RTI Bypass Blockset

Reference

For RTI Bypass Blockset 3.16

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 Reference

Contents

This RTI Reference is a complete description of the Real-Time Interface (RTI) blocks and their settings provided by the RTI Bypass Blockset. You can use this blockset to configure bypass interfaces and implement new ECU functions for bypassing purposes. The blockset supports bypassing on a dSPACE prototyping system (external bypassing) and ECU-internal bypassing (on-target bypassing). The blockset also supports function bypassing in a virtual environment.

For the external bypass approach, the blockset is supported by the DS1006 Processor Board / DS1007 PPC Processor Board with DS4121, DS4302, DS2202, DS2210, DS2211, DS4501 or DS4505 board, and the MicroAutoBox II.

For the on-target bypass approach, the blockset supports various target processor types of the Infineon TriCore, the Freescale MPC5xxx, the Renesas RH850X and V850X, and the ARM Cortex-R4/R5 target processor families.

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.
⍰	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.

Symbol	Description
📖	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.

Examples:

- Where you find terms such as `rti<XXXX>` replace them by the RTI platform support you are using, for example, `rti1007`.
- Where you find terms such as `<model>` or `<submodel>` in this document, replace them by the actual name of your model or submodel. For example, if the name of your Simulink model is `smd_1007_sl.slx` and you are asked to edit the `<model>_usr.c` file, you actually have to edit the `smd_1007_sl_usr.c` file.

RTI block name conventions All I/O blocks have default names based on dSPACE's board naming conventions:

- Most RTI block names start with the board name.
- A short description of functionality is added.
- Most RTI block names also have a suffix.

Suffix	Meaning
B	Board number (for PHS-bus-based systems)
M	Module number (for MicroAutoBox II)
C	Channel number
G	Group number
CON	Converter number
BL	Block number
P	Port number
I	Interrupt number

A suffix is followed by the appropriate number. For example, DS2201IN_B2_C14 represents a digital input block located on a DS2201 board. The suffix indicates board number 2 and channel number 14 of the block. For more general block naming, the numbers are replaced by variables (for example, DS2201IN_Bx_Cy).

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.

Safety Precautions

Introduction	To avoid risk of injury and/or damage, read and ensure compliance with the safety precautions given.

General Warning When Using the Internal Bypass Plug-In for the RTI Bypass Blockset

Introduction	Note the following warning when using the Internal Bypass Plug-In for the RTI Bypass Blockset.

Danger potential

Using this product can be dangerous. You must observe the following safety instructions and the relevant instructions in the user documentation.

⚠ WARNING

Improper or negligent use can result in serious personal injury and/or property damage.

The Internal Bypass Plug-In for the RTI Bypass Blockset allows the integration of function code and associated data in ECU image and ECU variable description files. Programming ECUs with these image files and accessing ECUs via calibration tools with these ECU description files can have a direct effect on networked electronic systems and may lead to unforeseeable system behavior with an increased risk of property damage or personal injury.

Only persons who are qualified to use the Internal Bypass Plug-In for the RTI Bypass Blockset, who have been informed about the above dangers, and who are able to assess the possible consequences to take appropriate precautions, are permitted to use this product.

All applications where malfunctions or misoperation involve danger of property damage, injury or death must be examined for potential hazards by the user, who must if necessary take additional measures for protection, for example, by implementing an emergency off switch, and/or by clearly labeling files to prevent original ECU image and ECU description files being confused with those modified by the Internal Bypass Plug-In for the RTI Bypass Blockset.

Liability

It is your responsibility to adhere to instructions and warnings. Any unskilled operation or other improper use of this product in violation of the respective safety instructions, warnings, or other instructions contained in the user documentation constitutes contributory negligence, which may lead to a limitation of liability by dSPACE GmbH, its representatives, agents and regional dSPACE companies, to the point of total exclusion, as the case may be. Any exclusion or limitation of liability according to other applicable regulations, individual agreements, and applicable general terms and conditions remain unaffected.

General Information on the RTI Bypass Blockset

Introduction

Here you get basic information on the RTI Bypass Blockset.

Where to go from here

Information in this section

Overview of the RTI Bypass Blockset

Introduction

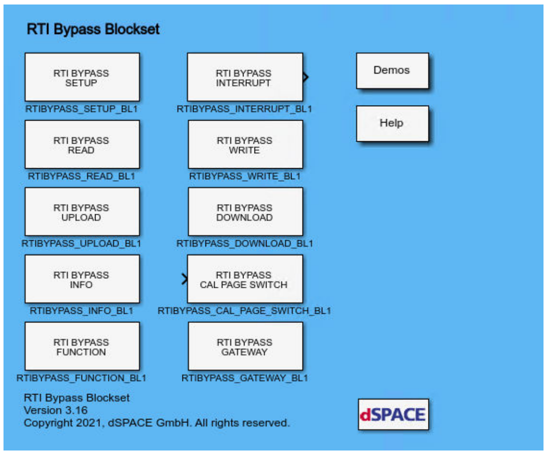
This topic gives you a short description of the RTI Bypass Blockset and its block library.

RTI Bypass Blockset

The RTI Bypass Blockset is a Simulink blockset for dialog-based configuration of bypass applications. It consists of several RTI blocks which let you configure bypass interfaces and implement new ECU functions for on-target (ECU-internal) or external bypassing purposes.

Library access

When **rtibypass** is entered in the MATLAB Command Window, the block library of the RTI Bypass Blockset is displayed.



Library components

The following components are available in the RTI Bypass Blockset library:

RTI Bypass Blockset Lets you configure the interface to the ECU (bypass interface) and functions for rapid prototyping. For detailed information on the RTI blocks contained in the blockset, refer to Components of the RTI Bypass Blockset on page 49.

Demos Opens a sublibrary with the available example models. You can also find the model files in the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS` folder.

You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.

Help Displays this reference information.

Compatibility to earlier blockset versions

RTI Bypass Blockset 3.15 is compatible with the earlier blockset versions 3.x and 2.x. However, there are some points to note:

Working with models from RTI Bypass Blockset 2.5 or earlier Data management was changed in comparison to the prior RTI Bypass Blockset versions. If you have a Simulink model built with RTI Bypass Blockset 2.5 or earlier and open it with RTI Bypass Blockset 2.6 or later, the old data dictionary file (with file name extension .dd) is replaced by a new data dictionary file (.vdb) using the information stored in the Setup block as soon as you open and close the Setup block dialog via OK, or open the Read, Write, Upload, or Download block dialog and click the Fill Variable Selector button on the Variables page.

If you have a model that was saved with RTI Bypass Blockset 2.6 or later and want to use it with RTI Bypass Blockset 2.5 or earlier, the model's data dictionary file required for blockset version 2.5 or earlier (file name extension .dd) is created as soon as you update the A2L files in the Setup block or open the Read, Write, Upload, or Download block and click the **Fill Variable Selector** button on the Variables page. The data dictionary file created under RTI Bypass Blockset 2.6 or later (*.vdb) remains on disk.

To make the RTI Bypass Blockset able to recreate the data dictionary, the database files specified in the Setup block must be accessible at the specified location and must be unchanged.

Working with models from RTI Bypass Blockset 2.6 up to and including RTI Bypass Blockset 3.14 If you have a Simulink model built with RTI Bypass Blockset 2.6 up to RTI Bypass Blockset 3.14 and open it with RTI Bypass Blockset 3.15, the old data dictionary file is replaced by a new data dictionary file. However, the new data dictionary file cannot be used in earlier RTI Bypass Blockset versions. If you want to reuse the model with RTI Bypass Blockset 2.6 up to RTI Bypass Blockset 3.14 again, you have to create a suitable database in the earlier RTI Bypass Blockset version by reimporting the database files (A2L files) specified in the Setup block.

Note

The current RTI Bypass Blockset is not compatible with RTI Bypass Blockset 1. Thus, Simulink models which are built with an RTI Bypass Blockset 1.x version cannot be used with the RTI Bypass Blockset 2.x and 3.x versions.

Related topics

Basics

Features of the RTI Bypass Blockset

Introduction

This topic lists the main features of the RTI Bypass Blockset.

Bypassing methods

The RTI Bypass Blockset supports several methods for function bypassing. You can perform external rapid prototyping by means of service-based and serviceless bypassing, and on-target rapid prototyping by means of internal bypassing. For further information, refer to Bypassing Methods Supported by the RTI Bypass Blockset on page 33.

You can mix on-target and external bypass parts in one Simulink model. During the build process, each on-target bypass part is extracted into a separate model, which then is built as a stand-alone application. Using the RTI BYPASS BUILD block, you can select the parts to be included in the build process.

Note

Currently, on-target bypassing in the context of real ECUs is supported for the following target processor families:

- Infineon TriCore
- Freescale MPC5xxx
- Renesas RH850X and V850X
- Cortex-ARM R4/R5

For an overview of the supported target processor types, refer to Target Processor Types Supported for On-Target Bypassing on page 24.

Modeling tools

Depending on the bypassing method used, the RTI Bypass Blockset supports different modeling tools:

- For external bypassing, the RTI Bypass Blockset always works with MATLAB®/Simulink® as the modeling tool.
- When performing on-target bypassing, the RTI Bypass Blockset supports MATLAB/Simulink and TargetLink as modeling tools.

Supported code generators

When performing on-target bypassing, the RTI Bypass Blockset supports Simulink® Coder™ and the TargetLink Code Generator for model code generation.

For the external bypassing methods, Simulink Coder is always used for code generation.

Dialog-based configuration of bypass applications

The RTI Bypass Blockset allows you to configure bypass applications via dialogs.

The configuration of the ECU services is performed simply by dragging the RTI Bypass blocks into the Simulink or TargetLink model and setting the corresponding parameters in the dialog pages of the blocks.

Import of A2L files as database

All the required information on the ECU and the bypass interface is contained in the ECU's database file (A2L file), which can be easily imported as a database. The A2L file holds the bypass interface-specific information in the corresponding IF_DATA element.

You can specify several database files (A2L files) for one bypass interface.

The A2L file entries specify the minimum and maximum values of the ECU parameters. They limit the data that is written to the ECU RAM. The limit check is done on the RCP system. The data type limits of the A2L file variables are also controlled when data is written to the ECU RAM.

Accessing V-ECU-internal functions and variables

The RTI Bypass Blockset supports the access to V-ECU-internal functions and variables. This lets you do tasks such as:

- Replace an existing (V-)ECU function with a new one, and test the new (V-)ECU function as early as possible
- Synchronize V-ECU application tests with the value of a V-ECU-internal status variable
- Connect V-ECU-internal variables to the I/O of the MicroAutoBox III or SCALEXIO system

Note

The V-ECU to be bypassed must be prepared for V-ECU access in SystemDesk using the Rapid Prototyping Access (RptAccess) module. For more information, refer to:

- Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on VEOS on page 248
- Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on a SCALEXIO System or on a MicroAutoBox III on page 250

Hierarchical view of ECU variables

The ECU variables specified in the ECU's ASAM MCD-2 MC (A2L) database file can be easily selected via a file browser, which is part of the RTI Bypass Blockset. The browser provides a hierarchical representation of variables. It also offers sorting methods and search options for variable selection to facilitate the handling of large A2L files.

Gateway functionality

For XCP on CAN, the RTI Bypass Blockset provides a gateway functionality. This allows arbitrated access from the dSPACE prototyping system and an external calibration tool, or from two external tools, to the same ECU at the same time, even if only one XCP service instance is implemented in the ECU code.

Note

The gateway functionality is supported only for XCP services providing static DAQ list handling.

Custom C code

You can add custom C code to the bypass model. The RTI Bypass Blockset provides the Function block for this.

Calling ECU-internal functions

You can use ECU-internal functions in the bypass model. When you select a Setup block configured for on-target bypassing in the Function block, all the associated A2L files with the global variable definitions and the available ECU-internal functions they contain are displayed for selection. ECU-internal functions are handled in the Function block in the same way as C functions and variables from external sources.

ECU-internal functions to be called by the Function block must be EABI-compliant.

Converting a source value into a converted value

The A2L file entries specify the conversion from converted (physical) to source values in the ECU memory and vice versa.

The RTI Bypass Blockset supports the following conversion types for computation methods:

- Fractional rational function (`RAT_FUNC`)
- Table with interpolation (`TAB_INTP`)
- Table without interpolation (`TAB_NOINTP`)
- Formula-based conversion (`FORM`)

If the A2L file specifies conversion formulas which are not supported or if a conversion method is explicitly disabled in the Read, Write, Upload or Download blocks, the identity y=u is used instead. However, there are some limitations in connection with computation methods. Refer to Limitations of the RTI Bypass Blockset on page 269.

Automatic mapping of variable names to addresses on the ECU

ECU variables can be accessed by their names. The blockset internally maps the variables and their names to the corresponding addresses in the ECU memory and also handles the formulas for computing converted values from source values and vice versa.

External variable definition

The RTI Bypass Blockset provides external variable configuration. If external variable definition is enabled, additional Simulink block ports are added to the Read, Write, Upload and Download blocks. The additional Simulink block ports allow you to configure variables during run time by external block inputs. You can selectively disable single external variables during run time.

Tip

You can find an example of configuring variables externally in your dSPACE installation. For the demo ECU application, refer to the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS` folder.
You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.

Related topics

Basics

Requirements for Using the RTI Bypass Blockset

Introduction

For bypassing with the RTI Bypass Blockset, the ECU code, the prototyping hardware and the ECU´s A2L file must meet some requirements.

Requirements for the ECU code

When using the RTI Bypass Blockset, there are a few requirements regarding the ECU code:

ECU code for external, service-based bypassing For service-based bypassing on an external RCP system with the RTI Bypass Blockset, the ECU code must be implemented with an XCP service, a CCP service, or the dSPACE Calibration and Bypassing Service.

The RTI Bypass Blockset requires the following versions of ECU services:

- XCP service (dSPACE XCP Service Ver. 2.0 or later, or custom XCP service)
- dSPACE Calibration and Bypassing Service Ver. 1.1 or later
- CCP protocol version 2.1 or later

  The RTI Bypass Blockset supports bypassing via CCP with and without ECU code changes. Inquire at dSPACE for further details.

Note

- The *dSPACE Calibration and Bypassing Service* is part of the installation of ECU Interface Software.
  After you install and decrypt ECU Interface Software, you will find the `dSPACECalibrationAndBypassingService_<version>.exe` file in the `%ProgramData%\dSPACE\<InstallationGUID>\dsECU\Services` folder. Run it to install the service and its documentation in a folder of your choice.
  You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.
- To get the *dSPACE XCP Service*, contact dSPACE Support.

Note

The dSPACE XCP Service does not support FlexRay as a transport layer.

ECU code for external, code-patch-based bypassing In connection with the DCI-GSI2, the RTI Bypass Blockset supports code-patch-based bypassing. That means the RTI Bypass Blockset can be used without having to integrate an ECU service in the ECU code. Only few ECU code modifications (customized code patches) must be implemented if you want to use the bit-based or byte-based subinterrupt mechanism or a single hardware interrupt.

ECU code for external, address-based, hardware-supported bypassing In connection with the DCI-GSI2, the RTI Bypass Blockset supports address-based, hardware-supported bypassing. This means the RTI Bypass Blockset can be used without having to integrate an ECU service in the ECU code and without any ECU code modification. To perform bypassing, different event sources can be used for data acquisition and data stimulation. For more information, refer to DCI-GSI2 Feature Reference.

ECU code for on-target, service-based bypassing For on-target service-based bypassing with the RTI Bypass Blockset, the ECU code must be implemented with the dSPACE Internal Bypassing Service.

Note

The *dSPACE Internal Bypassing Service* is part of the installation of ECU Interface Software.
After you install and decrypt ECU Interface Software, you will find the `dSPACEInternalBypassingService_<version>.exe` file in the `%ProgramData%\dSPACE\<InstallationGUID>\dsECU\Services` folder. Run it to install the service and its documentation in a folder of your choice.
You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.

External bypassing: Requirements for the prototyping hardware

The following table shows the dSPACE hardware that is supported by the RTI Bypass Blockset:

dSPACE Hardware	Service-Based Bypassing via ...					Serviceless Bypassing[1] via DCI-GSI2[2]
	DPMEM	CCP	XCP on CAN	XCP on UDP/IP (incl. DCI-GSI2[2])	XCP on FlexRay	
Modular system based on DS1006 or DS1007 with DS4121 board	✓	–	–	✓[3]	–	✓[3]
Modular system based on DS1006 or DS1007 with DS4302, DS2202, DS2210 or DS2211 board	–	✓	✓	–	–	–

dSPACE Hardware	Service-Based Bypassing via ...					Serviceless Bypassing[1] via DCI-GSI2[2]
	DPMEM	CCP	XCP on CAN	XCP on UDP/IP (incl. DCI-GSI2[2])	XCP on FlexRay	
Modular system based on DS1006 or DS1007 with DS4505 board equipped with DS4342 CAN FD Interface Modules	–	–	✓[4]	–	–	–
Modular system based on DS1006 or DS1007 with DS4501 board or DS4505 board equipped with FlexRay communication modules	–	–	–	–	✓	–
DS1007 PPC Processor Board (with Ethernet I/O interface)	–	–	–	✓	–	✓
MicroAutoBox II with ECU Type 1 module	✓	–	–	✓[5]	–	✓[5]
MicroAutoBox II with CAN Type 1 module	–	✓	✓	–	–	–
MicroAutoBox II equipped with DS4342 CAN FD Interface Modules	–	–	✓[4]	–	–	–
MicroAutoBox II with FlexRay communication modules	–	–	–	–	✓	–
MicroAutoBox II with ETH Type 1 module	–	–	–	✓	–	✓

1) Includes code-patch-based bypassing, address-based bypassing, and address-based, hardware-supported bypassing.
2) The DCI-GSI2 is accessed by using its generic XCP on Ethernet interface and the RTI Bypass Blockset's XCP on UDP/IP hardware layer. It is treated as a common XCP on UDP/IP device.
3) An LVDS_CAB14 LVDS-Ethernet Link Cable is required.
4) Implies CAN FD support.
5) To connect the MicroAutoBox II, an LVDS_CAB13 or LVDS_CAB14 LVDS-Ethernet Link Cable is required.

On-target bypassing: Requirements for accessing the ECU

No prototyping hardware is required. You can use any measurement and calibration system and the interfaces provided by your ECU to access the ECU for on-target bypassing.

To perform on-target bypassing with TargetLink, you must have installed both the RTI Bypass Blockset and TargetLink with valid licenses.

Requirements for the ECU's A2L file

To adapt the ECU for bypassing via RTI Bypass Blockset, the A2L file must meet some requirements.

A2L file The ASAM MCD-2 MC (A2L) file is the database. It contains a variable description of the ECU which specifies both the parameters and measurement variables of the related ECU and the interface. The A2L file is read to the RTI Bypass Blockset, that is, the addresses of the bypassing variables are read directly from the A2L file to the RTI blocks and the bypassing application is automatically configured.

The RTI Bypass Blockset supports the following versions of the ASAM MCD-2 MC standard:

- 1.4
- 1.5
- 1.6

Required interface-specific information A2L files must contain bypass interface-specific information in an `IF_DATA` element. This information must comply with a special format, which is described in ASAP2 Meta Language (AML) files. The AML specification of the interface must also be contained in the A2L file.

For information on the interface-specific `IF_DATA` elements and details on their data formats, refer to the descriptions of the bypass interfaces:

Bypass Interface Used	Refer to
▪ XCP on CAN ▪ XCP on UDP/IP[1] ▪ XCP on FlexRay	Interface Description Data for XCP (Interface Description Data Reference 📖)
CCP	Interface Description Data for CCP (Interface Description Data Reference 📖)
dSPACE on DPMEM	Interface Description Data for DPMEM PODs (Interface Description Data Reference 📖)
INTERNAL	Interface Description Data for Internal Bypassing (Interface Description Data Reference 📖)
VECU	Interface Description Data for V-ECU Access on MicroAutoBox II (Interface Description Data Reference 📖)

[1] The XCP on UDP/IP interface can also be used for external bypassing on ECUs equipped with a DCI-GSI2. For information on the required interface-specific information, refer to Overview of IF_DATA Entries and AML Files for DCI-GSI2s (Interface Description Data Reference 📖).

Related topics

Basics

Target Processor Types Supported for On-Target Bypassing

Introduction

The RTI Bypass Blockset supports various target processor types of different target processor families to perform on-target bypassing on real target ECUs or virtual ECUs.

Supported target processor types

The RTI Bypass Blockset installation comes with the `microcontroller.xml` file listing all the target processor types supported for on-target bypassing. You can find the file in the `<RCP_HIL_InstallationPath>\MATLAB\RTIBYPASS\M\Configuration` folder.

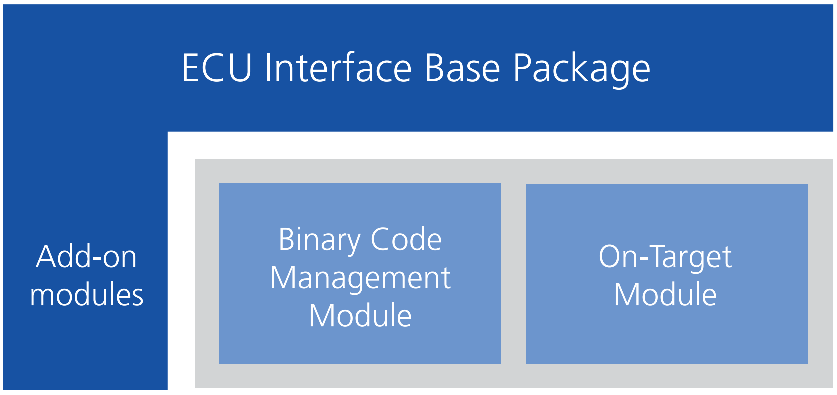
Related topics

Basics

ECU Interface Software: Packages and Modules

Introduction

The dSPACE ECU Interface Software consists of different packages and modules.



ECU Interface Base Package

A dSPACE product package consisting of the ECU Interface Manager, RTI Bypass Blockset, and dSPACE ECU services.

Features The package provides the following features:

- Performing external ECU interfacing if no (further) code modification is required
- Importing ECU applications
- Binary code analysis
- Configuring prepared bypass functions
- Exporting ECU interface container (EIC) files for ConfigurationDesk access

Binary Code Management Module (target-specific)

An optional module for preparing the binary code of a specific microcontroller family for ECU interfacing.

Features The module provides the following features:

- Binary code modification of ECU applications for internal and external ECU interfacing
- Integrating dSPACE ECU services into the binary code
- Integrating ECU service calls into the binary code
- Controlling and disabling the execution of code items such as write accesses

Supported microcontroller families The Binary Code Management Module is available for the following microcontroller families:

- ARM Cortex-R4/R5
- Freescale MPC5xxx
- Infineon TriCore
- Renesas V850x

On-Target Module (target-specific)

An optional module that enables the RTI Bypass Blockset to build a real-time application for a specific microcontroller family.

Features The module provides the following features:

- Blockset for implementing Simulink ECU interfacing functions on a specific target ECU that has been prepared for ECU interfacing
- Support of TargetLink and Simulink® Coder™

Supported microcontroller families The On-Target Module is available for the following microcontroller families:

- ARM Cortex-R4/R5
- Freescale/NXP MPC5xxx/STMicroelectronics SPC5xxx
- Infineon TriCore
- Renesas V850/RH850

Depending on the microcontroller family used, one of the following C compilers is required for building a real-time application:

On-Target Module	Target-Specific C Compiler
ARM Cortex-R4/R5	▪ GNU compiler[1)] ▪ Green Hills compiler ▪ Standard GNU compiler
Freescale MPC5xxx	▪ HighTec compiler ▪ Green Hills compiler ▪ Standard GNU compiler
Infineon TriCore	▪ HighTec compiler ▪ Green Hills compiler
Renesas V850x	▪ HighTec compiler ▪ Green Hills compiler ▪ Standard GNU compiler

1) Provided by the RCP and HIL installation.

Information on installation and licensing	For information on installing dSPACE software and handling dSPACE licenses, refer to What Do You Want To Do? (Installing dSPACE Software 🕮).

Migrating Models for DCI-GSI1-Based Interfaces

Migrating Models for DCI-GSI1-Based Interfaces

Introduction

To reuse an existing model that contains one or more DCI-GSI1-based bypass interfaces with the RTI Bypass Blockset 3.14 or later, you might have to carry out additional migration steps.

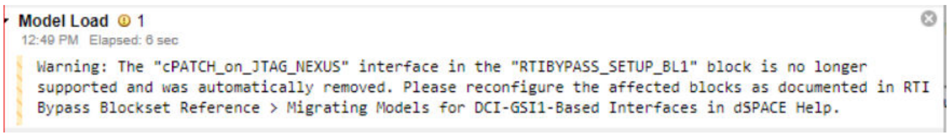
Discontinued DCI-GSI1-based bypass interfaces

Since RTI Bypass Blockset 3.14 (dSPACE Release 2020-A), the DCI-GSI1 has no longer been supported. As a consequence, the RTI Bypass Blockset no longer supports the following DCI-GSI1-based bypass interface types:

- DCI_GSI1
- dSPACE_on_JTAG_NEXUS
- dSPACE_on_JTAG_OCDS
- dSPACE_on_JTAG_SDI
- dSPACE_on_NBD_AUD
- dSPACE_on_NEXUS_READI
- cPATCH_on_JTAG_NEXUS
- cPATCH_on_JTAG_OCDS
- cPATCH_on_JTAG_SDI
- cPATCH_on_NBD_AUD
- cPATCH_on_NEXUS_READI

Models containing a DCI-GSI1-based bypass interface

When you open a model with one or more DCI-GSI1-based bypass interfaces specified in the imported database files in the RTI Bypass Blockset 3.14 or later, a message similar to the following is displayed, informing you that the support of the DCI-GSI1-based bypass interfaces is discontinued and therefore the obsolete interfaces were removed from the Setup block:


Model Load ⓘ 1
12:49 PM Elapsed: 6 sec

```
Warning: The "cPATCH_on_JTAG_NEXUS" interface in the "RTIBYPASS_SETUP_BL1" block is no longer
supported and was automatically removed. Please reconfigure the affected blocks as documented in RTI
Bypass Blockset Reference > Migrating Models for DCI-GSI1-Based Interfaces in dSPACE Help.
```

Whether and if so, the required subsequent migration steps depend on which bypass interface type was selected in your model and which bypass interface types are available in the Setup block. Refer to:

- Working with a model whose selected bypass interface type is still supported on page 30
- Working with a model whose selected bypass interface type is no longer supported on page 30

Working with a model whose selected bypass interface type is still supported

If you selected an interface in the Setup block of your model that is still supported, you can continue working with the model as usual. No migration steps are necessary.

However, all DCI-GSI1-based interface types that were previously in the list of available bypass interface types are removed from the list.

Working with a model whose selected bypass interface type is no longer supported

If an obsolete DCI-GSI1-based bypass interface was selected in the Setup block, you have to perform some migration steps.

Replacing the bypass interface type If an obsolete DCI-GSI1-based interface was selected in the Setup block, you must specify a different Bypass Interface Type on the Unit page of the Setup block.

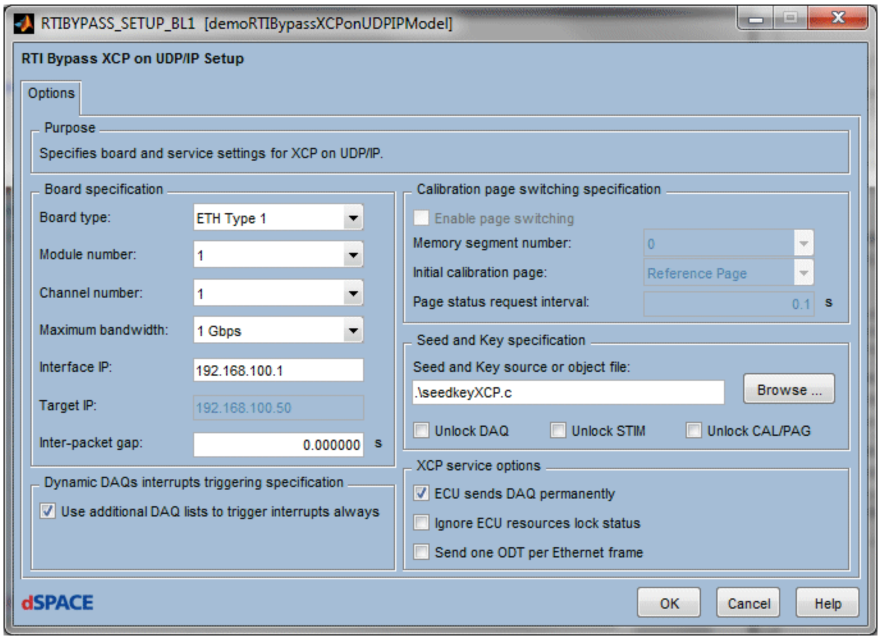
The list offers all the available bypass interfaces for selection. These are the bypass interfaces that are defined in the AML sections of the imported database files for the ECU and that are still supported by the RTI Bypass Blockset.

If no bypass interface type is displayed in the list, this means that the recently imported database files do not contain A2L sections for bypass interface types that are still supported. In this case, you have to import one or more database files with suitable interface definitions (`IF_DATA` entries) in the Setup block. Afterwards, you can select a new bypass interface type.

The DCI Configuration Tool lets you apply an A2L file that was originally configured for the DCI-GSI1 for use with the DCI-GSI2. Refer to Reusing DCI-GSI1 configuration settings with the DCI-GSI2 on page 31.

Adapting the RTI Bypass blocks After changing from a DCI-GSI1-based interface to the XCP on UDP/IP interface (which is used in the RTI Bypass Blockset to access a DCI-GSI2) in the Setup block, you must adapt the blocks of your RTI Bypass model to the new interface:

- On the Options page of the Setup block, specify the bypass interface settings.


RTIBYPASS_SETUP_BL1 [demoRTIBypassXCPonUDPIPModel]
RTI Bypass XCP on UDP/IP Setup
Options
dSPACE
OK
Cancel
Help
Browse ...

Purpose
Specifies board and service settings for XCP on UDP/IP.

Board specification

Board type:	ETH Type 1
Module number:	1
Channel number:	1
Maximum bandwidth:	1 Gbps
Interface IP:	192.168.100.1
Target IP:	192.168.100.50
Inter-packet gap:	0.000000 s

Dynamic DAQs interrupts triggering specification
☑ Use additional DAQ lists to trigger interrupts always

Calibration page switching specification

☐ Enable page switching	
Memory segment number:	0
Initial calibration page:	Reference Page
Page status request interval:	0.1 s

Seed and Key specification
Seed and Key source or object file: .\seedkeyXCP.c
☐ Unlock DAQ ☐ Unlock STIM ☐ Unlock CAL/PAG

XCP service options
☑ ECU sends DAQ permanently
☐ Ignore ECU resources lock status
☐ Send one ODT per Ethernet frame

Refer to Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) on page 167.

- Reconfigure the Read, Write, and Interrupt blocks of your model according to the DCI-GSI2 events.

> **Tip**
>
> All blocks that have to be reconfigured are displayed in the MATLAB Command Window.

The Info block gives you information concerning the model's RTI Bypass blocks, including what is not yet configured correctly.

Reusing DCI-GSI1 configuration settings with the DCI-GSI2

For easier migration from the DCI-GSI1 to the DCI-GSI2, the DCI Configuration Tool lets you import an A2L file originally adapted for use with the DCI-GSI1 and automatically apply some of its configuration settings to the device configuration of the DCI-GSI2. To do so, you have to perform the following steps in the DCI Configuration Tool:

1. On the Import A2L File page, select the A2L file originally adapted for use with the DCI-GSI1 and import it.

The imported file contains information about the services integrated into the ECU application that is required for bypassing and that is to be reused with the DCI-GSI2. The service settings are displayed in the DCI Configuration Tool, and the service event channel settings are imported to the device configuration of the DCI-GSI2.

Refer to Import A2L File Page (DCI Configuration 🕮).

2. On the **A2L File** page, specify an A2L file as the **Result A2L file**. This is the resulting A2L file, which is adapted for use with a DCI-GSI2. It is overwritten each time the A2L file is generated.
3. On the **A2L File** page, make other relevant settings:
   - Select the XCP protocol version for which to create the A2L structures. To let the DCI Configuration Tool automatically specify the suitable XCP protocol version, you can select 'Automatic'.
   - Select the A2L file structures to be modified and/or inserted for using a DCI-GSI2, and specify whether memory segments and calibration methods are to be imported into the device configuration.
4. On the **A2L File** page, start the A2L file generation by clicking **Generate A2L file**.

   Refer to A2L File Page (DCI Configuration 🕮).

The A2L file is adapted for use with the DCI-GSI2, i.e., the resulting A2L file contains a suitable IF_DATA XCP/XCPplus section for the DCI-GSI2. You can now import the resulting A2L file as a new database into the Setup block of your RTI Bypass model. As a result, XCP on UDP/IP is offered as an available bypass interface type in the Setup block.

Related topics

Basics

Configuring the Bypass Model (DCI-GSI2 Setup Application Note 🕮)

References

Unit Page (RTIBYPASS_SETUP_BLx).. 53

Bypassing Methods Supported by the RTI Bypass Blockset

Introduction

The RTI Bypass Blockset supports various bypassing methods.

Where to go from here

Information in this section

External, Service-Based Bypassing: Basic Information and Underlying Processes

Introduction

The RTI Bypass Blockset supports external, service-based bypassing.

Tip

To get information on which bypass interfaces support this bypassing method, refer to Requirements for Using the RTI Bypass Blockset on page 21.

External bypassing

A method for bypassing in which ECU functionalities are executed by an external rapid control prototyping (RCP) system, such as the MicroAutoBox II.

The RCP system is used in parallel to the ECU. The functions to be bypassed are offloaded from the ECU to the RCP system. The new algorithms are executed on the prototyping hardware, while the ECU still runs the existing code at the same time.

The ECU service makes the required ECU variables available to the bypass system and receives the calculated results from the bypass system. The results can be used for further processing by the ECU.

Service-based bypassing

A technique for external bypassing and on-target bypassing in which an ECU service (the part of the software on the ECU that allows access to the ECU´s variables) has to be integrated in the ECU code only once, and can be adapted individually to your requirements. You can easily change variables without having to recompile the code even after the service has been implemented.

With service-based bypassing as supported by the RTI Bypass Blockset, more than 100 functions in the ECU code can be prepared for bypassing by means of service calls (bypass hooks). Afterwards, there is no need to modify the ECU code again.

Service calls in the ECU code can be described in an ECU description file (A2L file) and evaluated by the RTI Bypass Blockset, so you can flexibly select the functions to be bypassed and the ECU variables to be read and written in the modeling environment.

The ECU code can be prepared for bypassing by modifying the ECU application source code (for example, your ECU supplier can do this), or by adapting the binary code of your ECU application in the dSPACE ECU Interface Manager. The ECU Interface Manager allows you to integrate service calls without touching the ECU application source code. For further information, refer to ECU Interface Manager Manual.

Service-based bypassing on an external RCP system

The requirements for and the principle procedure of service-based bypassing on an external RCP system with the RTI Bypass Blockset are described below.

Supported hardware layers The RTI Bypass Blockset supports the following hardware layers in connection with service-based bypassing on an external RCP system:

- XCP on CAN (implies CAN FD support)
- XCP on UDP/IP
- XCP on FlexRay
- CCP
- DPMEM (dual-port memory) PODs

For further information, refer to Requirements for Using the RTI Bypass Blockset on page 21.

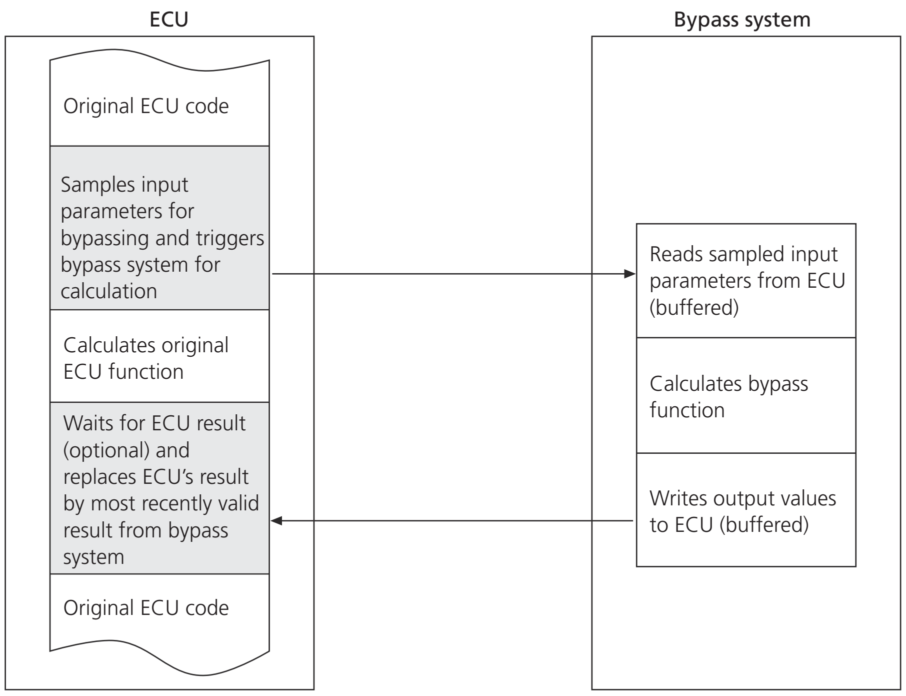
Requirements for ECU code To perform service-based bypassing on an external RCP system using the RTI Bypass Blockset, one of the following services must be implemented in the ECU code:

- XCP service (for bypassing via XCP on CAN (implies CAN FD support), XCP on UDP/IP, and XCP on FlexRay)
- CCP service (for bypassing via CCP)
- dSPACE Calibration and Bypassing Service (for bypassing via DPMEM)

For further information, refer to Requirements for Using the RTI Bypass Blockset on page 21.

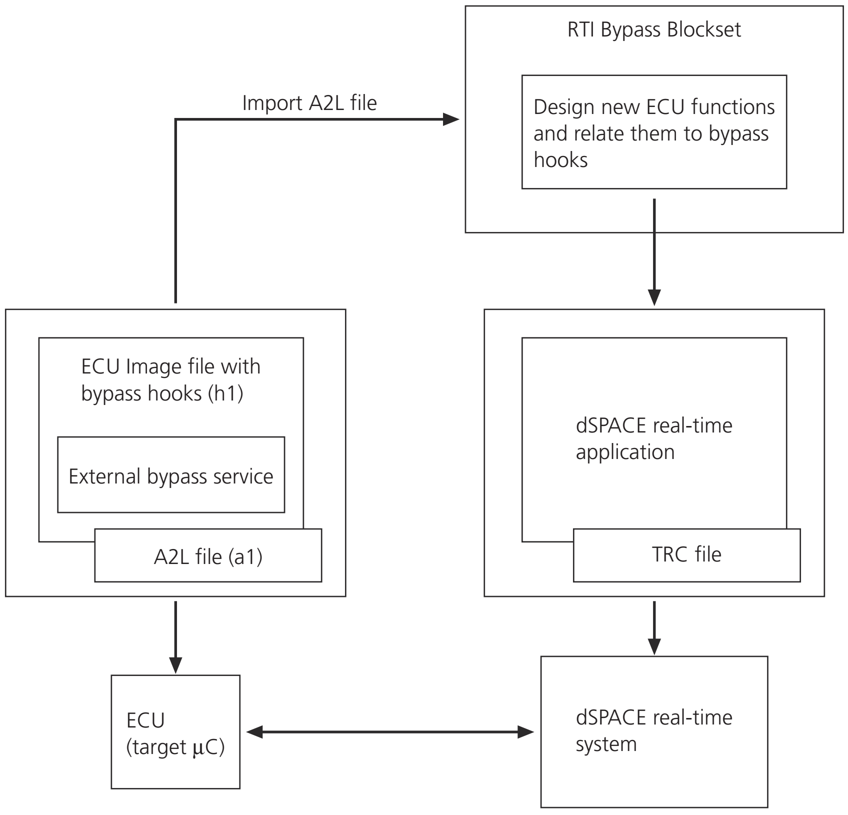
Data exchange between ECU and RCP system At the beginning of the ECU function or task to be bypassed, a service call is integrated in the ECU code to sample and supply the external bypass system with the required input values for calculation. An interrupt triggers the RCP system when the input data is available. The bypass system reads the input values from the ECU, calculates the function and returns the results to the ECU service. At the end of the ECU function or task to be bypassed, a service call is integrated in the ECU code to optionally wait for the bypass system results and write the calculated data back to the ECU. The ECU's result is replaced by the result from the bypass system.

The following illustration shows the typical sequence for external, service-based bypassing as described above:


ECU
Bypass system
Original ECU code
Samples input parameters for bypassing and triggers bypass system for calculation
Reads sampled input parameters from ECU (buffered)
Calculates original ECU function
Calculates bypass function
Waits for ECU result (optional) and replaces ECU's result by most recently valid result from bypass system
Writes output values to ECU (buffered)
Original ECU code

Overview of the external bypassing workflow

The illustration below is a schematic of the external bypassing workflow. The arrows indicate the processing sequence.


RTI Bypass Blockset
Design new ECU functions and relate them to bypass hooks
Import A2L file
ECU Image file with bypass hooks (h1)
External bypass service
A2L file (a1)
dSPACE real-time application
TRC file
ECU (target µC)
dSPACE real-time system

Performing external bypassing using the RTI Bypass Blockset includes the following steps:

- Integrating ECU service and creating an interface-specific A2L file

  To trigger external bypass functions, the ECU code must be implemented with an XCP service, a CCP service, or the dSPACE Calibration and Bypassing Service. An ECU Image file with integrated bypass hooks and a bypass interface-specific A2L file are generated. The A2L file contains a variable description of the ECU which specifies both the parameters and measurement variables of the related ECU and the interface.

  The ECU application binary is flashed into the ECU target controller. The A2L file is imported as a database to the RTI Bypass Blockset.
- Creating an external bypass model

  With the blocks from the RTI Bypass Blockset, you can create the external bypass model, and design the bypassing and target access functions.
- Building and downloading target-specific ECU code

  After you create the model, you start the build process. The bypass application is generated and downloaded to the external RCP system. The RCP system accesses the target system using the ECU interface according to the ECU service integrated in the ECU code.

Related topics

Basics

External, Code-Patch-Based Bypassing: Basic Information and Underlying Processes

Introduction

The RTI Bypass Blockset supports external, code-patch-based bypassing.

External bypassing

A method for bypassing in which ECU functionalities are executed by an external rapid control prototyping (RCP) system, such as the MicroAutoBox II.

The RCP system is used in parallel to the ECU. The functions to be bypassed are offloaded from the ECU to the RCP system. The new algorithms are executed on the prototyping hardware, while the ECU still runs the existing code at the same time.

The bypassing system directly accesses the variables by means of the ECU interface hardware.

Code-patch-based bypassing

A technique for external bypassing in which no ECU service needs to be integrated into the ECU code. Instead, customized code patches need to be integrated into the ECU code. If the input and output variables of functions change, the ECU code typically also needs to be modified.

The triggering of the bypass function on the RCP system can be implemented by means of subinterrupt handling, or by using a dedicated microcontroller pin as a single hardware interrupt.

The RTI Bypass Blockset supports a bit-based and a byte-based subinterrupt handling mechanism in connection with code-patch-based bypassing. To use a subinterrupt mechanism, code patches must be integrated into the ECU code. The ECU hardware event can either be configured by the DCI-GSI2 (e.g., with watchpoint mechanisms or a trace interface), or by the code patch integrated into the ECU code.

With code-patch-based bypassing, the ECU variables are accessed directly. Unlike service-based bypassing, code-patch-based bypassing does not provide mechanisms to ensure data consistency (for example, the double buffer mechanism). If necessary, you must implement them in the ECU application.

Code-patch-based bypassing is a derivative of address-based bypassing.

Code-patch-based bypassing on an external RCP system

The requirements for and the principle procedure of code-patch-based bypassing on an external RCP system with the RTI Bypass Blockset are described below.

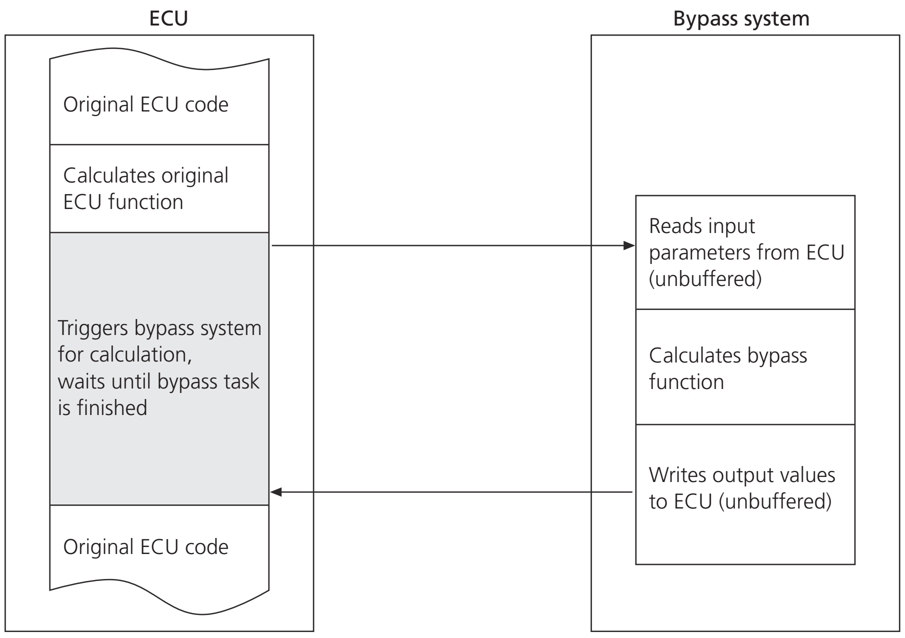
Supported hardware layers The RTI Bypass Blockset supports code-patch-based bypassing on an external RCP system for ECUs equipped with a DCI-GSI2.

For further information, refer to Requirements for Using the RTI Bypass Blockset on page 21.

Requirements for ECU code For code-patch-based bypassing, the ECU code can be equipped with bypass hooks. Only a few ECU code modifications need to be implemented if you want to use the bit-based or byte-based subinterrupt mechanism or a single hardware interrupt.

Data exchange between ECU and RCP system At the end of the ECU function or task to be bypassed, a code patch is integrated in the ECU code to trigger the RCP system and wait until the RCP system has finished the calculation of the bypass function. The bypass system reads the input values from the ECU, calculates the bypass function and writes the function results back to the ECU. Afterwards, the code patch continues the execution of the original ECU code.

The following illustration shows the typical sequence for external, code-patch-based bypassing as described above:


ECU
Bypass system
Original ECU code
Calculates original ECU function
Triggers bypass system for calculation, waits until bypass task is finished
Original ECU code
Reads input parameters from ECU (unbuffered)
Calculates bypass function
Writes output values to ECU (unbuffered)

Related topics

Basics

External, Address-Based, Hardware-Supported Bypassing: Basic Information and Underlying Processes

Introduction

The RTI Bypass Blockset supports external, address-based, hardware-supported bypassing.

External bypassing

A method for bypassing in which ECU functionalities are executed by an external rapid control prototyping (RCP) system, such as the MicroAutoBox II.

The RCP system is used in parallel to the ECU. The functions to be bypassed are offloaded from the ECU to the RCP system. The new algorithms are executed on the prototyping hardware, while the ECU still runs the existing code at the same time.

The ECU interface hardware makes the required ECU variables available to the bypass system and receives the calculated results from the bypass system. The results can be used for further processing by the ECU.

Address-based, hardware-supported bypassing

A technique for external bypassing that uses hardware resources, such as watchpoints or a data trace interface supported by the ECU/microcontroller and the DCI-GSI2. The triggering of the bypass function on the external RCP system can be implemented by using these hardware resources, i.e., by means of various event trigger mechanisms provided by ECU interface hardware such as the DCI-GSI2. This means, you can use several functionalities for data acquisition (DAQ) and data stimulation (STIM) purposes that use different event sources.

A data trace interface can also be used for reading and overwriting data consistently via the Fast Variable Overwrite mechanism.

Address-based, hardware-supported bypassing on an external RCP system

The requirements for and the principle procedure of address-based, hardware-supported bypassing on an external RCP system with the RTI Bypass Blockset are described below.

Supported hardware layers The RTI Bypass Blockset supports address-based, hardware-supported bypassing on an external RCP system for ECUs equipped with a DCI-GSI2.

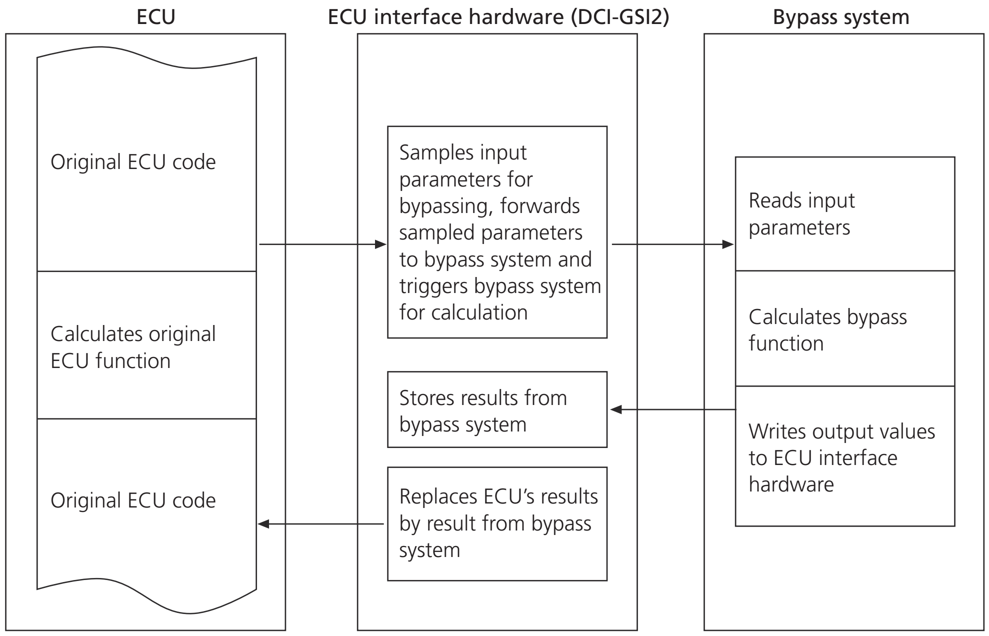
For further information, refer to Requirements for Using the RTI Bypass Blockset on page 21.

Requirements for ECU code The RTI Bypass Blockset can be used without any ECU code modification. To perform bypassing, different event sources can be used for data acquisition (DAQ) and data stimulation (STIM).

For further information, refer to DCI-GSI2 Feature Reference.

Data exchange between ECU and RCP system At the first specific code point, the ECU interface hardware samples and supplies the external bypass system with the required input values for calculation. An interrupt triggers the

RCP system when the input data is available. The bypass system receives the input values from the ECU interface hardware and calculates the function. The result is written back to the ECU interface hardware. At the second specific code point, the ECU interface hardware replaces the result of the original ECU function with the values provided by the RCP system.


ECU
ECU interface hardware (DCI-GSI2)
Bypass system
Original ECU code
Calculates original ECU function
Original ECU code
Samples input parameters for bypassing, forwards sampled parameters to bypass system and triggers bypass system for calculation
Stores results from bypass system
Replaces ECU's results by result from bypass system
Reads input parameters
Calculates bypass function
Writes output values to ECU interface hardware

Related topics

Basics

Address-Based, Hardware-Supported ECU Interfacing (ECU Interfacing Overview 🕮)
Bypassing Methods Supported by the RTI Bypass Blockset.. 33

On-Target, Service-Based Bypassing: Basic Information and Underlying Processes

Introduction

The RTI Bypass Blockset supports on-target, service-based bypassing.

Possible use scenarios

Unlike for external bypassing, the RTI Bypass Blockset supports different modeling tools (not only MATLAB/Simulink, but also TargetLink) and different code generators (not only Simulink® Coder™, but also the TargetLink Code Generator) for on-target, service-based bypassing. This results in the following possible use scenarios for implementing on-target bypassing:

- Working with a Simulink model and generating code with Simulink Coder
- Working with a Simulink model and generating code with the TargetLink Code Generator
- Working with a TargetLink model and generating code with the TargetLink Code Generator
- Working with a TargetLink model and generating code with Simulink Coder

Performing on-target bypassing with TargetLink requires valid licenses for both the RTI Bypass Blockset and TargetLink.

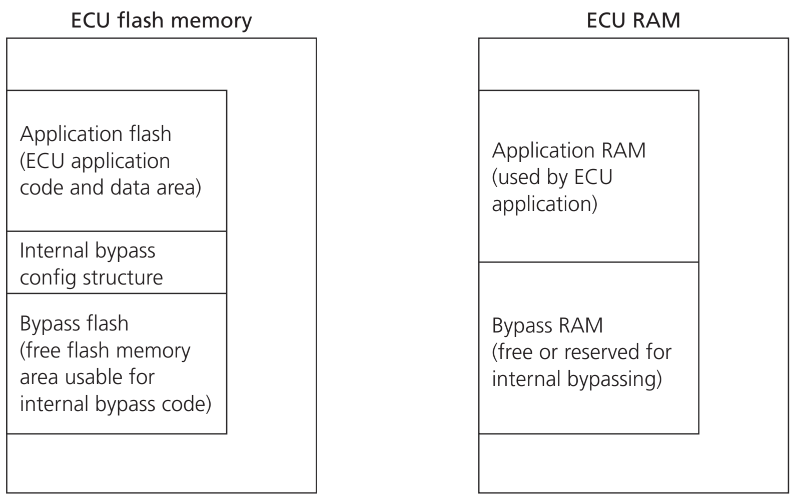
For more information on performing on-target bypassing with TargetLink, refer to Basics on Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing (TargetLink Interoperation and Exchange Guide).

On-target bypassing

A method for bypassing in which bypass functions are executed on the target ECU. No external rapid control prototyping (RCP) system is used.

The new ECU functionalities are executed by additional internal-bypass-specific ECU functions. If desired, the ECU still runs the original ECU code in addition to the on-target bypass functions. All ECU functions that remain unchanged are executed within the original ECU application.

On-target bypassing (also called internal bypassing) uses specific ECU-internal resources. The new ECU algorithms are executed in memory areas of the ECU RAM and the flash memory that are reserved for on-target bypassing.


ECU flash memory
Application flash (ECU application code and data area)
Internal bypass config structure
Bypass flash (free flash memory area usable for internal bypass code)
ECU RAM
Application RAM (used by ECU application)
Bypass RAM (free or reserved for internal bypassing)

Service-based bypassing

A technique for external bypassing and on-target bypassing in which an ECU service (the part of the software on the ECU that allows access to the ECU´s variables) has to be integrated in the ECU code only once, and can be adapted individually to your requirements. You can easily change variables without having to recompile the code even after the service has been implemented.

With the dSPACE Internal Bypassing Service as supported by the RTI Bypass Blockset, functions in the ECU code can be prepared for bypassing by means of service calls (bypass hooks).

Service calls in the ECU code can be described in an ECU description file (A2L file) and evaluated by the RTI Bypass Blockset, so you can flexibly select the functions to be bypassed and the ECU variables to be read and written in the modeling environment.

The ECU code can be prepared for bypassing by modifying the ECU application source code (for example, your ECU supplier can do this), or by adapting the binary code of your ECU application in the dSPACE ECU Interface Manager. The ECU Interface Manager allows you to integrate service calls without touching the ECU application source code. For further information, refer to ECU Interface Manager Manual.

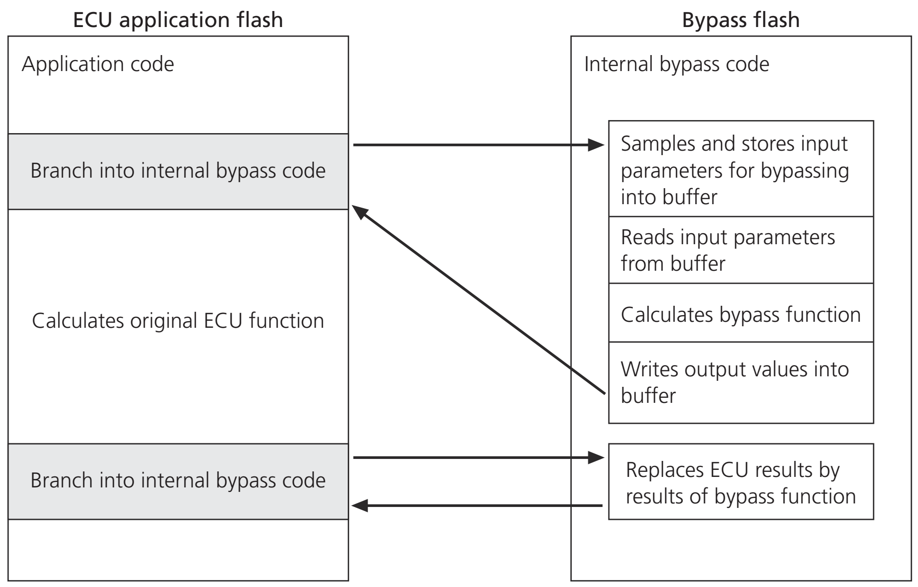
On-target, service-based bypassing with the RTI Bypass Blockset

The requirements for and the principle procedure of on-target, service-based bypassing with the RTI Bypass Blockset are described below.

Requirements for ECU code To perform on-target, service-based bypassing using the RTI Bypass Blockset, the dSPACE Internal Bypassing Service must be implemented in the ECU code.

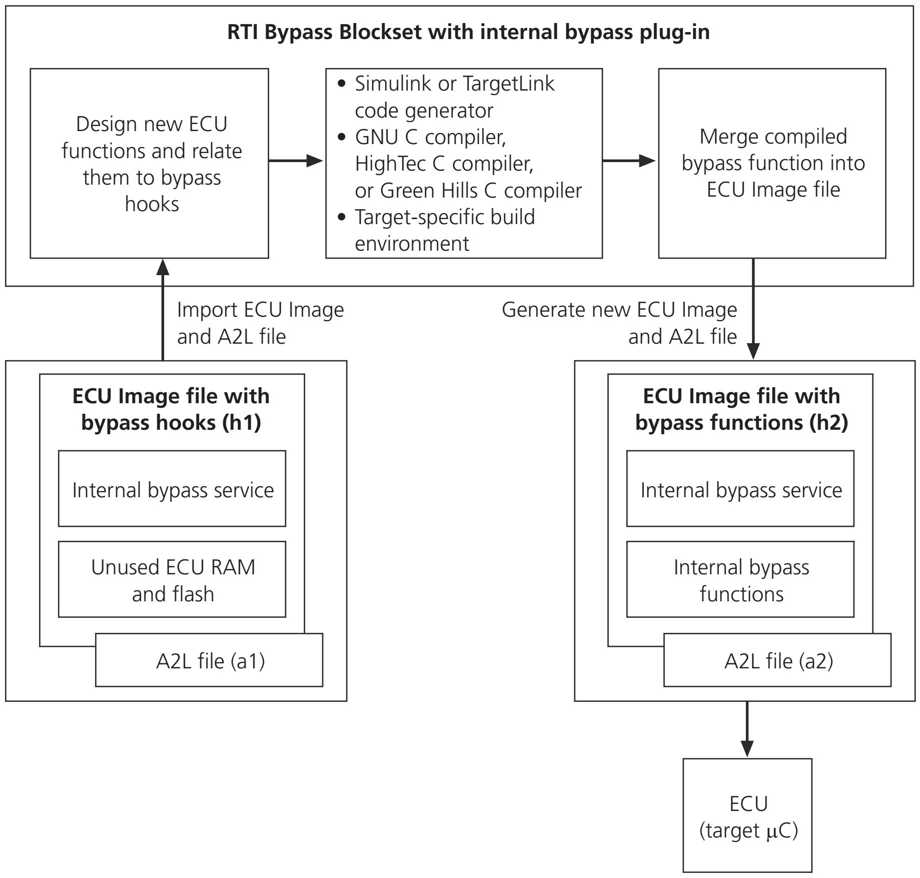
Data exchange between original ECU application and on-target bypass code At the beginning of the ECU function to be bypassed, a service call is integrated in the ECU code to trigger the bypass functionality. The relevant input values for calculation are read, and the bypass function is executed in the bypass flash. At the end of the ECU function to be bypassed, a service call is integrated in the ECU code to write the calculated values back to the original ECU application. This overwrites the result from the original ECU application with the result from the internal bypass function.

The following illustration shows the typical sequence for on-target, service-based bypassing as described above:


ECU application flash
Application code
Branch into internal bypass code
Calculates original ECU function
Branch into internal bypass code
Bypass flash
Internal bypass code
Samples and stores input parameters for bypassing into buffer
Reads input parameters from buffer
Calculates bypass function
Writes output values into buffer
Replaces ECU results by results of bypass function

Overview of the on-target bypassing workflow

The illustration below is a schematic of the on-target bypassing workflow. The arrows indicate the processing sequence.


RTI Bypass Blockset with internal bypass plug-in
Design new ECU functions and relate them to bypass hooks
• Simulink or TargetLink code generator
• GNU C compiler, HighTec C compiler, or Green Hills C compiler
• Target-specific build environment
Merge compiled bypass function into ECU Image file
Import ECU Image and A2L file
Generate new ECU Image and A2L file
ECU Image file with bypass hooks (h1)
Internal bypass service
Unused ECU RAM and flash
A2L file (a1)
ECU Image file with bypass functions (h2)
Internal bypass service
Internal bypass functions
A2L file (a2)
ECU (target µC)

On-target bypassing using the RTI Bypass Blockset includes the following steps:

- Integrating ECU service and creating an interface-specific A2L file

  To trigger on-target bypass functions, the dSPACE Internal Bypassing Service must be implemented in the ECU code. An ECU Image file with integrated bypass hooks and an internal bypass interface-specific A2L file are generated. The A2L file contains a variable description of the ECU which specifies both the parameters and measurement variables of the ECU and the internal bypass interface. It also provides information on the memory layout of the target ECU (free RAM and flash memory areas) and the fixed address of the internal bypass configuration structure. This information is required for compiling and linking the on-target bypass code into the memory areas that are not used by the original ECU application.

  The target application binary and the A2L file are imported to the RTI Bypass Blockset.

- Creating an on-target bypass model

  With the blocks from the RTI Bypass Blockset, you can create the on-target bypass model, and design the bypassing and target access functions. If you work with MATLAB®/Simulink® as the modeling tool, this is done in the same way as for Simulink models used to implement an external bypassing scenario. If you work with a TargetLink model, you can separate the TargetLink and the RTI Bypass parts of the model. This facilitates later reuse of the control algorithm part for production code generation with TargetLink. For more information, refer to Basics on Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing (TargetLink Interoperation and Exchange Guide 🕮).

You can implement several on-target bypass parts in one model, each part using its own Setup block.

- Building and flashing target-specific ECU code

  After you create the model, you start the build process to generate target-specific internal bypass ECU code.

> **Note**
>
> - When you work with the TargetLink Code Generator, the build process can be started via the RTI BYPASS BUILD block only.
> - If you generate code with Simulink Coder, you can also start the build process by clicking in the model or pressing `Ctrl+B`.
>   The command to start the build process is available on the C Code ribbon. If the C Code ribbon is hidden, go to the APPS ribbon and select Simulink Coder to display it.

Via the RTI BYPASS BUILD block, you can individually select the on-target bypass parts of your model that are to be included in the build process. If your model contains several Setup blocks with 'INTERNAL' bypass interface type, each interface individually generates specific code and extracts it into a separate subfolder. This means that each on-target bypass part is extracted into its own model and then built as a stand-alone application. During the build process, the Simulink parameters from the generated on-target bypass code are merged with the imported A2L file, and the on-target bypass ECU code is merged with the original ECU application according to the configuration settings you made in the related RTI Bypass Setup blocks. After the target-specific code is built, it can be flashed into the target system by means of the dSPACE ECU Flash Programming Tool (refer to the ECU Flash Programming 🕮 document) or any kind of custom flash tool chain.

Note

The following notes apply to working with a Simulink model and using Simulink® Coder™ as the code generator:

- System target files determine the structure of code generation in MATLAB/Simulink. Most of the basic settings in your modeling environment depend on the specified system target, so the system target file must fit the hardware you want to generate code for. The system target file is specified on the Code Generation page of the Configuration Parameters dialog. You can access the Configuration Parameters dialog via the Simulation ribbon. On the ribbon, click Prepare – Model settings.

  To build on-target bypass code, you must select rti_intbyp_<target type>.tlc as the system target file, where 'target type' stands for the microcontroller type the on-target bypass functions are to be generated for (for example, tricore). If your model contains only on-target bypass parts, the appropriate system target file is automatically selected in the Simulink® Coder™. If your model contains at least one external bypass part, the board-specific system target file rti<xxxx>.tlc is specified. In this case, only the target-specific make command is automatically set to the on-target bypass build command `make_rti_intbyp`, which forwards the extracted external model fragment to the original make command.
- If rti_intbyp_<target type>.tlc is selected as the system target file, the Simulink® Coder™ contains the RTI IntByp Build Options page. You can specify processor- and compiler-specific settings for the build process on the page. Refer to RTI IntByp Build Options Page on page 241.

Merging the on-target bypass code into your ECU application also merges its additional parameter descriptions into the ECU application's A2L file. You can access the on-target bypass parameters from within your calibration tool by using a dedicated calibration and/or bypassing interface provided by your ECU and ECU application (for example, by using ControlDesk and an XCP on CAN interface).

Simulink models with on-target and external bypassing interfaces If you implement on-target *and* external bypassing within one Simulink model, the RTI Bypass Blockset will split the respective elements into different models before building them. The resulting models are created the first time an internal/external interface is built, and then remain on disk.

The models can be created via a template. You can configure a template by means of the DSPACE_RTIBYPASS_Config.RTI_INTBYP_TEMPLATE parameter of the RTI Bypass configuration structure.

Since existing models are reused, you can also modify the models after generating them as well as build them separately.

Related topics

Basics

Bypassing Methods Supported by the RTI Bypass Blockset.. 33
Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on a SCALEXIO System or on a MicroAutoBox III.. 250
Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on VEOS.. 248
Implementing On-Target ECU Interfacing with RTI Bypass Blockset (ECU Interfacing Overview 🕮)
On-Target Bypassing: Bypassing Mechanisms.. 239
Service-Based ECU Interfacing (ECU Interfacing Overview 🕮)

Components of the RTI Bypass Blockset

Introduction

The RTI Bypass Blockset consists of several RTI blocks that are used to implement new ECU functions for bypassing purposes in Simulink models.

The blockset lets you configure the bypass interface, provides read access and write access to the variables of the ECU, lets you up- and download calibration memory contents to and from the ECU, and provides calibration page switching. It also provides a gateway functionality which allows arbitrated ECU access from the dSPACE prototyping system and up to two external calibration tools at the same time. You can also integrate custom C code into the ECU bypass application.

Where to go from here

Information in this section

RTIBYPASS_SETUP_BLx

Purpose

To set up the bypass interface which is the interface to the ECU.

Where to go from here

Information in this section

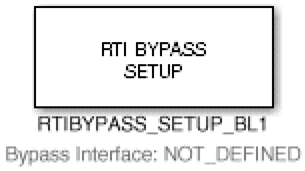
Information in other sections

Depending on the selected bypass interface, you have to specify interface-specific settings:

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_SETUP_BLx)

Block



Note

The RTIBYPASS_SETUP_BLx block has no name extension in the block library. When the block is copied to a Simulink model, the bypass interface name is added to the block name.

Purpose

To specify and configure the bypass interface.

Description

The RTIBYPASS_SETUP_BLx block handles the initialization and configuration of a single bypass interface.

Note

The RTIBYPASS_SETUP_BLx block is an essential block which must be placed within your Simulink model if you want to use the RTI Bypass Blockset. It is the only block containing hardware-related settings like board number and board type. Other blocks in the model depend (Read, Write, Interrupt, Upload, and Download blocks) or can depend (Gateway block) on the Setup block since they get all the required information about the bypass interface from the Setup block.

Dialog pages

The dialog settings can be specified on the following pages:

- Unit Page (RTIBYPASS_SETUP_BLx) on page 53
- Variables Options Page (RTIBYPASS_SETUP_BLx) on page 56
- DD Options Page (RTIBYPASS_SETUP_BLx) on page 56
- External Tools Page (RTIBYPASS_SETUP_BLx) on page 57

Note

A Setup block needs exclusive access. For this reason, all the RTIBYPASS block dialogs (except for the dialogs of the RTIBYPASS_INFO_BLx and RTIBYPASS_CAL_PAGE_SWITCH_BLx blocks) must be closed if you want to open a Setup block dialog, and cannot be opened until the Setup block dialog is closed again.

Related topics

Basics

Overview of the RTI Bypass Blockset.. 15

Unit Page (RTIBYPASS_SETUP_BLx)

Purpose

To specify a name for the bypass interface, define the database files for the ECU, and select one of the bypass interfaces specified in the imported database files.

Dialog settings

Bypass Interface Name Lets you enter a name for the bypass interface. The name must be a valid C identifier. Its length is restricted to 31 characters. The bypass interface name must be unique and not occur in any other Setup blocks in the model.

Use relative path for all Imported Database Files Lets you specify the database files with a path relative to the current model root folder. If the checkbox is cleared, the database files are specified with an absolute path.

Tip

Using a relative path makes it easy to move the model and all the files belonging to it to another location on your file system without having to change any path settings.

Tip

You can also specify a relative path offset globally for the entire model by setting the "DSPACE_RTIBYPASS_Config.A2L_ROOT_DIR" variable in the MATLAB workspace, for example:

```
DSPACE_RTIBYPASS_Config.A2L_ROOT_DIR = 'e:\temp\t1';
```

If you do so, the specified path is used as the relative path offset for all RTIBYPASS_SETUP_BLx blocks if the Use relative path for all Imported Database Files option is selected.

Imported Database Files Lists the imported database files for the ECU. Lets you select a database file from the list for modification. Actions like update or remove affect only the currently selected database file.

Add Database Lets you add one or more database files to the Imported Database Files list. No action is performed for database files that are already in the list. Database files are added temporarily, until you confirm the modification by pressing OK or Apply. Temporarily added database files are marked with a "+" in the Imported Database Files list.

When you access V-ECU-internal variables and functions from within a Simulink model on a MicroAutoBox II, specific database files generated by the RTI AUTOSAR Blockset are used. The A2L file information required by the RTI Bypass Blockset is automatically passed from the RTI AUTOSAR Blockset to the RTI Bypass Setup block, which means that the database files are selected automatically.

Update Database Lets you update the currently selected database file. Database files are updated temporarily, until you confirm the modification by pressing OK or Apply. Temporarily updated database files are marked with a "#" in the Imported Database Files list.

Remove Database Lets you remove the currently selected database file from the Imported Database Files list. Database files are deleted temporarily, until you confirm the modification by pressing OK or Apply. Temporarily removed database files are marked with a "-" in the Imported Database Files list.

Note

If a Setup block was changed by modifying the imported database files, the associated Simulink model must be saved before it is closed for consistency reasons.

Bypass Interface Type Lets you select the bypass interface type. The list displays all the available bypass interfaces. The available bypass interfaces are defined in the AML sections of the imported database files for the ECU, which are contained in the **Imported Database Files** list.

Tip

For a description of the interface-specific information (IF_DATA elements) which must be specified for the bypass interfaces in the A2L files, refer to the Interface Description Data Reference 🕮 .

Configure Opens the configuration dialog for specifying bypass interface settings. The dialog depends on the selected bypass interface type. If several instances of the selected bypass interface type are defined in the imported database files, a selection dialog first opens for you to specify the actual instance to be used, and then the configuration dialog opens.

Note

If the **VECU** bypass interface type is selected and you click **Configure**, no configuration dialogs are displayed. The configuration is done automatically.

For information on the bypass interface-specific configuration, refer to the corresponding topic:

- XCP on CAN: Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) on page 151
- XCP on UDP/IP: Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) on page 167
- XCP on FlexRay:
  - Options Page (RTIBYPASS_SETUP_BLx for XCP on FlexRay) on page 178
  - Buffers Configuration Page (RTIBYPASS_SETUP_BLx for XCP on FlexRay) on page 181
- CCP: Options Page (RTIBYPASS_SETUP_BLx for CCP) on page 195
- dSPACE on DPMEM: Options Page (RTIBYPASS_SETUP_BLx for dSPACE on DPMEM) on page 199
- INTERNAL:
  - Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) on page 204
  - Memory Page (RTIBYPASS_SETUP_BLx for INTERNAL) on page 216
  - Options Page (RTIBYPASS_SETUP_BLx for INTERNAL) on page 218

Related topics

References

Variables Options Page (RTIBYPASS_SETUP_BLx)

Purpose

To activate the display identifiers in the variable list, and to create unique signal labels for the outports automatically.

Dialog settings

Use display identifier Lets you enable display identifiers. If display identifiers are specified for variables in the A2L file, they are used instead of the variable names in the Variable Selector and in the Selected Variables list. They are also used as port names and signal labels in the Simulink model.

Create unique signal names from RTIBypass ports Lets you specify to convert the automatically assigned signal labels to model-wide unique identifiers. If this checkbox is selected and if the Select output ports signal labels option is activated on the Options page in a Read, Write, Upload or Download block, the names of the automatically assigned signal labels of the output ports are extended to be unique within your entire Simulink model.

Related topics

References

DD Options Page (RTIBYPASS_SETUP_BLx)

Purpose

To specify the deletion of obsolete data dictionaries files, or delete such data dictionary files immediately.

Dialog settings

Delete unused Data Dictionary files when model is saved Lets you specify to delete obsolete data dictionary files in the folder the current model is stored each time the model is saved. With this option enabled, the obsolete data dictionary files are deleted as soon as you save a modified model.

Delete now Lets you immediately delete obsolete data dictionary files in the folder the current model is saved in. The models located in the current working folder are opened sequentially to identify the data dictionaries that are no longer required.

Related topics

References

External Tools Page (RTIBYPASS_SETUP_BLx)

Purpose

To start up to five external tools by a single click each from within the RTI Bypass Blockset.

Note

The External Tools page is not displayed by default. You can activate it by setting the "DSPACE_RTIBYPASS_Config.ExternalTools" variable in the MATLAB workspace.

Description

You can specify up to five external tools to be displayed on the External Tools page. Each tool can then be started from here by a single click.

To make an external tool available on the External Tools page, you must set the following variables in the MATLAB workspace for it (where X is in the range 1 ... 5):

- DSPACE_RTIBYPASS_Config.ExternalTools(X).Name = '<tool_name>'

  (Example: `DSPACE_RTIBYPASS_Config.ExternalTools(1).Name = 'MyExternalTool'`)
- DSPACE_RTIBYPASS_Config.ExternalTools(X).Description = '<tool_description>'

  (Example: `DSPACE_RTIBYPASS_Config.ExternalTools(1).Description = 'Description of MyExternalTool'`)

- DSPACE_RTIBYPASS_Config.ExternalTools(X).Executable = '<file_name>'

  where 'file_name' stands for the executable file of the tool with its absolute path.

  (Example: `DSPACE_RTIBYPASS_Config.ExternalTools(1).Executable = 'C:\MyFolder\MyExternalTool.exe'`)

The `Name` and `Description` variables are optional. The `Executable` variable is mandatory to activate the button for starting tool execution.

Dialog settings

For each specified external tool, the following information and dialog settings are displayed:

Tool name Displays the name of the external tool you specified via the `Name` variable in the MATLAB workspace (see above). If no `Name` variable has been set in the MATLAB workspace, the default name "Tool X" is displayed.

Tool description Displays the description for the external tool you specified via the `Description` variable in the MATLAB workspace (see above). If no `Description` variable has been set in the MATLAB workspace, the tool description remains empty.

Run Lets you start the execution of the tool.

Related topics

References

RTIBYPASS_READ_BLx

Purpose

To read variables from the ECU.

Where to go from here

Information in this section

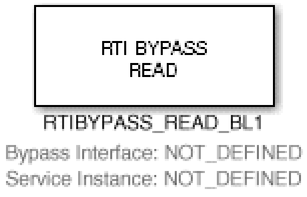
Information in other sections

Depending on the selected bypass interface, you have to specify interface-specific settings:

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_READ_BLx)

Block



Note

The RTIBYPASS_READ_BLx block has no name extension in the block library. When the block is copied to a Simulink model, the bypass interface name and the service instance name are added to the block name.

Purpose

To read variables from the ECU.

Description

The RTIBYPASS_READ_BLx block reads the ECU variables that are to be used as input variables for calculating the bypassed functions. Reading is event-triggered, that is, the variables are read synchronously with an ECU service call.

You must select the variables to be read from the ECU, specify the bypass interface and the service instance to use for reading the variables, and configure the system's behavior if no actual data is available (double buffer, wait, and failure checking mechanisms). The block also provides read status information or other service-specific information via the Status Port outport.

Note

The RTIBYPASS_READ_BLx block is supported by the following bypass interfaces:

- XCP on CAN
- XCP on UDP/IP
- XCP on FlexRay
- CCP
- DPMEM
- INTERNAL
- VECU

I/O characteristics

The table below shows the Simulink block input. The variable names are displayed at the inports only if the **Show port names** checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Input	Simulink Data Type	Description
Enable Service	Double	Lets you enable/disable the execution of the part of the service instance you specified on the Unit Page (RTIBYPASS_READ_BLx) and which is related to this Read block. The value indicates whether the service instance is enabled or not: ▪ > 0 : service instance enabled ▪ ≤ 0 : service instance disabled The port is available only if the **Use 'Enable Service' Port** checkbox is enabled on the Options Page (RTIBYPASS_READ_BLx). If the **Enable external variable ports** checkbox is selected on the Options Page (RTIBYPASS_READ_BLx), the **Enable Service** port is mandatory, since dynamic reconfiguration of externally defined variables during run time requires the Read block to be disabled during reconfiguration of the variable values, and then enabled again.
ExtVar Count	Double	Lets you define the number of externally defined variables actually used. The value must be in the range 0 ... **Maximum number of variables**. (The maximum number of variables is specified on the Options page.) For higher values, processing is reduced to the specified maximum number of variables. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Address	▪ Double (if block mode is enabled) ▪ Double array (if block mode is disabled)	Lets you define the address values of the externally defined variables. If the block mode is enabled on the Options page, the value indicates a single address specifying the start address for a whole data block. If the block mode is not enabled, an array of size 'Maximum number of variables' provides the address values for the single value accesses. (The maximum number of variables is specified on the Options page.) The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Type	▪ Double (if block mode is enabled)	Lets you define the data types for the externally defined variables. If the block mode is enabled on the Options page, a single value indicates the data type of the block members. If the block mode is not enabled, an array of size 'Maximum

Simulink Input	Simulink Data Type	Description
	▪ Double array (if block mode is disabled) The input values can have the following data types: ▪ 8: 8-bit unsigned ▪ 16: 16-bit unsigned ▪ 32: 32-bit unsigned ▪ -8: 8-bit signed ▪ -16: 16-bit signed ▪ -32: 32-bit signed ▪ -1032: float 32-bit ▪ 0: external variable disabled	number of variables' provides the data type values for the single values. (The maximum number of variables is specified on the Options page.) External variables can be disabled selectively during run time. The data type value of the related input element must be set to 0 for this. The block must be disabled during the change. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Conversion	▪ 1x6 double array (if block mode is enabled) ▪ nx6 double array, where n is the maximum number of variables (if block mode is disabled)	Lets you define the value conversion information for the externally defined variables. The conversion information is provided as a 1-dimensional array of 6 elements (if block mode is enabled) or of 'Maximum number of variables' x 6 elements (if block mode is disabled). The maximum number of variables is specified on the Options Page (RTIBYPASS_READ_BLx). Six coefficients a ... f are specified for each variable according to the fractional rational function $(a \cdot x^2 + b \cdot x + c) / (d \cdot x^2 + e \cdot x + f)$ used as the computation method for converting a source value into a converted value. **Note** ▪ Only linear conversion is possible for externally defined variables accessed by an RTIBYPASS_READ_BLx block. The conversion method therefore meets the linear function $(f \cdot x / b) - (c / b)$. ▪ The formula must always be defined in RCP to ECU direction, regardless of the block it is used with. ▪ Since the input is always a 1-dimensional array, it is recommended to use an S-function which integrates multidimensional data into a 1-dimensional array with this input port. For an implementation example, refer to the demo model in the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS` folder. You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows **Start** menu below **dSPACE RCP and HIL <version>**. The port is available only if the **Enable conversion port** checkbox is selected on the Options Page (RTIBYPASS_READ_BLx).

Note

- The ExtVar Count, ExtVar Address and ExtVar Type input port values are applied only if the block is executed for the first time after the dSPACE real-time application was started, or if the block was disabled and is then enabled again via the Enable Service input port.
  Since the ExtVar input ports are applied and the block containing external variables is enabled only if the block is executed, it must be guaranteed that the block is executed even if it is placed in a subsystem triggered by an RTI Bypass Interrupt block. The RTI Bypass Blockset usually implicitly guarantees that a subsystem is triggered independently of the enable state of the RTI Bypass Read and Write blocks. Under certain conditions, however, an additional Read block without external variables must be added to ensure an initial subsystem trigger and continuous triggering if all blocks with external variables are disabled. For further information, refer to Block Description (RTIBYPASS_INTERRUPT_BLx) on page 92.
- Depending on the configuration type (foreground or background configuration), the availability of read or written values is delayed for one or more subsystem execution steps. You can get information on the availability state of read or written values via the optional Status Port output port. For further information, refer to Perform variable configuration in foreground on the block's Option page.

The table below shows the Simulink block output. The variable names are displayed at the outports only if the Show port names checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Output	Simulink Data Type	Description
<name of variable>	Double	Reads the variable. The number of outports is equal to the number of selected variables on the Variables Page (RTIBYPASS_READ_BLx).
Status Port (or other service-specific outports)	Double	Provides run-time information on the selected service instance: ▪ 0 = new data available (data copied) ▪ -1 = old data available ▪ -2 = no data copied ▪ -4 = no data available ▪ -5 = invalid external variable configuration The port is available only if the Use 'Status Port' checkbox is enabled on the Options Page (RTIBYPASS_READ_BLx).
ExtVar Values	Double array	Provides the values of the externally defined variables. The read values are output as an array of size 'Maximum number of variables'. The maximum number of variables is specified on the Options Page (RTIBYPASS_READ_BLx). Only the values from 0 ... ExtVar Count are read and updated. The other values remain unchanged until they are affected by a modified ExtVar Count input value. If value conversion is enabled for the variables, the values are converted before they are read from the ECU. The port is available only if the Enable external variable ports checkbox is selected on the Options page.

Dialog pages

The dialog settings can be specified on the following pages:

- Unit Page (RTIBYPASS_READ_BLx) on page 64
- Variables Page (RTIBYPASS_READ_BLx) on page 65
- Options Page (RTIBYPASS_READ_BLx) on page 69

Note

A Read block needs exclusive access. For this reason, all the RTIBYPASS block dialogs (except for the dialogs of the RTIBYPASS_INFO_BLx and RTIBYPASS_CAL_PAGE_SWITCH_BLx blocks) must be closed if you want to open a Read block dialog, and cannot be opened until the Read block dialog is closed again.

Related topics

Basics

References

Unit Page (RTIBYPASS_READ_BLx)

Purpose

To specify the bypass interface and the service instance.

Dialog settings

Bypass interface Lets you select the bypass interface for reading the variables from the ECU. All bypass interfaces you configured via a Setup block in your Simulink model are listed.

Service instance Lets you select the service instance of the ECU where the block is to read the variables. An ECU service can be executed in different rasters (service instances) which are defined in the ECU´s database file. The names of the service instances are unique, and each service instance corresponds to one service ID. The **Service instance** drop-down list contains the available service instances, determined by the selected imported database files for the selected bypass interface. By default, no service instance is preset and the 'NOT_DEFINED' value is displayed.

Service ID Displays the service ID associated with the selected service instance. The service IDs are unique, and each ID corresponds to one service instance.

Related topics

References

Variables Page (RTIBYPASS_READ_BLx)

Purpose

To view the available ECU variables (parameters and measurement variables), and to select the variables that should be read for bypassing from the ECU.

Dialog settings

Fill Variable Selector Fills the Variable Selector with the variables defined in the database files you imported for the selected bypass interface.

Note

Depending on the size of the database file, filling the Variable Selector can take some seconds.

Variable Selector - Hierarchy tree In the hierarchy view mode, the structure of the imported variable description files is displayed in the hierarchy tree next to the variable list. The variable list shows the variables of the selected node and its subnodes. You can activate and deactivate the hierarchy view mode via the context menu of the variable list or hierarchy tree.

Variable Selector - Variable list The variable list of the Variable Selector displays the available ECU variables defined in the database files you imported for the selected bypass interface. Additional information on the ECU variables, for example, the description, memory address, and type, are also displayed.

If variables must be read for bypassing, you must add them from the variable list to the Selected Variables list. There are different ways of selecting variables:

- You can drag the variables from the variable list to the Selected Variables list.
- You can activate the checkboxes of the variables in the variable list using the mouse or via the **Space** key, and then press **Return** to add the checked variables to the Selected Variables list. When the variables are added to the Selected Variables list, their checkboxes are cleared. You can clear all checkboxes in the variable list without applying the variables to the Selected Variables list via the Uncheck All command, which is accessible via the context menu of a list entry.
- If the Select variable(s) with 'RETURN' checkbox is selected on the Options page, you can apply variables that are highlighted in the variable list to the Selected Variables list just by pressing the **Return** key without prior activation of their checkboxes.

Several methods of sorting, searching and filtering in the variable list are provided:

- *Sorting alphabetically*: You can sort the variable list alphabetically in ascending or descending order for a selected column. You can access the command by right-clicking the column header and choosing the function from the context menu.
- *Searching for a variable via buffered search*: You can search for a variable in the variable list using buffered search. When you type the first characters of one of its column entries, all keystrokes are stored in a buffer. The first item that matches the current keystroke sequence is selected. You can delete the whole keystroke sequence with **`Delete`**, but the selection remains. If the **Clear Search Buffer after variable(s) selection** option is selected on the Options page, the search buffer is cleared automatically after you select variables from the variable list. To differentiate between upper and lower case letters in your buffered search, open the context menu and select the **Case-Sensitive Search** command.
- *Filtering*: You can filter the list of variables by variable types. On the Options page, you can select whether only measurement variables are to be displayed in the variable list, or both measurement variables and parameters. The active filter setting is displayed in the header of the variable list.
- *Variable Properties*: You can view the properties of a variable. The information is displayed in a dialog which you can access via the context menu of the variable. The contents of the variable properties dialog depend on the type of the selected variable.

Selected Variables The **Selected Variables** list contains the variables you selected from the variable list for bypassing. The address, data type, conversion method from ECU representation (source value) to physical representation (converted value), variable description, and minimum and maximum values can be displayed for each variable in the **Selected Variables** list.

You can add custom variables to the **Selected Variables** list for bypassing via the **Add Custom** command. A custom variable has a *(custom)* declaration after its name.

You can add columns to the **Selected Variables** list or remove them from it via the **Customize Columns** command in the context menu of the list's column header. To add a column, drag the corresponding entry from the **Customize Columns** dialog to the column header. To remove a column, drag the corresponding column header entry from the **Selected Variables** list to the **Customize Columns** dialog. By default, the **Selected Variables** list contains the **Variable**, **Address**, **Type**, **Convert**, **Enable Remapping**, and **Port Data Type** columns.

- The variable can be converted from source value to converted value if the computation method is supported by the RTI Bypass Blockset.
- Variable conversion is never performed for a conversion formula that is not supported by the RTI Bypass Blockset. The source value is used in the Simulink model, and the checkbox beside the conversion method is disabled. For information on the supported computation methods, refer to Limitations of the RTI Bypass Blockset on page 269.

- You can deactivate the conversion of a variable by clearing its **Convert** checkbox in the **Selected Variables** list. In that case the input corresponds 1:1 to the value read from the ECU.
- If you perform on-target bypassing and the TargetLink Code Generator is selected as the code generator on the Build Page of the corresponding Setup block, you can specify the port data type for a variable with deactivated conversion via its **Port Data Type** field. You can select between Double and the fixed-point data type specified for the variable in the A2L file. If variable conversion is enabled for the variable later on, the RTI Bypass Blockset automatically uses Double as the port data type for the variable, regardless of which port data type setting you have made.

  If you use Simulink Coder as the code generator, the port data type of a variable is always set to Double.

The variables will be output from the Read block in the same order as they are displayed in the **Selected Variables** list. The variable names are used as port names. You can change the order of the variables and delete variables from the list.

Move Up Moves the currently selected variables further up the **Selected Variables** list. As a result, the corresponding outports of the Read block also move up.

Move Down Moves the currently selected variables further down the **Selected Variables** list. As a result, the corresponding outports of the Read block also move down.

Delete Deletes the currently selected variables from the **Selected Variables** list. The corresponding outports of the Read block are also deleted.

Delete All Deletes all variables from the **Selected Variables** list. The corresponding outports of the Read block are also deleted.

Add Custom Lets you add a custom variable. The Add/Edit Custom Variables Properties dialog opens for you to specify the variable settings.

View Variable Properties (Available only from the context menu of a variable in the Selected Variables list added to the Variable Selector) Lets you view the variable properties of the selected variable. The Variable Properties dialog opens and displays the properties.

Edit/View Custom Variables Properties (Available only from the context menu of a custom variable in the Selected Variables list) Lets you view and edit the properties of the selected custom variable. The Add/Edit Custom Variables Properties dialog opens for you to view and change the settings.

Add/Edit Custom Variables Properties dialog

Name Lets you specify the name of the custom variable. The name you enter must start with a letter. Otherwise, it will be prefixed with '_' automatically. Only the following characters are allowed: [a-z], [A-Z], [0-9], '.', '_', '[', and ']'.

Display identifier Displays the display identifier that will be used to name the output ports of the block, when the **Use Display Identifier** checkbox is selected on the Variables Options page in the Setup block dialog. It contains the capitalized variable name.

Address Lets you specify the address of the custom variable.

Data type Lets you select a data type for the custom variable. If you change the data type, the bitmask and the limit properties are reset to the defaults for the selected data type.

Byte order Displays the byte order specified in the interface description ASAP2 file and imported in the Setup block dialog. It can be either Motorola format (big endian, MSB at the memory location with the lowest address) or Intel format (little endian, LSB at the memory location with the lowest address).

Description Lets you specify a description text for the custom variable.

Start bit Lets you enter a bit offset for the bit mask, counted from the right side of the bit mask.

Number of bits Lets you specify the bit mask width by entering the number of contiguous 1-bits.

Hex (Bit mask) Displays the bit mask value in hexadecimal notation.

The following example shows the result of different bit mask property settings:

Data Type	No. of Bits	Start Bit	Bit Mask	Hex
8-bit	4	0	00001111	F
8-bit	4	2	00111100	3C
16-bit	5	5	0000001111100000	3E0

Min weak (phys.) Lets you specify the lower weak limit of the physical value. You can only enter values that are inside the hard limits.

Max weak (phys.) Lets you specify the upper weak limit of the physical value. You can only enter values that are inside the hard limits. If not specified a default value is automatically generated.

Hex (Limits) Displays the source values in hexadecimal notations.

Note

The limits are automatically adapted if you perform one of the following actions:

- You select another data type.
- You select another computation method.

Computation method Lets you specify a computation method. A computation method is used to transform a source value into a converted value.

Select one of the following computation methods and specify the conversion rule:

- `LINEAR`: You can specify a linear function (by means of a limited `RAT_FUNC` method).

- `RAT_FUNC`: You can specify a fractional rational function. Only linear conversion is possible for variables accessed by an RTIBYPASS_READ_BLx block or RTIBYPASS_UPLOAD_BLx block.
- `TAB_INTP`: You can specify a numeric conversion table for tabular conversion with interpolation.
- `TAB_NOINTP`: You can specify a numeric conversion table for tabular conversion without interpolation.
- `FORM`: You can specify a formula to calculate the physical value from the source value, and/or an inverse formula to calculate the source value from the converted value.

There are some limitations for using computation methods. Refer to Limitations of the RTI Bypass Blockset on page 269.

Related topics

References

Options Page (RTIBYPASS_READ_BLx)

Purpose

To set up block ports and specify bypass interface-specific settings for the RTIBYPASS_READ_BLx block.

Dialog settings

Use 'Enable Service' Port Lets you enable the use of the Enable Service inport. Via the Enable Service port, you can enable or disable the execution of the part of the service instance related to the corresponding RTIBYPASS block. If the checkbox is selected, the Read block gets an additional Enable Service inport. If the value fed to this Enable Service inport is equal to or less than 0, the variables are no longer sampled and, in the case of external bypassing, not transferred to the RCP system. The outputs of the Read block keep their last values. Enabling and disabling the Enable Service port for one Read block does not influence other Read, Write or Interrupt blocks using the same service instance.

If the **Enable external variable ports** checkbox is selected, the Enable Service inport is mandatory. The **Use 'Enable Service' Port** checkbox is selected and grayed out in this case.

Use 'Status Port' Lets you enable the use of the Status Port outport. You can read status information via this port. The following Status Port outport values of the Read block are available:

Status Port Outport Value	Description
0	New data available
-1	Old data available
-2	No data copied
-4	No data available
-5	Invalid external variable configuration

Show port names Lets you select whether the port names are displayed at the inports and outports. The ports themselves are always displayed.

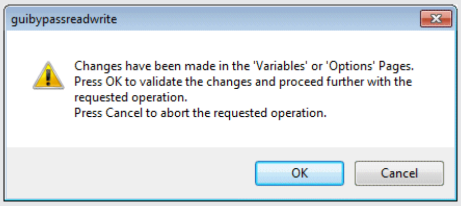
Tip

Deactivating this option reduces the size of the block.

Note

You cannot make changes to the port name settings and to the settings affecting the number of block ports at the same time. Instead, you have to apply the changes for one kind of setting first and then modify the other settings.

For example, if you add or delete variables to or from the **Selected Variables** list, or modify the **Use 'Status Port'** or **Use 'Enable Service' Port** options, the number of block ports is changed. Because of this, the **Show port names** option is grayed out until you apply your modifications for the port number by clicking **OK** or **Apply**. If you click on the grayed checkbox, a dialog similar to the following opens.


guibypassreadwrite

Changes have been made in the 'Variables' or 'Options' Pages.
Press OK to validate the changes and proceed further with the requested operation.
Press Cancel to abort the requested operation.

After you confirm the changes by clicking **OK** in the dialog, the settings are saved in the block parameters, and the **Show port names** option is no longer grayed out.

Set output ports signal labels Lets you specify to display the port names as signal labels for the outports.

If the **Create unique signal names from RTIBypass ports** option is activated on the Variables Options page in the Setup block (see Variables Options Page (RTIBYPASS_SETUP_BLx) on page 56), the automatically assigned signal names are extended to model-wide unique identifiers.

Filter 'Measurements' only Lets you select the variable types you want to filter the variable list on the Variables page by. If the checkbox is selected, only measurement variables are displayed in the variable list. If the checkbox is cleared, both measurement variables and parameters are displayed.

Fill 'Variable Selector' automatically Lets you select whether the variable selector on the Variables page is filled automatically each time you open the page.

Select variable(s) with 'RETURN' Lets you specify whether variables can be added to the Selected Variables list on the Variables page via the **`Return`** key. If the checkbox is activated, variables that are highlighted in the variable list can be added to the Selected Variables list by pressing **`Return`**, without having to activate their checkboxes first.

Clear Search Buffer after variable(s) selection Lets you select whether the search buffer is cleared automatically after you added variables from the variable list to the Selected Variables list on the Variables page via the **`Return`** key.

Enable Convert as Default Lets you specify the default setting for the Convert checkbox for variables that are added to the Selected Variables list on the Variables page. If enabled, the Convert checkbox is selected for a variable by default after the variable is added to the Selected Variables list, which means that variable conversion is activated for it. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

Enable Remapping as Default Lets you specify the default setting for the Enable Remapping checkbox for variables that are added to the Selected Variables list on the Variables page. If enabled, the Enable Remapping checkbox is selected for a variable by default after the variable is added to the Selected Variables list, which means that remapping to ECU internal variables is activated for it. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

Use Port Data Type 'Double' as Default (Available only if TargetLink Code Generator is selected as the code generator) Lets you specify the default setting for the Port Data Type setting for variables that are added to the Selected Variables list on the Variables page. If the checkbox is selected, Double is used as the default port data type for a newly selected variable with disabled conversion. If the checkbox is cleared, the fixed-point data type specified for a variable in the A2L file is selected as the port data type by default. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

If you work with Simulink Coder as the code generator, the port data type for the selected variables is always set to Double regardless of whether variable conversion is enabled or disabled for the variables.

Enable external variable ports Lets you specify whether variables to be used with this block can be defined during run time from outside the RTI Bypass Blockset. If the checkbox is selected, further Simulink block ports that allow you to configure variables by external block inputs are added to the Read block. For information on the additional ports, refer to Block Description (RTIBYPASS_READ_BLx) on page 60.

Note

- The ExtVar Count, ExtVar Address and ExtVar Type input port values are applied only if the block is executed for the first time after the dSPACE real-time application was started, or if the block was disabled and is then enabled again via the Enable Service input port.
  Since the ExtVar input ports are applied and the block containing external variables is enabled only if the block is executed, it must be guaranteed that the block is executed even if it is placed in a subsystem triggered by an RTI Bypass Interrupt block. The RTI Bypass Blockset usually implicitly guarantees that a subsystem is triggered independently of the enable state of the RTI Bypass Read and Write blocks. Under certain conditions, however, an additional Read block without external variables must be added to ensure an initial subsystem trigger and continuous triggering if all blocks with external variables are disabled. For further information, refer to Block Description (RTIBYPASS_INTERRUPT_BLx) on page 92.
- Depending on the configuration type (foreground or background configuration), the availability of read or written values is delayed for one or more subsystem execution steps. You can get information on the availability state of read or written values via the optional Status Port output port. For further information, refer to Perform variable configuration in foreground on the block's Option page.

Note

To ensure that Read blocks with external variable content acquire the external variable configuration, the related subsystem must be executed.

Maximum number of variables (Available only if Enable external variable ports is selected.) Lets you specify the maximum number of externally defined variables that can be used with this block.

The number of actually used variables is read by the ExtVar Count block inport. ExtVar Count can be in the range 0 ... 'Maximum number of variables'. If the specified number exceeds the upper limit, the value is reduced to 'Maximum number of variables'.

Enable block mode (Available only if Enable external variable ports is selected.) Lets you specify whether to use the block mode or single mode with this block. If the checkbox is selected, the block mode is enabled. With the block mode enabled, the ExtVar input ports allow several variables of the same data type to be read by just defining one start address for the whole data block and the number of values to be read. With the block mode disabled, the address, data type and value conversion must be defined separately for each variable.

Enable conversion port (Available only if Enable external variable ports is selected.) Lets you enable or disable the ExtVar Conversion block inport. If the checkbox is selected, the ExtVar Conversion inport is available and can be used to provide the computation methods for converting values of the variables that are read from the ECU with this block. If the checkbox is cleared, the ExtVar

Conversion inport is not available and data conversion is disabled for the variables read by this block.

Perform variable configuration in foreground (Available only if Enable external variable ports is selected.) Lets you enable or disable foreground reconfiguration of externally defined variables. If the checkbox is selected, the variables are reconfigured in the model foreground while the Read block is executed. The changed variable values are available in the subsequent sampling step, that is, directly after the associated ECU event (service instance) occurred. If the checkbox is cleared, variable reconfiguration is performed in the model background after the associated task is executed. The point in time the changed variable values are available is undefined, but not before the subsequent sampling step.

Via the Status Port outport of the Read block, you can get information on whether no or new data is available, or whether the external variable definitions are invalid.

Configuration in the foreground can increase the task execution time, depending on the bypass interface used. Configuration in the model background does not affect the task execution time.

Perform enable in foreground (Available only for external bypassing) Lets you specify whether enabling the block is processed in the foreground. If the checkbox is selected, the block is enabled in the model foreground. If the checkbox is cleared, enabling the block is performed in the model background.

Perform disable in foreground (Available only for external bypassing) Lets you specify whether disabling the block is processed in the foreground. If the checkbox is selected, the block is disabled in the model foreground. If the checkbox is cleared, disabling the block is performed in the model background.

Configure Opens a configuration dialog to specify bypass interface-specific settings for the RTIBYPASS_READ_BLx block. The dialog depends on the selected bypass interface type.

For information on the bypass interface-specific configuration, refer to the relevant topic:

- XCP on CAN: Options Page (RTIBYPASS_READ_BLx for XCP on CAN) on page 156
- XCP on UDP/IP: Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) on page 171
- XCP on FlexRay: Options Page (RTIBYPASS_READ_BLx for XCP on FlexRay) on page 184
- CCP: Options Page (RTIBYPASS_READ_BLx for CCP) on page 197
- dSPACE on DPMEM: Options Page (RTIBYPASS_READ_BLx for dSPACE on DPMEM) on page 200
- INTERNAL: Options Page (RTIBYPASS_READ_BLx for INTERNAL) on page 221

Related topics

References

Block Description (RTIBYPASS_READ_BLx).. 60

RTIBYPASS_WRITE_BLx

Purpose

To write variables back to the ECU.

Where to go from here

Information in this section

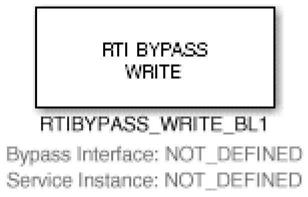
Information in other sections

Depending on the selected bypass interface, you have to specify interface-specific settings:

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_WRITE_BLx)

Block



Note

The RTIBYPASS_WRITE_BLx block has no name extension in the block library. When the block is copied to a Simulink model, the bypass interface name and the service instance name are added to the block name.

Purpose

To write variables back to the ECU.

Description

The RTIBYPASS_WRITE_BLx block writes the calculation results of the bypassing algorithms back to the ECU. Writing data is event-triggered, that is, the variables are written synchronously with an ECU service call.

In this block, you can specify the variables which are to be overwritten by the results calculated on the dSPACE prototyping system or the on-target bypass code, specify the bypass interface and the service instance to use for writing the variables, and configure the system's behavior if no actual data is available (double buffer, wait, and failure checking mechanisms). The block also provides write status information or other service-specific information on the Status Port outport.

Note

The RTIBYPASS_WRITE_BLx block is supported by the following bypass interfaces:
- XCP on CAN
- XCP on UDP/IP
- XCP on FlexRay
- DPMEM
- INTERNAL
- VECU

I/O characteristics

The table below shows the Simulink block input. The variable names are displayed at the inports only if the **Show port names** checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Input	Simulink Data Type	Description
<name of variable>	Double	Writes the variable. The number of inports is equal to the number of variables you selected on the Variables Page (RTIBYPASS_WRITE_BLx).
Enable Service	Double	Lets you enable/disable the execution of the part of the service instance you specified on the Unit page and which is related to this Write block. The value indicates whether the service instance is enabled or not: ▪ > 0 : service instance enabled ▪ ≤ 0 : service instance disabled The port is available only if the **Use 'Enable Service' Port** checkbox is enabled on the Options Page (RTIBYPASS_WRITE_BLx). If the **Enable external variable ports** checkbox is selected on the Options Page (RTIBYPASS_WRITE_BLx), the **Enable Service** port is mandatory, since dynamic reconfiguration of externally defined variables during run time requires the Write block to be disabled during reconfiguration of the variable values, and then enabled again.
ExtVar Count	Double	Lets you define the number of externally defined variables actually used. The value must be in the range 0 ... **Maximum number of variables**. (The maximum number of variables is specified on the Options page.) For higher values, processing is reduced to the specified maximum number of variables. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Address	▪ Double (if block mode is enabled) ▪ Double array (if block mode is disabled)	Lets you define the address values of the externally defined variables. If the block mode is enabled on the Options page, the value indicates a single address specifying the start address for a whole data block. If the block mode is not enabled, an array of size 'Maximum number of variables' provides the address values for the single value accesses. (The maximum number of variables is specified on the Options page.) The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Type	▪ Double (if block mode is enabled) ▪ Double array (if block mode is disabled)	Lets you define the data types for the externally defined variables. If the block mode is enabled on the Options page, a single value indicates the data type of the block members. If the block mode is not enabled, an array of size 'Maximum

Simulink Input	Simulink Data Type	Description
	The input values can have the following data types: ▪ 8: 8-bit unsigned ▪ 16: 16-bit unsigned ▪ 32: 32-bit unsigned ▪ -8: 8-bit signed ▪ -16: 16-bit signed ▪ -32: 32-bit signed ▪ -1032: float 32-bit ▪ 0: external variable disabled	number of variables' provides the data type values for the single values. (The maximum number of variables is specified on the Options page.) External variables can be disabled selectively during run time. The data type value of the related input element must be set to 0 for this. The block must be disabled during the change. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Conversion	▪ 1x6 double array (if block mode is enabled) ▪ nx6 double array, where n is the maximum number of variables (if block mode is disabled)	Lets you define the value conversion information for the externally defined variables. The conversion information is provided as a 1-dimensional array of 6 elements (if block mode is enabled) or of 'Maximum number of variables' x 6 elements (if block mode is disabled). The maximum number of variables is specified on the Options Page (RTIBYPASS_WRITE_BLx). Six coefficients a ... f are specified for each variable according to the fractional rational function $(a \cdot x^2 + b \cdot x + c) / (d \cdot x^2 + e \cdot x + f)$ used as the computation method for converting a source value into a converted value. **Note** ▪ The formula must always be defined in RCP to ECU direction, regardless of the block it is used with. ▪ Since the input is always a 1-dimensional array, it is recommended to use an S-function which integrates multidimensional data into a 1-dimensional array with this input port. For an implementation example, refer to the demo model in the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS` folder. You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows **Start** menu below **dSPACE RCP and HIL <version>**. The port is available only if the **Enable conversion port** checkbox is selected on the Options Page (RTIBYPASS_WRITE_BLx).
ExtVar Values	Double array	Defines the values of the externally defined variables. The values to be written to the ECU are input as an array of size 'Maximum number of variables'. The maximum number of variables is specified on the Options Page (RTIBYPASS_WRITE_BLx). Only the values from 0 ... **ExtVar Count** are written. The other values remain unchanged until they are affected by a modified **ExtVar Count** input value. If value conversion is enabled for the variables, the values are converted before they are written to the ECU. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.

Note

- The **ExtVar Count**, **ExtVar Address** and **ExtVar Type** input port values are applied only if the block is executed for the first time after the dSPACE real-time application was started, or if the block was disabled and is then enabled again via the **Enable Service** input port.
  Since the ExtVar input ports are applied and the block containing external variables is enabled only if the block is executed, it must be guaranteed that the block is executed even if it is placed in a subsystem triggered by an RTI Bypass Interrupt block. The RTI Bypass Blockset usually implicitly guarantees that a subsystem is triggered independently of the enable state of the RTI Bypass Read and Write blocks. Under certain conditions, however, an additional Read block without external variables must be added to ensure an initial subsystem trigger and continuous triggering if all blocks with external variables are disabled. For further information, refer to Block Description (RTIBYPASS_INTERRUPT_BLx) on page 92.
- Depending on the configuration type (foreground or background configuration), the availability of read or written values is delayed for one or more subsystem execution steps. You can get information on the availability state of read or written values via the optional **Status Port** output port. For further information, refer to **Perform variable configuration in foreground** on the block's Option page.

The table below shows the Simulink block output. The variable names are displayed at the outports only if the **Show port names** checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Output	Simulink Data Type	Description
Status Port (or other service-specific outports)	Double	Provides run-time information on the selected service instance: ▪ 0 = new data available (data copied) ▪ -1 = old data available ▪ -2 = no data copied ▪ -4 = no data available ▪ -5 = invalid external variable configuration The port is available only if the **Use 'Status Port'** checkbox is enabled on the Options Page (RTIBYPASS_WRITE_BLx).

Dialog pages

The dialog settings can be specified on the following pages:

- Unit Page (RTIBYPASS_WRITE_BLx) on page 80
- Variables Page (RTIBYPASS_WRITE_BLx) on page 81
- Options Page (RTIBYPASS_WRITE_BLx) on page 85

Note

A Write block needs exclusive access. For this reason, all the RTIBYPASS block dialogs (except for the dialogs of the RTIBYPASS_INFO_BLx and RTIBYPASS_CAL_PAGE_SWITCH_BLx blocks) must be closed if you want to open a Write block dialog, and cannot be opened until the Write block dialog is closed again.

Related topics

Basics

Overview of the RTI Bypass Blockset.......... 15

References

RTIBYPASS_SETUP_BLx.......... 51

Unit Page (RTIBYPASS_WRITE_BLx)

Purpose

To specify the bypass interface and the service instance.

Dialog settings

Bypass interface Lets you select the bypass interface for writing variables to the ECU. All the bypass interfaces you configured via Setup blocks in your Simulink model are listed.

Service instance Lets you select the service instance of the ECU where the block is to write the variables. An ECU service can be executed in different rasters (service instances) which are defined in the ECU´s database file. The names of the service instances are unique, and each service instance corresponds to one service ID. The **Service instance** drop-down list contains the available service instances, determined by the selected imported database files for the selected bypass interface. By default, no service instance is preset and the 'NOT_DEFINED' value is displayed.

Service ID Displays the service ID associated with the selected service instance. The service IDs are unique, and each ID corresponds to one service instance.

Related topics

References

Block Description (RTIBYPASS_WRITE_BLx).......... 76
Options Page (RTIBYPASS_WRITE_BLx).......... 85
Variables Page (RTIBYPASS_WRITE_BLx).......... 81

Variables Page (RTIBYPASS_WRITE_BLx)

Purpose

To view the available ECU variables (parameters and measurement variables), and to select the variables to be written to the ECU for bypassing.

Dialog settings

Fill Variable Selector Fills the Variable Selector with the variables defined in the database files you imported for the selected bypass interface.

Note

Depending on the size of the database file, filling the Variable Selector can take some seconds. You should use this button only if variables are to be added to the list.

Variable Selector - Hierarchy tree In the hierarchy view mode, the structure of the imported variable description files is displayed in the hierarchy tree next to the variable list. The variable list shows the variables of the selected node and its subnodes. You can activate and deactivate the hierarchy view mode via the context menu of the variable list or hierarchy tree.

Variable Selector - Variable list The variable list of the Variable Selector displays the available ECU variables defined in the database files you imported for the selected bypass interface. Additional information on the ECU variables, for example, the description, memory address, and type, are also displayed.

If variables must be written for bypassing, you must add them from the variable list to the Selected Variables list. There are different ways of selecting variables:

- You can drag the variables from the variable list to the Selected Variables list.
- You can activate the checkboxes of the variables in the variable list using the mouse or via the **`Space`** key, and then press **`Return`** to add the checked variables to the Selected Variables list. When the variables are added to the Selected Variables list, their checkboxes are cleared. You can clear all checkboxes in the variable list without applying the variables to the Selected Variables list via the Uncheck All command, which is accessible via the context menu of a list entry.
- If the Select variable(s) with 'RETURN' checkbox is selected on the Options page, you can apply variables that are highlighted in the variable list to the Selected Variables list just by pressing the **`Return`** key without prior activation of their checkboxes.

Several methods of sorting, searching and filtering in the variable list are provided:

- *Sorting alphabetically*: You can sort the variable list alphabetically in ascending or descending order for a selected column. You can access the command by right-clicking the column header and choosing the function from the context menu.

- *Searching for a variable via buffered search*: You can search for a variable in the variable list using buffered search. When you type the first characters of one of its column entries, all keystrokes are stored in a buffer. The first item that matches the current keystroke sequence is selected. You can delete the whole keystroke sequence with `Delete`, but the selection remains. If the **Clear Search Buffer after variable(s) selection** option is selected on the Options page, the search buffer is cleared automatically after you select variables from the variable list. To differentiate between upper and lower case letters in your buffered search, open the context menu and select the **Case-Sensitive Search** command.
- *Filtering*: You can filter the list of variables by variable types. On the Options page, you can select whether only measurement variables are to be displayed in the variable list, or both measurement variables and parameters. The active filter setting is displayed in the header of the variable list.
- *Variable Properties*: You can view the properties of a variable. The information is displayed in a dialog which you can access via the context menu of the variable. The contents of the variable properties dialog depend on the type of the selected variable.

Selected Variables The **Selected Variables** list contains the variables you selected from the variable list for bypassing. The address, data type, conversion method from physical representation (converted value) to ECU representation (source value), integer conversion method, variable description, and minimum and maximum values can be displayed for each variable in the **Selected Variables** list.

You can add custom variables to the **Selected Variables** list for bypassing via the **Add Custom** command. A custom variable has a *(custom)* declaration after its name.

You can add columns to the **Selected Variables** list or remove them from it via the **Customize Columns** command in the context menu of the list's column header. To add a column, drag the corresponding entry from the **Customize Columns** dialog to the column header. To remove a column, drag the corresponding column header entry from the **Selected Variables** list to the **Customize Columns** dialog. By default, the **Selected Variables** list contains the **Variable**, **Address**, **Type**, **Convert**, **Enable Remapping**, **Port Data Type**, and **Rounding Offset [0 ... 1)** columns.

- The variable can be converted from converted value to source value if the computation method is supported by the RTI Bypass Blockset.
- Variable conversion is never performed for a conversion formula that is not supported by the RTI Bypass Blockset. The converted value is used in the ECU, and the checkbox beside the conversion method is disabled. For information on the supported computation methods, refer to Limitations of the RTI Bypass Blockset on page 269.
- You can deactivate the conversion of a variable by clearing its **Convert** checkbox in the **Selected Variables** list. In that case the input corresponds 1:1 to the value written to the ECU.

- If you perform on-target bypassing and the TargetLink Code Generator is selected as the code generator on the Build Page of the corresponding Setup block, you can specify the port data type for a variable with deactivated conversion via its **Port Data Type** field. You can select between Double and the fixed-point data type specified for the variable in the A2L file. If variable conversion is enabled for the variable later on, the RTI Bypass Blockset automatically uses Double as the port data type for the variable, regardless of which port data type setting you have made.

  If you use Simulink Coder as the code generator, the port data type of a variable is always set to Double.

Before a variable value is written to the ECU, the value used by the bypass system must be converted to Integer data type. Conversion is done according to the following conversion method:

- For variables of unsigned data type:
  - convertedValue = int (unconvertedValue + offset)
- For variables of signed data type:
  - convertedValue = int (unconvertedValue + offset)

    if unconvertedValue ≥ 0
  - convertedValue = int (unconvertedValue - offset)

    if unconvertedValue < 0

You can configure the integer conversion method to be used by specifying the offset value which is responsible for rounding. Double-clicking the **Rounding Offset [0 … 1)** entry of the variable lets you enter a different offset value. Only offset values according to 0 ≤ offset < 1 are valid.

The variables will be input to the Write block in the same order as they are displayed in the **Selected Variables** list. The variable names are used as port names. You can change the order of the variables and delete variables from the list.

Move Up Moves the currently selected variables further up the **Selected Variables** list. As a result, the corresponding inports of the Write block also move up.

Move Down Moves the currently selected variables further down the **Selected Variables** list. As a result, the corresponding inports of the Write block also move down.

Delete Deletes the currently selected variables from the **Selected Variables** list. The corresponding inports of the Write block are also deleted.

Delete All Deletes all variables from the **Selected Variables** list. The corresponding inports of the Write block are also deleted.

Add Custom Lets you add a custom variable. The Add/Edit Custom Variables Properties dialog opens for you to specify the variable settings.

View Variable Properties (Available only from the context menu of a variable in the Selected Variables list added to the Variable Selector) Lets you view the variable properties of the selected variable. The Variable Properties dialog opens and displays the properties.

Edit/View Custom Variables Properties (Available only from the context menu of a custom variable in the Selected Variables list) Lets you view and edit the properties of the selected custom variable. The Add/Edit Custom Variables Properties dialog opens for you to view and change the settings.

Add/Edit Custom Variables Properties dialog

Name Lets you specify the name of the custom variable. The name you enter must start with a letter. Otherwise, it will be prefixed with '_' automatically. Only the following characters are allowed: [a-z], [A-Z], [0-9], '.', '_', '[' and ']'.

Display identifier Displays the display identifier that will be used to name the output ports of the block, when the Use Display Identifier checkbox is selected on the Variables Options page in the Setup block dialog. It contains the capitalized variable name.

Address Lets you specify the address of the custom variable.

Data type Lets you select a data type for the custom variable. If you change the data type, the bitmask and the limit properties are reset to the defaults for the selected data type.

Byte order Displays the byte order specified in the interface description ASAP2 file and imported in the Setup block dialog. It can be either Motorola format (big endian, MSB at the memory location with the lowest address) or Intel format (little endian, LSB at the memory location with the lowest address).

Description Lets you specify a description text for the custom variable.

Start bit Lets you enter a bit offset for the bit mask, counted from the right side of the bit mask.

Number of bits Lets you specify the bit mask width by entering the number of contiguous 1-bits.

Hex (Bit mask) Displays the bit mask value in hexadecimal notation.

The following example shows the result of different bit mask property settings:

Data Type	No. of Bits	Start Bit	Bit Mask	Hex
8-bit	4	0	00001111	F
8-bit	4	2	00111100	3C
16-bit	5	5	0000001111100000	3E0

Min weak (phys.) Lets you specify the lower weak limit of the physical value. You can only enter values that are inside the hard limits.

Max weak (phys.) Lets you specify the upper weak limit of the physical value. You can only enter values that are inside the hard limits. If not specified a default value is automatically generated.

Hex (Limits) Displays the source values in hexadecimal notations.

Note

The limits are automatically adapted if you perform one of the following actions:

- You select another data type.
- You select another computation method.

Computation method Lets you specify a computation method. A computation method is used to transform a source value into a converted value.

Select one of the following computation methods and specify the conversion rule:

- `LINEAR`: You can specify a linear function.
- `RAT_FUNC`: You can specify a fractional rational function.
- `TAB_INTP`: You can specify a numeric conversion table for tabular conversion with interpolation.
- `TAB_NOINTP`: You can specify a numeric conversion table for tabular conversion without interpolation.
- `FORM`: You can specify a formula to calculate the physical value from the source value, and/or an inverse formula to calculate the source value from the converted value.

Related topics

References

Block Description (RTIBYPASS_WRITE_BLx).................. 76
Options Page (RTIBYPASS_WRITE_BLx).................. 85
Unit Page (RTIBYPASS_WRITE_BLx).................. 80

Options Page (RTIBYPASS_WRITE_BLx)

Purpose

To set up block ports and specify bypass interface-specific settings for the RTIBYPASS_WRITE_BLx block.

Dialog settings

Use 'Enable Service' Port Lets you enable the use of the Enable Service inport. Via the Enable Service port, you can enable or disable the execution of the part of the service instance related to the corresponding RTIBYPASS block. If the checkbox is selected, the Write block gets an additional Enable Service inport. If the value fed to this Enable Service inport is equal to or smaller than 0, the variables are no longer written to the ECU by the RCP system or bypass flash, and the ECU service will no longer overwrite the selected variables in the ECU application memory. Enabling and disabling the Enable Service port for this Write block does not influence other Read, Write or Interrupt blocks using the same service instance.

Use 'Status Port' Lets you enable the use of the Status Port outport. You can read status information via this port. The following Status Port outport values of the Write block are available:

Status Port Outport Value	Description
0	New data available
-1	Old data available
-2	No data copied
-4	No data available
-5	Invalid external variable configuration

Note

Because the XCP protocol does not provide feedback of the success or failure of a data stimulation, the Write block currently only returns the following status values for XCP interfaces:

- 0 (data was transferred successfully)
- -4 (no XCP connection, TX error)

The fact that the Status Port outport of the Write block does not display all status values for XCP interfaces (and therefore not for the DCI-GSI2) can be annoying. However, if you work with the DCI-GSI2, there is a workaround to get the status information about the Write block with the help of the failure checking mechanism: Read the `GSI2.EVENT.StimFailureCounter` system variable supported by the DCI-GSI2 with a Read block, using the same event that is used to execute the Write block. This variable counts the failures of the event channel it is measured with. You can then infer the status of the Write block from the value of the system variable, as shown in the following table:

Value of the System Variable	Meaning for the Status of the Write Block
0 (no STIM timeout occurred)	Data was transferred successfully. New data is available.
Value is between 0 and failure limit	Old data is available. New data is not in time.
254	No data is available.
255 (configured failure limit is reached)	No data is copied.

For the failure checking mechanism to measure sampling step delays, the buffer synchronization must be enabled. For the DCI-GSI2, you set the buffer synchronization in the DCI Configuration Tool on the **Event Channels** page. When bypassing an ECU function, there usually is a pre-functional service call that is used to measure the ECU inputs, and a post-functional service call that is used to write the results of the bypass function to the ECU memory. To synchronize the post-functional service call with the pre-functional one, you must configure the service ID of the pre-functional event channel as the reference service ID of the post-functional event channel. As a result, the data stimulated in the service call (Parameter 1) is synchronized with the service call of the reference service ID (Parameter 2), which allows the service to detect a sampling step delay. Refer to Event Channels Page (DCI Configuration 📖).

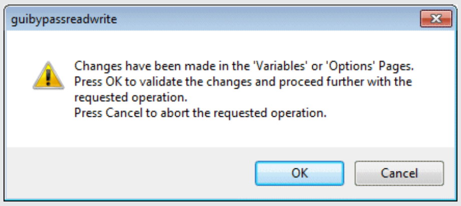
Show port names Lets you select whether the port names are displayed at the inports and outports. The ports themselves are always displayed.

Tip

Deactivating this option reduces the size of the block.

Note

You cannot make changes to the port name settings and to the settings affecting the number of block ports at the same time. Instead, you have to apply the changes for one kind of setting first and then modify the other settings.
For example, if you add or delete variables to or from the Selected Variables list, or modify the Use 'Status Port' or Use 'Enable Service' Port options, the number of block ports is changed. Because of this, the Show port names option is grayed out until you apply your modifications for the port number by clicking OK or Apply. If you click on the grayed checkbox, a dialog similar to the following opens.


guibypassreadwrite

Changes have been made in the 'Variables' or 'Options' Pages.
Press OK to validate the changes and proceed further with the requested operation.
Press Cancel to abort the requested operation.

After you confirm the changes by clicking OK in the dialog, the settings are saved in the block parameters, and the Show port names option is no longer grayed out.

Set output ports signal labels Lets you specify to display the port names as signal labels for the outports.

If the Create unique signal names from RTIBypass ports option is activated on the Variables Options page in the Setup block (see Variables Options Page (RTIBYPASS_SETUP_BLx) on page 56), the automatically assigned signal names are extended to model-wide unique identifiers.

Filter 'Measurements' only Lets you select the variable types you want to filter the variable list on the Variables page by. If the checkbox is selected, only measurement variables are displayed in the variable list. If the checkbox is cleared, both measurement variables and parameters are displayed.

Fill 'Variable Selector' automatically Lets you select whether the variable selector on the Variables page is filled automatically each time you open the page.

Select variable(s) with 'RETURN' Lets you specify whether variables can be added to the Selected Variables list on the Variables page via the `Return` key.

If the checkbox is activated, variables that are highlighted in the variable list can be added to the Selected Variables list by pressing `Return`, without having to activate their checkboxes first.

Clear Search Buffer after variable(s) selection Lets you select whether the search buffer is cleared automatically after you added variables from the variable list to the Selected Variables list on the Variables page via the `Return` key.

Enable Convert as Default Lets you specify the default setting for the Convert checkbox for variables that are added to the Selected Variables list on the Variables page. If enabled, the Convert checkbox is selected for a variable by default after the variable is added to the Selected Variables list, which means that variable conversion is activated for it. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

Enable Remapping as Default Lets you specify the default setting for the Enable Remapping checkbox for variables that are added to the Selected Variables list on the Variables page. If enabled, the Enable Remapping checkbox is selected for a variable by default after the variable is added to the Selected Variables list, which means that remapping to ECU internal variables is activated for it. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

Use Port Data Type 'Double' as Default (Available only if TargetLink Code Generator is selected as the code generator) Lets you specify the default setting for the Port Data Type setting for variables that are added to the Selected Variables list on the Variables page. If the checkbox is selected, Double is used as the default port data type for a newly selected variable with disabled conversion. If the checkbox is cleared, the fixed-point data type specified for a variable in the A2L file is selected as the port data type by default. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

If you work with Simulink Coder as the code generator, the port data type for the selected variables is always set to Double regardless of whether variable conversion is enabled or disabled for the variables.

Enable external variable ports Lets you specify whether variables to be used with this block can be defined during run-time from outside the RTI Bypass Blockset. If the checkbox is selected, further Simulink block ports that allow you to configure variables by external block inputs are added to the Write block. For information on the additional ports, refer to Block Description (RTIBYPASS_WRITE_BLx) on page 76.

Note

- The ExtVar Count, ExtVar Address and ExtVar Type input port values are applied only if the block is executed for the first time after the dSPACE real-time application was started, or if the block was disabled and is then enabled again via the Enable Service input port.
  Since the ExtVar input ports are applied and the block containing external variables is enabled only if the block is executed, it must be guaranteed that the block is executed even if it is placed in a subsystem triggered by an RTI Bypass Interrupt block. The RTI Bypass Blockset usually implicitly guarantees that a subsystem is triggered independently of the enable state of the RTI Bypass Read and Write blocks. Under certain conditions, however, an additional Read block without external variables must be added to ensure an initial subsystem trigger and continuous triggering if all blocks with external variables are disabled. For further information, refer to Block Description (RTIBYPASS_INTERRUPT_BLx) on page 92.
- Depending on the configuration type (foreground or background configuration), the availability of read or written values is delayed for one or more subsystem execution steps. You can get information on the availability state of read or written values via the optional Status Port output port. For further information, refer to Perform variable configuration in foreground on the block's Option page.

Note

To ensure that Write blocks with external variable content acquire the external variable configuration, the related subsystem must be executed.

Maximum number of variables (Available only if Enable external variable ports is selected.) Lets you specify the maximum number of externally defined variables that can be used with this block.

The number of actually used variables is read by the ExtVar Count block inport. ExtVar Count can be in the range 0 ... 'Maximum number of variables'. If the specified number exceeds the upper limit, the value is reduced to 'Maximum number of variables'.

Enable block mode (Available only if Enable external variable ports is selected.) Lets you specify whether to use the block mode or single mode with this block. If the checkbox is selected, the block mode is enabled. With the block mode enabled, the ExtVar input ports allow several variables of the same data type to be read by just defining one start address for the whole data block and the number of values to be read. With the block mode disabled, the address, data type and value conversion must be defined separately for each variable.

Enable conversion port (Available only if Enable external variable ports is selected.) Lets you enable or disable the ExtVar Conversion block inport. If the checkbox is selected, the ExtVar Conversion inport is available and can be used to define the computation methods for converting values of the variables that are written to the ECU with this block. If the checkbox is cleared, the ExtVar

Conversion inport is not available and data conversion is disabled for the variables written by this block.

Perform variable configuration in foreground (Available only if Enable external variable ports is selected.) Lets you enable or disable foreground reconfiguration of externally defined variables. If the checkbox is selected, the variables are reconfigured in the model foreground while the Write block is executed. The changed variable values are written immediately after the reconfiguration is performed. If the checkbox is cleared, variable reconfiguration is performed in the model background after the associated task is executed. The point in time the changed variable values are available in the ECU is undefined, but not before the subsequent sampling step.

Via the Status Port outport of the Write block, you can get information on whether no or new data is available, or whether the external variable definitions are invalid.

Configuration in the foreground can increase the task execution time, depending on the bypass interface used. Configuration in the model background does not affect the task execution time.

Perform enable in foreground (Available only for external bypassing) Lets you specify whether enabling the block is processed in the foreground. If the checkbox is selected, the block is enabled in the model foreground. If the checkbox is cleared, enabling the block is performed in the model background.

Perform disable in foreground (Available only for external bypassing) Lets you specify whether disabling the block is processed in the foreground. If the checkbox is selected, the block is disabled in the model foreground. If the checkbox is cleared, disabling the block is performed in the model background.

Configure Opens a configuration dialog for specifying bypass interface-specific settings for the RTIBYPASS_WRITE_BLx block. The dialog depends on the selected bypass interface type.

For information on the bypass interface-specific configuration, refer to the corresponding topic:

- XCP on CAN: Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) on page 159
- XCP on UDP/IP: Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) on page 173
- XCP on FlexRay: Options Page (RTIBYPASS_WRITE_BLx for XCP on FlexRay) on page 188
- dSPACE on DPMEM: Options Page (RTIBYPASS_WRITE_BLx for dSPACE on DPMEM) on page 201
- INTERNAL: Options Page (RTIBYPASS_WRITE_BLx for INTERNAL) on page 222

Related topics

Basics

Manual Service Parameter Configuration (DCI-GSI2 Setup Application Note 📖)
Status Information on the ECU and DCI-GSI2 (DCI-GSI2 Feature Reference 📖)

References

RTIBYPASS_INTERRUPT_BLx

Purpose

To make interrupts generated by the ECU available as trigger sources in a Simulink model.

Where to go from here

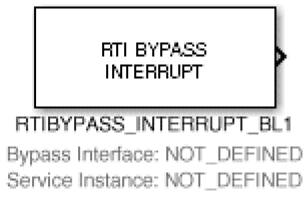
Information in this section

Information in other sections

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_INTERRUPT_BLx)

Block



Note

The RTIBYPASS_INTERRUPT_BLx block has no name extension in the block library. When the block is copied to a Simulink model, the bypass interface name and, if a bypass interface for service-based bypassing is used, the service instance name are added to the block name.

Purpose

To make interrupts generated by the ECU available as trigger sources in a Simulink model.

Description

The RTIBYPASS_INTERRUPT_BLx block configures the interrupts to be generated by the ECU during initialization and receives the interrupts from the ECU during run time.

Service-based bypassing For interrupt configuration, you must specify the bypass interface and the service instance to be used for interrupt generation. You must also select a subinterrupt number and define the location in the ECU service code where the interrupt should be triggered. All service instances process first the read blocks and then the write blocks. Interrupt generation is possible after all read operations are executed and before the write operations are performed or at the end of the service instance after all read and write operations are executed. The RTIBYPASS_INTERRUPT_BLx block receives an interrupt from the ECU during run time to trigger the bypass system that the input data is available.

Using RTIBYPASS_INTERRUPT_BLx blocks therefore requires the use of at least one explicit Read block to receive the interrupts, if any one of the following conditions applies:

- CCP is used.
- XCP on FlexRay is used.
- XCP on CAN or XCP on UDP/IP with static DAQ lists is used.
- XCP on CAN or XCP on UDP/IP with dynamic DAQ lists is used, and the **Use additional DAQ lists to trigger interrupts always** option is disabled.
- INTERNAL bypassing is used.

Note

The explicit Read block must contain static variables only. It must not contain external variables. The block must be configured for the service instance specified in the Interrupt block.

In all other cases, a dummy Read block is automatically generated for every interrupt when the ECU service is initialized by the model. If all the Read/Write blocks of the interrupt are disabled or no Read/Write block exists, the dummy Read/Write block is used instead. Thus, an interrupt is always triggered, even if you enabled neither a Read nor a Write block.

An explicit (non-dummy) Read block is also required, if DPMEM is used, and your model contains a Write block that is synchronized with the execution of the service instance specified in the Interrupt block. The Read block is used to ensure buffer synchronization between the Write block and the service instance.

Code-patch-based bypassing To trigger the bypass function on the RCP system, you can use subinterrupts or a single hardware interrupt.

- In connection with subinterrupt-based bypassing, you must specify the code-patch-based bypass interface and select the subinterrupt number to be used.

  Note

  The subinterrupt configuration specified in the `Interrupt_Handling` struct in the ECU's A2L file must match the subinterrupt configuration in the ECU code.

  The triggering of the bypass function on the RCP system must be implemented by means of subinterrupt handling in the ECU code. For further information on the supported subinterrupt mechanisms, refer to Code-Patch-Based Bypassing: Subinterrupt Mechanisms on page 232.
- If a single hardware interrupt is used, only one subsystem can be triggered in the Simulink model, so only one RTIBYPASS_INTERRUPT_BLx block can be used in the model. To use the hardware interrupt, the A2L file must not contain the `Interrupt_Handling` struct describing the subinterrupt mechanism. For hardware interrupt configuration, only the interrupt trigger (GPIO pin toggle, watchpoint initialization) must be realized in the ECU application. If the ECU code contains an appropriate memory access that can be adapted as a trigger condition, no ECU code modifications are required for bypass interfaces which support autonomous watchpoint initialization.

  Using a single hardware interrupt allows very fast transmission and interrupt decoding. However, the ECU code does not contain a waiting sequence for synchronization purposes. If necessary, you must implement a waiting sequence for receiving the results from the RCP system in the ECU code and the Simulink mode yourself.

I/O characteristics

This block is the trigger source for the function-call subsystem which normally contains the bypass algorithm that should replace the algorithm on the ECU.

Dialog pages

The dialog settings can be specified on the following pages:
- Unit Page (RTIBYPASS_INTERRUPT_BLx) on page 95
- Options Page (RTIBYPASS_INTERRUPT_BLx) on page 96

Note

An Interrupt block needs exclusive access. For this reason, all the RTIBYPASS block dialogs (except for the dialogs of the RTIBYPASS_INFO_BLx and RTIBYPASS_CAL_PAGE_SWITCH_BLx blocks) must be closed if you want to open an Interrupt block dialog, and cannot be opened until the Interrupt block dialog is closed again.

Related topics

Basics

References

Unit Page (RTIBYPASS_INTERRUPT_BLx)

Purpose

To specify the bypass interface and, if service-based bypassing is to be performed, the service instance.

Dialog settings

Bypass interface Lets you select the bypass interface. All bypass interfaces you configured in your Simulink model via Setup blocks are listed.

Service instance (Available only for service-based bypassing) Lets you select the service instance which specifies the ECU service code point where the ECU service is to trigger the interrupt. An ECU service can be executed in different rasters (service instances) which are defined in the ECU´s database file. The names of the service instances are unique, and each service instance corresponds to one service ID. The **Service instance** drop-down list contains the available service instances, determined by the selected imported database files for the selected bypass interface. By default, no service instance is preset and the 'NOT_DEFINED' value is displayed.

Service ID (Available only for service-based bypassing) Displays the service ID associated with the selected service instance. The service IDs are unique, and each ID corresponds to one service instance.

Related topics

References

Options Page (RTIBYPASS_INTERRUPT_BLx)

Purpose

To set up the interrupt handling and specify bypass interface-specific settings for the RTIBYPASS_INTERRUPT_BLx block.

Dialog settings

Subinterrupt Number Lets you assign the number of the subinterrupt. The number must be unique for the bypass interface used, so it may be used only once with each Setup block.

The range of valid numbers depends on the selected bypass interface:

- XCP on CAN, XCP on UDP/IP, XCP on FlexRay, and CCP: The valid numbers are 0 ... 127.
- DPMEM: The valid numbers are specified in the A2L file. The maximum number of subinterrupts must be an even number in the range 2 ... 128. If the maximum number is N, the valid subinterrupt numbers are 0 ... (N-1).
- Others: The number of subinterrupts is specified in the A2L file. The valid subinterrupt numbers are 0 … (maximum number of subinterrupts -1). However, the number of subinterrupts is limited to 127.

If the A2L file does not contain the `Interrupt_Handling` struct describing the subinterrupt mechanism, the Subinterrupt Number edit field is disabled, and you can only use the hardware interrupt.

Interrupt Location (Available only for service-based bypassing) The interrupt location defines the location in the ECU service code where the interrupt is to be generated. The ECU services execute the read operations before the write operations, that is, all service instances first process all read blocks and then all write blocks. You can choose between two possible interrupt locations:

- Generate interrupt after READS finished: Specifies that an interrupt is generated after all read operations are executed and before the write operations are performed. If no enabled Read block is assigned to the current service instance, the interrupt is triggered directly before the write operations are performed.
- Generate interrupt after WRITES finished: Specifies that an interrupt is generated after the write operations are executed, that means, at the end of the service instance. If no enabled Write block is assigned to the current service instance, the interrupt is triggered directly after the read operations are executed.

If no enabled Read or Write blocks are assigned to the current service instance, interrupts with 'Generate interrupt after READS finished' interrupt location and interrupts with 'Generate interrupt after WRITES finished' interrupt location are triggered directly one after another. This is realized by the dummy Read/Write blocks that are automatically generated for every interrupt, when the ECU service is installed.

Note

If XCP on CAN is used, additional CAN interrupts are automatically generated:

- RX interrupt: interrupt that is triggered after a specified RX (Receive) message is received
- TX interrupt: interrupt that is triggered after a specified TX (Transmit) message was sent successfully

If XCP on UDP/IP is used, additional interrupts are automatically generated:

- RX interrupt: interrupt that is triggered after at least one specified RX message is received

The TX and RX interrupts must have higher priorities than the software interrupts. You should check the task priorities of your model in the RTI Task Configuration dialog.

Configure Opens a configuration dialog to specify bypass interface-specific interrupt settings.

Note

Configuring bypass interface-specific interrupt settings is necessary only for some bypass interfaces. Depending on the bypass interface you selected on the Unit page, this button might not be available.

Related topics

References

Block Description (RTIBYPASS_INTERRUPT_BLx).. 92
Unit Page (RTIBYPASS_INTERRUPT_BLx)... 95

RTIBYPASS_UPLOAD_BLx

Purpose

To upload variables from the ECU.

Where to go from here

Information in this section

Information in other sections

Depending on the selected bypass interface, you have to specify interface-specific settings:

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_UPLOAD_BLx)

Block

RTI BYPASS
UPLOAD
RTIBYPASS_UPLOAD_BL1
Bypass Interface: NOT_DEFINED

Note

The RTIBYPASS_UPLOAD_BLx block has no name extension in the block library. When the block is copied to a Simulink model, the bypass interface name is added to the block name.

Purpose

To upload variables from the ECU.

Description

The RTIBYPASS_UPLOAD_BLx block uploads ECU variables and parameters from the ECU to the prototyping hardware. An upload is performed each time the RTIBYPASS_UPLOAD_BLx block is executed in the Simulink model.

In this block, you must select the variables to be uploaded and specify the bypass interface to use for the upload.

Note

The RTIBYPASS_UPLOAD_BLx block is supported by the following bypass interfaces:

- XCP on CAN
- XCP on UDP/IP
- XCP on FlexRay
- CCP
- INTERNAL
- VECU

Note

The RTIBYPASS_UPLOAD_BLx block neither provides real-time capabilities nor ensures data consistency for the transferred data. It is recommended to use the RTIBYPASS_READ_BLx block instead.

I/O characteristics

The table below shows the Simulink block input. The variable names are displayed at the inports only if the **Show port names** checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Input	Simulink Data Type	Description
Enable Service	Double	Lets you enable/disable the execution of the RTIBYPASS_UPLOAD_BLx block: ▪ > 0 : block is enabled ▪ ≤ 0 : block is disabled The port is available only if the **Use 'Enable Service' Port** checkbox is enabled on the Options Page (RTIBYPASS_UPLOAD_BLx). If the **Enable external variable ports** checkbox is selected on the Options Page (RTIBYPASS_UPLOAD_BLx), the **Enable Service** port is mandatory, since dynamic reconfiguration of externally defined variables during run time requires the Upload block to be disabled during reconfiguration of the variable values, and then enabled again.
ExtVar Count	Double	Lets you define the number of externally defined variables actually used. The value must be in the range 0 ... **Maximum number of variables**. (The maximum number of variables is specified on the Options page.) For higher values, processing is reduced to the specified maximum number of variables. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Address	▪ Double (if block mode is enabled) ▪ Double array (if block mode is disabled)	Lets you define the address values of the externally defined variables. If the block mode is enabled on the Options page, the value indicates a single address specifying the start address for a whole data block. If the block mode is not enabled, an array of size 'Maximum number of variables' provides the address values for the single value accesses. (The maximum number of variables is specified on the Options page.) The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Type	▪ Double (if block mode is enabled) ▪ Double array (if block mode is disabled) The input values can have the following data types: ▪ 8: 8-bit unsigned ▪ 16: 16-bit unsigned ▪ 32: 32-bit unsigned ▪ -8: 8-bit signed ▪ -16: 16-bit signed ▪ -32: 32-bit signed ▪ -1032: float 32-bit ▪ 0: external variable disabled	Lets you define the data types for the externally defined variables. If the block mode is enabled on the Options page, a single value indicates the data type of the block members. If the block mode is not enabled, an array of size 'Maximum number of variables' provides the data type values for the single values. (The maximum number of variables is specified on the Options page.) External variables can be disabled selectively during run time. The data type value of the related input element must be set to 0 for this. The block must be disabled during the change. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.

Simulink Input	Simulink Data Type	Description
ExtVar Conversion	▪ 1x6 double array (if block mode is enabled) ▪ nx6 double array, where n is the maximum number of variables (if block mode is disabled)	Lets you define the value conversion information for the externally defined variables. The conversion information is provided as a 1-dimensional array of 6 elements (if block mode is enabled) or of 'Maximum number of variables' x 6 elements (if block mode is disabled). The maximum number of variables is specified on the Options Page (RTIBYPASS_UPLOAD_BLx). Six coefficients a ... f are specified for each variable according to the fractional rational function (a · x^2 + b · x + c) / (d · x^2 + e · x + f) used as the computation method for converting a source value into a converted value. **Note** ▪ Only linear conversion is possible for externally defined variables accessed by an RTIBYPASS_UPLOAD_BLx block. The conversion method therefore meets the linear function (f · x / b) - (c / b). ▪ The formula must always be defined in RCP to ECU direction, regardless of the block it is used with. ▪ Since the input is always a 1-dimensional array, it is recommended to use an S-function which integrates multidimensional data into a 1-dimensional array with this input port. For an implementation example, refer to the demo model in the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS` folder. You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>. The port is available only if the Enable conversion port checkbox is selected on the Options Page (RTIBYPASS_UPLOAD_BLx).

Note

- The ExtVar Count, ExtVar Address and ExtVar Type input port values are applied only if the block is executed for the first time after the dSPACE real-time application was started, or if the block was disabled and is then enabled again via the Enable Service input port.
  Since the ExtVar input ports are applied and the block containing external variables is enabled only if the block is executed, it must be guaranteed that the block is executed even if it is placed in a subsystem triggered by an RTI Bypass Interrupt block. The RTI Bypass Blockset usually implicitly guarantees that a subsystem is triggered independently of the enable state of the RTI Bypass Read and Write blocks. Under certain conditions, however, an additional Read block without external variables must be added to ensure an initial subsystem trigger and continuous triggering if all blocks with external variables are disabled. For further information, refer to Block Description (RTIBYPASS_INTERRUPT_BLx) on page 92.
- Depending on the configuration type (foreground or background configuration), the availability of read or written values is delayed for one or more subsystem execution steps. You can get information on the availability state of read or written values via the optional Status Port output port. For further information, refer to Perform variable configuration in foreground on the block's Option page.

The table below shows the Simulink block output. The variable names are displayed at the outports only if the **Show port names** checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Output	Simulink Data Type	Description
<name of variable>	Double	Uploads the variable. The number of outports is equal to the number of selected variables on the Variables Page (RTIBYPASS_UPLOAD_BLx).
Status Port (or other service-specific outports)	Double	Provides run-time information on the status of the block's current upload operation: ▪ 0 = new data available (data uploaded) ▪ -1 = old data available (error during upload, last recent valid data set used) ▪ -4 = no data available (error during upload, no valid data set available) ▪ -5 = invalid external variable configuration The port is available only if the **Use 'Status Port'** checkbox is enabled on the Options Page (RTIBYPASS_UPLOAD_BLx).
ExtVar Values	Double array	Provides the values of the externally defined variables. The uploaded values are output as an array of size 'Maximum number of variables'. The maximum number of variables is specified on the Options Page (RTIBYPASS_UPLOAD_BLx). Only the values from 0 ... **ExtVar Count** are uploaded. The other values remain unchanged until they are affected by a modified **ExtVar Count** input value. If value conversion is enabled for the variables, the values are converted before they are uploaded from the ECU. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.

Dialog pages

The dialog settings can be specified on the following pages:

- Unit Page (RTIBYPASS_UPLOAD_BLx) on page 103
- Variables Page (RTIBYPASS_UPLOAD_BLx) on page 103
- Options Page (RTIBYPASS_UPLOAD_BLx) on page 107

Note

An Upload block needs exclusive access. For this reason, all the RTIBYPASS block dialogs (except for the dialogs of the RTIBYPASS_INFO_BLx and RTIBYPASS_CAL_PAGE_SWITCH_BLx blocks) must be closed if you want to open an Upload block dialog, and cannot be opened until the Upload block dialog is closed again.

Related topics

Basics

Overview of the RTI Bypass Blockset.. 15

References

RTIBYPASS_SETUP_BLx.. 51

Unit Page (RTIBYPASS_UPLOAD_BLx)

Purpose

To specify the bypass interface.

Dialog settings

Bypass interface Lets you select the bypass interface. All bypass interfaces you configured in your Simulink model via Setup blocks are listed.

Related topics

References

Variables Page (RTIBYPASS_UPLOAD_BLx)

Purpose

To view the available ECU variables (parameters and measurement variables), and to select the variables that should be uploaded from the ECU.

Dialog settings

Fill Variable Selector Fills the Variable Selector with the variables defined in the database files you imported for the selected bypass interface.

Note

Depending on the size of the database file, filling the Variable Selector can take some seconds. You should use this button only if variables are to be added to the list.

Variable Selector - Hierarchy tree In the hierarchy view mode, the structure of the imported variable description files is displayed in the hierarchy tree next to the variable list. The variable list shows the variables of the selected node and its subnodes. You can activate and deactivate the hierarchy view mode via the context menu of the variable list or hierarchy tree.

Variable Selector - Variable list The variable list of the Variable Selector displays the available ECU variables defined in the database files you imported for the selected bypass interface. Additional information on the ECU variables, for example, the description, memory address, and type, are also displayed.

If variables are to be uploaded from the ECU, you must add them from the variable list to the Selected Variables list. There are different ways of selecting variables:

- You can drag the variables from the variable list to the Selected Variables list.

- You can activate the checkboxes of the variables in the variable list using the mouse or via the **Space** key, and then press **Return** to add the checked variables to the Selected Variables list. When the variables are added to the Selected Variables list, their checkboxes are cleared. You can clear all checkboxes in the variable list without applying the variables to the Selected Variables list via the Uncheck All command, which is accessible via the context menu of a list entry.
- If the Select variable(s) with 'RETURN' checkbox is selected on the Options page, you can apply variables that are highlighted in the variable list to the Selected Variables list just by pressing the **Return** key without prior activation of their checkboxes.

Several methods of sorting, searching and filtering in the variable list are provided:

- *Sorting alphabetically*: You can sort the variable list alphabetically in ascending or descending order for a selected column. You can access the command by right-clicking the column header and choosing the function from the context menu.
- *Searching for a variable via buffered search*: You can search for a variable in the variable list using buffered search. When you type the first characters of one of its column entries, all keystrokes are stored in a buffer. The first item that matches the current keystroke sequence is selected. You can delete the whole keystroke sequence with **Delete**, but the selection remains. If the Clear Search Buffer after variable(s) selection option is selected on the Options page, the search buffer is cleared automatically after you select variables from the variable list. To differentiate between upper and lower case letters in your buffered search, open the context menu and select the Case-Sensitive Search command.
- *Filtering*: You can filter the list of variables by variable types. On the Options page, you can select whether only measurement variables are to be displayed in the variable list, or both measurement variables and parameters. The active filter setting is displayed in the header of the variable list.
- *Variable Properties*: You can view the properties of a variable. The information is displayed in a dialog which you can access via the context menu of the variable. The contents of the variable properties dialog depend on the type of the selected variable.

Selected Variables The Selected Variables list contains the variables you selected from the variable list for being uploaded from the ECU. The address, data type, conversion method from ECU representation (source value) to physical representation (converted value), variable description, and minimum and maximum values can be displayed for each variable in the Selected Variables list.

You can add custom variables to the Selected Variables list for being uploaded from the ECU via the Add Custom command. A custom variable has a *(custom)* declaration after its name.

You can add columns to the Selected Variables list or remove them from it via the Customize Columns command in the context menu of the list's column header. To add a column, drag the corresponding entry from the Customize Columns dialog to the column header. To remove a column, drag the corresponding column header entry from the Selected Variables list to the

Customize Columns dialog. By default, the **Selected Variables** list contains the **Variable**, **Address**, **Type**, **Convert**, and **Port Data Type** columns.

- The variable can be converted from source value to converted value if the computation method is supported by the RTI Bypass Blockset.
- Variable conversion is never performed for a conversion formula that is not supported by the RTI Bypass Blockset. The source value is used in the Simulink model, and the checkbox beside the conversion method is disabled. For information on the supported computation methods, refer to Limitations of the RTI Bypass Blockset on page 269.
- You can deactivate the conversion of a variable by clearing its **Convert** checkbox in the **Selected Variables** list. In that case the input corresponds 1:1 to the value read from the ECU.
- If you perform on-target bypassing and the TargetLink Code Generator is selected as the code generator on the Build Page of the corresponding Setup block, you can specify the port data type for a variable with deactivated conversion via its **Port Data Type** field. You can select between Double and the fixed-point data type specified for the variable in the A2L file. If variable conversion is enabled for the variable later on, the RTI Bypass Blockset automatically uses Double as the port data type for the variable, regardless of which port data type setting you have made.

  If you use Simulink Coder as the code generator, the port data type of a variable is always set to Double.

The variables will be output from the Upload block in the same order as they are displayed in the **Selected Variables** list. The variable names are used as port names. You can change the order of the variables and delete variables from the list.

Move Up Moves the currently selected variables further up the **Selected Variables** list. As a result, the corresponding outports of the Upload block also move up.

Move Down Moves the currently selected variables further down the **Selected Variables** list. As a result, the corresponding outports of the Upload block also move down.

Delete Deletes the currently selected variables from the **Selected Variables** list. The corresponding outports of the Upload block are also deleted.

Delete All Deletes all variables from the **Selected Variables** list. The corresponding outports of the Upload block are also deleted.

Add Custom Lets you add a custom variable for being uploaded from the ECU. The Add/Edit Custom Variables Properties dialog opens for you to specify the variable settings.

View Variable Properties (Available only from the context menu of a variable in the Selected Variables list added to the Variable Selector) Lets you view the variable properties of the selected variable. The Variable Properties dialog opens and displays the properties.

Edit/View Custom Variables Properties (Available only from the context menu of a custom variable in the Selected Variables list) Lets you view and edit the properties of the selected custom variable. The Add/Edit Custom Variables Properties dialog opens for you to view and change the settings.

Add/Edit Custom Variables Properties dialog

Name Lets you specify the name of the custom variable. The name you enter must start with a letter. Otherwise, it will be prefixed with '_' automatically. Only the following characters are allowed: [a-z], [A-Z], [0-9], '.', '_', '[', and ']'.

Display identifier Displays the display identifier that will be used to name the output ports of the block, when the **Use Display Identifier** checkbox is selected on the Variables Options page in the Setup block dialog. It contains the capitalized variable name.

Address Lets you specify the address of the custom variable.

Data type Lets you select a data type for the custom variable. If you change the data type, the bitmask and the limit properties are reset to the defaults for the selected data type.

Byte order Displays the byte order specified in the interface description ASAP2 file and imported in the Setup block dialog. It can be either Motorola format (big endian, MSB at the memory location with the lowest address) or Intel format (little endian, LSB at the memory location with the lowest address).

Description Lets you specify a description text for the custom variable.

Start bit Lets you enter a bit offset for the bit mask, counted from the right side of the bit mask.

Number of bits Lets you specify the bit mask width by entering the number of contiguous 1-bits.

Hex (Bit mask) Displays the bit mask value in hexadecimal notation.

The following example shows the result of different bit mask property settings:

Data Type	No. of Bits	Start Bit	Bit Mask	Hex
8-bit	4	0	00001111	F
8-bit	4	2	00111100	3C
16-bit	5	5	0000001111100000	3E0

Min weak (phys.) Lets you specify the lower weak limit of the physical value. You can only enter values that are inside the hard limits.

Max weak (phys.) Lets you specify the upper weak limit of the physical value. You can only enter values that are inside the hard limits. If not specified a default value is automatically generated.

Hex (Limits) Displays the source values in hexadecimal notations.

Note

The limits are automatically adapted if you perform one of the following actions:

- You select another data type.
- You select another computation method.

Computation method Lets you specify a computation method. A computation method is used to transform a source value into a converted value.

Select one of the following computation methods and specify the conversion rule:

- `LINEAR`: You can specify a linear function (by means of a limited `RAT_FUNC` method).
- `RAT_FUNC`: You can specify a fractional rational function. Only linear conversion is possible for variables accessed by an RTIBYPASS_READ_BLx block or RTIBYPASS_UPLOAD_BLx block.
- `TAB_INTP`: You can specify a numeric conversion table for tabular conversion with interpolation.
- `TAB_NOINTP`: You can specify a numeric conversion table for tabular conversion without interpolation.
- `FORM`: You can specify a formula to calculate the physical value from the source value, and/or an inverse formula to calculate the source value from the converted value.

There are some limitations for using computation methods. Refer to Limitations of the RTI Bypass Blockset on page 269.

Related topics

References

Block Description (RTIBYPASS_UPLOAD_BLx).. 99
Options Page (RTIBYPASS_UPLOAD_BLx).. 107
Unit Page (RTIBYPASS_UPLOAD_BLx).. 103

Options Page (RTIBYPASS_UPLOAD_BLx)

Purpose

To set up block ports and specify bypass interface-specific settings for the RTIBYPASS_UPLOAD_BLx block.

Dialog settings

Use 'Enable Service' Port Lets you enable the use of the Enable Service inport. Via the Enable Service port, you can enable or disable the execution of the RTIBYPASS_UPLOAD_BLx block. If the checkbox is selected, the Upload block gets an additional Enable Service inport. If the value fed to this Enable Service inport is equal to or less than 0, the variables are no longer uploaded from the ECU. The outputs of the Upload block keep their last values.

Use 'Status Port' Lets you enable the use of the Status Port outport. You can read status information via this port. The following Status Port outport values of the Upload block are available:

Status Port Outport Value	Description
0	New data uploaded
-1	Old data available
-4	No data available
-5	Invalid external variable configuration

Show port names Lets you select whether the port names are displayed at the inports and outports. The ports themselves are always displayed.

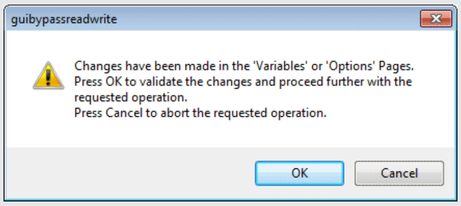
Tip

Deactivating this option reduces the size of the block.

Note

You cannot make changes to the port name settings and to the settings affecting the number of block ports at the same time. Instead, you have to apply the changes for one kind of setting first and then modify the other settings.

For example, if you add or delete variables to or from the Selected Variables list, or modify the Use 'Status Port' or Use 'Enable Service' Port options, the number of block ports is changed. Because of this, the Show port names option is grayed out until you apply your modifications for the port number by clicking OK or Apply. If you click on the grayed checkbox, a dialog similar to the following opens.



guibypassreadwrite

Changes have been made in the 'Variables' or 'Options' Pages.
Press OK to validate the changes and proceed further with the requested operation.
Press Cancel to abort the requested operation.

After you confirm the changes by clicking OK in the dialog, the settings are saved in the block parameters, and the Show port names option is no longer grayed out.

Set output ports signal labels Lets you specify to display the port names as signal labels for the outports.

If the Create unique signal names from RTIBypass ports option is activated on the Variables Options page in the Setup block (see Variables Options Page (RTIBYPASS_SETUP_BLx) on page 56), the automatically assigned signal names are extended to model-wide unique identifiers.

Filter 'Measurements' only Lets you select the variable types you want to filter the variable list on the Variables page by. If the checkbox is selected, only measurement variables are displayed in the variable list. If the checkbox is cleared, both measurement variables and parameters are displayed.

Fill 'Variable Selector' automatically Lets you select whether the variable selector on the Variables page is filled automatically each time you open the page.

Select variable(s) with 'RETURN' Lets you specify whether variables can be added to the Selected Variables list on the Variables page via the `Return` key. If the checkbox is activated, variables that are highlighted in the variable list can be added to the Selected Variables list by pressing `Return`, without having to activate their checkboxes first.

Clear Search Buffer after variable(s) selection Lets you select whether the search buffer is cleared automatically after you added variables from the variable list to the Selected Variables list on the Variables page via the `Return` key.

Enable Convert as Default Lets you specify the default setting for the Convert checkbox for variables that are added to the Selected Variables list on the Variables page. If enabled, the Convert checkbox is selected for a variable by default after the variable is added to the Selected Variables list, which means that variable conversion is activated for it. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

Use Port Data Type 'Double' as Default (Available only if TargetLink Code Generator is selected as the code generator) Lets you specify the default setting for the Port Data Type setting for variables that are added to the Selected Variables list on the Variables page. If the checkbox is selected, Double is used as the default port data type for a newly selected variable with disabled conversion. If the checkbox is cleared, the fixed-point data type specified for a variable in the A2L file is selected as the port data type by default. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

If you work with Simulink Coder as the code generator, the port data type for the selected variables is always set to Double regardless of whether variable conversion is enabled or disabled for the variables.

Enable external variable ports Lets you specify whether variables to be used with this block can be defined during run-time from outside the RTI Bypass Blockset. If the checkbox is selected, further Simulink block ports that allow you to configure variables by external block inputs are added to the Upload block. For information on the additional ports, refer to Block Description (RTIBYPASS_UPLOAD_BLx) on page 99.

Note

- The ExtVar Count, ExtVar Address and ExtVar Type input port values are applied only if the block is executed for the first time after the dSPACE real-time application was started, or if the block was disabled and is then enabled again via the Enable Service input port.

  Since the ExtVar input ports are applied and the block containing external variables is enabled only if the block is executed, it must be guaranteed that the block is executed even if it is placed in a subsystem triggered by an RTI Bypass Interrupt block. The RTI Bypass Blockset usually implicitly guarantees that a subsystem is triggered independently of the enable state of the RTI Bypass Read and Write blocks. Under certain conditions, however, an additional Read block without external variables must be added to ensure an initial subsystem trigger and continuous triggering if all blocks with external variables are disabled. For further information, refer to Block Description (RTIBYPASS_INTERRUPT_BLx) on page 92.
- Depending on the configuration type (foreground or background configuration), the availability of read or written values is delayed for one or more subsystem execution steps. You can get information on the availability state of read or written values via the optional Status Port output port. For further information, refer to Perform variable configuration in foreground on the block's Option page.

Maximum number of variables (Available only if Enable external variable ports is selected.) Lets you specify the maximum number of externally defined variables that can be used with this block.

The number of actually used variables is read by the ExtVar Count block inport. ExtVar Count can be in the range 0 ... 'Maximum number of variables'. If the specified number exceeds the upper limit, the value is reduced to 'Maximum number of variables'.

Enable block mode (Available only if Enable external variable ports is selected.) Lets you specify whether to use the block mode or single mode with this block. If the checkbox is selected, the block mode is enabled. With the block mode enabled, the ExtVar input ports allow several variables of the same data type to be read by just defining one start address for the whole data block and the number of values to be read. With the block mode disabled, the address, data type and value conversion must be defined separately for each variable.

Enable conversion port (Available only if Enable external variable ports is selected.) Lets you enable or disable the ExtVar Conversion block inport. If the checkbox is selected, the ExtVar Conversion inport is available and can be used to define the computation methods for converting values of the variables that are uploaded from the ECU with this block. If the checkbox is cleared, the ExtVar Conversion inport is not available and data conversion is disabled for the variables uploaded by this block.

Perform variable configuration in foreground (Available only if Enable external variable ports is selected.) Lets you enable or disable foreground reconfiguration of externally defined variables. If the checkbox is selected, the variables are reconfigured in the model foreground while the Upload block is executed. The changed variable values are immediately available. If the checkbox is cleared, variable reconfiguration is performed in the model background after the associated task is executed. The point in time the changed variable values are available is undefined, but not before the subsequent sampling step.

Via the optional Status Port outport of the Upload block, you can get information on whether no or new data is available, or whether the external variable definitions are invalid.

Configuration in the foreground can increase the task execution time, depending on the bypass interface used. Configuration in the model background does not affect the task execution time.

Tip

When working with externally defined variables, it is recommended to enable foreground reconfiguration for the RTIBYPASS_UPLOAD_BLx block.

Perform enable in foreground (Available only for external bypassing) Lets you specify whether enabling the block is processed in the foreground. If the checkbox is selected, the block is enabled in the model foreground. If the checkbox is cleared, enabling the block is performed in the model background.

Perform disable in foreground (Available only for external bypassing) Lets you specify whether disabling the block is processed in the foreground. If the checkbox is selected, the block is disabled in the model foreground. If the checkbox is cleared, disabling the block is performed in the model background.

Configure Opens a configuration dialog to specify bypass interface-specific settings for the RTIBYPASS_UPLOAD_BLx block. The dialog depends on the selected bypass interface type.

Note

Bypass interface-specific upload settings do not have to be configured for all bypass interfaces. Depending on the bypass interface you selected on the Unit page, this button may therefore be disabled.

For information on the bypass interface-specific configuration, refer to the relevant topic:

- XCP on CAN: Options Page (RTIBYPASS_UPLOAD_BLx for XCP on CAN) on page 162
- XCP on UDP/IP: Options Page (RTIBYPASS_UPLOAD_BLx for XCP on UDP/IP) on page 176
- XCP on FlexRay: Options Page (RTIBYPASS_UPLOAD_BLx for XCP on FlexRay) on page 193
- CCP: Options Page (RTIBYPASS_UPLOAD_BLx for CCP) on page 198
- INTERNAL: Options Page (RTIBYPASS_UPLOAD_BLx for INTERNAL) on page 223

Related topics

References

Block Description (RTIBYPASS_UPLOAD_BLx)........ 99
Unit Page (RTIBYPASS_UPLOAD_BLx)........ 103
Variables Page (RTIBYPASS_UPLOAD_BLx)........ 103

RTIBYPASS_DOWNLOAD_BLx

Purpose

To download variables to the ECU.

Where to go from here

Information in this section

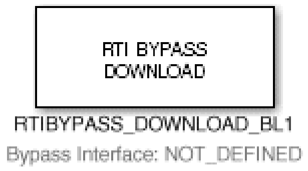
Information in other sections

Depending on the selected bypass interface, you have to specify interface-specific settings:

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_DOWNLOAD_BLx)

Block



Note

The RTIBYPASS_DOWNLOAD_BLx block has no name extension in the block library. When the block is copied to a Simulink model, the bypass interface name is added to the block name.

Purpose

To download variables to the ECU.

Description

The RTIBYPASS_DOWNLOAD_BLx block downloads ECU variables and parameters from the prototyping hardware to the ECU. A download is performed each time the RTIBYPASS_DOWNLOAD_BLx block is executed in the Simulink model.

Tip

Executing Upload and Download blocks increases the interrupt execution times. For XCP and CCP, you can configure to shift the writing of data to the model background. When the block is executed, the variables are buffered and will be written later on with low priority. This option prevents drastic increases in the interrupt execution times.

Note

The RTIBYPASS_DOWNLOAD_BLx block does not support writing bit variables for the CCP interface.

In this block, you must select the variables to be downloaded and specify the bypass interface to use for the download.

Note

The RTIBYPASS_DOWNLOAD_BLx block is supported by the following bypass interfaces:
- XCP on CAN
- XCP on UDP/IP
- XCP on FlexRay
- CCP
- INTERNAL
- VECU

Note

The RTIBYPASS_DOWNLOAD_BLx block neither provides real-time capabilities nor ensures data consistency for the transferred data. It is recommended to use the RTIBYPASS_WRITE_BLx block instead.

I/O characteristics

The table below shows the Simulink block input. The variable names are displayed at the inports only if the **Show port names** checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Input	Simulink Data Type	Description
<name of variable>	Double	Downloads the variable. The number of inports is equal to the number of variables you selected on the Variables Page (RTIBYPASS_DOWNLOAD_BLx).
Enable Service	Double	Lets you enable/disable the execution of the RTIBYPASS_DOWNLOAD_BLx block: ▪ > 0 : block is enabled ▪ ≤ 0 : block is disabled The port is available only if the **Use 'Enable Service' Port** checkbox is enabled on the Options Page (RTIBYPASS_DOWNLOAD_BLx). If the **Enable external variable ports** checkbox is selected on the Options Page (RTIBYPASS_DOWNLOAD_BLx), the **Enable Service** port is mandatory, since dynamic reconfiguration of externally defined variables during run time requires the Download block to be disabled during reconfiguration of the variable values, and then enabled again.
ExtVar Count	Double	Lets you define the number of externally defined variables actually used. The value must be in the range 0 ... **Maximum number of variables**. (The maximum number of variables is specified on the Options page.) For higher values, processing is reduced to the specified maximum number of variables. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.

Simulink Input	Simulink Data Type	Description
ExtVar Address	▪ Double (if block mode is enabled) ▪ Double array (if block mode is disabled)	Lets you define the address values of the externally defined variables. If the block mode is enabled on the Options page, the value indicates a single address specifying the start address for a whole data block. If the block mode is not enabled, an array of size 'Maximum number of variables' provides the address values for the single value accesses. (The maximum number of variables is specified on the Options page.) The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Type	▪ Double (if block mode is enabled) ▪ Double array (if block mode is disabled) The input values can have the following data types: ▪ 8: 8-bit unsigned ▪ 16: 16-bit unsigned ▪ 32: 32-bit unsigned ▪ -8: 8-bit signed ▪ -16: 16-bit signed ▪ -32: 32-bit signed ▪ -1032: float 32-bit ▪ 0: external variable disabled	Lets you define the data types for the externally defined variables. If the block mode is enabled on the Options page, a single value indicates the data type of the block members. If the block mode is not enabled, an array of size 'Maximum number of variables' provides the data type values for the single values. (The maximum number of variables is specified on the Options page.) External variables can be disabled selectively during run time. The data type value of the related input element must be set to 0 for this. The block must be disabled during the change. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.
ExtVar Conversion	▪ 1x6 double array (if block mode is enabled) ▪ nx6 double array, where n is the maximum number of variables (if block mode is disabled)	Lets you define the value conversion information for the externally defined variables. The conversion information is provided as a 1-dimensional array of 6 elements (if block mode is enabled) or of 'Maximum number of variables' x 6 elements (if block mode is disabled). The maximum number of variables is specified on the Options Page (RTIBYPASS_DOWNLOAD_BLx). Six coefficients a ... f are specified for each variable according to the fractional rational function $(a \cdot x^2 + b \cdot x + c) / (d \cdot x^2 + e \cdot x + f)$ used as the computation method for converting a source value into a converted value. **Note** ▪ The formula must always be defined in RCP to ECU direction, regardless of the block it is used with. ▪ Since the input is always a 1-dimensional array, it is recommended to use an S-function which integrates multidimensional data into a 1-dimensional array with this input port. For an implementation example, refer to the demo model in the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS` folder. You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows **Start** menu below **dSPACE RCP and HIL <version>**. The port is available only if the **Enable conversion port** checkbox is selected on the Options Page (RTIBYPASS_DOWNLOAD_BLx).
ExtVar Values	Double array	Defines the values of the externally defined variables. The values to be downloaded to the ECU are input as an array of size 'Maximum number of

Simulink Input	Simulink Data Type	Description
		variables'. The maximum number of variables is specified on the Options Page (RTIBYPASS_DOWNLOAD_BLx). Only the values from 0 ... **ExtVar Count** are downloaded. The other values remain unchanged until they are affected by a modified **ExtVar Count** input value. If value conversion is enabled for the variables, the values are converted before they are downloaded to the ECU. The port is available only if the **Enable external variable ports** checkbox is selected on the Options page.

Note

- The **ExtVar Count**, **ExtVar Address** and **ExtVar Type** input port values are applied only if the block is executed for the first time after the dSPACE real-time application was started, or if the block was disabled and is then enabled again via the **Enable Service** input port.

  Since the ExtVar input ports are applied and the block containing external variables is enabled only if the block is executed, it must be guaranteed that the block is executed even if it is placed in a subsystem triggered by an RTI Bypass Interrupt block. The RTI Bypass Blockset usually implicitly guarantees that a subsystem is triggered independently of the enable state of the RTI Bypass Read and Write blocks. Under certain conditions, however, an additional Read block without external variables must be added to ensure an initial subsystem trigger and continuous triggering if all blocks with external variables are disabled. For further information, refer to Block Description (RTIBYPASS_INTERRUPT_BLx) on page 92.
- Depending on the configuration type (foreground or background configuration), the availability of read or written values is delayed for one or more subsystem execution steps. You can get information on the availability state of read or written values via the optional **Status Port** output port. For further information, refer to **Perform variable configuration in foreground** on the block's Option page.

The table below shows the Simulink block output. The variable names are displayed at the outports only if the **Show port names** checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Output	Simulink Data Type	Description
Status Port (or other service-specific outports)	Double	Provides run-time information on the status of the block's current download operation: ▪ 0 = new data available (data downloaded) ▪ -1 = old data available (error during download) ▪ -4 = no data available (error during download) ▪ -5 = invalid external variable configuration The port is available only if the **Use 'Status Port'** checkbox is enabled on the Options Page (RTIBYPASS_DOWNLOAD_BLx).

Dialog pages

The dialog settings can be specified on the following pages:

- Unit Page (RTIBYPASS_DOWNLOAD_BLx) on page 118
- Variables Page (RTIBYPASS_DOWNLOAD_BLx) on page 119
- Options Page (RTIBYPASS_DOWNLOAD_BLx) on page 123

Note

A Download block needs exclusive access. For this reason, all the RTIBYPASS block dialogs (except for the dialogs of the RTIBYPASS_INFO_BLx and RTIBYPASS_CAL_PAGE_SWITCH_BLx blocks) must be closed if you want to open a Download block dialog, and cannot be opened until the Download block dialog is closed again.

Related topics

Basics

Overview of the RTI Bypass Blockset.. 15

References

RTIBYPASS_SETUP_BLx.. 51

Unit Page (RTIBYPASS_DOWNLOAD_BLx)

Purpose

To specify the bypass interface.

Dialog settings

Bypass interface Lets you select the bypass interface. All bypass interfaces you configured in your Simulink model via Setup blocks are listed.

Related topics

References

Block Description (RTIBYPASS_DOWNLOAD_BLx).. 114
Options Page (RTIBYPASS_DOWNLOAD_BLx).. 123
Variables Page (RTIBYPASS_DOWNLOAD_BLx).. 119

Variables Page (RTIBYPASS_DOWNLOAD_BLx)

Purpose

To view the available ECU variables (parameters and measurement variables), and to select the variables that should be downloaded to the ECU.

Dialog settings

Fill Variable Selector Fills the Variable Selector with the variables defined in the database files you imported for the selected bypass interface.

Note

Depending on the size of the database file, filling the Variable Selector can take some seconds. You should use this button only if variables are to be added to the list.

Variable Selector - Hierarchy tree In the hierarchy view mode, the structure of the imported variable description files is displayed in the hierarchy tree next to the variable list. The variable list shows the variables of the selected node and its subnodes. You can activate and deactivate the hierarchy view mode via the context menu of the variable list or hierarchy tree.

Variable Selector - Variable list The variable list of the Variable Selector displays the available ECU variables defined in the database files you imported for the selected bypass interface. Additional information on the ECU variables, for example, the description, memory address, and type, are also displayed.

If variables are to be downloaded to the ECU, you must add them from the variable list to the Selected Variables list. There are different ways of selecting variables:

- You can drag the variables from the variable list to the Selected Variables list.
- You can activate the checkboxes of the variables in the variable list using the mouse or via the **`Space`** key, and then press **`Return`** to add the checked variables to the Selected Variables list. When the variables are added to the Selected Variables list, their checkboxes are cleared. You can clear all checkboxes in the variable list without applying the variables to the Selected Variables list via the Uncheck All command, which is accessible via the context menu of a list entry.
- If the Select variable(s) with 'RETURN' checkbox is selected on the Options page, you can apply variables that are highlighted in the variable list to the Selected Variables list just by pressing the **`Return`** key without prior activation of their checkboxes.

Several methods of sorting, searching and filtering in the variable list are provided:

- *Sorting alphabetically*: You can sort the variable list alphabetically in ascending or descending order for a selected column. You can access the command by right-clicking the column header and choosing the function from the context menu.

- *Searching for a variable via buffered search*: You can search for a variable in the variable list using buffered search. When you type the first characters of one of its column entries, all keystrokes are stored in a buffer. The first item that matches the current keystroke sequence is selected. You can delete the whole keystroke sequence with `Delete`, but the selection remains. If the **Clear Search Buffer after variable(s) selection** option is selected on the Options page, the search buffer is cleared automatically after you select variables from the variable list. To differentiate between upper and lower case letters in your buffered search, open the context menu and select the **Case-Sensitive Search** command.
- *Filtering*: You can filter the list of variables by variable types. On the Options page, you can select whether only measurement variables are to be displayed in the variable list, or both measurement variables and parameters. The active filter setting is displayed in the header of the variable list.
- *Variable Properties*: You can view the properties of a variable. The information is displayed in a dialog which you can access via the context menu of the variable. The contents of the variable properties dialog depend on the type of the selected variable.

Selected Variables The **Selected Variables** list contains the variables you selected from the **Variable Selector** list for being downloaded to the ECU. The address, data type, conversion method from physical representation (converted value) to ECU representation (source value), integer conversion method, variable description, and minimum and maximum values can be displayed for each variable in the **Selected Variables** list.

You can add custom variables to the **Selected Variables** list for being downloaded to the ECU via the **Add Custom** command. A custom variable has a *(custom)* declaration after its name.

You can add columns to the **Selected Variables** list or remove them from it via the **Customize Columns** command in the context menu of the list's column header. To add a column, drag the corresponding entry from the **Customize Columns** dialog to the column header. To remove a column, drag the corresponding column header entry from the **Selected Variables** list to the **Customize Columns** dialog. By default, the **Selected Variables** list contains the **Variable**, **Address**, **Type**, **Convert**, and **Port Data Type** columns.

- The variable can be converted from converted value to source value if the computation method is supported by the RTI Bypass Blockset.
- Variable conversion is never performed for a conversion formula that is not supported by the RTI Bypass Blockset. The converted value is used in the ECU, and the checkbox beside the conversion method is disabled. For information on the supported computation methods, refer to Limitations of the RTI Bypass Blockset on page 269.
- You can deactivate the conversion of a variable by clearing its **Convert** checkbox in the **Selected Variables** list. In that case the input corresponds 1:1 to the value written to the ECU.

- If you perform on-target bypassing and the TargetLink Code Generator is selected as the code generator on the Build Page of the corresponding Setup block, you can specify the port data type for a variable with deactivated conversion via its **Port Data Type** field. You can select between Double and the fixed-point data type specified for the variable in the A2L file. If variable conversion is enabled for the variable later on, the RTI Bypass Blockset automatically uses Double as the port data type for the variable, regardless of which port data type setting you have made.

  If you use Simulink Coder as the code generator, the port data type of a variable is always set to Double.

Before a variable value is downloaded to the ECU, the value used by the bypass system must be converted to Integer data type. Conversion is done according to the following conversion method:

- For variables of unsigned data type:
  - convertedValue = int (unconvertedValue + offset)
- For variables of signed data type:
  - convertedValue = int (unconvertedValue + offset)

    if unconvertedValue ≥ 0
  - convertedValue = int (unconvertedValue - offset)

    if unconvertedValue < 0

You can configure the integer conversion method to be used by specifying the offset value which is responsible for rounding. Double-clicking the **Rounding Offset [0 … 1)** entry of the variable lets you enter a different offset value. Only offset values according to 0 ≤ offset < 1 are valid.

The variables will be input to the Download block in the same order as they are displayed in the **Selected Variables** list. The variable names are used as port names. You can change the order of the variables and delete variables from the list.

Move Up Moves the currently selected variables further up the **Selected Variables** list. As a result, the corresponding inports of the Download block also move up.

Move Down Moves the currently selected variables further down the **Selected Variables** list. As a result, the corresponding inports of the Download block also move down.

Delete Deletes the currently selected variables from the **Selected Variables** list. The corresponding inports of the Download block are also deleted.

Delete All Deletes all variables from the **Selected Variables** list. The corresponding inports of the Download block are also deleted.

Add Custom Lets you add a custom variable for being downloaded to the ECU. The Add/Edit Custom Variables Properties dialog opens for you to specify the variable settings.

View Variable Properties (Available only from the context menu of a variable in the Selected Variables list added to the Variable Selector) Lets you view the variable properties of the selected variable. The Variable Properties dialog opens and displays the properties.

Edit/View Custom Variables Properties (Available only from the context menu of a custom variable in the Selected Variables list) Lets you view and edit the properties of the selected custom variable. The Add/Edit Custom Variables Properties dialog opens for you to view and change the settings.

Add/Edit Custom Variables Properties dialog

Name Lets you specify the name of the custom variable. The name you enter must start with a letter. Otherwise, it will be prefixed with '_' automatically. Only the following characters are allowed: [a-z], [A-Z], [0-9], '.', '_', '[' and ']'.

Display identifier Displays the display identifier that will be used to name the output ports of the block, when the Use Display Identifier checkbox is selected on the Variables Options page in the Setup block dialog. It contains the capitalized variable name.

Address Lets you specify the address of the custom variable.

Data type Lets you select a data type for the custom variable. If you change the data type, the bitmask and the limit properties are reset to the defaults for the selected data type.

Byte order Displays the byte order specified in the interface description ASAP2 file and imported in the Setup block dialog. It can be either Motorola format (big endian, MSB at the memory location with the lowest address) or Intel format (little endian, LSB at the memory location with the lowest address).

Description Lets you specify a description text for the custom variable.

Start bit Lets you enter a bit offset for the bit mask, counted from the right side of the bit mask.

Number of bits Lets you specify the bit mask width by entering the number of contiguous 1-bits.

Hex (Bit mask) Displays the bit mask value in hexadecimal notation.

The following example shows the result of different bit mask property settings:

Data Type	No. of Bits	Start Bit	Bit Mask	Hex
8-bit	4	0	00001111	F
8-bit	4	2	00111100	3C
16-bit	5	5	0000001111100000	3E0

Min weak (phys.) Lets you specify the lower weak limit of the physical value. You can only enter values that are inside the hard limits.

Max weak (phys.) Lets you specify the upper weak limit of the physical value. You can only enter values that are inside the hard limits. If not specified a default value is automatically generated.

Hex (Limits) Displays the source values in hexadecimal notations.

Note

The limits are automatically adapted if you perform one of the following actions:

- You select another data type.
- You select another computation method.

Computation method Lets you specify a computation method. A computation method is used to transform a source value into a converted value.

Select one of the following computation methods and specify the conversion rule:

- `LINEAR`: You can specify a linear function.
- `RAT_FUNC`: You can specify a fractional rational function.
- `TAB_INTP`: You can specify a numeric conversion table for tabular conversion with interpolation.
- `TAB_NOINTP`: You can specify a numeric conversion table for tabular conversion without interpolation.
- `FORM`: You can specify a formula to calculate the physical value from the source value, and/or an inverse formula to calculate the source value from the converted value.

Related topics

References

Options Page (RTIBYPASS_DOWNLOAD_BLx)

Purpose

To set up block ports and specify bypass interface-specific settings for the RTIBYPASS_DOWNLOAD_BLx block.

Dialog settings

Use 'Enable Service' Port Lets you enable the use of the Enable Service inport. Via the Enable Service port, you can enable or disable the execution of the RTIBYPASS_DOWNLOAD_BLx block. If the checkbox is selected, the Download block gets an additional Enable Service inport. If the value fed to this

Enable Service inport is equal to or less than 0, the variables are no longer downloaded to the ECU.

Use 'Status Port' Lets you enable the use of the Status Port outport. You can read status information via this port. The following Status Port outport values of the Download block are available:

Status Port Outport Value	Description
0	New data downloaded
-1	Old data available
-4	No data available
-5	Invalid external variable configuration

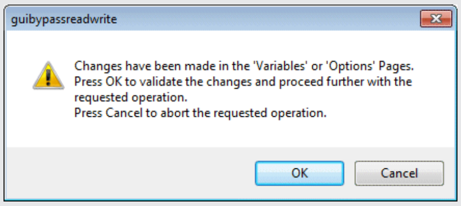
Show port names Lets you select whether the port names are displayed at the inports and outports. The ports themselves are always displayed.

Tip

Deactivating this option reduces the size of the block.

Note

You cannot make changes to the port name settings and to the settings affecting the number of block ports at the same time. Instead, you have to apply the changes for one kind of setting first and then modify the other settings.
For example, if you add or delete variables to or from the Selected Variables list, or modify the Use 'Status Port' or Use 'Enable Service' Port options, the number of block ports is changed. Because of this, the Show port names option is grayed out until you apply your modifications for the port number by clicking OK or Apply. If you click on the grayed checkbox, a dialog similar to the following opens.



Changes have been made in the 'Variables' or 'Options' Pages.
Press OK to validate the changes and proceed further with the requested operation.
Press Cancel to abort the requested operation.

After you confirm the changes by clicking OK in the dialog, the settings are saved in the block parameters, and the Show port names option is no longer grayed out.

Set output ports signal labels Lets you specify to display the port names as signal labels for the outports.

If the Create unique signal names from RTIBypass ports option is activated on the Variables Options page in the Setup block (see Variables Options Page

(RTIBYPASS_SETUP_BLx) on page 56), the automatically assigned signal names are extended to model-wide unique identifiers.

Filter 'Measurements' only Lets you select the variable types you want to filter the variable list on the Variables page by. If the checkbox is selected, only measurement variables are displayed in the variable list. If the checkbox is cleared, both measurement variables and parameters are displayed.

Fill 'Variable Selector' automatically Lets you select whether the variable selector on the Variables page is filled automatically each time you open the page.

Select variable(s) with 'RETURN' Lets you specify whether variables can be added to the Selected Variables list on the Variables page via the `Return` key. If the checkbox is activated, variables that are highlighted in the variable list can be added to the Selected Variables list by pressing `Return`, without having to activate their checkboxes first.

Clear Search Buffer after variable(s) selection Lets you select whether the search buffer is cleared automatically after you added variables from the variable list to the Selected Variables list on the Variables page via the `Return` key.

Enable Convert as Default Lets you specify the default setting for the Convert checkbox for variables that are added to the Selected Variables list on the Variables page. If enabled, the Convert checkbox is selected for a variable by default after the variable is added to the Selected Variables list, which means that variable conversion is activated for it. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

Use Port Data Type 'Double' as Default (Available only if TargetLink Code Generator is selected as the code generator) Lets you specify the default setting for the Port Data Type setting for variables that are added to the Selected Variables list on the Variables page. If the checkbox is selected, Double is used as the default port data type for a newly selected variable with disabled conversion. If the checkbox is cleared, the fixed-point data type specified for a variable in the A2L file is selected as the port data type by default. Switching this setting applies only to variables that will be added to the Selected Variables list later. It does not apply to the variables that are already selected.

If you work with Simulink Coder as the code generator, the port data type for the selected variables is always set to Double regardless of whether variable conversion is enabled or disabled for the variables.

Enable external variable ports Lets you specify whether variables to be used with this block can be defined during run-time from outside the RTI Bypass Blockset. If the checkbox is selected, further Simulink block ports that allow you to configure variables by external block inputs are added to the Download block. For information on the additional ports, refer to Block Description (RTIBYPASS_DOWNLOAD_BLx) on page 114.

Note

- The ExtVar Count, ExtVar Address and ExtVar Type input port values are applied only if the block is executed for the first time after the dSPACE real-time application was started, or if the block was disabled and is then enabled again via the Enable Service input port.
  Since the ExtVar input ports are applied and the block containing external variables is enabled only if the block is executed, it must be guaranteed that the block is executed even if it is placed in a subsystem triggered by an RTI Bypass Interrupt block. The RTI Bypass Blockset usually implicitly guarantees that a subsystem is triggered independently of the enable state of the RTI Bypass Read and Write blocks. Under certain conditions, however, an additional Read block without external variables must be added to ensure an initial subsystem trigger and continuous triggering if all blocks with external variables are disabled. For further information, refer to Block Description (RTIBYPASS_INTERRUPT_BLx) on page 92.
- Depending on the configuration type (foreground or background configuration), the availability of read or written values is delayed for one or more subsystem execution steps. You can get information on the availability state of read or written values via the optional Status Port output port. For further information, refer to Perform variable configuration in foreground on the block's Option page.

Maximum number of variables (Available only if Enable external variable ports is selected.) Lets you specify the maximum number of externally defined variables that can be used with this block.

The number of actually used variables is read by the ExtVar Count block inport. ExtVar Count can be in the range 0 ... 'Maximum number of variables'. If the specified number exceeds the upper limit, the value is reduced to 'Maximum number of variables'.

Enable block mode (Available only if Enable external variable ports is selected.) Lets you specify whether to use the block mode or single mode with this block. If the checkbox is selected, the block mode is enabled. With the block mode enabled, the ExtVar input ports allow several variables of the same data type to be read by just defining one start address for the whole data block and the number of values to be read. With the block mode disabled, the address, data type and value conversion must be defined separately for each variable.

Enable conversion port (Available only if Enable external variable ports is selected.) Lets you enable or disable the ExtVar Conversion block inport. If the checkbox is selected, the ExtVar Conversion inport is available and can be used to define the computation methods for converting values of the variables that are downloaded to the ECU with this block. If the checkbox is cleared, the ExtVar Conversion inport is not available and data conversion is disabled for the variables downloaded by this block.

Perform variable configuration in foreground (Available only if Enable external variable ports is selected.) Lets you enable or disable foreground reconfiguration of externally defined variables. If the checkbox is selected, the variables are reconfigured in the model foreground while the Download block is executed. The changed variable values are immediately written to the ECU. If the checkbox is cleared, variable reconfiguration is performed in the model background after the associated task is executed. The point in time the changed variable values are available in the ECU is undefined, but not before the subsequent sampling step.

Via the **Status Port** outport of the Download block, you can get information on whether no or new data is available, or whether the external variable definitions are invalid.

Configuration in the foreground can increase the task execution time, depending on the bypass interface used. Configuration in the model background does not affect the task execution time.

Tip

For working with externally defined variables, it is recommended to enable foreground reconfiguration for the RTIBYPASS_DOWNLOAD_BLx block.

Perform enable in foreground (Available only for external bypassing) Lets you specify whether enabling the block is processed in the foreground. If the checkbox is selected, the block is enabled in the model foreground. If the checkbox is cleared, enabling the block is performed in the model background.

Perform disable in foreground (Available only for external bypassing) Lets you specify whether disabling the block is processed in the foreground. If the checkbox is selected, the block is disabled in the model foreground. If the checkbox is cleared, disabling the block is performed in the model background.

Configure Opens a configuration dialog to specify bypass interface-specific settings for the RTIBYPASS_DOWNLOAD_BLx block. The dialog depends on the selected bypass interface type.

Note

Bypass interface-specific download settings do not have to be configured for all bypass interfaces. Depending on the bypass interface you selected on the Unit page, this button may therefore be disabled.

For information on the bypass interface-specific configuration, refer to the relevant topic:

- XCP on CAN: Options Page (RTIBYPASS_DOWNLOAD_BLx for XCP on CAN) on page 163
- XCP on UDP/IP: Options Page (RTIBYPASS_DOWNLOAD_BLx for XCP on UDP/IP) on page 176
- XCP on FlexRay: Options Page (RTIBYPASS_DOWNLOAD_BLx for XCP on FlexRay) on page 193
- CCP: Options Page (RTIBYPASS_DOWNLOAD_BLx for CCP) on page 198
- INTERNAL: Options Page (RTIBYPASS_DOWNLOAD_BLx for INTERNAL) on page 223

Related topics

References

RTIBYPASS_CAL_PAGE_SWITCH_BLx

Purpose

To read the active calibration page and switch between the calibration pages.

Where to go from here

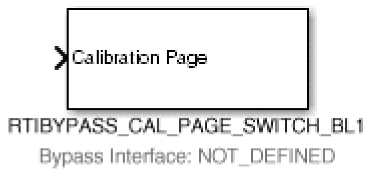
Information in this section

Information in other sections

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_CAL_PAGE_SWITCH_BLx)

Block



Note

The RTIBYPASS_CAL_PAGE_SWITCH_BLx block has no name extension in the block library. When the block is copied to a Simulink model, the bypass interface name is added to the block name.

Purpose

To read the active calibration page and switch between the calibration pages.

Description

The RTIBYPASS_CAL_PAGE_SWITCH_BLx block reads the currently active calibration page and lets you switch between the reference and the working page. Checking the currently active calibration page allows the block to recognize a page switch performed by the calibration tool. However, page switches executed by the RTI Bypass Blockset are not visible to other tools.

Note

The RTIBYPASS_CAL_PAGE_SWITCH_BLx block is supported by the following bypass interfaces:
- XCP on CAN
- XCP on UDP/IP
- XCP on FlexRay

I/O characteristics

The table below shows the Simulink block input:

Simulink Input	Simulink Data Type	Description
Calibration Page	Double	Lets you switch between the memory calibration pages. The value indicates which calibration page is currently active: ▪ < 0.5 : reference page is active ▪ ≥ 0.5 : working page is active The port is available only if the **Use 'Calibration Page' Port** checkbox is enabled on the Unit Page (RTIBYPASS_CAL_PAGE_SWITCH_BLx).
Enable Switching	Double	Lets you enable/disable the switching between the calibration pages. The port is available only if the **Use 'Enable Switching' Port** checkbox is enabled on the Unit Page (RTIBYPASS_CAL_PAGE_SWITCH_BLx).

The table below shows the Simulink block output:

Simulink Output	Simulink Data Type	Description
Page Status Port (or other service-specific outports)	Double	Provides run-time information on the currently active calibration page: ▪ -5 = page switching supported by the bypass interface but not by the ECU, since no calibration memory pages are defined in the A2L file ▪ -1 = page switching not possible, because the **Enable page switching** checkbox is not selected in the XCP-specific configuration dialog of the Setup block ▪ 0 … 255 = number of the active calibration page The port is available only if the **Use 'Page Status' Port** checkbox is enabled on the Unit Page (RTIBYPASS_CAL_PAGE_SWITCH_BLx).

To get the current page active, the **Memory segment number** selected in the XCP-specific configuration dialog of the Setup block is used. Refer to:
- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) on page 151
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) on page 167
- Options Page (RTIBYPASS_SETUP_BLx for XCP on FlexRay) on page 178

The working page number and reference page number that are specified in the selected memory segment are used when the page switching with the RTIBYPASS_CAL_PAGE_SWITCH_BLx block is performed.

When you switch between the calibration memory pages via the RTIBYPASS_CAL_PAGE_SWITCH_BLx block, the page switch takes place for all segments that are present in the block.

There are some limitations in connection with the RTIBYPASS_CAL_PAGE_SWITCH_BLx block. Refer to Limitations of the RTI Bypass Blockset on page 269.

Dialog pages

The dialog settings can be specified on the following pages:

- Unit Page (RTIBYPASS_CAL_PAGE_SWITCH_BLx) on page 131

Related topics

Basics

Overview of the RTI Bypass Blockset.. 15

Unit Page (RTIBYPASS_CAL_PAGE_SWITCH_BLx)

Purpose

To specify the bypass interface and block port settings.

Dialog settings

Bypass Interface Lets you select the bypass interface. All the bypass interfaces you configured in your Simulink model via Setup blocks are listed.

Use 'Calibration Page' Port Lets you enable the use of the Calibration Page inport. Via the Calibration Page inport, you can switch between the reference and the working page. If the value fed to this Calibration Page inport is < 0.5, the reference page is active. Input values ≥ 0.5 activate the working page.

Use 'Enable Switching' Port Lets you enable the use of the Enable Switching inport. Via the Enable Switching port, you can enable or disable page switching between the reference and working page via the RTI Bypass Blockset. If the checkbox is selected, the Page Switching block gets an additional Enable Switching inport.

Use 'Page Status' Port Lets you enable the use of the Page Status outport. You can read status information on the currently active calibration page via this port. The following Page Status outport values of the Page Switching block are available:

Page Status Outport Value	Description
-5	Calibration page switching not supported by the ECU, since no calibration memory pages are defined in the A2L file

Page Status Outport Value	Description
-1	Calibration page switching not possible, because the Enable page switching checkbox is not selected in the XCP-specific configuration dialog of the Setup block
0 … 255	Number of the currently active calibration page

Related topics

References

RTIBYPASS_INFO_BLx

Purpose

To list information concerning the RTI Bypass blocks in your Simulink model.

Where to go from here

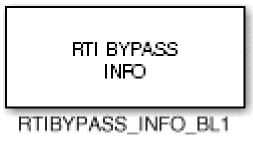
Information in this section

Information in other sections

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_INFO_BLx)

Block



Purpose

To display information concerning the model's RTI Bypass blocks.

Dialog pages

The information is displayed on the Info Page (RTIBYPASS_INFO_BLx) on page 134.

Related topics

Basics

Info Page (RTIBYPASS_INFO_BLx)

Purpose

To display information on the RTI Bypass blocks used in your Simulink model.

Dialog settings

Model summary Lists information concerning the model's blocks, as follows:

- Information on the bypass interface
  - Bypass interface name
  - Bypass interface type
- Information on the memory segments (available only in connection with bypass interfaces for on-target bypassing)
  - Segment name
  - Segment start address
  - Segment length (as hexadecimal value, and in kbit)
  - Memory type (RAM, flash)
  - Segment type (CODE, PARAMETER, VARIABLE)
  - Current fill level (as an absolute value in kbit, and as a relative value in %)
- Information on the Simulink parameters and Simulink signals from the generated on-target bypass code (available only in connection with bypass interfaces for on-target bypassing and only if Simulink® Coder™ is used for code generation)
  - Name and data type of each parameter
  - Name and data type of each signal
- Information on the board settings
  - Board type
  - Board number/module number
  - Channel number
  - Interface IP (available only for XCP on UDP/IP)
  - ECU IP (available only for XCP on UDP/IP)
- Information on the imported database files
  - Name, file path, and creation date of each imported A2L file
- Information on the Data Dictionary (DD) project file
  - File name
  - File path
  - Creation date and time

- Information on interrupt configuration
  - Subinterrupt numbers and service identifiers of the service instances used for interrupt generation (available only in connection with bypass interface types for service-based bypassing)
  - Subinterrupt numbers (available only in connection with bypass interfaces types for code-patch-based bypassing)
  - Hardware interrupt (available only in connection with bypass interface types for code-patch-based bypassing where only the hardware interrupt is used)
- Information on service instances (available only in connection with bypass interface types for service-based bypassing)
  - Service identifiers of the service instances that are used for reading or writing ECU variables
  - Number of bytes written by the RTIBYPASS_WRITE_BLx block or read by the RTIBYPASS_READ_Blx block for each service instance
  - Names of the service instances that are used for reading or writing ECU variables
  - Identifiers of the services the service instance is synchronized with (displayed in brackets)
- Information on settings made in the model's Read blocks (available only in connection with bypass interface types for service-based bypassing)

  The information that is displayed for each Read block depends on the interface type used:
  - Name of the Read block
  - Name and identifier of the service instance used for reading
  - Settings for the double buffer mechanism, wait mechanism, and failure checking mechanism (available only for DPMEM, XCP on CAN, and XCP on UDP/IP)
  - Packet identifier (PID) transmission settings (available only for XCP on CAN)
  - CAN ID when PID OFF (available only for XCP on CAN with dynamic DAQ lists)
  - DAQ list number and the corresponding CAN ID (available only for CCP and XCP on CAN with static DAQ lists)
  - DAQ list number (available only for XCP on UDP/IP)
  - DAQ list priority assignment type (available only for XCP interfaces)
  - Priority auto decrement setting (available only for XCP interfaces, and if 'Service instance' or 'Manual' is selected as the priority assignment type)
  - Manually assigned DAQ list priority (only available for XCP interfaces, and if 'Manual' is selected as the priority assignment type)
  - Configuration type (available only for XCP on FlexRay)
  - Selected FlexRay buffers for transmitting DAQ data from the ECU (available only for XCP on FlexRay, and if 'Select specific buffers, write configuration to ECU' is selected as the configuration type)
- Information on settings made in the model's Write blocks (available only in connection with bypass interface types for service-based bypassing)

  The information that is displayed for each Write block depends on the interface type used:
  - Name of the Write block
  - Name and identifier of the service instance used for writing
  - Settings for the double buffer mechanism, wait mechanism, and failure checking mechanism (available only for DPMEM, XCP on CAN, and XCP on UDP/IP)
  - Identifier of the service that is synchronized with the service instance assigned to the Write block (available only for DPMEM)
  - Packet identifier (PID) transmission settings (available only for XCP on CAN)
  - CAN ID when PID OFF (available only for XCP on CAN with dynamic DAQ lists)
  - DAQ list number and the corresponding CAN ID (available only for CCP and XCP on CAN with static DAQ lists)
  - DAQ list number (available only for XCP on UDP/IP)
  - DAQ list priority assignment type (available only for XCP interfaces)
  - Priority auto decrement setting (available only for XCP interfaces, and if 'Service instance' or 'Manual' is selected as the priority assignment type)
  - Manually assigned DAQ list priority (only available for XCP interfaces, and if 'Manual' is selected as the priority assignment type)
  - Configuration type (available only for XCP on FlexRay)
  - Selected FlexRay buffers for transmitting DAQ data from the ECU (available only for XCP on FlexRay, and if 'Select specific buffers, write configuration to ECU' or 'Select specific buffers' is selected as the configuration type)
  - Settings for the data transmission object (DTO) (available only for XCP on UDP/IP)
- Information on the ECU RAM/tool RAM needed for bypassing (available only in connection with bypass interface types for service-based bypassing)
  - Number of bytes needed for bypassing displayed as decimal and hexadecimal values (available only for DPMEM)

If an invalid configuration is detected, "(!)" is displayed and an error explanation appears.

Copy to Clipboard Copies all the model summary information listed to the Clipboard.

Refresh Refreshes the model summary information listed on the Info page.

Related topics

References

Block Description (RTIBYPASS_INFO_BLx)..133

RTIBYPASS_GATEWAY_BLx

Purpose

To transmit commands between up to two external calibration tools and the ECU, perform static data acquisition (DAQ) and data stimulation (STIM), and forward events generated by the ECU to the connected tools.

Where to go from here

Information in this section

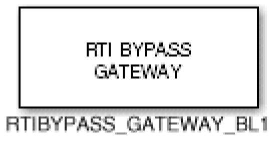
Information in other sections

You have to specify tool interface-specific and ECU interface-specific settings:

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_GATEWAY_BLx)

Block



Purpose

To forward messages from up to two external calibration tools to the ECU and forward the ECU responses back to the tools, to transfer DAQ and STIM data to and from the external tools, and to transfer events generated by the ECU to the connected tools.

Description

With the RTIBYPASS_GATEWAY_BLx block, the RTI Bypass Blockset enables arbitrated access to an ECU from a dSPACE prototyping system and up to two external calibration tools at the same time, even if only one service instance is implemented in the ECU code. This allows you, for example, to use ControlDesk and the RTI Bypass Blockset in parallel. The ControlDesk PC(s) and the ECU must be connected to different CAN channels of the RCP system for this.

Note

The RTIBYPASS_GATEWAY_BLx block is supported only by the XCP on CAN bypass interface.

Before you can use the XCP gateway functionality for ControlDesk, you must start the bypassing model on the RCP system. As long as the model is running, all XCP commands sent from ControlDesk are forwarded to the ECU and the ECU responses are forwarded back to ControlDesk.

Note

The XCP gateway functionality is available only for static DAQ lists, and if packet identifier (PID) transmission is enabled.

Dialog pages

The dialog settings can be specified on the following pages:

- Unit Page (RTIBYPASS_GATEWAY_BLx) on page 139

Note

If you want to open a Gateway block dialog, all RTIBYPASS_SETUP_BLx block dialogs must be closed.

Related topics

Basics

Overview of the RTI Bypass Blockset.. 15

Unit Page (RTIBYPASS_GATEWAY_BLx)

Purpose

To specify the gateway type, the interfaces to the external tools, and the ECU interface.

Dialog settings

Gateway type Lets you select the gateway type you want to use with the RTIBYPASS_GATEWAY_BLx block. Currently, only the XCP gateway is supported.

Tool 1 configuration – Board type or Bypass Interface Name Lets you select the board type or bypass interface name you use for connection of the first external calibration tool to the CAN bus.

- The available boards are:
  - DS4302, DS2202, DS2210 or DS2211 (in connection with a DS1006 or DS1007 processor board)
  - CAN Type 1 (in connection with a MicroAutoBox II)
- The list also contains the names of all the bypass interfaces whose board type is set to 'XCP Gateway' in the Setup block.

Note

- A bypass interface name can only be assigned to the first external tool.
- If the referenced bypass interface is to communicate via the gateway, you must select a bypass interface name.

- If tool 1 does not need to be routed to the ECU, select the '<not_defined>' value from the list. However, at least one tool interface must be configured.

Configure Opens a configuration dialog for specifying the interface settings for tool 1. The **Configure** button is available only if you selected a board type as the tool interface. If you selected a bypass interface name from the list, the settings from the corresponding Setup block are used and the **Configure** button is disabled. For information on the tool interface configuration, refer to Options Page (RTIBYPASS_GATEWAY_BLx for XCP on CAN) on page 163.

Tool 2 configuration – Board type Lets you select the board type you use for connection of the second external calibration tool to the CAN bus. The available boards are:

- DS4302, DS2202, DS2210 or DS2211 (in connection with a DS1006 or DS1007 processor board)
- CAN Type 1 (in connection with a MicroAutoBox II)

If tool 2 does not need to be routed to the ECU, select the '<not_defined>' value from the list. However, at least one tool interface must be configured.

Configure Opens a configuration dialog for specifying the interface settings for tool 2. For information on the tool interface configuration, refer to Options Page (RTIBYPASS_GATEWAY_BLx for XCP on CAN) on page 163.

ECU configuration – Board type Lets you select the board type you use for connection of the ECU to the CAN bus. The available boards are:

- DS4302, DS2202, DS2210 or DS2211 (in connection with a DS1006 or DS1007 processor board)
- CAN Type 1 (in connection with a MicroAutoBox II)

The ECU interface must be configured.

Configure Opens a configuration dialog for specifying the ECU interface. For information on the ECU interface configuration settings, refer to Options Page (RTIBYPASS_GATEWAY_BLx for XCP on CAN) on page 163.

Related topics

References

Block Description (RTIBYPASS_GATEWAY_BLx).. 137

RTIBYPASS_FUNCTION_BLx

Purpose

To add custom code to the bypass application or call an ECU-internal function from an on-target bypass model.

Where to go from here

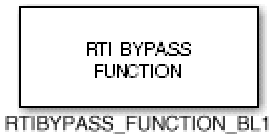
Information in this section

Information in other sections

For detailed information on the other blocks of the RTI Bypass Blockset, see:

Block Description (RTIBYPASS_FUNCTION_BLx)

Block



Note

The RTIBYPASS_FUNCTION_BLx block has no name extension in the block library. When the block is copied to a Simulink model and a function to be called by the Function block is selected in the block, the function name is added to the block name. If an ECU-internal function is selected, the name of the associated bypass interface is added to the block name, too.

Purpose

To add custom code to the bypass application, or call an ECU function that already exists in the ECU code in an on-target bypass model.

Description

The RTIBYPASS_FUNCTION_BLx block allows you to integrate custom C code. The block handles the selection of the C function to be called and lets you configure preprocessing settings for parsing the function from the added source files.

The RTIBYPASS_FUNCTION_BLx block also allows you to call an ECU-internal function, that is, an ECU function which already exists in the ECU code, in an on-target bypass model. After you add a bypass interface specified in an RTIBYPASS_SETUP_BLx block belonging to the model to the Function block, all the ECU-internal functions available for the bypass interface are offered for selection in the Function block.

The function input values and locally read global variables are represented by block inports. The call-by-reference function parameters, locally written global variables and the return value are represented by block outports. Additional global variables can be selected and configured as input and/or output values, which results in further block ports.

Tip

You can find examples for using the Function block in the demo model in the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS` folder.
You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.

I/O characteristics

The table below shows the Simulink block input. The variable names are displayed at the inports only if the Show port names checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Input	Simulink Data Type	Description
Enable Service	Double	Lets you enable/disable the execution of the RTIBYPASS_FUNCTION_BLx block: ▪ > 0 : block is enabled ▪ ≤ 0 : block is disabled

Simulink Input	Simulink Data Type	Description
		The port is available only if the **Use 'Enable Service' Port** checkbox is enabled on the Options Page (RTIBYPASS_FUNCTION_BLx).
<name of argument>	Double	Provides the function argument value or the value of the locally read global variable.

The table below shows the Simulink block output. The variable names are displayed at the outports only if the **Show port names** checkbox is enabled on the Options page. The ports themselves are always displayed.

Simulink Output	Simulink Data Type	Description
<name of argument>	Double	Provides the value of the passed call-by-reference argument or the value of the locally written global variable.
return	Double	Provides the return value of the called function.

Dialog pages

The dialog settings can be specified on the following pages:

- Variables Page (RTIBYPASS_FUNCTION_BLx) on page 143
- Options Page (RTIBYPASS_FUNCTION_BLx) on page 148

Related topics

Basics

Overview of the RTI Bypass Blockset.. 15

Variables Page (RTIBYPASS_FUNCTION_BLx)

Purpose

To select the C function or ECU-internal function and the global variables to be used with the Function block.

Dialog settings

Function - Hierarchy tree Displays the source files and A2L files already added to the model with the functions and global variable definitions they contain, and lets you select the function to be called by the Function block as well as global variables to be written/read before/after the function is executed.

The displayed source files and functions are available to each Function block in the model.

Note

All C-source files are added to and compiled during the build process of your model.

The related parameters and globally referenced variables for each function are displayed with a symbol indicating the access type:

Symbol	Description	Associated Function Block Port Type(s)
↱	Variable read by the function	Inport
↴	Variable written by the function	Outport
⇅	Variable read and written by the function	Inport and outport
!	Not used in the function	None

Parameters and variables of the following data types are supported:

- Simple data types
- Arrays (n-dimensional)
- Pointers (n-dimensional)

When you select a function to be used with the block, the parameters and locally referenced global variables are automatically selected, too. Parameters are mandatory and cannot be deselected as input ports. Global variables and the return value are optional, and therefore can be selected or deselected manually in the hierarchy tree. Each selected parameter and global variable is added to the Selected Variables list.

Note

Whether additional global A2L variables are used with a selected ECU-internal function depends on whether a `FUNCTION` element whose name matches the name of the ECU-internal function exists in one of the A2L files imported for the selected bypass interface. If so, the variables defined in the `FUNCTION` element are automatically selected in the tree. However, if an A2L variable is specified to be read *and* written in the ECU function (for example, via the `IN_MEASUREMENT` and `OUT_MEASUREMENT` attributes), it is selected as only one port type (input or output) in the Selected Variables list. You must select the other port manually.

There are several commands available via the context menu for handling source files and their functions:

- *Hierarchy View*: Lets you specify to display the functions and global variables in a hierarchy or flat view. If selected, the functions and global parameters are displayed hierarchically according to the source code file. If cleared, you get a flat view of the functions and global parameters of all source files.
- *Relative Paths*: Lets you specify whether the source files are set relative to the model root folder or with an absolute path.

  Tip

  Using a relative path makes it easy to move the model and all the files belonging to it to another location on your file system without having to change any path settings.

- *Expand All*: Lets you expand all the elements in the tree.

- *Collapse All*: Lets you collapse all the elements in the tree.
- *Update*: Lets you parse the currently selected source files and A2L files again and update the contents in the hierarchy tree and variables list.

Tip

As an alternative, you can also press `F5` to perform the update.

You can specify further settings for the selected function on the Options page.

Add - Source Files (*.c, *.h, *.a2l) (Also available via context menu) Lets you add further source files providing functions to be called to the model. A dialog opens for you to select the files to be added. The newly added source files are also displayed in the tree.

The RTI Bypass Blockset supports C-compliant source files (c99, gnu99, c89, gnu89, c94). You can add C, H and A2L files to the model. If you want to perform value conversion for input and output variables in your model, you must add the A2L file(s) associated to the C module to make the conversion information for the function's variables available.

Add - Setup Block (Also available via context menu) Lets you add a reference to a Setup block providing an internal bypass interface to an ECU application with built-in, callable ECU functions. A submenu with the bypass interface(s) specified for on-target bypassing in the Setup block(s) in your model is displayed. After you select a bypass interface from the submenu, the tree displays the bypass interface with its associated A2L files. The available ECU-internal functions and global variables contained in these A2L files are also displayed in the tree for selection.

Only one bypass interface can be used as the source for ECU-internal functions at a time. If you add a further bypass interface as a source for ECU-internal functions to the model, any previously added bypass interface and its A2L files are removed.

The available ECU-internal functions are defined in the `FUNCTION_PROTOTYPE` elements in the `IF_DATA dSPACE_INTERNAL_BYPASS` entry specified for the selected bypass interface.

The RTI Bypass Blockset supports ECU-internal functions that are compliant with the following embedded application binary interface (EABI) standards:

Target Processor Family	EABI Standard
TriCore	TriCore® 1 32-bit Unified Processor Core Embedded Applications Binary Interface (EABI)
MPC5xxx	PowerPC^{TM} e500 Application Binary Interface
RH850X	RH850 Compiler ABI Specification
ARM	Procedure Call Standard for the ARM Architecture (AAPCS)

Nevertheless, the RTI Bypass Blockset allows you to also use an ECU-internal function that is not compliant with these EABIs. The condition is that an appropriate proprietary adapter function is assigned to the noncompliant ECU function in its `FUNCTION_PROTOTYPE` element in the A2L file. The Function block calls the adapter function instead of the ECU-internal function. The adapter function calls the ECU-internal function and returns the ECU's return values, which are then provided at the Function block's outports. To make the adapter function available for the build process, it must be defined in a C source file, and this C file must be added to the model via the **Add - Source Files** command. For further information on adapter functions and their implementation, refer to Function Adaptation of Non-EABI-Compliant ECU Functions on page 244.

Remove (Also available via context menu) Lets you remove the currently selected source file(s) and/or bypass interface from the tree. The removed sources are no longer available for other Function blocks in the model. When you remove a bypass interface, all its associated A2L files are removed from the tree.

Selected Variables The **Selected Variables** list contains all the parameters and global variables that are currently selected in the functions tree. The selected variables are used as input and/or output values for the function call and represent the current port configuration. The **Selected Variables** list consists of several predefined columns. They display the symbol name and data type for each selected variable, let you specify the port configuration and allow you to enable/disable value conversion for variables that can be mapped to A2L variables.

There are some points to note when specifying the settings:

- Input/output ports:
  - The port configuration of a Function block is done via checkboxes. A selected **In** checkbox indicates that the variable will be input to the function. An inport is created for the Function block. A selected **Out** checkbox indicates that the variable will be output from the function. An outport is created for the Function block.
  - Depending on the access type, the **In** and **Out** checkboxes are preconfigured for each selected variable. If you want to change these settings, you must consider the following restrictions:
    - Function parameters are mandatory for calling the function. Since the inports must be available, you cannot clear the **In** checkboxes for function parameters.
    - Call-by-reference function parameters can also be selected as outports. However, you cannot select the **Out** checkboxes for simple data types.
    - The return value cannot be selected as inport.
    - Global variables with a const qualifier cannot be selected as inport, because they cannot be written.
  - The resulting inports and outports of the Function block are arranged in the same order as the variables are displayed in the **Selected Variables** list. You can change the order of the variables via the **Move Up** and **Move Down** buttons. The variable names can be used as port names.
- Value conversion:
  - The values of global variables that are mappable to A2L variables can be converted before/after their values are written to/read from the RCP system. To make value conversion available to a global parameter used with a C function, an appropriate A2L file which is associated to the C module must be added to the source files list. If a variable is to be converted from a source value to a converted value, the required conversion information is taken from the A2L file, where mapping is done via the symbol name. If the

A2L file does not contain conversion information for a selected variable, it cannot be converted.

- The variable can be converted from a source value to a converted value if the computation method is supported by the RTI Bypass Blockset. Variable conversion is never performed for a conversion formula that is not supported by the RTI Bypass Blockset. In this case, the **Conversion** checkbox is disabled. For information on the supported computation methods, refer to Limitations of the RTI Bypass Blockset on page 269.
- You can deactivate the conversion of a variable by clearing its **Conversion** checkbox in the **Selected Variables** list. In that case the input corresponds 1:1 to the value read from/written to the ECU/RCP memory.

Variable Properties (Available via context menu) Lets you open the **Variable Properties** dialog to view the properties of the selected variable. The contents of the dialog depend on the type of the selected variable.

Tip

For example, the computation method that is used to convert the variable's source value into a converted value is displayed in the variable properties.

Jump to Variable (Available via context menu) Lets you highlight all the occurrences of the selected variable in the hierarchy tree and variables list.

Move Up Moves the currently selected variable further up the **Selected Variables** list. As a result, the corresponding inport and/or outport of the Function block also move(s) up.

Move Down Moves the currently selected variable further down the **Selected Variables** list. As a result, the corresponding inport and/or outport of the Function block also move(s) down.

Function parameters and the function return value cannot be moved via the **Move Up** and **Move Down** buttons. They are always placed at the beginning and end of the **Selected Variables** list.

Related topics

References

Block Description (RTIBYPASS_FUNCTION_BLx).. 141
Options Page (RTIBYPASS_FUNCTION_BLx).. 148

Options Page (RTIBYPASS_FUNCTION_BLx)

Purpose

To set up block ports and specify preprocessing settings.

Dialog settings

Use 'Enable Service' Port Lets you enable the use of the Enable Service inport. Via the Enable Service port, you can enable or disable the execution of the RTIBYPASS_FUNCTION_BLx block. If the checkbox is selected, the Function block gets an additional Enable Service inport. If the value fed to this Enable Service inport is equal to or less than 0, the function specified in the block is no longer called. The outputs of the Function block keep their last values.

Show port names Lets you select whether the port names are displayed at the inports and outports. The ports themselves are always displayed.

Tip

Deactivating this option reduces the size of the block.

Set output ports signal labels Lets you select whether the port names are displayed as signal labels for the outports.

Defines Lists the configured preprocessing defines with their name value pairs, and lets you handle them as follows:

- To configure a new preprocessing define for parsing and compiling the source files, click Add. A new name value pair entry is made in the list. Change the define name and the value for the define as required. The define name must be unique.
- To modify the configuration of an existing preprocessing define, double-click its name or value entry in the Defines list, or select it from the list and select Edit from its context menu.
- To remove a define from the list of preprocessing defines, select it from the list and click Remove.

Includes Lists the folders selected for inclusion in source file preprocessing during parsing and compiling, and lets you handle them as follows:

- To add a further folder to the Includes list, click Add. A standard Browse For Folder dialog opens for you to select the folder to be included.
- To remove a folder from the Includes list, select it from the list and click Remove.

Options Lets you pass additional command line arguments to the parsing and build process of the corresponding C-modules.

Tip

Type `-help` to get a list of the available options.

Related topics

References

Block Description (RTIBYPASS_FUNCTION_BLx).......... 141
Variables Page (RTIBYPASS_FUNCTION_BLx).......... 143

Interface-Specific RTI Bypass Pages

Introduction

When using the RTI Bypass Blockset, you must specify some interface-specific configuration settings.

Where to go from here

Information in this section

RTI Bypass Pages for XCP on CAN

Introduction

The RTI Bypass Blockset supports service-based bypassing of ECU control functions via XCP on CAN (which implies CAN FD support), based on the dSPACE XCP Service. For XCP on CAN, the blockset can also be used as a gateway which allows CAN message routing between up to two external calibration tools and the ECU. When configuring the bypass hardware, bypass functions and the gateway via the RTI Bypass Blockset, you must specify XCP on CAN-specific configuration settings for some blocks.

Where to go from here

Information in this section

Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN)

Purpose

To configure XCP on CAN-specific settings for the Setup block.

Dialog settings

Board type Lets you select the board type of your bypass interface. The available boards are DS4302, DS2202, DS2210, DS2211, and DS4505 (in connection with a DS1006 or DS1007 processor board), and CAN Type 1 and IP Type 1 (in connection with a MicroAutoBox II).

If your model is to be used as an XCP gateway to transfer CAN messages between an external calibration tool (for example, ControlDesk) and the ECU, and if your interface is to communicate via the gateway, select 'XCP Gateway' as the board type. (The 'XCP Gateway' entry is available only if the XCP interface is configured for static DAQ in the ECU's A2L file.)

Board number (Not available if XCP Gateway is selected as the board type) Lets you select the board number of the bypass interface board in the range 1 ... 16. If your system contains several boards of the same type, RTI uses the board number to distinguish between them. If you use a MicroAutoBox II, this parameter is called the module number.

Module number (Available only if DS4505 or IP Type 1 is selected as the board type) Lets you select the number of the CAN FD interface module (1 ... 4) on the IP carrier board you want to use.

Module type (Available only if DS4505 or IP Type 1 is selected as the board type) Lets you select the type of the interface module used for CAN FD support.

Controller number (Not available if XCP Gateway is selected as the board type) Lets you select the number of the CAN controller you want to use. The range of the valid numbers depends on the selected board type:

Board	Controller Number
DS4302	1 ... 4
DS2202	1 ... 2
DS2210	1 ... 2
DS2211	1 ... 2
DS4505	1 ... 2
CAN Type 1	1 ... 2
IP Type 1	1 ... 2

Transceiver type (Not available if XCP Gateway is selected as the board type) Lets you select the transceiver type of your CAN controller board. The available transceiver types depend on the selected board type:

- The DS4302 supports the following transceiver types:
  - ISO11898 high-speed transceiver
  - RS485 transceiver
  - C252 fault-tolerant transceiver
- The DS2202, DS2210, DS2211, and the CAN Type 1 board support only the ISO11898 transceiver. You cannot change the setting.
- The DS4505 and the IP Type 1 board support the following transceiver types:
  - ISO11898_2
  - ISO11898_6

Termination (Not available if XCP Gateway is selected as the board type) Lets you specify the termination resistance for the CAN controller. The valid values depend on the selected board type and transceiver type.

Board/Module	Transceiver	Valid Value for Termination Resistance			
		Off	**120 Ω**	**1.6 kΩ**	**10 kΩ**
DS4302	ISO11898	Yes	Yes (default)	No	No
	RS485	Yes	Yes (default)	No	No
	C252	No	No	Yes (default)	Yes
DS2202	ISO11898	Yes	Yes (default)	No	No
DS2210	ISO11898	Yes	Yes (default)	No	No
DS2211	ISO11898	Yes	Yes (default)	No	No
DS4505/DS4342	ISO11898_2	Yes	Yes (default)	No	No
	ISO11898_6	Yes	Yes (default)	No	No
CAN Type 1	ISO11898	Yes (default)	No	No	No
IP Type 1/DS4342	ISO11898_2	Yes	Yes (default)	No	No
	ISO11898_6	Yes	Yes (default)	No	No

Inter-packet gap (Not available if XCP Gateway is selected as the board type) Lets you specify a time interval in s, used as a delay between two messages which are sent from the bypass system. Negative values are not allowed.

CAN FD support (Available only if DS4505 or IP Type 1 is selected as the board type) Lets you activate the CAN FD support and select the CAN FD protocol to be used to transmit and receive CAN FD messages. The following values are possible:

- OFF: CAN FD support is disabled. Only classic CAN messages are supported.
- NONE_ISO_CAN_FD: CAN FD support is enabled, and the non-ISO CAN FD protocol (which is Bosch's original CAN FD protocol) is used. Both messages in CAN format and in CAN FD format are supported.
- ISO_CAN_FD: CAN FD support is enabled, and the ISO CAN FD protocol (which is according to the ISO 11898-1:2015 standard) is used. Both messages in CAN format and in CAN FD format are supported.

Use additional DAQ lists to trigger interrupts always (Not available for static DAQ service configurations) Lets you enable or disable the usage of additional DAQ lists for interrupt triggering.

Note

The option is available only for dynamic DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 🕮).

Enable page switching Lets you enable or disable switching between the calibration memory pages.

Note

When a calibration memory page is switched via the RTI Bypass Blockset, a connected external calibration tool usually does not detect the page switch. This may lead to conflicts or unpredictable results, for example, when the external tool reads from or writes to the wrong memory page.

Memory segment number Lets you select the memory segment that contains the calibration memory page information from the memory segments list. The list shows all the memory segments defined in the ECU's A2L file that have at least one working page and one reference page. (The associated working page number and reference page number are used when a page switch is performed with the RTIBYPASS_CAL_PAGE_SWITCH_BLx block.) This edit field is enabled only if the Enable page switching checkbox is selected.

If an ECU provides memory pages containing calibratable parameters, the memory page information (such as the number of pages and their addresses and address extensions) is specified in the `MEMORY_SEGMENT` definition in the `IF_DATA` element of the ECU's A2L file.

Initial calibration page Lets you select the calibration memory page you want to be the active page when you start working. This edit field is enabled only if the Enable page switching checkbox is selected.

Page status request interval Lets you enter a time interval in s, used for periodic requesting of the current calibration page status from the Page Switching block. Negative values are not allowed. This edit field is enabled only if the Enable page switching checkbox is selected.

Seed and Key source or object file Lets you specify the Seed&Key algorithm that is to be used to control access to ECU resources for data acquisition, data stimulation, and calibration and page switching. The Seed&Key algorithm must be specified as a source file (C file) or object file (OBJ file). Click the Browse button to select the path and name of the Seed&Key source or object file.

Note

- One or more Seed&Key mechanisms can be supported in a Simulink model. If the mechanisms differ in implementation, the names of the C or OBJ files must also be different.
- A Seed&Key source or object file must meet the following conditions:
  - The file must have the same name as the function contained in it.
  - The file name must be C-compliant: for example, it must not contain a blank character.
  - The file name is case-sensitive, i.e., you must differentiate between upper and lower case letters.

Unlock DAQ Lets you unlock the ECU data acquisition (DAQ) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for data acquisition.

Unlock STIM Lets you unlock the ECU data stimulation (STIM) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for data stimulation.

Unlock CAL/PAG Lets you unlock the ECU calibration (CAL) and page switching (PAG) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for calibration and page switching.

Note

If the Simulink model needs one or more ECU resources which have not been unlocked here, the application will not run. An appropriate warning entry is shown in the Log Viewer.

ECU sends DAQ permanently Lets you enable or disable automatic ECU reinitialization if the ECU does not send DAQ lists continuously, for example, because of an engine stop. If the checkbox is selected, the ECU is reinitialized if DAQ data is missing. With this option disabled, the ECU will not be reinitialized if DAQ data is missing, although an XCP command-based alive check assumes that the ECU is not alive.

Ignore ECU resources lock status Lets you specify whether the lock statuses of the ECU resources are to be considered or ignored during ECU initialization. XCP resources can be locked by the Seed&Key mechanism. They must be unlocked by the RTI Bypass Blockset before being used. When a resource is unlocked, its lock status changes. To check whether the unlock operation was successful, the lock status of the resource is checked. This option allows you to ignore the lock statuses of XCP resources. With the checkbox selected, ECU initialization is not stopped even if ECU resources with locked statuses are

detected. This can be useful, for example, if the lock status of an XCP resource is set incorrectly.

Send Extended Identifier Bit (bit 31 is set in all CAN IDs) Lets you enable or disable the transmission of bit 31 when the extended CAN identifier format is used in XCP on CAN transport layer commands. If this checkbox is selected, bit 31 is sent to the ECU.

Note

When you use standard and extended identifiers in combination with the XCP service, the transmission of bit 31 in the XCP on CAN transport layer commands is mandatory.

Related topics

References

RTIBYPASS_CAL_PAGE_SWITCH_BLx....................129
Unit Page (RTIBYPASS_SETUP_BLx)....................53

Options Page (RTIBYPASS_READ_BLx for XCP on CAN)

Purpose

To configure XCP on CAN-specific settings for the Read block.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer is always available regardless of whether the `IF_DATA dSPACE_XCP` element is added in the A2L file. The double buffer mechanism ensures that only consistently transferred data blocks are used in the RTI Bypass Blockset (Read block) or the dSPACE XCP Service (Write block).

If this checkbox is cleared, the following items are disabled:

- The Enable wait mechanism checkbox
- The Timeout value edit field
- The Failure count value edit field

Enable wait mechanism Lets you select whether the wait mechanism is to be enabled or disabled. The wait mechanism is used to enable the prototyping system to wait for data from the ECU. If the wait mechanism is used, you must specify a timeout value. For further information on the wait mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Timeout value Lets you specify a timeout value for the wait mechanism in the range 0.000001 ... 10.0 s. This edit field is enabled only if the Enable double buffer mechanism checkbox and the Enable wait mechanism checkbox are selected.

Failure checking mechanism The failure checking mechanism counts the number of failed data transfers. If the failure checking mechanism is used, you must specify a failure count value as the limit. If the failure limit is exceeded, the prototyping system does not read any data from the ECU. If the optional Status Port outport is used, the outport will go to one of the following values:

Status Port Outport Value	Description
0	Data is copied
-1	Number of failures is between 1 and the specified failure count value
-2	Failure count value is exceeded, and no data is copied
-4	No data is available

Failure count value Lets you specify a failure count value for the failure checking mechanism in the range 0 ... 127. This edit field is enabled only if the Enable double buffer mechanism checkbox is selected.

Enable PID transmission Lets you enable or disable packet identifier (PID) transmission.

In "PID on" mode, the packet identifier of a CAN message is transmitted. In the "PID off" mode, the packet identifier of a CAN message is not transmitted. This increases the data throughput for the transmission of dynamic DAQ lists. The "PID off" mode has to be defined with the implementation of the dSPACE XCP Service. For further information on the PID transmission, refer to Measurement (DAQ) Features (XCP Feature Reference 📖).

CAN ID when PID OFF (0xXXXXXXXX) Lets you enter the CAN ID defined in the DAQ list. The edit field is available only if the PID transmission is disabled. The ID must be unique and hexadecimal.

Note

The option is available only for dynamic DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 📖).

DAQ list number Lets you select a number of a static DAQ list. The DAQ list number drop-down list contains the available DAQ list numbers found in the ECU´s database file (A2L file).

Note

The option is available only for static DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 🕮).

DAQ list CAN ID (0xXXXXXXXX) Lets you specify a CAN ID for the selected DAQ list. The ID must be unique and hexadecimal. If PID transmission is enabled, you can also select the 'NOT_DEFINED' entry.

Note

The option is available only for static DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 🕮).

Assignment type Lets you specify how the DAQ list priority is specified. You can select one of the following values:

Assignment Type Value	**Description**
Default (recommended setting)	To use internally defined standard priorities.
Service instance	To use the DAQ list priority according to the service instance selected in the block. The priority value is taken from the corresponding `EVENT` element description in the `IF_DATA XCP` or `IF_DATA XCPplus` section of the A2L file.
Manual	To let you specify the DAQ list priority manually in the Manually assigned priority [0xXX] edit field.

Manually assigned priority [0xXX] (Available only if Manual is selected as the assignment type.) Lets you enter the DAQ list priority as a hexadecimal value. 0xFF indicates the highest, 0x00 the lowest priority.

Tip

It is recommended to perform bypassing with the highest priority in the overall system.

Disable DAQ list priority auto decrement (Available only if Service instance or Manual is selected as the assignment type.) Lets you specify whether the DAQ list priority is decremented automatically. If the checkbox is cleared, the automatic priority decrement is enabled, that is, the last DAQ list assigned to the current service instance has automatically lower priority. This ensures that the DAQ list is the last to be sent. If the checkbox is selected, the automatic priority decrement is disabled, and all the DAQ lists have the same priority.

Note

To ensure consistent subinterrupt triggering for subsystems triggered by an Interrupt block which is related to the service instance that is selected in the current Read block, it is recommended not to disable the automatic DAQ list priority decrement.

Related topics

References

Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN)

Purpose

To configure XCP on CAN-specific settings for the Write block.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer mechanism ensures that only consistent transferred data blocks are used in the RTI Bypass Blockset (Read block) or the dSPACE XCP Service (Write block).

If this checkbox is cleared, the following items are disabled:

- The Enable wait mechanism checkbox
- The Timeout value edit field
- The Failure count value edit field

For further information on the double buffer mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Enable wait mechanism Lets you select whether the wait mechanism is to be enabled or disabled. The wait mechanism is used to enable the ECU to wait for a valid response from the prototyping system. If the wait mechanism is used, you must specify a timeout value.

Note

If the ECU receives data from the bypassing system after the timeout, the dSPACE XCP Service will use this data for the next bypassing cycle. This means that the dSPACE XCP Service will use wrong bypassing data in the next cycle. To avoid this, select a timeout value which is comparable to or larger than the maximum expected data transfer time.

For further information on the wait mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Timeout value Lets you specify a timeout value for the wait mechanism in the range 0.000001 ... 10.0 s. This edit field is enabled only if the Enable double buffer mechanism checkbox and the Enable wait mechanism checkbox are selected.

Failure checking mechanism The failure checking mechanism is enabled if the ECU supports it. The failure checking mechanism counts the number of failed data transfers. If the failure checking mechanism is used, you must specify a failure count value as the limit. If the failure limit is exceeded, the ECU service does not copy any data to the ECU. If the optional Status Port outport is used, the outport will go to one of the following values:

Status Port Outport Value	Description
0	Data is copied by the ECU
-1	Number of failures is between 1 and the specified failure count value
-2	Failure count value is exceeded, and no data is copied by the ECU
-4	No data is available

For further information on the failure checking mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Failure count value Lets you specify a failure count value for the failure checking mechanism in the range 0 ... 127. This edit field is enabled only if the Enable double buffer mechanism checkbox is selected and the ECU supports the failure checking mechanism.

Note

The following options are only available if you use the dSPACE XCP Service and if you have added the `IF_DATA dSPACE_XCP` element in the A2L file:
- Enable double buffer mechanism
- Enable wait mechanism
- Timeout value
- Failure checking mechanism
- Failure count value

Enable PID transmission Lets you enable or disable packet identifier (PID) transmission.

In "PID on" mode, the packet identifier of a CAN message is transmitted. In the "PID off" mode, the packet identifier of a CAN message is not transmitted. This increases the data throughput for the transmission of dynamic DAQ lists. The "PID off" mode has to be defined with the implementation of the dSPACE XCP

Service. For further information on the PID transmission, refer to Measurement (DAQ) Features (XCP Feature Reference 📖).

CAN ID when PID OFF (0xXXXXXXXX) Lets you enter the CAN ID defined in the DAQ list. The edit field is available only if the PID transmission is disabled. The ID must be unique and hexadecimal.

Note

The option is available only for dynamic DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 📖).

DAQ list number Lets you select the number of a static DAQ list. The DAQ list number drop-down list contains the available DAQ list numbers found in the ECU´s database file (A2L file).

Note

The option is available only for static DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 📖).

DAQ list CAN ID (0xXXXXXXXX) Lets you specify a CAN ID for the selected DAQ list. The ID must be unique and hexadecimal. If PID transmission is enabled, you can also select the 'NOT_DEFINED' entry.

Note

The option is available only for static DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 📖).

Assignment type Lets you specify how the DAQ list priority is specified. You can select one of the following values:

Assignment Type Value	Description
Default (recommended setting)	To use internally defined standard priorities.
Service instance	To use the DAQ list priority according to the service instance selected in the block. The priority value is taken from the corresponding `EVENT` element description in the `IF_DATA XCP` or `IF_DATA XCPplus` section of the A2L file.
Manual	To let you specify the DAQ list priority manually in the Manually assigned priority [0xXX] edit field.

Manually assigned priority [0xXX] (Available only if Manual is selected as the assignment type.) Lets you enter the DAQ list priority as a hexadecimal value. 0xFF indicates the highest, 0x00 the lowest priority.

Tip

It is recommended to perform bypassing with the highest priority in the overall system.

Disable DAQ list priority auto decrement (Available only if Service instance or Manual is selected as the assignment type.) Lets you specify whether the DAQ list priority is decremented automatically. If the checkbox is cleared, the automatic priority decrement is enabled, that is, the last DAQ list assigned to the current service instance has lower priority. This ensures that the DAQ list is the last to be sent. If the checkbox is selected, the automatic priority decrement is disabled, and all the DAQ lists have the same priority.

Related topics

References

Options Page (RTIBYPASS_WRITE_BLx).. 85

Options Page (RTIBYPASS_UPLOAD_BLx for XCP on CAN)

Purpose

To configure XCP on CAN-specific settings for the Upload block.

Dialog settings

Enable transfer in foreground Specifies foreground processing of the upload.

Note

An upload is always processed in the foreground. For this reason, the Enable transfer in foreground checkbox is selected and grayed out.

Related topics

References

Options Page (RTIBYPASS_UPLOAD_BLx)... 107

Options Page (RTIBYPASS_DOWNLOAD_BLx for XCP on CAN)

Purpose

To configure XCP on CAN-specific settings for the Download block.

Dialog settings

Enable transfer in foreground Lets you enable or disable foreground processing of the download. If the checkbox is selected, the selected variables are downloaded in a foreground task.

Note

Processing downloads in foreground will increase the task execution time of the task related to the subsystem the block is executed with. Depending on the amount of data transferred or possible communication time-outs on instable ECU connections, long task execution times may cause task overruns.

Related topics

References

Options Page (RTIBYPASS_DOWNLOAD_BLx).. 123

Options Page (RTIBYPASS_GATEWAY_BLx for XCP on CAN)

Purpose

To configure XCP on CAN-specific settings for the Gateway block regarding the calibration tool interfaces and the ECU interface.

Dialog settings

Board type Displays the board type you selected on the Unit page as the calibration tool interface or ECU interface.

Board number Lets you select the number of the board you use for connecting the tool or ECU to the CAN bus in the range 1 ... 16. If your system contains several boards of the same type, RTI uses the board number to distinguish between them. If you use a MicroAutoBox II, this parameter is called the module number.

Controller number Lets you select the number of the CAN controller you want to use to connect the calibration tool or ECU to the RCP system. You must specify different CAN controllers for the tool interfaces and the ECU interface. The range of the valid numbers depends on the selected board type:

Board	Controller Number
DS4302	1 ... 4
DS2202	1 ... 2
DS2210	1 ... 2
DS2211	1 ... 2
CAN Type 1	1 ... 2

Transceiver type Lets you select the transceiver type of your CAN controller board. The available transceiver types depend on the selected board type:

- The DS4302 supports the following transceiver types:
  - ISO11898 high-speed transceiver
  - RS485 transceiver
  - C252 fault-tolerant transceiver
- The DS2202, DS2210, DS2211 and the CAN Type 1 board support only the ISO11898 transceiver. In those cases, you cannot change the setting.

Termination resistance Lets you specify the termination resistance for the CAN controller, if possible. The valid values depend on the selected board type and transceiver type.

Board	Transceiver	Valid Value for Termination Resistance			
		Off	**120 Ω**	**1.6 kΩ**	**10 kΩ**
DS4302	ISO11898	Yes	Yes (default)	No	No
	RS485	Yes	Yes (default)	No	No
	C252	No	No	Yes (default)	Yes
DS2202	ISO11898	Yes	Yes (default)	No	No
DS2210	ISO11898	Yes	Yes (default)	No	No
DS2211	ISO11898	Yes	Yes (default)	No	No
CAN Type 1	ISO11898	Yes (default)	No	No	No

Inter-packet gap Lets you specify a time interval in s, used as a delay between two messages which are sent from the bypass system. Negative values are not allowed.

Use the same settings as in the Setup block Lets you specify whether to use the same CAN protocol settings as specified in the appropriate Setup block configured for XCP on CAN or to specify the CAN protocol settings manually on this page. If the checkbox is enabled, the CAN protocol settings are taken automatically from the database (A2L file) specified in the Setup block. If your model does not contain an appropriate Setup block, the checkbox cannot be selected.

Baudrate Lets you specify the baud rate of the CAN bus in kbit/s. If Use the same settings as in the Setup block is selected, the baud rate is read automatically from the `IF_DATA XCP` or `IF_DATA XCPplus` element in the A2L file and cannot be modified.

Master CAN ID (0xXXXXXXXX) Lets you specify the master CAN identifier. The calibration tool uses it to send its XCP commands. If Use the same settings as in the Setup block is selected, the identifier is read automatically from the `IF_DATA XCP` or `IF_DATA XCPplus` element in the A2L file and cannot be modified.

Slave CAN ID (0xXXXXXXXX) Lets you specify the slave CAN identifier for the ECU with XCP on CAN. The ECU with XCP on CAN sends its XCP responses to the calibration tool using this CAN identifier. If Use the same settings as in the Setup block is selected, the identifier is read automatically from the `IF_DATA XCP` or `IF_DATA XCPplus` element in the A2L file and cannot be modified.

Use maximum size (8) for master to slave CAN frames -> MAX_DLC_REQUIRED Lets you specify whether to use the maximum data byte number for the CAN messages. If the checkbox is selected, only messages of 8 data bytes are sent.

T1 (Available only for ECU interface configuration.) Lets you enter the timeout value T1. The value must be decimal.

T2 (Available only for ECU interface configuration.) Lets you specify the timeout value T2. The value must be decimal.

T3 (Available only for ECU interface configuration.) Lets you specify the timeout value T3. The value must be decimal.

T4 (Available only for ECU interface configuration.) Lets you specify the timeout value T4. The value must be decimal.

T5 (Available only for ECU interface configuration.) Lets you specify the timeout value T5. The value must be decimal.

If Use the same settings as in the Setup block is selected, the timeout values are read automatically from the `IF_DATA XCP` or `IF_DATA XCPplus` element in the A2L file and cannot be modified.

Block read splitting size (Available only for calibration tool interface configuration.) Lets you specify the number of data bytes to be read within one block read access in the range 1 ... 255. Larger XCP block sequences will be divided into several block sequences of this size rounded down to a full factor of MAX_CTO-1.

Note

For XCP on CAN, MAX_CTO has a value of 8.

Block write splitting size (Available only for calibration tool interface configuration.) Lets you specify the number of data bytes that are to be written within one block write access in the range 1 ... 255. Larger XCP block sequences will be divided into several block sequences of this size rounded down to a full factor of MAX_CTO-2.

Note

For XCP on CAN, MAX_CTO has a value of 8.

Enable 'Command Pending' event (Available only for calibration tool interface configuration.) Lets you configure the gateway block to request the calibration tool to restart timeout detection using the command pending event. If the gateway received a request from the calibration tool, but is not able to process the request to the ECU, it informs the calibration tool via the command pending event. If the calibration tool receives this information from the gateway, it does not repeat its request and restarts the timer used for timeout detection. As soon as the gateway is able to process the request, the pending command is performed to the ECU and the resulting positive or negative ECU response is forwarded to the causing calibration tool.

'Command Pending' event period (Available only for calibration tool interface configuration and if the command pending event is enabled.) Lets you enter the time period in s for the generation of the command pending event. It specifies the period for restarting timeout detection. The value must be lower than the timeout value T1 specified in the `IF_DATA XCP` or `IF_DATA XCPplus` element in the A2L file.

Related topics

References

Unit Page (RTIBYPASS_GATEWAY_BLx)... 139

RTI Bypass Pages for XCP on UDP/IP

Introduction

The RTI Bypass Blockset supports service-based bypassing of ECU control functions via XCP on UDP/IP, based on the dSPACE XCP Service. When configuring the bypass hardware and bypass functions via the RTI Bypass Blockset, you must specify XCP on UDP/IP-specific configuration settings for some blocks.

Where to go from here

Information in this section

Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP)

Purpose

To configure XCP on UDP/IP-specific settings for the Setup block.

Dialog settings

Board type Lets you select the board type of your bypass interface. The available boards are:

- DS4121 ETH (in connection with a DS1006 or DS1007 processor board)
- DS1007 ETH Type 1 (in connection with a DS1007 processor board)
- ECU Type 1 ETH (in connection with a MicroAutoBox II)
- ETH Type 1 (in connection with a MicroAutoBox II)

Board number Lets you select the board number of the bypass interface board in the range 1 ... 16. If your system contains several boards of the same type, RTI uses the board number to distinguish between them. If you use a MicroAutoBox II, this parameter is called the module number.

Channel number Lets you specify the channel number of the bypass interface board. The valid number range depends on the selected board type.

Board Type	Number Of Channels
ECU Type 1 ETH	1
ETH Type 1	1
DS1007 ETH Type 1	1
DS4121 ETH	2

Maximum bandwidth Lets you select the maximum bandwidth value, which determines the maximum Ethernet link speed.

Interface IP Lets you specify the IP address of the Ethernet interface used to connect the bypass system to the ECU. Specify a unique IP address in the same subnet as the **Target IP** address. To avoid communication conflicts, the IP address must not be used by any other participant that might be connected to the same network, for example, a second ECU that is connected via an Ethernet switch.

Tip

You can also configure local IP addresses globally for the entire model using MATLAB workspace variables. You have to specify the variables as arrays, for example:

```
DSPACE_RTIBYPASS_Config.LOCAL_IP{1,1,1} = '1.12.4.15';
DSPACE_RTIBYPASS_Config.LOCAL_IP{1,2,1} = '112.12.4.15';
% boardType: 1 - DS1401, 2 - DS4121
% boardNumber or moduleNumber: 1-16; ChannelNo: 1-2;
```

If you do so, the specified IP addresses are used for all the RTIBYPASS_SETUP_BLx blocks. On the Options page, the interface IP of the connected interface is displayed, but read-only.

Target IP Displays the IP address of the connected ECU as specified by the `ADDRESS` or `HOST_NAME` parameter in the A2L file. If the IP address is specified by the `HOST_NAME` parameter, it can be resolved via the `nslookup` command. You have to specify the ECU IP address manually when resolving is not possible.

Inter-packet gap Lets you specify a time interval in s, used as a delay between two messages which are sent from the bypass system. Negative values are not allowed.

Use additional DAQ lists to trigger interrupts always (Not available for static DAQ service configurations) Lets you enable or disable the usage of additional DAQ lists for interrupt triggering.

Note

The option is available only for dynamic DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 🕮).

Enable page switching Lets you enable or disable switching between the calibration memory pages.

Note

When a calibration memory page is switched via the RTI Bypass Blockset, a connected external calibration tool usually does not detect the page switch. This may lead to conflicts or unpredictable results, for example, when the external tool reads from or writes to the wrong memory page.

Memory segment number Lets you select the memory segment that contains the calibration memory page information from the memory segments list. The list shows all the memory segments defined in the ECU's A2L file that have at least one working page and one reference page. (The associated working page number and reference page number are used when a page switch is performed with the RTIBYPASS_CAL_PAGE_SWITCH_BLx block.) This edit field is enabled only if the Enable page switching checkbox is selected.

If an ECU provides memory pages containing calibratable parameters, the memory page information (such as the number of pages and their addresses and address extensions) is specified in the `MEMORY_SEGMENT` definition in the `IF_DATA` element of the ECU's A2L file.

Initial calibration page Lets you select the calibration memory page you want to be the active page when you start working. This edit field is enabled only if the Enable page switching checkbox is selected.

Page status request interval Lets you enter a time interval in s, used for periodic requesting of the current calibration page status from the Page Switching block. Negative values are not allowed. This edit field is enabled only if the Enable page switching checkbox is selected.

Seed and Key source or object file Lets you specify the Seed&Key algorithm that is to be used to control access to ECU resources for data acquisition, data stimulation, and calibration and page switching. The Seed&Key algorithm must be specified as a source file (C file) or object file (OBJ file). Click the **Browse** button to select the path and name of the Seed&Key source or object file.

Note

- One or more Seed&Key mechanisms can be supported in a Simulink model. If the mechanisms differ in implementation, the names of the C or OBJ files must also be different.
- A Seed&Key source or object file must meet the following conditions:
  - The file must have the same name as the function contained in it.
  - The file name must be C-compliant: for example, it must not contain a blank character.
  - The file name is case-sensitive, i.e., you must differentiate between upper and lower case letters.

Unlock DAQ Lets you unlock the ECU data acquisition (DAQ) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for data acquisition.

Unlock STIM Lets you unlock the ECU data stimulation (STIM) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for data stimulation.

Unlock CAL/PAG Lets you unlock the ECU calibration (CAL) and page switching (PAG) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for calibration and page switching.

Note

If the Simulink model needs one or more ECU resources which have not been unlocked here, the application will not run. An appropriate warning entry is shown in the Log Viewer.

ECU sends DAQ permanently Lets you enable or disable automatic ECU reinitialization if the ECU does not send DAQ lists continuously, for example, because of an engine stop. If the checkbox is selected, the ECU is reinitialized if DAQ data is missing. With this option disabled, the ECU will not be reinitialized if DAQ data is missing, although an XCP command-based alive check assumes that the ECU is not alive.

Ignore ECU resources lock status Lets you specify whether the lock statuses of the ECU resources are to be considered or ignored during ECU initialization. XCP resources can be locked by the Seed&Key mechanism. They must be unlocked by the RTI Bypass Blockset before being used. When a resource is unlocked, its lock status changes. To check whether the unlock operation was successful, the lock status of the resource is checked. This option allows you to ignore the lock statuses of XCP resources. With the checkbox selected, ECU initialization is not stopped even if ECU resources with locked statuses are

detected. This can be useful, for example, if the lock status of an XCP resource is set incorrectly.

Send one ODT per Ethernet frame Lets you specify whether only one object descriptor table (ODT) is sent in one Ethernet frame, or if several ODTs can be sampled before being sent in one Ethernet frame. With this option enabled, the Send DTO immediately option on the Options page of the Write blocks is selected and grayed out.

Related topics

References

Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP)

Purpose

To configure XCP on UDP/IP-specific settings for the Read block.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer is always available regardless of whether the `IF_DATA dSPACE_XCP` element is added in the A2L file. The double buffer mechanism ensures that only consistently transferred data blocks are used in the RTI Bypass Blockset (Read block) or the dSPACE XCP Service (Write block).

If this checkbox is cleared, the following items are disabled:

- The Enable wait mechanism checkbox
- The Timeout value edit field
- The Failure count value edit field

Enable wait mechanism Lets you select whether the wait mechanism is to be enabled or disabled. The wait mechanism is used to enable the prototyping system to wait for data from the ECU. If the wait mechanism is used, you must specify a timeout value. For further information on the wait mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Timeout value Lets you specify a timeout value for the wait mechanism in the range 0.000001 ... 10.0 s. This edit field is enabled only if the Enable double buffer mechanism checkbox and the Enable wait mechanism checkbox are selected.

Failure checking mechanism The failure checking mechanism counts the number of failed data transfers. If the failure checking mechanism is used, you must specify a failure count value as the limit. If the failure limit is exceeded, the prototyping system does not read any data from the ECU. If the optional Status Port outport is used, the outport will go to one of the following values:

Status Port Outport Value	Description
0	Data is copied
-1	Number of failures is between 1 and the specified failure count value
-2	Failure count value is exceeded, and no data is copied
-4	No data is available

Failure count value Lets you specify a failure count value for the failure checking mechanism in the range 0 ... 127. This edit field is enabled only if the Enable double buffer mechanism checkbox is selected.

DAQ list number Lets you select a number of a static DAQ list. The DAQ list number drop-down list contains the available DAQ list numbers found in the ECU´s database file (A2L file).

Note

The option is available only for static DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 📖).

Assignment type Lets you specify how the DAQ list priority is specified. You can select one of the following values:

Assignment Type Value	Description
Default (recommended setting)	To use internally defined standard priorities.
Service instance	To use the DAQ list priority according to the service instance selected in the block. The priority value is taken from the corresponding `EVENT` element description in the `IF_DATA XCP` or `IF_DATA XCPplus` section of the A2L file.
Manual	To let you specify the DAQ list priority manually in the Manually assigned priority [0xXX] edit field.

Manually assigned priority [0xXX] (Available only if Manual is selected as the assignment type.) Lets you enter the DAQ list priority as a hexadecimal value. 0xFF indicates the highest, 0x00 the lowest priority.

Tip

It is recommended to perform bypassing with the highest priority in the overall system.

Disable DAQ list priority auto decrement (Available only if Service instance or Manual is selected as the assignment type.) Lets you specify whether the DAQ list priority is decremented automatically. If the checkbox is cleared, the automatic priority decrement is enabled, that is, the last DAQ list assigned to the current service instance has automatically lower priority. This ensures that the DAQ list is the last to be sent. If the checkbox is selected, the automatic priority decrement is disabled, and all the DAQ lists have the same priority.

Note

To ensure consistent subinterrupt triggering for subsystems triggered by an Interrupt block which is related to the service instance that is selected in the current Read block, it is recommended not to disable the automatic DAQ list priority decrement.

Related topics

References

Options Page (RTIBYPASS_READ_BLx).. 69

Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP)

Purpose

To configure XCP on UDP/IP-specific settings for the Write block.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer mechanism ensures that only consistent transferred data blocks are used in the RTI Bypass Blockset (Read block) or the dSPACE XCP Service (Write block).

If this checkbox is cleared, the following items are disabled:

- The Enable wait mechanism checkbox
- The Timeout value edit field
- The Failure count value edit field

For further information on the double buffer mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 🕮).

Enable wait mechanism Lets you select whether the wait mechanism is to be enabled or disabled. The wait mechanism is used to enable the ECU to wait for a valid response from the prototyping system. If the wait mechanism is used, you must specify a timeout value.

Note

If the ECU receives data from the bypassing system after the timeout, the dSPACE XCP Service will use this data for the next bypassing cycle. This means that the dSPACE XCP Service will use wrong bypassing data in the next cycle. To avoid this, select a timeout value which is comparable to or larger than the maximum expected data transfer time.

For further information on the wait mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Timeout value Lets you specify a timeout value for the wait mechanism in the range 0.000001 ... 10.0 s. This edit field is enabled only if the Enable double buffer mechanism checkbox and the Enable wait mechanism checkbox are selected.

Failure checking mechanism The failure checking mechanism is enabled if the ECU supports it. The failure checking mechanism counts the number of failed data transfers. If the failure checking mechanism is used, you must specify a failure count value as the limit. If the failure limit is exceeded, the ECU service does not copy any data to the ECU. If the optional Status Port outport is used, the outport will go to one of the following values:

Status Port Outport Value	Description
0	Data is copied by the ECU
-1	Number of failures is between 1 and the specified failure count value
-2	Failure count value is exceeded, and no data is copied by the ECU
-4	No data is available

For further information on the failure checking mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Failure count value Lets you specify a failure count value for the failure checking mechanism in the range 0 ... 127. This edit field is enabled only if the Enable double buffer mechanism checkbox is selected and the ECU supports the failure checking mechanism.

Note

The following options are only available if you use the dSPACE XCP Service and if you have added the `IF_DATA dSPACE_XCP` element in the A2L file:

- Enable double buffer mechanism
- Enable wait mechanism
- Timeout value
- Failure checking mechanism
- Failure count value

DAQ list number Lets you select a number of a static DAQ list. The DAQ list number drop-down list contains the available DAQ list numbers found in the ECU´s database file (A2L file).

Note

The option is available only for static DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 📖).

Assignment type Lets you specify how the DAQ list priority is specified. You can select one of the following values:

Assignment Type Value	Description
Default (recommended setting)	To use internally defined standard priorities.
Service instance	To use the DAQ list priority according to the service instance selected in the block. The priority value is taken from the corresponding `EVENT` element description in the `IF_DATA XCP` or `IF_DATA XCPplus` section of the A2L file.
Manual	To let you specify the DAQ list priority manually in the Manually assigned priority [0xXX] edit field.

Manually assigned priority [0xXX] (Available only if Manual is selected as the assignment type.) Lets you enter the DAQ list priority as a hexadecimal value. 0xFF indicates the highest, 0x00 the lowest priority.

Tip

It is recommended to perform bypassing with the highest priority in the overall system.

Disable DAQ list priority auto decrement (Available only if Service instance or Manual is selected as the assignment type.) Lets you specify whether the DAQ list priority is decremented automatically. If the checkbox is cleared, the automatic priority decrement is enabled, that is, the last DAQ list assigned to the current service instance has lower priority. This ensures that the DAQ list is the last to be sent. If the checkbox is selected, the automatic priority decrement is disabled, and all the DAQ lists have the same priority.

Send DTO immediately Lets you specify whether the Data Transmission Object (DTO) is sent immediately or not. If disabled, the DTOs of all read blocks are sampled until they fill a communication frame or the last enabled read block of the configured service instance is executed. This allows the RTI Bypass Blockset to send several DTOs in one communication frame for performance reasons. If the option is enabled for a write block, the DTOs of that write block and of all previous write blocks are sent immediately after the current block is executed.

If you specified to send only one ODT per Ethernet frame in the Setup block, the Send DTO immediately option is selected and grayed out.

Related topics

References

Options Page (RTIBYPASS_WRITE_BLx)..85

Options Page (RTIBYPASS_UPLOAD_BLx for XCP on UDP/IP)

Purpose

To configure XCP on UDP/IP-specific settings for the Upload block.

Dialog settings

Enable transfer in foreground Specifies foreground processing of the upload.

Note

An upload is always processed in the foreground. For this reason, the Enable transfer in foreground checkbox is selected and grayed out.

Related topics

References

Options Page (RTIBYPASS_UPLOAD_BLx)...107

Options Page (RTIBYPASS_DOWNLOAD_BLx for XCP on UDP/IP)

Purpose

To configure XCP on UDP/IP-specific settings for the Download block.

Dialog settings

Enable transfer in foreground Lets you enable or disable foreground processing of the download. If the checkbox is selected, the selected variables are downloaded in a foreground task.

Note

Processing downloads in foreground will increase the task execution time of the task related to the subsystem the block is executed with. Depending on the amount of data transferred or possible communication time-outs on instable ECU connections, long task execution times may cause task overruns.

Related topics

References

RTI Bypass Pages for XCP on FlexRay

Introduction

The RTI Bypass Blockset supports service-based bypassing of ECU control functions via XCP on FlexRay. When configuring the bypass hardware and bypass functions via the RTI Bypass Blockset, you must specify XCP on FlexRay-specific configuration settings for some blocks.

Where to go from here

Information in this section

Options Page (RTIBYPASS_SETUP_BLx for XCP on FlexRay)

Purpose

To configure XCP on FlexRay-specific settings for the Setup block.

Dialog settings

Interface type Lets you select the type of your FlexRay interface. Currently, only the dSPACE FlexRay Configuration Package is supported for communication.

FLEXRAYCONFIG Update block Lets you select a RTIFLEXRAYCONFIG Update block contained in the model. On single processor systems, only one Update block is available and preselected automatically.

The RTIFLEXRAYCONFIG Update block was generated by the dSPACE FlexRay Configuration Tool and added to the RTI FlexRay Configuration Blockset model during the generation process, from where it was then copied to the RTI Bypass model.

ECU sends DAQ permanently Lets you enable or disable automatic ECU reinitialization if the ECU does not send DAQ lists continuously, for example, because of an engine stop. If the checkbox is selected, the ECU is reinitialized if DAQ data is missing. With this option disabled, the ECU will not be reinitialized if DAQ data is missing, although an XCP command-based alive check assumes that the ECU is not alive.

Ignore ECU resources lock status Lets you specify whether the lock statuses of the ECU resources are to be considered or ignored during ECU initialization. XCP resources can be locked by the Seed&Key mechanism. They must be unlocked by the RTI Bypass Blockset before being used. When a resource is unlocked, its lock status changes. To check whether the unlock operation was successful, the lock status of the resource is checked. This option allows you to ignore the lock statuses of XCP resources. With the checkbox selected, ECU initialization is not stopped even if ECU resources with locked statuses are

detected. This can be useful, for example, if the lock status of an XCP resource is set incorrectly.

Enable page switching Lets you enable or disable switching between the calibration memory pages.

Note

When a calibration memory page is switched via the RTI Bypass Blockset, a connected external calibration tool usually does not detect the page switch. This may lead to conflicts or unpredictable results, for example, when the external tool reads from or writes to the wrong memory page.

Memory segment number Lets you select the memory segment that contains the calibration memory page information from the memory segments list. The list shows all the memory segments defined in the ECU's A2L file that have at least one working page and one reference page. (The associated working page number and reference page number are used when a page switch is performed with the RTIBYPASS_CAL_PAGE_SWITCH_BLx block.) This edit field is enabled only if the Enable page switching checkbox is selected.

If an ECU provides memory pages containing calibratable parameters, the memory page information (such as the number of pages and their addresses and address extensions) is specified in the `MEMORY_SEGMENT` definition in the `IF_DATA` element of the ECU's A2L file.

Initial calibration page Lets you select the calibration memory page you want to be the active page when you start working. This edit field is enabled only if the Enable page switching checkbox is selected.

Page status request interval Lets you enter a time interval in s, used for periodic requesting of the current calibration page status from the Page Switching block. Negative values are not allowed. This edit field is enabled only if the Enable page switching checkbox is selected.

Seed and Key source or object file Lets you specify the Seed&Key algorithm that is to be used to control access to ECU resources for data acquisition, data stimulation, and calibration and page switching. The Seed&Key algorithm must be specified as a source file (C file) or object file (OBJ file). Click the Browse button to select the path and name of the Seed&Key source or object file.

Note

- One or more Seed&Key mechanisms can be supported in a Simulink model. If the mechanisms differ in implementation, the names of the C or OBJ files must also be different.
- A Seed&Key source or object file must meet the following conditions:
  - The file must have the same name as the function contained in it.
  - The file name must be C-compliant: for example, it must not contain a blank character.
  - The file name is case-sensitive, i.e., you must differentiate between upper and lower case letters.

Unlock DAQ Lets you unlock the ECU data acquisition (DAQ) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for data acquisition.

Unlock STIM Lets you unlock the ECU data stimulation (STIM) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for data stimulation.

Unlock CAL/PAG Lets you unlock the ECU calibration (CAL) and page switching (PAG) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for calibration and page switching.

Note

If the Simulink model needs one or more ECU resources which have not been unlocked here, the application will not run. An appropriate warning entry is shown in the Log Viewer.

Related topics

References

Buffers Configuration Page (RTIBYPASS_SETUP_BLx for XCP on FlexRay)................................181
RTIBYPASS_CAL_PAGE_SWITCH_BLx..129
Unit Page (RTIBYPASS_SETUP_BLx)...53

Buffers Configuration Page (RTIBYPASS_SETUP_BLx for XCP on FlexRay)

Purpose

To view and modify the FlexRay buffer configuration.

Dialog settings

Buffers for command processing Lets you specify which buffers are to be used for command processing. You can select one of the following options:

- *Use initial CMD and RES/ERR buffers only*: Commands to the ECU are sent by the initial and statically preconfigured CMD buffers only. Responses from the ECU are sent by the initial and statically preconfigured RES_ERR buffers only.
- *Use all available buffers*: All currently available FlexRay buffers of suitable length and with activated CMD or RES/ERR are used to send commands and receive responses.

Tip

Performance enhancements for XCP command processing can be achieved only if several FlexRay frames of a FlexRay communication cycle are used for XCP command and response processing. Several Com tasks must be created with the FlexRay Configuration Tool for this.

Configured FlexRay buffers list Displays the FlexRay buffers provided by the XCP slave that are currently configured to be used for ECU communication, with their configuration settings. Each configured buffer has a Data Link Layer Protocol Data Unit (LPDU) assignment to a specific frame defined by the RTI FlexRay Configuration Package configuration used. The LPDU describes the slot, cycle and FlexRay channel.

The following table lists the information that is shown for each configured buffer in the buffers list:

Column Header	Meaning
Type	Buffer type
No.	Buffer number
ID	Slot ID
BC	Base cycle
CR	Cycle repetition
Ch	Channel
Len	Length
Dir	Direction (from the XCP master's point of view): ▪ TX: data transmission from dSPACE prototyping system to ECU ▪ RX: data transmission from ECU to dSPACE prototyping system
Frame	Frame name

FlexRay buffers without LPDU assignment are not on the list. To get an overview of all the FlexRay buffers that are defined in the imported A2L files, click Configure buffers.

Configure buffers Lets you view and edit the configuration of the FlexRay buffers. The Buffers configuration dialog opens for you to view and edit the LPDU assignment of FlexRay buffers.

Buffers configuration dialog

Buffer list Displays all FlexRay buffers that are defined in the imported A2L files. Buffers with a valid LPDU assignment are displayed in black, buffers without an LPDU assignment are displayed in gray, buffers with an invalid LPDU assignment are displayed in red.

Assign LPDU(s) Lets you do the LPDU assignment for the currently selected FlexRay buffers. The Assign LPDU(s) dialog opens for you to select the XCP frames to be assigned to the buffers. If the currently selected buffers have a valid LPDU assignment, the Assign LPDU(s) dialog does not open, and a message is displayed.

Clear LPDU(s) assignment Lets you clear the LPDU assignment for the currently selected FlexRay buffers.

Mapping information Opens the LPDUs to XCP Buffers Mapping Information dialog for you to show which frames of your FlexRay network can be assigned to which of the currently selected FlexRay buffers. The mapping information can be used, for example, to investigate which frames match a specific XCP buffer or why a frame is not assignable.

Assign LPDU(s) dialog

List of available frames Displays the unassigned frames of your FlexRay network that are made available by the current RTIFLEXRAY Configuration Package configuration and that match the requirements of the currently selected buffers, and lets you select XCP FlexRay frames for LPDU assignment. The available frames are listed hierarchically in the XCP master node and in the nodes of the ECUs they belong to.

Checkboxes next to the frames let you select FlexRay frames for assignment to the selected buffers. An activated checkbox indicates that the frame is selected for assignment. Activating the checkbox next to the XCP master or an ECU node in the tree selects all the FlexRay frames that belong to that node. This means you can automatically select all the FlexRay frames belonging to a specific ECU for assignment to the selected buffers.

Frame properties Displays LPDU information on the currently selected frame. The following list contains explanations on some properties:

- *Possible direction*: Displays the possible data direction of the selected frame, as seen from the relevant communication node in each case. For the XCP master node, the possible direction is the direction seen from the prototyping system. For all other nodes, the possible direction is seen from the ECU's point of view.
- *Configured*: Displays whether the frame was configured for use in ECU communication in the FlexRay Configuration Tool. If it was, the direction it was configured for is displayed in parentheses.

Note

In principle, only frames that are listed in the XCP master node are configured. However, it is possible to select FlexRay frames from each node in the tree for assignment to buffers regardless of their Configured value. If a frame from a node other than the XCP master is selected for assignment, the resulting data direction for the XCP master will be handled implicitly.

Assign Performs the assignment of the selected FlexRay frames to the selected FlexRay buffers and closes the dialog.

Tip

You can let the RTI Bypass Blockset make the LPDU assignment for you. You have to perform the following steps for this:

- In the Buffers configuration dialog, select all the buffers that are to be assigned from the list.
  Multiple selection is possible by pressing `Ctrl` or `Shift` when selecting the buffers. `Ctrl+A` selects all buffers in a single step.
- Click Assign LPDU(s).
- In the Assign LPDU(s) dialog, select the checkbox next to the node of the ECU you want to communicate with.
- Click Assign.

The RTI Bypass Blockset makes as many LPDU assignments as possible for the selected buffers, according to the definitions in the A2L file.

Note

You should only make the LPDU assignment manually if you are familiar with the buffer configuration. A wrong LPDU assignment can prevent you from making subsequent LPDU assignments. If LPDU assignment is impossible for a buffer, you must check the previous LPDU assignments and reassign the affected buffers.

LPDUs to XCP Buffers Mapping Information dialog

Selected XCP buffer(s) Displays the buffer numbers of the XCP buffers you selected in the XCP Buffers Configuration dialog before you opened the LPDUs to XCP Buffers Mapping Information dialog.

Mapping information Displays all the XCP frames of your FlexRay network with their configurations, including the specification of the XCP buffers the frames are currently assigned to. The Assignable Buffer(s) column lists the XCP buffers each LPDU can be assigned to. If no assignable buffer was found, a brief description text is displayed.

To determine the matching buffers, RTI Bypass verifies the following buffer properties in the displayed order for each of the selected XCP buffers.

- Direction
- Channel
- Slot ID
- Base cycle
- Cycle repetition
- Length

As soon as an incompatible property is detected for an XCP buffer, the algorithm stops and changes to the next XCP buffer. Different text colors and background colors are used to indicate whether a check was successful or not:

Text Color	Background Color	Description
Gray	White	LPDU is not configured.
Green	White	LPDU can be assigned to one or more XCP buffers. All the buffer properties are compatible for the assignable buffer(s).
Red	White	LPDU cannot be assigned because a buffer property is mismatching for all the selected XCP buffers. The tested properties of the LPDU are marked by colored background. The remaining untested properties are displayed in red.
White	Green	Matching property
White	Red	First mismatching property

Related topics

References

Options Page (RTIBYPASS_SETUP_BLx for XCP on FlexRay)........ 178
Unit Page (RTIBYPASS_SETUP_BLx)........ 53

Options Page (RTIBYPASS_READ_BLx for XCP on FlexRay)

Purpose

To configure XCP on FlexRay-specific settings for the Read block.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer is always available regardless of whether the `IF_DATA dSPACE_XCP` element is added in the A2L file. The double buffer mechanism ensures that only consistently transferred data blocks are used in the RTI Bypass Blockset (Read block) or the dSPACE XCP Service (Write block).

If this checkbox is cleared, the following items are disabled:

- The Enable wait mechanism checkbox
- The Timeout value edit field
- The Failure count value edit field

Enable wait mechanism Lets you select whether the wait mechanism is to be enabled or disabled. The wait mechanism is used to enable the prototyping system to wait for data from the ECU. If the wait mechanism is used, you must specify a timeout value. For further information on the wait mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 🕮).

Timeout value Lets you specify a timeout value for the wait mechanism in the range 0.000001 ... 10.0 s. This edit field is enabled only if the Enable double buffer mechanism checkbox and the Enable wait mechanism checkbox are selected.

Failure checking mechanism The failure checking mechanism counts the number of failed data transfers. If the failure checking mechanism is used, you must specify a failure count value as the limit. If the failure limit is exceeded, the prototyping system does not read any data from the ECU. If the optional Status Port outport is used, the outport will go to one of the following values:

Status Port Outport Value	Description
0	Data is copied
-1	Number of failures is between 1 and the specified failure count value
-2	Failure count value is exceeded, and no data is copied
-4	No data is available

Failure count value Lets you specify a failure count value for the failure checking mechanism in the range 0 ... 127. This edit field is enabled only if the Enable double buffer mechanism checkbox is selected.

DAQ list number Lets you select a number of a static DAQ list. The DAQ list number drop-down list contains the available DAQ list numbers found in the ECU´s database file (A2L file).

Note

The option is available only for static DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 🕮).

Assignment type Lets you specify how the DAQ list priority is specified. You can select one of the following values:

Assignment Type Value	Description
Default (recommended setting)	To use internally defined standard priorities.
Service instance	To use the DAQ list priority according to the service instance selected in the block. The priority value is taken from the corresponding `EVENT` element description in the `IF_DATA XCP` or `IF_DATA XCPplus` section of the A2L file.
Manual	To let you specify the DAQ list priority manually in the Manually assigned priority [0xXX] edit field.

Manually assigned priority [0xXX] (Available only if Manual is selected as the assignment type.) Lets you enter the DAQ list priority as a hexadecimal value. 0xFF indicates the highest, 0x00 the lowest priority.

Tip

It is recommended to perform bypassing with the highest priority in the overall system.

Disable DAQ list priority auto decrement (Available only if Service instance or Manual is selected as the assignment type.) Lets you specify whether the DAQ list priority is decremented automatically. If the checkbox is cleared, the automatic priority decrement is enabled, that is, the last DAQ list assigned to the current service instance has automatically lower priority. This ensures that the DAQ list is the last to be sent. If the checkbox is selected, the automatic priority decrement is disabled, and all the DAQ lists have the same priority.

Note

To ensure consistent subinterrupt triggering for subsystems triggered by an Interrupt block which is related to the service instance that is selected in the current Read block, it is recommended not to disable the automatic DAQ list priority decrement.

Configuration type Lets you specify how FlexRay buffers are to be used for ECU communication via the Read block. You can select one of the following options:

- *Use all available buffers*: All currently available FlexRay buffers with RX frame direction configured in the Setup block are used to transmit DAQ data from the ECU.
- *Select specific buffers, write configuration to ECU*: Specific FlexRay buffers are selected for transmitting DAQ data from the ECU. The specified buffer configuration is written to the ECU.

Selected FlexRay buffers list Displays the FlexRay buffers that are currently selected for the Read block to transmit DAQ data from the ECU, with their configuration settings. If Use all available buffers is selected as the configuration type, no specific buffer selection is made and the list is empty.

Select buffers (Not available if Use all available buffers is selected as the configuration type.) Lets you specify the FlexRay buffers for transmitting DAQ data from the ECU. All the FlexRay buffers assigned to a FlexRay frame (LPDU) which is configured for RX data direction are available to be selected for the Read block. The Buffers configuration dialog opens for you to select the buffers and modify the LPDU assignment of buffers, if necessary.

Buffers configuration dialog

Buffer list Displays all the FlexRay buffers defined in the imported A2L files and their LPDU assignments. In the list, all the buffers currently specified for data transmission from the ECU with this Read block are selected in the column on the left. You can change the buffer selection for the Read block by selecting or clearing the checkboxes of the buffers. You can only select buffers that are assigned to a FlexRay frame (LPDU) which is configured for RX data direction.

Assign LPDU(s) Lets you make the LPDU assignment for the currently selected FlexRay buffers. The Assign LPDU(s) dialog opens for you to select the frames to be assigned to the buffers. If the currently selected buffers have a valid LPDU assignment, the Assign LPDU(s) dialog does not open, and a message is displayed.

Clear LPDU(s) assignment Lets you clear the LPDU assignment for the currently selected FlexRay buffers.

Assign LPDU(s) dialog

List of available frames Displays the unassigned frames of your FlexRay network that are made available by the current RTIFLEXRAY Configuration Package configuration and that match the requirements of the currently selected buffers, and lets you select XCP FlexRay frames for LPDU assignment. The available frames are listed hierarchically in the XCP master node and in the nodes of the ECUs they belong to.

Checkboxes next to the frames let you select FlexRay frames for assignment to the selected buffers. An activated checkbox indicates that the frame is selected for assignment. Activating the checkbox next to the XCP master or an ECU node in the tree selects all the FlexRay frames that belong to that node. This means you can automatically select all the FlexRay frames belonging to a specific ECU for assignment to the selected buffers.

Frame properties Displays LPDU information on the currently selected frame. The following list contains explanations on some properties:

- *Possible direction*: Displays the possible data direction of the selected frame, as seen from the relevant communication node in each case. For the XCP master node, the possible direction is the direction seen from the prototyping system. For all other nodes, the possible direction is seen from the ECU's point of view.
- *Configured*: Displays whether the frame was configured for use in ECU communication in the FlexRay Configuration Tool. If it was, the direction it was configured for is displayed in parentheses.

Note

In principle, only frames that are listed in the XCP master node are configured. However, it is possible to select FlexRay frames from each node in the tree for assignment to buffers regardless of their Configured value. If a frame from a node other than the XCP master is selected for assignment, the resulting data direction for the XCP master will be handled implicitly.

Assign Performs the assignment of the selected FlexRay frames to the selected FlexRay buffers and closes the dialog.

Tip

You can let the RTI Bypass Blockset make the LPDU assignment for you. You have to perform the following steps for this:

- In the Buffers configuration dialog, select all the buffers that are to be assigned from the list.
  Multiple selection is possible by pressing `Ctrl` or `Shift` when selecting the buffers. `Ctrl+A` selects all buffers in a single step.
- Click Assign LPDU(s).
- In the Assign LPDU(s) dialog, select the checkbox next to the node of the ECU you want to communicate with.
- Click Assign.

The RTI Bypass Blockset makes as many LPDU assignments as possible for the selected buffers, according to the definitions in the A2L file.

Note

You should only make the LPDU assignment manually if you are familiar with the buffer configuration. A wrong LPDU assignment can prevent you from making subsequent LPDU assignments. If LPDU assignment is impossible for a buffer, you must check the previous LPDU assignments and reassign the affected buffers.

Related topics

References

Options Page (RTIBYPASS_WRITE_BLx for XCP on FlexRay)

Purpose

To configure XCP on FlexRay-specific settings for the Write block.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer mechanism ensures that only consistent transferred data blocks are used in the RTI Bypass Blockset (Read block) or the dSPACE XCP Service (Write block).

If this checkbox is cleared, the following items are disabled:

- The Enable wait mechanism checkbox
- The Timeout value edit field
- The Failure count value edit field

For further information on the double buffer mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Enable wait mechanism Lets you select whether the wait mechanism is to be enabled or disabled. The wait mechanism is used to enable the ECU to wait for a valid response from the prototyping system. If the wait mechanism is used, you must specify a timeout value.

Note

If the ECU receives data from the bypassing system after the timeout, the dSPACE XCP Service will use this data for the next bypassing cycle. This means that the dSPACE XCP Service will use wrong bypassing data in the next cycle. To avoid this, select a timeout value which is comparable to or larger than the maximum expected data transfer time.

For further information on the wait mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Timeout value Lets you specify a timeout value for the wait mechanism in the range 0.000001 ... 10.0 s. This edit field is enabled only if the **Enable double buffer mechanism** checkbox and the **Enable wait mechanism** checkbox are selected.

Failure checking mechanism The failure checking mechanism is enabled if the ECU supports it. The failure checking mechanism counts the number of failed data transfers. If the failure checking mechanism is used, you must specify a failure count value as the limit. If the failure limit is exceeded, the ECU service does not copy any data to the ECU. If the optional **Status Port** outport is used, the outport will go to one of the following values:

Status Port Outport Value	Description
0	Data is copied by the ECU
-1	Number of failures is between 1 and the specified failure count value
-2	Failure count value is exceeded, and no data is copied by the ECU
-4	No data is available

For further information on the failure checking mechanism, refer to Bypassing (DAQ + STIM) Features (XCP Feature Reference 📖).

Failure count value Lets you specify a failure count value for the failure checking mechanism in the range 0 ... 127. This edit field is enabled only if the **Enable double buffer mechanism** checkbox is selected and the ECU supports the failure checking mechanism.

Note

Since the dSPACE XCP Service does not support FlexRay as transport layer, the following options are not available:

- Enable double buffer mechanism
- Enable wait mechanism
- Timeout value
- Failure checking mechanism
- Failure count value

DAQ list number Lets you select the number of a static DAQ list. The DAQ list number drop-down list contains the available DAQ list numbers found in the ECU´s database file (A2L file).

Note

The option is available only for static DAQ lists. For further information on configuring DAQ lists, refer to *Measurement setup via DAQ lists* in Measurement (DAQ) Features (XCP Feature Reference 🕮).

Assignment type Lets you specify how the DAQ list priority is specified. You can select one of the following values:

Assignment Type Value	**Description**
Default (recommended setting)	To use internally defined standard priorities.
Service instance	To use the DAQ list priority according to the service instance selected in the block. The priority value is taken from the corresponding `EVENT` element description in the `IF_DATA XCP` or `IF_DATA XCPplus` section of the A2L file.
Manual	To let you specify the DAQ list priority manually in the Manually assigned priority [0xXX] edit field.

Manually assigned priority [0xXX] (Available only if Manual is selected as the assignment type.) Lets you enter the DAQ list priority as a hexadecimal value. 0xFF indicates the highest, 0x00 the lowest priority.

Tip

It is recommended to perform bypassing with the highest priority in the overall system.

Disable DAQ list priority auto decrement (Available only if Service instance or Manual is selected as the assignment type.) Lets you specify whether the DAQ list priority is decremented automatically. If the checkbox is cleared, the automatic priority decrement is enabled, that is, the last DAQ list assigned to the current service instance has lower priority. This ensures that the

DAQ list is the last to be sent. If the checkbox is selected, the automatic priority decrement is disabled, and all the DAQ lists have the same priority.

Configuration type Lets you specify how FlexRay buffers are to be used for ECU communication via the Write block. You can select one of the following options:

- *Use all available buffers*: All the FlexRay buffers with TX frame direction configured in the Setup block which are currently available are used to send stimulation data to the ECU.
- *Select specific buffers, write configuration to ECU*: Specific FlexRay buffers are selected for transmitting stimulation data to the ECU. The specified buffer configuration is written to the ECU.
- *Select specific buffers*: Specific FlexRay buffers are selected for transmitting stimulation data to the ECU. The buffer configuration is not written to the ECU.

Selected FlexRay buffers list Displays the FlexRay buffers that are currently selected for the Write block to transmit stimulation data to the ECU, with their configuration settings. If **Use all available buffers** is selected as configuration type, no specific buffer selection is made and the list is empty.

Select buffers (Not available if **Use all available buffers** is selected as configuration type.) Lets you specify the FlexRay buffers for transmitting stimulation data to the ECU. All the FlexRay buffers assigned to a FlexRay frame (LPDU) which is configured for TX data direction are available to be selected for the Write block. The Buffers configuration dialog opens for you to select the buffers and modify the LPDU assignment of buffers, if necessary.

Buffers configuration dialog

Buffer list Displays all the FlexRay buffers defined in the imported A2L files and their LPDU assignments. In the list, all the buffers currently specified for being used for data transmission to the ECU with the Write block are selected in the column on the left. You can change the buffer selection for the Write block by selecting or clearing the checkboxes of the buffers. You can only select buffers that are assigned to a FlexRay frame (LPDU) which is configured for TX data direction.

Assign LPDU(s) Lets you make the LPDU assignment for the currently selected FlexRay buffers. The Assign LPDU(s) dialog opens for you to select the frames to be assigned to the buffers. If the currently selected buffers have a valid LPDU assignment, the Assign LPDU(s) dialog does not open, and a message is displayed.

Clear LPDU(s) assignment Lets you clear the LPDU assignment for the currently selected FlexRay buffers.

Assign LPDU(s) dialog

List of available frames Displays the unassigned frames of your FlexRay network that are made available by the current RTIFLEXRAY Configuration Package configuration and that match the requirements of the currently selected buffers, and lets you select XCP FlexRay frames for LPDU assignment. The available frames are listed hierarchically in the XCP master node and in the nodes of the ECUs they belong to.

Checkboxes next to the frames let you select FlexRay frames for assignment to the selected buffers. An activated checkbox indicates that the frame is selected for assignment. Activating the checkbox next to the XCP master or an ECU node in the tree selects all the FlexRay frames that belong to that node. This means you can automatically select all the FlexRay frames belonging to a specific ECU for assignment to the selected buffers.

Frame properties Displays LPDU information on the currently selected frame. The following list contains explanations on some properties:

- *Possible direction*: Displays the possible data direction of the selected frame, as seen from the relevant communication node in each case. For the XCP master node, the possible direction is the direction seen from the prototyping system. For all other nodes, the possible direction is seen from the ECU's point of view.
- *Configured*: Displays whether the frame was configured for use in ECU communication in the FlexRay Configuration Tool. If it was, the direction it was configured for is displayed in parentheses.

Note

In principle, only frames that are listed in the XCP master node are configured. However, it is possible to select FlexRay frames from each node in the tree for assignment to buffers regardless of their Configured value. If a frame from a node other than the XCP master is selected for assignment, the resulting data direction for the XCP master will be handled implicitly.

Assign Performs the assignment of the selected FlexRay frames to the selected FlexRay buffers and closes the dialog.

Tip

You can let the RTI Bypass Blockset make the LPDU assignment for you. You have to perform the following steps for this:

- In the Buffers configuration dialog, select all the buffers that are to be assigned from the list.
  Multiple selection is possible by pressing **`Ctrl`** or **`Shift`** when selecting the buffers. **`Ctrl+A`** selects all buffers in a single step.
- Click Assign LPDU(s).
- In the Assign LPDU(s) dialog, select the checkbox next to the node of the ECU you want to communicate with.
- Click Assign.

The RTI Bypass Blockset makes as many LPDU assignments as possible for the selected buffers, according to the definitions in the A2L file.

Note

You should only make the LPDU assignment manually if you are familiar with the buffer configuration. A wrong LPDU assignment can prevent you from making subsequent LPDU assignments. If LPDU assignment is impossible for a buffer, you must check the previous LPDU assignments and reassign the affected buffers.

Related topics

References

Options Page (RTIBYPASS_WRITE_BLx)..85

Options Page (RTIBYPASS_UPLOAD_BLx for XCP on FlexRay)

Purpose

To configure XCP on FlexRay-specific settings for the Upload block.

Dialog settings

Enable transfer in foreground Specifies foreground processing of the upload.

Note

An upload is always processed in the foreground. For this reason, the Enable transfer in foreground checkbox is selected and grayed out.

Related topics

References

Options Page (RTIBYPASS_UPLOAD_BLx)..107

Options Page (RTIBYPASS_DOWNLOAD_BLx for XCP on FlexRay)

Purpose

To configure XCP on FlexRay-specific settings for the Download block.

Dialog settings

Enable transfer in foreground Lets you enable or disable foreground processing of the download. If the checkbox is selected, the selected variables are downloaded in a foreground task.

Note

Processing downloads in foreground will increase the task execution time of the task related to the subsystem the block is executed with. Depending on the amount of data transferred or possible communication time-outs on instable ECU connections, long task execution times may cause task overruns.

Related topics

References

Options Page (RTIBYPASS_DOWNLOAD_BLx).. 123

RTI Bypass Pages for CCP

Introduction

The RTI Bypass Blockset supports service-based bypassing of ECU control functions via CCP, based on a CCP service. When configuring the bypass hardware and bypass functions via the RTI Bypass Blockset, you must specify CCP-specific configuration settings for some blocks.

Where to go from here

Information in this section

Options Page (RTIBYPASS_SETUP_BLx for CCP)

Purpose

To configure CCP-specific settings for the Setup block.

Dialog settings

Board type Lets you select the board type of your bypass interface. The available boards are DS4302, DS2202, DS2210 and DS2211 (in connection with a DS1006 or DS1007 processor board), and CAN Type 1 (in connection with a MicroAutoBox II).

Board number Lets you select the board number of the bypass interface board in the range 1 ... 16. If your system contains several boards of the same type, RTI uses the board number to distinguish between them. If you use a MicroAutoBox II, this parameter is called the module number.

Controller number Lets you select the number of the CAN controller you want to use. The range of the valid numbers depends on the selected board type:

Board	Controller Number
DS4302	1 ... 4
DS2202	1 … 2
DS2210	1 … 2
DS2211	1 … 2
CAN Type 1	1 ... 2

Transceiver type Lets you select the transceiver type of your CAN controller board. The available transceiver types depend on the selected board type:

- The DS4302 supports the following transceiver types:
  - ISO11898 high-speed transceiver
  - RS485 transceiver
  - C252 fault-tolerant transceiver
- The DS2202, DS2210, DS2211 and the CAN Type 1 board support only the ISO11898 transceiver. You cannot change the setting.

Termination resistance Lets you specify the termination resistance for the CAN controller. The valid values depend on the selected board type and transceiver type.

Board	Transceiver	Valid Value for Termination Resistance			
		Off	120 Ω	1.6 kΩ	10 kΩ
DS4302	ISO11898	Yes	Yes (default)	No	No
	RS485	Yes	Yes (default)	No	No
	C252	No	No	Yes (default)	Yes
DS2202	ISO11898	Yes	Yes (default)	No	No
DS2210	ISO11898	Yes	Yes (default)	No	No

Board	Transceiver	Valid Value for Termination Resistance			
		Off	**120 Ω**	**1.6 kΩ**	**10 kΩ**
DS2211	ISO11898	Yes	Yes (default)	No	No
CAN Type 1	ISO11898	Yes (default)	No	No	No

Inter-packet gap Lets you specify a time interval in s, used as a delay between two messages which are sent from the bypass system. Negative values are not allowed.

Seed and Key source or object file Lets you specify the Seed&Key algorithm that is to be used to control access to ECU resources for data acquisition and calibration. The Seed&Key algorithm must be specified as a source file (C file) or object file (OBJ file). Click the **Browse** button to select the path and name of the Seed&Key source or object file.

Note

- One or more Seed&Key mechanisms can be supported in a Simulink model. If the mechanisms differ in implementation, the names of the C or OBJ files must also be different.
- A Seed&Key source or object file must meet the following conditions:
  - The file must have the same name as the function contained in it.
  - The file name must be C-compliant. It must not contain a blank character, for example.
  - The file name is case-sensitive, i.e., you must differentiate between upper and lower case letters.

Unlock DAQ Lets you unlock the ECU data acquisition (DAQ) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for data acquisition.

Unlock CAL Lets you unlock the ECU calibration (CAL) resources. If the checkbox is selected, the specified Seed&Key mechanism controls access to the ECU resources for calibration.

Note

If the Simulink model needs one or more ECU resources which have not been unlocked here, the application will not run. An appropriate warning entry is shown in the Log Viewer.

ECU sends DAQ permanently Lets you enable or disable automatic ECU reinitialization if the ECU does not send DAQ lists continuously, for example, because of an engine stop. If the checkbox is selected, the ECU is reinitialized if DAQ data is missing. With this option disabled, the ECU will not be reinitialized if DAQ data is missing, although a CCP command-based alive check assumes that the ECU is not alive.

Ignore ECU resources lock status Lets you specify whether the lock statuses of the ECU resources are to be considered or ignored during ECU initialization. CCP resources can be locked by the Seed&Key mechanism. They must be unlocked by the RTI Bypass Blockset before being used. When a resource is unlocked, its lock status changes. To check whether the unlock operation was successful, the lock status of the resource is checked. This option allows you to ignore the lock statuses of CCP resources. With the checkbox selected, ECU initialization is not stopped even if ECU resources with locked statuses are detected. This can be useful, for example, if the lock status of a CCP resource is set incorrectly.

Use maximum size (8) for CAN messages (MAX_DLC) Lets you specify whether to use the maximum data byte number for the CAN messages. If the checkbox is selected, frames of 8 data bytes are always transmitted as defined in the specification of the CCP protocol. If the checkbox is cleared, only the actually used bandwidth is transmitted. This will decrease the transmission time of CCP packets but might be incompatible with some CCP service implementations.

CCP command timeout Lets you specify the CCP command timeout value for common CCP commands. The CCP protocol specification defines a default timeout value of 25 ms for common CCP commands. If this default value is not suitable, you can specify a different timeout.

Related topics

References

Unit Page (RTIBYPASS_SETUP_BLx).. 53

Options Page (RTIBYPASS_READ_BLx for CCP)

Purpose

To configure CCP-specific settings for the Read block.

Dialog settings

DAQ list number Lets you select a number of a static DAQ list. The DAQ list number drop-down list contains the available DAQ list numbers found in the ECU''s database file (A2L file).

DAQ list CAN ID [0xXXXXXXXX] Lets you specify a CAN ID for the selected DAQ list. The ID must be unique and hexadecimal.

Related topics

References

Options Page (RTIBYPASS_READ_BLx).. 69

Options Page (RTIBYPASS_UPLOAD_BLx for CCP)

Purpose

To configure CCP-specific settings for the Upload block.

Dialog settings

Enable transfer in foreground Specifies foreground processing of the upload.

Note

An upload is always processed in the foreground. For this reason, the Enable transfer in foreground checkbox is selected and grayed out.

Related topics

References

Options Page (RTIBYPASS_UPLOAD_BLx).. 107

Options Page (RTIBYPASS_DOWNLOAD_BLx for CCP)

Purpose

To configure CCP-specific settings for the Download block.

Dialog settings

Enable transfer in foreground Lets you enable or disable foreground processing of the download. If the checkbox is selected, the selected variables are downloaded in a foreground task.

Note

Processing downloads in foreground will increase the task execution time of the task related to the subsystem the block is executed with. Depending on the amount of data transferred or possible communication time-outs on instable ECU connections, long task execution times may cause task overruns.

Related topics

References

Options Page (RTIBYPASS_DOWNLOAD_BLx).. 123

RTI Bypass Pages for dSPACE on DPMEM

Introduction

The RTI Bypass Blockset supports service-based bypassing of ECU control functions via DPMEM (dual-port memory), based on the dSPACE Calibration and Bypassing Service. When configuring the bypass hardware and bypass functions via the RTI Bypass Blockset, you must specify DPMEM-specific configuration settings for some blocks.

Where to go from here

Information in this section

Options Page (RTIBYPASS_SETUP_BLx for dSPACE on DPMEM)

Purpose

To specify basic board settings for dSPACE on DPMEM.

Dialog settings

Board type Lets you select the board type of your bypass interface. The available boards are DS4121 (in connection with a DS1006 or DS1007 processor board) and ECU Type 1 (in connection with a MicroAutoBox II).

Board number Lets you select the board number of the bypass interface board in the range 1 ... 16. If your system contains several boards of the same type, RTI uses the board number to distinguish between them. If you use a MicroAutoBox II, this parameter is called the module number.

Channel number Lets you specify the channel number of the bypass interface board. The valid number range depends on the selected board type.

Related topics

References

Options Page (RTIBYPASS_READ_BLx for dSPACE on DPMEM)

Purpose

To enable or disable the double buffer mechanism, the wait mechanism and the failure checking mechanism.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer mechanism ensures that only consistent data blocks are transferred from the ECU to the prototyping system and vice versa.

If this checkbox is cleared, the following items are disabled:

- The Enable wait mechanism checkbox
- The Timeout value edit field
- The Failure count value edit field

For further information on the double buffer mechanism, refer to Bypassing Features (dSPACE Calibration and Bypassing Service Feature Reference 📖).

Enable wait mechanism Lets you select whether the wait mechanism should be enabled or disabled. The wait mechanism is used to enable the prototyping system to wait for a valid response from the ECU. If the wait mechanism is used, you must specify a timeout value. For further information on the wait mechanism, refer to Bypassing Features (dSPACE Calibration and Bypassing Service Feature Reference 📖).

Tip

If the Read block is placed in an interrupt-triggered subsystem triggered by the same service instance, the wait mechanism does not have to be enabled because the ECU service guarantees that the input data is copied before the interrupt is generated.

Timeout value Lets you specify a timeout value for the wait mechanism in the range 0.00001 ... 0.65535 s. This edit field is enabled only if the Enable double buffer mechanism checkbox and the Enable wait mechanism checkbox are selected.

Failure checking mechanism The failure checking mechanism is enabled if the ECU supports it. The failure checking mechanism counts the number of failed data transfers. If the failure checking mechanism is used, you must specify a failure count value as the limit. If the failure limit is exceeded, the prototyping system does not read any data from the ECU. If the optional **Status Port** outport is used, the outport will go to one of the following values:

Status Port Outport Value	Description
0	Data is copied
-1	Number of failures is between 1 and the specified failure count value
-2	Failure count value is exceeded, and no data is copied
-4	No data is available

For further information on the failure checking mechanism, refer to Bypassing Features (dSPACE Calibration and Bypassing Service Feature Reference 📖).

Failure count value Lets you specify a failure count value for the failure checking mechanism in the range 0 ... 127. This value specifies how often the Read block reads old data from the ECU before reading is deactivated and the Status port of the RTI Bypass Read block returns 'No data copied'.

This edit field is enabled only if the **Enable double buffer mechanism** checkbox is selected and the ECU supports the failure checking mechanism.

Related topics

References

Options Page (RTIBYPASS_READ_BLx)... 69

Options Page (RTIBYPASS_WRITE_BLx for dSPACE on DPMEM)

Purpose

To enable or disable the double buffer mechanism, the wait mechanism, the failure checking mechanism, and the buffer synchronization.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer mechanism ensures that only consistent data blocks are transferred from the ECU to the prototyping system and vice versa.

If this checkbox is cleared, the following items are disabled:

- The **Enable wait mechanism** checkbox
- The **Timeout value** edit field
- The **Failure count value** edit field
- The **Service Instance** list

For further information on the double buffer mechanism, refer to Bypassing Features (dSPACE Calibration and Bypassing Service Feature Reference 🕮).

Enable wait mechanism Lets you select whether the wait mechanism should be enabled or disabled. The wait mechanism is used to enable the ECU to wait for a valid response from the prototyping system. If the wait mechanism is used, you must specify a timeout value. For further information on the wait mechanism, refer to Bypassing Features (dSPACE Calibration and Bypassing Service Feature Reference 🕮).

Timeout value Lets you specify a timeout value for the wait mechanism in the range 0.00001 ... 0.65535 s. This edit field is enabled only if the Enable double buffer mechanism checkbox and the Enable wait mechanism checkbox are selected.

Failure checking mechanism The failure checking mechanism is enabled if the ECU supports it. The failure checking mechanism counts the number of failed data transfers. If the failure checking mechanism is used, you must specify a failure count value as the limit. If the failure limit is exceeded, the ECU service does not copy any data to the ECU. If the optional Status Port outport is used, the outport will go to one of the following values:

Status Port Outport Value	Description
0	Data is copied
-1	Number of failures is between 1 and the specified failure count value
-2	Failure count value is exceeded, and no data is copied
-4	No data is available

For further information on the failure checking mechanism, refer to Bypassing Features (dSPACE Calibration and Bypassing Service Feature Reference 🕮).

Failure count value Lets you specify a failure count value for the failure checking mechanism in the range 0 ... 127. This edit field is enabled only if the Enable double buffer mechanism checkbox is selected and the ECU supports the failure checking mechanism.

Service Instance Lets you select the service instance of the Read block to enable buffer synchronization. The Service Instance drop-down list contains the available service instances, determined by the selected service instance on the Unit page of the Read block.

The prototyping system reads a trigger counter during execution of a Read block and writes the trigger counter back to the ECU together with the data block during execution of the Write block. You can also synchronize more than one Write block using the same service instance.

With buffer synchronization enabled, delayed results from the previous sampling step are recognized as old data. This avoids results of the previous sampling step being erroneously interpreted as results of the current sampling step.

Note

The precondition for buffer synchronization is that the service instance is used by at least one Read block.

Service ID Displays the service ID associated with the selected service instance. The service IDs are unique, and each ID corresponds to one service instance.

Related topics

References

Options Page (RTIBYPASS_WRITE_BLx)..85

RTI Bypass Pages for Internal

Introduction

The RTI Bypass Blockset supports on-target, service-based bypassing of ECU control functions, based on the dSPACE Internal Bypassing Service. When configuring the target ECU hardware and bypass functions via the RTI Bypass Blockset, you must specify internal-bypass-specific configuration settings for some blocks.

Where to go from here

Information in this section

Build Page (RTIBYPASS_SETUP_BLx for INTERNAL)

Purpose

To select the code generator and compiler to be used, and to specify settings concerning A2L file generation, reusing existing ECU variables by the on-target bypass model, merging the ECU application, postprocessing, and flashing the ECU application.

Dialog settings

Code Generator Lets you select the code generator to be used for model code generation.

Compiler Lets you select the compiler to be used for the target-specific build process. The drop-down list contains all the compilers available for the selected code generator and the target that are provided by dSPACE (default compilers), all automatically detected Green Hills compilers that were installed separately (scanned compilers), and any valid user-defined compilers that were specified manually for the target in the Compiler Configurations dialog.

Note

Currently, the following compilers are supported:

- Default compilers:
  - HighTec GNU GCC for TriCore
  - HighTec GNU GCC for MPC5xxx
  - HighTec GNU GCC for V850X
  - GNU GCC for X86
  - GNU GCC for ARM
- Scanned compilers:
  - Green Hills compiler for TriCore
  - Green Hills compiler for MPC5xxx
  - Green Hills compiler for V850X
  - Green Hills compiler for ARM
- Only certain combinations of compiler type and target are supported as user-defined compilers. The following table shows which combinations are supported:

Target	HighTec	Green Hills	GNU
TriCore	✓	✓	–
MPC5xxx	✓	✓	–
V850X	✓	✓	✓
X86	–	–	✓
ARM	–	✓	✓

Config Opens the Compiler Configuration dialog for you to view and edit the compiler configurations.

Convert Double to Single Float Precision (IEEE754) Lets you specify to convert double to IEEE 754 single-precision float. Activating the conversion can be useful, for example, if your CPU does not contain a processing unit which handles 64-bit floats and software-based 64-bit processing is too time- and resource-consuming.

Note

- The **Convert Double to Single Float Precision (IEEE754)** option cannot be used in combination with the external variable feature. If this option is enabled, the external variable feature cannot be activated. This is due to the fact that a 32-bit address cannot be stored in a single-precision float without precision loss.
- For the INTERNAL interface, the **Convert Double to Single Float Precision (IEEE754)** option is available depending on the target and compiler used.

Target	Compiler	Option Available
ARM	GNU GCC for ARM	–
	Green Hills compiler for ARM	✓
MPC5xxx	HighTec GNU GCC for MPC5xxx	✓
	Green Hills compiler for MPC5xxx	✓[1)]
TriCore	HighTec GNU GCC for TriCore	✓
	Green Hills compiler for TriCore	–
V850X	HighTec GNU GCC for V850X	–
	Green Hills compiler for V850X	✓[2)]
X86	GNU GCC for X86	–

1) The **Convert Double to Single Float Precision (IEEE754)** option is only available for processors with mixed floating-point support (hardware: support for 32-bit, software: support for 64-bit). This applies to the following processors (in brackets: the associated ID, refer to the `microcontroller.xml` file in the `<RCP_HIL_InstallationPath>\MATLAB\RTIBYPASS\M\Configuration` folder): generic (0x1000), MPC5553 (0x1200), MPC5554 (0x1201), MPC5566 (0x1300), MPC5567 (0x1301), MPC5634M/SPC563M (0x1502), MPC5646C/SPC564B (0x1620), MPC5674F (0x1801), MPC5676R (0x1820).

2) The **Convert Double to Single Float Precision (IEEE754)** option is only available for processors in the RH850X target processor family. It is not supported for processors in the V850X target processor family.

Generate extended A2L Lets you specify whether to merge the Simulink parameters/signals from the generated on-target bypass code with the content of one of the imported database file(s). If the checkbox is selected, an extended database file is generated. It is based on the imported database file you select from the **Input A2L File** list, and it is stored under the name you specify via the **Output A2L File** field.

An extended database file allows you to access the Simulink parameters/signals for on-target bypassing in addition to the variables from the imported database file within your calibration tool (for example, ControlDesk) at the same time.

Regardless of whether this option is enabled, the stand-alone A2L file containing only the RTIBypass variables is always generated into the build folder of the model, if ASAP2 generation is activated on the **Interfaces** page of the Code Generation dialog.

Input A2L File (Available only if **Generate extended A2L** is selected) Lets you select the database file you want to use as the basis for the extended A2L file. The list contains all the imported database files.

Output A2L File (Available only if **Generate extended A2L** is selected) Lets you specify the path and the name of the extended A2L file.

Click the **Select** button next to the **Output A2L File** edit field to open the **Macros** dialog. In the dialog, click **Select** to select an existing file or create a new one. By using predefined macros, you can define the label for the output A2L file dynamically. For example, you can include the file creation date and time. The following macros are available:

Macro	Description
$DATE	Inserts the date as follows: [YYYY.MM.DD].
$DD	Inserts the day as follows: [01 ... 31].
$MO	Inserts the month as follows: [01 ... 12].
$YY	Inserts the year as follows: [YYYY].
$TIME	Inserts the time as follows: [HH.MI.SS].
$HH	Inserts the hour as follows: [00 ... 23].
$MI	Inserts the minute as follows: [00 ... 59].
$SS	Inserts the second as follows: [00 ... 59].
$USER	Inserts the current user name.
$PRODUCT	Inserts 'RTI Bypass Blockset' as the product name.
$VERSION	Inserts the version of the RTI Bypass Blockset.

Generate Simulink Objects from Signal Names Lets you specify to automatically generate signals of the Simulink model into the A2L file during code generation. These signals are automatically defined as `Simulink.Signal` instances in the MATLAB workspace. Refer to Adding Parameters and Signals to the A2L File (Simulink Model A2L File Generation Manual 🕮).

For *signals* to be added to the A2L file, their signal names must be unique within the entire Simulink model. To ensure that automatically assigned signal labels for the ports are unique, select the **Create unique signal names from RTIBypass ports** option. Refer to Variables Options Page (RTIBYPASS_SETUP_BLx) on page 56.

Note

There are some points to note when working with TargetLink:

- If you work with a Simulink model and use the TargetLink Code Generator: During model preparation, TargetLink searches for Simulink signals. For each Simulink signal with the `ExportedGlobal` property, a new measurement variable is generated. The new variable is automatically generated into the A2L file during code generation.

  If **Map Signals to ECU internal variables** is selected, a variable of the variable class `BYPASSING_EXTERN_DISP` is created automatically in the TargetLink Data Dictionary for each Simulink signal whose name, data type, scaling and offset values match those of the A2L file variable.
- If you work with a TargetLink model and use the TargetLink Code Generator: Simulink objects are ignored. To define a new measurement variable, you must define a new variable with variable class `BYPASSING_DISP` in the TargetLink Data Dictionary. As an alternative, you can set the variable class `BYPASSING_DISP` directly in the TargetLink block dialog. If you want to reuse an existing A2L file variable in the on-target bypass model, you must define an external variable of variable class `BYPASSING_EXTERN_DISP` in the TargetLink Data Dictionary.

For more information, refer to Basics on Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing (TargetLink Interoperation and Exchange Guide 📖).

Generate Simulink Objects from Parameters Lets you specify to automatically generate parameters of the Simulink model into the A2L file during code generation. Existing referenced numeric workspace variables are converted to `Simulink.Parameter` instances. Multiple referenced workspace variables refer to the same variable in the A2L file. Refer to Adding Parameters and Signals to the A2L File (Simulink Model A2L File Generation Manual 📖).

Note

There are some points to note when working with TargetLink:

- If you work with a Simulink model and use the TargetLink Code Generator: During model preparation, TargetLink searches for Simulink parameters. For each Simulink parameter with the `ExportedGlobal` property, a new calibration variable is generated. The new variable is automatically generated into the A2L file during code generation.

  If Map Parameters to ECU internal variables is selected, a variable of the variable class `BYPASSING_EXTERN_CAL` is created automatically in the TargetLink Data Dictionary for each Simulink parameter whose name, data type, scaling and offset values match those of the A2L file variable.
- If you work with a TargetLink model and use the TargetLink Code Generator: Simulink objects are ignored. To define a new calibration variable, you must define a new variable with variable class `BYPASSING_CAL` in the TargetLink Data Dictionary. As an alternative, you can set the variable class `BYPASSING_CAL` directly in the TargetLink block dialog. If you want to reuse an existing A2L file variable in the on-target bypass model, you must define an external variable of variable class `BYPASSING_EXTERN_CAL` in the TargetLink Data Dictionary.

For more information, refer to Basics on Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing (TargetLink Interoperation and Exchange Guide 🕮).

Map Signals to ECU internal variables Lets you specify to reuse existing signals of the original ECU application in the on-target bypass model. If the checkbox is selected, ECU variables that are already defined in the original ECU code and its associated A2L file are reused in the on-target bypass code. This means that the ECU application and the on-target bypass application share the same signal memory locations. If the checkbox is cleared, ECU variables are not reused. Instead, the on-target bypass code declares its own signal representations.

Reusing ECU variables saves ECU flash memory.

Note

Reusing ECU variables requires the same data types for the ECU variable and MATLAB Simulink object. If the data types differ or if there is another inconsistency, a warning is displayed and the signal is duplicated.

Map Parameters to ECU internal variables Lets you specify to reuse existing parameters of the original ECU application in the on-target bypass model. With the checkbox selected, ECU parameters that are already defined in the original ECU code and in the associated A2L file are reused in the on-target bypass code. This means that the ECU application and the on-target bypass application share the same parameter memory locations. If the checkbox is cleared, ECU parameters are not reused. Instead, the on-target bypass code declares its own parameter representations.

Reusing ECU variables saves ECU flash memory.

Note

Reusing ECU variables requires the same data types for the ECU variable and MATLAB Simulink object. If the data types differ or if there is another inconsistency, a warning is displayed and the parameter is duplicated.

Map Function Block internal variables to ECU internal variables (Available only for the INTERNAL interface in combination with an X86 target (V-ECU on VEOS or SCALEXIO) or ARM target (V-ECU on MicroAutoBox III) Lets you specify to reuse existing signals and parameters of the original ECU application with the variables of functions called by the RTI Bypass Function Call block.

If the checkbox is selected, ECU signals and parameters that are already defined in the A2L file imported in the Setup block are reused for function internal variables called by the FUNCTION block. This means that the ECU application and the function internal variables share the same parameter memory locations.

If the checkbox is cleared, ECU signals and parameters are not reused. Instead, the Function block code declares its own parameter representations for the function internal variables.

Reusing ECU variables in a Function block lets you access the ECU variables inside the on-target bypass model. It saves ECU flash memory and allows to connect the function execution directly to ECU-internal values, e.g., states or queues.

Note

Reusing ECU variables in a Function block requires the same data types for the ECU variable and the variable in the Function block included in the C file or the header file. If the data types differ or if there is another inconsistency, the variable is duplicated.

The A2L file imported in the Function block must have the same conversion method as the A2L file imported in the Setup block.

Merge internal bypass code into ECU application (Not available for the INTERNAL interface in combination with an X86 target (V-ECU on VEOS or SCALEXIO) or ARM target (V-ECU on MicroAutoBox III)). In case of VEOS, the code is loaded directly to VEOS and merging is not necessary. In case of MicroAutoBox III or SCALEXIO, the ECU service is already in the RTLib of MicroAutoBox III or SCALEXIO.) Lets you specify whether to merge the generated on-target bypass code binary into an original ECU application. If the checkbox is selected, the on-target bypass code is merged with the ECU application you select as source from the Input ECU application list, and the new ECU application is stored under the name you specify via the Output ECU application field.

Input ECU application (Available only if Merge internal bypass code into ECU application is selected) Lets you select the original ECU application you want to merge the on-target bypass code with. The ECU application can be in the Intel HEX file format (HEX, IHEX) or Motorola S-record file format (S19, SREC). The list contains all the ECU applications defined in the imported A2L files and, if available, a manually added ECU application. To add an input ECU application to the list, select the **<Browse>** entry from the list and select the ECU application file. Only one manually added ECU application can be in the **Input ECU application** list at a time.

Output ECU application (Available only if **Merge internal bypass code into ECU application** is selected) Lets you specify the path and the name of the merged ECU application. The merged ECU application can be stored in Intel HEX file format (as HEX or IHEX file) or Motorola S-record file format (as S19 or SREC file). The file format of the output ECU application is independent of the input ECU application's file format. Click the **Edit** button next to the **Output A2L File** edit field to open the **Macros** dialog. In the dialog, click **Select** to select an existing file or create a new one. Using predefined macros, you can define the label for the output ECU application file dynamically. For example, you can include the file creation date and time. The following macros are available:

Macro	Description
$DATE	Inserts the date as follows: [YYYY.MM.DD].
$DD	Inserts the day as follows: [01 ... 31].
$MO	Inserts the month as follows: [01 ... 12].
$YY	Inserts the year as follows: [YYYY].
$TIME	Inserts the time as follows: [HH.MI.SS].
$HH	Inserts the hour as follows: [00 ... 23].
$MI	Inserts the minute as follows: [00 ... 59].
$SS	Inserts the second as follows: [00 ... 59].
$USER	Inserts the current user name.
$PRODUCT	Inserts 'RTI Bypass Blockset' as the product name.
$VERSION	Inserts the version of the RTI Bypass Blockset.

Perform Post Process Lets you specify whether to execute a postprocessing DOS command after the build process finishes. If the checkbox is selected, the command specified in the **Post Process Command Line** field is invoked.

Post Process Command Line (Available only if **Perform Post Process** is selected) Lets you enter the DOS command to be invoked after the build process finishes. You can type text in the edit field and/or select predefined macros. Clicking the **Edit** button opens a dialog for you to select the predefined macros and insert them at the current position. Predefined macros are available for the following elements:

- Model name
- Input A2L file
- Output A2L file
- Input ECU application
- Output ECU application

Note

If the path or file name includes spaces it must be enclosed by quotation marks. Predefined macros are quoted automatically.

Flash ECU application (Not available for the INTERNAL interface in combination with an X86 target (V-ECU on VEOS). In this case, the code is loaded directly to VEOS and flashing is not necessary.) Lets you select whether the generated ECU application is automatically flashed to the ECU after the build process finishes. If the checkbox is selected, you must specify the way the ECU application is to be flashed. You can choose to use the dSPACE ECU Flash Programming Tool or a custom DOS command.

Flash Project Root (Available only if dSPACE ECU Flash Programming Tool is selected as the flashing method) Lets you specify the working folder which contains the flash project you want to use for flashing the ECU application. Click the Select button next to the Flash Project Root edit field to select an existing folder.

Flash project (Available only if dSPACE ECU Flash Programming Tool is selected as the flashing method) Lets you select the flash project to be used. The list contains all the flash projects that are related to the selected working folder.

Custom command line (Available only if Custom Flash Command is selected as the flashing method) Lets you enter a DOS command to be invoked for flashing the ECU application. You can type text in the edit field and/or select predefined macros. Clicking the Edit button opens a dialog for you to select the predefined macros and insert them at the current position. Predefined macros are available for the following elements:

- Model name
- Input A2L file
- Output A2L file
- Input ECU application
- Output ECU application

Note

If the path or file name includes spaces it must be enclosed by quotation marks. Predefined macros are quoted automatically.

Download Internal Bypass application to VECU (Available only for the INTERNAL interface in combination with an X86 target (V-ECU on VEOS or SCALEXIO) or ARM target (V-ECU on MicroAutoBox III)) Lets you select whether to download the generated Simulink model code to the virtual ECU. If the checkbox is selected, the `downIntBypService.exe` tool (coming with the RTI Bypass Blockset) is used to load the Simulink model code (ELF file) to the V-ECU.

The `downIntBypService.exe` tool is located in the `<RCP_HIL_InstallationPath>\Exe` folder. For information on the syntax and the command line parameters that can be used with the `downIntBypService.exe` tool, refer to the tool help. To display the help, enter **`downIntBypService.exe -h`** in a Command Prompt window. When you use the tool, you have to specify the entire path to it.

IP Address of VECU Bypass Service (Available only if Download Internal Bypass application to VECU is selected) Lets you enter the destination IP address or host name to which the Simulink model code is loaded.

Port (Available only if Download Internal Bypass application to VECU is selected) Lets you enter the number of a destination port to which the Simulink model code is loaded.

For more information on the port that is specified when using the `-p <port>` parameter with the `downIntBypService.exe` tool, refer to Rapid Prototyping Service Interface (SystemDesk Manual 📖).

Note

When the A2L file is generated, the IP Address of VECU Bypass Service and the Port parameters are written to the A2L file and should not be modified afterwards.

Note

The V-ECU to be bypassed must be prepared for V-ECU access in SystemDesk using the Rapid Prototyping Access (RptAccess) module. For more information, refer to:

- Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on VEOS on page 248
- Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on a SCALEXIO System or on a MicroAutoBox III on page 250

Compiler Configuration dialog

Compiler Configurations Lists the compiler configurations that are available for the current bypass model and lets you select a compiler configuration to modify.

The default compilers cannot be reconfigured; therefore, their edit fields for the configuration are grayed out. However, you can copy a default compiler configuration and then make changes to the copy. Scanned compilers can be partly reconfigured, i.e., some edit fields are writable.

Add Lets you add a compiler configuration to the Compiler Configurations list. A dialog opens for you to specify the path of the folder that contains the compiler. After you click the Select Folder button, the dialog is closed and the specified folder is scanned for compiler configurations. Any compiler configurations that are found are added to the Compiler Configurations list. They are initialized with the scanned configuration data.

Note

If there are several compiler configurations in the selected folder, it is not possible to guarantee that all configuration data is scanned appropriately, i.e., that all configuration data belong to one and the same compiler configuration. Therefore, you must check the configuration data added to the **Compiler Configuration** dialog.

Copy Lets you create a copy of the selected existing compiler configuration and add it as a new compiler configuration to the **Compiler Configurations** list with a default name. You can modify its configuration manually later.

Delete Lets you delete the selected compiler configuration from the **Compiler Configurations** list.

You cannot delete a default compiler configuration or scanned compiler configuration. If you do so, a message is displayed in the **Status** info field. Only user-defined compiler configurations can be deleted.

Import Lets you import one or more scanned compiler configurations or user-defined compiler configurations to the RTI Bypass Blockset. The **Import Compiler Configuration** dialog opens for you to specify the name and path of the file containing the compiler configurations to be imported. All the contained compiler configurations are displayed. To select a compiler configuration for the import, select the checkbox in the column on the left. Because compiler configurations must have unique names, you must specify for each configuration via the **Operation** option how to proceed if the import causes a naming conflict:

- **Discard import**: The selected compiler configuration is not imported. The existing compiler configuration remains.
- **Overwrite**: The existing compiler configuration is overwritten with the selected compiler configuration during the import.
- **Rename**: The existing compiler configuration remains unchanged. The selected compiler configuration is imported and renamed.

You can use the **OK** button to start the import process and close the **Import Compiler Configuration** dialog. The imported compiler configurations are displayed in the **Compiler Configurations** list according to your settings.

Export Lets you export compiler configurations. The **Export Compiler Configuration** dialog opens for you to specify the path and name of the export file and select the compiler configurations to be exported. The export file must be a MAT file. Only the scanned compiler configurations and user-defined compiler configurations can be exported. Because the configurations of the default compilers cannot be exported, they are not displayed in the dialog. To select a compiler configuration to be exported, select the checkbox in the column on the left.

You can use the OK button to start the export process and close the Export Compiler Configuration dialog.

Name Displays or lets you enter the name for the selected compiler configuration. The new name is temporary until you confirm the modification by pressing OK or Apply.

Path Displays the path to the folder that contains the compiler, or lets you specify the folder location via the Browse button. The new path is temporary until you confirm it by pressing OK or Apply.

Type Displays or lets you select the compiler type. The new compiler type is temporary until you confirm it by pressing OK or Apply.

Target Displays or lets you select the target microcontroller family to be used with the selected compiler configuration. The new target is temporary until you confirm the modification by pressing OK or Apply.

C Compiler Displays or lets you select the C compiler file to be used with the selected compiler configuration. The new C compiler file is temporary until you confirm the modification by pressing OK or Apply.

Assembler Displays or lets you select the assembler file to be used with the selected compiler configuration. The new assembler file is temporary until you confirm the modification by pressing OK or Apply.

Linker Displays or lets you select the linker file to be used with the selected compiler configuration. The new linker file is temporary until you confirm the modification by pressing OK or Apply.

Symbol Table Reader (nm) Displays or lets you select the symbol table reader file to be used with the selected compiler configuration. The new symbol table reader file is temporary until you confirm the modification by pressing OK or Apply.
A typical symbol table reader is `nm.exe` in the GNU Compiler Collection.

Binary Format Converter Displays or lets you select the binary format converter file to be used with the compiler configuration. The new converter file is temporary until you confirm the modification by pressing OK or Apply.
A typical binary format converter is `objcopy.exe` in the GNU Compiler Collection.

Restore Defaults Scans the compiler path specified in the Path edit field for C compiler, assembler, linker, symbol table reader, and binary format converter files, and overwrites the current compiler configuration settings in the C Compiler, Assembler, Linker, Symbol Table Reader (nm), and Binary Format Converter fields with the file names found.

Status Displays warning and error messages concerning the currently executed action.

Related topics

References

Memory Page (RTIBYPASS_SETUP_BLx for INTERNAL)..216

Memory Page (RTIBYPASS_SETUP_BLx for INTERNAL)

Purpose

To get address information on the service configuration structures, and to view or change the memory layout of the target ECU.

The on-target bypass build process uses these settings to generate a compiler-specific linker command file, which is required for the target-specific build process.

Note

Normally, the memory layout is specified by the IF_DATA INTERNAL_BYPASS of the imported A2L file. Modifying or extending this configuration is for advanced use only. You should only modify the memory layout for on-target bypassing if you are familiar with the memory structure of the imported ECU application. Misconfiguration of the on-target bypass memory layout can cause your ECU application to malfunction.

Note

This page is not available for the INTERNAL interface in combination with an X86 target (V-ECU on VEOS or SCALEXIO) or ARM target (V-ECU on MicroAutoBox III).

Dialog settings

IntByp Config Address Displays the start address of the service configuration structure used by the RTI Bypass Blockset. The configuration structure is declared and kept by the dSPACE Internal Bypassing Service, and will be overwritten by information generated by the RTI Bypass Blockset. The structure is used by the dSPACE Internal Bypassing Service to access the on-target bypass code.

Service Config Address Displays the start address of the service configuration structure used by the dSPACE Internal Bypassing Service. The service uses the structure to store information about the current service state of the on-target bypass code at run time.

Address Displays or lets you enter the start address of the memory segment currently selected in the list of memory segments.

Length Displays or lets you enter the length of the selected memory segment as a hexadecimal number.

Type Displays or lets you select the memory type of the selected memory segment. The type can be either flash or RAM.

Inst. Set (Available only for target processors of the Freescale MPC5xxx target processor family) Displays or lets you select the instruction set of the selected memory segment. The instruction set can be either BOOK-E or VLE. If you select **BOOKE**, the code will be interpreted as BOOK-E code. If **VLE** is selected, VLE-specific code will be compiled for the target application.

Note

- All the memory segments of CODE type must have identical **Instruction set** settings. Do not use VLE and BOOK-E code for the same target application.
- If you specify memory segments of CODE type, you must ensure that they are sufficiently large to hold entire functions.
  During code generation the segments are used in increasing order of the segment lengths, i. e., first the smaller memory segments are filled.

Code Lets you specify that the selected memory segment is for content of CODE type. If the checkbox is selected, the code section of the on-target bypass code will be placed in this segment. It is mandatory to specify at least one CODE segment to perform on-target bypassing.

Parameter Lets you specify that the selected memory segment is for content of PARAMETER type. If the checkbox is selected, the Simulink calibration parameters will be linked to this segment. The PARAMETER segment is optional. If no PARAMETER segment is specified, the Simulink parameters are placed in the CODE segment.

Variable Lets you specify that the selected memory segment is for content of VARIABLE type. If the checkbox is selected, the heap and the global variable sections will be linked to this segment. It is mandatory to specify at least one VARIABLE segment to perform on-target bypassing.

Note

Do not select VARIABLE for memory segments of 'flash' memory type.

List of memory segments Displays the memory segments that are available for on-target bypassing purposes with their configuration settings. The memory segments are numbered for identification purposes. The list contains all the memory segments specified in the imported database files and the manually added custom memory segments. The currently selected memory segments are colored green. Segments defined in the A2L file cannot be changed.

As soon as the on-target bypass application has been built, the **Fill Level** column indicates how much memory space is currently used for each memory segment. The fill level is displayed as an absolute value (in kbit), and also relative to the segment's length (in %).

A detailed summary of the segment fill levels is also printed in the MATLAB Command Window after the build process has finished.

Add Lets you add a custom memory segment. Since a new segment is always created as a copy of an existing memory segment, you must select a memory segment from the list and click Add. Afterwards, you can modify the configuration settings of the new memory segment via the edit fields and checkboxes above the segment list.

Remove Lets you remove the selected memory segment from the list. You can remove custom segments only.

Related topics

References

Build Page (RTIBYPASS_SETUP_BLx for INTERNAL)..204
Options Page (RTIBYPASS_SETUP_BLx for INTERNAL)....................................218
Unit Page (RTIBYPASS_SETUP_BLx)..53

Options Page (RTIBYPASS_SETUP_BLx for INTERNAL)

Purpose

To specify the service instance where the timer task (i.e., the root subsystem and all its blocks) is to be called.

Dialog settings

Service instance Lets you select the service instance where the timer task (i.e., the root subsystem with all its blocks) is to be called. An ECU service can be executed in different rasters (service instances) which are defined in the ECU´s database file. The names of the service instances are unique, and each service instance corresponds to one service ID. The Service instance drop-down list contains the available service instances, determined by the selected imported database files for the selected bypass interface. By default, no service instance is preset and the 'NOT_DEFINED' value is displayed.

Service ID Displays the service ID associated with the selected service instance. The service IDs are unique, and each ID corresponds to one service instance.

Service instance sample rate (sec) Displays the sampling rate defined for the selected service instance.

Note

- The fixed-step size (fundamental sample time) specified in the solver options in the Simulink configuration parameters (see Solver Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 📖)) should be an integer multiple of the sampling rate associated with the selected service instance. If it is not, the base sample time for the model is ignored, and the timer task is called in the raster defined for the selected service instance instead.
- If you use service instances of *non-deterministic* periodicity (for example, events inserted with the ECU Interface Manager), ensure that the trigger periodicity of these events matches the fixed-step size. Otherwise, if the execution of the timer task (model root) differs from the fixed-step size, the absolute time does not represent the correct simulation time. In this case, some MATLAB Simulink blocks such as Signal Generator, Discrete-Time Integrator, Rate Limiter, etc., will not work correctly. For further information on the absolute time, refer to the Simulink documentation from MathWorks®.
- If the root task mapping is not defined, the build process is performed without an error. However, only the samples relevant for subsystems that are triggered by an RTI Bypass Interrupt block are performed. Blocks that depend on the base sampling rate of the model (e.g., Integrator blocks) can provide invalid results.

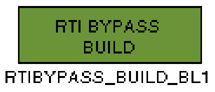
Related topics

References

Build Page (RTIBYPASS_SETUP_BLx for INTERNAL)	204
Memory Page (RTIBYPASS_SETUP_BLx for INTERNAL)	216
Unit Page (RTIBYPASS_SETUP_BLx)	53

RTI BYPASS BUILD Block

Block



Purpose

To configure and start the build process for the on-target and external bypass parts in the model.

Description

After you configure a Setup block for on-target bypassing, an RTI BYPASS BUILD block is automatically added to the Simulink model.

> **Note**
>
> Configuring a Setup block for external bypassing does not add an RTI BYPASS BUILD block to the Simulink model.

If you have a Simulink model with on-target and external bypass parts, the RTI BYPASS BUILD block contains configuration settings for both external and on-target bypass model parts. You can select the parts to be included in the build process and specify the actions to be performed.

Dialog settings

External To specify settings relevant for building external bypass code. The specified settings apply to all external bypass parts in the Simulink model.

- Clean: Lets you specify whether to remove the previously generated files from the working folder.
- Build: Lets you specify whether to include the external bypass parts of the model in the build process. All generated files for external bypass have the suffix EXTBYP.
- Load: Lets you specify whether to automatically download the generated external bypass ECU application to the RCP system after the build process has finished.

The options are grayed out if the model contains on-target bypass parts only.

Internal To specify settings relevant for building on-target bypass code. All bypass interfaces specified in the model for on-target bypassing are listed. The specified settings apply to all the bypass interfaces currently selected from the list.

- Clean: Lets you specify whether to remove the previously generated files from the working folder.
- Build: Lets you specify whether to include the corresponding Setup blocks in the on-target build process. All the files generated for on-target bypassing have the suffix INTBYP. The bypass interface name of the corresponding Setup block is an element of the file name. This facilitates identification of the individual on-target bypass parts.
- Flash: Lets you specify whether to automatically flash the generated on-target bypass ECU applications after the build process has finished.

> **Note**
>
> An on-target bypass ECU application can be flashed only if flashing is also activated on the Build page of the corresponding Setup block.

Generate code only Lets you only generate code from your Simulink model.

- If selected, the model code, makefiles, batch files and XML interface data files are generated. The code is only generated, but it is neither compiled nor downloaded to the hardware. Files that are relevant to the generated real-time application, such as the TRC file, are not generated.
- If cleared, the model code is generated, compiled, and downloaded to the dSPACE hardware (default).

This setting applies to code generation with Simulink Coder or the TargetLink Code Generator.

Run Lets you start the build process for the selected bypass parts. The selected actions are performed.

Stop Lets you stop the build process for the selected bypass parts. The build process stops after the currently active operation finishes.

Close Lets you close the block dialog.

Related topics

Basics

External, Service-Based Bypassing: Basic Information and Underlying Processes........................ 34
On-Target, Service-Based Bypassing: Basic Information and Underlying Processes..................... 41

References

Build Page (RTIBYPASS_SETUP_BLx for INTERNAL).. 204
Memory Page (RTIBYPASS_SETUP_BLx for INTERNAL).. 216
Options Page (RTIBYPASS_SETUP_BLx for INTERNAL)... 218
Unit Page (RTIBYPASS_SETUP_BLx).. 53

Options Page (RTIBYPASS_READ_BLx for INTERNAL)

Purpose

To enable or disable the double buffer mechanism and the failure checking mechanism, and to make settings for hardware interrupts.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer mechanism is used to ensure consistent read operations. If the checkbox is selected, only consistent data is transferred between the original ECU application and the on-target bypass code.

If the checkbox is cleared, the Failure count value edit field is also disabled.

For further information on the double buffer mechanism, refer to Extended Bypassing Mechanisms for On-Target Bypassing on page 239.

Failure count value Lets you specify a failure count value in the range 0 ... 127. The value specifies how often the Read block reads old data from the ECU before reading is deactivated and the Status port of the RTI Bypass Read block returns 'No data copied'.

This edit field is enabled only if the **Enable double buffer mechanism** checkbox is selected.

For further information on the failure checking mechanism, refer to Extended Bypassing Mechanisms for On-Target Bypassing on page 239.

Disable interrupts before reading Lets you specify whether to disable hardware interrupts while the variables of the current Read block are sampled. It is advisable to disable interrupts if the ECU application provides tasks that interrupt themselves.

Related topics

References

Options Page (RTIBYPASS_WRITE_BLx for INTERNAL)

Purpose

To enable or disable the double buffer mechanism and the failure checking mechanism, and to make settings for hardware interrupts.

Dialog settings

Enable double buffer mechanism Lets you enable or disable the double buffer mechanism. The double buffer mechanism is used to ensure consistent write operations. If the checkbox is selected, only consistent data is transferred between the original ECU application and the on-target bypass code.

If the checkbox is cleared, the **Failure count value** edit field is also disabled.

For further information on the double buffer mechanism, refer to Extended Bypassing Mechanisms for On-Target Bypassing on page 239.

Failure count value Lets you specify a failure count value in the range 0 ... 127. The value specifies how often the Write block writes old data to the ECU before the bypass hooks are deactivated and the data calculated by the control algorithm in the original ECU application is no longer overwritten.

This edit field is enabled only if the **Enable double buffer mechanism** checkbox is selected.

For further information on the failure checking mechanism, refer to Extended Bypassing Mechanisms for On-Target Bypassing on page 239.

Disable interrupts before writing Lets you specify whether to disable hardware interrupts while the variables of the current Write block are written to the original ECU application memory. It is advisable to disable interrupts if the ECU application provides tasks that interrupt themselves.

Related topics

References

Options Page (RTIBYPASS_WRITE_BLx).. 85

Options Page (RTIBYPASS_UPLOAD_BLx for INTERNAL)

Purpose

To make settings for hardware interrupts.

Dialog settings

Disable interrupts before reading Lets you specify whether to disable hardware interrupts while the variables of the current Upload block are sampled. It is advisable to disable interrupts if the ECU application provides tasks that interrupt themselves.

Related topics

References

Options Page (RTIBYPASS_UPLOAD_BLx).. 107

Options Page (RTIBYPASS_DOWNLOAD_BLx for INTERNAL)

Purpose

To make settings for hardware interrupts.

Dialog settings

Disable interrupts before writing Lets you specify whether to disable hardware interrupts while the variables of the current Download block are downloaded to the original ECU application memory. It is advisable to disable interrupts if the ECU application provides tasks that interrupt themselves.

Related topics

References

Options Page (RTIBYPASS_DOWNLOAD_BLx)... 123

Glossary

Introduction

The glossary briefly explains the most important expressions and naming conventions used in the RTI Bypass Blockset documentation.

Where to go from here

Information in this section

A

A2L file A file that contains all the relevant information on measurement and calibration variables in an ECU application and the ECU's communication interface(s). This includes information on the variables' memory addresses and conversion methods, the memory layout and data structures in the ECU, as well as interface-specific data.

Address-based bypassing A technique for external bypassing in which the ECU variables are accessed directly via their addresses in the ECU memory.

There is no synchronization between the ECU application and the bypass function.

Address-based, hardware-supported bypassing A technique for external bypassing that uses hardware resources, such as watchpoints or a data trace interface supported by the ECU/microcontroller and the DCI-GSI2. The triggering of the bypass function on the external RCP system can be implemented by using these hardware resources, i.e., by means of various event trigger mechanisms provided by ECU interface hardware such as the DCI-GSI2. This means, you can use several functionalities for data acquisition (DAQ) and data stimulation (STIM) purposes that use different event sources.

A data trace interface can also be used for reading and overwriting data consistently via the Fast Variable Overwrite mechanism.

B

Bypass function A function that is part of a real-time application, and that has read/write access to the functions and variables of an ECU application during ECU application run time.

Depending on the used bypassing method, bypass functions are executed:

- On an external RCP system (external bypassing), or
- On the ECU itself (on-target bypassing)

Bypass hook Service call

Bypassing A method for replacing an existing ECU function by running a new function.

C

CCP Abbreviation of CAN Calibration Protocol. This protocol can be implemented on electronic control units (ECUs) and allows users to access ECUs with measurement and calibration systems (MCS) such as ControlDesk.

Code-patch-based bypassing A technique for external bypassing in which no ECU service needs to be integrated into the ECU code. Instead, customized code patches need to be integrated into the ECU code. If the input and output variables of functions change, the ECU code typically also needs to be modified.

The triggering of the bypass function on the RCP system can be implemented by means of subinterrupt handling, or by using a dedicated microcontroller pin as a single hardware interrupt.

Code-patch-based bypassing is a derivative of address-based bypassing.

D

DCI-GSI2 A dSPACE-specific interface between the on-chip debug interface of an ECU's microcontroller and the host PC and/or dSPACE real-time hardware. It can be used for ECU calibration, measurement, ECU flash programming and external bypassing.

DPMEM POD A dual-port memory plug-on device (DPMEM POD) is an interface between an ECU's external memory bus and dSPACE real-time hardware. It can be used for external ECU interfacing such as function bypassing.

dSPACE real-time hardware Hardware on which a real-time application is executed.

In the ECU interfacing context, dSPACE real-time hardware can be used for external bypassing.

E

ECU Abbreviation of *electronic control unit*.

ECU application A sequence of operations executed by a target system. An ECU application mostly is represented by a group of files such as ECU Image files, map files, A2L files and/or software module description files.

ECU Image file A binary file that is part of the ECU application. It usually contains the code of an ECU application and the data of the parameters within the application.

ECU interface A combination of ECU interface hardware and/or ECU interface software enabling a tool to access ECU application variables and/or to influence the ECU application.

ECU interface hardware A hardware module such as the DCI-GSI2 used to access an ECU. This module adapts a hardware interface of an ECU, such as the ECU microcontroller's debug interface, to make it accessible from a tool.

ECU interface software A software component that can be represented by a software module, providing a service for accessing an ECU application. ECU interface software lets tools such as dSPACE's ControlDesk access ECU application values and/or influence the ECU application's behavior.

ECU interfacing Methods and tools to synchronously read and/or write individual variables of an ECU application.

ECU service A service to be integrated into the ECU software for accessing the ECU software for purposes such as service-based measurement, calibration, service-based bypassing, and/or ECU flash programming.

The ECU service type determines which mechanism can be implemented to ensure data consistency (e.g., the double buffer mechanism).

The RTI Bypass Blockset supports the following ECU services:

ECU Service	Supported Bypassing Method
CCP service	External bypassing
dSPACE Calibration and Bypassing Service	External bypassing
dSPACE Internal Bypassing Service	On-target bypassing
XCP service – either the dSPACE XCP Service or a custom XCP service	External bypassing

ECU with CCP An ECU on which a CCP service is implemented.

ECU with DCI-GSI2 An ECU on which a DCI-GSI2 is mounted.

ECU with XCP on CAN An ECU on which an XCP service is implemented. The transport layer is *XCP on CAN*.

ECU with XCP on Ethernet (UDP/IP) An ECU on which an XCP service is implemented. The transport layer is *XCP on Ethernet (UDP/IP)*.

ECU with XCP on FlexRay An ECU on which an XCP service is implemented. The transport layer is *XCP on FlexRay*.

External bypassing A method for bypassing in which ECU functionalities are executed by an external rapid control prototyping (RCP) system, such as the MicroAutoBox II.

The RCP system is used in parallel to the ECU. The functions to be bypassed are offloaded from the ECU to the RCP system. The new algorithms are executed on the prototyping hardware, while the ECU still runs the existing code at the same time.

The calculated results from the bypass system can be used for further processing by the ECU.

I

Interface description data (IF_DATA) An information structure, mostly provided by an A2L file, describing the type, features, and configuration of an implemented ECU interface.

Internal bypassing On-target bypassing

O

On-target bypassing A method for bypassing in which bypass functions are executed on the target ECU. No external rapid control prototyping (RCP) system is used.

On-target bypassing uses specific ECU-internal resources. Bypass functions are executed in remaining free memory areas of the ECU RAM and flash memory.

R

Real-time application An application that can be executed in real time on dSPACE real-time hardware. A real-time application can be built from a Simulink model containing RTI blocks, for example.

S

Service call A call of a function of an ECU service. It can be placed in an ECU application.

Service calls are used with service-based bypassing.

Service-based bypassing A technique for external bypassing and on-target bypassing in which an ECU service (the part of the software on the ECU that allows access to the ECU´s variables) has to be integrated in the ECU code only once, and can be adapted individually to your requirements. You can easily change variables without having to recompile the code even after the service has been implemented.

Functions in the ECU code are prepared for bypassing by inserting service calls. Service calls in the ECU code can be described in an ECU description file (A2L file) and evaluated by the RTI Bypass Blockset, so you can flexibly select the functions to be bypassed and the ECU variables to be read and written in the modeling environment.

There are two ways to prepare the ECU code for bypassing:

	ECU Source Code Modification Required	**Recompilation of the ECU Application Required**
Pre-build integration[1]	✓	✓
Post-build integration[2]	–	–

[1] Modification of the ECU application source code, i.e., integration of the ECU service *as source code*.

[2] Adaptation of the ECU application binary code, i.e., integration of the ECU service *as binary code*.

Serviceless bypassing Generic term for the following techniques for external bypassing in which no ECU service has to be integrated in the ECU code:

- Address-based bypassing
- Address-based, hardware-supported bypassing
- Code-patch-based bypassing

X

XCP Abbreviation of *Universal Measurement and Calibration Protocol*. A protocol that is implemented on electronic control units (ECUs) and provides access to ECUs with measurement and calibration systems (MCS) such as ControlDesk. The "X" stands for the physical layers for communication between the ECU and the MCS, such as CAN (Controller Area Network) and Ethernet.

Appendix

Where to go from here

Information in this section

Code-Patch-Based Bypassing: Subinterrupt Mechanisms

Introduction

For code-patch-based bypassing, the RTI Bypass Blockset supports two different subinterrupt mechanisms: the bit-based and the byte-based subinterrupt mechanism. Both mechanisms are used to generate and handle multiple subinterrupts using a single hardware interrupt line.

Where to go from here

Information in this section

Bit-Based Subinterrupt Mechanism

Introduction

The bit-based subinterrupt mechanism triggers multiple subinterrupts using a single interrupt trigger line. One bit is used per subinterrupt.

Subinterrupt handling locations

The bit-based subinterrupt mechanism makes use of a certain amount of ECU memory starting from the `subinterrupts start address` (as specified in the `Interrupt_Handling` struct in the A2L file) to store the current status of the respective subinterrupts.

Note

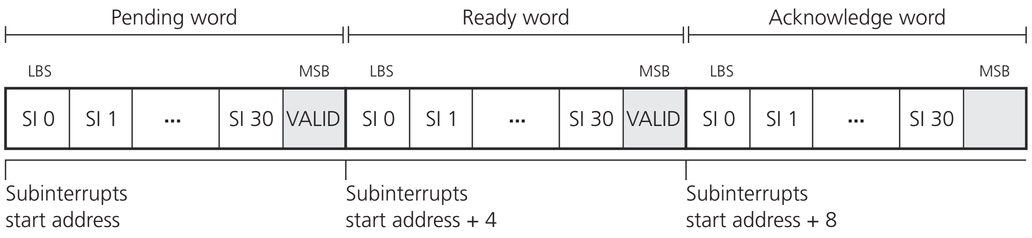
The subinterrupt configuration specified in the `Interrupt_Handling` struct in the ECU's A2L file must match the subinterrupt configuration in the ECU code.

The subinterrupt memory area is divided into a pending, a valid, and an acknowledge area (see examples below). Each of these areas consists of ((NumberOfSubinterrupts / 32) + 1) 32-bit words. This means that one subinterrupt pending word, one subinterrupt ready word, and one subinterrupt acknowledge word are required for a number of subinterrupts of up to 31. If the number of subinterrupts is between 32 and 63, two words are needed for each bit category. If more subinterrupts are needed, a proportionate number of additional words is required. In general, subinterrupt N corresponds to bit (N modulo 31) in word (N/32) of the pending, ready and acknowledge memory areas.

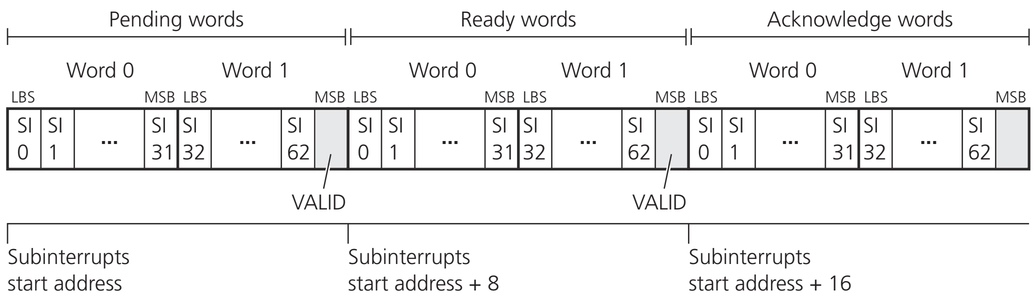
The first subinterrupt pending word starts at the `subinterrupts start address`. It is followed by further subinterrupt pending words (if existent), the subinterrupt ready word(s) containing the ready bits, and the subinterrupt acknowledge word(s) containing the acknowledge bits.

For each bit category, the bits are assigned to the subinterrupts. Starting with the least significant bit (LSB) of the first word, the bits are assigned to subinterrupt 0, subinterrupt 1, ... The most significant bits (MSB) of the last words of the pending and the ready area are each used as global valid bit. The other bits are used to encode the subinterrupt status.

Example 1 Suppose 31 subinterrupts are used (NumberOfSubinterrupts = 31). One 32-bit word each is used for the pending, the valid and the acknowledge area. The lower 31 bits of the pending, ready and acknowledge words are used to code the subinterrupt state of subinterrupts 0 ... 30. The most significant bits of the pending and ready words are used for subinterrupt validation/invalidation. The subinterrupt area in the ECU memory looks as follows ("SI x" stands for the bit position which is used to code the state of subinterrupt x):



Example 2 Suppose 63 subinterrupts are used. In this case, two 32-bit words each are used for the pending, valid and acknowledge areas. The subinterrupt states of subinterrupts 0 ... 31 are coded using bit 0 ... 31 of the first 32-bit word of each memory area. The subinterrupt states of subinterrupts 32 ... 62 are coded using bit 0 ... 30 of the second 32-bit word of each memory area. The subinterrupt area in the ECU memory looks as follows ("SI x" stands for the bit position which is used to code the state of subinterrupt x):



Pending words
Ready words
Acknowledge words
Word 0
Word 1
LBS
MSB
SI 0
SI 1
...
SI 31
SI 32
SI 62
VALID
Subinterrupts start address
Subinterrupts start address + 8
Subinterrupts start address + 16

Bit-based subinterrupt handling method

Subinterrupt states For the bit-based subinterrupt mechanism, the subinterrupts can be in one of four states: pending, acknowledged, ready, invalid. The subinterrupt states are coded as follows:

- *Pending*: Subinterrupt N is in the pending state, if bit N in the pending word is different to both bit N in the ready word and bit N in the acknowledge word.
- *Acknowledged*: Subinterrupt N is in the acknowledged state, if bit N in the pending word is different to bit N in the ready word, but equal to bit N in the acknowledge word.
- *Ready*: Subinterrupt N is in the ready state, if the bit N values in the pending, ready and acknowledge words are equal.

The *invalid* subinterrupt state is used at ECU or RCP system startup or restart to avoid interrupt inconsistencies or task overruns caused by pending subinterrupts of a previous bypassing session.

Subinterrupt processing Subinterrupt processing consists of several steps. These are the steps for subinterrupt N (starting from the ready subinterrupt state):

1. The ECU sets the subinterrupt state to pending by toggling bit N in the pending word, and triggers a hardware interrupt afterwards.
2. After receiving the hardware interrupt, the RCP system acknowledges the subinterrupt by toggling bit N in the acknowledge word, and starts processing the tasks of the Simulink subsystem triggered by subinterrupt N.
3. After all the tasks of the interrupt-driven Simulink subsystem connected to the subinterrupt have finished, the RCP system sets the subinterrupt state to ready by toggling bit N in the ready word. Now the subinterrupt is ready to be triggered by the ECU again.

Note

After indicating a new subinterrupt, the trigger of the hardware interrupt line is required to indicate the RCP system that a new subinterrupt is available.

Subinterrupt validation/invalidation To avoid inconsistent subinterrupt states at ECU or RCP system startup, the most significant bits of the last pending and ready words are used as valid bits. The subinterrupt states coded by the other bits are valid only if both the pending valid bit and the ready valid bit are set. Otherwise all subinterrupts have the invalid state.

At ECU startup, the ECU has to write '0' to all pending, valid and acknowledge words. This sets all subinterrupts to invalid and all subinterrupt bits to 0, allowing a consistent subinterrupt state at ECU startup. During normal operation, the ECU only writes to the pending word(s), whereas the RCP system writes to the ready and acknowledge word(s).

ECU subinterrupt validation/invalidation is performed according to the following rules:

- Set the pending valid bit, if the ready valid bit is set.
- Reset the pending valid bit and write '0' to all pending words, if the ready valid bit is reset.

The ECU can change the subinterrupt state from invalid to pending and trigger a subinterrupt only if the ready valid bit is set.

Depending on the ECU alive state transitions and the bypassing simulation state, the RCP system validates the subinterrupts by setting the ready valid bit, or invalidates the subinterrupts by resetting the ready valid bit and writing '0' to all ready and acknowledge words.

Demo ECU application

The RTI Bypass Blockset comes with demo ECU applications containing source code examples for bit-based subinterrupt handling. You can find the demo ECU applications for the different code-patch-based bypass interfaces in `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS`.

You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.

Related topics

Basics

External, Code-Patch-Based Bypassing: Basic Information and Underlying Processes................. 38

Byte-Based Subinterrupt Mechanism

Introduction

The byte-based subinterrupt mechanism triggers multiple subinterrupts using a single interrupt trigger line. One byte per subinterrupt of the subinterrupt memory area is reserved for coding the respective subinterrupt state.

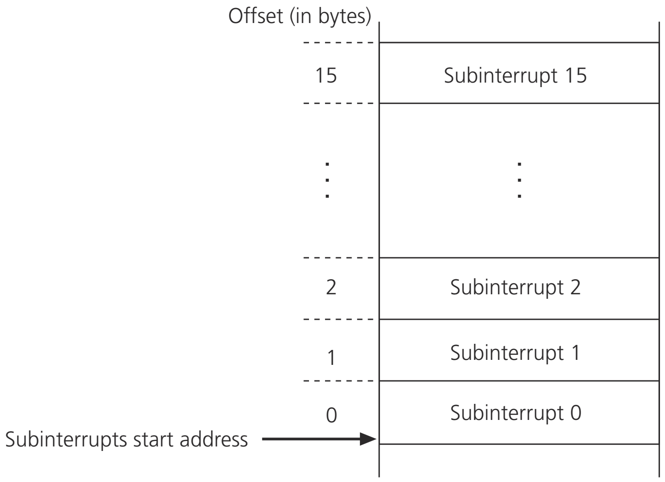
Subinterrupt handling locations

The byte-based subinterrupt mechanism makes use of a certain amount of ECU memory starting from the `subinterrupts start address` (as specified in the `Interrupt_Handling` struct in the A2L file) to store the current status of the respective subinterrupts.

Note

The subinterrupt configuration specified in the `Interrupt_Handling` struct in the ECU's A2L file must match the subinterrupt configuration in the ECU code.

The first subinterrupt handling location starts at `subinterrupts start address`, followed by the other subinterrupt handling locations. One byte is used per subinterrupt. The following illustration shows the subinterrupt memory area for byte-based subinterrupt generation with 16 subinterrupts:



Byte-based subinterrupt handling method

Subinterrupt states For the byte-based subinterrupt mechanism, the subinterrupts can be in one of four states: pending, acknowledged, ready, invalid. The subinterrupt states are coded as follows:

- *Pending*: Subinterrupt N is in the pending state, if byte N of the subinterrupt memory area contains the value 7.
- *Acknowledged*: Subinterrupt N is in the acknowledged state, if byte N of the subinterrupt memory area contains the value 6.
- *Ready*: Subinterrupt N is in the ready state, if byte N of the subinterrupt memory area contains the value 0.
- *Invalid*: Subinterrupt N is in the invalid state, if byte N of the subinterrupt memory area contains the value 4.

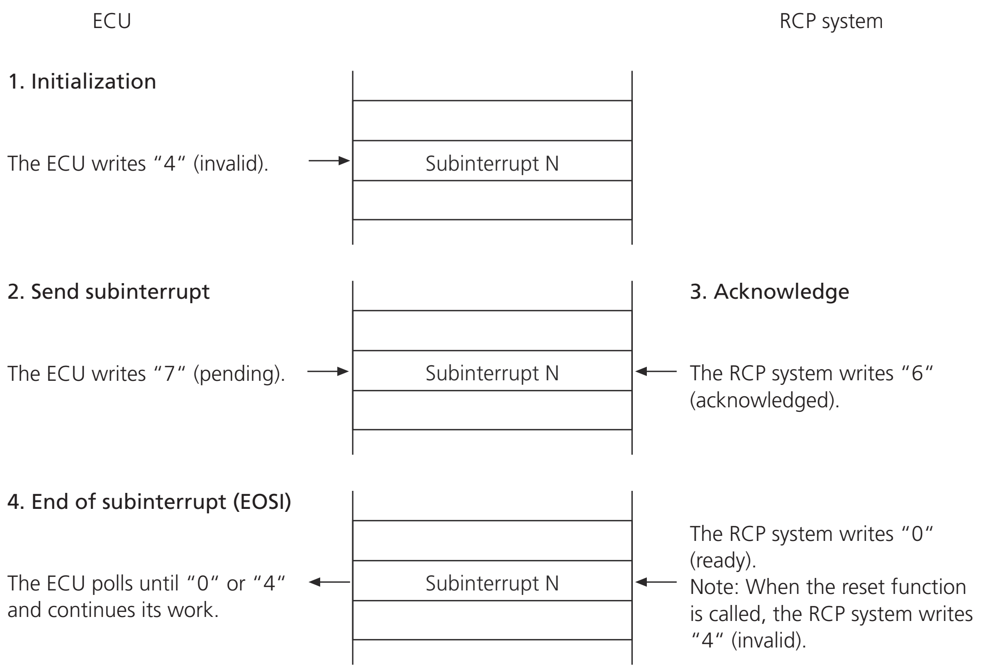
Subinterrupt processing Subinterrupt processing consists of several steps. These are the steps for subinterrupt N (starting from the ready subinterrupt state):

1. The ECU sets the subinterrupt state to pending by writing the value 7 to byte N of the subinterrupt memory area, and triggers a hardware interrupt.
2. After receiving the hardware interrupt, the RCP system acknowledges the subinterrupt by writing the value 6 to byte N of the subinterrupt memory area, and starts processing the tasks of the Simulink subsystem triggered by subinterrupt N.

3. After all the tasks of the interrupt-driven Simulink subsystem connected to the subinterrupt have finished, the RCP system sets the subinterrupt state to ready by writing the value 0 to byte N of the subinterrupt memory area. This is the signal for the ECU that the bypass data can be read. The subinterrupt is ready to be triggered by the ECU again.

At bypassing simulation startup, the RCP system invalidates all subinterrupts by writing the value 4 to all bytes of the subinterrupt memory area.

The following illustration shows the principles of byte-based subinterrupt handling:


ECU
RCP system
1. Initialization
The ECU writes “4“ (invalid).
Subinterrupt N
2. Send subinterrupt
The ECU writes “7“ (pending).
Subinterrupt N
3. Acknowledge
The RCP system writes “6“ (acknowledged).
4. End of subinterrupt (EOSI)
The ECU polls until “0“ or “4“ and continues its work.
Subinterrupt N
The RCP system writes “0“ (ready).
Note: When the reset function is called, the RCP system writes “4“ (invalid).

For the byte-based subinterrupt mechanism, the ECU can trigger subinterrupts if the subinterrupt is in ready state or in invalid state.

Note

After indicating a new subinterrupt, the trigger of the hardware interrupt line is required to indicate the RCP system that a new subinterrupt is available.

Demo ECU application

The RTI Bypass Blockset comes with demo ECU applications containing source code examples for byte-based subinterrupt handling. You can find the demo ECU applications for the different code-patch-based bypass interfaces in `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS`.

You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.

Related topics

Basics

External, Code-Patch-Based Bypassing: Basic Information and Underlying Processes.................. 38

On-Target Bypassing: Bypassing Mechanisms

Introduction

For on-target bypassing, the RTI Bypass Blockset provides different mechanisms to ensure data consistency: the double buffer mechanism, the failure checking mechanism and an interrupt disabling mechanism.

Extended Bypassing Mechanisms for On-Target Bypassing

Introduction

The RTI Bypass Blockset provides the following extended bypassing mechanisms in connection with on-target bypassing:

- Double buffer mechanism
- Failure checking mechanism
- Interrupt disabling mechanism

These mechanisms are interdependent. They are used to ensure operational reliability.

Note

Unlike external bypassing, the extended bypassing mechanisms for on-target service-based bypassing are not supported by the ECU service, but by the RTI Bypass Blockset.

Double buffer mechanism

The double buffer mechanism is used to ensure consistent read/write operations. If the mechanism is enabled, two buffers are used for data exchange between the ECU application RAM/flash (ECU-internal memory areas used by the original ECU application) and the bypass RAM/flash (ECU-internal memory areas reserved for on-target bypassing). For example, when reading data from the bypass RAM/flash, the ECU application continues reading old data (= data already read before) as long as no new data is available from the bypass RAM/flash. This ensures data consistency. If you disable the double buffer mechanism, only one buffer is used for data exchange.

You must enable or disable the double buffer mechanism explicitly in every Read and Write block of the RTI Bypass Blockset. If the double buffer mechanism is disabled, the failure checking mechanism is not available.

Note

It is recommended to enable the double buffer mechanism if the Read, Interrupt and/or Write blocks are configured for different service instances which are placed in different tasks of the ECU application.

Failure checking mechanism

The failure checking mechanism is supported for reading data from/writing data to the ECU application RAM. The failure checking mechanism counts missing data transmissions when the on-target bypass code receives data from the ECU application content or the ECU application code receives data from the on-target bypass code content. The failure number is incremented whenever the ECU receives no new data from the ECU application memory or the bypass RAM/flash. When the failure number reaches the predefined failure limit, the bypass hooks are automatically deactivated. The data calculated by the control algorithm in the original ECU application is no longer overwritten, and the Status port of the Read block provides 'No data copied'. If new data is available, the failure count is reset to 0.

For an on-target bypass scenario, missing data transmissions can occur only if service instances are executed in succession without executing the related service instances in between. This can happen due to a disabled trigger of a service instance or if related service instances are placed in tasks with different periodicity, for example.

Interrupt disabling mechanism

The interrupt disabling mechanism is used to guarantee that a complete data set is consistently read from/written to the ECU memory without being interrupted by a concurrent interrupt service routine.

Related topics

References

Options Page (RTIBYPASS_READ_BLx for INTERNAL)......................................221
Options Page (RTIBYPASS_WRITE_BLx for INTERNAL)....................................222

On-Target Bypassing: Build Options

Introduction

Before you build the on-target bypass code, you must specify some build options for the on-target bypass parts of your Simulink model.

RTI IntByp Build Options Page

Access

This page is contained in the **Code Generation** dialog of the **Configuration Parameters** dialog. You can access the dialog via the **Simulation** ribbon. On the ribbon, click **Prepare – Model settings**.

The **RTI IntByp Build Options** page is available only if **rti_intbyp_<target type>.tlc** is selected as the system target file, where 'target type' stands for the microcontroller type the on-target bypass functions are to be generated for (for example, tricore). The system target file is specified on the **General** page of the **Code Generation** dialog.

Purpose

To specify compiler-specific settings relating to the build process of on-target bypass code.

Description

All the options described below are relevant for models containing only on-target bypass parts.

Note

The **RTI IntByp Build Options** page is available only if Simulink® Coder™ is installed.

- If both Simulink Coder and the TargetLink Code Generator are installed, the settings made on this page are used for both code generators.
- If only the TargetLink Code Generator is installed, the page is not visible. You can still access the parameters by using the Configuration API. Refer to Configuration via API on page 242.

Dialog settings

Compiler arguments Lets you specify the compiler arguments.

- If "Default" is selected, the model is compiled with the compiler-specific default arguments.
- If "User-defined" is selected, you can define the arguments via the Compiler edit field.

Compiler Displays the compiler arguments currently specified for the selected compiler. If **Compiler arguments** is set to **User-defined**, you can enter user-defined compiler arguments.

Linker arguments Lets you specify the linker arguments.

- If "Default" is selected, the model is linked with the compiler-specific default arguments.
- If "User-defined" is selected, you can define the arguments via the Linker flags edit field.

Linker flags Displays the arguments that are currently specified for the linker. If **Linker arguments** is set to **User-defined**, you can enter user-defined linker arguments.

Heap size Lets you specify how much heap RAM memory in byte is reserved for dynamic memory allocation (malloc). The default value is 0, which means that the bypass application's RAM memory usage is minimized, but malloc is not available for the generated code.

Configuration via API

You can also access the configuration settings for the compiler parameters, linker parameters, and the heap size via API.

Compiler parameters To configure the *mode of the optional compiler parameters*, use the following API command:

```
cfgSet = getActiveConfigSet(bdroot);
cfgSet.setProp('CompilerArgumentPopup', <mode_param>);
```

The following parameters are available:

Parameter	Description
mode_param	Mode of the optional compiler parameters. The valid values are: ▪ 'Default' The model is always compiled with the compiler-specific default arguments. Example: `'-static -meabi -mno-version-info -Os -mhard-float'` ▪ 'User-defined' You can define the compiler arguments to be used (see below).

To define the *user-defined compiler parameters*, use the following API command:

```
cfgSet = getActiveConfigSet(bdroot);
cfgSet.setProp('DSIntBypCompilerOpts', <optional_compiler_param>);
```

For information on the possible `optional_compiler_param` parameters, refer to the documentation of your compiler.

Linker parameters To configure the *mode of the optional linker parameters*, use the following API command:

```
cfgSet = getActiveConfigSet(bdroot);
cfgSet.setProp('DSIntBypLinkerOptsPopup', <mode_param>);
```

The following parameters are available:

Parameter	Description
mode_param	Mode of the optional linker parameters. The valid values are: ▪ 'Default' The model is always linked with the compiler-specific default arguments. Example: `'-Wl,-e,0x0 -Wl,--oformat,elf32-tricore -Wl,--no-warn-mismatch -Wl,-lm -Wl,-lnosys -Wl,-lgccoptfp -Wl,-lc'` ▪ 'User-defined' You can define the linker arguments to be used (see below).

To define the *user-defined linker parameters*, use the following API command:

```
cfgSet = getActiveConfigSet(bdroot);
cfgSet.setProp('DSIntBypLinkerOpts', <optional_linker_param>);
```

For information on the possible `optional_linker_param` parameters, refer to the documentation of your compiler.

Heap size To configure the heap size, use the following API command:

```
cfgSet = getActiveConfigSet(bdroot);
cfgSet.setProp('IntBypHeapSize', <heap_size_param>);
```

The following parameters are available:

Parameter	Description
heap_size_param	Heap RAM memory size in bytes. You must enter a hexadecimal value using the '0x' or '0X' notation (e.g., '0x1000' or '0X1000').

Related topics

Basics

On-Target, Service-Based Bypassing: Basic Information and Underlying Processes..................... 41

On-Target Bypassing: Adapting ECU-Internal Functions that are not Compliant with the EABI

Introduction

To call an ECU-internal function that is not compliant with the EABI standard in an on-target bypass model, you must define an appropriate adapter function for the ECU function.

Function Adaptation of Non-EABI-Compliant ECU Functions

Introduction

The RTIBYPASS_FUNCTION_BLx block allows you to call an ECU function that already exists in the ECU code within the on-target bypass model. ECU-internal functions that are compliant with the microcontroller-related EABI standard are supported by the RTI Bypass Blockset without further configurations. ECU-internal functions that are not compliant with the EABI standard must be adapted via adapter function.

Tip

The RTI Bypass Blockset comes with a demo model for using the Function block. The demo includes examples of calling non-EABI-compliant ECU functions by the Function block. Refer to the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS` folder.

You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.

Supported EABI standards

The RTI Bypass Blockset supports ECU-internal functions that are compliant with the following embedded application binary interface (EABI) standards:

Target Processor Family	EABI Standard
TriCore	TriCore® 1 32-bit Unified Processor Core Embedded Applications Binary Interface (EABI)
MPC5xxx	PowerPC^{TM} e500 Application Binary Interface
RH850X	RH850 Compiler ABI Specification
ARM	Procedure Call Standard for the ARM Architecture (AAPCS)

For each available ECU-internal function that is to be called by the Function block, but that is not compliant with the microcontroller-related EABI standard, an appropriate adapter function defined in an external source file must be assigned to the ECU function. The assignment is made in the `IF_DATA dSPACE_INTERNAL_BYPASS` entry, namely in the `FUNCTION_PROTOTYPE` element, which describes the ECU-internal function, via the optional `FUNCTION_ADAPTATION` member.

Tip

For details on the required interface-specific information, refer to IF_DATA Definition for Internal Bypassing (Interface Description Data Reference 🕮).

Implementing adapter functions

If you want to use a non-EABI-compliant ECU-internal function in an on-target bypass model, you first must implement an appropriate adapter function for the ECU function and then assign this adapter function to the ECU function. To implement an adapter function, you must be familiar with the noncompliant ECU function and know how to call it.

There are two types of adapter functions that can be used with non-EABI-compliant ECU functions:

- A *specific adapter function* is defined individually for one ECU function.
- A *generic adapter function* can be used with several different ECU-internal functions. The generic adapter function is implemented for a variable number or parameters. To identify the different function calls handled by the same adapter implementation, an internal identifier can be specified in the `IF_DATA dSPACE_INTERNAL_BYPASS` entry.

The adapter function (generic or specific) must be defined in a C source file. This C file must then be added to the model as an external source file to make the adapter function implementation available for the build process. A source file can contain several adapter functions.

Structure of adapter functions

An adapter function must have a particular structure. This includes the requirement that each adapter function (specific or generic) must use the following function parameters:

- `id`
- `address`
- `return_pointer`
- `num`

This parameter list then can be supplemented by further parameters individually for each adapter function.

Structure of a specific adapter function The following listing shows the principle structure of an adapter function for an ECU function with two parameters of integer and short integer data type. It shows the implementation for an Infineon TriCore microcontroller architecture.

```
#include <stdarg.h>
double return_value1;
int adapterfunction(unsigned int id, unsigned int address, void **return_pointer, int num, int a, short b)
{
   asm("mov.d      %d7,%a10      #Backup SP");
   asm("sub.a      %a10,4        #Decrement SP");
   asm("st.w       [%%a10],%0    #Store on stack" :: "r"(a));
   asm("sub.a      %a10,4        #Decrement SP");
   asm("st.w       [%%a10],%0    #Store on stack" :: "r"(b));
   asm("mov.a      %%a7,%0       #Get address argument" :: "r"(address) : "%%a7");
   asm("jli        %a7           #Jump to function");
   asm("ld.w       %0,[%%a10]    #Get return value from stack" :"=r"(return_value1));
   asm("mov.a      %a10,%d7      #Restore SP");

   *return_pointer = &return_value1;
return 1;
}
```

Structure of a generic adapter function The following listing shows the principle structure of a generic adapter function. It shows an assembler code sequence storing a dedicated number of function arguments on the stack and getting the return value from the stack. It shows the implementation for an Infineon TriCore microcontroller architecture.

```
#include <stdarg.h>
double return_value2;
int adapterfunction_generic(unsigned int id, unsigned int address, void **return_pointer, int num, ...)
{
   va_list arguments;
   va_start ( arguments, num );

   switch (id)
   {
      case 0:
      {
         int Saw1MinParam = va_arg ( arguments, signed int);
         char Saw1MaxParam = va_arg ( arguments, int);

         asm("mov.d      %d7,%a10      #Backup SP");
         asm("sub.a      %a10,4        #Decrement SP");
         asm("mov        %%d3,%0       #Get argument 1" :: "r"(Saw1MinParam) : "%%d3");
         asm("st.w       [%a10],%d3    #Store on stack");
         asm("sub.a      %a10,4        #Decrement SP");
         asm("mov        %%d3,%0       #Get argument 2" :: "r"(Saw1MaxParam) : "%%d3");
         asm("st.w       [%a10],%d3    #Store on stack");
         asm("mov.a      %%a7,%0       #Get address argument" :: "r"(address) : "%%a7" );
         asm("jli        %a7           #Jump to function");
         asm("ld.w       %0,[%%a10]    #Get return value from stack" :"=r"(return_value2));
         asm("mov.a      %a10,%d7      #Restore SP");

         *return_pointer = &return_value2;
      }
   }
   va_end ( arguments );
return 1;
}
```

Tip

You can find a source file with adapter function implementations in the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS` folder.

You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.

Related topics

References

Variables Page (RTIBYPASS_FUNCTION_BLx)... 143

On-Target Bypassing: Dynamically Integrating a Simulink or TargetLink Model into a V-ECU

Introduction

You can use RTI Bypass Blockset to integrate a Simulink® or TargetLink model dynamically into a V-ECU. V-ECU-internal variables and functions can be accessed by the Simulink or TargetLink model for ECU interfacing purposes.

Where to go from here

Information in this section

Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on VEOS

Introduction

You can use RTI Bypass Blockset to integrate a Simulink® or TargetLink model dynamically into a V-ECU executed in a VEOS offline simulation. V-ECU-internal variables and functions can be accessed by the Simulink or TargetLink model for ECU interfacing purposes.

Advantages of accessing V-ECU-internal variables and functions by a Simulink or TargetLink model

- Accessing V-ECU-internal variables and functions by a Simulink or TargetLink model lets you perform tasks such as rapid control prototyping by implementing a function bypass based on that V-ECU.
- Accessing V-ECU-internal variables and functions is possible even if the V-ECU software architecture must not be modified, and if the V-ECU is available only as a binary file.

Integrating the Simulink or TargetLink model

V-ECU implementations with Rapid Prototyping Access (RptAccess) module and Rapid Prototyping Service Interface (RptSi) module If a V-ECU implementation contains the Rapid Prototyping Access (RptAccess) module and the Rapid Prototyping Service Interface (RptSi) module, the dSPACE Internal Bypassing Service is integrated in the V-ECU. This lets you use the RTI Bypass Blockset to integrate a Simulink® or TargetLink model dynamically into a V-ECU.

Implementing the Simulink or TargetLink model The V-ECU's A2L file created during the V-ECU build process is the starting point for implementing the Simulink or TargetLink model with the RTI Bypass Blockset. For this purpose, you have to import the A2L file to the RTIBYPASS_SETUP_BLx block as a new database. Refer to Unit Page (RTIBYPASS_SETUP_BLx) on page 53.

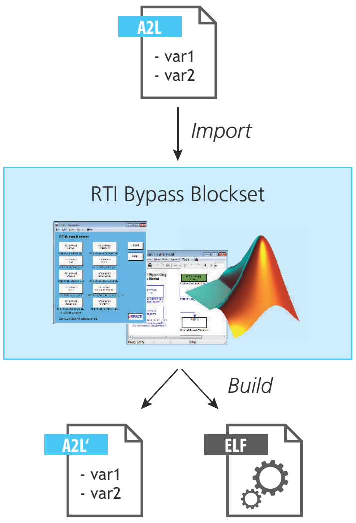
Information on dSPACE Internal Bypassing Service calls required by the RTI Bypass Blockset is contained in the A2L file's `IF_DATA` section.

After you import the A2L file, you can model the V-ECU access with the RTI Bypass Blockset as usual.

Building the Simulink or TargetLink model The build process for the Simulink or TargetLink model mainly creates the following files:

- An *Executable and Linkable Format* (ELF) file containing the model code
- The V-ECU's A2L file extended by variables of the model code (A2L')

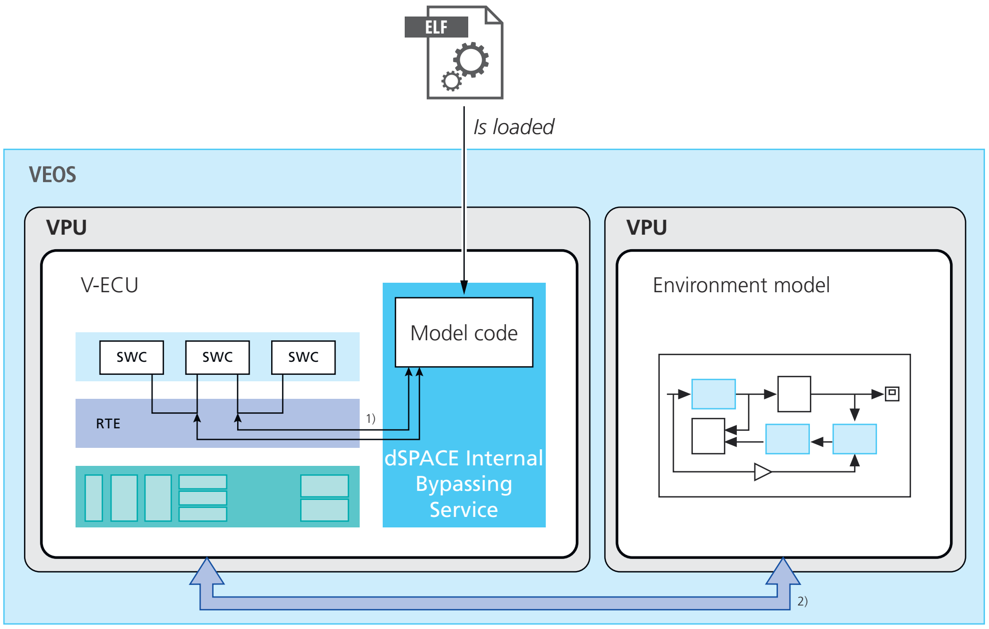
The following illustration shows this step:



Loading the Simulink or TargetLink model code to the V-ECU There are two ways to load the generated ELF file to the V-ECU:

- You can have the ELF file loaded to the V-ECU automatically after the build process. Refer to Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 📖).
- You can load the ELF file by using the `downIntBypService.exe` command line tool. Refer to Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) on page 204.

Simulation system The following illustration shows the simulation system for VEOS with integrated model code and with a connected environment model:



[1] Access to all A2L variables of the V-ECU code.
[2] VPU connection

The model code is a functional extension to the V-ECU.

Variable accesses and code executions can be synchronized with service points configured in the Rapid Prototyping Access (RptAccess) module of the V-ECU. The model code has access to all A2L variables of the V-ECU code.

Related topics

Basics

Basics on Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing (TargetLink Interoperation and Exchange Guide 🕮)
Implementing On-Target ECU Interfacing with RTI Bypass Blockset (ECU Interfacing Overview 🕮)
Rapid Prototyping Access to V-ECU-Internal Variables and Functions (SIL Testing Overview 🕮)

Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on a SCALEXIO System or on a MicroAutoBox III

Introduction

You can use the RTI Bypass Blockset to integrate a Simulink® or TargetLink model dynamically into a V-ECU executed in a SCALEXIO or MicroAutoBox III real-time application. V-ECU-internal variables and functions can be accessed by the Simulink or TargetLink model for ECU interfacing purposes.

Advantages of accessing V-ECU-internal variables and functions by a Simulink or TargetLink model

- Accessing V-ECU-internal variables and functions by a Simulink or TargetLink model lets you perform tasks such as rapid control prototyping by implementing a function bypass based on that V-ECU.
- Accessing V-ECU-internal variables and functions is possible even if the V-ECU software architecture must not be modified, and if the V-ECU is available only as a binary file.

Integrating the Simulink or TargetLink model

V-ECU implementations with Rapid Prototyping Access (RptAccess) module and Rapid Prototyping Service Interface (RptSi) module If a V-ECU implementation contains the Rapid Prototyping Access (RptAccess) module and the Rapid Prototyping Service Interface (RptSi) module, the dSPACE Internal Bypassing Service is integrated in the V-ECU. This lets you use the RTI Bypass Blockset to integrate a Simulink® or TargetLink model dynamically into a V-ECU.

Implementing the Simulink or TargetLink model The V-ECU's A2L file created during the V-ECU build process is the starting point for implementing the Simulink or TargetLink model with the RTI Bypass Blockset. For this purpose, you have to import the A2L file to the RTIBYPASS_SETUP_BLx block as a new database. Refer to Unit Page (RTIBYPASS_SETUP_BLx) on page 53.

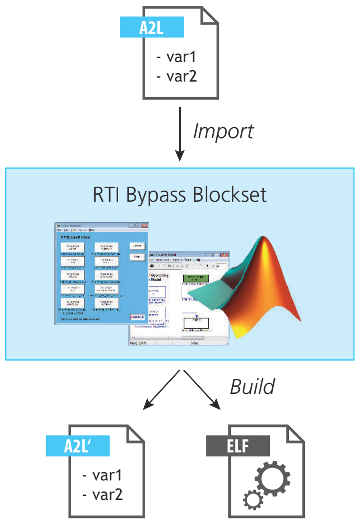
Information on dSPACE Internal Bypassing Service calls required by the RTI Bypass Blockset is contained in the A2L file's `IF_DATA` section.

After you import the A2L file, you can model the V-ECU access with the RTI Bypass Blockset as usual.

Building the Simulink or TargetLink model The build process for the Simulink or TargetLink model mainly creates the following files:

- An *Executable and Linkable Format* (ELF) file containing the model code
- The V-ECU's A2L file extended by variables of the model code (A2L')

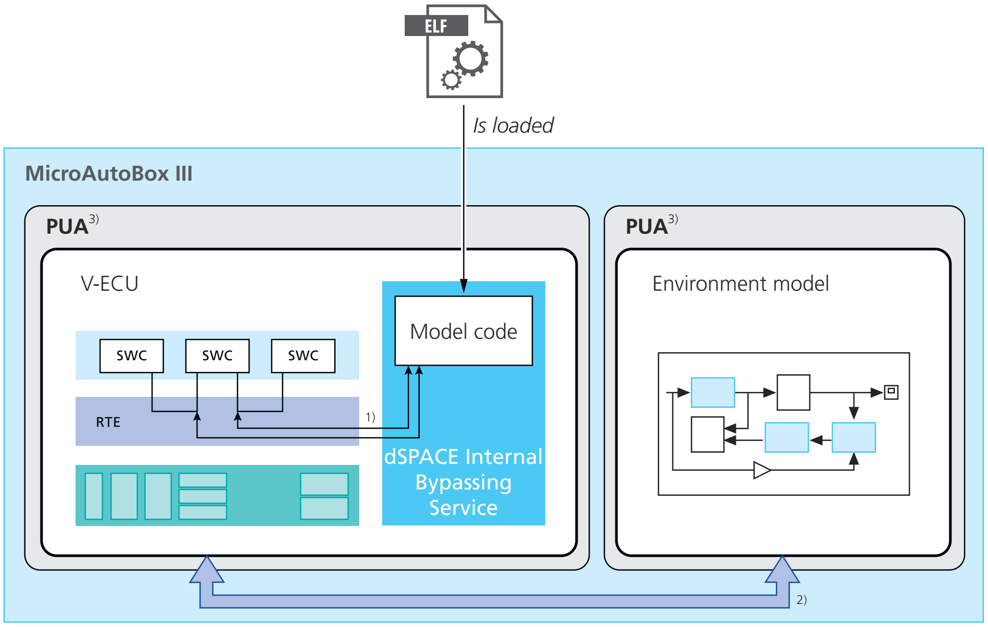
The following illustration shows this step:



Loading the Simulink or TargetLink model code to the V-ECU There are two ways to load the generated ELF file to the V-ECU:

- You can have the ELF file loaded to the V-ECU automatically after the build process. Refer to Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).
- You can load the ELF file by using the `downIntBypService.exe` command line tool. Refer to Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) on page 204.

Simulation system The following illustration shows an example of a simulation system executed on a MicroAutoBox III. The simulation system contains a V-ECU with integrated model code and an environment model connected to that V-ECU:



1) Access to all A2L variables of the V-ECU code.
2) Connection
3) PUA = processing unit application

The model code is a functional extension to the V-ECU.

Variable accesses and code executions can be synchronized with service points configured in the Rapid Prototyping Access (RptAccess) module of the V-ECU. The model code has access to all A2L variables of the V-ECU code.

Related topics

Basics

Basics on Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing (TargetLink Interoperation and Exchange Guide 📖)
Implementing On-Target ECU Interfacing with RTI Bypass Blockset (ECU Interfacing Overview 📖)
Rapid Prototyping Access to V-ECU-Internal Variables and Functions (SIL Testing Overview 📖)
Working with V-ECU Implementations That Provide Virtual Bypassing Support (ConfigurationDesk Real-Time Implementation Guide 📖)

Message Reader API

Introduction

You can use the Message Reader API to access all binary message log files of the dSPACE Log.

Where to go from here

Information in this section

Introduction to the Message Reader API

Where to go from here

Information in this section

Reading dSPACE Log Messages via the Message Reader API

Introduction

You can read log messages of the dSPACE Log via the Message Reader API.

dSPACE Log

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

The dSPACE Log is saved as a collection of binary message log files. These files are created when a dSPACE product is running. A single run of a dSPACE product is called a *log session*.

Note

If the maximum file size for the binary message log file is reached, messages at the beginning of the dSPACE Log might get deleted. Contact dSPACE Support to solve this.

Message Reader API

You can use the Message Reader API to access all binary message log files of the dSPACE Log. You can combine multiple filters to display only log messages according to your specifications. For example, you can configure the Message Reader API to display only log messages from a specific dSPACE product.

The Message Reader API is available as of dSPACE Release 2020-A. For information on the dSPACE products and components that support the Message Reader API, refer to Supported dSPACE Products and Components on page 256.

dSPACE.Common.MessageReader.dll The Message Reader API is implemented by the `dSPACE.Common.MessageReader.dll` file. It is located in the `bin` subfolder of the installation folder of each dSPACE product that supports the Message Reader API.

Supported dSPACE Releases

The Message Reader API lets you access log messages written by dSPACE products since dSPACE Release 2016-B.

Message Reader API change in dSPACE Release 2021-A

There is a migration issue specific to the Message Reader API. The issue occurs if you use the API with Python. The issue was caused by the migration to Python 3.9/pythonnet 2.5.3 with dSPACE Release 2021-A.

There is no migration issue to consider if you use the API with C#.

Specifying a product filter As of dSPACE Release 2021-A, the `Products` property of the `MessageReaderSettings` class can no longer be used to set the list of products for which to filter in the log sessions. The Message Reader API provides the `SetProducts` method for this purpose.

The following table shows how to specify a product filter before and after migration:

Using Message Reader API of ...	
... dSPACE Release 2020-B and Earlier (Python 3.6)	**... dSPACE Release 2021-A and Later (Python 3.9)**

`# Specify products whose messages to read:`
`Settings = MessageReaderSettings()`
`Settings.Products.Add('ControlDesk')`
`Settings.Products.Add('AutomationDesk')`
	`# Specify products whose messages to read:`
	`Settings = MessageReaderSettings()`
	`Settings.SetProducts(['ControlDesk', 'AutomationDesk'])`

Related topics

Basics

Examples

References

Supported dSPACE Products and Components

Supported dSPACE products and components

You can use the Message Reader API to access messages from the following dSPACE products and components:

- ASM KnC
- AutomationDesk
- Bus Manager (stand-alone)
- cmdloader
- ConfigurationDesk
- Container Management
- ControlDesk
- dSPACE AUTOSAR Compare
- dSPACE XIL API .NET Implementation
- Firmware Manager
- ModelDesk
- MotionDesk
- Real-Time Testing
- RTI Bypass Blockset
- SYNECT client

- SystemDesk
- TargetLink Property Manager
- VEOS

Related topics

Basics

Example of Reading Messages with Python

Introduction

You can read the log messages via Python by using the `clr` module. You can combine multiple filters to display only messages according to your specifications.

Referencing a message reader assembly

You have to reference a `dSPACE.Common.MessageReader.dll` assembly. For information on the location of the assembly, refer to dSPACE.Common.MessageReader.dll on page 255.

In the following examples it is assumed that the dSPACE Installation Manager is installed and that the message reader assembly is installed in `C:\Program Files\Common Files\dSPACE\InstallationManager\bin`.

The following code references and imports the message reader assembly.

```
# Insert path of message log file access assembly:
import sys
AssemblyPath = r'C:\Program Files\Common Files\dSPACE\InstallationManager\bin'
if not sys.path.count(AssemblyPath):
    sys.path.insert(1, AssemblyPath)

# Add reference to assembly and import it:
import clr
clr.AddReference('dSPACE.Common.MessageReader')
from dSPACE.Common.MessageHandler.Logging import *
```

Reading all messages

The following example reads all existing message log files and prints all messages via Python. It is assumed that the message reader assembly is referenced and imported. Refer to Referencing a message reader assembly on page 257.

```
# Create message reader and print text of each message:
Reader = MessageReader(None)
for Message in Reader.ReadMessages():
    print(Message.MessageText)
Reader.Dispose()
```

Filtering messages by severity, product, and session

The following example reads and prints messages with a severity of `Error`, `SevereError`, or `SystemError`. Also, only messages of the last sessions of ControlDesk and AutomationDesk are read and printed. It is assumed that the message reader assembly is referenced and imported. Refer to Referencing a message reader assembly on page 257.

```
# Define error severities:
SEVERITY_ERROR = 3
SEVERITY_SEVERE_ERROR = 4
SEVERITY_SYSTEM_ERROR = 5

# Configure products and sessions whose messages to read:
Settings = MessageReaderSettings()
Settings.MaximalSessionCount = 1
Settings.SetProducts(['ControlDesk', 'AutomationDesk'])

# Create message reader and print text of each error message:
Reader = MessageReader(Settings)
for Message in Reader.ReadMessages():
    # Print error messages only:
    if Message.Severity == SEVERITY_ERROR or \
       Message.Severity == SEVERITY_SEVERE_ERROR or \
       Message.Severity == SEVERITY_SYSTEM_ERROR:
        print('%s: %s' % (Message.Session.ProductName, Message.MessageText))
Reader.Dispose()
```

Note

The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time. Refer to MessageReader Class on page 264.

Filtering messages by time

Times are given by .NET DateTime objects. Times are given as UTC times (Coordinated Universal Time). You can obtain the current UTC time by `System.DateTime.UtcNow`.

The following example reads all messages after a certain start time. It is assumed that the message reader assembly is referenced and imported. Refer to Referencing a message reader assembly on page 257.

```
import System
Settings = MessageReaderSettings()
Settings.MessageTimeAfter = System.DateTime.UtcNow # Read messages after now

# Create message reader and print time and text of each message:
Reader = MessageReader(Settings)
for Message in Reader.ReadMessages():
    print('%s: %s' % (Message.UtcTimeStamp, Message.MessageText))
Reader.Dispose()
```

Related topics

Basics

References

Example of Reading Messages with C#

Introduction

You can read the log messages via C#. You can combine multiple filters to display only messages according to your specifications.

Referencing a message reader assembly

You have to reference a `dSPACE.Common.MessageReader.dll` assembly. For information on the location of the assembly, refer to dSPACE.Common.MessageReader.dll on page 255.

Reading all messages

The following example reads all existing message log files and prints the messages:

```
using dSPACE.Common.MessageHandler.Logging;
...

// Create message reader and print text of each message:
using (MessageReader reader = new MessageReader(null))
{
    foreach (message in reader.ReadMessages())
    {
        Console.WriteLine(message.MessageText);
    }
}
```

Filtering messages by severity, product, and session

The following example reads and prints messages with a severity of `Error`, `SevereError`, or `SystemError`. Also, only messages of the last sessions of ControlDesk and AutomationDesk are read and printed.

```
using dSPACE.Common.MessageHandler.Logging;
...

// Read the last log sessions of ControlDesk and AutomationDesk only:
MessageReaderSettings settings = new MessageReaderSettings();
settings.MaximalSessionCount = 1;
settings.Products.Add("ControlDesk");
settings.Products.Add("AutomationDesk");

using (MessageReader reader = new MessageReader(settings))
{
    foreach (ILogMessage message in reader.ReadMessages())
    {
        // Print error messages only:
        if (message.Severity == Severity.Error
            || message.Severity == Severity.SevereError
            || message.Severity == Severity.SystemError)
        {
            Console.WriteLine(message.Session.ProductName + ": " + message.MessageText);
        }
    }
}
```

Note

The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time. Refer to MessageReader Class on page 264.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API........254
Supported dSPACE Products and Components........256

References

MessageReaderSettings Class........265

dSPACE.Common.MessageHandler.Logging Reference

Where to go from here

Information in this section

ILogMessage Interface

Namespace `dSPACE.Common.MessageHandler.Logging`

Description To access information about a message as written to a log file.

Properties The element has the following properties:

Name	Description	Get/Set	Type
IsStartMessage	Gets a value indicating whether the message is a session start message.	Get	*Boolean*
IsStopMessage	Gets a value indicating whether the message is a session stop message.	Get	*Boolean*
MainModuleNumber	Gets the main module number of the message.	Get	*Integer*
MessageCode	Gets the error code of the message.	Get	*Integer*
MessageText	Gets the text of the message.	Get	*String*
ModuleName	Gets the module name of the message.	Get	*String*
Session	Gets the log session which issued the message.	Get	ILogSession (refer to ILogSession Interface on page 262)
Severity	Gets the severity of the message.	Get	Severity (refer to Severity Enumeration on page 267)
SubmoduleNumber	Gets the submodule number of the message.	Get	*Integer*
ThreadId	Gets the thread ID of the submitting thread.	Get	*Integer*
TimeStamp	Gets the time when the message was submitted. Given as local time in the time zone of the session.	Get	*DateTime*
UtcTimeStamp	Gets the time when the message was submitted in UTC time.	Get	*DateTime*

Methods

The element has no methods.

Related topics

Basics

Examples

References

ILogSession Interface

Namespace

`dSPACE.Common.MessageHandler.Logging`

Description

To access information about a message log session.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
CloseTime	Gets the time when the session was closed. Returns an undefined time (0, DateTimeKind.Unspecified) if the session is still open or was not closed successfully. Given as local time in the time zone of the session.	Get	*DateTime*
IsOpen	Gets a value indicating whether the session is still open. If true, the session is still open and new messages can be written.	Get	*Boolean*
IsValid	Gets a value indicating whether the session is valid. A session can become invalid if its log files are corrupted.	Get	*Boolean*
MetaData	Gets the products metadata as read from log file session info.	Get	*Dictionary< String, String >*
ProcessId	Gets the process ID of the log session.	Get	*Integer*

Name	Description	Get/Set	Type
ProductName	Gets the product name of the log session.	Get	*String*
SessionId	Gets the ID of the log session. This ID is unique in the context of its session reader.	Get	*Integer*
StartTime	Gets the sessions start time. Given as local time in the time zone of the session.	Get	*DateTime*
TimezoneName	Gets the standard time zone name of the session.	Get	*String*
TimezoneOffset	Gets the time zone offset of the session relative to UTC.	Get	*TimeSpan*
UtcCloseTime	Gets the time when the session was closed as UTC time. Returns an undefined time (0, DateTimeKind.Unspecified) if the session is still open or was not closed successfully.	Get	*DateTime*
UtcStartTime	Gets the start time of the log session as UTC time.	Get	*DateTime*

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
ToSessionTime	Converts UTC time to time zone used when the session was written.	▪ <*DateTime*> `utcTime`: Specifies the UTC time to convert.	Time in the time zone of the logging session. ▪ DateTime

[1] <Type> `Name`: Description

Related topics

Basics

Examples

MessageReader Class

Description

To read serialized messages written by dSPACE products.

Constructor

The element has the following constructor:

Name	**Description**	**Parameter**[1]	**Returns**
MessageReader	Initializes a new instance of the MessageReader class.	▪ <MessageReaderSettings>[2] `settings`: Settings which allow to specify which sessions and messages are read. Can be null, causing all existing log files to be read.	None

[1] <Type> `Name`: Description

[2] Refer to MessageReaderSettings Class on page 265

Properties

The element has no properties.

Methods

The element has the following methods:

Name	**Description**	**Parameter**[1]	**Returns**
Dispose	Performs application-specific tasks associated with freeing, releasing, or resetting unmanaged resources.	None	None

Name	Description	Parameter[1)]	Returns
ReadMessages	Reads the messages written to the log files of the sessions up to now. The messages are returned in chronological order according to their time stamps. **Note** The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time.	None	Messages read from log file. ▪ IEnumerable< ILogMessage (refer to ILogMessage Interface on page 261) >

1) <Type> `Name`: Description

Related topics

Basics

Examples

References

MessageReaderSettings Class

Description

To define the settings of a message reader.

Used to filter the log sessions and messages read.

Constructor The element has the following constructor:

Name	Description	Parameter[1)]	Returns
MessageReaderSettings	Initializes a new instance of the MessageReaderSettings class.	None	None

1) <Type> `Name`: Description

Properties The element has the following properties:

Name	Description	Get/Set	Type
DirectoryNames	Gets a list of specific directory names from which to read log files. If the list is empty, all standard directories are searched for log files.	Get	*List< String >*
MaximalSessionCount	Gets or sets the maximal number of log sessions read for each product. If the count is a positive number n, only the last n sessions are read. If the count is not positive, an unlimited number of sessions is read. The default value is zero, i.e., unlimited.	Get/Set	*Integer*
MessageTimeAfter	Gets or sets the minimal time for which messages are read, given as UTC time. Only messages submitted after the message time are read. The message time may be in the past. The message time must be given as valid UTC time. The default time is undefined, i.e., each message time is allowed.	Get/Set	*DateTime*
Products	Gets the list of product names for which to read log sessions. If the list is empty sessions of all products are read.	Get	*List< String >*
StartTimeAfter	Gets or sets the minimal start time for which sessions are read, given as UTC time. Only sessions which started after the start time are read. The start time may be in the past. The start time must be given as valid UTC time. The default time is undefined, i.e., each start time is allowed.	Get/Set	*DateTime*

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
SetDirectoryNames	Sets the list of specific directory names from which to read log files. You do not have to specify a list. If the list is empty, all standard directories are searched for log files.	<*string[]*> `names`: Array of directory names.	None
SetProducts	Sets the list of product names for which to read log sessions.	<*string[]*> `products`: Array of product names.	None

[1] <Type> `Name`: Description

Related topics

Basics

Examples

Severity Enumeration

Description

To specify the severity of a message.

Enumeration values

The enumeration has the following values:

Value	Name	Description
0	Trace	A trace message. Trace messages are usually not created. It depends on the host application if it is possible to configure the message handler to create trace messages.
1	Info	An information message.
2	Warning	A warning message.
3	Error	An error message.
4	SevereError	A severe error message.
5	SystemError	A system error message.
6	Question	A question message.
7	Advice	An advice message.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..254

Examples

Example of Reading Messages with C#..259
Example of Reading Messages with Python..257

Limitations

Limitations of the RTI Bypass Blockset

Introduction

When working with the RTI Bypass Blockset, you have to consider some limitations.

Limitations regarding plug-on devices

- The RTI Bypass Blockset with the dSPACE Calibration and Bypassing Service is limited to plug-on devices (PODs) and the DCI-GSI2 for the DS4121 ECU Interface Board and for the MicroAutoBox II.
- The RTI Bypass Blockset supports only dual-port memory-based (DPMEM-based) PODs and the DCI-GSI2.
- The RTI Bypass Blockset does not support other debug port-based PODs.

Limitations regarding computation methods

- When data is read from the ECU, the `RAT_FUNC` conversion method is limited to linear formulas.
- When data is written to the ECU, the `TAB_INTP` and `TAB_NOINTP` conversion methods are limited to strictly monotone value sequences.
- The `FORM` conversion method can be used with formulas based on ASAM MCD-2 MC Ver. 1.6 or later.
- With the `FORM` conversion method, the following operators within the formulas are not supported: '&', '|', '>>', '<<', 'XOR', '~'.

Limitations regarding supported variable types

- The RTI Bypass Blockset only supports parameters, measurements, measurement arrays (measurement variables with `ARRAY_SIZE` or `MATRIX_DIM` parameter set) and value blocks (`CHARACTERISTIC` of `VAL_BLK` type).
- The RTI Bypass Blockset does not support variables of the Int64, UInt64, and Float64 (Double) data type in Read, Write, Upload, and Download blocks.

Limitations regarding masked variables

The RTI Bypass Blockset supports only contiguous bit masks. For example, the 0x70 bit mask is supported (0x70 corresponds to 1110000 in binary notation), but the 0x33 bit mask is not supported (0x33 corresponds to 110011 in binary notation).

Limitations regarding XCP

- The RTI Bypass Blockset is limited to byte granularity for addresses.
- The granularity of ODT entry size of DAQ lists with DAQ direction and DAQ lists with STIM direction is limited to byte size.

- The RTI Bypass Blockset is limited to the following ODT entry optimization types:
  - DEFAULT
  - ODT_TYPE_32
  - ODT_TYPE_64
  - ODT_TYPE_ALIGNMENT
- The RTI Bypass Blockset supports only the ADDRESS_EXTENSION_FREE address extension type.

Limitations regarding CAN FD

- The RTI Bypass Blockset does not check whether the specified CAN FD configuration is compatible to the hardware environment used. This applies regardless of whether the CAN FD configuration settings were set via user interface, automation interface, or A2L file parser.
- If an imported database file contains descriptions in the CAN FD format but the assigned CAN interface does not support CAN FD, the interface operates in classic CAN mode only.
- The RTI Bypass Blockset does not support the following parameters for CAN FD configurations:
  - `bit timing`
  - `BTL_CYCLES`
  - `SJW`
  - `SYNC_EDEG`
  - `SECONDARY_SAMPLE_POINT`
  - `TRANSCEIVER_DELAY_COMPENSATION`

  If these parameters are set, the RTI Bypass Blockset ignores the specified parameter values and uses default values instead.
- The following parameters for CAN FD configurations are not supported by the RTI Bypass Blockset MATLAB API:
  - `MAX_DLC`
  - `MAX_DLC_REQUIRED`

Limitations regarding the RTIBYPASS_CAL_PAGE_SWITCH_BLx block

- The XCP service might not allow switching the active page if one of the following conditions is met:
  - Not all the segments have two pages.
  - The page number to be switched to is not available for all segments.

  Depending on how tolerant the ECU responds to the attempt, the RTIBYPASS_CAL_PAGE_SWITCH_BLx block might not perform a page switch in these cases.
- Switching to a dedicated working page is not possible if multiple working pages are specified for a segment.

Limitations regarding the RTIBYPASS_GATEWAY_BLx block

- The gateway functionality is supported only for XCP on CAN.
- The XCP gateway functionality is available only for static DAQ lists.
- Packet identifier (PID) transmission must be enabled.

- Identical CAN IDs must be used for processing XCP commands and performing data acquisition (DAQ) and stimulation (STIM).
- One DAQ list cannot be used by two external tools at the same time. Only one external tool can perform DAQ at a time. If the RTI Bypass Blockset and an external calibration tool are used at the same time, make sure that the DAQ list number specified in the Read and Write blocks does not collide with the DAQ list used by the external tool.
- Using the XCP gateway functionality for the external tools is possible only if the RCP application is running.

Limitations on writing bit variables

The RTIBYPASS_DOWNLOAD_BLx block does not support writing bit variables for DCI-GSI2-based bypass interfaces and the CCP interface.

Limitations on bypassing via CCP and XCP on CAN

If you perform bypassing via CCP or XCP on CAN, do not use the same CAN controller for the RTI Bypass Blockset and the RTI CAN Blockset. The same applies to the RTI Bypass Blockset and the RTI CAN MultiMessage Blockset, which must also each use a separate controller.

Limitations on on-target bypassing

- For on-target bypassing, the RTI Bypass Blockset does not support block functionalities that depend on an absolute time. Therefore the following MATLAB Simulink blocks are not supported or their functionalities are limited:
  - Backlash
  - ChirpSignal
  - Clock
  - Derivative
  - Digital Clock
  - Discrete-Time Integrator (if used in triggered subsystems)
  - Pulse Generator (only if the Pulse type parameter is set to `Time-based`)
  - Ramp
  - Rate Limiter
  - Repeating Sequence
  - Signal Generator
  - SineWave (only if the Sine type parameter is set to `Time-based`)
  - Stateflow (only if the chart uses the reserved word `t` to reference time)
  - Step
  - Transport Delay
  - Variable Time Delay
  - Variable Transport Delay

  For a list of all the blocks that depend on the absolute time, refer to the Simulink documentation from MathWorks® or the blockset documentation of your third-party libraries.

- The RTI Bypass Blockset lets you specify to convert double to IEEE 754 single-precision float. However, there are some limitations to note:
  - The **Convert Double to Single Float Precision (IEEE754)** option cannot be used in combination with the external variable feature. If this option is enabled, the external variable feature cannot be activated. This is due to the fact that a 32-bit address cannot be stored in a single-precision float without precision loss.
  - The **Convert Double to Single Float Precision (IEEE754)** option is also not available for the INTERNAL interface in combination with ARM, X86, and V850X targets.
- The RTI Bypass Blockset does not support Simulink model referencing. As a consequence, all the RTI Bypass blocks must be located in the same model.
- With the TargetLink Code Generator selected as the code generator, the RTI Bypass Blockset does not support external variable configuration. If external variable definition is enabled in any Read, Write, Upload or Download block, the build process aborts with an error.
- If the TargetLink Code Generator is used, the on-target bypass model must contain only Simulink blocks that are supported by the current TargetLink version for production code generation. If the model contains unsupported Simulink blocks, the build process fails. For an overview of the TargetLink supported Simulink blocks and Stateflow objects, refer to the TargetLink Model Element Reference 🕮. For information on how to add support for unsupported blocks, refer to Overview of Methods for Preparing Unsupported Simulink Blocks for TargetLink Code Generation (TargetLink Preparation and Simulation Guide 🕮).
- TargetLink does not support the Simulink Rate Transition block with deterministic data transfer, which is used in Simulink models to connect blocks/subsystems executed with different sample times. Contact dSPACE Support.
- The RTI Bypass Blockset generates model-specific template files for user-defined source code and the user-defined makefile. Unlike external bypassing, for on-target bypassing not all the C functions of the template User-Code file and not all the optional options for the user makefile are supported. The following tables show which functions and options are supported (✓) and which are not supported (–) for the different code generators:

User-Code C Function	Simulink Coder	TargetLink Code Generator
usr_initialize	✓	✓
usr_sample_input	✓	–
usr_input	✓	✓
usr_output	✓	✓
usr_terminate	–	–
usr_background	–	–

User Makefile Option	Simulink Coder	TargetLink Code Generator
SFCN_DIR	✓	–
USER_SRCS	✓	✓

User Makefile Option	Simulink Coder	TargetLink Code Generator
USER_ASM_SRCS	✓	–
USER_SRCS_DIR	✓	✓
USER_INCLUDES_PATH	✓	–
USER_OBJS	✓	✓
USER_LIBS	✓	–

Note

To avoid unpredictable build consequences, note the following:

- Do not delete the template files.
- Do not modify the template version information in the template files.
- Do not delete the function definitions in the User-Code file template.

- There are further limitations that apply to working with TargetLink when implementing on-target bypassing. Refer to Limitations When Using TargetLink with the RTI Bypass Blockset (TargetLink Interoperation and Exchange Guide 🕮).

Limitations regarding using the RTI Bypass Blockset with other blocksets

If you use the RTI Bypass Blockset together with other blocksets in your Simulink model, the Custom Code page of the Code Generation Dialog shows unexpected behavior. In the Include list of additional frame, the Include directories option is not usable. Instead, you can add custom code files via the Source files option, using absolute paths or relative paths from the model root folder (e.g., ".\LIBS\CAN\CANEncode.c"). If the referenced source code files are not found, the build process fails.

Related topics

Basics