ConfigurationDesk

Tutorial Starting with Simulink

For ConfigurationDesk 6.7

Release 2021-A – May 2021

dSPACE

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

Contents

About This Tutorial

Content

This tutorial shows you the basic steps in ConfigurationDesk when you start out with a Simulink behavior model.

Tip

- If you want to learn how to work with ConfigurationDesk starting with an external device, such as an ECU, refer to ConfigurationDesk Tutorial Starting with External Devices 🕮.
- For a ConfigurationDesk tutorial using MicroAutobox III hardware, refer to ConfigurationDesk Tutorial MicroAutoBox III 🕮.

Required knowledge

This tutorial is primarily intended for engineers who implement and build real-time applications. Knowledge of Windows applications, the basics of dSPACE hardware, and the basics of MATLAB/Simulink is assumed.

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 Tutorial

Where to go from here

Information in this section

Working with the Tutorial

Purpose of the tutorial

This tutorial shows you the basic configuration steps in ConfigurationDesk when you start out with a Simulink behavior model.

Tip

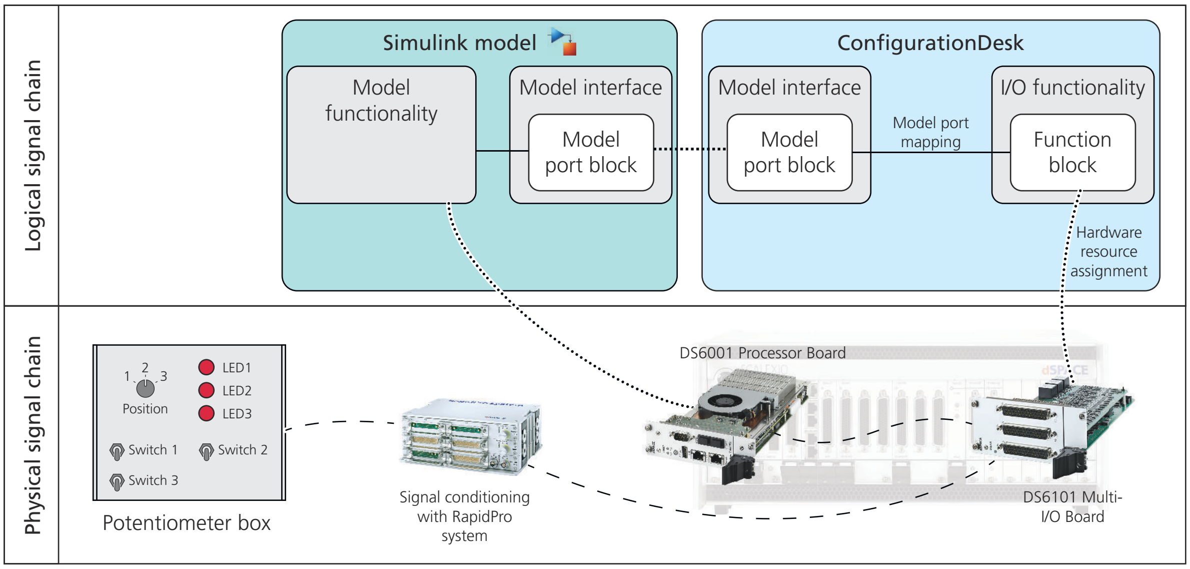
- If you want to learn how to work with ConfigurationDesk starting with an external device, such as an ECU, refer to ConfigurationDesk Tutorial Starting with External Devices 📖.
- For a ConfigurationDesk tutorial using MicroAutobox III hardware, refer to ConfigurationDesk Tutorial MicroAutoBox III 📖.

Recommended knowledge

Knowledge of the basic concepts of ConfigurationDesk will help you understand the context of the different lessons. For more information, refer to Basic Concepts of ConfigurationDesk (ConfigurationDesk Getting Started 📖).

Use scenario

The tutorial use scenario is based on a simple controller model for a potentiometer box with three switches, three LEDs, and a potentiometer. The following illustration shows the logical and the physical signal chain for the tutorial use scenario:



Tip

The hardware used in the tutorial use scenario is just an example. You can also use different dSPACE hardware as long as the required channels are provided.

The controller model implements the following behavior:

Switch	Position	Behavior
Switch 1	Off	Automatic mode: LEDs 1 to 3 are automatically triggered in sequence.
	On	Manual mode: LEDs 1 to 3 are triggered according to the position of the potentiometer.
Switch 2	Off	LED 2 is triggered according to the mode set by Switch 1.
	On	LED 2 is lit.
Switch 3	Off	LED 3 is triggered according to the mode set by Switch 1.
	On	LED 3 is lit.
Potentiometer	1 2 3	LED 1 is lit in manual mode (Switch 1 set to On).
	1 2 3	LED 2 is lit in manual mode (Switch 1 set to On).
	1 2 3	LED 3 is lit in manual mode (Switch 1 set to On).

Lessons of the tutorial

The real-time application for the tutorial use scenario is implemented in several lessons that deal with different areas of the implementation. Additionally, you can do a lesson on resolving conflicts and an advanced lesson that expands the use scenario. Refer to Overview of Lessons on page 9.

Completing specific lessons

Instead of starting with lesson 1, you can activate a specific application of an existing `CfgStartingWithSimulinkTutorial` demo project to complete or start with a specific lesson, for example, Lesson_3 to learn how to synchronize the ConfigurationDesk project and the Simulink model. For more information, refer to Completing Specific Lessons by Using the CfgStartingWithSimulinkTutorial Demo Project on page 43.

Note

There is no Lesson_1 application because the tutorial starts in Simulink.

Conflict handling

While you complete the lessons of the tutorial, several conflicts will be displayed in and removed from the Conflicts Viewer depending on the current configuration steps. You can ignore these conflicts while you complete the lessons, because ConfigurationDesk allows for a flexible configuration without strict constraints to let you work more freely.

Optional lesson on resolving conflicts The tutorial contains an optional lesson on resolving conflicts, for which you can use the ResolvingConflicts application of the demo project. This application is a modified version of the Results application and must be activated separately to start the lesson on resolving conflicts.

Overview of Lessons

Lessons and applications

The following table shows the contents of the tutorial lessons and the associated demo applications.

Content	Lesson	Demo Application
▪ How to create a ConfigurationDesk project from a Simulink behavior model	Lesson 1: Creating a ConfigurationDesk Project from a Simulink Model on page 11	No application available because the tutorial starts with a prepared model in Simulink.
▪ How to add I/O functionality to the ConfigurationDesk application by using function blocks ▪ How to map function blocks to the ConfigurationDesk model interface	Lesson 2: Adding I/O Functionality in ConfigurationDesk on page 17	Lesson_2

Content	Lesson	Demo Application
▪ How to configure function blocks ▪ How to assign hardware resources to function blocks ▪ How to create and map function blocks starting from hardware resources		
▪ How to propagate changes from ConfigurationDesk to the Simulink behavior model ▪ How to analyze changes made in the Simulink behavior model from ConfigurationDesk	Lesson 3: Synchronizing the ConfigurationDesk I/O Functionality and the Simulink Behavior Model on page 29	Lesson_3
▪ How to prepare the build process and build a real-time application	Lesson 4: Building the Real-Time Application on page 37	Lesson_4
▪ How to show and resolve conflicts	Optional Lesson: Resolving Conflicts on page 47	ResolvingConflicts
▪ How to replace the connected Simulink behavior model ▪ How to enable and configure event generation	Advanced Lesson: Adding I/O Events on page 53	Results

Tip

The workflow steps in the tutorial show you one method of obtaining the result. In many cases, other methods are also possible.

Lesson 1: Creating a ConfigurationDesk Project from a Simulink Model

Where to go from here

Information in this section

Overview of Lesson 1

From Simulink behavior model to ConfigurationDesk project

You can create a ConfigurationDesk project from a Simulink behavior model. The project contains a model interface representing the structure of the Simulink behavior model.

What will you do?

In this lesson you will:

- Create a ConfigurationDesk project from a Simulink behavior model. The project will include a ConfigurationDesk application with a model interface representing the structure of the behavior model and a hardware topology representing dSPACE real-time hardware in ConfigurationDesk.

Steps

This lesson contains the following steps:

- Step 1: How to Create a ConfigurationDesk Project from a Simulink Behavior Model on page 12

Step 1: How to Create a ConfigurationDesk Project from a Simulink Behavior Model

Precondition

A compatible MATLAB version (including Simulink) must be installed on your host PC and connected to your RCP and HIL installation (containing the ConfigurationDesk software).

Method

To create a ConfigurationDesk project from a Simulink behavior model

1 Copy the following Simulink model to a working directory of your choice:

```
<Documents folder>\Tutorials\CfgStartingWithSimulinkTutorial\Models\
Controller_model.slx
```

Note

The model is available only if ConfigurationDesk has been started at least once.

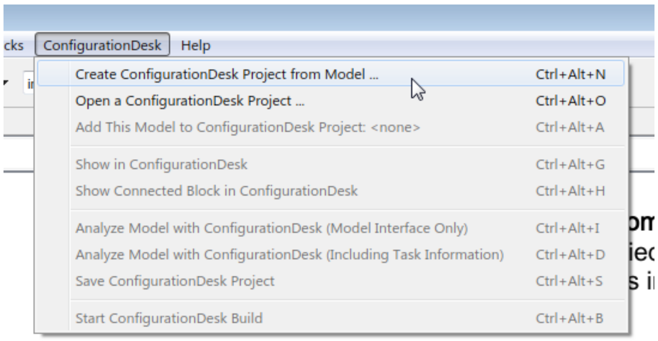
Tip

If you are not sure if the model has already been modified by a different user, copy the following model and rename it:

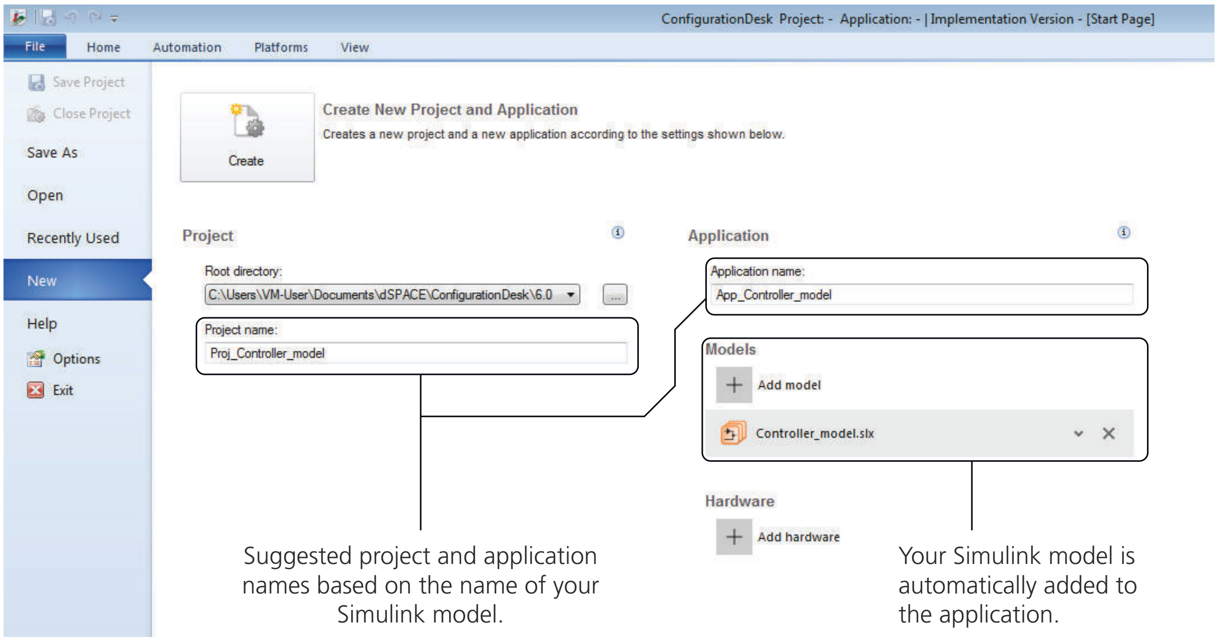
```
<Documents folder>\Tutorials\CfgStartingWithSimulinkTutorial\Models\
Controller_model_backup.slx
```

2 Open the copied model in Simulink.

3 In Simulink , select Create ConfigurationDesk Project from Model from the ConfigurationDesk menu.


cks
ConfigurationDesk
Help
Create ConfigurationDesk Project from Model ... Ctrl+Alt+N
Open a ConfigurationDesk Project ... Ctrl+Alt+O
Add This Model to ConfigurationDesk Project: <none> Ctrl+Alt+A
Show in ConfigurationDesk Ctrl+Alt+G
Show Connected Block in ConfigurationDesk Ctrl+Alt+H
Analyze Model with ConfigurationDesk (Model Interface Only) Ctrl+Alt+I
Analyze Model with ConfigurationDesk (Including Task Information) Ctrl+Alt+D
Save ConfigurationDesk Project Ctrl+Alt+S
Start ConfigurationDesk Build Ctrl+Alt+B

ConfigurationDesk is started (if necessary) and the New backstage view opens.



Create New Project and Application

Creates a new project and a new application according to the settings shown below.

Root directory: C:\Users\VM-User\Documents\dSPACE\ConfigurationDesk\6.0

Project name: Proj_Controller_model

Application name: App_Controller_model

Controller_model.slx

Suggested project and application names based on the name of your Simulink model.

Your Simulink model is automatically added to the application.

4 Change Project name to `MyCfgStartingWithSimulinkTutorial` and Application name to `Lesson_1`.

5 From the Add hardware menu, select Add hardware from file.

The Add Hardware File dialog opens.

6 Select the `dSPACE LabBox DS6001 Example System.htfx` hardware topology file from the

```
<Documents folder>\PredefinedHardware
```

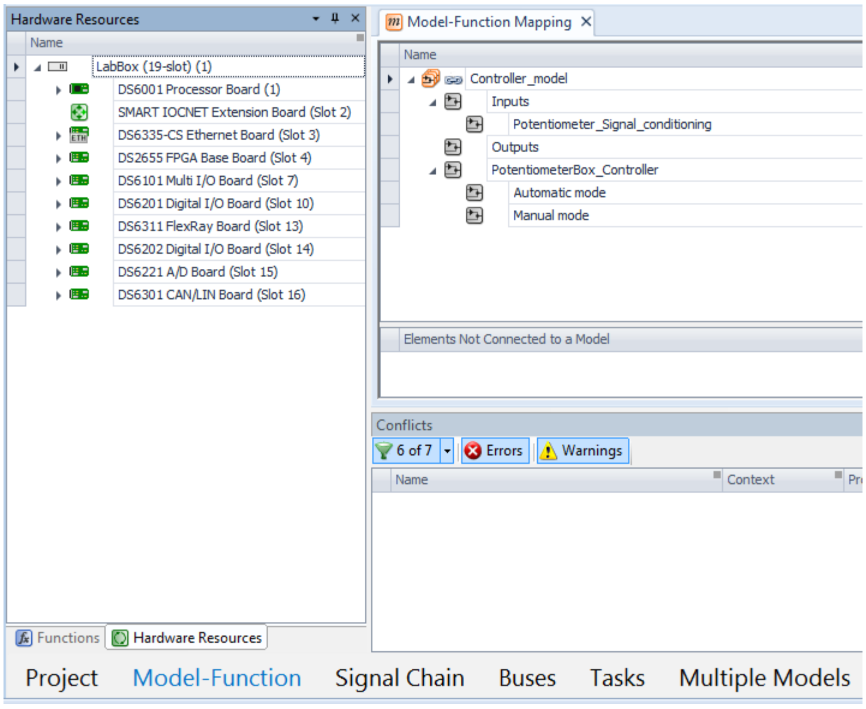
folder and click Open.

7 Click Create.

- A new ConfigurationDesk project with the specified project name is created in the selected root directory.
- A ConfigurationDesk application with the specified application name is added to it.
- The new application is active and opens in the Model-Function view set.

8 The Model-Function Mapping Browser contains a model interface with the `Controller_model` model and its subsystems.

The Hardware Resources Browser contains the hardware topology imported from the file you selected.


Hardware Resources
Name
LabBox (19-slot) (1)
DS6001 Processor Board (1)
SMART IOCNET Extension Board (Slot 2)
DS6335-CS Ethernet Board (Slot 3)
DS2655 FPGA Base Board (Slot 4)
DS6101 Multi I/O Board (Slot 7)
DS6201 Digital I/O Board (Slot 10)
DS6311 FlexRay Board (Slot 13)
DS6202 Digital I/O Board (Slot 14)
DS6221 A/D Board (Slot 15)
DS6301 CAN/LIN Board (Slot 16)
Functions
Hardware Resources
Model-Function Mapping
Name
Controller_model
Inputs
Potentiometer_Signal_conditioning
Outputs
PotentiometerBox_Controller
Automatic mode
Manual mode
Elements Not Connected to a Model
Conflicts
6 of 7
Errors
Warnings
Name
Context
Project
Model-Function
Signal Chain
Buses
Tasks
Multiple Models

Result

You created a ConfigurationDesk project and application from a Simulink behavior model. The ConfigurationDesk application contains a reference to the model so that both are connected as long as the ConfigurationDesk application is active.

Result of Lesson 1

Result

You created a ConfigurationDesk project from a Simulink behavior model. The project includes a ConfigurationDesk application with a model interface and a hardware topology.

Now, your application is equivalent to the Lesson_2 application of the `CfgStartingWithSimulinkTutorial` demo project.

What's next

You can now continue with Lesson 2: Adding I/O Functionality in ConfigurationDesk on page 17 for your `MyCfgStartingWithSimulinkTutorial` project, or you can open the existing `CfgStartingWithSimulinkTutorial` demo project and activate an application to continue with the associated lesson. For instructions, refer to:

- How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
- How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

Lesson 2: Adding I/O Functionality in ConfigurationDesk

Where to go from here

Information in this section

Overview of Lesson 2

Adding I/O functionality to the signal chain

In ConfigurationDesk, I/O functionality is added to the signal chain by using function blocks. The data flow between the function blocks and the connected behavior model is realized with model interfaces containing model port blocks. The model port blocks of the ConfigurationDesk model interface are mapped to the function blocks. Function blocks also provide access to hardware resources of the dSPACE real-time hardware (hardware resource assignment).

What will you do?

In this lesson you will:

- Add I/O functionality to the ConfigurationDesk application by using function blocks.
- Map function blocks to the ConfigurationDesk model interface.
- Configure function blocks.
- Assign hardware resources to function blocks.
- Create and map more function blocks starting from hardware resources.

Before you begin

Before you begin this lesson, one of the following preconditions must be met:

- You completed all the lessons of the tutorial up to this point.

or

- You opened the `CfgStartingWithSimulinkTutorial` demo project and activated the **Lesson_2** application. For instructions, refer to:
  - How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
  - How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

Steps

This lesson contains the following steps:

- Step 1: How to Add I/O Functionality to the ConfigurationDesk Application on page 18
- Step 2: How to Map Function Blocks to the ConfigurationDesk Model Interface on page 20
- Step 3: How to Configure Function Blocks on page 22
- Step 4: How to Assign Hardware Resources to Function Blocks on page 23
- Step 5: How to Create and Map Function Blocks Starting from Hardware Resources on page 25

Step 1: How to Add I/O Functionality to the ConfigurationDesk Application

Objective

I/O functionality is added to ConfigurationDesk applications by using function blocks.

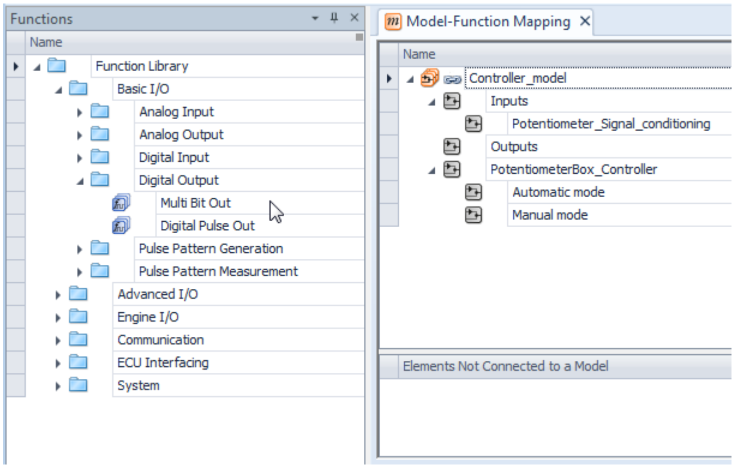
In the tutorial demo scenario, you will first add the function blocks to the signal chain that do not need to be mapped to the model interface, because their data is not required in the behavior model.

Method

To add function blocks to the ConfigurationDesk application

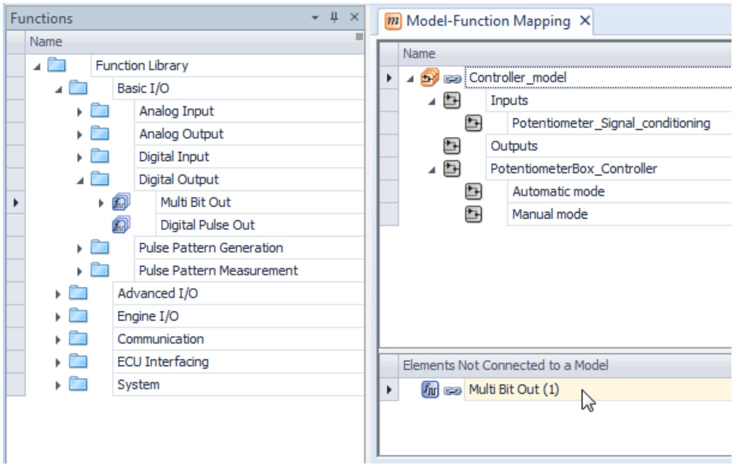
1 Switch to the Model-Function view set if necessary.

2 In the Function Browser, open the Basic I/O – Digital Output folder.



3 Drag the Multi Bit Out function block type to the Elements Not Connected to a Model area in the Model-Function Mapping Browser.

A function block named Multi Bit Out (1) is created.



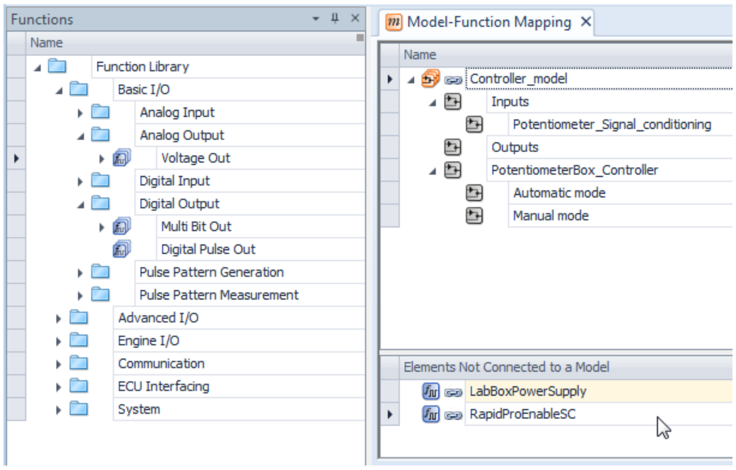
4 Click the function block and name it RapidProEnableSC.

5 Repeat steps 2 to 4 to add a Voltage Out function block from the Basic I/O – Analog Output folder and name it LabBoxPowerSupply.

Result

You added I/O functionality to the ConfigurationDesk application by using function blocks.

In the tutorial demo scenario, these function blocks do not need to be mapped to the model interface, because their data is not required in the behavior model.



Step 2: How to Map Function Blocks to the ConfigurationDesk Model Interface

Objective

To establish a connection between ConfigurationDesk's I/O functionality and the Simulink behavior model, you first have to map function blocks to the ConfigurationDesk model interface.

In the tutorial demo scenario, you will now add the function blocks for the output signals of the potentiometer box to the signal chain.

Method

To map function blocks to the ConfigurationDesk model interface

1 In the Function Browser, open the Basic I/O – Digital Output folder.

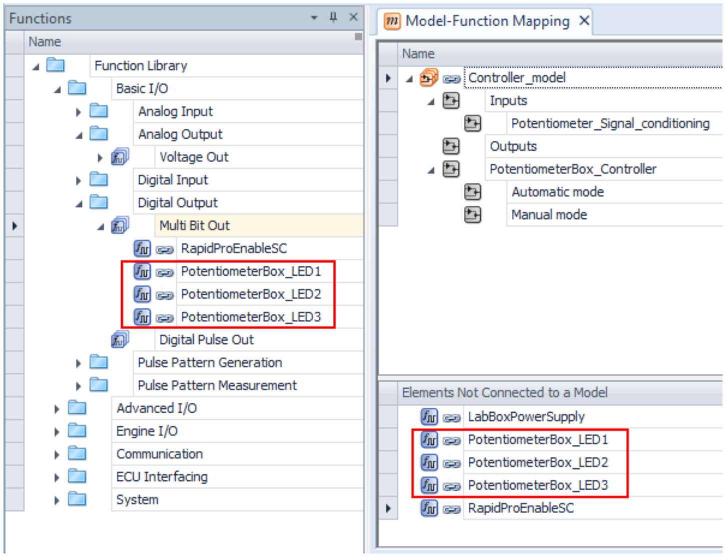
2 Right-click the Multi Bit Out function block type and select New – Multiple Multi Bit Out.

A Create Multiple Multi Bit Out dialog opens.

3 Set Name pattern to `PotentiometerBox_LED\` and Number of instances to 3.

4 Click **OK**.

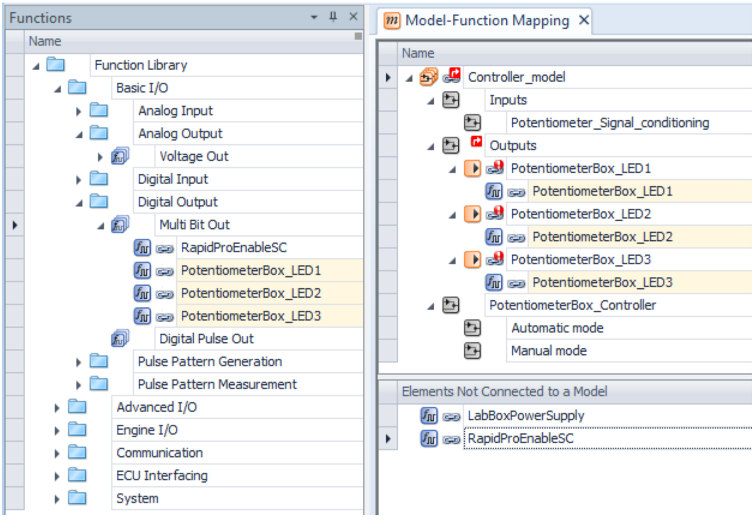
Three new function blocks are displayed in the **Function Browser** and the **Elements Not Connected to a Model** area.



5 Select the **PotentiometerBox_LED1** function block in the **Elements Not Connected to a Model** area, then press `Ctrl` and select the other two **PotentiometerBlock_LED** function blocks.

6 Drag the selected function blocks to the **Outputs** subsystem in the **Model-Function Mapping Browser**.

In the **Outputs** subsystem, a model port block is created for each function block and mapped to it. The model port blocks are named after the function blocks.



The symbol shows that a model port block is unresolved. This means it is used in the ConfigurationDesk application, but not in the connected Simulink model.

The symbol shows that a model or subsystem contains differences to the connected Simulink behavior model.

Result

You mapped function blocks to the ConfigurationDesk model interface. Model port blocks were automatically created for this purpose.

The model port blocks are not available in the Simulink model interface yet. This will be done in Lesson 3: Synchronizing the ConfigurationDesk I/O Functionality and the Simulink Behavior Model on page 29.

Step 3: How to Configure Function Blocks

Objective

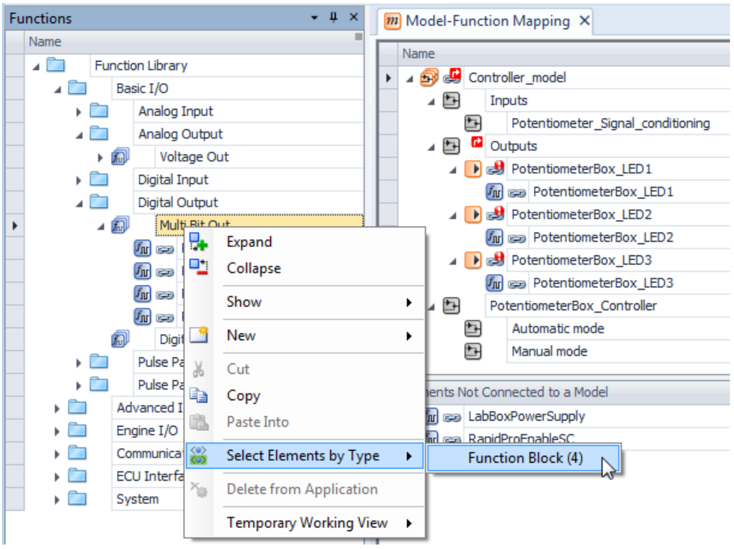
Some function block properties have to be adjusted for the demo scenario.

Method

To configure function blocks

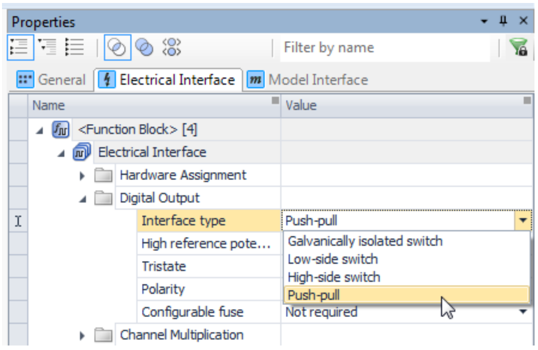
1 In the Function Browser, right-click the Multi Bit Out function block type.

2 From the context menu, select Select Elements by Type – Function Block (4) to select all function blocks of the Multi Bit Out type.



3 In the Properties Browser, switch to the Electrical Interface page.

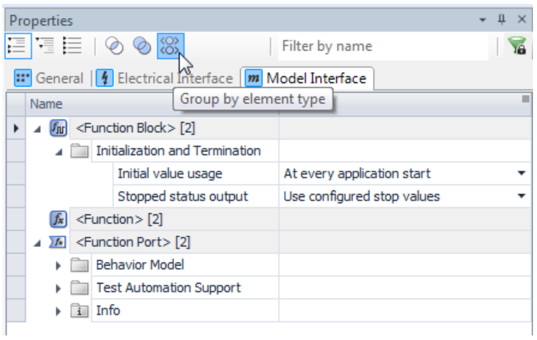
4 Expand the Digital Output category and set the Interface type property to Push-pull.



Name	Value
<Function Block> [4]	
Electrical Interface	
Hardware Assignment	
Digital Output	
Interface type	Push-pull
High reference pote…	Galvanically isolated switch
Tristate	Low-side switch
Polarity	High-side switch
	Push-pull
Configurable fuse	Not required
Channel Multiplication	

5 In the Elements Not Connected to a Model area, select the LabBoxPowerSupply function block, then press **Ctrl** and select the RapidProEnableSC function block.

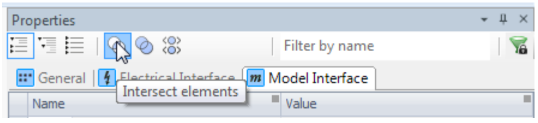
6 In the Properties Browser, switch to the Model Interface page and activate the Group by element type intersection mode.



Name	
<Function Block> [2]	
Initialization and Termination	
Initial value usage	At every application start
Stopped status output	Use configured stop values
<Function> [2]	
<Function Port> [2]	
Behavior Model	
Test Automation Support	
Info	

7 Expand the Behavior Model category and set the Model access property to Disabled.

8 Reactivate the Intersect elements mode.



Result You configured function blocks.

Step 4: How to Assign Hardware Resources to Function Blocks

Objective To assign hardware resources (channels on an I/O board or I/O unit) to function blocks, you can use the automatic hardware resource assignment provided by ConfigurationDesk.

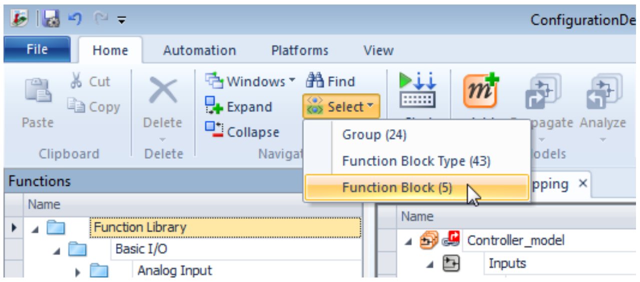
Function blocks require hardware resources of the dSPACE real-time hardware in order to be executed and to perform the I/O functionality.

Method

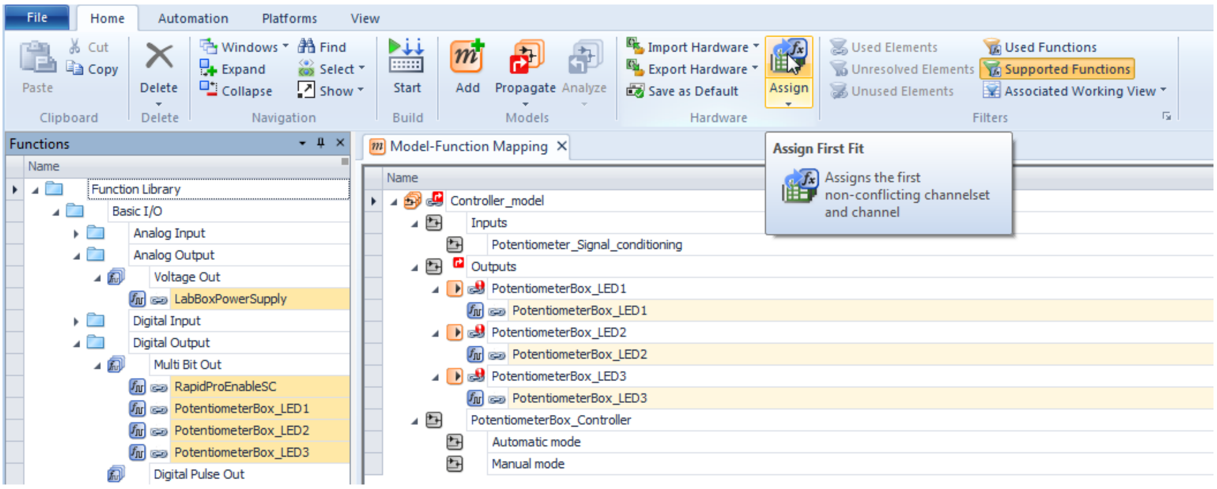
To assign hardware resources to function blocks

1 In the Function Browser, select the Function Library folder.

2 On the Home ribbon, select Navigation – Select – Function Block (5) to select all function blocks in the Function Browser.

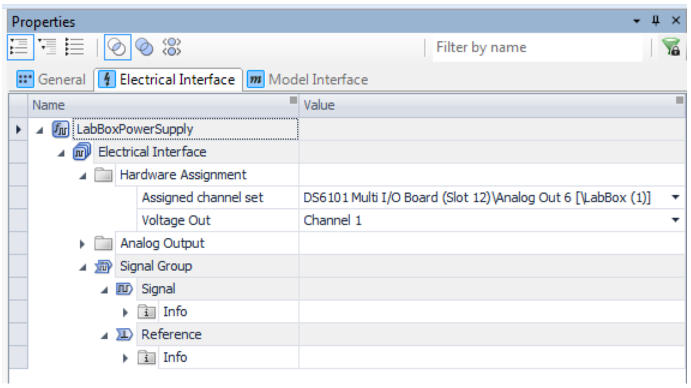


3 On the Home ribbon, click Hardware – Assign.



The first suitable and available channel sets and channels of the hardware topology are assigned to the function blocks.

4 To show the hardware resource assignment, select a function block, switch to the Electrical Interface page in the Properties Browser, and expand the Hardware Assignment category. The following screenshot illustrates the hardware resource assignment settings for the LabBoxPowerSupply function block.



Name	Value
LabBoxPowerSupply	
Electrical Interface	
Hardware Assignment	
Assigned channel set	DS6101 Multi I/O Board (Slot 12)\Analog Out 6 [\LabBox (1)]
Voltage Out	Channel 1
Analog Output	
Signal Group	
Signal	
Info	
Reference	
Info	

Result

You assigned hardware resources to function blocks.

Step 5: How to Create and Map Function Blocks Starting from Hardware Resources

Objective

ConfigurationDesk offers a simple way to create function blocks, map them to the model interface, and assign hardware resources to them in one step starting from hardware resources.

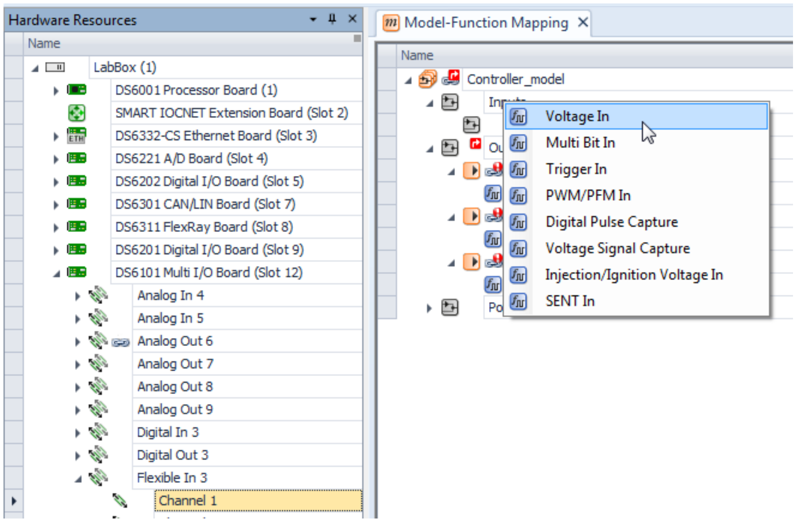
In the tutorial demo scenario, you will add the function blocks for the input signals this way.

Method

To create and map function blocks starting from hardware resources

1. In the Model-Function view set, open the Hardware Resource Browser.
2. Expand the LabBox, the DS6101 Multi I/O Board, and the Flexible In 3 channel set.
3. Drag the Channel 1 channel from the Flexible In 3 channel set to the Inputs subsystem in the Model-Function Mapping Browser.

A list of suitable function block types for the channel is displayed.


Hardware Resources
Name
LabBox (1)
DS6001 Processor Board (1)
SMART IOCNET Extension Board (Slot 2)
DS6332-CS Ethernet Board (Slot 3)
DS6221 A/D Board (Slot 4)
DS6202 Digital I/O Board (Slot 5)
DS6301 CAN/LIN Board (Slot 7)
DS6311 FlexRay Board (Slot 8)
DS6201 Digital I/O Board (Slot 9)
DS6101 Multi I/O Board (Slot 12)
Analog In 4
Analog In 5
Analog Out 6
Analog Out 7
Analog Out 8
Analog Out 9
Digital In 3
Digital Out 3
Flexible In 3
Channel 1
Model-Function Mapping
Name
Controller_model
Voltage In
Multi Bit In
Trigger In
PWM/PFM In
Digital Pulse Capture
Voltage Signal Capture
Injection/Ignition Voltage In
SENT In

4 Select the Voltage In function block type. The following actions are executed with this single step:

- A Voltage In (1) function block is created.
- A Voltage In (1) model port block is created.
- The Voltage In (1) function block is mapped to the Voltage In (1) model port block in the Inputs subsystem.
- The Channel 1 channel and the Flexible In 3 channel set from the DS6101 Multi I/O Board are assigned to the Voltage In (1) function block.

5 In the Hardware Resource Browser, expand the Digital In 3 channel set.

6 Select Channel 1, then press **`Ctrl`** and select channels 2 and 3 of the Digital In 3 channel set.

7 Drag the selection to the Inputs subsystem in the Model-Function Mapping Browser.

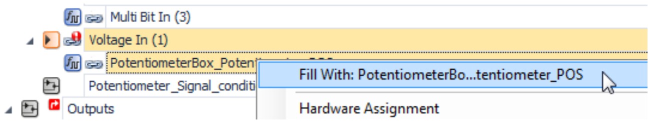
8 Select the Multi Bit In function block type.

Three Multi Bit In function blocks are created, mapped to model port blocks, and have the selected channels assigned to them.

9 Rename the Voltage In (1) function block PotentiometerBox_Potentiometer_POS.

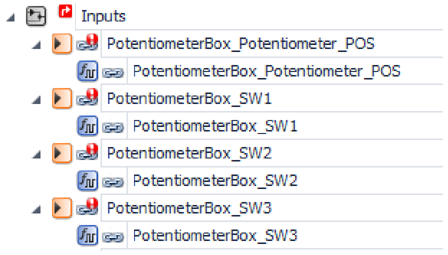
10 Press **`Ctrl`** and select the PotentiometerBox_Potentiometer_POS function block and the Voltage In (1) model port block.

11 Right-click the PotentiometerBox_Potentiometer_POS and select Fill With: PotentiometerBo...tentiometer_POS.



The model port block is named after the function block.

12 Repeat steps 9 to 11 to rename the Multi Bit In function and model port blocks according to the following illustration:



Inputs
- PotentiometerBox_Potentiometer_POS
  - PotentiometerBox_Potentiometer_POS
- PotentiometerBox_SW1
  - PotentiometerBox_SW1
- PotentiometerBox_SW2
  - PotentiometerBox_SW2
- PotentiometerBox_SW3
  - PotentiometerBox_SW3

Result

You created and mapped function blocks from hardware resources.

Result of Lesson 2

Result

You added I/O functionality to the ConfigurationDesk application, configured function blocks and assigned hardware resources to them, and mapped the function blocks to the model interface via model port blocks.

Now, your application is equivalent to the Lesson_3 demo application of the `CfgStartingWithSimulinkTutorial` demo project.

Further information

For detailed descriptions and instructions on the topics in this lesson, refer to:

- Implementing I/O Functionality (ConfigurationDesk Real-Time Implementation Guide 📖)
- Assigning Hardware Resources to Function Blocks (ConfigurationDesk Real-Time Implementation Guide 📖)
- User-Friendly Connection of ConfigurationDesk and Simulink Models (ConfigurationDesk Real-Time Implementation Guide 📖)

What's next

You can now continue with Lesson 3: Synchronizing the ConfigurationDesk I/O Functionality and the Simulink Behavior Model on page 29 for your `MyCfgStartingWithSimulinkTutorial` project, or you can open the existing `CfgStartingWithSimulinkTutorial` demo project and activate an application to continue with the associated lesson. For instructions, refer to:

- How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
- How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

Lesson 3: Synchronizing the ConfigurationDesk I/O Functionality and the Simulink Behavior Model

Where to go from here

Information in this section

Overview of Lesson 3

Synchronizing model interfaces

To synchronize the I/O functionality you added in Lesson 2: Adding I/O Functionality in ConfigurationDesk on page 17 with the Simulink behavior model, you must propagate the model port blocks from the ConfigurationDesk model interface to the Simulink model interface and connect them in the Simulink behavior model.

If you modify the Simulink behavior model, you must analyze it from ConfigurationDesk to synchronize.

Tip

Depending on the state of the implementation work in a ConfigurationDesk application or a Simulink behavior model, propagating changes to Simulink or analyzing a Simulink behavior model might have different effects.

For example, it is easier to completely finish configuring the I/O functionality in ConfigurationDesk before propagation, because this generates only correctly configured model port blocks in Simulink. Also, any undo operation in ConfigurationDesk does not undo changes that you have already propagated to Simulink.

On the other hand, you can modify the structure, e.g., add subsystems, only in Simulink and then analyze the modifications from ConfigurationDesk.

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.

For more information on synchronizing ConfigurationDesk applications and Simulink models, refer to User-Friendly Connection of ConfigurationDesk and Simulink Models (ConfigurationDesk Real-Time Implementation Guide 📖).

What will you do?

In this lesson you will:

- Propagate changes from ConfigurationDesk to the Simulink behavior model.
- Analyze changes in the Simulink behavior model from ConfigurationDesk.

Before you begin

Before you begin this lesson, one of the following preconditions must be met:

- You completed all the lessons of the tutorial up to this point.

or

- You opened the `CfgStartingWithSimulinkTutorial` demo project and activated the Lesson_3 application. For instructions, refer to:
  - How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
  - How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

Steps

This lesson contains the following steps:

- Step 1: How to Propagate Changes from ConfigurationDesk to the Simulink Behavior Model on page 31
- Step 2: How to Analyze Changes Made in the Simulink Behavior Model from ConfigurationDesk on page 33

Step 1: How to Propagate Changes from ConfigurationDesk to the Simulink Behavior Model

Objective

You have to propagate the added I/O functionality and the resulting model port blocks to the Simulink model interface and connect them in the behavior model.

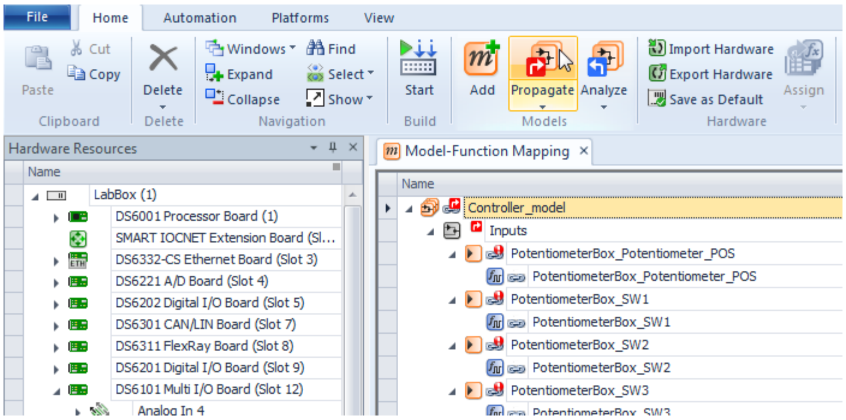
Method

To propagate changes from ConfigurationDesk to the Simulink behavior model

1 Switch to the Model-Function view set if necessary.

2 In the Model-Function Mapping Browser, select the Controller_model model.

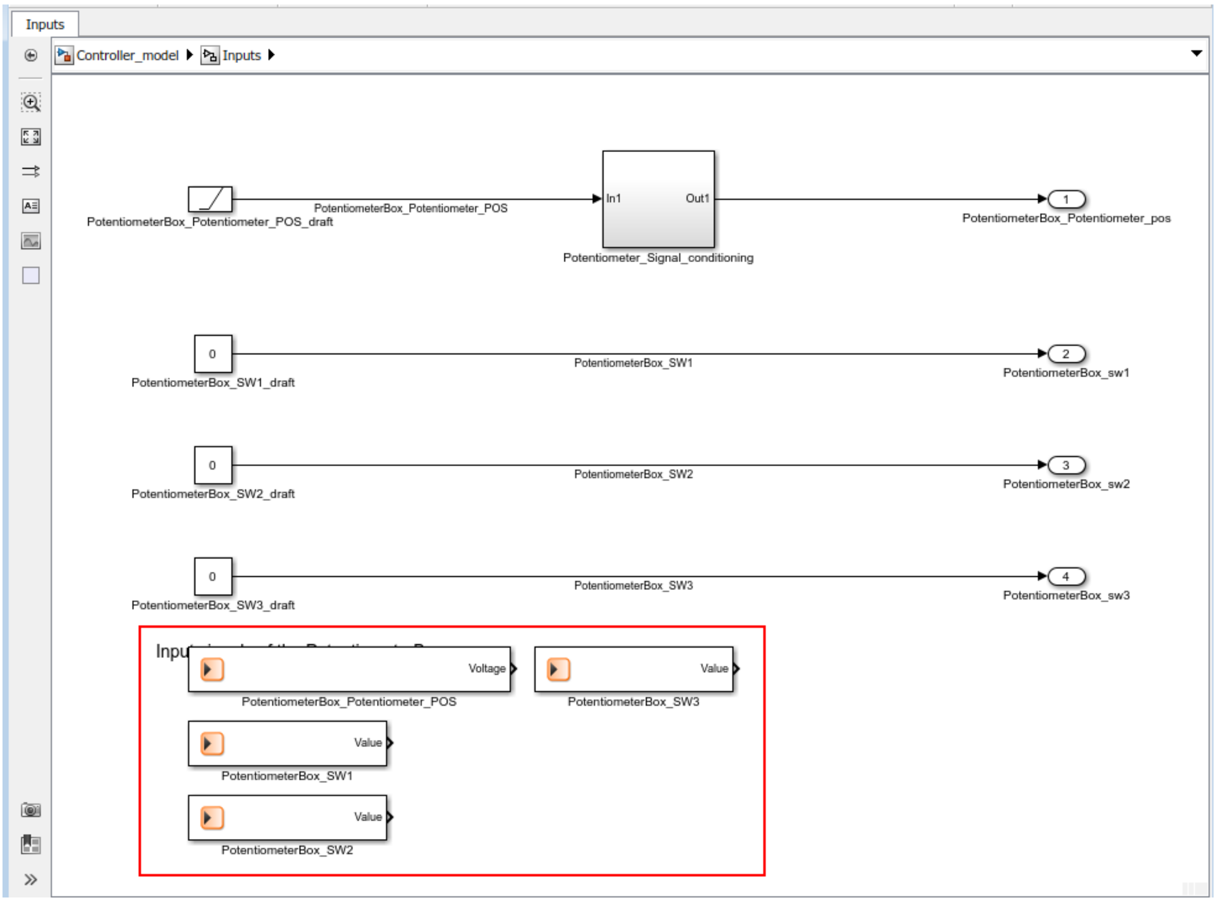
3 On the Home ribbon, click Models – Propagate.



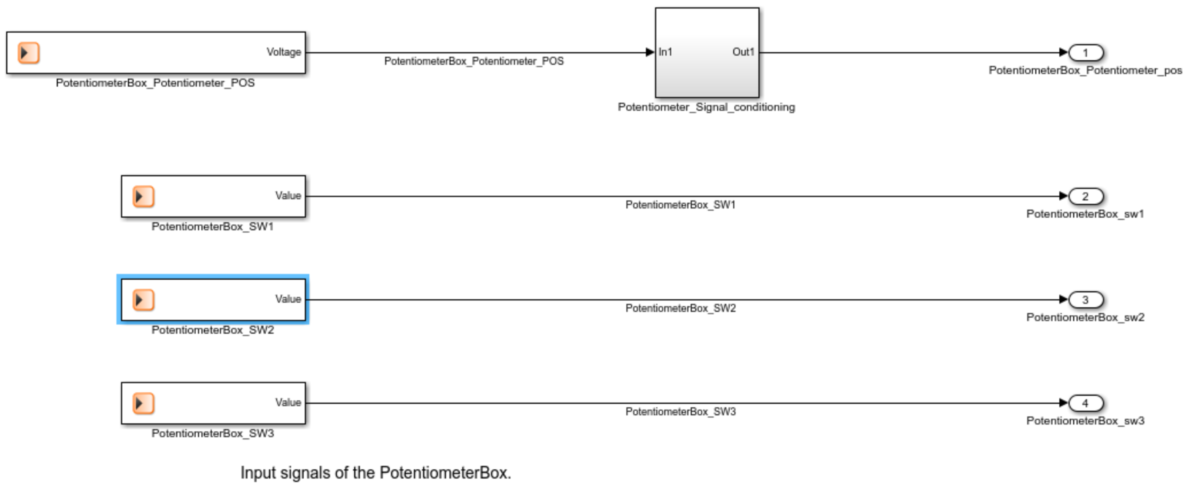
ConfigurationDesk propagates the changes from Lesson 2: Adding I/O Functionality in ConfigurationDesk on page 17 to the Simulink model interface. All model port blocks that are not available in the Simulink behavior model yet are created according to the function block settings in ConfigurationDesk.

The symbols marking the unresolved model port blocks and the model/subsystems containing changes are removed.

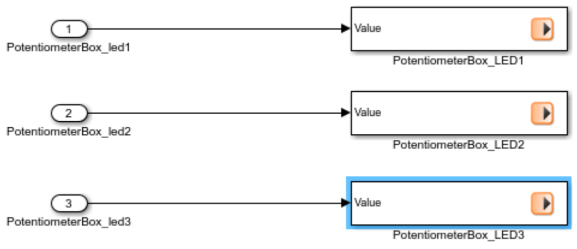
4 In the Simulink model, open the **Inputs** subsystem.
Model port blocks were created according to the model port blocks in ConfigurationDesk.


Inputs
Controller_model
Inputs
PotentiometerBox_Potentiometer_POS_draft
PotentiometerBox_Potentiometer_POS
In1
Out1
Potentiometer_Signal_conditioning
1
PotentiometerBox_Potentiometer_pos
0
PotentiometerBox_SW1_draft
PotentiometerBox_SW1
2
PotentiometerBox_sw1
0
PotentiometerBox_SW2_draft
PotentiometerBox_SW2
3
PotentiometerBox_sw2
0
PotentiometerBox_SW3_draft
PotentiometerBox_SW3
4
PotentiometerBox_sw3
Voltage
PotentiometerBox_Potentiometer_POS
Value
PotentiometerBox_SW3
Value
PotentiometerBox_SW1
Value
PotentiometerBox_SW2

5 Remove the placeholder input blocks to the left and connect the newly created model port blocks instead.



6 Open the **Outputs** subsystem and replace the output blocks to the right.



Output signals to trigger the three LEDs of the PotentiometerBox.

Result

You propagated changes from ConfigurationDesk to the Simulink behavior model.

The model interfaces are now synchronized.

Step 2: How to Analyze Changes Made in the Simulink Behavior Model from ConfigurationDesk

Objective

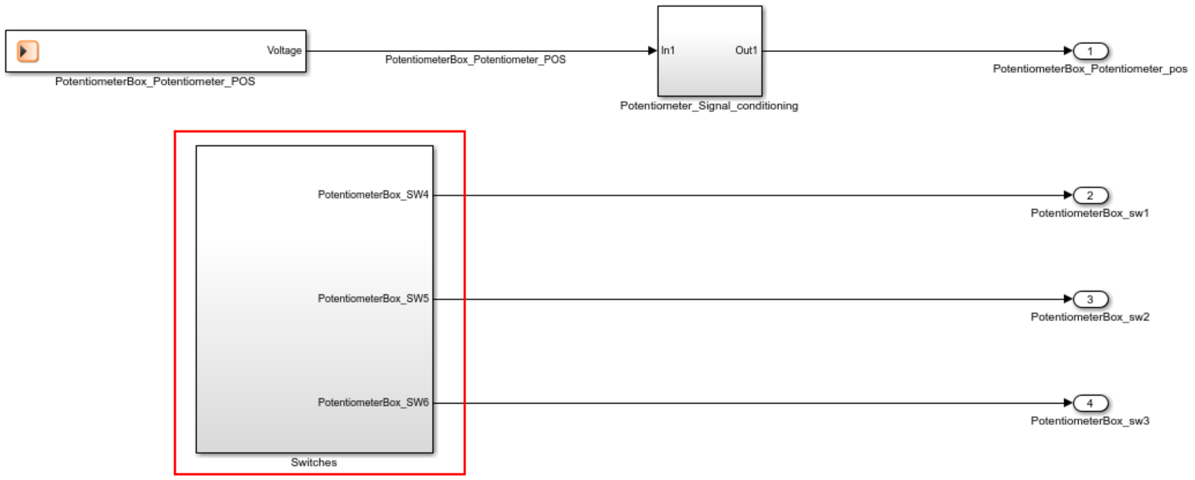
If you apply changes to the Simulink behavior model, you have to analyze it from ConfigurationDesk to synchronize the model interfaces.

In the tutorial demo scenario, you will create an additional subsystem in the Simulink model.

Method

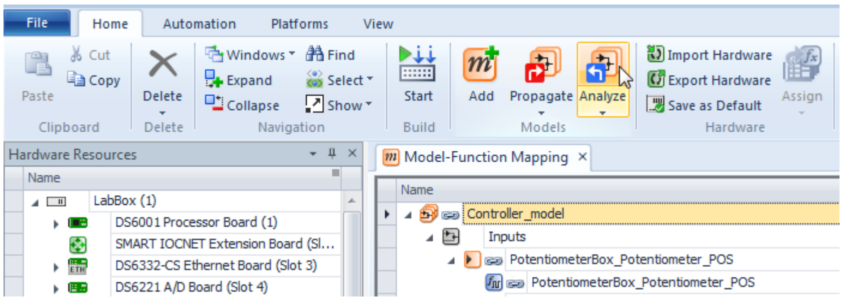
To analyze changes made in the Simulink behavior model from ConfigurationDesk

1 In the Inputs subsystem, select the PotentiometerBox_SW blocks and create a Switches subsystem from them.

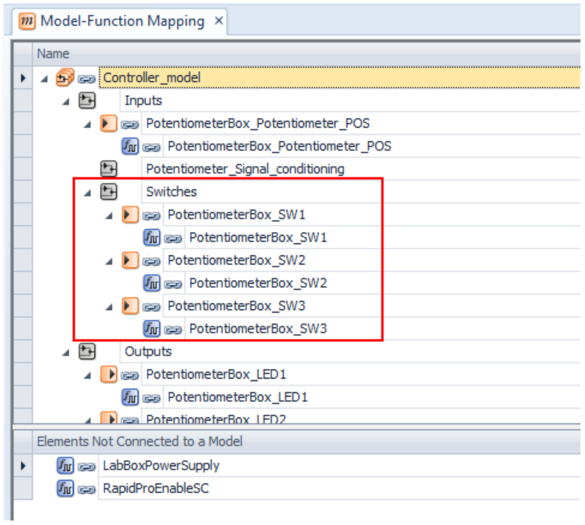


2 In ConfigurationDesk's Model-Function Mapping Browser, select the Controller_model model.

3 On the Home ribbon, click Models – Analyze.



The structural change you made in the Simulink behavior model is synchronized in ConfigurationDesk.



Result

You analyzed changes made in the Simulink behavior model from ConfigurationDesk.

The model interfaces are now synchronized.

Result of Lesson 3

Result

You synchronized the ConfigurationDesk I/O functionality and the Simulink behavior model by propagating the ConfigurationDesk model interface to the Simulink behavior model and analyzing changes made in the Simulink behavior model from ConfigurationDesk.

Now your application is equivalent to the Lesson_4 demo application of the `CfgStartingWithSimulinkTutorial` demo project.

Further information

For details on synchronizing ConfigurationDesk projects and Simulink behavior model, refer to User-Friendly Connection of ConfigurationDesk and Simulink Models (ConfigurationDesk Real-Time Implementation Guide 📖).

What's next

You can now continue with Lesson 4: Building the Real-Time Application on page 37 for your `MyCfgStartingWithSimulinkTutorial` project, or you can open the existing `CfgStartingWithSimulinkTutorial` demo project and activate an application to continue with the associated lesson. For instructions, refer to:

- How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
- How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

Lesson 4: Building the Real-Time Application

Where to go from here

Information in this section

Overview of Lesson 4

Build process

Building real-time applications is a process which generates executable code that can be run on the real-time hardware.

What will you do?

In this lesson you will:

- Prepare the build process by applying build configuration settings.
- Build the executable real-time application.

Before you begin

Before you begin this lesson, one of the following preconditions must be met:

- You completed all the lessons of the tutorial up to this point.

or

- You opened the `CfgStartingWithSimulinkTutorial` demo project and activated the Lesson_4 application. For instructions, refer to:

- How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
- How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

Steps

This lesson contains the following step:

- Step 1: How to Prepare the Build Process and Build the Real-Time Application on page 38

Step 1: How to Prepare the Build Process and Build the Real-Time Application

Objective

To execute a real-time application, you need to build it. You can then download it to a matching hardware platform.

Before you build the real-time application, you can specify several build settings that influence the build process.

Precondition

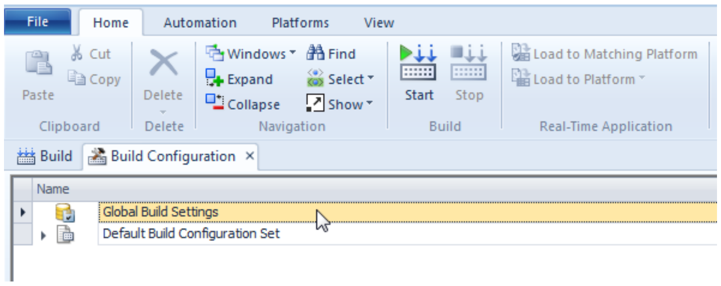
Make sure that no Simulink model from other lessons of the tutorial is open.

Method

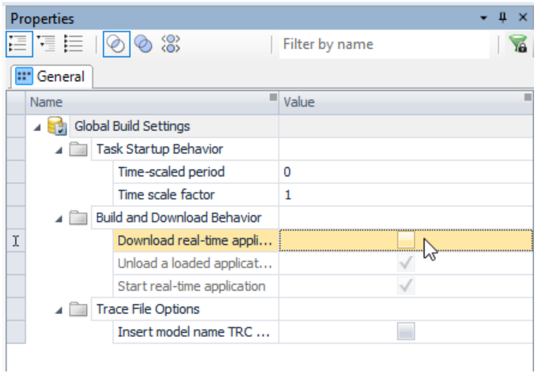
To prepare the build process and build the real-time application

1 Switch to the Build view set.

2 In the Build Configuration table, select Global Build Settings.



3 In the Properties Browser, clear the checkbox next to the Download real-time application after build property.



Properties

Filter by name

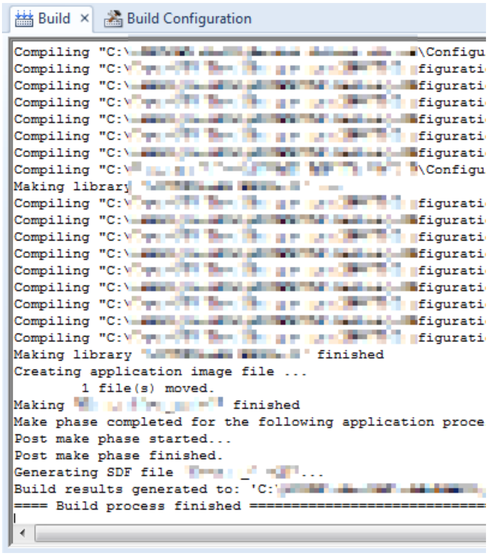
General

Name	Value
Global Build Settings	
Task Startup Behavior	
Time-scaled period	0
Time scale factor	1
Build and Download Behavior	
Download real-time appli...	☐
Unload a loaded applicat...	✓
Start real-time application	✓
Trace File Options	
Insert model name TRC ...	☐

4 On the Home ribbon, click Build – Start.

The build process starts. If MATLAB is not running when you start the build process, it opens automatically for model code generation. The build process is aborted only for serious errors. Most problems that occur during the build process are categorized as warnings. They are displayed in the Build Log Viewer. Messages regarding model code generation are displayed in the MATLAB Command Window.

5 Wait until the message Build process finished is displayed in the Build Log Viewer.



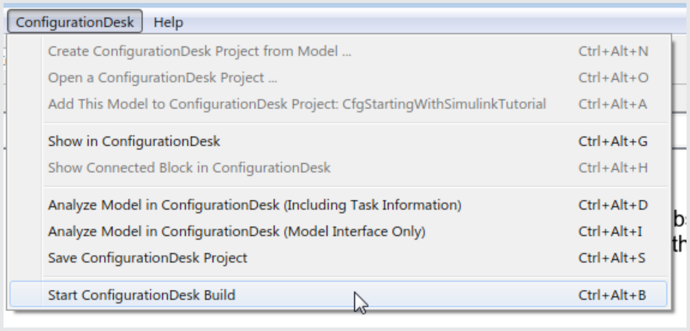
```
Compiling "C:\[redacted]\Configu
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]\Configu
Making library[redacted]
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Compiling "C:\[redacted]figurati
Making library [redacted] finished
Creating application image file ...
        1 file(s) moved.
Making [redacted] finished
Make phase completed for the following application proce
Post make phase started...
Post make phase finished.
Generating SDF file [redacted]...
Build results generated to: 'C:'[redacted]
==== Build process finished ============================
```

Result You prepared the build process and built the real-time application.

The build result files are displayed in the **Project Manager**.

Tip

You can also start the build process from the Simulink model by using the **Start ConfigurationDesk Build** command from the **ConfigurationDesk** menu.


ConfigurationDesk
Help
Create ConfigurationDesk Project from Model ... Ctrl+Alt+N
Open a ConfigurationDesk Project ... Ctrl+Alt+O
Add This Model to ConfigurationDesk Project: CfgStartingWithSimulinkTutorial Ctrl+Alt+A
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

In a real scenario, you could now download the real-time application to a hardware platform. For more information, refer to Downloading and Executing Real-Time Applications (ConfigurationDesk Real-Time Implementation Guide 📖).

The **Build Configuration** table lets you specify various build settings. You can create different build configuration sets and assign application processes to them. For more information, refer to Building Real-Time Applications (ConfigurationDesk Real-Time Implementation Guide 📖).

Result of Lesson 4

Result

You prepared the build process and built the executable real-time application.

Now, your application is equivalent to the **Results** demo application of the `CfgStartingWithSimulinkTutorial` demo project.

Further information

For detailed descriptions and instructions on build settings and building real-time applications, refer to Building Real-Time Applications (ConfigurationDesk Real-Time Implementation Guide 📖).

What's next

You finished the last lesson of the tutorial. To repeat a previous lesson or complete a lesson that you skipped, open the existing `CfgStartingWithSimulinkTutorial` demo project and activate the associated application. For instructions, refer to:

- How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
- How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

You can also:

- Activate the **ResolvingConflicts** application of the `CfgStartingWithSimulinkTutorial` demo project to complete the Optional Lesson: Resolving Conflicts on page 47.
- Complete the Advanced Lesson: Adding I/O Events on page 53.

For a summary of your working results, refer to Summary on page 59.

Completing Specific Lessons by Using the CfgStartingWithSimulinkTutorial Demo Project

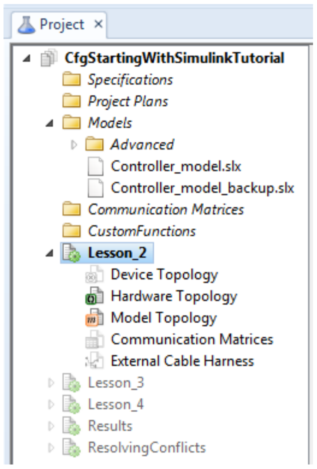
Where to go from here

Information in this section

Overview of the CfgStartingWithSimulinkTutorial Demo Project

Components of the project

The demo project contains applications for the tutorial's lessons. Each lesson contains only very few work steps, so you can easily switch between the applications to complete different lessons.



The applications build on one another. For example, all the results of the worksteps in lesson 2 are included in the Lesson_3 application.

Note

There is no Lesson_1 application because the tutorial starts in Simulink.

File location

The files of the demo project are available in the *Documents folder*.

They are also backed up in the `CfgStartingWithSimulinkTutorial.zip` archive, which is located in the `<RCP and HIL installation folder>\Demos\ConfigurationDesk\Tutorial` folder after you install the dSPACE software.

How to Open the CfgStartingWithSimulinkTutorial Demo Project

Objective

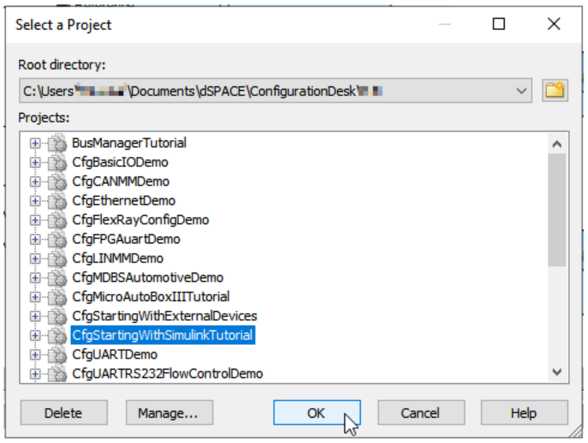
The `CfgStartingWithSimulinkTutorial` demo project is available in the *Documents folder*, so you can open it like a normal project.

Method

To open the tutorial demo project

1 From the Start menu, select dSPACE RCP and HIL <dSPACE Release> – dSPACE ConfigurationDesk <x.y>.

2 On the Start Page, click Open Project + Application.
ConfigurationDesk opens the Select a Project dialog.



Root directory:
C:\Users\[redacted]\Documents\dSPACE\ConfigurationDesk\[redacted]

Projects:
- BusManagerTutorial
- CfgBasicIODemo
- CfgCANMMDemo
- CfgEthernetDemo
- CfgFlexRayConfigDemo
- CfgFPGAuartDemo
- CfgLINMMDemo
- CfgMDBSAutomotiveDemo
- CfgMicroAutoBoxIIITutorial
- CfgStartingWithExternalDevices
- CfgStartingWithSimulinkTutorial
- CfgUARTDemo
- CfgUARTRS232FlowControlDemo

3 Select the CfgStartingWithSimulinkTutorial project and click OK.

Result

You opened the tutorial demo project. The Lesson_2 application is active.

How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project

Objective

To work with a specific tutorial demo project application, you must first activate it. The active application is displayed in bold letters.

Precondition

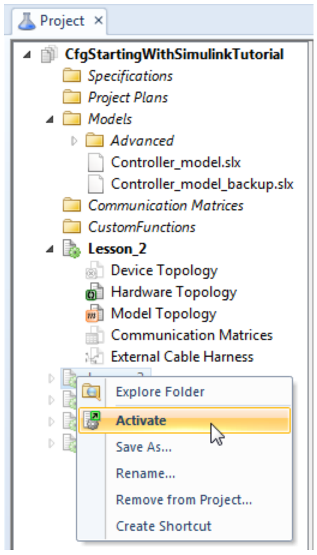
- The demo project is open.
- The Project Manager is displayed.

Method

To activate an application in the tutorial demo project

1 In the Project Manager, right-click the inactive application you want to activate.

2 From the context menu, select Activate as shown in the following screenshot.



Result

ConfigurationDesk activates the selected application and deactivates the application that was active before. You can now use the active application to complete further steps of the tutorial.

Optional Lesson: Resolving Conflicts

Where to go from here

Information in this section

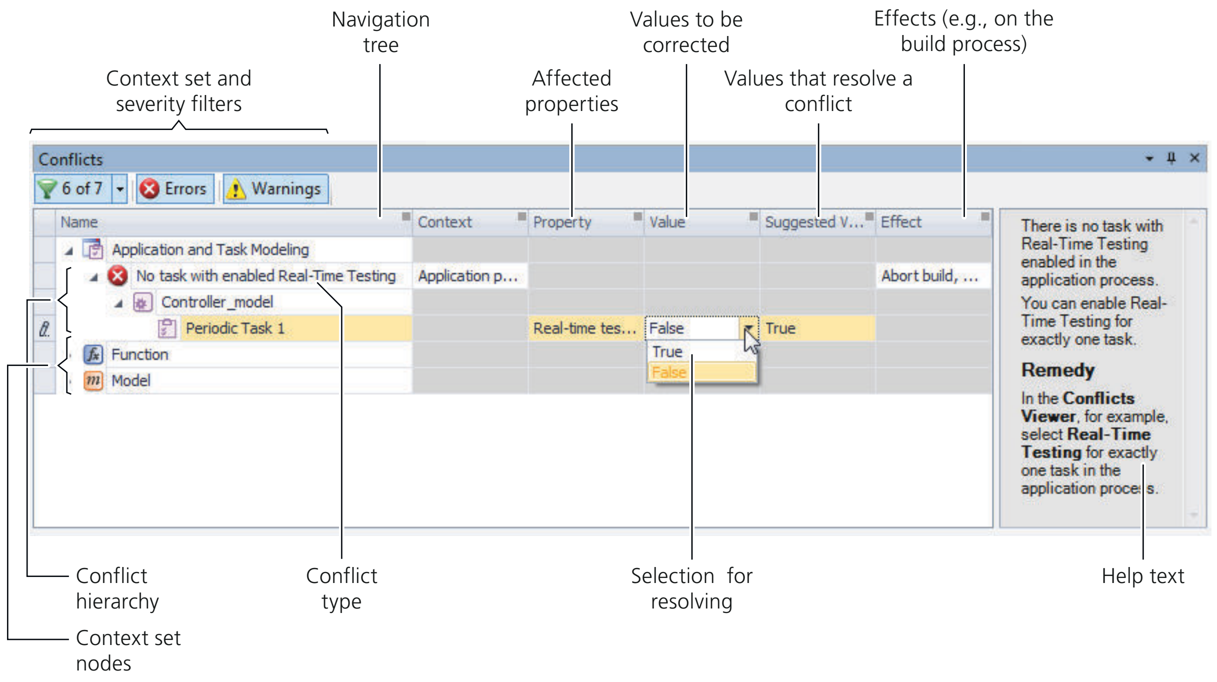
Overview of the Resolving Conflicts Lesson

Conflicts

Tip

Unlike in the other lessons of the tutorial, you cannot continue with your working results from previous lessons. Refer to Before you begin on page 48.

ConfigurationDesk allows for flexible configuration without strict constraints. This lets you work more freely, but it can lead to conflicting configuration settings. ConfigurationDesk automatically detects conflicts and provides the **Conflicts Viewer** to display and help resolve them. Before you build a real-time application, you have to resolve at least the most severe conflicts to get proper build results.



Name	Context	Property	Value	Suggested V…	Effect
Application and Task Modeling					
No task with enabled Real-Time Testing	Application p…				Abort build, …
Controller_model					
Periodic Task 1		Real-time tes…	False (True / False)	True	
Function					
Model					

There is no task with Real-Time Testing enabled in the application process.

You can enable Real-Time Testing for exactly one task.

Remedy

In the **Conflicts Viewer**, for example, select **Real-Time Testing** for exactly one task in the application process.

What will you do?

In this lesson you will:

- Analyze conflicts in the Conflicts Viewer and resolve them.

Before you begin

Before you begin this lesson, the following precondition must be met:

- You opened the `CfgStartingWithSimulinkTutorial` demo project and activated the ResolvingConflicts application. This application is a modified version of the Results application. For instructions, refer to:
  - How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
  - How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

Steps

This lesson contains the following step:

- Step 1: How to Resolve Conflicts on page 48

Step 1: How to Resolve Conflicts

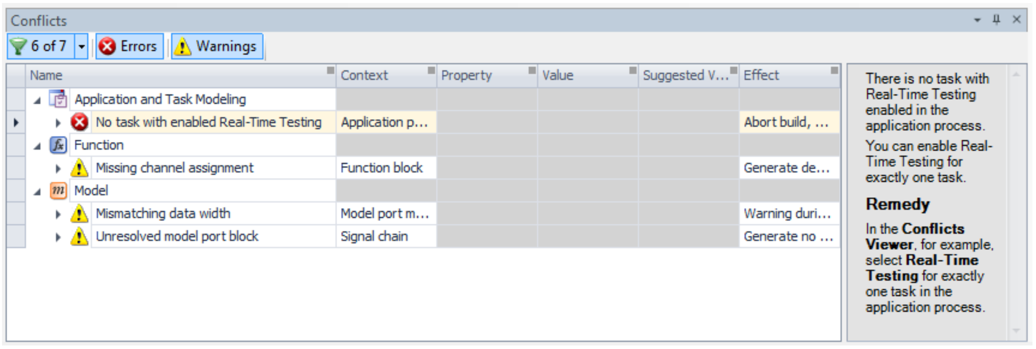
Objective

To indicate configuration problems, ConfigurationDesk generates conflicts. You can use the Conflicts Viewer to analyze and resolve the conflicts.

Method

To resolve conflicts

1 Open the Conflicts Viewer if necessary. It shows a number of conflicts.



Conflicts

6 of 7 | Errors | Warnings

Name	Context	Property	Value	Suggested V...	Effect
Application and Task Modeling					
No task with enabled Real-Time Testing	Application p...				Abort build, ...
Function					
Missing channel assignment	Function block				Generate de...
Model					
Mismatching data width	Model port m...				Warning duri...
Unresolved model port block	Signal chain				Generate no ...

There is no task with Real-Time Testing enabled in the application process.

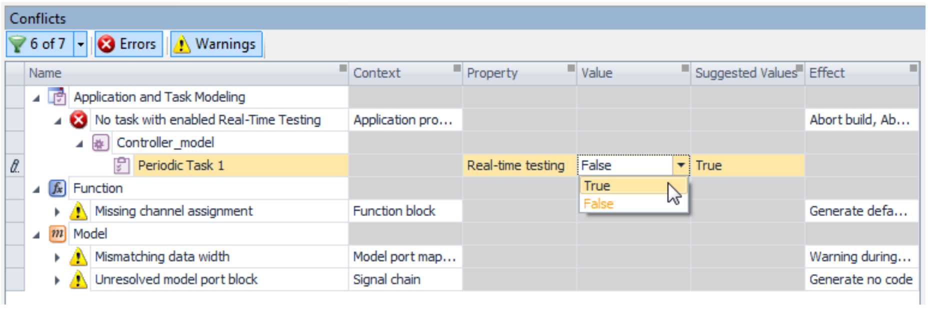
You can enable Real-Time Testing for exactly one task.

Remedy

In the **Conflicts Viewer**, for example, select **Real-Time Testing** for exactly one task in the application process.

2 Expand the Application and Task Modeling – No task with enabled Real-Time Testing conflict in the navigation tree.

3 In the Value column, set the Real-time testing property to True.



Conflicts

6 of 7 | Errors | Warnings

Name	Context	Property	Value	Suggested Values	Effect
Application and Task Modeling					
No task with enabled Real-Time Testing	Application pro...				Abort build, Ab...
Controller_model					
Periodic Task 1		Real-time testing	False	True	
Function			True		
Missing channel assignment	Function block		False		Generate defa...
Model					
Mismatching data width	Model port map...				Warning during...
Unresolved model port block	Signal chain				Generate no code

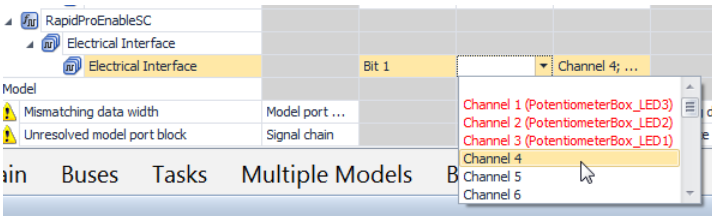
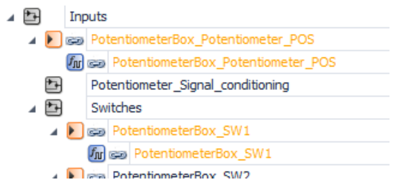
Tip

Real-time testing must be enabled for exactly one task. Usually the Real-time testing property is specified in the Task Configuration table of the Tasks view set.

You resolved the conflict and it is removed from the Conflicts Viewer.

4 Resolve the other conflicts with the help of the remedies described in the following table:

Conflict Type	Description/Remedy
Missing channel assignment	After expanding the conflict, you can see that channel assignments are missing for the electrical interfaces of two function blocks. For the RapidProEnableSC function block, select Channel 4 for the channel request Bit 1 in the Value column.

Conflict Type	Description/Remedy
	Note that some channels in the value list are highlighted in red. This means that they are already in use and selecting them will cause other channel assignment conflicts.  RapidProEnableSC Electrical Interface Electrical Interface · Bit 1 · Channel 4; ... Model Mismatching data width · Model port ... Unresolved model port block · Signal chain Channel 1 (PotentiometerBox_LED3) Channel 2 (PotentiometerBox_LED2) Channel 3 (PotentiometerBox_LED1) Channel 4 Channel 5 Channel 6 You can ignore the channel assignment conflict for the PotentiometerBox_SW1 function block. It will be resolved when you resolve a Mismatching data width conflict. This shows you that the same configuration might cause different conflicts. Sometimes, you have to decide which of the possible remedies is suitable for your use case.
Mismatching data width	After expanding the conflict, you can see that two model port mappings connect ports with mismatching values for the Data width property. Select a function or model port in the Conflicts Viewer to see the affected function or model port block in the Properties Browser. In the Model-Function Mapping Browser, the blocks are highlighted in yellow.  Inputs PotentiometerBox_Potentiometer_POS PotentiometerBox_Potentiometer_POS Potentiometer_Signal_conditioning Switches PotentiometerBox_SW1 PotentiometerBox_SW1 To resolve the conflict, the data width of either the function port or the model port has to be adjusted. However, you cannot change the value of the Data width property in the Conflicts Viewer. PotentiometerBox_SW1 (correct value in Simulink assumed): 1. In the Model-Function Mapping Browser, select the PotentiometerBox_SW1 function block. 2. In the Properties Browser on the General page, set the Number of bits property to 1. This leads to the following results: ▪ The Data width property of the function port is changed to 1 as well. ▪ One of the Mismatching data width conflicts is removed. ▪ The PotentiometerBox_SW1 function block and model port block are no longer highlighted. PotentiometerBox_Potentiometer_POS (correct value in ConfigurationDesk assumed): 1. In the Model-Function Mapping Browser, select the PotentiometerBox_Potentiometer_POS function block. 2. On the Home ribbon, click Models – Propagate. **Note** First, close any model that might be open from previous lessons. This leads to the following results: ▪ The Data width property of the model port block in ConfigurationDesk is changed to 1. ▪ The Controller_model Simulink model is opened if necessary. ▪ The Width property of the model port block in the Simulink model is changed to 1. ▪ The second Mismatching data width conflict is removed.

Conflict Type	Description/Remedy
	▪ The PotentiometerBox_Potentiometer_POS function block and model port block are no longer highlighted The channel assignment conflict for the PotentiometerBox_SW1 function block is now also removed.
Unresolved model port block	After expanding the conflict, you can see that the PotentiometerBox_LED1 model port block is affected. In the Model-Function Mapping Browser, the symbol marks the unresolved model port block. This means it is used in the ConfigurationDesk application, but not in the connected Simulink behavior model. The symbol at the Outputs subsystem and the Controller_model model also signals a discrepancy between the ConfigurationDesk application and the Simulink behavior model. To resolve the conflict: 1. In the Model-Function Mapping Browser, select the PotentiometerBox_LED1 model port block. 2. On the Home ribbon, click Models – Propagate. The conflict and the symbols are removed. However, the model port block is not connected in the Simulink behavior model yet. 3. In Simulink, go to the Outputs subsystem in the Controller_model model. 4. Connect the PotentiometerBox_LED1 model port block to the PotentiometerBox_led1 inport.

Note

For conflicts regarding discrepancies between ConfigurationDesk and the Simulink behavior model, always be aware where to apply the correct setting and which elements to propagate to the Simulink behavior model. For example, you could have resolved both Mismatching data width conflicts and the Unresolved model port block conflict in one step by selecting the Controller_model model in ConfigurationDesk and clicking Propagate. But in the tutorial scenario, this would not have led to the desired result for the PotentiometerBox_SW1 function block and model port block, because the correct setting in Simulink would have been overwritten.

Result

You resolved all conflicts.

Tip

The Conflicts Viewer provides various filters that help you focus on specific conflicts. This is particularly useful for large ConfigurationDesk applications with many conflicts. For more information, refer to Details on Filtering Conflicts (ConfigurationDesk Real-Time Implementation Guide 🕮).

Result of the Resolving Conflicts Lesson

Result

You analyzed conflicts and resolved them.

Now, your application is equivalent to the Lesson_4 application of the `CfgStartingWithSimulinkTutorial` demo project.

Further information

For detailed descriptions and instructions regarding conflicts, refer to Resolving Conflicts (ConfigurationDesk Real-Time Implementation Guide 🕮).

What's next

You finished the optional lesson on conflicts. To repeat a previous lesson or complete a lesson that you skipped, open the existing `CfgStartingWithSimulinkTutorial` demo project and activate the associated application. For instructions, refer to:

- How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
- How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

You can also:

- Complete the Advanced Lesson: Adding I/O Events on page 53.

For a summary of your working results, refer to Summary on page 59.

Advanced Lesson: Adding I/O Events

Where to go from here

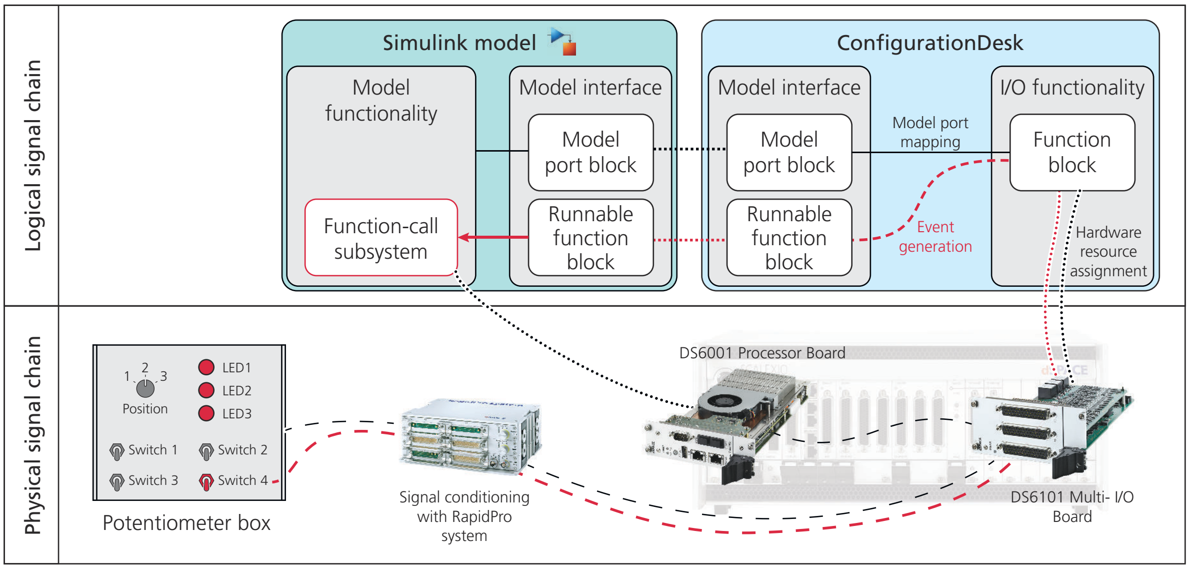
Information in this section

Overview of the Adding I/O Events Lesson

Triggering asynchronous tasks with I/O events

You can trigger asynchronous tasks in your real-time application by using I/O events that are provided by ConfigurationDesk function blocks. For this purpose, Runnable Function blocks are created in ConfigurationDesk and Simulink.

In the tutorial demo scenario, you will trigger a Simulink function-call subsystem with an additional switch at the potentiometer box. The switch sets all LEDs of the potentiometer box to On.



Tip

The hardware used in the tutorial use scenario is just an example. You can also use different dSPACE hardware as long as the required channels are provided.

What will you do?

In this lesson you will:

- Replace the connected Simulink behavior model in ConfigurationDesk with an advanced Simulink behavior model.
- Enable and configure event generation.

Before you begin

Before you begin this lesson, one of the following preconditions must be met:

- You completed lessons 1-4 of the tutorial in your `MyCfgStartingWithSimulinkTutorial` project.

or

- You opened the `CfgStartingWithSimulinkTutorial` demo project and activated the Results application. For instructions, refer to:
  - How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
  - How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

Steps

This lesson contains the following steps:

- Step 1: How to Replace the Connected Simulink Behavior Model on page 55
- Step 2: How to Enable and Configure Event Generation on page 56

Step 1: How to Replace the Connected Simulink Behavior Model

Objective

You can replace the connected Simulink behavior model in ConfigurationDesk without having to resynchronize elements and their configurations that are identical in the new model.

In the tutorial demo scenario, you replace the model for the basic use scenario with an advanced model containing an additional function-call subsystem.

Method

To replace the connected Simulink behavior model

1 In the Model-Function Mapping Browser, right-click an empty area and select Import – Replace Topology.

  The Import Model Topology dialog opens.

2 Click Add model and select Add model from file.

  A standard file selection dialog opens.

3 Select the `Controller_model.slx` file from the following folder:

```
<Documents folder>\Tutorials\CfgStartingWithSimulinkTutorial\Models\Advanced
```

4 Click Open.

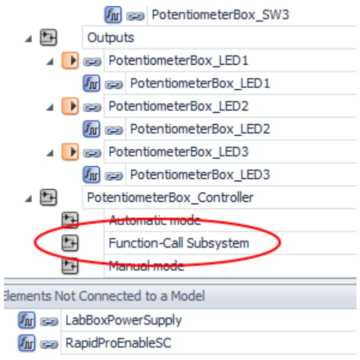
5 In the Import Model Topology dialog, click OK.

  Note

  First, close any model that might be open from previous lessons.

  ConfigurationDesk opens the model in Simulink and analyzes it before importing it. Elements with the same name and configuration as in the basic model stay synchronized with the advanced Simulink behavior model.

In the Model-Function Mapping Browser, you can see the added function-call subsystem:


PotentiometerBox_SW3
Outputs
PotentiometerBox_LED1
PotentiometerBox_LED1
PotentiometerBox_LED2
PotentiometerBox_LED2
PotentiometerBox_LED3
PotentiometerBox_LED3
PotentiometerBox_Controller
Automatic mode
Function-Call Subsystem
Manual mode
Elements Not Connected to a Model
LabBoxPowerSupply
RapidProEnableSC

Tip

If you use the same file name for the imported model, all application process assignments are automatically reused. Otherwise, the Conflicts Viewer will guide you through the necessary adaptations.

Result

You replaced the connected Simulink behavior model.

Step 2: How to Enable and Configure Event Generation

Objective

To trigger the function-call subsystem in the Simulink behavior model you need to enable and configure an I/O event in ConfigurationDesk.

Method

To enable and configure event generation

1 In the Function Browser, open the Basic I/O – Digital Input folder.

2 Drag the Multi Bit In function block type to the Elements Not Connected to a Model area in the Model-Function Mapping Browser.

A function block named Multi Bit In (1) is created.

3 Select the function block and specify the following settings in the Properties Browser:

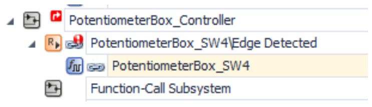
Page – Category in Properties Browser	Property	Value
General – Identification	Name	PotentiometerBox_SW4
General – Event Configuration	Event generation	Enabled[1)]
Electrical Interface – Hardware Assignment	Assigned channel set	DS6101 Multi I/O Board (Slot 12)\Digital In 3 [\LabBox (1)]

Page – Category in Properties Browser	Property	Value
Electrical Interface – Hardware Assignment	Bit 1 (channel request)	Channel 4
Model Interface – Behavior Model	Model access	Disabled[2]

[1] This creates an event port at the model interface of the function block.
[2] Only the function port's model access is disabled, not the event port's.

4 Drag the PotentiometerBox_SW4 function block from the Elements Not Connected to a Model area to the PotentiometerBox_Controller subsystem in the Model-Function Mapping Browser.

ConfigurationDesk creates an unresolved Runnable Function block named after the function block and its event port.



PotentiometerBox_Controller
PotentiometerBox_SW4\Edge Detected
PotentiometerBox_SW4
Function-Call Subsystem

Tip

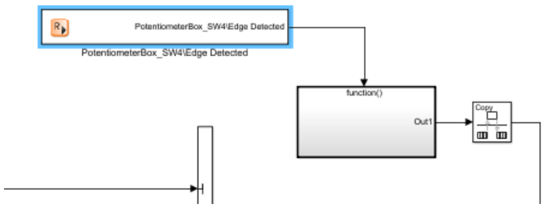
If Model access were set to Enabled for the function block, a model port block would be created in addition to the Runnable Function block. This is not required in the tutorial use scenario.

5 Right-click the PotentiometerBox_SW4\Edge Detected Runnable Function block and select Propagate to Simulink Model.

6 Right-click the PotentiometerBox_SW4\Edge Detected Runnable Function block and select Show – Show in Simulink Model.

ConfigurationDesk opens the Simulink model if necessary and selects and highlights the block in its subsystem.

7 Connect the PotentiometerBox_SW4\Edge Detected Runnable Function block to the Function-Call Subsystem.


PotentiometerBox_SW4\Edge Detected
PotentiometerBox_SW4\Edge Detected
function()
Out1
Copy

Result

You enabled and configured event generation.

In the tutorial demo scenario, using switch 4 will now cause all LEDs of the potentiometer box to be set to On.

Result of the Adding I/O Events Lesson

Result

You added an I/O event in ConfigurationDesk to trigger a function-call subsystem in an advanced Simulink behavior model.

Now, you could start the build process again as described in Lesson 4: Building the Real-Time Application on page 37.

Further information

For detailed descriptions and instructions, refer to Modeling Executable Applications and Tasks (ConfigurationDesk Real-Time Implementation Guide 📖).

What's next

You finished the advanced lesson on I/O events. To repeat a previous lesson or complete a lesson that you skipped, open the existing `CfgStartingWithSimulinkTutorial` demo project and activate the associated application. For instructions, refer to:

- How to Open the CfgStartingWithSimulinkTutorial Demo Project on page 44
- How to Activate an Application in the CfgStartingWithSimulinkTutorial Demo Project on page 45

You can also:

- Activate the **Resolving_Conflicts** application of the `CfgStartingWithSimulinkTutorial` demo project to complete the Optional Lesson: Resolving Conflicts on page 47.

For a summary of your working results, refer to Summary on page 59.

Summary

Your Working Results

What you learned

You learned:

- How to create a ConfigurationDesk project from a Simulink behavior model.
- How to add I/O functionality to your model in a ConfigurationDesk application.
- How to synchronize the ConfigurationDesk I/O functionality with the Simulink behavior model.
- How to prepare the build process and build an executable real-time application.
- How to show and resolve conflicts (only if you completed the optional lesson Resolving Conflicts).
- How to add I/O events (only if you completed the advanced lesson Adding I/O Events).

ConfigurationDesk Glossary

Introduction

The glossary briefly explains the most important expressions and naming conventions used in the ConfigurationDesk documentation.

Where to go from here

Information in this section

A

Application There are two types of applications in ConfigurationDesk:

- A part of a ConfigurationDesk project: ConfigurationDesk application.
- An application that can be executed on dSPACE real-time hardware: real-time application.

Application process A component of a processing unit application. An application process contains one or more tasks.

Application process component A component of an application process. The following application process components are available in the **Components** subfolder of an application process:

- Behavior models that are assigned to the application process, including their predefined tasks, runnable functions, and events.
- Function blocks that are assigned to the application process.

AutomationDesk A dSPACE software product for creating and managing any kind of automation tasks. Within the dSPACE tool chain, it is mainly used for automating tests on dSPACE hardware.

AUTOSAR system description file An AUTOSAR XML (ARXML) file that describes 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. The described network communication usually consists of more than one bus system (e.g., CAN, LIN, FlexRay).

B

Basic PDU A general term used in the documentation to address all the PDUs the Bus Manager supports, except for container IPDUs, multiplexed IPDUs, and secured IPDUs. Basic PDUs are represented by the ▶ or ▣ symbol in tables and browsers. The Bus Manager provides the same functionalities for all basic PDUs, such as ISignal IPDUs or NMPDUs.

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 nor access to the hardware. Behavior models can be modeled, for example, in MATLAB/Simulink by using Simulink Blocksets and Toolboxes from the MathWorks®.

You can add Simulink behavior models to a ConfigurationDesk application. You can also add code container files containing a behavior model such as Functional Mock-up Units, or Simulink implementation containers to a ConfigurationDesk application.

Bidirectional signal port A signal port that is independent of a data direction or current flow. This port is used, for example, to implement bus communication.

BSC file A bus simulation container file that is generated with the Bus Manager and contains the configured bus communication of one application process.

Build Configuration table A pane that lets you create build configuration sets and configure build settings, for example, build options, or the build and download behavior.

Build Log Viewer A pane that displays messages and warnings during the build process.

Build process A process that generates an executable real-time application based on your ConfigurationDesk application that can be run on a SCALEXIO system or MicroAutoBox III system. The build process can be controlled and configured via the Build Log Viewer. If the build process is successfully finished, the build result files (build results) are added to the ConfigurationDesk application.

Build results The files that are created during the build process. Build results are named after the ConfigurationDesk application and the application process from which they originate. You can access the build results in the Project Manager.

Bus access The representation of a run-time communication cluster. By assigning one or more bus access requests to a bus access, you specify which communication clusters form one run-time communication cluster.

In ConfigurationDesk, you can use a bus function block (CAN, LIN) to implement a bus access. The hardware resource assignment of the bus function block specifies the bus channel that is used for the bus communication.

Bus access request The representation of a request regarding the bus access. There are two sources for bus access requests:

- At least one element of a communication cluster is assigned to the Simulated ECUs, Inspection, or Manipulation part of a bus configuration. The related bus access requests contain the requirements for the bus channels that are to be used for the cluster's bus communication.

- A frame gateway is added to the **Gateways** part of a bus configuration. Each frame gateway provides two bus access requests that are required to specify the bus channels for exchanging bus communication.

Bus access requests are automatically included in BSC files. To build a real-time application, each bus access request must be assigned to a bus access.

Bus Access Requests table A pane that lets you access bus access requests of a ConfigurationDesk application and assign them to bus accesses.

Bus configuration A Bus Manager element that implements bus communication in a ConfigurationDesk application and lets you configure it for simulation, inspection, and/or manipulation purposes. The required bus communication elements must be specified in a communication matrix and assigned to the bus configuration. Additionally, a bus configuration lets you specify gateways for exchanging bus communication between communication clusters. A bus configuration can be accessed via specific tables and its related Bus Configuration function block.

Bus Configuration Function Ports table A pane that lets you access and configure function ports of bus configurations.

Bus Configurations table A pane that lets you access and configure bus configurations of a ConfigurationDesk application.

Bus Inspection Features table A pane that lets you access and configure bus configuration features of a ConfigurationDesk application for inspection purposes.

Bus Manager

- Bus Manager in ConfigurationDesk

  A ConfigurationDesk component that lets you configure bus communication and implement it in real-time applications or generate bus simulation containers.
- Bus Manager (stand-alone)

  A dSPACE software product based on ConfigurationDesk that lets you configure bus communication and generate bus simulation containers.

Bus Manipulation Features table A pane that lets you access and configure bus configuration features of a ConfigurationDesk application for manipulation purposes.

Bus simulation container A container that contains bus communication configured with the Bus Manager. Bus simulation container (BSC) files can be used in the VEOS Player and in ConfigurationDesk. In the VEOS Player, they let you implement the bus communication in an offline simulation application.

In ConfigurationDesk, they let you implement the bus communication in a real-time application independently from the Bus Manager.

Bus Simulation Features table A pane that lets you access and configure bus configuration features of a ConfigurationDesk application for simulation purposes.

Buses Browser A pane that lets you display and manage the communication matrices of a ConfigurationDesk application. For example, you can access communication matrix elements and assign them to bus configurations. This pane is available only if you work with the Bus Manager.

C

Cable harness A bundle of cables that provides the connection between the I/O connectors of the real-time hardware and the external devices, such as the ECUs to be tested. In ConfigurationDesk, it is represented by an external cable harness component.

CAFX file A ConfigurationDesk application fragment file that contains signal chain elements that were exported from a user-defined working view or the Temporary working view of a ConfigurationDesk application. This includes the elements' configuration and the mapping lines between them.

CDL file A ConfigurationDesk application file that contains links to all the documents related to an application.

Channel multiplication A feature that allows you to enhance the max. current or max. voltage of a single hardware channel by combining several channels. ConfigurationDesk uses a user-defined value to calculate the number of hardware channels needed. Depending on the function block type, channel multiplication is provided either for current enhancement (two or more channels are connected in parallel) or for voltage enhancement (two or more channels are connected in series).

Channel request A channel assignment required by a function block. ConfigurationDesk determines the type(s) and number of channels required for a function block according to the assigned channel set, the function block features, the block configuration and the required physical ranges. ConfigurationDesk provides a set of suitable and available hardware resources for each channel request. This set is produced according to the hardware topology added to the active ConfigurationDesk application. You have to assign each channel request to a specific channel of the hardware topology.

Channel set A number of channels of the same channel type located on the same I/O board or I/O unit. Channels in a channel set can be combined, for example, to provide a signal with channel multiplication.

Channel type A term to indicate all the hardware resources (channels) in the hardware system that provide exactly the same characteristics. Examples for channel type names: Flexible In 1, Digital Out 3, Analog In 1. An I/O board in a hardware system can have channel sets of several channel types. Channel sets of one channel type can be available on different I/O boards.

Cluster Communication cluster.

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>`

Communication cluster A communication network of network nodes that are connected to the same physical channels and share the same bus protocol and address range.

Communication matrix A file that defines the communication of a bus network. It can describe the bus communication of one communication cluster or a bus network consisting of different bus systems and clusters. Files of various file formats can be used as a communication matrix: For example, AUTOSAR system description files, DBC files, LDF files, and FIBEX files.

Communication package A package that bundles Data Inport blocks which are connected to Data Outport blocks. Hence, it also bundles the signals that are received by these blocks. If Data Inport blocks are executed within the same task and belong to the same communication package, their data inports are read simultaneously. If Data Outport blocks that are connected to the Data Inport blocks are executed in the same task, their output signals are sent simultaneously in one data package. Thus, communication packages guarantee simultaneous signal updates within a running task (data snapshot).

Configuration port A port that lets you create the signal chain for the bus communication implemented in a Simulink behavior model. The following configuration ports are available:

- The configuration port of a Configuration Port block.
- The Configuration port of a CAN, LIN, or FlexRay function block.

To create the signal chain for bus communication, the configuration port of a Configuration Port block must be mapped to the Configuration port of a CAN, LIN, or FlexRay function block.

Configuration Port block A model port block that is created in ConfigurationDesk during model analysis for each of the following blocks found in the Simulink behavior model:

- RTICANMM ControllerSetup block
- RTILINMM ControllerSetup block
- FLEXRAYCONFIG UPDATE block

Configuration Port blocks are also created for bus simulation containers. A Configuration Port block provides a configuration port that must be mapped to the Configuration port of a CAN, LIN, or FlexRay function block to create the signal chain for bus communication.

ConfigurationDesk application A part of a ConfigurationDesk project that represents a specific implementation. You can work with only one application at a time, and that application must be activated.

An application can contain:

- Device topology
- Hardware topology
- Model topology
- Communication matrices
- External cable harness
- Build results (after a successful build process has finished)

You can also add folders with application-specific files to an application.

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.

Conflict A result of conflicting configuration settings that is displayed in the Conflicts Viewer. ConfigurationDesk allows flexible configuration without strict constraints. This lets you work more freely, but it can lead to conflicting configuration settings. ConfigurationDesk automatically detects conflicts and provides the Conflicts Viewer to display and help resolve them. Before you build a real-time application, you have to resolve at least the most severe conflicts (e.g., errors that abort the build process) to get proper build results.

Conflicts Viewer A pane that displays the configuration conflicts that exist in the active ConfigurationDesk application. You can resolve most of the conflicts directly in the Conflicts Viewer.

Container IPDU A term according to AUTOSAR. An IPDU that contains one or more other IPDUs (i.e., contained IPDUs). When a container IPDU is mapped to a frame, all its contained IPDUs are included in that frame as well.

ControlDesk A dSPACE software product for managing, instrumenting and executing experiments for ECU development. ControlDesk also supports calibration, measurement and diagnostics access to ECUs via standardized protocols such as CCP, XCP, and ODX.

CTLGZ file A ZIP file that contains a V-ECU implementation. CTLGZ files are exported by TargetLink or SystemDesk. You can add a V-ECU implementation based on a CTLGZ file to the model topology just like adding a Simulink model based on an SLX file.

Cycle time restriction A value of a runnable function that indicates the sample time the runnable function requires to achieve correct results. The cycle time restriction is indicated by the Period property of the runnable function in the Properties Browser.

D

Data inport A port that supplies data from ConfigurationDesk's function outports to the behavior model.

In a multimodel application, data inports also can be used to provide data from a data outport associated to another behavior model (model communication).

Data outport A port that supplies data from behavior model signals to ConfigurationDesk's function inports.

In a multimodel application, data outports also can be used to supply data to a data inport associated to another behavior model (model communication).

DBC file A Data Base Container file that describes CAN or LIN bus systems. Because the DBC file format was primarily developed to describe CAN networks, it does not support definitions of LIN masters and schedules.

Device block A graphical representation of devices from the device topology in the signal chain. It can be mapped to function blocks via device ports.

Device connector A structural element that lets you group device pins in a hierarchy in the External Device Connectors table to represent the structure of the real connector of your external device.

Device pin A representation of a connector pin of your external device. Device ports are assigned to device pins. ConfigurationDesk can use the device pin assignment together with the hardware resource assignment and the device port mapping to calculate the external cable harness.

Device port An element of a device topology that represents the signal of an external device in ConfigurationDesk.

Device port group A structural element of a device topology that can contain device ports and other device port groups.

Device topology A component of a ConfigurationDesk application that represents external devices in ConfigurationDesk. You can create a device topology from scratch or easily extend an existing device topology. You can also merge device topologies to extend one. To edit or create device topologies independently of ConfigurationDesk, you can export and import DTFX and XLSX files.

Documents folder A standard folder for user-specific documents.

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

DSA file A dSPACE archive file that contains a ConfigurationDesk application and all the files belonging to it as one unit. It can later be imported to another ConfigurationDesk project.

dSPACE Help The dSPACE online help that contains all the relevant user documentation for dSPACE products. Via the **F1** key or the **Help** button in the dSPACE software you get context-sensitive help on the currently active context.

dSPACE Log A collection of errors, warnings, information, questions, and advice issued by all dSPACE products and connected systems over more than one session.

DTFX file A device topology export file that contains information on the interface to the external devices, such as the ECUs to be tested. The information includes details of the available device ports, their characteristics, and the assigned pins.

E

ECHX file An external cable harness file that contains the wiring information for the external cable harness. The external cable harness is the connection between the I/O connectors of the real-time hardware and the devices to be tested, for example, ECUs.

ECU Abbreviation of *electronic control unit*.

An embedded computer system that consists of at least one CPU and associated peripherals. An ECU contains communication controllers and communication connectors, and usually communicates with other ECUs of a bus network. An ECU can be member of multiple bus systems and communication clusters.

ECU application An application that is executed on an ECU. In ECU interfacing scenarios, parts of the ECU application can be accessed (e.g., by a real-time application) for development and testing purposes.

ECU function A function of an ECU application that is executed on the ECU. In ECU interfacing scenarios, an ECU function can be accessed by functions that are part of a real-time application, for example.

ECU Interface Manager A dSPACE software product for preparing ECU applications for ECU interfacing. The ECU Interface Manager can generate ECU interface container (EIC) files to be used in ConfigurationDesk.

ECU interfacing A generic term for methods and tools to read and/or write individual ECU functions and variables of an ECU application. In ECU interfacing scenarios, you can access ECU functions and variables for development and testing purposes while the ECU application is executed on the ECU. For example, you can perform ECU interfacing with SCALEXIO systems or MicroAutoBox III systems to access individual ECU functions by a real-time application.

EIC file An ECU interface container file that is generated with the ECU Interface Manager and describes an ECU application that is configured for ECU interfacing. You can import EIC files to ConfigurationDesk to perform ECU interfacing with SCALEXIO systems or MicroAutoBox III systems.

Electrical interface unit A segment of a function block that provides the interface to the external devices and to the real-time hardware (via hardware resource assignment). Each electrical interface unit of a function block usually needs a channel set to be assigned to it.

Event A component of a ConfigurationDesk application that triggers the execution of a task. The following event types are available:

- Timer event
- I/O event
- Software event

Event port An element of a function block. The event port can be mapped to a runnable function port for modeling an asynchronous task.

Executable application The generic term for real-time applications and offline simulation applications. In ConfigurationDesk, an executable application is always a real-time application since ConfigurationDesk does not support offline simulation applications.

Executable application component A component of an executable application. The following components can be part of an executable application:

- Imported behavior models including predefined tasks, runnable functions, and events. You can assign these behavior models to application processes via drag & drop or by selecting the **Assign Model** command from the context menu of the relevant application process.
- Function blocks added to your ConfigurationDesk application including associated I/O events. Function blocks are assigned to application processes via their model port mapping.

Executable Application table A pane that lets you model executable applications (i.e., real-time applications) and the tasks used in them.

EXPSWCFG file An experiment software configuration file that contains configuration data for automotive fieldbus communication. It is created during the build process and contains the data in XML format.

External cable harness A component of a ConfigurationDesk application that contains the wiring information for the external cable harness (also known as cable harness). It contains only the logical connections and no additional information such as cable length, cable diameters, dimensions or the arrangement of connection points, etc. It can be calculated by ConfigurationDesk or imported from a file so that you can use an existing cable harness and do not have to build a new one. The wiring information can be exported to an ECHX file or XLSX file.

External device A device that is connected to the dSPACE hardware, such as an ECU or external load. The external device topology is the basis for using external devices in the signal chain of a ConfigurationDesk application.

External Device Browser A pane that lets you display and manage the device topology of your active ConfigurationDesk application.

External Device Configuration table A pane that lets you access and configure the most important properties of device topology elements via table.

External Device Connectors table A pane that lets you specify the representation of the physical connectors of your external device including the device pin assignment.

F

FIBEX file An XML file according the ASAM MCD-2 NET standard (also known as Field Bus Exchange Format) defined by ASAM. The file can describe more than one bus system (e.g., CAN, LIN, FlexRay). It is used for data exchange between different tools that work with message-oriented bus communication.

Find Results Viewer A pane that displays the results of searches you performed via the Find command.

FMU file A Functional Mock-up Unit file that describes and implements the functionality of a model. It is an archive file with the file name extension FMU. The FMU file contains:

- The functionality defined as a set of C functions provided either in source or in binary form.
- The model description file (`modelDescription.xml`) with the description of the interface data.
- Additional resources needed for simulation.

You can add an FMU file to the model topology just like adding a Simulink model based on an SLX file.

Frame A piece of information of a bus communication. It contains an arbitrary number of non-overlapping PDUs and the data length code (DLC). CAN frames and LIN frames can contain only one PDU. To exchange a frame via bus channels, a frame triggering is needed.

Frame triggering An instance of a frame that is exchanged via a bus channel. It includes transmission information of the frame (e.g., timings, ID, sender, receiver). The requirements regarding the frame triggerings depend on the bus system (CAN, LIN, FlexRay).

Function block A graphical representation in the signal chain that is instantiated from a function block type. A function block provides the I/O functionality and the connection to the real-time hardware. It serves as a container for functions, for electrical interface units and their logical signals. The function block's ports (function ports and/or signal ports), provide the interfaces to the neighboring blocks in the signal chain.

Function block type A software plug-in that provides a specific I/O functionality. Every function block type has unique features which are different from other function block types.

To use a function block type in your ConfigurationDesk application, you have to create an instance of it. This instance is called a function block. Instances of function block types can be used multiple times in a ConfigurationDesk application. The types and their instantiated function blocks are displayed in the function library of the Function Browser.

Function Browser A pane that displays the function library in a hierarchical tree structure. Function block types are grouped in function classes. Instantiated function blocks are added below the corresponding function block type.

Function inport A function port that inputs the values from the behavior model to the function block to be processed by the function.

Function library A collection of function block types that allows access to the I/O functionality in ConfigurationDesk. The I/O functionality is based on function block types. The function library provides a structured tree view on the available function block types. It is displayed in the Function Browser.

Function outport A function port that outputs the value of a function to be used in the behavior model.

Function port An element of a function block that provides the interface to the behavior model via model port blocks.

Functional Mock-up Unit An archive file that describes and implements the functionality of a model based on the Functional Mock-up Interface (FMI) standard.

G

Global working view The default working view that always contains all signal chain elements.

H

Hardware resource A hardware element (normally a channel on an I/O board or I/O unit) which is required to execute a function block. A hardware resource can be localized unambiguously in a hardware system. Every hardware resource has specific characteristics. A function block therefore needs a hardware resource that matches the requirements of its functionality. This means that not every function block can be executed on every hardware resource. There could be limitations on a function block's features and/or the physical ranges.

Hardware resource assignment An action that assigns the electrical interface unit of a function block to one or more hardware resources. Function blocks can be assigned to any hardware resource which is suitable for the functionality and available in the hardware topology of your ConfigurationDesk application.

Hardware Resource Browser A pane that lets you display and manage all the hardware components of the hardware topology that is contained in your active ConfigurationDesk application in a hierarchical structure.

Hardware topology A component of a ConfigurationDesk application that contains information on a specific hardware system which can be used with ConfigurationDesk. It provides information on the components of the system, such as channel type and slot numbers. It can be scanned automatically from a registered platform, created in ConfigurationDesk's Hardware Resource Browser from scratch, or imported from an HTFX file.

HTFX file A file containing the hardware topology after an explicit export. It provides information on the components of the system and also on the channel properties, such as board and channel types and slot numbers.

I/O event An asynchronous event triggered by I/O functions. You can use I/O events to trigger tasks in your application process asynchronously. You can assign the events to the tasks via drag & drop, via the Properties Browser if you have selected a task, or via the Assign Event command from the context menu of the relevant task.

Interface model A temporary Simulink model that contains 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.

Interpreter A pane that lets you run Python scripts and execute line-based commands.

Inverse model port block A model port block that has the same configuration (same name, same port groups, and port names) but the inverse data direction as the original model port block from which it was created.

IOCNET Abbreviation of I/O carrier network.

A dSPACE proprietary protocol for internal communication in a SCALEXIO system between the real-time processors and I/O units. The IOCNET lets you connect more than 100 I/O nodes and place the parts of your SCALEXIO system long distances apart.

IPDU Abbreviation of interaction layer protocol data unit.

A term according to AUTOSAR. An IPDU contains the communication data that is routed from the interaction layer to a lower communication layer and vice versa. An IPDU can be implemented, for example, as an ISignal IPDU, multiplexed IPDU, or container IPDU.

ISignal A term according to AUTOSAR. A signal of the interaction layer that contains communication data as a coded signal value. To transmit the communication data on a bus, ISignals are instantiated and included in ISignal IPDUs.

ISignal IPDU A term according to AUTOSAR. An IPDU whose communication data is arranged in ISignals. ISignal IPDUs allow the exchange of ISignals between different network nodes.

L

LDF file A LIN description file that describes networks of the LIN bus system according to the LIN standard.

LIN master A member of a LIN communication cluster that is responsible for the timing of LIN bus communication. A LIN master provides one LIN master task and one LIN slave task. The LIN master task transmits frame headers on the bus, and provides LIN schedule tables and LIN collision resolver tables. The LIN slave task transmits frame responses on the bus. A LIN cluster must contain exactly one LIN master.

LIN schedule table A table defined for a LIN master that contains the transmission sequence of frame headers on a LIN bus. For each LIN master, several LIN schedule tables can be defined.

LIN slave A member of a LIN communication cluster that provides only a LIN slave task. The LIN slave task transmits frame responses on the bus when they are triggered by a frame header. The frame header is sent by a LIN master. A LIN cluster can contain several LIN slaves.

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>
```

Logical signal An element of a function block that combines all the signal ports which belong together to provide the functionality of the signal. Each logical signal causes one or more channel requests. Channel requests are available after you have assigned a channel set to the logical signal.

Logical signal chain A term that describes the logical path of a signal between an external device and the behavior model. The main elements of the logical signal chain are represented by different graphical blocks (device blocks, function blocks and model port blocks). Every block has ports to provide the mapping to neighboring blocks.
In the documentation, usually the short form 'signal chain' is used instead.

M

MAP file A file that maps symbolic names to physical addresses.

Mapping line A graphical representation of a connection between two ports in the signal chain. You can draw mapping lines in a working view.

MCD file A model communication description file that is used to implement a multimodel application. It lets you add several behavior models that were separated from an overall model to the model topology.

The MCD file contains information on the separated models and information on the connections between them. The file is generated with the Model Separation Setup Block in MATLAB/Simulink. The block resides in the Model Interface Blockset (dsmpblib) from dSPACE.

MDL file A Simulink model file that contains the behavior model. You can add an MDL file to your ConfigurationDesk application.

As of MATLAB® R2012a, the file name extension for the Simulink model file has been changed from MDL to SLX by The MathWorks®.

Message Viewer A pane that displays a history of all error and warning messages that occur during work with ConfigurationDesk.

Model analysis A process that analyzes the model to determine the interface of a behavior model. You can select one of the following commands:

- Analyze Simulink Model (Model Interface Only)

  Analyzes the interface of a behavior model. The model topology of your active ConfigurationDesk application is updated with the properties of the analyzed behavior model.

- Analyze Simulink Model (Including Task Information)

  Analyzes the model interface and the elements of the behavior model that are relevant for the task configuration. The task configuration in ConfigurationDesk is then updated accordingly.

Model Browser A pane that lets you display and access the model topology of an active ConfigurationDesk application. The Model Browser provides access to all the model port blocks available in the behavior models which are linked to a ConfigurationDesk application. The model elements are displayed in a hierarchy, starting with the model roots. Below the model root, all the subsystems containing model port blocks are displayed as well as the associated model port blocks.

Model communication The exchange of signal data between the models within a multimodel application. To set up model communication, you must use a mapping line to connect a data outport (sending model) to a data inport (receiving model). The best way to set up model communication is using the Model Communication Browser.

Model Communication Browser A pane that lets you open and browse working views like the **Signal Chain Browser**, but shows only the Data Outport and Data Inport blocks and the mapping lines between them.

Model Communication Package table A pane that lets you create and configure model communication packages which are used for model communication in multimodel applications.

Model implementation An implementation of a behavior model. It can consist of source code files, precompiled objects or libraries, variable description files and a description of the model's interface. Specific model implementation types are, for example, model implementation containers, such as Functional Mock-up Units or Simulink implementation containers.

Model implementation container A file archive that contains a model implementation. Examples are FMUs, SIC files, and VECU files.

Model interface An interface that connects ConfigurationDesk with a behavior model. This interface is part of the signal chain and is implemented via model port blocks. The model port blocks in ConfigurationDesk can provide the interface to:

- Model port blocks (from the Model Interface Package for Simulink) in a Simulink behavior model. In this case, the model interface is also called ConfigurationDesk model interface to distinguish it from the Simulink model interface available in the Simulink behavior model.
- Different types of model implementations based on code container files, e.g., Simulink implementation containers, Functional Mock-up Units, and V-ECU implementations.

Model Interface Package for Simulink A dSPACE software product that lets you specify the interface of a behavior model that you can directly use in ConfigurationDesk. You can also create a code container file (SIC file) that contains the model code of a Simulink behavior model. The SIC file can be used in ConfigurationDesk and VEOS Player.

Model port An element of a model port block. Model ports provide the interface to the function ports and to other model ports (in multimodel applications).

These are the types of model ports:

- Data inport
- Data outport
- Runnable function port
- Configuration port

Model port block A graphical representation of the ConfigurationDesk model interface in the signal chain. Model port blocks contain model ports that can be mapped to function blocks to provide the data flow between the function blocks in ConfigurationDesk and the behavior model. The model ports can also be mapped to the model ports of other model port blocks with data inports or data outports to set up model communication. Model port blocks are available in different types and can provide different port types:

- Data port blocks with data inports and data outports
- Runnable Function blocks with runnable function ports
- Configuration Port blocks with configuration ports. Configuration Port blocks are created during model analysis for each of the following blocks found in the Simulink behavior model:
  - RTICANMM ControllerSetup block
  - RTILINMM ControllerSetup block
  - FLEXRAYCONFIG UPDATE block

  Configuration Port blocks are also created for bus simulation containers.

Model Separation Setup Block A block that is contained in the Model Interface Package for Simulink. It is used to separate individual models from an overall model in MATLAB/Simulink. Additionally, model separation generates 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.

Model topology A component of a ConfigurationDesk application that contains information on the subsystems and the associated model port blocks of all the behavior models that have been added to a ConfigurationDesk application.

Model-Function Mapping Browser A pane that lets you create and update signal chains for Simulink behavior models. It directly connects them to I/O functionality in ConfigurationDesk.

MTFX file A file containing a model topology when explicitly exported. The file contains information on the interface to the behavior model, such as the implemented model port blocks including their subsystems and where they are used in the model.

Multicore real-time application A real-time application that is executed on several cores of one PU of the real-time hardware.

Multimodel application A real-time application that executes several behavior models in parallel on dSPACE real-time hardware (SCALEXIO or MicroAutoBox III).

Multiplexed IPDU A term according to AUTOSAR. An IPDU that consists of one dynamic part, a selector field, and one optional static part. Multiplexing is used to transport varying ISignal IPDUs via the same bytes of a multiplexed IPDU.

- The dynamic part is one ISignal IPDU that is selected for transmission at run time. Several ISignal IPDUs can be specified as dynamic part alternatives. One of these alternatives is selected for transmission.

- The selector field value indicates which ISignal IPDU is transmitted in the dynamic part during run time. For each selector field value, there is one corresponding ISignal IPDU of the dynamic part alternatives. The selector field value is evaluated by the receiver of the multiplexed IPDU.
- The static part is one ISignal IPDU that is always transmitted.

Multi-PU application Abbreviation of multi-processing-unit application. A multi-PU application is a real-time application that is partitioned into several processing unit applications. Each processing unit application is executed on a separate PU of the real-time hardware. The processing units are connected via IOCNET and can be accessed from the same host PC.

N

Navigation bar An element of ConfigurationDesk's user interface that lets you switch between view sets.

Network node A term that describes the bus communication of an ECU for only one communication cluster.

O

Offline simulation A purely PC-based simulation scenario without a connection to a physical system, i.e., neither simulator hardware nor ECU hardware prototypes are needed. Offline simulations are independent from real time and can run on VEOS.

Offline simulation application An application that runs on VEOS to perform offline simulation. An offline simulation application can be built with the VEOS Player and the resulting OSA file can be downloaded to VEOS.

OSA file An offline simulation application file that is built with the VEOS Player and can be downloaded to VEOS to perform offline simulation.

P

Parent port A port that you can use to map multiple function ports and model ports. All child ports with the same name are mapped. ConfigurationDesk enforces the mapping rules and allows only mapping lines which agree with them.

PDU Abbreviation of protocol data unit.

A term according to AUTOSAR. A PDU transports data (e.g., control information or communication data) via one or multiple network layers according to the AUTOSAR layered architecture. Depending on the involved layers and the function of a PDU, various PDU types can be distinguished, e.g., ISignal IPDUs, multiplexed IPDUs, and NMPDUs.

Physical signal chain A term that describes the electrical wiring of external devices (ECU and loads) to the I/O boards of the real-time hardware. The physical signal chain includes the external cable harness, the pinouts of the connectors and the internal cable harness.

Pins and External Wiring table A pane that lets you access the external wiring information

Platform A dSPACE real-time hardware system that can be registered and displayed in the Platform Manager.

Platform Manager A pane that lets you handle registered hardware platforms. You can download, start, and stop real-time applications via the Platform Manager. You can also update the firmware of your SCALEXIO system or MicroAutoBox III system.

Preconfigured application process An application process that was created via the Create preconfigured application process command. If you use the command, ConfigurationDesk creates new tasks for each runnable function provided by the model which is not assigned to a predefined task. ConfigurationDesk assigns the corresponding runnable function and (for periodic tasks) timer events to the new tasks. The tasks are preconfigured (e.g., DAQ raster name, period).

Processing Resource Assignment table A pane that lets you configure and inspect the processing resources in an executable application. This table is useful especially for multi-processing-unit applications.

Processing unit application A component of an executable application. A processing unit application contains one or more application processes.

Project A container for ConfigurationDesk applications and all project-specific documents. You must define a project or open an existing one to work with ConfigurationDesk. Projects are stored in a project root folder.

Project Manager A pane that provides access to ConfigurationDesk projects and applications and all the files they contain.

Project root folder A folder on your file system to which ConfigurationDesk saves all project-relevant data, such as the applications and documents of a project. Several projects can use the same project root folder. ConfigurationDesk uses the Documents folder as the default project root folder. You can specify further project root folders. Each can be made the default project root folder.

Properties Browser A pane that lets you access the properties of selected elements.

PU Abbreviation of processing unit.

A hardware assembly that consists of a motherboard or a dSPACE processor board, possibly additional interface hardware for connecting I/O boards, and an enclosure, i.e., a SCALEXIO Real-Time PC.

R

Real-time application An application that can be executed in real time on dSPACE real-time hardware. The real-time application is the result of a build process. It can be downloaded to real-time hardware via an RTA file. There are different types of real-time applications:

- Single-core real-time application.
- Multicore real-time application.
- Multi-PU application.

Restbus simulation A simulation method to test real ECUs by connecting them to a simulator that simulates the other ECUs in the communication clusters.

RTA file A real-time application file. An RTA file is an executable object file for processor boards. It is created during the build process. After the build process it can be downloaded to the real-time hardware.

Runnable function A function that is called by a task to compute results. A model implementation provides a runnable function for its base rate task. This runnable function can be executed in a task that is triggered by a timer event. In addition, a Simulink behavior model provides a runnable function for each Hardware-Triggered Runnable Function block contained in the Simulink behavior model. This runnable function is suitable for being executed in an asynchronous task.

Runnable Function block A type of model port block. A Runnable Function block provides a runnable function port that can be mapped to an event port of a function block for modeling an asynchronous task.

Runnable function port An element of a Runnable Function block. The runnable function port can be mapped to an event port of a function block for modeling an asynchronous task.

RX Communication data that is received by a bus member.

S

SCALEXIO system A dSPACE hardware-in-the-loop (HIL) system consisting of one or more real-time industry PCs (PUs), I/O boards, and I/O units. They communicate with each other via the IOCNET. The system simulates the environment to test an ECU. It provides the sensor signals for the ECU, measures the signals of the ECU, and provides the power (battery voltage) for the ECU and a bus interface for restbus simulation.

SDF file A system description file that contains information on the CPU name(s), the corresponding object file(s) to be downloaded and the corresponding variable description file(s). It is created during the build process.

Secured IPDU A term according to AUTOSAR. An IPDU that secures the payload of another PDU (i.e., authentic IPDU) for secure onboard communication (SecOC). The secured IPDU contains the authentication information that is used to secure the authentic IPDU's payload. If the secured IPDU is configured as a cryptographic IPDU, the secured IPDU and the authentic IPDU are mapped to different frames. If the secured IPDU is not configured as a cryptographic IPDU, the authentic IPDU is directly included in the secured IPDU.

SIC file A Simulink implementation container file that contains the model code of a Simulink behavior model. The SIC file can be used in ConfigurationDesk and in VEOS Player.

Signal chain A term used in the documentation as a short form for logical signal chain. Do not confuse it with the physical signal chain.

Signal Chain Browser A pane that lets you open and browse working views such as the Global working view or user-defined working views.

Signal inport A signal port that represents an electrical connection point of a function block which provides signal measurement (= input) functionality.

Signal outport A signal port that represents an electrical connection point of a function block which provides signal generation (= output) functionality.

Signal port An element of a function block that provides the interface to external devices (e.g., ECUs) via device blocks. It represents an electrical connection point of a function block.

Signal reference port A signal port that represents a connection point for the reference potential of inports, outports and bidirectional ports. For example: With differential signals, this is a reference signal, with single-ended signals, it is the ground signal (GND).

Simulink implementation container A container that contains the model code of a Simulink behavior model. A Simulink implementation container is generated from a Simulink behavior model by using the Model Interface Package for Simulink. 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 connected Simulink behavior model.

Single-core real-time application An executable application that is executed on only one core of the real-time hardware.

Single-PU system Abbreviation of single-processing-unit system.

A system consisting of exactly one PU and the directly connected I/O units and I/O routers.

SLX file A Simulink model file that contains the behavior model. You can add an SLX file to your ConfigurationDesk application.

As of MATLAB® R2012a, the file name extension for the Simulink model file has been changed from MDL to SLX by The MathWorks®.

Software event An event that is activated from within a task to trigger another subordinate task. Consider the following example: A multi-tasking Simulink behavior model has a base rate task with a sample time = 1 ms and a periodic task with a sample time = 4 ms. In this case, the periodic task is triggered on every fourth execution of the base rate task via a software event. Software events are available in ConfigurationDesk after model analysis.

Source Code Editor A Python editor that lets you open and edit Python scripts that you open from or create in a ConfigurationDesk project in a window in the working area. You cannot run a Python script in a **Source Code Editor** window. To run a Python script you can use the **Run Script** command in the Interpreter or on the **Automation** ribbon or the **Run** context menu command in the Project Manager.

Structured data port A hierarchically structured port of a Data Inport block or a Data Outport block. Each structured data port consists of more structured and/or unstructured data ports. The structured data ports can consist of signals with different data types (single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, Boolean).

SystemDesk A dSPACE software product for development of distributed automotive electrics/electronics systems according to the AUTOSAR approach.

SystemDesk is able to provide a V-ECU implementation container (as a VECU file) to be used in ConfigurationDesk.

T

Table A type of pane that offers access to a specific subset of elements and properties of the active ConfigurationDesk application in rows and columns.

TargetLink A dSPACE software product for production code generation. It lets you generate highly efficient C code for microcontrollers in electronic control units (ECUs). It also helps you implement control systems that have been modeled graphically in a Simulink/Stateflow diagram on a production ECU.

TargetLink is able to provide a V-ECU implementation container (as a VECU file) or a Simulink implementation container (SIC file) to be used in ConfigurationDesk.

Task A piece of code whose execution is controlled by a real-time operating system (RTOS). A task is usually triggered by an event, and executes one or more runnable functions. In a ConfigurationDesk application, there are predefined tasks that are provided by executable application components. In addition, you can create user-defined tasks that are triggered, for example, by I/O events. Regardless of the type of task, you can configure the tasks by specifying the priority, the DAQ raster name, etc.

Task Configuration table A pane that lets you configure the tasks of an executable application.

Temporary working view A working view that can be used for drafting a signal chain segment, like a notepad.

Timer event A periodic event with a sample rate and an optional offset.

Topology A hierarchy that serves as a repository for creating a signal chain. All the elements of a topology can be used in the signal chain, but not every element needs to be used. You can export each topology from and import it to a ConfigurationDesk application. Therefore a topology can be used in several applications.

A ConfigurationDesk application can contain the following topologies:

- Device topology
- Hardware topology
- Model topology

TRC file A variable description file that contains all variables (signals and parameters) which can be accessed via the experiment software. It is created during the build process.

TX Communication data that is transmitted by a bus member.

U

User function An external function or program that is added to the Automation – User Functions ribbon group for quick and easy access during work with ConfigurationDesk.

V

VECU file A ZIP file that contains a V-ECU implementation. A VECU file can contain data packages for different platforms. VECU files are exported by TargetLink or SystemDesk. You can add a V-ECU implementation based on a VECU file to the model topology in the same way as adding a Simulink model based on an SLX file.

VEOS The dSPACE software product for performing offline simulation. VEOS is a PC-based simulation platform which allows offline simulation independently from real time.

VEOS Player A software running on the host PC for building offline simulation applications. Offline simulation applications can be downloaded to VEOS to perform offline simulation. ConfigurationDesk lets you generate a bus simulation container (BSC file) via the Bus Manager. You can then import the BSC file into the VEOS Player.

View set A specific arrangement of some of ConfigurationDesk's panes. You can switch between view sets by using the navigation bar. ConfigurationDesk provides preconfigured view sets for specific purposes. You can customize existing view sets and create user-defined view sets.

VSET file A file that contains all view sets and their settings from the current ConfigurationDesk installation. A VSET file can be exported and imported via the View Sets page of the Customize dialog.

W

Working area The central area of ConfigurationDesk's user interface.

Working view A view of the signal chain elements (blocks, ports, mappings, etc.) used in the active ConfigurationDesk application. A working view can be opened in the Signal Chain Browser or the Model Communication Browser. ConfigurationDesk provides two default working views: The Global working view and the Temporary working view. In the Working View Manager, you can also create user-defined working views that let you focus on specific signal chain segments according to your requirements.

Working View Manager A pane that lets you manage the working views of the active ConfigurationDesk application. You can use the Working View Manager for creating, renaming, and deleting working views, and also to open a working view in the Signal Chain Browser or the Model Communication Browser.

X

XLSX file A Microsoft Excel™ file format that is used for the following purposes:

- Creating or configuring a device topology outside of ConfigurationDesk.
- Exporting the wiring information for the external cable harness.
- Exporting the configuration data of the currently active ConfigurationDesk application for documentation purposes.