 1: Data which is specified via the TRC file is sent. ▪ 8: For the specified number of transmission cycles (`Countdown Value`), the dynamic CRC manipulation data specified via the TRC file is used for GTS manipulation. When `Countdown Value` reaches 0, the source is automatically switched back to the switch state that was set before
E2E Sequence Counter	The `E2E Sequence Counter` group contains the following variables: ▪ `Value`: Value of the end-to-end protection sequence counter (SC). ▪ `Dynamic Value`: End-to-end protection sequence counter (SC) value to be sent during dynamic global time synchronization manipulation.[1] ▪ `Countdown Value`: Number of transmission cycles for which dynamic E2E sequence counter manipulation is performed.[1] ▪ `Source Switch`: To switch the source for the TX PDU: ▪ 0: Data of a fixed E2E_P02 algorithm is sent. ▪ 1: Data specified via the TRC file is sent. ▪ 8: For the specified number of transmission cycles (`Countdown Value`), the dynamic E2E sequence counter manipulation data specified via the TRC file is used for GTS manipulation. When `Countdown Value` reaches 0, the source is automatically switched back to the switch state that was set before
Time Gateway Synchronization Status	The `Time Gateway Synchronization Status` group contains the following variables: ▪ `Value`: Value of the SYNC_TO_GATEWAY (SGW) bit from the Time Base status of the time base manager instance. ▪ `Dynamic Value`: Value of the SYNC_TO_GATEWAY (SGW) bit to be sent during dynamic global time synchronization manipulation.[1] ▪ `Countdown Value`: Number of transmission cycles for which dynamic time gateway synchronization manipulation is performed.[1] ▪ `Source Switch`: To switch the source for the TX PDU: ▪ 0: Data of the time base manager is sent. ▪ 1: Data specified via the TRC file is sent.

Variable	Description
	▪ 8: For the specified number of transmission cycles (`Countdown Value`), the manipulation data of the time gateway synchronization status specified via the TRC file is used for GTS manipulation. When `Countdown Value` reaches 0, the source is automatically switched back to the switch state that was set before
User Byte 0	The `User Byte 0` group contains the following variables: ▪ `Value`: Value of the User Byte 0. ▪ `Dynamic Value`: User Byte 0 value to be sent during dynamic global time synchronization manipulation.[1] ▪ `Countdown Value`: Number of transmission cycles for which dynamic user byte 0 manipulation is performed.[1] ▪ `Source Switch`: To switch the source for the TX PDU: ▪ 0: Data of the time base manager is sent. ▪ 1: Data specified via the TRC file is sent. ▪ 8: For the specified number of transmission cycles (`Countdown Value`), the dynamic user byte 0 manipulation data specified via the TRC file is used. When `Countdown Value` reaches 0, the source is automatically switched back to the switch state that was set before
User Byte 1	The `User Byte 1` group contains the following variables: ▪ `Value`: Value of the User Byte 1. ▪ `Dynamic Value`: User Byte 1 value to be sent during dynamic global time synchronization manipulation.[1] ▪ `Countdown Value`: Number of transmission cycles for which dynamic user byte 1 manipulation is performed.[1] ▪ `Source Switch`: To switch the source for the TX PDU: ▪ 0: Data of the time base manager is sent. ▪ 1: Data specified via the TRC file is sent. ▪ 8: For the specified number of transmission cycles (`Countdown Value`), the dynamic user byte 1 manipulation data specified via the TRC file is used. When `Countdown Value` reaches 0, the source is automatically switched back to the switch state that was set before
Total Time	The `Total Time` group contains the following variables: ▪ `Value`: Value of the total time. ▪ `Dynamic Value`: Total time value to be sent during dynamic global time synchronization manipulation.[1] ▪ `Countdown Value`: Number of transmission cycles for which dynamic total time manipulation is performed.[1] ▪ `Source Switch`: To switch the source for the TX PDU: ▪ 0: Data of the time base manager is sent. ▪ 1: Data specified via the TRC file is sent. ▪ 8: For the specified number of transmission cycles (`Countdown Value`), the dynamic total time manipulation data specified via the TRC file is used for GTS manipulation. When `Countdown Value` reaches 0, the source is automatically switched back to the switch state that was set before

[1] For more information, refer to Dynamic Manipulation of GTS Communication on page 214.

Sent Values The `Sent Values` group contains the following variables showing the actually transmitted GTS-relevant values:

Variable	Description
CRC	Actually transmitted CRC value for the time synchronization message.
Time Domain Id	Identifier of the global time domain that is transferred in the time synchronization message.
E2E Sequence Counter	Actually transmitted value of the end-to-end protection sequence counter (SC).
FlexRay Cycle Counter	Value of the FlexRay cycle counter (FCNT) at the time the global time synchronization PDU is built.
Time Gateway Synchronization Status	Value of the SYNC_TO_GATEWAY (SGW) bit from the Time Base status of the time base manager instance.
User Byte 0	Actually transmitted value of the User Byte 0.
User Byte 1	Actually transmitted value of the User Byte 1.
Seconds	Displays the seconds that are transferred via the FlexRay bus in the time synchronization message.
Nanoseconds	Displays the nanoseconds that are transferred via the FlexRay bus in the time synchronization message.
Total Time	Actually transmitted value of the total time.

For more information on global time synchronization, refer to Aspects of Miscellaneous Supported AUTOSAR Features (FlexRay Configuration Tool Guide 📖).

Receiving group and Monitoring group

The `Receiving` and `Monitoring` groups contain all the variables needed to receive or monitor RX PDUs. PDUs that are included in the TRC file must be selected in the FlexRay Configuration Tool, see How to Prepare the Manipulation or Monitoring of Frames/PDUs and Signals via TRC File (FlexRay Configuration Tool Guide 📖).

<Signal_x> group A `<Signal_x>` group contains the variables to read one signal of the PDU.

Variable	Description
Value	Value of the received or monitored signal This variable is optional. To generate it, you must have selected the **Signal Value Access** feature in the FlexRay Configuration Tool.
Physical Value	Physical value of the received or monitored signal. This variable is optional. To generate it, you must have selected the **Signal Value Access** feature and enabled the **Use physical data type** of the TRC File Generator, and conversion must be defined in the FIBEX file or AUTOSAR system description file.
Text Table Status	Indicates whether the received value is a text table value. ▪ 0: The received value is not a text table value. The received value can be read via the `Physical Value` variable. ▪ 1: The received value is a text table value. The received value can be read via the `Value` variable.

Variable	Description
	This variable is optional. It is available for signals with the SCALE_LINEAR_TEXTTABLE computation method only. To generate it, the following conditions must be met: - You must have selected the **Signal Value Access** feature. - You must have enabled the **Use physical data type** and **Activate support for text tables** options of the Trace File Generator. - The **Physical data type is usable** property must be set to True.
Status	Status information of the received or monitored signal: - 0: No error, signal is valid - 1: Access error - 2: Signal is not received - 4: Signal is not valid (signal validity status ≠ 'VALID') - 8: CRC is incorrect It is available only if the signal is selected for the **Signal RX Status Access** feature in the FlexRay Configuration Tool (see How to Prepare the Manipulation or Monitoring of Frames/PDUs and Signals via TRC File (FlexRay Configuration Tool Guide 🕮))
Update Bit Value	Signal update bit value of the received or monitored signal. For further information, refer to Sending and Receiving Signal- and Signal-Group-Specific Update Bits on page 167. This variable is optional. To generate it, you must have selected the **Signal Value Access** feature.

CRC Data group The variables of the `CRC Data` group control the CRC calculation of RX PDUs. The following variables are available when you have selected a PDU for the **Frame CRC Control** feature:

Variable	Description
CRC Enable	To enable or disable the CRC calculation of the PDU. - 0: CRC calculation is disabled - 1: CRC calculation is enabled
Dynamic CRC Enable	To enable or disable the CRC calculation of the PDU during dynamic checksum calculation manipulation.[1)]
Type	To select the algorithm that is used for CRC calculation. The CRC algorithms which are specified in the FlexRay Configuration Tool are documented in the variable's description. For details, refer to Using User-Defined Checksum Algorithms (FlexRay Configuration Tool Guide 🕮).
Dynamic Type	To select the algorithm that is used for CRC calculation during dynamic checksum calculation manipulation.[1)]
Countdown Value	Number of transmission cycles for which dynamic checksum calculation manipulation is performed.[1)]
Status	To get the status of the CRC calculation. - 0: CRC calculation OK - 1: CRC calculation not OK - 2: CRC not calculated.
Source Switch	To switch the source for the RX PDU. - 0: Data of the Simulink model is used - 1: Data which is specified via the TRC file is used

Variable	Description
	▪ 8: Dynamic CRC manipulation data specified via the TRC file is used for the specified number of transmission cycles (`Countdown Value`). When `Countdown Value` reaches 0, the source is automatically switched back to the switch state that was set before.

[1] For further information, refer to Dynamic Checksum Calculation Manipulation on page 212.

Update Bit group The variables of the `Update Bit` group are for reading the values of the PDU's update bit and the Ignore Update Bit property. The following variables are available if you selected a PDU for the Frame Update Bit Control feature:

Variable	Description
Value	Value of the received update bit.
Ignore Update Bit	Value of the Ignore Update Bit property which is specified in the FlexRay Configuration Tool.

Status group The variables of the `Status` group are for reading status information of an RX PDU. The following variables are available if you selected a PDU for the Frame Receive Status Access feature:

Variable	Description
Error	Indicates various statuses. Each status has its own bit, so parallel statuses are possible. In that case the error values are added. For information on the values, refer to CTR<n> Controller Functions (FlexRay) (ConfigurationDesk Function Block Properties 📖).
Data Receive Counter	Counts the number of received FlexRay frames, including null frames.
Nullframe	Indicates whether the received frame was a null frame (only for static PDUs): ▪ 0: The last received frame was not a null frame. ▪ 1: The last received frame was a null frame.
Multiplexer Switch Code	Indicates the switch code that was used for a send PDU (only for PDUs containing sub-PDUs).
RX Timestamps	Displays the time at which the corresponding LPDU was received on the FlexRay bus.

Raw Data group The variables of the `Raw Data` group return information on the raw data and the corresponding payload length. The following variables are available if you selected a PDU for the Frame Raw Data Access feature:

Variable	Description
Rx Bytes	Raw data of the RX PDU
Payload Length Value	Payload length of the RX raw data. It is only available if the Payload length readable property of the PDU is enabled.

Update Contained PDU The variables of the `Update Contained PDU` group are for reading status information on whether the contained IPDU was contained in the container IPDU. The following variables are available if you have selected a contained IPDU for the Frame Container Control feature:

Variable	Description
Value	Indicates whether the contained IPDU was contained in the container IPDU: ▪ 0: Contained IPDU was not contained in the container IPDU. ▪ 1: Contained IPDU was contained in the container IPDU.

For more information on container IPDUs and contained IPDUs, refer to Aspects of Miscellaneous Supported AUTOSAR Features (FlexRay Configuration Tool Guide 🕮).

Secure Onboard Communication group The variables of the `Secure Onboard Communication` group are used to get information on received secured IPDUs. The following variables are available if you have selected an authentic IPDU for the Frame Authentication Control feature:

Variable	Description
Authenticator Value	The `Authenticator Value` group contains the following groups and variables: ▪ `Value`: The value shows the actually received (truncated) authenticator value. This variable does not correspond to any user code variable. ▪ The `Control` group contains the following variables: ▪ `Enable`: The meaning of this value is user-code-dependent. Proposed values are: ▪ 0: Verification is disabled. ▪ 1: Verification is enabled. This variable corresponds to the `EnableVerification` user code variable. ▪ `Status`: The meaning of this value is user-code-dependent. Proposed values are: ▪ 0: Verification successful. ▪ 1: Verification not successful. ▪ 2: Verification not successful due to wrong freshness value. ▪ 0x3F: Verification not executed. This variable corresponds to the `VerificationResult` user code variable.
Freshness Value	The `Freshness Value` group contains the following groups and variables: ▪ `Value`: The value shows the actually received (truncated) freshness value. This variable does not correspond to any user code variable. ▪ The `Control` group contains the following variables: ▪ `Calculated Value`: The variable contains the calculated freshness value, i.e., the full value reconstructed in the user code. This variable corresponds to the `CalculatedFreshnessValue` user code variable.

For more information on secure onboard communication, refer to Aspects of Miscellaneous Supported AUTOSAR Features (FlexRay Configuration Tool Guide 🕮).

Global Time Synchronization group The variables of the `Global Time Synchronization` group provide information on a received global time synchronization PDU. The following variables are available if you have selected global time synchronization messages for the Frame Global Time Sync Access feature:

Variable	Description
CRC	CRC value that was transferred with the global time synchronization message via the FlexRay bus.

Variable	Description
E2E Protection Status	Result of the CRC calculation, and, if applicable, an error state when the E2E profile is calculated: ▪ 0x00: E2E_P02STATUS_OK and E2E_E_OK OK: New data has been received. CRC is correct. No data has been lost since the last correct data reception. ▪ 0x01: E2E_P02STATUS_NONEWDATA Error: Check function has been invoked, but no new data is available since the last call. As a result, no E2E checks have been consequently executed. ▪ 0x02: E2E_P02STATUS_WRONGCRC Error: Data has been received, but the CRC is incorrect. ▪ 0x03: E2E_P02STATUS_SYNC NOT VALID: New data has been received after detection of an unexpected behavior of the counter. Data has a correct CRC and a counter within the expected range with respect to the most recent data received, but the determined continuity check for the counter is not finalized yet. ▪ 0x04: E2E_P02STATUS_INITIAL Initial: New data has been received. CRC is correct, but this is the first data since the receiver's initialization or reinitialization, so the counter cannot be verified yet. ▪ 0x08: E2E_P02STATUS_REPEATED Error: New data has been received. CRC is correct, but the counter is identical to the most recent data received with Status _INITIAL, _OK, or _OKSOMELOST. ▪ 0x20: E2E_P02STATUS_OKSOMELOST OK: New data has been received. CRC is correct. Some data in the sequence has probably been lost since the last correct/initial reception, but this is within the configured tolerance range. ▪ 0x40: E2E_P02STATUS_WRONGSEQUENCE Error: New data has been received. CRC is correct, but too many data in the sequence has probably been lost since the last correct/initial reception. ▪ 0x13: E2E_E_INPUTERR_NULL At least one pointer parameter is a NULL pointer. ▪ 0x17: E2E_E_INPUTERR_WRONG At least one input parameter is erroneous, e.g., out of range. ▪ 0x19: E2E_E_INTERR An internal library error has occurred. ▪ 0x1A: E2E_E_WRONGSTATE Function executed in wrong state.
Time Domain Id	Identifier of the global time domain that is transferred in the global time synchronization message.
E2E Sequence Counter	End-to-end protection sequence counter that is transferred in the global time synchronization message.
FlexRay Cycle Counter	FlexRay cycle counter (FCNT) that was current at the time when the global time synchronization PDU was built.
Time Gateway Synchronization Status	Displays the value of the SYNC_TO_GATEWAY (SGW) bit from the Time Base status of the time base manager instance: ▪ 0: Sync to gateway. ▪ 1: Sync to time subdomain.

Variable	Description
User Byte 0	User byte 0 that is transferred in the global time synchronization message.
User Byte 1	User byte 1 that is transferred in the global time synchronization message.
Seconds	Value for seconds that is transferred via the FlexRay bus in the global time synchronization message. Because the FlexRay bus is used to always transfer the time of the next FlexRay Cycle 0 start, the second ratio can deviate from the integer part of `Total Time`.
Nanoseconds	Value for nanoseconds that is transferred via the FlexRay bus in the global time synchronization message. Because the FlexRay bus is used to always transfer the time of the next FlexRay Cycle 0 start, the nanosecond ratio almost always deviates from the decimal places of `Total Time`.
Total Time	Time value that is transferred via the FlexRay bus. This is the time that was read when the global time synchronization message was coded.
Time Base Status	Time base status information of the global time synchronization PDU (name, bit position, value): ▪ TIMEOUT, Bit 0 (LSB), ▪ 0: No synchronization timeout ▪ 1: Synchronization timeout. The time base has not been synchronized for a longer period than specified by the Loss Timeout, if a time slave is connected to the time base. ▪ Reserved, Bit 1, 0 ▪ SYNC_TO_GATEWAY, Bit 2, ▪ 0: Time base and global time master are synchronous. ▪ 1: The forwarding of the global time is interrupted and the time base is synchronized with an ECU that is located on a sublevel of the global time master. ▪ GLOBAL TIME_BASE, Bit 3, ▪ 0: The time base has never been synchronized with the global time master and is executed based on a local time. ▪ 1: The time base has been synchronized with the global time master at least once since the start. ▪ TIMELEAP_FUTURE, Bit 4, ▪ 0: The time did not leap further into the future than specified in the Time leap future parameter. ▪ 1: The time has leapt further into the future than specified by the Time leap future threshold parameter and has not been synchronized properly in the time leap future/past threshold intervals as specified in the time leap healing counter. ▪ TIMELEAP_PAST, Bit 5, ▪ 0: The time did not leap further into the past than specified in the Time leap past parameter. ▪ 1: The time has leapt further into the past than specified by the Time leap past threshold parameter and has not been synchronized properly in the time leap future/past threshold intervals as specified in the Time leap healing counter parameter. ▪ Reserved, Bit 6-31, 0
Status	Status information on the global time synchronization PDU (name, bit position, description): ▪ MSG_TYPE_UNSUPPORTED, Bit 0, The first eight bits of the messages do not have the value 0x20, which is required for GTS messages. ▪ Reserved, Bit 1, – ▪ Reserved, Bit 2, – ▪ MSG_TD_WRONG, Bit 3, The bits 3-0 of byte 2 contain an unexpected Time Domain ID.

Variable	Description
	▪ MSG_NANOSECONDS_INVALID, Bit 4, The value for nanoseconds in the bytes 12-15 is larger than 1,000,000,000. ▪ Reserved, Bit 5, – ▪ E2E_PROTECTION_API_ERROR, Bit 6, The value for E2E_Protection does not match E2E_P02STATUS_OK. ▪ Reserved, Bit 7, – ▪ Reserved, Bit 8, – ▪ TBM_UPDATED_BUT_SC_NOT_OK, Bit 9, The value of the E2E_Protection does not match the E2E_P02STATUS_OK, but the time base was synchronized because the return value of the E2E_Protection was E2E_P02STATUS_OKSOMELOST or E2E_P02STATUS_INITIAL, for example. ▪ TBM_UPDATED_BUT_CRC_NOT_OK, Bit 10, The CRC protection was negative but the time base was still synchronized because CRC-IGNORED is specified in the database for the CRC, for example. ▪ TBM_UPDATED_ALL_OK, Bit 11, The time base was synchronized and both the CRC and the sequence counter check were completed successfully.

For more information on global time synchronization, refer to Aspects of Miscellaneous Supported AUTOSAR Features (FlexRay Configuration Tool Guide 🕮).

Related topics

Basics

Sending and Receiving Signal- and Signal-Group-Specific Update Bits.................................... 167
TRC File Containing Several PDUs with the Same Short Name... 207
Working with ControlDesk (SCALEXIO – Hardware and Software Overview 🕮)

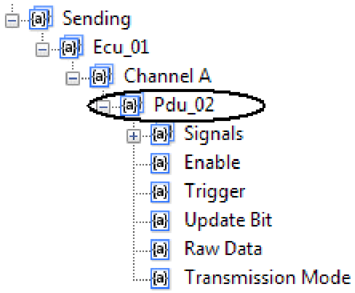
TRC File Containing Several PDUs with the Same Short Name

Introduction

If several PDUs are sent or received by the same ECU, the structure of the TRC file is modified to distinguish the individual PDU instances. The group name of the PDUs is therefore extended. Some cases are shown below.

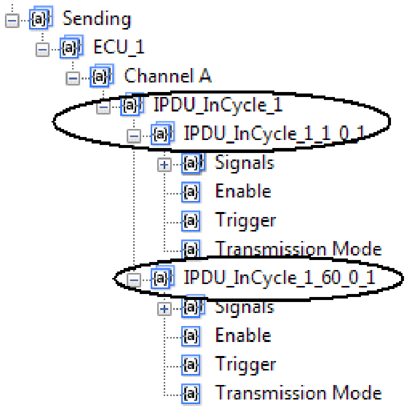
PDU is sent or monitored once

If a PDU is sent or monitored only once, the group name is the short name of the PDU. The following example shows the `Pdu_02` PDU.



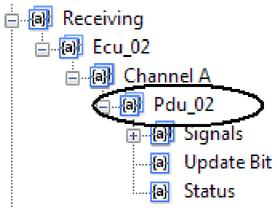
PDU is sent or monitored several times

If a PDU is sent or monitored several times by an ECU, several groups are created. The group names are the short name of the PDU plus the different timings (SlotID_BaseCycle_CycleRepetition). These groups are subordinate to a group whose name is the short name of the PDU. The following example shows the `IPDU_InCycle_1` PDU, which is sent with two different timings (`IPDU_InCycle_1_1_0_1` and `IPDU_InCycle_1_60_0_1`).



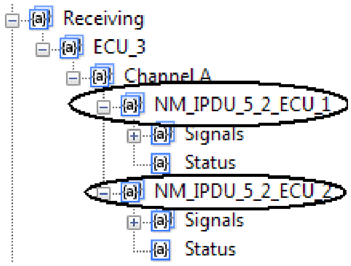
PDU is received once

If a PDU is received once and sent by one ECU, the group name is the short name of the PDU. The following example shows the `Pdu_02` PDU.



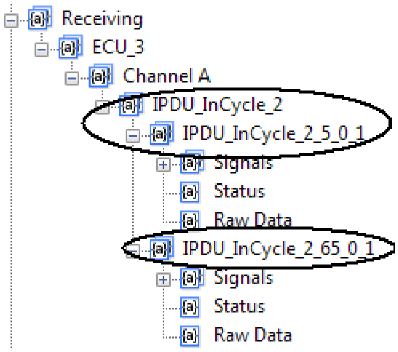
PDU is received once but was sent by several ECUs

If a PDU is received once but sent by several ECUs, the group name is the short name of the PDU plus the short name of the sending ECU. The following example shows the `NM_IPDU_5_2` PDU which is sent by ECU_1 and ECU_2 (`NM_IPDU_5_2_ECU_1` and `NM_IPDU_5_2_ECU_2`).



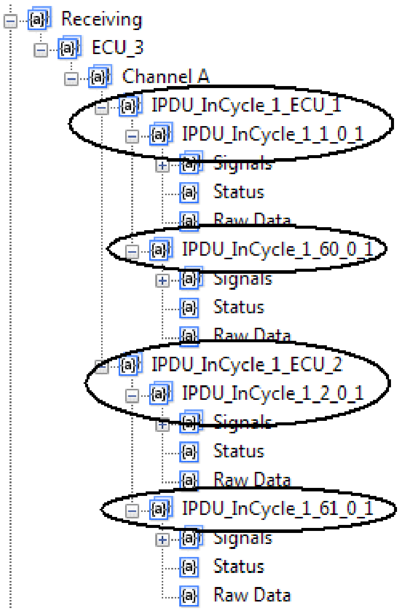
PDU is received several times and was sent by one ECU

If a PDU is received several times and sent by one ECU, the group name is the short name of the PDU plus its timing (SlotID_BaseCycle_CycleRepetition). These groups are subordinate to a group whose name is the short name of the PDU. The following example shows the `IPDU_InCycle_2` PDU, which is sent with different timings (`IPDU_InCycle_2_5_0_1` and `IPDU_InCycle_2_65_0_1`).



PDU is received several times and was sent by several ECUs

If a PDU is received several times and sent by several ECUs, several groups are created. The group names are the short name of the PDU plus the different timings (SlotID_BaseCycle_CycleRepetition). These groups are subordinate to a group whose name is the short name of the PDU plus the short name of the ECU. The following example shows the `IPDU_InCycle_1` PDU, which is sent by ECU_1 with different timings (`IPDU_InCycle_1_1_0_1` and `IPDU_InCycle_1_60_0_1`), and which is also sent by ECU_2 with different timings (`IPDU_InCycle_1_2_0_1` and `IPDU_InCycle_1_61_0_1`).



Dynamic Signal Manipulation

Introduction

You can manipulate a send signal for a specified number of transmission cycles. After the manipulation, the signal has the value of the signal source that was selected before.

Basics

It is possible to modify the data of a TX PDU. In dynamic signal manipulation, the modification is valid only for a specified time span. The time span can be specified by a parameter. After this time has elapsed, the real-time application uses the signal source that was selected before. In addition, you can set the validity status of the signal to a specific value during this time.

TRC entries

To handle dynamic signal manipulation, the following TRC entries are inserted in the TRC file for each signal.

Name	Description	Optional
Dynamic Value	Value to be sent during dynamic signal manipulation. The value must be specified as the coded data type.	No
Dynamic Physical Value	Value to be sent during dynamic signal manipulation, specified as the physical data type.	Yes
Dynamic Validity	Signal validity to be sent during dynamic signal manipulation.	Yes
Countdown Value	Time span for dynamic signal manipulation. It specifies how often the dynamic physical value and dynamic validity status must be sent.	No

You can connect the variables to ControlDesk instruments to set their values.

Selecting the data source

The TRC file contains the `Source Switch` variable for selecting the data source. The following values are defined for the `Source Switch` variable.

Value	Description
0	The value of the Simulink model is sent. If there is no corresponding signal block, the coded default value or the last value that was sent is used.
1	The value of the already used `Value` variable is used.
2[1]	The current value of the alive counter is used for sending.
8	The value of the `Dynamic Value` variable is sent together with the `Dynamic Validity` value for the specified time span (`Countdown Value`). When `Countdown Value` reaches 0, the `Source Switch` variable is automatically switched back to the switch state that was set before.
12[1]	The alive counter is stopped for the time span specified by the `Countdown Value`. The last value of the alive counter is sent. When `Countdown Value` reaches 0, it is switched back to the last `Source Switch` value.

[1] This value must only be set if you have configured the signal as an alive counter, see Using an Alive Counter in the Simulation on page 215.

The `Source Switch` variable is not changed if you use invalid values for it.

Related topics

Basics

Dynamic Update Bit Manipulation

Introduction

You can manipulate the update bit of a TX PDU for a specified number of transmission cycles. After the manipulation, the update bit has the value of the data source that was selected before.

Basics

It is possible to modify the update bit of a TX PDU. In *dynamic* update bit manipulation, the modification is valid only for a specified number of transmission cycles. You can specify this countdown value. After that the update bit manipulation behaves as before: that is, the update bit value and the update bit enable values are reset to the values of the data source that was selected before.

TRC entries

To manage dynamic update bit manipulation, the following TRC entries are inserted in the TRC file for each TX PDU.

Name	Description	Optional
Dynamic Value	Update bit value to be sent during dynamic update bit manipulation.	No
Dynamic Update Bit Enable	Update bit enable value to be used during dynamic update bit manipulation.	No
Countdown Value	Number of transmission cycles the dynamic update bit value and the dynamic update bit enable value are used for update bit manipulation. After that the values are reset to the previous update bit and enable update bit values.	No

You can connect the variables to ControlDesk instruments to set their values.

Selecting the data source

The TRC file contains the `Switch` variable for selecting the data source. The following values are defined for the `Switch` variable.

Value	Description
0	The values of the Simulink model are used for update bit manipulation.
1	The values of the already used `Value` and `Update Bit Enable` variables are used.
8	The values of the `Dynamic Value` and `Dynamic Update Bit Enable` variables are used for the specified number of transmission cycles (`Countdown Value`). When `Countdown Value` reaches 0, the `Switch` variable is automatically switched back to the switch state that was set before.

The `Switch` variable is not changed if you use invalid values for it.

Related topics

Basics

Using the Generated TRC File of PDU-Based Modeling in ControlDesk.................................. 189

Dynamic Checksum Calculation Manipulation

Introduction

You can manipulate checksum calculation of TX PDUs for a specified number of transmission cycles and checksum calculation of RX PDUs for a specified number of receive cycles. After the manipulation, checksum calculation returns to the previous mode, i. e., the real-time application uses the data sources for CRC type and CRC enable that were selected before.

Basics

It is possible to modify checksum calculation for TX and RX PDUs. In *dynamic* checksum calculation manipulation, the modification is valid only for a specified number of transmission or receive cycles. You can specify this number by a parameter. After that the checksum calculation manipulation behaves as before:

that is, the real-time application uses the data sources for CRC type and CRC enable that were selected before.

TRC entries

If CRC calculation is enabled and selected for the TRC file in the FlexRay Configuration Tool, variables to manage dynamic checksum calculation manipulation are also written to the TRC file. The following TRC entries are inserted for each TX PDU and RX PDU.

Name	Description	Optional
Dynamic Type	Type value to be sent during dynamic checksum calculation manipulation.	No
Dynamic CRC Enable	CRC enable value to be used during dynamic checksum calculation manipulation.	No
Countdown Value	Number of transmission cycles the dynamic type value and the dynamic CRC enable value are used for checksum calculation manipulation. After that the values are reset to the previous type and enable CRC values.	No

You can connect the variables to ControlDesk instruments to set their values.

Selecting the data source

The TRC file contains the `Source Switch` variable for selecting the data source. The following values are defined for the `Source Switch` variable.

Value	Description
0	The values of the Simulink model are used for checksum calculation manipulation.
1	The values of the already used `Type` and `CRC Enable` variables are used for checksum calculation manipulation.
8	The values of the `Dynamic Type` and `Dynamic CRC Enable` variables are used for checksum calculation manipulation for the specified number of transmission cycles (`Countdown Value`). When `Countdown Value` reaches 0, the `Source Switch` variable is automatically switched back to the switch state that was set before.

The `Source Switch` variable is not changed if you use invalid values for it.

Related topics

Basics

Using the Generated TRC File of PDU-Based Modeling in ControlDesk.................................... 189

Manipulating the Authenticator Value of a Secured IPDU

Introduction

The FlexRay Configuration Tool supports secure onboard communication (SecOC) according to AUTOSAR. Secure onboard communication means securing the payload of authentic IPDUs by authentication mechanisms. Authentication information that is included in a secured IPDU consists of an authenticator and a freshness value.

Authenticator invalidation of TX secured IPDUs

You can manipulate the authenticator value of a secured IPDU that is to be transmitted during run time. You cannot specify an explicit authenticator value, but you can determine whether the authenticator value should be interpreted as a correct or an incorrect value.

Access to the invalidation activation is via the TRC variable `Authenticator Value`. Refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

Related topics

Basics

Aspects of Miscellaneous Supported AUTOSAR Features (FlexRay Configuration Tool Guide 📖)

Dynamic Manipulation of GTS Communication

Introduction

You can manipulate the GTS communication of a TX PDU for a specified number of transmission cycles. After the manipulation, the time synchronization PDUs have the values of the data source that was selected before.

Basics

It is possible to modify some GTS-relevant data of a time synchronization PDU to be sent. In *dynamic* GTS communication manipulation, the modification is valid only for a specified number of transmission cycles. You can specify this countdown value. After this the GTS communication manipulation behaves as before, that is, the GTS values are reset to the values of the data source that was selected before.

TRC entries

To manage dynamic manipulation of GTS communication, the following TRC entries are inserted in the TRC file for each global time synchronization PDU that is sent.

Name	Description	Optional
Dynamic Value	Value of the GTS parameter to be sent during dynamic manipulation.	No
Countdown Value	Number of transmission cycles the dynamic value is used for GTS communication manipulation. After that the values are reset to the previous values.	No

You can connect the variables to ControlDesk instruments to set their values.

Selecting the data source

To select the data source, the TRC file contains the `Source Switch` variable. The following values are defined for the `Source Switch` variable.

Value	Description
0	The value of the time base manager (time gateway synchronization status, user byte 0, user byte 1, total time) or the fixed E2E_P02 algorithm (CRC, E2E sequence counter) is used for GTS communication manipulation.
1	The value of the already used `Value` variable is used.
8	The value of the `Dynamic Value` variable is used for the specified number of transmission cycles (`Countdown Value`). When `Countdown Value` reaches 0, the `Source Switch` variable is automatically switched back to the switch state that was set before.

If you use invalid values for the `Source Switch` variable, it is not changed.

Related topics

Basics

Aspects of Miscellaneous Supported AUTOSAR Features (FlexRay Configuration Tool Guide 🕮)
Using the Generated TRC File of PDU-Based Modeling in ControlDesk.................................. 189

Using an Alive Counter in the Simulation

Introduction

You can activate and monitor an alive counter in ControlDesk if you specified a signal as alive counter in the FlexRay Configuration Tool and activated the **Signal Alive Counter Control** feature in the **Element Selection** dialog. Refer to How to Configure an Alive Counter (FlexRay Configuration Tool Guide 🕮).

Configuring an alive counter at run time

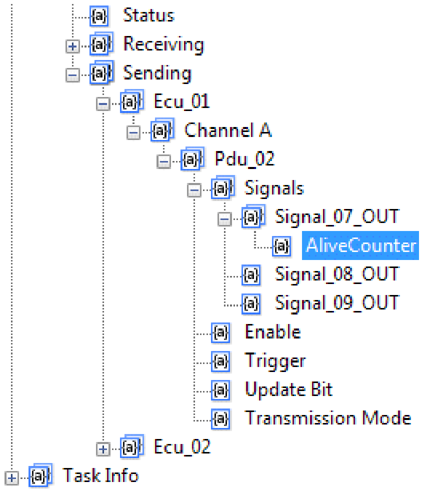
To configure the alive counter during run time, the `Source Switch` variable created for dynamic signal manipulation has two additional states (for more information on the `Source Switch` variable and its states, refer to Dynamic Signal Manipulation on page 210).

Value	Description
2	The current value of the alive counter is used for sending.
12	The alive counter is stopped for the time span specified by the `Countdown Value`. The last value of the alive counter is sent. When `Countdown Value` reaches 0, it is switched back to the last `Source Switch` value.

If a signal is specified as an alive counter, the initial value of the `Source Switch` variable is 2. Sending therefore begins with the alive counter.

Manipulating an alive counter at run time

The TRC file contains the variables for controlling the alive counter in an `AliveCounter` group below the send signal.



```
Status
Receiving
Sending
  Ecu_01
    Channel A
      Pdu_02
        Signals
          Signal_07_OUT
            AliveCounter
          Signal_08_OUT
          Signal_09_OUT
        Enable
        Trigger
        Update Bit
        Transmission Mode
  Ecu_02
Task Info
```

The `AliveCounter` group comprises the following variables:

Variable Name	Description	Range
Value	The current value of the alive counter.	Coded data type
Runtime Behavior	Specifies the behavior of the alive counter: ▪ 0: The alive counter runs independently of the selected source for the signal (`Source Switch` variable). **Note** There is one exception: If `Source Switch` is 12, the alive counter stops independently of the `Runtime Behavior`. ▪ 1: The alive counter runs only if its value is really sent (`Source Switch` is 2). If transmission is switched to a source other than the alive counter, for example, a value from the Simulink model, the alive counter stops. ▪ 2: The alive counter stops at the current value.	0, 1, 2
Offset	Allows you to add an offset to the value of the alive counter at run time.	Coded data type

You can view and configure the variables in ControlDesk. The variables can be connected to appropriate ControlDesk instruments to control the alive counter.

Related topics

Examples

Example of Using an Alive Counter......................................217

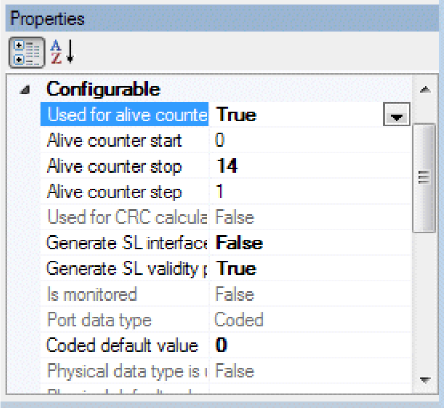
Example of Using an Alive Counter

Introduction

The example demonstrates the configuration and manipulation of an alive counter.

Configuring the TX node

The alive counter must be configured for the TX node in the FlexRay Configuration Tool. After importing the FIBEX file or AUTOSAR system description file, you must drag the send signal you want to use as the alive counter into the **Configuration** view. Then set the **Used for alive counter** property to **True** and specify the parameters of the alive counter, see the following illustration.



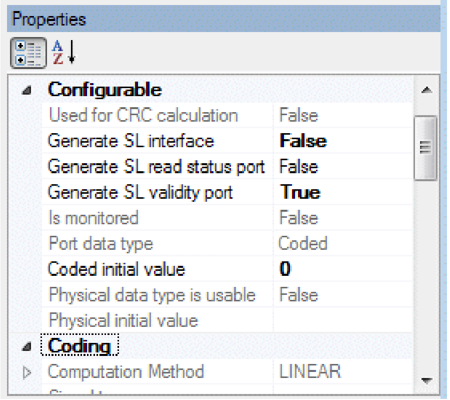
Properties

Configurable	
Used for alive counte	True
Alive counter start	0
Alive counter stop	14
Alive counter step	1
Used for CRC calcula	False
Generate SL interface	False
Generate SL validity p	True
Is monitored	False
Port data type	Coded
Coded default value	0
Physical data type is	False

The alive counter starts at 0 at the beginning of the simulation. It is increased by 1 every time the signal is sent. When a value of 14 is reached, the alive counter starts at 0 again. In addition, the **Generate validity port** parameter is set to True. To generate a variable for the alive counter in the TRC file, the **Signal Alive Counter Control** feature must be selected in the **Element Selection** dialog (see How to Prepare the Manipulation or Monitoring of Frames/PDUs and Signals via TRC File (FlexRay Configuration Tool Guide 📖)). After the automatic task generation has been executed, the code for the TX node can be generated.

Configuring an RX node

RX nodes are the nodes that receive the alive counter. To configure an RX node, you must drag the RX signal which contains the alive counter into the **Configuration** view after importing the FIBEX file or AUTOSAR system description file. The RX signal is configured as follows.



Properties

Configurable	
Used for CRC calculation	False
Generate SL interface	**False**
Generate SL read status port	False
Generate SL validity port	**True**
Is monitored	False
Port data type	Coded
Coded initial value	**0**
Physical data type is usable	False
Physical initial value	
Coding	
Computation Method	LINEAR

After automatic task generation has been executed, the code for the RX node can be generated.

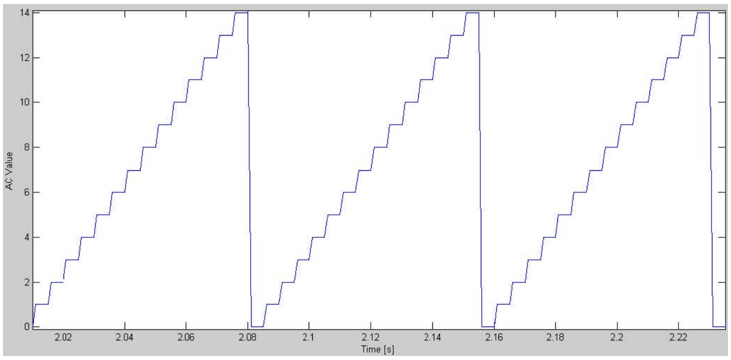
Connecting the variables to instruments

To control the alive counter, you must set some variables. This can be done as usual in ControlDesk.You must connect the variables of the `AliveCounter` group to suitable instruments to view and configure the variables of the alive counter.

To monitor the alive counter, you must create a simple layout for the RX node. It is sufficient to add one Display instrument that is connected to the `Value` variable of the `AliveCounter` group to the layout.

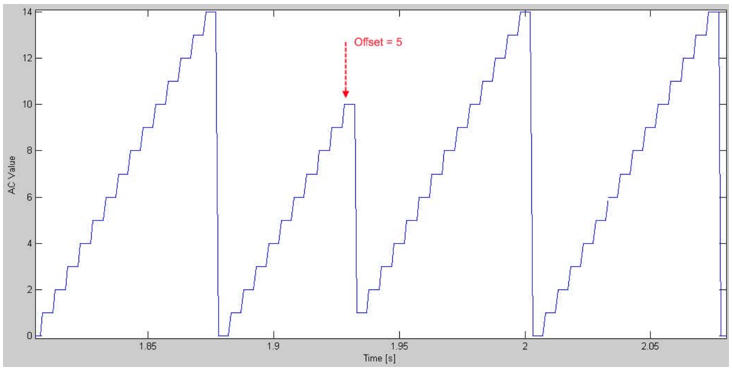
Default behavior

As the send signal has been configured as an alive counter, the `Source Switch` variable is configured to the default value 2. This means that the alive counter value is sent from the beginning of the simulation, see the following illustration.



Adding an offset to the alive counter

You can add an offset value to the alive counter during run time. This is done via the `Offset` variable which is in the `AliveCounter` group of the respective signal. The value is added as soon as a value for the offset is defined and confirmed. The following illustration shows an example. When the alive counter has a value of 10, an offset of 5 is added. The value of the alive counter is 1 in the next step and continues running as before.

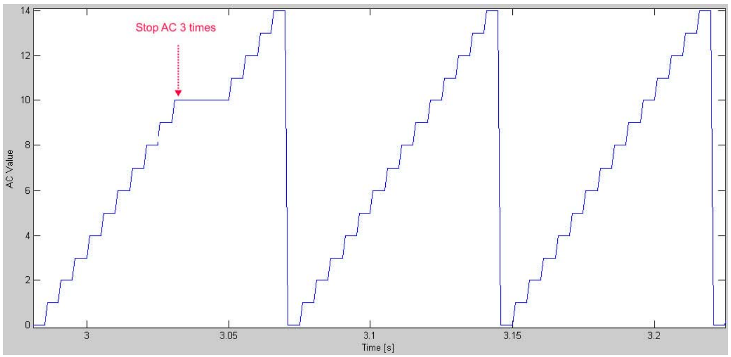


Stopping the alive counter

You can stop the alive counter for a certain time or permanently.

To stop it permanently, you can use the `Runtime Behavior` variable which is contained in the `AliveCounter` group. As soon as this variable is set to 2, the alive counter stops at its current value.

To stop the alive counter for a certain time only, the dynamic signal manipulation feature is used (see Dynamic Signal Manipulation on page 210). The `Countdown Value` parameter must be set first. In this example, the value is set to 3 to stop the alive counter for 3 transmissions. After the value is set, the `Source Switch` parameter is set to 12. The alive counter is stopped for 3 transmissions and continues counting afterwards.



Switching to a constant value

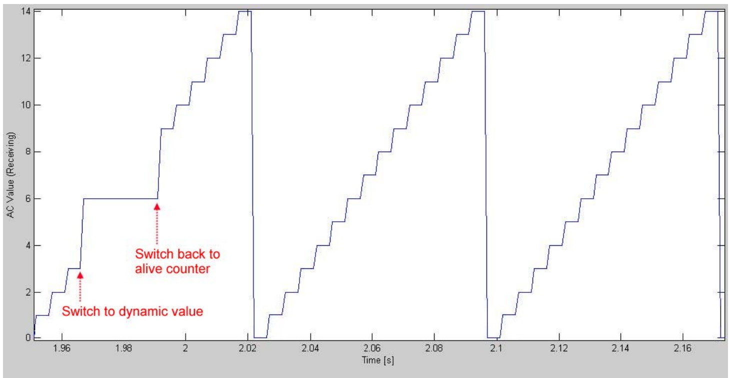
While the alive counter is being transmitted, you can switch to a predefined value for a certain time or permanently. You can also switch to the Simulink model value (`Source Switch` = 0). You can use the `Runtime Behavior` variable to set the behavior of the alive counter during that time.

- 0: The alive counter continues running in the background.
- 1: The alive counter stops at the current value.

In the following example, a value of 6 is transmitted for 5 transmissions. During this time, the alive counter continues running in the background. The parameter must be set to the following values for this:

- `Countdown Value`: 5
- `Dynamic Value`: 6
- `Runtime Behavior`: 0

To switch to the dynamic value, set the `Source Switch` variable to 8. After 5 transmissions, `Source Switch` is automatically reset to the previous value (2) and the value of the alive counter is used for transmission again. See the following illustration.



Related topics

Basics

Using an Alive Counter in the Simulation..215

Modeling a LIN Bus Interface

Introduction

You can model a LIN bus interface in Simulink by using the RTI LIN MultiMessage Blockset. The following topics provide information on using the RTI LIN MultiMessage Blockset.

Note

VEOS Player does not support Simulink implementation containers generated for models that contain blocks from the RTI LIN MultiMessage Blockset.

Where to go from here

Information in this section

General Information on Modeling LIN Communication

Introduction

The SCALEXIO system can simulate a LIN bus. You can connect your ECU to the simulated LIN bus to perform restbus simulation for communication testing purposes, and to simulate LIN frame errors and errors on the physical layer of the LIN bus.

An ECU is to a LIN bus simulated by the SCALEXIO system in several steps. One of these is to model the LIN communication in Simulink. For information on the whole workflow, refer to LIN Bus Connection (SCALEXIO – Hardware and Software Overview 🕮).

Modeling LIN communication in Simulink

Several steps have to be performed to make communication via the LIN bus available. One of them is to model the LIN communication in Simulink using blocks from the RTI LIN MultiMessage Blockset. The RTI LIN MultiMessage Blockset lets you configure and handle a large number of LIN frames. All the incoming and outgoing frames of an entire LIN controller can be controlled by a single Simulink block. For detailed information on the RTI LIN MultiMessage Blockset, refer to the RTI LIN MultiMessage Blockset Reference 🕮.

If you model the LIN communication for a SCALEXIO system and want to perform model separation later on in ConfigurationDesk to execute the separated models on single cores of the SCALEXIO system, you need to know some specifics. Refer to Separating Models That Contain CAN or LIN Bus Communication on page 113.

After LIN communication is modeled in Simulink, the Simulink model is analyzed in ConfigurationDesk. During model analysis, one Configuration Port block is generated for every identified **RTILINMM ControllerSetup** block contained in the Simulink model. For further information, refer to Building the Signal Chain for LIN Bus Communication (ConfigurationDesk Real-Time Implementation Guide 📖).

Demo model

ConfigurationDesk comes with a LINMM demo giving you an example of how to implement LIN communication using the RTILINMM blocks in the behavior model. The demo is available in the project root folder after you first start ConfigurationDesk.

Note

If you work with RCP and HIL (64-bit) software: To use the LINMM demo model, you must create the S-functions for all the RTILINMM blocks in the model and save the model before analyzing it in ConfigurationDesk and building a real-time application. Use the **Create S-Function for all LIN Blocks** command from the **Options** menu of the **RTILINMM GeneralSetup** block for this. For further information, refer to Options Menu (RTILINMM GeneralSetup) (RTI LIN MultiMessage Blockset Reference 📖).

Basics on the RTI LIN MultiMessage Blockset

Introduction

The RTI LIN MultiMessage Blockset is a Simulink blockset for efficient and dynamic handling of complex LIN setups in hardware-in-the-loop (HIL) applications. All the incoming RX frames and outgoing TX frames of an entire LIN controller can be controlled by a single Simulink block. LIN communication is configured via database files (DBC file format, LDF file format, FIBEX file format, MAT file format, or AUTOSAR XML file format).

Supported dSPACE platforms

The RTI LIN MultiMessage Blockset is supported by the following platforms:

- SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, and/or DS6351 LIN Board
- PHS-bus-based systems (DS1006 or DS1007 modular systems) with a DS4330 LIN Interface Board
- MicroAutoBox II with LIN interface

You have to recreate all the RTI LIN MultiMessage blocks if you switch platform (for example, from SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, and/or DS6351 LIN Board to a DS1007, or from a DS1007 to a MicroAutoBox II). To recreate all the RTILINMM blocks at once, select Create S-function for All LIN Blocks from the options menu of the GeneralSetup block (refer to Options Menu (RTILINMM GeneralSetup) (RTI LIN MultiMessage Blockset Reference 🕮)).

> **Note**
>
> The RTI LIN MultiMessage Blockset is not supported by the MicroAutoBox III. If you work with the MicroAutoBox III, you must use the Bus Manager to implement LIN bus simulation.

Managing large LIN frame bundles

With the RTI LIN MultiMessage Blockset, you can configure and control a large number of LIN frames from a single Simulink block. This reduces the size of model files and the time required for code generation and the build process.

Manipulating signals to be transmitted

The RTI LIN MultiMessage Blockset can manipulate the counter values of signals before they are transmitted.

When simulating with signal manipulation, you can specify whether to use the signal from the model or the TRC file. This is useful for simulating error values.

Specifying signal saturation

You can use the signal limits specified in the database file or specify other limits. If the input signal is outside of the specified range, it is set to the minimum or maximum limit. Refer to Saturation Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference 🕮).

Updating a model

The RTI LIN MultiMessage Blockset creates an S-function for the specified database file. You can easily update the LIN configuration of a model by replacing the database file and updating the S-function.

Modifying model parameters during run time

Model parameters such as frames or signal values can be modified during run time either via model input or via the Bus Navigator in ControlDesk. For modifying model parameters via the Bus Navigator a variable description file (TRC) is automatically generated each time you create an S-function for the RTILINMM MainSetup block. The entries of the TRC file let you analyze received signals, change the values of signals to be transmitted, etc. In ControlDesk, you can access the settings specified in the TRC file via the model's system description file (SDF). The SDF file bundles all the TRC files and additional information for the application.

(Relevant only for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, and/or DS6351 LIN Board) For information on where to find the signals of the LIN bus in the TRC file, refer to Details on the Variable Groups of the Behavior Model on page 238.

Variables of custom code

You can include variables of custom code in a `USR.TRC` file in addition to the parameters of the RTI LIN MultiMessage Blockset.

TRC file entries with initial data

TRC/SDF files generated for Simulink models including blocks from the RTI LIN MultiMessage Blockset contain initial data. The RTI LIN MultiMessage Blockset supplies all variables with initial values when they are included in the TRC file. TRC files with initial data allow you to perform offline calibration with ControlDesk.

Visualization with the Bus Navigator

The RTI LIN MultiMessage Blockset supports visualization with the Bus Navigator. Layouts/instruments are generated on demand and provide access to all the LIN signals and all the switches required to configure LIN communication during run time. You do not have to preconfigure layouts by hand.

Monitoring and logging LIN bus communication

The RTI LIN MultiMessage Blockset supports filtered and unfiltered monitoring and logging of LIN bus communication with the Bus Navigator. You can observe the raw and physical data of LIN frames on a LIN bus, and log the raw data of LIN frames. You can monitor and log LIN bus events.

Transmitting and Receiving LIN Frames

Introduction

Large LIN frame bundles (up to 63 frames) can be managed from a single Simulink block provided by the RTI LIN MultiMessage Blockset.

Defining LIN communication

To define the LIN communication of a LIN controller, you must provide a database file.

LDF file as the database You can use the LDF file format as the database for LIN communication. The LDF file format describes a complete LIN network and contains all the information necessary to configure it. The file format was specially developed for LIN networks, so it can describe all the features of a LIN network. For more information on the LDF file format, refer to the *LIN Configuration Language Specification*.

DBC file as the database You can also use the DBC file format as the database for LIN communication. The DBC file format was developed by Vector Informatik GmbH, Stuttgart, Germany. For the RTI LIN MultiMessage Blockset, you can use all the DBC files that pass the consistency check of Vector Informatik's CANdb++ Editor.

The DBC file format was developed for CAN buses, but it can also be used for LIN buses with some limitations:

- As the CAN protocol has no standard definitions for master nodes and schedules, such definitions are not supported by the DBC file formats.
- The LIN specification supports only byte layouts in the Intel format. Byte layouts in Motorola format are not supported.

FIBEX file as the database The Field Bus Exchange (FIBEX) format is an XML exchange file format. It is used for data exchange between different tools that work with message-oriented bus communication. A FIBEX file usually describes more than one bus system. You therefore have to select one of the available bus systems if you work with a FIBEX file as the database.

MAT file as the database You can also use the MAT file format as the database for LIN communication, or specify other database file formats as the database. You must convert your database files into the MAT file format for this purpose.

AUTOSAR system description file as the database You can also use AUTOSAR system description files as the database for LIN communication. AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership that aims to develop and establish an open standard for automotive electric/electronic (E/E) architectures.

AUTOSAR system description files are files of AUTOSAR XML file type that describe a system according to AUTOSAR. A system is a combination of a hardware topology, a software architecture, a network communication, and information on the mappings between these elements. AUTOSAR system description files are instances of the AUTOSAR System Template. The AUTOSAR System Template contains a description of the network communication and hardware topology according to the FIBEX standard defined by ASAM e.V.

Defining RX frames and TX frames

You can receive and/or transmit each frame defined in the database file that you specify for LIN communication.

Defining RX frames You can define RX frames on the RX Frames Page (RTILINMM MainSetup).

Defining TX frames You can define TX frames on the TX Frames Page (RTILINMM MainSetup).

Working with event-triggered frames

The RTI LIN MultiMessage Blockset allows you to work with event-triggered frames. In contrast to a standard LIN frame, the frame header of an event-triggered frame is assigned to several frame responses of different LIN nodes. This allows you to transmit the frames of different nodes via the same frame slot. If you work with LIN 2.1 or later, you can specify collision resolver schedules to resolve collisions which might occur if two or more event-triggered frames are sent at the same time via the same frame slot. For details, refer to Eventtriggered Frames Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference 📖).

Working with raw data

The RTI LIN MultiMessage Blockset allows you to work with the raw data of frames. You have to select the frames for this purpose. The RTILINMM MainSetup block then provides the raw data of these frames to the model byte-wise for further manipulation. You can easily access the raw data of RX frames via a Simulink Bus Selector block. For details, refer to Raw Data Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference 🕮).

Implementing checksum algorithms

In addition to the checksums of the LIN frames, you can use a signal to transmit a user-defined checksum. You can implement checksum algorithms for the checksum calculation of TX frames and checksum verification of RX frames.

Checksum header file You have to specify the checksum algorithms in a checksum header file. This needs to have a C-coded switch-case directive to switch between algorithms.

Checksum calculation for TX frames You can assign a checksum algorithm to each TX frame. A checksum is calculated according to the algorithm and assigned to the TX frame. Then the frame is transmitted together with the calculated checksum.

Checksum check for RX frames You can assign a checksum algorithm to each RX frame. A checksum is calculated for the frame and compared to the checksum in the received frame. If they differ, this is indicated at the error ports for RX frames if these ports are enabled. For details, refer to User Checksum Definition Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference 🕮).

LIN frame checksum manipulation You can manipulate the checksums of LIN frames. The checksum is increased by a specified fixed offset value for a defined number of times.

Receiving frames with a different frame length than specified in database file

If the length of a received frame is different than that specified in the database file, how the frame is handled depends on its length.

Received frame shorter than expected If a frame is shorter than specified in the database file, it is not decoded. The error port is set to `Slave not responding` (`RX_Error = 16`). The frame status remains 0. The RX_Time is updated. The received data is output at the RX Raw Data port. Raw data bytes which are not received are set to `0`. You can read the frame length at the Frame Length port.

Received frame longer than expected If a frame is longer than specified in the database file, only the specified frame length is read. The next byte is interpreted as a checksum. A checksum error is therefore displayed in most cases. The raw data, RX_Time and RX_DeltaTime are updated. The error port is set to `Checksum Error` (`RX_Error = 8`). If the last byte is the correct checksum, the frame is decoded as usual.

How to Define a Checksum Algorithm

Objective

You can specify your own checksum algorithm for frames.

Basics

You can implement checksum algorithms for frames via a *checksum header file*. Each algorithm must have a C-coded switch-case directive in the header file.

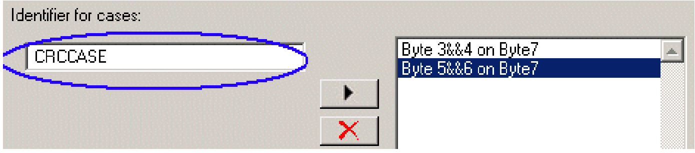
Method

To define a checksum algorithm

1 Open the User Checksum Definition page.

2 Click h to specify an existing checksum header file, or enter a checksum header file name in the Header file edit field.

3 Name the checksum algorithms in the Identifier for cases field.



Identifier for cases:

CRCCASE

Byte 3&&4 on Byte7
Byte 5&&6 on Byte7

4 In the checksum header file, sort the algorithms.

The order you specify for the algorithms corresponds to the `crctype` index of the switch-case directives in the header file.

5 Click Create Header File to create the header file if you entered its file name in the Header file edit field in step 1.

6 Click Edit Header File to edit the file in MATLAB's M-File Editor.

7 In the checksum header file, edit your checksum algorithms. Note that the header file must contain at least one switch-case directive.

You have access to the following input parameters of the header file:

Input Parameter	Description
`crcoption`	▪ '0' if applied to a TX frame ▪ '1' if applied to an RX frame See User Checksum Frames Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference 📖).
`UInt8* FrameRAW_DATA`	Pointer to 8 bytes of raw data
`crctype`	Index of the switch-case directives for the checksum algorithms. Corresponds to the order you specify in step 3 (see above).
`CsBitPos`	Start position of the checksum signal
`CsLength`	Length of the checksum signal

Input Parameter	Description
`MsgLength`	Frame length (in range 1 ... 8)
`MsgId`	Frame ID (1 ... 59)

Result

Your checksum algorithm is used.

Related topics

References

RTILINMM MainSetup (RTI LIN MultiMessage Blockset Reference 🕮)
User Checksum Definition Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference 🕮)

Manipulating Signals to be Transmitted

Introduction

All the signals of all the RX and TX frames (see Transmitting and Receiving LIN Frames on page 224) automatically get corresponding entries in the generated TRC file. This allows you to analyze them (signals of RX frames) or change their values (signals of TX frames) in the experiment software.

Manipulating signals to be transmitted

The RTI LIN MultiMessage Blockset provides several options to manipulate the values of signals before they are transmitted. You can switch between the options you have specified via entries in the generated TRC file.

You can switch between these options in the experiment software.

TX signals The **RTILINMM MainSetup** block allows you to get the value of TX signals either from the Simulink model or from a variable in ControlDesk. The values of these TX signals can be changed from within the model. Their values cannot be changed in ControlDesk if the **Input Manipulation** option is cleared. Refer to Input Manipulation Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference 🕮).

Counter signals You can specify a counter signal that provides the number of frame transmissions. You can specify the counter start value, the increment, and the counter stop value. Each time the counter reaches the stop value, it turns around to the counter start value. Refer to Counter Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference 🕮).

Related topics

Basics

Transmitting and Receiving LIN Frames..224

Introduction to the Build Process

Where to go from here

Information in this section

Preparing the Build Process

Introduction

After you add a Simulink implementation container or a Simulink behavior model to a ConfigurationDesk application, you can start the build process in ConfigurationDesk. If you want to add custom code to the real-time application or adapt the generated variable description file, you can do this in your modeling environment or in ConfigurationDesk.

Where to go from here

Information in this section

General Information on the Build Process

Introduction

After implementing the behavior model and integrating it in your ConfigurationDesk application, you can start the build process in ConfigurationDesk to generate the real-time application and download it to the hardware. The build process is completely controlled by ConfigurationDesk, but you can prepare your model to influence the build results. A command for starting the build process is available in ConfigurationDesk and in the Simulink model.

> **Note**
>
> Before you start a build process or code generation, the Model Interface Package for Simulink checks whether the model meets the requirements for code generation. You can also execute this model check manually via the **Check Model for DSRT Code Generation** command which is available in the Model Port Blocks menu. Refer to Check Model for DSRT Code Generation (Model Interface Package for Simulink Reference 🕮).

Basics

The build process is separated into the following phases:

1. Model code generation including model analysis

   > **Note**
   >
   > This step is relevant only for MDL files or SLX files, not for SIC files.

   If not already open, MATLAB and the behavior model linked to the ConfigurationDesk application are opened. The model interface of the behavior model is analyzed.

   ConfigurationDesk configures the parameters of the behavior model, for example, the dSPACE Run-Time Target (dsrt.tlc) is set as system target. Afterwards, the Simulink® Coder^{TM} code generation for the behavior model is started.

2. System integration code generation

   System integration code is generated by ConfigurationDesk, including code for I/O functionality and task handling.

3. Make process

   The code generated in the previous process steps is compiled and linked to the final real-time application, which can be downloaded to the real-time hardware.

After the build process is finished, its results can be found at `<Application>\Build Results` in your ConfigurationDesk application. Files which you created yourself and which are processed during the build process, for example user variable description files, are not changed by the build process.

Note

The build process is aborted only for significant errors. The build process is fault-tolerant and considers some standard failures, such as unmapped hardware devices, by generating default code. Most problems found during the build process are therefore categorized as warnings. Look at the messages in the Log Viewer and in the Conflicts Viewer before starting the real-time application to get an assessment of its behavior. You can use the Build Model command in Simulink to check your behavior model for values or variables with unsupported data types, for example.

Build options

ConfigurationDesk configures the model parameters and parameters of the model code generator, such as the Make command and the Embedded hardware parameters in Simulink's Hardware Implementation dialog. Automatic configuration is performed during the build process according to the values needed by ConfigurationDesk, but only if the configuration of the behavior model differs from the configuration required by ConfigurationDesk. Information on configured parameters is displayed in MATLAB's command window during the build process.

Note

- The original configuration is not restored after model code generation.
- For models created with earlier versions of the Model Interface Package for Simulink, the dynamic memory allocation option in Function blocks is deactivated. This affects models created with the Model Interface Package for Simulink of dSPACE Release 2017-A up to and including 2020-A with MATLAB R2017a or later. To activate the option, type the following commands in the MATLAB Command Window:

```
cfgSet = getActiveConfigSet(<mdlName>);
cfgSet.setPropEnabled('MATLABDynamicMemAlloc', 'on');
```

Custom code support

ConfigurationDesk supports custom code in your model. The custom code must be specified in your modeling tool. For further information, refer to Using Custom Code on page 232.

Advanced handling of the variable description file

A variable description file is generated when the real-time application is built. You can adapt the contents of the variable description file. For further information, refer to the following topics:

- Adapting the Generation of the Variable Description File on page 243
- Information on Variable Description Files (TRC Files) on page 233

Related topics

Basics

Building Real-Time Applications (ConfigurationDesk Real-Time Implementation Guide 🕮)

Using Custom Code

Introduction

ConfigurationDesk supports custom code that you specified in your modeling environment.

Integrating custom code in the behavior model

Depending on the code complexity and your modeling procedures, you can add custom code to your behavior model in different ways:

- You can enter C or C++ code in the **Code Generation - Custom Code** pane of the **Configuration Parameters** dialog.
- You can use the Custom Code library (`custcode`) to add code graphically at a specific place within your model.
- You can use the following Simulink blocks:
  - You can add S-Function and S-Function Builder blocks to your model for including C MEX files.
  - You can add Fcn and Embedded MATLAB Function blocks to your model for including mathematical expressions and MATLAB functions.
- You can add source files, search paths and compiler libraries to the make process by using the `rtwmakecfg.m` file.

Note

With dSPACE Release 2020-B, the base operating system on SCALEXIO platforms has changed from a QNX-based to a Linux-based distribution. Simulink models with binaries from dSPACE Release 2020-A or earlier are therefore no longer compatible with the current SCALEXIO operating system. The import into ConfigurationDesk is aborted with an error message. You must recompile the binaries for the current SCALEXIO platform.

For further information, refer to the user documentation from MathWorks®.

Using C++11 custom code

To use custom code that complies with the C++11 standard, you must activate the C++11 standard for the relevant source files or header files. To do so, you must add the following lines to the relevant files:

```
#if defined(__GNUC__) && (__GNUC__ > 5)

#pragma GCC diagnostic warning "-std=c++11"
```

```
#endif
```

Note

dSPACE has not fully quality-assured the C++11 functionality. In addition, dSPACE has not performed any tests on C++11 in conjunction with Microsoft Visual Studio. Thus, dSPACE does not guarantee that all C++11 features are usable or work correctly. If you encounter any problems, contact dSPACE Support (www.dspace.com/go/supportrequest).

Related topics

Basics

Customizing the Build Process with User-Specific Files (For Simulink Behavior Models) (ConfigurationDesk Real-Time Implementation Guide 📖)

Information on Variable Description Files (TRC Files)

Introduction

The TRC file provides information on the variables of a real-time application that is required to connect variables to instruments in a layout of the experiment software, for example. It is an ASCII file that can either be generated automatically by ConfigurationDesk, or written manually. A variable description file is generated during the build process.

Where to go from here

Information in this section

Generating TRC Files

Where to go from here

Information in this section

Basics on the Generation of TRC Files

Introduction

A TRC file provides information on the variables that are used in an executable application. This variable description is a data interface between the executable application executed on dSPACE hardware or VEOS and the dSPACE experimentation software.

In the experimentation software, you get access to the variables by loading a system description file (SDF file). However, the variable description itself is stored in a trace file (TRC file), that is then also loaded. The following description refers to the contents of variable description files in TRC format.

Generating and consuming software products

Build processes that include the generation of TRC files can be started via:

- RTI and RTI-MP
- ConfigurationDesk
- Model Interface Package for Simulink

  The generated intermediate files are used by VEOS and ConfigurationDesk to create a platform-specific TRC file.

dSPACE products that need a generated TRC file to access variables in an executable application are, for example, ControlDesk, AutomationDesk, ModelDesk, and also custom applications based on dSPACE XIL API.

Data source of the variable description

When you build a simulation application, Simulink Coder provides information on the model's variables and parameters in an internal format. This data is used by dSPACE within the build process to generate a variable description in TRC file syntax.

The contents of a variable description depend on the following points:

- The way you have implemented and prepared your model. For example, variables in a subsystem can be excluded by using the subsystem omission tag (`DsVdOmit`).

- Simulink Coder version (represented by the MATLAB Release number).
- Configuration settings for code generation, refer to Simulink configuration settings on page 235.

Configuration settings for the TRC file generation

The contents of the TRC file can be configured by various settings. The most relevant settings are described below.

Simulink configuration settings The behavior of Simulink Coder can be influenced generally by the Code Generation options, that you can find in the model's Configuration Parameters. Also the optimization settings, such as the Default parameter behavior setting or the Signal storage reuse setting, have a direct effect on the available variables in the generated code and therefore on the TRC file generation.

> **Note**
>
> Not all configuration settings lead to the expected behavior, for example, variables can be missing in the TRC file as a result of the Simulink Coder internal optimizations.

dSPACE configuration settings If you use a system target file for a dSPACE platform, you can use the additional code generation options in the model's Configuration Parameters. Especially the DSRT variable description file options page provides settings to configure the contents of a generated TRC file. For further information, refer to Adapting the Generation of the Variable Description File on page 243.

Available TRC File Variable Entries and Their Locations in the TRC File

Introduction

To access the variables in your experiment or test software, it is helpful to know where you can find them in the variable description file (TRC file).

TRC file structure

The TRC file of a real-time application is structured according to the following groups:

- Simulation and RTOS group
- Variable groups from the behavior model
- Signal Chain group
- Diagnostics group
- XIL API/EESPort group (available as of dSPACE Release 2016-A)

Simulation and RTOS The Simulation and RTOS group contains variables related to simulation and the real-time operating system. Refer to Simulation and RTOS Group (ConfigurationDesk Real-Time Implementation Guide 🕮).

Variable groups from the behavior model In the TRC file, the variables of the behavior model are organized in the following groups:

- Model Root
- Tunable Parameters
- State Machine Data
- Labels
- BusSystems
- User Variables from <model>_usr.trc

For more information on the group contents, refer to Inclusion of Variables from the Simulink Model into the TRC File on page 237.

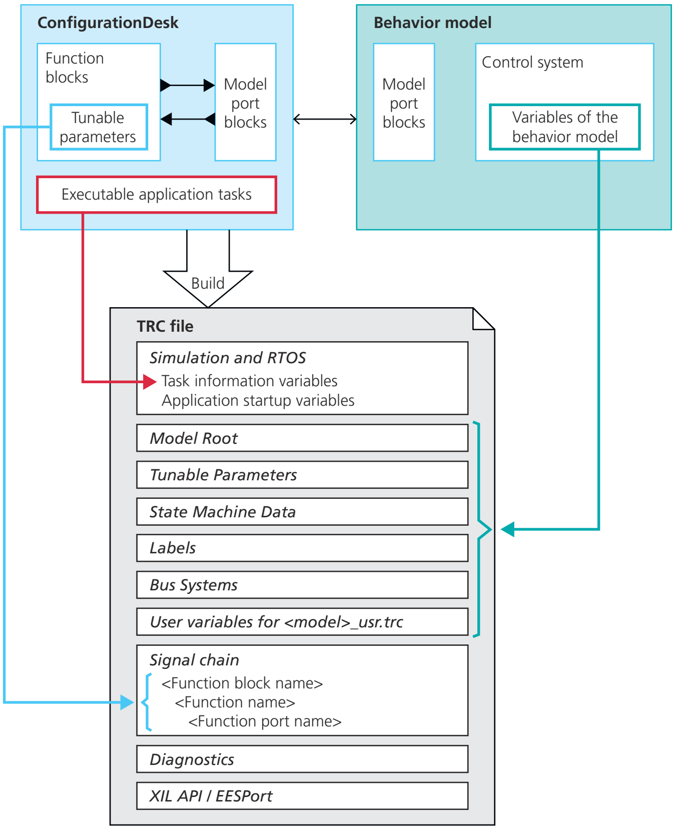
Signal Chain The Signal Chain group contains all the variables of the I/O functionality in ConfigurationDesk. It also contains variables of function ports that are configured for test automation support. If you enabled test automation support for a function port, the generated variable description file contains additional variables that can be accessed by the experiment and test software. These variables are generated in a subgroup of the affected function block. Refer to Signal Chain Group (ConfigurationDesk Real-Time Implementation Guide 🕮).

Diagnostics The Diagnostics group contains variables related to electronic fuses and the failure simulation components of the target system. Refer to Diagnostics Group (ConfigurationDesk Real-Time Implementation Guide 🕮).

XIL API/EESPort (Available as of dSPACE Release 2016-A) The XIL API/EESPort group contains variables for monitoring the switching behavior of electrical error simulation hardware. Refer to XIL API/EESPort Group (ConfigurationDesk Real-Time Implementation Guide 🕮).

Graphical overview

The following illustration gives an overview of the TRC file variables of a real-time application:



For multicore real-time applications and multiprocessing unit applications ConfigurationDesk creates one TRC file for each application process.

Inclusion of Variables from the Simulink Model into the TRC File

Simulink variables

The TRC file contains information on the variables and signals of a Simulink model.

Where to go from here

Information in this section

Details on the Variable Groups of the Behavior Model

Introduction

The variables of the behavior model are organized in the following groups, which are described in detail below:

- Model Root
- Tunable Parameters
- State Machine Data
- Labels
- BusSystems
- User Variables from <model>_usr.trc

Note

The generation of groups and their contents are influenced by the settings on the DSRT variable description file options page of a model's Configuration Parameters dialog and the contents of a model. Note that a group might be empty or omitted.

Model Root

This group contains the block and subsystem groups and the labeled signals for the top level of the Simulink model. The hierarchy of the block and subsystem groups in the Model Root group reflects the hierarchy of the Simulink model. Variables from the blocks and subsystems are located on lower hierarchical levels.

- Each subsystem group contains the subsystem's outputs, and the block groups for the blocks within the subsystem. Masked subsystems might contain parameters.
- Each block group contains the variables of one block, for example, outputs and parameters.

- Block groups for Stateflow® charts contain the outputs to Simulink, Stateflow test points and parameters.

  All Stateflow charts together form the state machine, which can have global data. This data is available via the **State Machine Data** group (see below).

Tunable Parameters

This group contains the global parameters of the behavior model.

Note

- MATLAB workspace variables and Simulink.Parameter objects, which are used as block parameters in the model, are generated as global variables in the **Tunable Parameters** group. Internal optimizations during code generation might be the reason that a variable will not be generated into the variable description.
- Structured workspace variables and Simulink.Parameter objects that are used as block parameters in the model are generated as global structured parameters in the **Tunable Parameters** group.

  The structure has to fulfill the Simulink Coder conditions for a tunable structured parameter.
- For model referencing hierarchies, a **Tunable Parameters** group is generated only for the top-level model. Global parameters referenced in referenced models are also generated into the **Tunable Parameters** group of the top-level model.

State Machine Data

This group contains data objects of Exported scope defined at the State Machine level. The group is created only if the model contains Stateflow charts. Data objects of Output, Parameter or Local scopes defined for individual Stateflow charts are available via the **Model Root** group. Depending on the MATLAB Release used, Data objects of **Local** and **Output** scopes are generated into the TRC file only if the value attribute **Test point** is set.

Labels

This group contains all labeled signals. It appears only if labeled signals are found in the Simulink model. In this group, the model hierarchy is ignored to give quick access to important signals. The labeled signals are also available in the subsystem groups from which they originate.

This group is only available if the **Include signal labels** checkbox on the on the **DSRT variable description file options** page of a model's **Configuration Parameters** dialog is selected.

Multiple occurrences of labels with the same name As of dSPACE Release 2019-B, multiple labels with the same name no longer have an entry in the TRC file by default. If required, you can temporarily activate TRC file entries for multiple labels.

Type the following commands in the MATLAB Command Window:

- To activate the option

```
ds_trc_multiplelabeloccurrence('set', 1)
```

- To reset the option to the default behavior

```
ds_trc_multiplelabeloccurrence('set', 0)
```

Note

- We strongly recommend not to activate this option.
  Using multiple labels with the same name in your variable description is on your own risk.
- The option to activate TRC file entries for multiple labels will no longer be available in one of the next dSPACE Releases. It will be available only temporarily for migrating existing models or scripts that handle multiple labels.

BusSystems

This group contains all the bus systems modeled in the Simulink model. It contains all the signals which are sent or received by the dSPACE real-time hardware.

CAN bus For details on the BusSystems group structured for a CAN bus, refer to TRC Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖).

FlexRay bus For details on the BusSystems group structured for a FlexRay bus, refer to Using the Generated TRC File of PDU-Based Modeling in ControlDesk on page 189.

LIN bus For details on the BusSystems group structured for a LIN bus, refer to TRC Options Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference 📖).

User Variables from <model>_usr.trc

This group is filled with the user-specific contents of the user variable description file (`<model>_usr.trc`) whenever code is generated from the Simulink model. It is created only if a user variable description file is available in the working folder of the model.

Rules and Limitations

There are certain rules and limitations regarding the inclusion variables from the Simulink model. Refer to the following topics:

- Conditions for the Inclusion of Variables from the Simulink Model into the TRC File on page 247
- Conditions for the Inclusion of Variables From Referenced Models on page 248
- Limitations Regarding Simulink Variables in the TRC File on page 286

Methods of Including Variables of the Behavior Model in the TRC File

Introduction

There are different options for adapting the contents of the TRC file.

Simulink options and their effects on the TRC file

Simulink provides the following code optimization options in the Model Configuration Parameters dialog to reduce memory consumption:

Default parameter behavior If you set the Default parameter behavior option in the Code Generation ⇒ Optimization dialog to Inlined, the numerical values of block parameters are used in the generated code. In this case, Simulink Coder does not generate any global variables for the block parameters, unless you specify them as tunable parameters or create Simulink.Parameter objects with a global storage class for them.

Signal Storage Reuse If you activate the Signal Storage Reuse option in the Simulation Target dialog, Simulink Coder does not always use a separate variable for each block output but attempts to assign a single buffer to several block outputs. This means that in general, the signals are not available in the TRC file unless you make specific settings for them.

Adapting parameter and signal entries in the TRC file

You can generate variables for block parameters or signals in the TRC file even if Default parameter behavior is set to Inlined and/or Signal Storage Reuse is activated.

Specifying which parameters are available in the TRC file When you set the Default parameter behavior optimization option to Inlined there are methods for you to specify block parameters that must nevertheless be available in the TRC file (including their names and properties). Set Default parameter behavior to Inlined and use one of the following methods:

- Create variables in the MATLAB workspace, and declare them as tunable parameters via Simulink's Model Parameter Configuration dialog.
- Create Simulink.Parameter objects with the ExportedGlobal, ImportedExtern, ImportedExternPointer, or ModelDefault storage class in the MATLAB workspace, and reference them in the model. For details, refer to Rules for Simulink.Signal and Simulink.Parameter objects on page 247. You must use this method for referenced models.

In both cases, the block parameter entries are displayed in the Tunable Parameters group of the generated TRC file. The complete model hierarchy is still available in the Model Root group.

The block groups within the Model Root group contain references to the global parameters in the Tunable Parameters group.

Specifying which signals are available in the TRC file To make sure that an entry for a block output signal is generated into the TRC file, even if the Signal Storage Reuse option is enabled, you can use one of the following methods:

- Enable the Test Point option for the signal.
- Specify a label and a global storage class for the signal.
- Specify a label for the signal and create an appropriate Simulink.Signal object with the ExportedGlobal, ImportedExtern, ImportedExternPointer, or ModelDefault storage class in the MATLAB workspace (for details, refer to Rules for Simulink.Signal and Simulink.Parameter objects on page 247). In addition, you must link the signal to the Simulink.Signal object. To do so, you must specify one of the following Simulink options:
  - Either select the Signal must resolve to Simulink.Signal object checkbox in the signal's Signal Properties dialog.
  - Or set the Signal resolution option to Explicit and implicit on the Data Validity page of the Diagnostics dialog.

Displaying TRC file entries as a flat list

With Simulink's code optimization options described above, the model hierarchy is still available in the Model Root group of the TRC file. If you do not need the model hierarchy, you can select the Include only Simulink.Parameter and Simulink.Signal objects with global storage class option (Configuration Parameters dialog, - Code Generation - DSRT variable description file options page). With this option activated, the TRC file provides only the Labels group showing the Simulink.Signal objects, and the Tunable Parameters group showing Simulink.Parameter objects with the ExportedGlobal, ImportedExtern, or ImportedExternPointer storage class as a flat list. It does not contain the entire model hierarchy. Selecting this option can reduce the time needed for the build process.

Note

If the Include only Simulink.Parameter and Simulink.Signal objects with global storage class option is enabled, `LookupTableData` entries are not available.

Content of the TRC file groups depending on the code option settings

You can use the DSRT variable description file options to control which variables are available in the TRC file.

Include only Signal.Parameter and Simulink.Signal objects with global storage class = ON:

- The Tunable Parameters group contains entries for all Simulink.Parameter objects with a global storage class.
- The Labels group contains entries for all Simulink.Signal objects with a global storage class.

Include only Signal.Parameter and Simulink.Signal objects with global storage class = OFF:

- The Model Root group contains the model hierarchy with:
  - Entries for the available signals
  - Entries for the available block parameters
  - Further entries such as mask parameters, states, and derivatives.
- The Tunable Parameters group contains entries for all the available global parameters.
- The Labels group contains entries for all the labeled signals if Include signal labels is selected.

Note

- Amongst other factors, it depends on the Simulink Coder optimization settings, such as Signal storage reuse, Inline invariant signals, and Enable local block outputs, if a signal entry is available in the TRC file (see below). To make sure that the Simulink Coder generates a separate entry for an output signal, you can designate any signal in a model as a test point, or specify a global storage class for the signal, either directly or via a Simulink.Parameter object.
- It depends on the setting of the Default parameter behavior option, if a block parameter is available in the TRC file. If Tunable is selected, the block parameters are generated as variables. If Inlined is selected, the numeric values of the block parameters are used in the generated C code, which makes them non-modifiable. However, you can use Simulink.Parameter objects or configure a variable as tunable in the Model Configuration Parameters dialog to make individual block parameters available in the TRC file.

For more information, refer to the Simulink and Simulink Coder documentation.

Adapting the Generation of the Variable Description File

Introduction

You can adapt the contents of the variable description file generated by ConfigurationDesk by adding and excluding variables using specific options. You can specify the variable description file options in the behavior model's Configuration Parameters dialog on the Code Generation - DSRT variable description file options page.

Note

The System target file parameter must be set to `dsrt.tlc` or `dsrt64.tlc` on the Code Generation page.

Variable description file options in Simulink

The following options are available for the DSRT system target:

Variable description file format Lets you select a format for the variable description file that is created during the build process. The following formats are available:

- TRC
- A2L

Note

The following options only apply to the generation of TRC files.

Include only Simulink.Parameter and Simulink.Signal objects with global storage class Indicates that only parameters and signals are included in the variable description file which reference a `Simulink.Parameter` or a `Simulink.Signal` object in the MATLAB workspace. For more information, refer to Methods of Including Variables of the Behavior Model in the TRC File on page 241.

In the variable description file, Description, DocUnits, Min and Max entries of `Simulink.Signal` objects and `Simulink.Parameter` objects are available.

Note

When you use model referencing, this setting must be the same for all referenced models.

Include signal labels Indicates that the signal labels of the model are available in the variable description file.

- If selected, the signal labels of the model are generated into the variable description file.
- If cleared, the signal labels of the model are not generated into the variable description file. For large models especially, this might significantly reduce the time needed for the code generation process (default).

Include virtual blocks Indicates that the block outputs of virtual blocks (for example, From blocks) are available in the variable description file.

- If selected, the virtual blocks are available in the variable description file.
- If cleared, the virtual blocks are not available in the variable description file. This might significantly reduce the time needed for the code generation process (default).

If you just want to have the outputs of a few virtual blocks available in the variable description file, you can add test points to the relevant signals.

Include states Lets you make the states of blocks available in the variable description file.

- If selected, the block states are generated into the variable description file.
- If cleared, the block states are not generated into the variable description file (default).

Include derivatives Lets you make the derivatives of blocks available in the variable description file.

- If selected, the derivatives of blocks are generated into the variable description file.
- If cleared, the derivatives of blocks are not generated into the variable description file (default).

Note

The Include states and Include Derivatives options are not supported for model referencing, neither for top-level models nor for referenced ones.

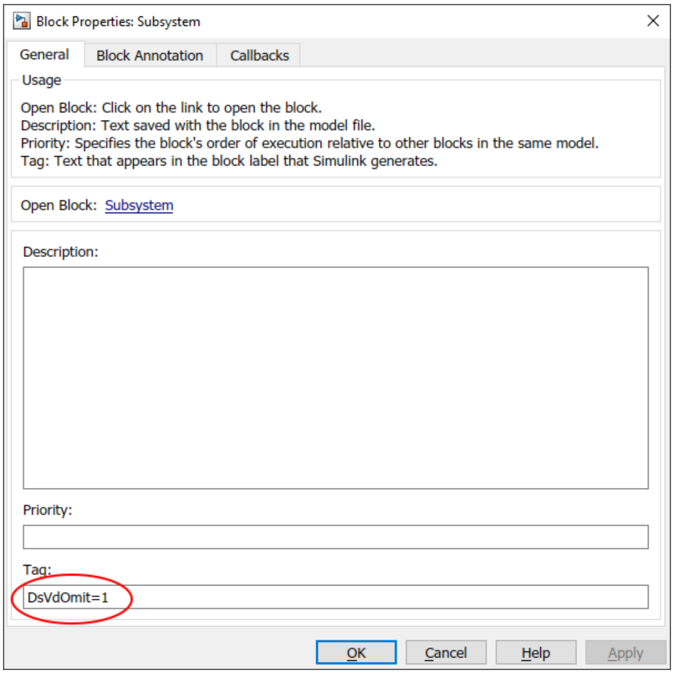
Include initial parameter values Lets you make the initial parameter values available in the variable description file. Initial parameter values are needed to carry out offline calibration tasks.

- If selected, the initial parameter values are generated into the variable description file.
- If cleared, the initial parameters are not generated into the variable description file.

Note

Deactivating this option might significantly reduce the time needed for code generation.

Apply subsystem omission tags Indicates that `DsVdOmit` tags are evaluated. These tags can be used to exclude the variables of specific subsystems from being generated into the variable description file.



Open Block: Click on the link to open the block.
Description: Text saved with the block in the model file.
Priority: Specifies the block's order of execution relative to other blocks in the same model.
Tag: Text that appears in the block label that Simulink generates.

Open Block: Subsystem

Description:

Priority:

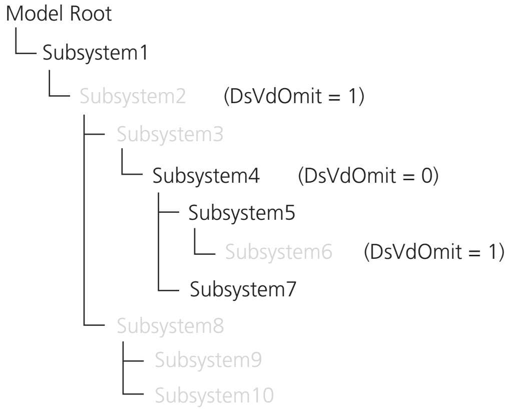
Tag: DsVdOmit=1

If the Apply subsystem omission tags checkbox is cleared, settings of the `DsVdOmit` tag are ignored for all subsystems in the model. All subsystems and their contents appear in the generated variable description file. If selected, `DsVdOmit` tags of subsystems have the following effects on the generated variable description file:

Setting	Description
`DsVdOmit` tag is set to 1	The subsystem contents including all blocks beneath this subsystem do not appear in the generated variable description file. Use `set_param(gcb,'Tag', 'DsVdOmit=1').`
`DsVdOmit` tag is set to 0	The subsystem contents including all blocks beneath this subsystem do appear in the generated variable description file, even if a subsystem above this subsystem has set the `DsVdOmit` tag to 1. Use `set_param(gcb,'Tag', 'DsVdOmit=0').`
`DsVdOmit` tag is set to -1	The subsystem contents including all blocks beneath this subsystem do not appear in the generated variable description file. `DsVdOmit` settings of subsystems included in this subsystem are ignored. Use `set_param(gcb,'Tag', 'DsVdOmit=-1').`

Exclusion is applied recursively through the model hierarchy. To re-include subsystems, you can enter `DsVdOmit=0`. You can set the Omit flag by using workspace variables. For example, if WSVar1 is a workspace variable, you can use `set_param(subsystemHandle,'Tag','DsVdOmit=$(WSVar1)')` to let the Omit tag value being evaluated during build process.

Example: If you set the Apply subsystem omission tags option, the variables of the grayed subsystems in the illustration below are not generated into the variable description file.



Adding custom code variables to the variable description file

If you use custom code, the variables in it are initially not accessible to the experiment software. To make the global variables accessible, you must provide an additional variable description file. When writing a user variable description file, use the syntax described in ConfigurationDesk Syntax of the TRC File 🕮, and name it `<model>_usr.trc`. During the build process, the user file is inserted into the main variable description file. It must be created before the build process is started. It has to be located in the working folder of the behavior model. You can write variable description files also for each referenced model in your behavior model.

Conditions for the Inclusion of Variables from the Simulink Model into the TRC File

Introduction

Not all variables of a Simulink model are always available in the TRC file. The following conditions define whether a variable and its properties are included.

Rules for Simulink.Signal and Simulink.Parameter objects

`Simulink.Parameter` and `Simulink.Signal` objects can be used to specify the characteristics of tunable parameters and labeled signals in a model.

Parameters and signals specified by `Simulink.Parameter` and `Simulink.Signal` objects are available in the TRC file by default. `Simulink.Parameter` and `Simulink.Signal` objects are represented in the variable description file with the following properties:

- Description
- DocUnits

  Is displayed as *Unit* in your experiment software.
- Min and Max (represented by a range in the TRC file)

Note the following preconditions and restrictions:

- The properties for labels in the Labels group and the subsystem groups are used, if the labeled signal references a `Simulink.Signal` object of global storage class (ExportedGlobal, ImportedExtern, ImportedExternPointer). At the entries for the block output signals, these properties are omitted.

  > **Note**
  >
  > The Include Labels checkbox must be selected. It is not selected per default.

- The properties for parameters in the Tunable Parameters group are used, if the parameters are specified by `Simulink.Parameter` objects of global storage class (ExportedGlobal, ImportedExtern, ImportedExternPointer).
- The range for Min and Max is generated only if both values are available.
- The range for Min and Max is not generated for fixed-point data.

Rules for virtual Simulink blocks

Virtual blocks are not available in the TRC file by default. If you want to include them, you can select the *Include virtual blocks* option. For details on this option refer to Adapting the Generation of the Variable Description File on page 243.

Rules for virtual signals

Entries for virtual signals, e.g., from a Mux block or a Bus Creator block, are not generated into the TRC file.

Rules for structured variables

Structured variables, such as non-virtual buses or tunable structured parameters, are generated into the code and represented in the variable description as a `struct` element.

Rules for complex variables

Code generated with Simulink Coder stores the real and imaginary parts of complex variables separately. The display of the variable dimensions in ControlDesk is different from the one displayed in MATLAB or Simulink, since the TRC file generator converts a complex number into a 2-dimensional vector: Twice as many rows or columns are displayed in comparison to the Simulink model. This applies also to complex data in the fields of a structured variable.

Conditions for the Inclusion of Variables From Referenced Models

Introduction

When you use model referencing there are important specifics regarding the inclusion of variables in the TRC file.

Specifics for model referencing

The following list shows the specifics regarding model referencing:

- The TRC file contains the variables of the top-level model and of all the referenced models. The variables for the referenced models are stored in groups in a similar way to variables of subsystems.
- Parameters defined in the model workspace of referenced models can be accessed via the **Model Parameters** group of the referenced model. Block-level references to these parameters can be accessed via the referenced model group of the respective model.
- The TRC file information for a referenced model is rebuilt only if new code is generated for the model. Incremental code generation thus also includes incremental TRC file generation.

- Whether variables of a referenced model are written to the TRC file depends on the setting of the **Total number of instances per top model** option in the **Model Referencing** dialog. If the total number of instances is set to
  - "One", the contents of the referenced model appear in the TRC file.
  - "Multiple" (default), the contents of the referenced model do not appear in the TRC file. This means that such a model can be used to build real-time applications, but you cannot access internal variables for the model via the TRC file.

  For details, refer to the Simulink online help by MathWorks.
- Variables of Inport blocks located on the root level of a referenced model are never generated into the TRC file. This also applies to the outputs of virtual blocks (Mux, Demux, Goto, From, subsystems), which are directly connected to such an Inport block.
- It can happen that block outputs of blocks which drive an Outport block located on the root level of a referenced model are not generated. To solve this problem, a test point can be set for the respective signal.
- Like subsystem outputs, Model block outputs are also contained in the TRC file.
- The following applies to the entries for block parameters of referenced models:

  If the **Default parameter behavior** of the referenced model is set to **Tunable**, the tunable parameters are available:
  - In the **Tunable Parameters** group, if they are defined globally.
  - In the **Model Parameters** group, if they are defined in the model workspace.
  - Directly at the block, if they are defined locally.

  If the **Default parameter behavior** of the referenced model is set to **Inlined**, parameters are available in the TRC file, which are specified as tunable via *Simulink.Parameter* objects .
- Consider the following regarding the use of options on the **DSRT variable description file options** page:
  - The **Include states** and **Include Derivatives** options are not supported for model referencing, neither for top-level models nor for referenced ones. If you select these options you are informed during the build process that they are ignored.
  - All the other options on the **DSRT variable description file options** page are supported for top-level as well as for referenced models.

    For details on the above-mentioned options, refer to Adapting the Generation of the Variable Description File on page 243.
  - The setting for the **Include only Simulink.Parameter and Simulink.Signal objects with global storage class** option must be the identical for all models in a Model Referencing hierarchy.

Preparing Your Model for Test Automation

Introduction

ConfigurationDesk allows you to include stimulus signals in your real-time application for test purposes. You can enable a switch in the signal chain for replacing the original signal of a function port with a stimulus signal.

> **Note**
>
> As of dSPACE Release 2013-B, the Test Automation Blockset is not provided on the DVD. Only ConfigurationDesk lets you include stimulus signals in your real-time application.

General Information on Test Automation

Introduction

You can interactively modify variables of your real-time application to check its behavior using your experiment software. If you want to automate these tests, your test automation software, for example, AutomationDesk, requires platform variables to feed certain input and output signals of your real-time application with stimulus signals. In ConfigurationDesk, you can enable a switch in the signal chain for replacing the original signal of a function port with a stimulus signal.

Test automation support in ConfigurationDesk

You can enable test automation support for each function inport and function outport separately. These settings are independent of the behavior model. You can replace the behavior model without modifying stimulus signal access in your test automation tool. For further information, refer to Configuring Test Automation Support (ConfigurationDesk I/O Function Implementation Guide 🕮).

If you have enabled test automation support for a function port, the generated variable description file contains four additional variables that can be accessed by the experiment and test software:

- IO_Signal
- MDL_Signal
- TA_Switchvalue
- TA_Replacevalue

Demo Models for Special Function Block Types

Introduction

There are sensors/actuators which cannot be fully simulated via function block types in ConfigurationDesk. Some characteristics of the I/O functionality must be simulated in the behavior model for maximum flexibility. Demos give you an example of how to implement these characteristics in the behavior model.

You can use the demos as templates and copy them to your behavior models.

Demo Model of a Wheelspeed Out Interface

Purpose

The following characteristics of the wheel speed sensors' data protocol must be simulated in the Behavior model:

- The logical coding.

  The logical coding sets the number of data bits and describes the meaning of each data bit
- The physical coding.

  The physical coding describes how each data bit is physically transmitted to the external device via data pulses. For details, refer to Encoding the Data Protocol with the Wheelspeed Out Function Block (ConfigurationDesk I/O Function Implementation Guide).
- The reduction of the data protocol if the time between two speed pulses is less than the time needed to transmit all the data pulses. This might be the case if the wheel rotates very fast. For details, refer to Notes on the Data Protocol for Fast Rotating Wheels (ConfigurationDesk I/O Function Implementation Guide).
- The specification of a gap to separate the speed and data output.

  Whether you need to specify a gap depends on the wheel speed sensor that is simulated.

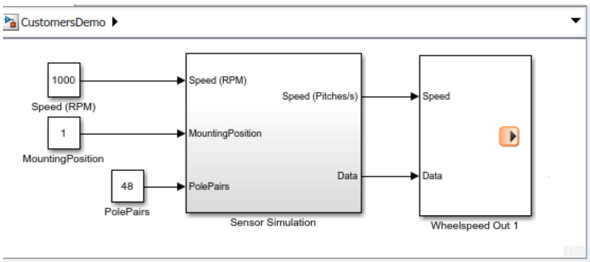
Location of files

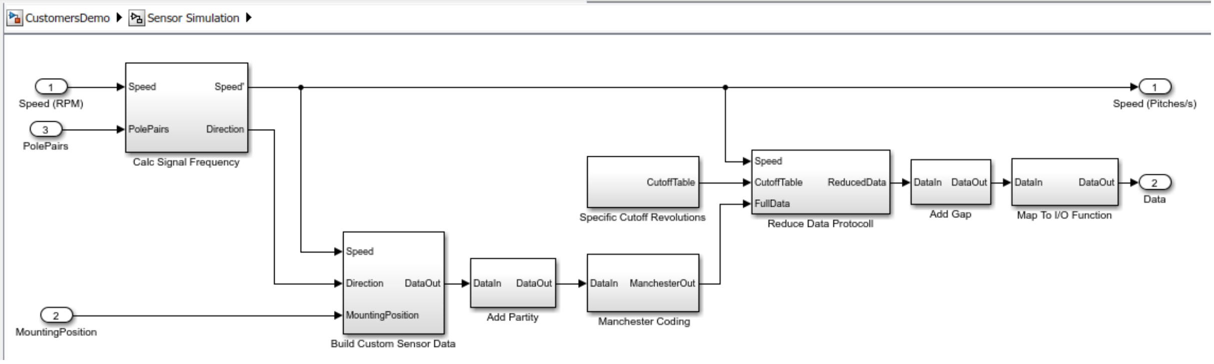
The demo is available in the project root folder after you first start ConfigurationDesk.

The Simulink model file (MDL file) is a part of the ConfigurationDesk demo. You can open the MDL file in the **Project Manager** of ConfigurationDesk.

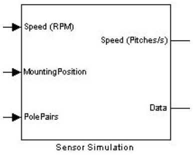
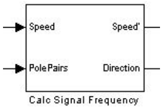
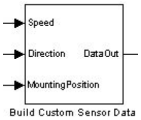
Overview

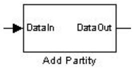
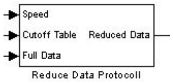
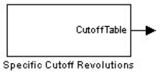
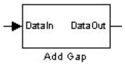
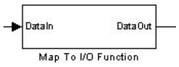
The illustration shows you the Simulink demo model together with the subsystem of the **Sensor Simulation** model block. The demo simulates a common wheel speed sensor interface with data protocol.





Description of the model The following table describes the functionality of the Sensor Simulation model block and its subsystem.

Model Block	Description
Sensor Simulation  Speed (RPM) MountingPosition PolePairs Speed (Pitches/s) Data Sensor Simulation	This model block: ▪ Scales the Speed input value from revolution per minute (RPM) to pitches per second (Pitches/s) of the Speed output. ▪ Builds the data protocol of the Data output. The Speed, MountingPosition, and PolePairs inputs are the basis for the data protocol. To detect the direction of rotation, the orientation of the mounted sensor must be clear. You can specify this with the MountingPosition setting. This setting indicates whether positive speed values represent wheels rotating clockwise or anticlockwise. A setting to 1 indicates wheels rotating clockwise at positive speed values. The number of PolePairs depends on the simulated magnetically encoded wheel/ferromagnetic wheel.
Calc Signal Frequency  Speed PolePairs Speed' Direction Calc Signal Frequency	This subsystem model block: ▪ Scales the Speed input value from revolution per minute (RPM) to pitches per second (Pitches/s) of the Speed output. ▪ Sets the Direction output value. A setting to 1 indicates negative speed values.
Build Custom Sensor Data  Speed Direction MountingPosition DataOut Build Custom Sensor Data	This subsystem model block: ▪ Sets the first 8 bits of the logical coding. The DataOut output is a vector with 8 bits with the following logical coding in the demo model: ▪ Bit 0 - Bit 2 are status bits. Each bit is set to 0 in the demo. ▪ Bit 3 indicates whether the detected direction of rotation is valid. A setting to 1 indicates a valid detection. The detected direction is valid if the detection is reliable. The detection is reliable if the following requirements are all fulfilled: ▪ The current speed is below the maximum speed limit.

Model Block	Description
	The maximum speed limit is set to 1600 pitches/s in the demo. ▪ The air gap between the wheel and the measuring bridge is below the maximum air gap limit. Within the **Build Custom Sensor Data** subsystem model block, the air gap is set to a fixed value less than the maximum air gap limit. ▪ Bit 4 represents the direction of rotation. A setting to 1 indicates wheels rotating clockwise at positive speed values. The value is inverted if the **MountingPosition** input of the **Sensor Simulation** model block is set to 0. ▪ Bit 5 - Bit 7 represent the air gap. Each bit is set to 1 in the demo. The logical coding is set by the simulated sensor. The **DataOut** vector size can be reduced or increased as required. The connected model blocks do not limit the **DataOut** vector size.
Add Parity  	This subsystem model block: ▪ Adds a parity bit to the **DataIn** vector. In the demo, the subsystem model block adds an even parity bit.
Manchester Coding DataIn ManchesterOut Manchester Coding	This subsystem model block: ▪ Transforms the logical coded **DataIn** vector to the physical Manchester coding. For details, refer to Encoding the Data Protocol with the Wheelspeed Out Function Block (ConfigurationDesk I/O Function Implementation Guide 📖).
Reduce Data Protocol  Speed Cutoff Table Reduced Data Full Data Reduce Data Protocoll	This subsystem model block: ▪ Reduces the data protocol for fast rotating wheels. The **Speed** and **Cutoff Table** inputs are the basis for the reduction. The **Cutoff Table** is a look-up table with cutoff speed values (see **Specific Cutoff Revolutions**). More or fewer bits of the data protocol are cut according to the **Speed** input value.
Specific Cutoff Revolutions  	This subsystem model block: ▪ Provides a look-up table for the **Reduce Data Protocol** block.
Add Gap  	This subsystem model block: ▪ Adds a gap before and after the data protocol. This gap separates the data output and the speed output when they are combined in one serial protocol by the **Wheelspeed Out** function block in ConfigurationDesk.
Map to I/O Function  	This subsystem model block: ▪ Maps the **DataIn** vector size to the **Data** inport vector size of the **Wheelspeed Out** function block in ConfigurationDesk. Notation of the **DataOut** vector: ▪ All 64 data pulses of the data protocol are written in a vector with 8 elements. For example: {90;103;255;0;0;0;0;0}. ▪ The decimal number of an element is a binary number with 8 digits, each of which stands for a pulse. For example: $\{90;...\}_{decimal} = \{01011010;...\}_{binary}$. A 1 represents the pulse level that is specified by the **Data pulse current** property. A 0 represents the pulse level that is specified by the **Quiescent current** property. The least significant digit represents the first output pulse, the most significant digit represents the last output pulse. ▪ The first element represents the first 8 output pulses, the second element the next 8 pulses, etc. in the decimal notation. ▪ There must be 8 vector elements, unused data pulses must be set to 0.

Simulation of sensors without data protocol

If you want to simulate an active wheel speed sensor without data protocol, you can reduce the Sensor Simulation model block subsystem to the following model blocks:

- Calc Frequency model block
- Map to I/O Function model block

  All elements of the DataOut vector must be set to 0.

Model Port Block Behavior During Simulink Simulation and Real-Time Execution

Introduction

The behavior of the model port blocks in a Simulink simulation of the behavior model differs from their behavior when the real-time application is executed on the hardware.

Where to go from here

Information in this section

General Information on Execution Modes

Introduction

Working with a behavior model allows you to simulate its behavior in the Simulink environment. Because the real-time hardware, the functions and the behavior model are separate, the simulation in Simulink can only use the behavior model. There is no access to the functions or the real-time platform. This differs from executing the real-time application on the real-time hardware. To evaluate a Simulink simulation, you must know the differences.

Simulink simulation

Simulink provides three simulation modes:

- Normal

  Simulation with interpreted model code
- Accelerator

  Simulation with compiled target code
- Rapid Accelerator

  Simulation with stand-alone executable

For information on the simulation modes, refer to the user documentation from MathWorks®. There are some overall limitations you must note when performing accelerated simulations.

All of these simulation modes are supported by behavior models that contain model port blocks. To predict the behavior of a model, you must know the behavior of the model port blocks.

For further information, refer to Behavior of Data Ports on page 256 and Behavior of Hardware-Triggered Runnable Function Blocks on page 258.

Real-time execution

If you execute the real-time application on the real-time platform, the model port blocks get their signals from the connected hardware via the functions. If the ConfigurationDesk application is erroneous, for example, a model port block is unresolved, the conflicts are reported when you build the real-time application. In some cases, for example, if a function has ambiguous trigger sources, the generated code contains default instructions without accessing the hardware. To predict the behavior of the real-time application, you must know the behavior of the model port blocks.

For further information, refer to Behavior of Data Ports on page 256 and Behavior of Hardware-Triggered Runnable Function Blocks on page 258.

Related topics

Basics

Details on the Build Process (ConfigurationDesk Real-Time Implementation Guide)

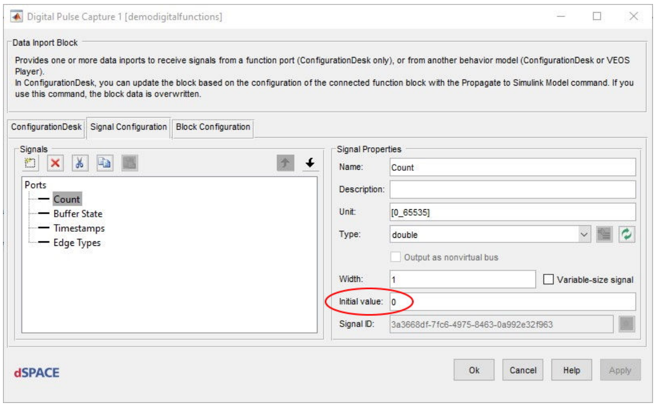
Behavior of Data Ports

Introduction

Data port blocks in your behavior model provide the interface between the behavior model and the I/O functionality in ConfigurationDesk. The data ports behave differently in Simulink simulation and in execution of the real-time application.

Behavior of data ports during Simulink simulation

In Simulink simulation mode, the port of a Data Inport block outputs the specified initial value.



Digital Pulse Capture 1 [demodigitalfunctions]

Data Inport Block

Provides one or more data inports to receive signals from a function port (ConfigurationDesk only), or from another behavior model (ConfigurationDesk or VEOS Player).
In ConfigurationDesk, you can update the block based on the configuration of the connected function block with the Propagate to Simulink Model command. If you use this command, the block data is overwritten.

Name: Count
Description:
Unit: [0_65535]
Type: double
□ Output as nonvirtual bus
Width: 1 □ Variable-size signal
Initial value: 0
Signal ID: 3a3668df-7fc6-4975-8463-0a992e32f963

A port of a Data Outport block receives the simulated signal, but it has no further functionality.

The data types and the widths of data port blocks must be compatible with the behavior model.

Behavior of data ports during real-time execution

During real-time execution, the behavior of data ports depends on their mapping status. It is not enough for the behavior model to be analyzed by ConfigurationDesk. You must also guarantee that the signal chain is implemented and configured completely. The build process generates warning messages for several problems and conflicts, but it creates an executable real-time application.

You must be aware of the following situations to evaluate the data port behavior:

- Real-time application has been built without warnings.
  - Ports of a Data Inport block transmit the values from the mapped function ports to the model.

    Note that they deliver the specified default values until the block has been executed the first time, for example, if they are used within an enabled subsystem.
  - Ports of a Data Outport block transmit the values from the model to the mapped function ports.
- Real-time application contains model port blocks that are used for model communication.
  - If the Data Inport block's Protocol property is set to Blocking, the following applies:

    The Data Inport block always provides the actual value provided from the connected Data Outport block.
  - If the Data Inport block's Protocol property is set to Non-Blocking, the following applies:

  The **Data Inport** block provides the default value until the **Data Inport** block receives a value from the connected **Data Outport** block for the first time. After that, the **Data Inport** block always provides the most recent value provided by the connected **Data Outport** block.

  For details, refer to Setting Up Model Communication (ConfigurationDesk Real-Time Implementation Guide 🕮).

- Real-time application contains breaks within the configured signal chain (data port is not mapped to a function port, or the hardware is not assigned to the function block).
  - Ports of a **Data Inport** block deliver the specified default value.
  - Ports of a **Data Outport** block receive signals from the model, but they have no further functionality.
- Real-time application contains mapped data port blocks that are not available in the behavior model
  - The real-time application will only contain code for initializing and terminating the functions, but no read or write access to the configured data ports.

Related topics

Basics

Behavior of Hardware-Triggered Runnable Function Blocks

Introduction

Hardware-Triggered Runnable Function blocks in your behavior model export a function-call subsystem as a runnable function for modeling asynchronous tasks in ConfigurationDesk. The function-call subsystem triggered by the **Hardware-Triggered Runnable Function** block is treated differently in running a Simulink simulation and in executing the real-time application.

Behavior of Hardware-Triggered Runnable Function blocks during Simulink simulation

A **Hardware-Triggered Runnable Function** block is used to trigger a function-call subsystem asynchronously. In Simulink simulation mode, the subsystem is triggered periodically with the sample time specified for the **Hardware-Triggered Runnable Function** block.

Behavior of Hardware-Triggered Runnable Function blocks during real-time execution

During real-time execution, the runnable function is executed within the task it is assigned to. It is not enough for the behavior model to be analyzed completely by ConfigurationDesk. You must also guarantee that the executable application is modeled and configured completely. The build process generates warning messages for several problems and conflicts, but it creates an executable real-time application.

You must be aware of the following situations to evaluate the treatment of the runnable function:

- Real-time application has been built without warnings.
  - The function-call subsystem is triggered by the I/O event that is assigned to the same task as the runnable function.
- The runnable function is assigned to a task, but the task does not have an event assigned to it.
  - The related task is never triggered and thus, the runnable function is never executed.

    A conflict is shown in the **Conflicts Viewer** before you start the build process and you will get a warning when you build the real-time application.
- The runnable function is not assigned to a task.
  - The runnable function is never executed.

    A conflict is shown in the **Conflicts Viewer** before you start the build process.

Related topics

Basics

Behavior of Data Ports..256

Generating Simulink Implementation Containers

Introduction

The Model Interface Package for Simulink lets you generate Simulink implementation container files (SIC files) that you can add to a ConfigurationDesk application or import to VEOS Player. You can also generate SIC files that include A2L file fragments for the parameters and signals of a Simulink model. You can use such SIC files in VEOS Player.

Where to go from here

Information in this section

Working with Simulink Implementation Containers

Introduction

When you work with Simulink implementation containers, you should be familiar with the contents of SIC files and the workflow for generating them. In some cases, it might be necessary to add external files to the SIC file. The Model Interface Package for Simulink provides specific API commands for this purpose.

Where to go from here

Information in this section

Basics on Simulink Implementation Containers

Description of a Simulink implementation container

The Model Interface Package for Simulink lets you generate a Simulink implementation container file (SIC file) for your model. The SIC file is a container file containing the model code of a Simulink behavior model. If your model code requires files that are not known to Simulink Coder as dependencies, you can add them to a Simulink implementation container via an API command. The Simulink implementation container can be used in ConfigurationDesk or in VEOS Player.

Note

For this to be possible, be careful when using static libraries: e.g., for precompiled S-functions, or when using the Custom Code feature of Simulink Coder. Consider using the `dsrt_addfiles()` API command (refer to dsrt_addfiles (Model Interface Package for Simulink API Reference)) to define which static libraries are suited for a specific target platform. Note that the generation of Simulink implementation containers uses the packNGo feature of Simulink Coder, and is hence subject to its limitations.

Benefits of Simulink implementation containers

With a Simulink implementation container file (SIC file), you can separate the model code generation from the build process. This approach has the following benefits:

- You can use the same system target file for Simulink behavior models in VEOS Player (running on Windows) and in ConfigurationDesk.

- A MATLAB installation is necessary only for building SIC files. For using of SIC files in ConfigurationDesk or VEOS Player, a MATLAB installation is not required.
- Executable applications can contain Simulink implementation containers from different MathWorks Releases and different dSPACE Releases.
- In ConfigurationDesk, you can precompile an SIC file to save time in the build process. Optionally, you can create precompiled SIC files without readable source files. This can be useful for IP protection.

Different methods for generating an SIC file

The Model Interface Package for Simulink provides the following methods for generating an SIC file:

- By executing the Build command (`Ctrl` + `B`) for the model.
- By using the `dsrt_build()` API command.

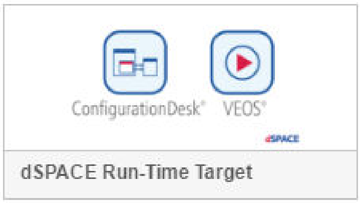
Note

For both methods, the dSPACE Run-Time Target must be specified as the system target file.

For a detailed description, refer to How to Generate a Simulink Implementation Container on page 264.

Using the DSRT model template

The Model Interface Package for Simulink provides a template for Simulink models that is preconfigured for the dsrt.tlc system target file. You can create new Simulink models on the basis of the template that is accessible via the Simulink start page.



The following applies to Simulink models based on the DSRT template:

- General options for DSRT code generation are set.
- On the **Callbacks** page of the **Model Properties** dialog, the model callbacks are configured for the dsrt.tlc system target file.

Note

You cannot create model templates with blocks from the Model Interface Blockset.

Contents of an SIC file

An SIC file contains the model code and any external dependencies. It also contains the required static code from the MATLAB/Simulink installation that was used to generate the SIC file, and a description of the model interface.

Variable descriptions in SIC files

During SIC file generation, a variable description is generated as a part of the SIC file. Depending on your settings, the variable description can be one of the following:

- A TRT file
- An A2L file

TRT file A TRT file is an intermediate, platform-independent variable description file. During the build process, ConfigurationDesk or VEOS Player creates platform-specific variable addresses into that file and safes it as a TRC file. For details on TRC files, refer to Adapting the Generation of the Variable Description File on page 243.

A2L file By default, the Model Interface Package for Simulink generates a TRT file in connection with model code generation. However, you can enable A2L file generation, if you need to. Refer to Generating A2L Files on page 271.

How to Generate a Simulink Implementation Container

Objective

You can generate a Simulink implementation container for a behavior model to add it to your ConfigurationDesk application or import it into VEOS Player.

Possible methods

The Model Interface Package for Simulink provides the following methods for generating a Simulink implementation container (SIC file):

- By using the Build command (`Ctrl` + B). Refer to Method 1 on page 264.
- By using the `dsrt_build()` API command. Refer to Method 2 on page 265.

Method 1

To generate a Simulink implementation container using the Build command (Ctrl + B)

1. In Simulink, create a behavior model.
2. In your Simulink model, open the Configuration Parameters dialog (`Ctrl` + E), and select the Code Generation page.
3. Locate the System target file edit field and click the Browse button.

   The System Target File Browser opens.

4 Depending on your use scenario, select a suitable system target file from the list. Refer to the following table:

Use Scenario	Required System Target File
SIC files for VEOS running on Windows and for ConfigurationDesk	`dsrt.tlc`
SIC files for VEOS running on Linux	`dsrt64.tlc`

5 Click OK.

6 In your model, press **`Ctrl`** + **`B`** .

Method 2

To generate a Simulink implementation container via the dsrt_build() API command

1 In the MATLAB Command Window, enter the `dsrt_build()` API command with the required parameter names and parameter values.

The following example shows how to create an SIC file for the `myModel` model and the `dsrt.tlc` system target file:

```
dsrt_build('system', 'myModel', 'generateSIC', true,
'target', 'dsrt')
```

Note

Additional function parameters are available for the `dsrt_build()` API command. You can specify a destination path for the created SIC file, for example. For more information, refer to dsrt_build (Model Interface Package for Simulink API Reference 🕮).

Result

You have generated a new SIC file from your behavior model. It is named after the model. If you generated the SIC file via Simulink Coder, it is generated to the MATLAB working directory. If you generated the SIC file via the `dsrt_build()` API command, and you specified a destination path, the SIC file is generated to the specified path.

Next steps

You can now perform one of the following actions:

- Add the SIC file to a ConfigurationDesk application. Refer to Adding Simulink Implementation Containers to a ConfigurationDesk Application (ConfigurationDesk Real-Time Implementation Guide 🕮).
- Import the SIC file into VEOS Player. Refer to How to Import Simulink Implementations (VEOS Manual 🕮).

Related topics

Basics

Adding External Files to a Simulink Implementation Container

Adding external files via API command

The Model Interface Package for Simulink lets you add external files to a Simulink implementation container (SIC file). You can use the `dsrt_addfiles()` API command to add, for example, binary files, source files, or header files to an SIC file.

Processing of different file types

If you add external files to an SIC file, it depends on the file type how the files are processed. The following table gives you an overview:

File Type	Description	Result
Binary files	Libraries, object files, precompiled files	▪ The files are copied to the SIC file. ▪ The file path is added to the makefile to link it to the real-time application.
Source files	C or C++ source code files	▪ The files are copied to the SIC file. ▪ The file path is added to the makefile to compile it and link it to the real-time application.
Header files	C or C++ source code header files	▪ The files are copied to the SIC file. ▪ The file path is added to the include paths that are used for the build process of the real-time application.
Dynamic link libraries	DLL and shared object files	▪ The files are copied to the SIC file. ▪ ConfigurationDesk downloads the files to the target platform, where they are available for the real-time application. VEOS makes sure that the files are accessible during the simulation.

Note

- You can add unspecific files by not specifying the file type. The files are copied to the SIC file and remain unchanged.
- If a specified file that you want to add does not exist, an error message is displayed and code generation is aborted.
- Files to be added can be specified individually or as a cell array.

Examples

```
dsrt_addfiles('myFile.c' 'Platform', 'SCALEXIO_LNX',
'FileType', 'SrcFile');
```

adds the `myFile.c` source file to the SIC file. `myFile.c` is used for the SCALEXIO_LNX target platform.

```
dsrt_addfiles({'myFile.txt', 'myDoc.doc'});
```

adds `myfile.txt` and `myDoc.doc` to the SIC file. The added files are used for all platforms.

For more information, refer to dsrt_addfiles (Model Interface Package for Simulink API Reference 🕮).

Adding shared objects to an SIC file

You can add shared objects that must be accessed during simulation to an SIC file. For use in ConfigurationDesk, shared objects must be added as SO files. For use in the VEOS Player, shared objects must be added as DLL files.

Examples

```
dsrt_addfiles('shared.so', 'Filetype',
'DynamicLinkLibrary', 'Platform', 'SCALEXIO_LNX');
```

adds the shared.so file to the SIC file. During simulation, the shared.so file is available on the SCALEXIO_LNX platform.

```
dsrt_addfiles('shared.dll', 'Filetype', 'DynamicLinkLibrary',
'Platform', 'VEOS_WIN64_GCC');
```

adds the shared.dll file to the SIC file. During simulation, the shared.dll file is available on the VEOS_WIN64_GCC platform.

Adding unspecific files to a specified SIC folder

You can add unspecific files, i.e., files without a file type, to an SIC file via the `Destination` parameter of the `dsrt_addfiles()` API command. The `Destination` parameter specifies the destination folder in the SIC file relatively to the root folder of the SIC file, or for files outside the code generation directory, the `otherFiles` folder of the SIC file. The files are then copied to the destination folder of the SIC file.

> **Note**
>
> The `Filetype` parameter and the `Platform` parameter must not be specified. Otherwise, the code generation aborts.

Example

```
dsrt_addfiles('myFile.c','destination','');
```

copies file myFile.c to `<root>\myFile.c`.

Specifying a platform for external files

The `dsrt_addfiles()` API command lets you specify a platform for which you want to use the external files. If no platform is specified, the external files are used with any platform. For binaries, you must always specify a platform. Otherwise, an error message is issued.

For details on available platforms, refer to dsrt_addfiles (Model Interface Package for Simulink API Reference 🕮).

Note

With dSPACE Release 2020-B, the base operating system on SCALEXIO platforms has changed from a QNX-based to a Linux-based distribution. Thus, SIC files generated with dSPACE Release 2020-A or earlier that contain binaries cannot be added to the current version of ConfigurationDesk. You must recompile the binaries for Linux and add them to the SIC files using the `SCALEXIO_LNX` platform identifier.

Where to place the API command

The `dsrt_addfiles()` API command must be called during the code generation of the SIC file. You can call the command as follows:

- In the `rtwmakecfg.m` file.

  For details, refer to the Simulink® Coder™ documentation.
- In DSRT build hook functions provided by the Model Interface Package for Simulink.

  DSRT build hook functions are M files that are called during code generation for the `dsrt.tlc` or the `dsrt64.tlc` system target file. The M files must reside on the MATLAB search path and must be named as follows:

  `<prefix>_dsrtbuildhook.m`
- In a `PostCodeGenCommand`.

  For details, refer to the Simulink® Coder™ documentation.

Note

- If you call the API command outside code generation, the Model Interface Package for Simulink issues an error message and ignores the files to be added.
- Previously, you had to use the dsrt_addnonbuildfiles() API command to add the source code directory of a generated S-function to an SIC file. The Model Interface Package for Simulink tries to detect whether your model contains generated S-functions. If an S-function is used and its name ends with "_sf" (which is the default name postfix when generating S-functions), the Model Interface Package for Simulink adds the relevant source code directory to the SIC file.
- To completely reset the behavior, delete the `slprj` folder and the `<model name>.slxc` file in the current working directory.

DSRT build hook function

Syntax To add external files to an SIC file via a hook function, you must create a function with the following syntax:

```
function [errCode, errMsg] = <prefix>_dsrtbuildhook(phase,
hookStruct)
```

Input and output parameters The DSRT build hook function has the following input parameters:

- phase: A string that specifies the code generation phase in which the function must be executed. The following table shows the possible values:

Value	Description
entry	The function is executed when the DSRT code generation has started.
before_tlc	The function is executed before the code generation by Simulink Coder starts.
after_tlc	The function is executed after the code generation by Simulink Coder has started.

- hookstruct: A struct parameter that contains the following fields:

Field name	Description
modelName	Specifies the name of the model.
modelHandle	Specifies the handle of the Simulink model.
workDir	Specifies the code output directory.
buildDir	Specifies the build directory. Build directory = `<workDir>\<modelName>_<target>`
target	Specifies the target. Possible values are dsrt or dsrt64.
GenerateSIC	Specifies whether an SIC file must be generated. ▪ True: Generate an SIC file. ▪ False: Do not generate an SIC file.

The following table shows the output parameters of the DSRT build hook function:

Output Parameter	Description
errCode	Numeric error code. 0: No error occurred. 1: An error occurred. **Note** When errCode is set to a value other than 0, DSRT code generation aborts and an error message is displayed.

Output Parameter	Description
errMsg	Error message describing the error that occurred. Empty string: No error occurred.

Example In the example below, the `mydsrtbuildhook.m` file is created with the following content:

```
function [errCode, errMsg] = my_dsrtbuildhook(phase, hookStruct)
    errCode = 0;
    errMsg = '';

    if strcmp(phase, 'before_tlc')
        dsrt_addfiles(...
            {'myFile1.c', 'myFile2.c'}, ...
            'model', hookStruct.modelName, ...
            'FileType', 'SrcFile');
    end
end
```

Note

If you do not specify the `phase` parameter, the code is executed once in each phase. However, the files are added only once to the SIC file.

Related topics

Basics

Basics on Simulink Implementation Containers..262

HowTos

How to Generate a Simulink Implementation Container..264

Generating A2L Files

Introduction

The Model Interface Package for Simulink lets you generate A2L files for the parameters and signals of a Simulink model.

Where to go from here

Information in this section

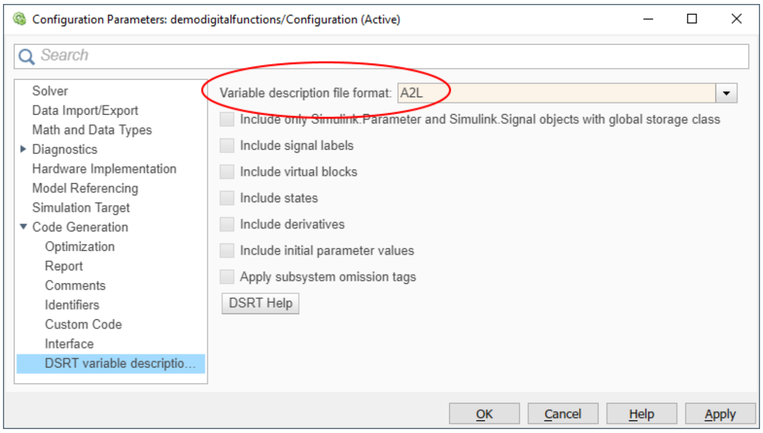
Basics on A2L File Generation

Introduction

By default, the Model Interface Package for Simulink generates a TRT file in connection with model code generation. You can, however, enable A2L file generation if you need to.

Enabling A2L file generation

To enable A2L file generation for the dsrt.tlc or the dsrt64.tlc system target file, you must specify the following setting on the DSRT variable description file options page of the Configuration Parameters dialog of your Simulink model:



Variable description file format: A2L

☐ Include only Simulink.Parameter and Simulink.Signal objects with global storage class

☐ Include signal labels

☐ Include virtual blocks

☐ Include states

☐ Include derivatives

☐ Include initial parameter values

☐ Apply subsystem omission tags

For details, refer to How to Generate an A2L File on page 272.

Note

- In the case of a model referencing hierarchy, all models must have the same configuration.
- After the code generation (in MATLAB/Simulink), the generated A2L file is not ready to use yet. It is finalized during the build process in VEOS Player or in ConfigurationDesk.

For a detailed description of A2L file generation, refer to Simulink Model A2L File Generation Manual 🕮.

How to Generate an A2L File

Objective

Generating an A2L file with Simulink Coder lets you list the variables located in the memory of the dSPACE simulation platform and make them available to measurement and calibration systems.

Preconditions

- Ensure that the dsrt.tlc or dsrt64.tlc system target file is selected. Refer to How to Generate a Simulink Implementation Container on page 264.
- Parameters and signals of a Simulink model appear in the A2L file if:
  - They are defined as `Simulink.Parameter` and `Simulink.Signal` objects in the MATLAB workspace.
  - Their `StorageClass` is set to `ExportedGlobal`.

  Tip

  To create suitable objects in the MATLAB workspace for workspace parameters and labeled signals referenced by the Simulink model, you can use the `ds_asap2paramcreate` and the `ds_asap2signalcreate` functions. See Adding Parameters and Signals to the A2L File (Simulink Model A2L File Generation Manual 🕮) for function details.

- To let signals appear in the A2L file, you also have to ensure that one of the following conditions is fulfilled:
  - Either the **Signal must resolve to Simulink.Signal object** option is selected in the signal's **Signal Properties** dialog.
  - Or the **Signal resolution** option is set to **Explicit and implicit** on the **Data Validity** page of the **Diagnostics** dialog.

Restrictions

For signals to appear in the A2L file, the following restriction applies:

Virtual signals, such as the output signals of a Mux block, are not stored in consecutive memory locations and cannot be accessed as an array of variables. For this reason, output signals of virtual blocks cannot be added to the A2L file.

Tip

To ensure that all parts of a vectorized signal are stored in consecutive memory locations, use a Signal Conversion block with the Output parameter set to Signal copy and label the Signal Conversion block output instead of the output of the virtual block.

Method

To generate an A2L file

1. From the menu bar of the Simulink model, select Code – C/C++ Code – Code Generation Options to open the Configuration Parameters dialog.
2. Expand the Code Generation node, if necessary.
3. On the Code Generation - DSRT variable description file options page, select A2L from the Variable description file format list.
4. Click OK to apply your settings and close the dialog.
5. Start the build process.

Result

A Simulink implementation container (SIC) file containing an A2L file fragment is generated.

Next step

You can now import the SIC file containing the A2L fragment to VEOS Player or add it to a ConfigurationDesk application. The A2L file fragment is extended by simulation-platform-specific information, such as memory addresses, during the build process.

Related topics

Basics

Adding Parameters and Signals to the A2L File (Simulink Model A2L File Generation Manual 🕮)

Information for Former RTI Users

Introduction

If you previously worked with RTI blocksets, you need to know the main differences and migration aspects.

Where to go from here

Information in this section

Main Differences Between Modeling with RTI and ConfigurationDesk

Introduction

This provides you with the information you need if you previously worked with RTI.

Conceptual differences

Working with RTI If you work with RTI, the following applies:

- The Simulink model represents the application.
- The Simulink model is specific to exactly one of the supported platforms.
- The Simulink model contains the behavior model and RTI blocks representing specific I/O functionality.
- Specific Simulink models are required for single-processor applications, multicore applications, and multi-processing-unit applications.

Working with ConfigurationDesk If you work with ConfigurationDesk, the following applies:

- The Simulink model contains the behavior model but no I/O functionality. Generic interface blocks (called model port blocks) specify the inputs and outputs of the behavior model. I/O functionality is implemented in ConfigurationDesk via function blocks that are mapped to the ports of the model port blocks.
- In ConfigurationDesk, you create a ConfigurationDesk project with one or more ConfigurationDesk applications. You can add multiple Simulink models or SIC files as behavior models to a ConfigurationDesk application.
- The model is not specific to a supported platform. You can use the same model with different ConfigurationDesk applications and with different I/O functionality.
- Specific Simulink models for single-processor applications, multicore applications, and multi-processing-unit applications are not needed.
- Executing multiple models in the same application process lets you combine multiple small behavior models into one real-time application without having to combine them into one overall model in Simulink.

RTI versus ConfigurationDesk The following table shows the main differences between working with ConfigurationDesk and working with RTI:

Action	Working with RTI	Working with ConfigurationDesk
Hardware mapping	The hardware is implicitly defined by the added RTI block, for example, DS2004ADC_BLx works only on a DS2004 I/O board. The mapping is a setting in the RTI block.	The behavior model does not contain any information about the hardware. Suitable hardware is assigned and can be replaced in ConfigurationDesk.
Modeling asynchronous tasks	Asynchronous tasks are modeled via board-specific RTI Interrupt blocks.	Asynchronous tasks are modeled via the Hardware-Triggered Runnable Function block from the Model Interface Blockset. Each Hardware-Triggered Runnable Function block in the Simulink model has a representative in ConfigurationDesk that must be connected to the event port of a function block. A preconfigured task triggered by an I/O event is then available in ConfigurationDesk.
	Software-generated interrupts are made available as trigger sources for tasks via Software Interrupt blocks.	For each Software-Triggered Runnable Function block in the Simulink model, a predefined task with an assigned software event is available in ConfigurationDesk.
Triggering periodic tasks by an asynchronous event	The periodic rates of the Simulink model are indicated by RTI as timer tasks. Per default, all timer tasks are driven by a processor-board-specific timer interrupt. To overwrite this behavior, you can use the Timer Task Assignment block. This block lets an RTI Interrupt block drive the timer tasks.	In ConfigurationDesk, a task with an assigned timer event is created for the fastest periodic task of the Simulink model. For the other periodic tasks of the Simulink model, predefined tasks with an assigned software event are created in ConfigurationDesk. These tasks are triggered by the fastest periodic task. Instead of the timer event, you can assign an I/O event to the fastest periodic task. As a result, all the predefined tasks are triggered asynchronously.

Action	Working with RTI	Working with ConfigurationDesk
Data acquisition service	By default, the data acquisition service is only activated for the base rate task[1)]. It can be activated for other tasks by using the Data Capture block.	By default, the data acquisition service is only activated for the base rate task. It can be activated for other tasks by using the DAQ raster name property in ConfigurationDesk.
Avoiding task overruns in the first simulation steps of the fastest timer task	The FIRST_SIMSTEP_INCREASEMENT option on the Build Options page (CPU options dialog) lets you avoid task overruns.	The Time-Scaled Period and Time Scale Factor properties of the global build settings in ConfigurationDesk let you avoid task overruns. These settings let you specify the requirements of the real-time application in more detail.
Modeling single-core multimodel applications	The feature is not available.	ConfigurationDesk lets you execute multiple models in the same application process. This lets you combine multiple small behavior models into one real-time application without having to combine them into one overall model in Simulink.
Modeling multicore applications and multiprocessor applications	Models for multicore applications and multiprocessor applications are modeled via blocks of the RTI-MP block library.	ConfigurationDesk lets you build multicore applications and multi-processing-unit applications. In case of a multicore application, ConfigurationDesk automatically assigns application processes to cores for execution during the download process. In case of a multi-processing unit application, available processing units are assigned to processing unit applications.
Specifying interprocessor communication	Interprocessor communication is modeled via the IPC block of the RTI-MP blockset. IPC blocks support only basic data types, such as scalar or array. Only one data type can be used per IPC block. Bus signals are not supported.	Interprocessor communication is implemented as model communication in ConfigurationDesk. The models involved in model communication belong to application processes in processing unit applications that are assigned to different processing units. For model communication, signals with different data types can be used in one model port block. Bus signals are also supported for model communication.
Specifying the communication protocol	Model communication is specified as a swinging buffer protocol in a synchronized or unsynchronized mode in the IPC block from the RTI-MP blockset.	Blocking/non-blocking communication: Model communication is specified as blocking or non-blocking communication in the Model Communication Browser in ConfigurationDesk.
Specifying the system target	If you work with RTI, you specify one of the available board-specific system target files, for example, rti1005.tlc to specify a DS1005 PPC Board as the system target and configure the modeling environment for the board-specific RTI blockset. The system target is automatically configured in a new behavior model if you let MATLAB set the configuration preferences automatically when activating an RTI platform for the first time. You can also modify these settings manually.	If you work with ConfigurationDesk, you use `dsrt.tlc` as the system target file. Alternatively, you can select the dSPACE Run-Time Target template from the Simulink Start Page to use a preconfigured model.

Action	Working with RTI	Working with ConfigurationDesk
Starting the build process	You configure and start the build process in the behavior model. An initial configuration of the build options for a new behavior model is automatically set if MATLAB automatically sets the configuration preferences when you activate an RTI platform for the first time.	You configure and start the build process in ConfigurationDesk.[2] ConfigurationDesk adapts the settings in the behavior model to the specific requirements. Some settings are therefore overwritten by predefined ConfigurationDesk settings. During the build process, the model code is generated and then ConfigurationDesk compiles and links it to the final real-time application. If you add a Simulink model to a ConfigurationDesk application via a Simulink implementation container (SIC file), the SIC file already contains the generated code for the Simulink model. The build process in ConfigurationDesk generates the code for the I/O functions, and then compiles and links the model code provided by the SIC file and the code generated for the I/O functions.
Downloading the application	Optionally, RTI downloads the application after the build.	Optionally, ConfigurationDesk downloads the application after the build. You can configure the download behavior according to your requirements.

[1] The base rate task is the fastest periodic task in the Simulink model.

[2] Alternatively, you can start the build process from the ConfigrationDesk menu of the Simulink model.

Related topics

Basics

Migrating Models..278

Migrating Models

Introduction

If you previously worked with RTI and the behavior model contains blocksets that are not supported by ConfigurationDesk, you must prepare your behavior model before you can use it with ConfigurationDesk.

Behavior model contains Simulink blocks only

If your behavior model does not contain any RTI blocks, you can use the model in ConfigurationDesk without any modifications. For further information, refer to Creating the Interface of Behavior Models on page 51.

When you start the build process in ConfigurationDesk, all the required configuration parameters in the behavior model are set automatically.

Behavior model contains supported RTI blocks

If your behavior model contains blocks from RTI blocksets that are supported by ConfigurationDesk, you do not need to modify the behavior model. The blocks are automatically migrated for use with the **dsrt.tlc** system target file. The blocks from the following RTI blocksets are supported:

- RTI CAN MultiMessage Blockset
- RTI LIN MultiMessage Blockset

When you start the build process in ConfigurationDesk, all the required configuration parameters of the behavior model are set automatically.

> **Note**
>
> The RTI CAN MultiMessage Blockset and the RTI LIN MultiMessage Blockset are not supported for the MicroAutoBox III.

Behavior model contains unsupported RTI blocks

Blocks from RTI blocksets other than the blocksets mentioned above are not supported. You must delete them from the model. Unsupported RTI blocks in your model are reported during the build process.

> **Note**
>
> You can contact dSPACE Support if you need help to find unsupported blocks in a behavior model.

To implement a comparable I/O functionality, you must add blocks from the Model Interface Blockset to your model and connect them with function blocks in ConfigurationDesk. Refer to Creating the Interface of Behavior Models on page 51.

When you start the build process in ConfigurationDesk, all the required configuration parameters of the behavior model are set automatically.

Special unsupported RTI blocks

RTI FlexRay Configuration Blockset ConfigurationDesk does not support blocks from the RTI FlexRay Configuration Blockset. If your behavior model contains such blocks, you must delete them and replace them with the corresponding blocks from the FlexRay Configuration Blockset. The FlexRay Configuration Blockset is part of the FlexRay Configuration Package. Refer to Modeling a FlexRay Bus Interface on page 141.

TRC Exclusion block Instead of RTI's **TRC Exclusion** block, which excludes all blocks in a subsystem from the generated variable description file (TRC file), you must use subsystem omission tags (`DsVdOmit`). These tags allow you to reduce the content of the generated TRC file even more specifically, for example, by excluding variables. Refer to Adapting the Generation of the Variable Description File on page 243.

Using models with RTI that were configured for ConfigurationDesk

If you configured the behavior model for ConfigurationDesk and want to use it in an RTI environment, you must implement the required functionality in your behavior model by replacing all model port blocks with board-specific blocks.

Before you start the build process using the Simulink® Coder^{TM}, you must set the required configuration parameters in the behavior model, for example, the board-specific system target.

Related topics

Basics

Main Differences Between Modeling with RTI and ConfigurationDesk.................................275

Limitations

Introduction

There are some limitations to note when working with the Model Interface Package for Simulink.

Where to go from here

Information in this section

General Limitations

Message dialog

The dialog that prompts you to confirm a message might not always be displayed in the foreground.

Limitations of Model Port Blocks

Limitations for placing model port blocks

You can place model port blocks anywhere in your Simulink model, except in:

- Subsystems with Read/Write permissions parameter set to NoReadOrWrite, because they cannot be accessed by ConfigurationDesk's model analysis feature.

  If you have a subsystem with mapped model port blocks and later configure it as a NoReadOrWrite subsystem, the model port blocks will be displayed as unresolved model port blocks in ConfigurationDesk after model analysis has been performed again. If you reconfigure the subsystem again, the mapping is automatically restored.
- Referenced models

  Model port blocks are allowed only in the top-level model when Simulink's model referencing feature is used.
- Simulink Function blocks
- Simulink functions within Stateflow charts
- Configurable subsystems

Model port blocks in Reusable Function subsystems It is allowed to place model port blocks in Reusable Function subsystems. If you do, model analysis and code generation are not aborted. However, no reusable code is generated for these model port blocks.

Limitation for element names

Depending on the tool chain you are working with (operating system, experiment software, modeling tool, etc.), the full range of UTF-16 characters might not be supported. To avoid problems, use only ASCII characters for names of model port blocks and model ports.

In addition, to avoid conflicts in the TRC file generation, do not use the following characters in block names and signals:

- "
- /
- .

Length of signal and runnable function names

The length of signal and runnable function names is limited to a maximum of 255 characters.

Only scalar trigger ports are supported

Hardware-Triggered Runnable Function blocks must be connected only to scalar trigger ports.

Unsupported signal properties

The following signal types and properties are not supported:

- Signals with data types that are neither a Simulink built-in numeric data type, nor a Simulink.Bus object

- Matrix signals
- Signals that are bus arrays
- Complex signals
- Frame-based signals

Only one sample time per model port block

All the data ports within one data port block have the same sample time.

Limitation for model port blocks in linked subsystems

If you break the library link of a linked subsystem, the Model Interface Package for Simulink cannot preserve the block and signal IDs of model port blocks in the subsystem. In this case, new IDs are assigned to the model port blocks and their signals..

No Simulink.Bus object consistency checks

The Model Interface Package for Simulink does not check the validity of Simulink.Bus objects that are assigned to model port blocks. Thus, invalid Simulink.Bus objects can result in unexpected behavior.

Limitations of Using Simulink Root Inports and Outports

Rules for using Simulink Inports and Outports

You can also use Simulink Inports and Outports as interface blocks for your model instead of the **Data Inport** and **Data Outport** blocks of the Model Interface Blockset, with the following limitations:

- The inports and outports must be on the root level of your model.
- The configuration of the inports and outports allows:
  - Port dimension ≥ 1 (scalars and vectors)
  - Data type = double, single, int8, uint8, int16, uint16, int32, uint32, int64, uint64, Boolean, and Simulink.Bus objects

> **Note**
>
> Bus signals defined by Simulink.Bus objects must be scalar. Bus arrays are not supported.

- The following parameters of the signal specification must be set to:
  - Signal type: real
  - Sampling mode: Sample based
- Signals from I/O function that are connected to Simulink Inports and Outports are always read at the beginning of a simulation step or written at the end of a simulation step. In contrast to this, the signals of I/O functions that are connected to data ports are read and written when the data port block is executed within a simulation step.

- For Simulink Inports, the variable description file option Include virtual blocks must be set to true. Otherwise, the build process returns an error message. The ports will be correctly displayed in ConfigurationDesk's model topology even if the option is set to false.
- The signals of the Simulink Inport and Outport blocks must be represented by global variables in the generated model code. Some build options prevent the generation of global variables, for example, Signal storage reuse. By enabling the Test point option, however, you can force the generation of global variables.
- The Model Interface Package for Simulink does not provide identities for Simulink inports/outports that are independent of the port name. This means that you must remap them manually in ConfigurationDesk after renaming them.

Limitations Concerning the Interaction Between Simulink and ConfigurationDesk

No analysis of the model interface if the analysis of the task information fails

A complete analysis of a Simulink model in ConfigurationDesk includes an analysis of the task information and the Simulink model interface. If you perform a complete analysis, and the analysis of the task information fails, the Simulink model interface is not analyzed either.

Limitation for model port blocks in Variant Subsystems

If you work with Variant Subsystems in your behavior model and perform an analysis of the model interface via the Analyze Model (Model Interface Only) command, changes in the model port block status (active/inactive) might not be detected by ConfigurationDesk.

No analysis of a model topology with In Bus Element and Out Bus Element blocks

The analysis of a model that contains root-level In Bus Element and Out Bus Element blocks requires an initialization of the model. Therefore, analyzing the model interface via the Analyze Simulink Model (Model Interface Only) command is not possible.

Different results after model analysis and code generation for In Bus Element blocks

In the following case, the results of the model analysis and the code generation differ:

- The signal of an In Bus Element block is specified by a Simulink.Bus object.
- The In Bus Element block is connected to a Bus Selector block that selects a subset of the bus signals.
- No other signal is used.

In the above case, a model port block with the complete bus hierarchy is displayed in ConfigurationDesk after a model analysis. However, after code generation, a model port block with only those bus signals that are selected by the Bus Selector block is displayed in ConfigurationDesk.

Limitations of Model Separation

Signal properties constraints

If input and output signals of a subsystem to be separated are connected to subsystems in other separated models, the following signal properties are not supported:

- Frame-based signals
- Complex signals
- Bus array signals
- Matrix signals
- Signals with the enum data type
- Signals with scaled fixed-point data type
- Integer signals whose word width is not an element of the following set {8 bits, 16 bits, 32 bits, 64 bits}
- Action-call signals
- Function-call signals

Bus analysis constraints

For model separation, the model-specific StrictBusMsg parameter must be set to one of the following values:

- ErrorLevel1
- WarnOnBusTreatedAsVector
- ErrorOnBusTreatedAsVector

Otherwise, model separation is aborted and an error message is displayed.

Limitations of Simulink Implementation Containers

Unsupported dSPACE blocksets

If your behavior model contains blocks from the dSPACE RTI CAN MultiMessage Blockset, the dSPACE LIN MultiMessage Blockset, or the FlexRay Configuration Blockset, you cannot generate a Simulink implementation container for it. However, you can use such models in ConfigurationDesk by adding them directly to a ConfigurationDesk application.

Platform-specific custom code cannot be detected

If you add platform-specific custom code to a behavior model that you want to generate an SIC file for, it might not be possible to build the code contained in the SIC for another platform. The Model Interface Package for Simulink does not issue a warning message about using platform-specific code in this case.

Limited C++ support

The VEOS Player lets you import and build model implementations containing C and C++ source code for the host PC target. However, the following limitations apply to the C++ support in connection with the MSVC compiler:

- Do not enable Simulink Coder's C++ code generation when you generate an SIC file. However, you can integrate your C++ code in an S-function.
- The names of variables to be generated in the variable description must comply with the C symbol syntax.

Limitations concerning the Code generation folder and Code generation folder structure options

Simulink lets you specify a code generation folder via the File generation control option in the Simulink Preferences dialog. However, the Model Interface Package for Simulink does not support user-specific settings for the Code generation folder property for Simulink models that are used for SIC file generation or in ConfigurationDesk. For the Code Generation Folder Structure option, only the default setting `Model Specific` is supported.

Limitations Regarding Simulink Variables in the TRC File

Block output signals

In some cases, Simulink Coder does not generate separate variables for block output signals. To make sure that Simulink Coder generates a separate variable for an output signal, you can designate any signal in a model as a test point.

Handling of the Selector block and the Bus Selector block

The outports of the Selector block, the Bus Selector block, and other virtual blocks that are connected to them are not generated into the variable description file, if the inport of the Selector block is connected to a contiguous array, or if the inport of the Bus Selector block is connected to a non-virtual bus.

To access a signal of a bus, do the following:

- Access the signal directly from the bus/array

  or
- Connect a non-virtual block to the signal, for example, a Signal Conversion block whose Output property is set to `Signal Copy`. Its outport is available in the variable description file.

If the inport of the Bus Selector block is connected via virtual bus, the outports are generated into the variable description file.

No TRC file entries for unsupported data types

The Model Interface Package for Simulink does not generate any TRC file entries for the following elements:

- String signals
- Fixed-point variables that are longer than 64 bits

> **Note**
>
> To use 64 bit integer variables in the generated code, the SupportLongLong flag must be set in the Simulink model.

- Structured variables, such as bus signals or structured parameters whose elements have an unsupported data type
- Simulink messages

No TRC file entries for virtual signals

Entries for virtual signals, e.g., from a Mux block or a Bus Creator block, are not generated into the TRC file.

Naming conflicts for signals and parameters

A Simulink block can have output signals and parameters with the same name, for example, in the case of masked subsystems or Stateflow charts. For this block, multiple entries with the same name are generated at the same level in the TRC file. To avoid this, you must specify different names for output signals and parameters of a Simulink block.

Limitations of Code Generation

No support of custom compiler options in Simulink

The Model Interface Package for Simulink does not support custom compiler options in Simulink. This means that such settings are ignored (without an error message). You must specify custom compiler options in ConfigurationDesk/VEOS Player.

No user-defined code generation folder for ConfigurationDesk

The Model Interface Package for Simulink does not support user-defined code generation folders for the build process with ConfigurationDesk.

Limitations Concerning MATLAB Compatibility

Introduction

The following limitations apply when you work with MATLAB®.

User S-functions must provide source code

For a `<sfunname>` user S-function, a `<sfunname>.c` or `<sfunname>.cpp` source file has to be present during model code generation. Such a file can be empty if the S-function implementation is provided otherwise, e.g., by a static library.

Model referencing

If you work with model referencing, the following limitations apply:

- The build process does not support referenced models which contain model port blocks. It is aborted with an error message. Model port blocks must be used only on the top level of a model referencing hierarchy.
- To access variables of a referenced model during simulation, the Total number of instances allowed per top model option on the Model Referencing page of the Configuration Parameters dialog must be set to One. This means that the referenced model cannot be included multiple times. If the Total number of instances allowed per top model option of a referenced model is set to Multiple, no entries for variables of this model are generated into the TRC file.
- In Simulink, protected models can contain other protected referenced models. Nested protected models are not supported.

Naming conventions

You have to follow the MATLAB naming conventions for files and directories.

Restricted character encoding

- Block names, signal labels and annotations using the full range of UTF-16 encoding are not supported by ConfigurationDesk. Such characters might lead to problems when you generate code. Use only ASCII characters to avoid this.
- In addition, to avoid conflicts in the TRC file generation, do not use the following characters in block names and signals:
  - "
  - /
  - .
- Variable names, e.g., parameter names, must not contain double underscores. The names also must not start with an underscore followed by an uppercase letter.

Restricted support of model templates

You cannot use model port blocks from the Model Interface Blockset, in a model template.

Restricted support of incremental code generation for top-level models

The MATLAB Incremental code generation for top-level models feature is supported by ConfigurationDesk with the following restriction:

If you perform a build process in ConfigurationDesk, including model code generation, and then just change labels in the Simulink model, Simulink Coder will not recognize this as a change in the model. Thus, Simulink Coder does not trigger a new code generation. As a result, ConfigurationDesk does not generate a new variable description file. The variable description file will contain the old label names.

No locked Simulink models

Locked Simulink models are not supported.

No support for Emulation hardware option

The Simulink® Coder™ option called Emulation hardware on the Hardware Implementation dialog is not supported.

No support of unit checks

Model port blocks do not support automatic unit checks.

Unsupported statements in custom code

Functions that access host hardware, such as file I/O, network access, etc., are not supported in custom code.

No code generation for recursive functions

You are not recommended to use recursive functions when generating real-time applications executed on real-time hardware.

Limitation for Upgrade Advisor API

The `upgradeadvisor` function lets you analyze and perform required migration steps for your model when you upgrade to a newer MATLAB version. The Simulink libraries of third-party blocksets that you use, such as the Model Interface Blockset, cannot be upgraded by this function. Do not change the default value of the `SkipBlocksets` argument from `true` to `false`.

Unsupported Simulink blocks

The following new Simulink blocks are not supported:

- Initialize Function block
- Reset Function block
- Terminate Function block

No custom folder settings for code generation and simulation fragments

Custom folder settings for code generation and simulation fragments are not supported.

No concurrent execution behavior

Simulink Coder provides the `Allow tasks to execute concurrently on target` option. The Model Interface Package for Simulink does not support this feature.

Restricted support of message signals

The following applies to message signals provided by the blocks of the Messages & Events library:

- Message signals do not appear in the TRC file, or they do not point to readable variables.
- You cannot connect message signals to blocks from dSPACE blocksets.
- Model separation is not supported if the separated models are connected via message signals.

No TRC file entries for string signals

Simulink signals support the String data type. A string signal can be used for modeling, but it does not appear in the generated TRC file, and can therefore not be accessed, e.g., for experimenting in ControlDesk.

No Simulink parameter dependency

You can define a Simulink parameter with a dependency to another Simulink parameter. For example, you can define the Simulink parameter myParamPlusOne with the value `myParam + 1`. However, due to a Simulink Coder limitation, parameter dependencies are not propagated to the generated code.

Code obfuscation not supported

Simulink Coder provides the `ObfuscateCode` option. This option is not supported by Model Interface Package for Simulink.

No support of matrices generated in row major format

You can specify whether the layout of matrices with two or more dimensions must be *column major* or *row major*. You can specify this for each model. If you select the *column major* option, the first index indicates the column. If you select the *row major* option, the first index indicates the row.

A2L file generation for Simulink models with matrices generated in *row major* format is not supported by Model Interface Package for Simulink.

No support of overriding parameters of referenced configuration sets

Overriding parameters of referenced configuration sets is not supported. The Model Interface Package for Simulink does not check if you have specified parameter overrides for your model.

> **Note**
>
> The use of parameter overrides can lead to misconfiguration of the behavior model and thus of the generated code.

No 64-bit integers for A2L file generation

64-bit integers are not supported for A2L file generation.

No modification of model port block configurations in subsystem references

The configuration of model port blocks in subsystem reference instances is read-only. The Model Interface Package for Simulink intercepts attempts to modify the data of model port blocks using an API command or block dialog.

Limitations when using MATLAB R2020a and newer

The following limitation applies when you work with MATLAB R2020a and newer:

- The 16-bit floating-point data type can be used with the Model Interface Package for Simulink. However, signals of this data type cannot be connected to model port blocks. Variables of this type are not generated into the variable description file.

Limitations when using MATLAB R2020b and later

The following limitations apply when you work with MATLAB R2020b:

- As of MATLAB R2020b, code generation settings for Simulink signals and Data Store Memory blocks can be configured only via the Code Mappings editor. Settings of the Code Mappings editor are not copied to the target models created during model separation. You must configure the code generation settings via the Code Mappings editor also in the target models.
- In Variant Subsystem blocks, Variant Sink blocks, and Variant Source blocks, you can specify variants for Simulink simulation and code generation. If model port blocks are used in such variants, only the active variant of a model port block is recognized during model analysis. The code generation variant is recognized during code generation and the creation of SIC files. Therefore, use model port blocks in such variants only for generating SIC files.

Limitations when using MATLAB R2021a

As of MATLAB R2021a, Simulink libraries can reference data dictionaries as data sources. A Simulink model that references blocks from these libraries inherits access to the referenced data dictionaries. The Model Interface Package for Simulink does not support model interfaces with data types,for example, Simulink.Bus objects, defined in the data dictionaries referenced by libraries.

Model Interface Package for Simulink Glossary

Introduction

The glossary briefly explains the most important expressions and naming conventions used in the Model Interface Package for Simulink documentation.

Where to go from here

Information in this section

B

Behavior model A model that contains the control algorithm for a controller (function prototyping system) or the algorithm of the controlled system (hardware-in-the-loop system). It does not contain I/O functionality and access to the hardware.

You can add a Simulink behavior model directly to a ConfigurationDesk application. Alternatively, you can generate a Simulink implementation container file (SIC file) from the Simulink model to add this file to a ConfigurationDesk application, or to import it into VEOS Player.

C

ConfigurationDesk A software tool for configuring and implementing real-time applications. There are two independent versions:

- ConfigurationDesk
- ConfigurationDesk for RapidPro

Models created with the Model Interface Package for Simulink can be used with ConfigurationDesk. ConfigurationDesk lets you add Simulink behavior models or SIC files to ConfigurationDesk projects and map them to I/O functions. You can implement real-time applications for the ConfigurationDesk projects and execute them on dSPACE real-time hardware.

ConfigurationDesk model interface The part of the model interface that is available in ConfigurationDesk. This specific term is used to explicitly distinguish between the model interface in ConfigurationDesk and the model interface in Simulink.

D

Data Inport block A model port block of the Model Interface Blockset that provides one or more data inports to receive data from the ports of another model implementation (ConfigurationDesk and VEOS Player) or from function ports (ConfigurationDesk only). Data Inport blocks specify data interfaces that are available in ConfigurationDesk or in VEOS Player for further use. For this purpose, a graphical representation is created for them in the respective tool during the import.

Data Outport block A model port block of the Model Interface Blockset that provides one or more data outports to send data to the ports of another model implementation (ConfigurationDesk and VEOS Player) or to function ports (ConfigurationDesk only). Data Outport blocks specify data interfaces that are available in ConfigurationDesk or in VEOS Player for further use. For this purpose, a graphical representation is created for them in the respective tool during the import.

H

Hardware-Triggered Runnable Function block A model port block that lets you execute parts of a model asynchronously and in a separate task. For this purpose, a Hardware-Triggered Runnable Function block exports a function-call subsystem in the Simulink behavior model as a runnable function. In ConfigurationDesk, you can then create a task from the runnable function and an I/O event.

I

Interface model A temporary Simulink model created with ConfigurationDesk, that contains model port blocks from the Model Interface Blockset. ConfigurationDesk initiates the creation of an interface model in Simulink. You can copy the blocks with their identities from the interface model and paste them into an existing Simulink behavior model. You can use the newly created interface model to create a new Simulink behavior model.

Inverse model port block A model port block that was created as a suitable counterpart for a selected model port block. Its purpose is setting up model communication. An inverse model port block has the same configuration, for example, the same port structure and port names, but the reverse data direction of the model port block from which it was created.

Inverse model port blocks cannot be created from Hardware-Triggered Runnable Function blocks or Software-Triggered Runnable Function blocks.

M

Model interface The part of the Simulink behavior model that is created using the model port blocks from the Model Interface Blockset. The model interface is used to connect the Simulink model with ConfigurationDesk or VEOS Player.

- If you connect the Simulink behavior model with ConfigurationDesk:

  If you add a Simulink model containing model port blocks to a ConfigurationDesk application, the model port blocks are a part of the model topology in ConfigurationDesk and can be used as an interface to the behavior model. Model port blocks can also be created in ConfigurationDesk. They can then be exported to a Simulink model and thus create or update the model interface of the Simulink model.

- If you connect the Simulink behavior model with VEOS Player:

  The model port blocks of the Simulink model interface are provided via a Simulink implementation container file (SIC file). The model port blocks correspond to VPU port groups in VEOS Player.

Model Interface Blockset A library of the Model Interface Package for Simulink that contains blocks to implement communication between a Simulink model and ConfigurationDesk or VEOS Player.

The Model Interface Blockset provides the following blocks:

- Data Inport block
- Data Outport block
- Hardware-Triggered Runnable Function block
- Software-Triggered Runnable Function block
- Model Separation Setup block

Model Interface Package for Simulink A dSPACE software product that lets you specify the interface of a Simulink behavior model that you can directly use in ConfigurationDesk. For the Simulink behavior model, you can also create a Simulink implementation container file (SIC file) that you can use in ConfigurationDesk and VEOS Player.

Model port An element of a model port block. There are the following types of model ports:

- Data inports:

  Data inports receive data from other model ports (ConfigurationDesk or VEOS Player), or from function ports (ConfigurationDesk only).
- Data outports:

  Data outports send data to other model ports (ConfigurationDesk or VEOS Player), or to function ports (ConfigurationDesk only).
- Runnable function ports:

  Runnable function ports are provided by Hardware-Triggered Runnable Function blocks or Software-Triggered Runnable Function blocks. A runnable function port can be connected to a function-call subsystem that must be executed asynchronously in a separate task.

Model port block A block of the Model Interface Blockset. A model port block is part of the Simulink model interface. Model port blocks specify data interfaces or export runnable functions. There are the following types of model port blocks:

- Data Inport block and Data Outport block

  In ConfigurationDesk, **Data Inport** blocks and **Data Outport** blocks correspond to model port blocks in the ConfigurationDesk model interface.
- Hardware-Triggered Runnable Function block

  Hardware-Triggered Runnable Function blocks are available as runnable function blocks for modeling asynchronous tasks (ConfigurationDesk only).
- Software-Triggered Runnable Function block

  Software-Triggered Runnable Function blocks are available as predefined tasks with an assigned software event (ConfigurationDesk and VEOS Player).

Model Separation Setup block A block of the Model Interface Blockset that opens the Model Separation Setup. The Model Separation Setup is used to separate individual models from an overall model in MATLAB/Simulink. Additionally, model separation starts the generation of a model communication description file (MCD file) that contains information on the separated models and their connections. You can use this MCD file in ConfigurationDesk.

R

Runnable function A function that must be called within a task to compute results. The runnable functions provided by a Simulink behavior model result either from different sample rates specified in the behavior model, or from the Hardware-Triggered Runnable Function block, which exports a function-call subsystem as a runnable function. In ConfigurationDesk, runnable functions are available after an analysis of the Simulink behavior model or after the import of an SIC file. In VEOS Player, the tasks in which the runnable functions are executed are available as measurement rasters in an A2L file.

S

Simulink implementation container A container that contains the model code of a Simulink behavior model. The Model Interface Package for Simulink lets you generate a Simulink implementation container file for a behavior model. The Simulink implementation container file can be used in ConfigurationDesk or in VEOS Player. A MATLAB installation is not required for this purpose. You can also use SIC files in ConfigurationDesk or VEOS Player that were created with an earlier dSPACE Release. This simplifies the reuse of the behavior model code. The file name extension of a Simulink implementation container is SIC.

Simulink model interface The part of the model interface that is available in the Simulink behavior model. This specific term is used to explicitly distinguish between the model interface in ConfigurationDesk and the model interface in Simulink.

Software-Triggered Runnable Function block A model port block that lets you execute parts of a model asynchronously and in a separate task. The Software-Triggered Runnable Function block exports the runnable function from the function-call subsystem as a predefined task with an assigned software event. The predefined task can be used in ConfigurationDesk or VEOS Player.

V

VEOS Player A software tool for configuring and implementing offline simulation applications on a VEOS system. Simulink implementation container files (SIC files) can be used with VEOS Player.