

MicroLabBox

# Features

Release 2021-A – May 2021

## How to Contact dSPACE

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

## 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](mailto:support@dspace.de)

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

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

## Software Updates and Patches

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

## Important Notice

This publication contains proprietary information that is protected by copyright. All rights are reserved. The publication may be printed for personal or internal use provided all the proprietary markings are retained on all printed copies. In all other cases, the publication must not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of dSPACE GmbH.

© 2014 - 2021 by:  
dSPACE GmbH  
Rathenaustraße 26  
33102 Paderborn  
Germany

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

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

# Contents

About This Document	7
Introduction to the Features of MicroLabBox	9
Basics on MicroLabBox.....	9
Block Diagram.....	13
Feature Overview.....	14
Using MicroLabBox as Multicore Platform.....	16
MicroLabBox Application Start	17
Starting Operation.....	17
Firmware.....	19
Running an Application from Local Memory.....	20
Running an Application from Flash Memory.....	21
Running an Application from a USB Mass Storage Device.....	22
Interfaces	25
Host Interface.....	25
Ethernet I/O Interface.....	26
USB Interface.....	27
PGI Interface.....	28
JTAG Interface.....	28
Basic Features	29
Interrupt Handling.....	29
Trigger Signals.....	31
Timer Services.....	34
Sensor Supply.....	35
Buzzer.....	36
LED Control.....	37

I/O Features	39
A/D Conversion.....	40
ADC Class 1.....	40
ADC Class 2.....	50
D/A Conversion.....	52
DAC Class 1.....	52
Bit I/O.....	54
Bit I/O (DIO Class 1).....	54
Bit I/O (DIO Class 2).....	56
Timing I/O.....	58
PWM Signal Generation (DIO Class 1).....	58
PWM Signal Measurement (DIO Class 1).....	62
Pulse Signal Generation (DIO Class 1).....	65
Pulse Width Measurement (DIO Class 1).....	67
Nonvolatile Data Handling (NVDATA)	71
General Information on Handling Nonvolatile Data.....	71
Using the Web Interface for Nonvolatile Data Handling.....	73
USB Flight Recorder	75
Basics on USB Flight Recorder.....	75
Handling the Data of the USB Flight Recorder.....	79
Serial Interface	81
Serial Interface of MicroLabBox.....	81
Serial Peripheral Interface	83
Serial Peripheral Interface (DIO Class 1).....	83
Electric Motor Control	91
Basics on Electric Motor Control.....	91
Hall Sensor Interface.....	95
Incremental Encoder Interface.....	98
Resolver Interface.....	101
EnDat Interface.....	103

SSI Interface.....	105
Block-Commutated PWM Signal Generation (DIO Class 1).....	108
Multichannel PWM Signal Generation (DIO Class 1).....	112
<b>CAN Support</b>	<b>123</b>
Setting Up a CAN Controller.....	124
Initializing the CAN Controller.....	124
CAN Transceiver Types.....	126
Defining CAN Messages.....	130
Implementing a CAN Interrupt.....	132
Using RX Service Support.....	132
Removing a CAN Controller (Go Bus Off).....	134
Getting CAN Status Information.....	135
CAN Partial Networking.....	136
Using the RTI CAN MultiMessage Blockset.....	138
Basics on the RTI CAN MultiMessage Blockset.....	138
Basics on Working with CAN FD.....	143
Basics on Working with a J1939-Compliant DBC File.....	148
Transmitting and Receiving CAN Messages.....	154
Manipulating Signals to be Transmitted.....	157
CAN Signal Mapping.....	161
CAN Signal Mapping.....	161
<b>FPGA Support</b>	<b>163</b>
General Information on FPGA Support.....	163
Accessing the I/O FPGA Type 1 Unit.....	165
<b>Limitations</b>	<b>167</b>
Details on the Limitations for MicroLabBox.....	167
Limited Number of CAN Messages.....	167
Limitations with RTICANMM.....	169
<b>Appendix</b>	<b>175</b>
Troubleshooting.....	175
Glossary.....	176
<b>Index</b>	<b>177</b>











# About This Document

## Content

This document provides feature-oriented access to the information you need to implement your control models on a MicroLabBox.

## Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
 <b>DANGER</b>	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
 <b>WARNING</b>	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
 <b>CAUTION</b>	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
 <b>NOTICE</b>	Indicates a hazard that, if not avoided, could result in property damage.
 <b>Note</b>	Indicates important information that you should take into account to avoid malfunctions.
 <b>Tip</b>	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.
	Precedes the document title in a link that refers to another document.

## Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

---

## Special folders

Some software products use the following special folders:

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

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

or

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

**Documents folder** A standard folder for user-specific documents.

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

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

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

---

## Accessing dSPACE Help and PDF Files


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

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

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

**dSPACE Help (Web)** You can access the Web version of dSPACE Help at [www.dspace.com/go/help](http://www.dspace.com/go/help).

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

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



# Introduction to the Features of MicroLabBox

<b>Introduction</b>	MicroLabBox is a ready-to-use rapid control prototyping (RCP) system for the laboratory environment.
---------------------	--

**Where to go from here**

**Information in this section**

Basics on MicroLabBox.....	9
Block Diagram.....	13
Feature Overview.....	14
Using MicroLabBox as Multicore Platform.....	16

## Basics on MicroLabBox

<b>Introduction</b>	MicroLabBox is a compact RCP system that is easy and safe to handle.
---------------------	--

<b>Overview</b>	MicroLabBox was designed for easy handling in a laboratory environment. For example, it can be connected to the mains without using an additional power supply or a transformer. It provides standard interfaces to external devices such as Ethernet and USB connectors. For generating and measuring I/O signals, the board provides analog and digital input and output channels with included signal conditioning.
-----------------	--

---

## Hardware concept

MicroLabBox consists of two boards:

- DS1202

This is the base board of MicroLabBox, which is based on the Freescale Power Architecture® technology. It provides the communication and computation features.

The DS1202 board controls:

- The Ethernet interface, including the switch configuration for host communication and I/O access
- The USB interface for data recording and booting an application via a USB mass storage device
- Flash management for booting MicroLabBox and loading real-time applications from the flash memory
- Communication and data exchange with the DS1302 I/O Board

- DS1302

This is the I/O board of MicroLabBox. It provides the board's standard I/O features.

The DS1302 controls:

- The analog I/O channels
- The digital I/O channels
- The serial interface (RS232 and RS422/485)
- The CAN interface
- Two different sensor supply outputs
- The customizable LEDs
- The buzzer
- The resolver interfaces

---

## FPGA module

To make the I/O features of MicroLabBox flexible, they are implemented as FPGA code. The FPGA module provided on the DS1302 is automatically programmed with the standard I/O FPGA firmware when you load a real-time application.

You can replace the standard I/O FPGA firmware by a custom FPGA application that you built by using the RTI FPGA Programming Blockset. For further information on using an FPGA application, refer to [FPGA Support](#) on page 163.

---

## Connector panel types

There are different panel types available for the I/O connectors of MicroLabBox.

- Front connector panel with Sub-D connectors

All I/O signals and the programmable LEDs are available on the front side of MicroLabBox.

The digital and analog I/O pins are available via 50-pin Sub-D connectors. A serial interface and a CAN interface are available via 9-pin Sub-D connectors. There are two additional 9-pin Sub-D connectors prepared for resolver interfaces.

- Top connector panel with BNC connectors

All I/O signals and the programmable LEDs are available on the top side of MicroLabBox.

In contrast to the front connector panel with Sub-D connectors, the analog I/O pins are accessible via separate BNC connectors. This allows a fast and flexible modification of the cable harness for the analog signals.

- Top connector panel with spring-cage terminal blocks

All I/O signals and the programmable LEDs are available on the top side of MicroLabBox.

In contrast to the top connector panel with BNC connectors, the analog and digital I/O pins, and the two resolver interfaces are accessible via separate spring-cage terminal blocks. This allows a fast and flexible modification of the cable harness for the digital and analog signals.

The type of the I/O connector panel is displayed as a property of the board in dSPACE software, for example, in ControlDesk's Platform Manager.

The connectors for the Ethernet interface, USB interface, and sensor supply are placed on the rear side of MicroLabBox. They are the same for each MicroLabBox variant.

---

## Channel classes

The analog I/O and digital I/O channels are available with different characteristics. To distinguish between them, the associated function names include the class identifiers *Class 1* or *Class 2*.

For further information on the available I/O channels, refer to *MicroLabBox I/O features* in [Feature Overview](#) on page 14.

---

## Software support and further information

You can implement a real-time application for MicroLabBox by handcoding with the Real-Time Library (RTLib) or by using MATLAB/Simulink and the RTI blockset for MicroLabBox.


**RTI support** The Real-Time Interface for the DS1202 Base Board, also called RTI1202, provides graphical access to the board's I/O features. You only have to add an RTI block to your Simulink model to provide I/O functionality of MicroLabBox.


For detailed information, refer to [MicroLabBox RTI Reference](#) .

**RTLib support** The real-time library provides C functions to implement the application framework, such as timers and interrupts, to manage synchronous and asynchronous tasks. It also provides C functions to access the buzzer, the customizable LEDs and the sensor supplies of the board. Feature-specific functions are available for generating and measuring analog and digital signals.

For detailed information, refer to [MicroLabBox RTLib Reference](#) .

**Further information** This *Features* document provides basic information on the board's features. It is relevant for RTLib users and RTI users.

If you need more detailed information about the hardware, e.g., about the characteristics of an I/O channel, refer to [MicroLabBox Hardware Installation and Configuration](#) .

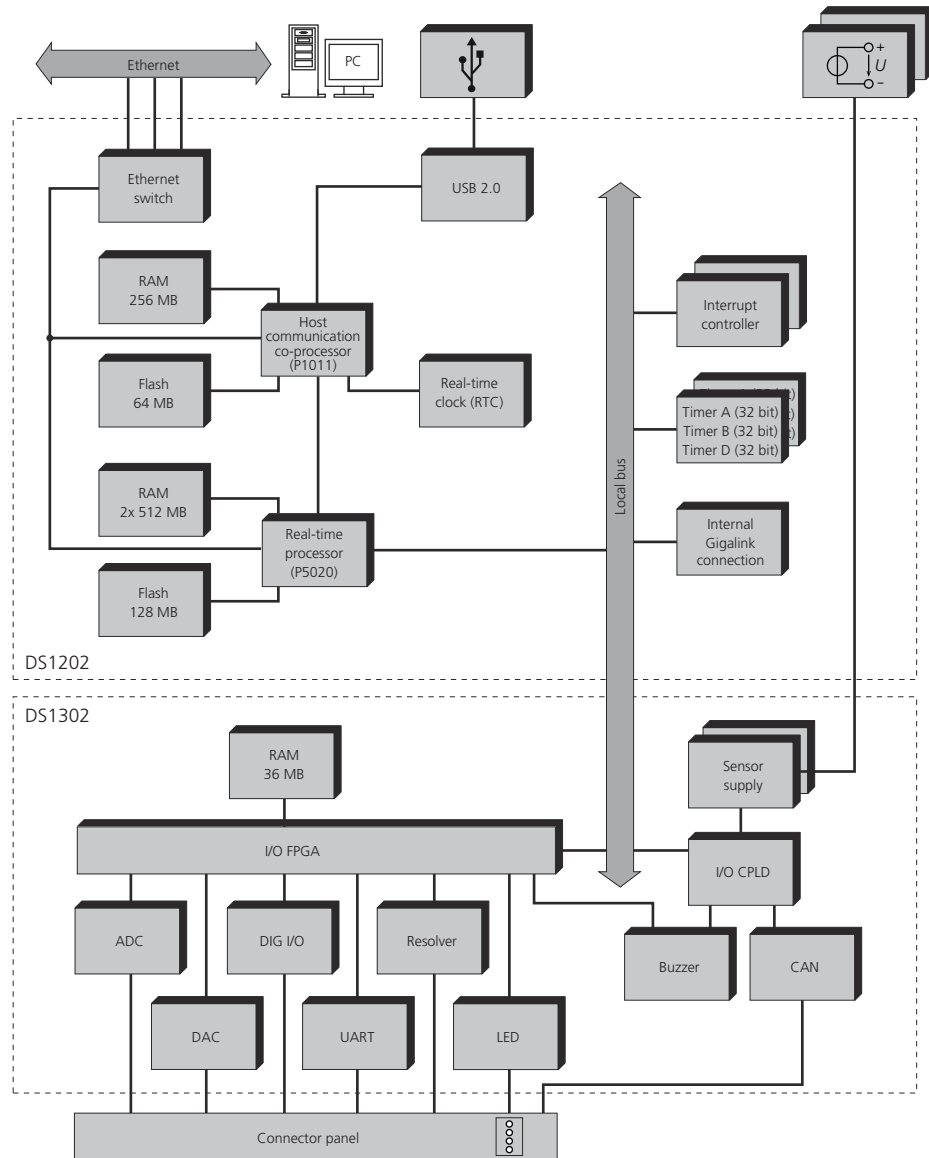
For a first overview of the tool chain that you can use MicroLabBox with, refer to [DS100x](#), [DS110x](#), [MicroAutoBox II](#), [MicroLabBox – Software Getting Started](#) .

Working with hardware might be dangerous. It is therefore recommended to also read *dSPACE General Safety Precautions*. This printed document is delivered together with your hardware. You find the document in PDF format on the dSPACE DVD.

# Block Diagram

## Introduction

The following illustration gives an overview of the functional units of MicroLabBox:



## Feature Overview

### Introduction

To give you an overview of the board features.

### MicroLabBox basic features

The DS1202 Base Board provides the following features:

**Real-time processor** MicroLabBox uses the Freescale P5020 processor as the real-time processor. It is also called computation node (CN). The real-time processor calculates your real-time models and accesses the I/O board via the local bus. This processor has the following characteristics:

- Dual-core processor with 2 GHz CPU clock
- Cache sizes:

Cache Type	Cache Size
L1 data cache	32 KB per core
L1 instruction cache	32 KB per core
L2 cache	512 KB per core
L3 cache	2 MB total

- Temperature sensor to prevent the processor from overheating

The CPU temperature is continuously monitored. If the temperature exceeds the predefined limit, a host message is generated.

For detailed information on the Freescale P5020 processor, refer to the Freescale website at <http://www.freescale.com> and search for "P5020".

The dual-core processor also lets you run real-time applications that were implemented for a multiprocessor platform, see [Using MicroLabBox as Multicore Platform](#) on page 16.

#### Note

The multiprocessor features described in the user documentation for MicroLabBox are restricted to the multicore use case using the internal virtual Gigalinks.

**Memory** MicroLabBox's real-time processor has the following memory features.

The memory of the host communication co-processor is only internally used and not relevant to custom applications.

Memory Type	Available Memory Size
RAM	< 1 GB DRAM The available local memory is reduced by the memory that is allocated by the required real-time services.
FRAM	Available Memory Size: 64 KB The FRAM has a size of 128 KB, but only 64 KB can be used with the NVDATA functions because of the double-buffer update mechanism.

Memory Type	Available Memory Size
Flash	32 MB You can use 32 MB out of the 128 MB built-in flash memory to store real-time applications and customer-specific libraries. The rest of the memory is reserved, for example, for the boot firmware.
Memory for flight recording	Depends on the size of the connected USB mass storage device and whether the device is also used for booting an application. The recommended maximum size is 32 GB.

**Timers and Time Base Counters** MicroLabBox comprises:

- Timer A and Timer D: Sample rate timers with interrupt function
- Timer B: Interval timer with interrupt function
- Time Stamp Counter: Time base for single-processor systems
- Synchronization: Synchronized time base for multicore systems

See [Timer Services](#) on page 34.

**Interrupt control** Provides various hardware and software interrupts, see [Interrupt Handling](#) on page 29.

**Host interface** For setting up MicroLabBox, downloading programs and transferring run-time data to and from the host PC, see [Host Interface](#) on page 25.

**USB interface** For accessing a USB mass storage device to store flight recording data and/or boot an application, see [USB Interface](#) on page 27.

**Ethernet I/O interface** For setting up Ethernet-based communication to external devices using the UDP/IP or TCP/IP protocol, see [Ethernet I/O Interface](#) on page 26. The host interface also uses this interface.

## MicroLabBox I/O features

The DS1302 I/O Board and its FPGA module provide the I/O capabilities of MicroLabBox.

The following I/O features are implemented in the standard FPGA code.

**A/D conversion** For information on the control functions of the analog input channels, see [A/D Conversion](#) on page 40.

**D/A conversion** For information on the control functions of the analog output channels, see [D/A Conversion](#) on page 52.

**Bit I/O** For information on the control functions of the digital input and output channels for simple bit access, see [Bit I/O](#) on page 54.

**Timing I/O** For information on the control functions of the digital input and output channels for generating and measuring signal shapes, such as pulse signals or PWM signals, see [Timing I/O](#) on page 58.

The following timing I/O features are available:

- PWM signal generation
- PWM signal measurement
- Pulse signal generation
- Pulse width measurement

**Note**

The digital I/O channels are used for bit I/O, timing I/O and the optional electric motor control (see [Electric Motor Control](#) on page 91). If you specified a digital I/O channel for one of these features, it is no longer available for the other digital I/O features.

**Serial interface** For information on the control functions of the digital input and output channels used for serial interface communication using the RS232 or RS422/485 modes, see [Serial Interface](#) on page 81.

**Serial peripheral interface (SPI)** For information on the control functions of the digital input and output channels used for serial peripheral interface communication, see [Serial Peripheral Interface](#) on page 83.

**CAN support** For information on the control functions of the digital input and output channels used for CAN communication, see [CAN Support](#) on page 123.

**FPGA support** For information on the control functions of the FPGA module as a whole, see [FPGA Support](#) on page 163.

## Using MicroLabBox as Multicore Platform

### Introduction

To give you important information on using a dual-core MicroLabBox.

### Network topology for dual-core systems

While MicroLabBox does not support external Gigalinks, you can anyway use a multiprocessor application, if it consists of only two subapplications, because the real-time processor of the board has two CPU cores. Each core provides four internal virtual Gigalinks, which can handle the IPC blocks of the RTI-MP Blockset.

If you load the real-time application (RTA file) to the platform, its application processes are automatically assigned to the board's cores. The internal network topology will be dynamically generated.



# MicroLabBox Application Start

**Introduction** After power-up, MicroLabBox boots automatically and executes the firmware located in the onboard flash memory.

**Where to go from here** Information in this section

Starting Operation.....	17
Firmware.....	19
Running an Application from Local Memory.....	20
Running an Application from Flash Memory.....	21
Running an Application from a USB Mass Storage Device.....	22

## Starting Operation

**Introduction** Before you can load a real-time application to the board by using either the non-volatile flash memory, a USB mass storage device, or the volatile local memory, you have to know how to establish the host PC communication and how to build a real-time application.

**Establishing the host PC communication** MicroLabBox provides an Ethernet interface for host communication. To integrate MicroLabBox into your TCP/IP network, you have to set a custom IP address for the board. Afterwards, you can use the board's web interface to configure further network settings. The board can then be detected by Platform Manager. After registering the board in Platform Manager, for example, via ControlDesk, you can load a real-time application to it.

For further information on installing and configuring MicroLabBox, refer to [Setting up a Connection Between the Host PC and MicroLabBox \(MicroLabBox Hardware Installation and Configuration !\[\]\(529949c2c3dadbaa4e538e8c643454bc\_img.jpg\)](#)).

### MicroLabBox's web interface

MicroLabBox has an internal web server providing the web interface for configuring and managing the board. This allows you to directly access the board without using other dSPACE software tools.

Some tools can be used only via the web interface, others are also available in ControlDesk.

For detailed information on the web interface, refer to [Using the Web Interface \(MicroLabBox Hardware Installation and Configuration !\[\]\(339a16584d5da0f0a3ca4e9ec17bf6a1\_img.jpg\)](#)).

### Building, downloading and starting your application

To run your model on the registered MicroLabBox, the appropriate application must be built and downloaded beforehand.

**Using RTI** By selecting the DS1202 platform support in MATLAB/Simulink, the required platform-specific settings for building a real-time application are automatically set. If the build process is finished successfully, the generated real-time application is downloaded to the RAM of the board by default. If you want to download the real-time application to the flash memory, you have to set the Load to flash memory option in the RTI load options page of the Code Generation dialog.

For further information, refer to [How to Create a Real-Time Application for MicroLabBox \(MicroLabBox RTI Reference !\[\]\(6059a5aa8b4ca7bb793408023d6c6e42\_img.jpg\)](#)).

**Using RTLib** After you have handcoded your real-time application, you can compile, link, and download it to the board by using the **Down1202** command line utility. It is recommended to use the Command Prompt for dSPACE RCP and HIL shortcut in the Windows Start menu to open a Command Prompt window. By this, the required paths and environment settings are automatically set. If the build process finishes successfully, the generated real-time application is downloaded to the RAM of the board by default. If you want to download the real-time application to the flash memory, you have to call the **Down1202** command with the **/f1** option.


When using RTLib, you have to manually edit the variable description file to access variables of the real-time application.

For further information, refer to [Compiling, Linking and Downloading an Application \(MicroLabBox RTLib Reference !\[\]\(9c2e8d1b5bd77cb5c9f83b7a9cff79fd\_img.jpg\)](#)).

For information on how to use a USB mass storage device for real-time applications, refer to [Running an Application from a USB Mass Storage Device](#) on page 22.

### Further information and examples

For further information on starting to work with your MicroLabBox, refer to [DS100x, DS110x, MicroAutoBox II, MicroLabBox – Software Getting Started !\[\]\(f1c5da15572e3e09d343161be98f508d\_img.jpg\)](#).

For further information on working with RTI, refer to [RTI and RTI-MP Implementation Guide](#) .

The RCP and HIL installation includes examples for Simulink models and handcoded applications. You find them in  
<RCP\_HIL\_InstallationPath>\Demos\DS1202.

## Firmware

### Introduction

After power-up and reset, MicroLabBox boots from the on-board flash memory, which holds the preinstalled firmware.

### Characteristics of the firmware

The firmware carries out the following steps:


- It determines whether a user-specific application exists in the on-board flash memory.  
If the on-board flash memory contains an application, it is started automatically.
- If the on-board flash memory does not contain an application, the firmware determines whether an application is available in the autostart folder of the connected USB mass storage device. If so, the application is started automatically.

#### Note

##### Use the latest firmware

A real-time application that you built with the latest dSPACE software version usually requires the latest firmware version on your platform for execution.

The firmware is available on the dSPACE Release DVD. You should regularly check the availability of new firmware at

<http://www.dspace.com/go/firmware>. To update the firmware, use the dSPACE Firmware Manager, refer to [Firmware Manager Manual](#) .

### Running the board in secured mode

If MicroLabBox cannot correctly boot its firmware, it automatically starts in *secured mode*. In secured mode, the user can restore the firmware by using the dSPACE Firmware Manager. Loading and running real-time applications is disabled in secured mode.

If you have problems with the board that might be caused by corrupted firmware, but the board does not automatically boot in secured mode, you can force secured mode.

The board's web interface provides a function to reboot the board to secured mode.

For further information, refer to [How to Solve Problems Related to the Firmware \(MicroLabBox Hardware Installation and Configuration !\[\]\(21199eb166cc97331a0c54c649195dcc\_img.jpg\)](#)).

#### ⚠ CAUTION

**Risk of injury and/or material damage. Updating the firmware can cause uncontrolled movements of connected devices.**

Disconnect actuators and sensors from the hardware, before you start MicroLabBox in secured mode, or before you start a firmware update.

### Related topics

#### Basics

[Firmware Manager Manual](#)

#### HowTos

[How to Solve Problems Related to the Firmware \(MicroLabBox Hardware Installation and Configuration !\[\]\(aa53ad6fea213b8b2226d3077e30533a\_img.jpg\)](#))

#### References

[General Settings Properties \(ControlDesk Platform Management !\[\]\(758ebdf4629c903da74c2e079717ae32\_img.jpg\)](#))

## Running an Application from Local Memory

### Introduction

Applications in the local memory are cleared if you turn off the hardware. You have to download an application from the host PC to the local memory of the board before you can run it.

### Download time

Download times can differ considerably. It takes longer, for example, if a formerly loaded application cannot be unloaded due to errors in the application.

### Running an application

After download, you have to explicitly start the application, for example via the Platforms/Devices controlbar in ControlDesk. To start the application automatically after download, use the **Real-Time Application / Offline Simulation Application - Load and Start** command.

### Reloading an application

Reloading an application is only necessary if you want to overwrite the currently loaded application. If you stop an application, it is still loaded and can be restarted by using the **Start** command, for example, in ControlDesk.

For details of handling real-time applications, refer to [Handling Real-Time Applications with ControlDesk \(DS100x, DS110x, MicroAutoBox II, MicroLabBox – Software Getting Started !\[\]\(8af806fb1314382d09bc5ec5b767526c\_img.jpg\)](#)).

## Related topics

## References

[Real-Time Application / Offline Simulation Application - Load \(ControlDesk Platform Management !\[\]\(e2376d476d06eb31946dc01a69a4403a\_img.jpg\)](#))  
[Real-Time Application / Offline Simulation Application - Load and Start \(ControlDesk Platform Management !\[\]\(bbb3388d591ef640dd8a8c4262f2866a\_img.jpg\)](#))  
[Reload \(ControlDesk Platform Management !\[\]\(ef6e697e79b33cfafe8ba6744dc11bd6\_img.jpg\)](#))  
[Reload and Start \(ControlDesk Platform Management !\[\]\(36a26e5b369c5d231b75de2efc184e39\_img.jpg\)](#))  
[Start \(ControlDesk Platform Management !\[\]\(c5cee65d8128a7c1c31c4ac9cbd38372\_img.jpg\)](#))  
[Stop \(ControlDesk Platform Management !\[\]\(3d68e4eb958ce14892acdd7ab347bcfa\_img.jpg\)](#))

# Running an Application from Flash Memory

## Introduction

If you want an application start automatically after power-up, you have to load it to the board's flash memory.

You can also start an application from a connected USB mass storage device, refer to [Running an Application from a USB Mass Storage Device](#) on page 22.

## Basics

MicroLabBox provides an onboard flash memory. You can use 32 MB of the flash memory for real-time applications and custom libraries. To load an application to the flash memory, you can use the Load to Flash command, for example, in ControlDesk. After an application has been written to the flash memory, it is also copied to the local memory. To start the application, use the Start command or the Load to Flash and Start command.

Upon every power-up of MicroLabBox, the firmware copies the application from the flash memory to the local memory and starts it, regardless of whether the board is connected to the host PC or not.

If MicroLabBox is not connected to the host PC, you can stop and restart the application by resetting MicroLabBox.

If MicroLabBox is connected to the host PC, you can use the dSPACE experiment software, for example, ControlDesk, to stop and restart the application without reloading it.

You can clear an application from the flash memory via the Clear Flash command, for example, in ControlDesk. Further functions for managing the flash memory are available in the board's web interface. For further information, refer to [Using the Web Interface \(MicroLabBox Hardware Installation and Configuration !\[\]\(799877f5c2f906134441300079881630\_img.jpg\)](#)).

**Note**

When you switch off MicroLabBox, the contents of the local memory are lost. When you switch on the power again, the contents of the flash memory are copied to the local memory and the application starts.

For information on the flash memory's characteristics, refer to [Feature Overview](#) on page 14.

For information on handling real-time applications using ControlDesk, refer to [Handling Real-Time Applications with ControlDesk \(DS100x, DS110x, MicroAutoBox II, MicroLabBox – Software Getting Started !\[\]\(d3fb9f94af8b26d1c844efa9a98805b0\_img.jpg\)](#)).

**Note**

Some platform management activities require exclusive access to the platform. For further information, refer to [Synchronized Platform Management with Several dSPACE Products \(ControlDesk Platform Management !\[\]\(5a132f13505a6571904d622757b7a8f0\_img.jpg\)](#)).

**Related topics****References**

[Clear Flash \(ControlDesk Platform Management !\[\]\(73002692dd5e7a64e60946be3158e719\_img.jpg\)](#))  
[Real-Time Application - Load to Flash \(DS1007/MicroLabBox/MicroAutoBox II/SCALEXIO\) \(ControlDesk Platform Management !\[\]\(42837a1907e26cf155e215b5440e265d\_img.jpg\)](#))  
[Real-Time Application - Load to Flash and Start \(DS1007/MicroLabBox/MicroAutoBox II/SCALEXIO\) \(ControlDesk Platform Management !\[\]\(42c4abe8a012119f15571407ccb34aff\_img.jpg\)](#))

## Running an Application from a USB Mass Storage Device

**Introduction**

If you want an application to start automatically after power-up, you can use a connected USB mass storage device for it.

**Basics**

If no application is stored in the board's flash memory, the boot firmware can load an application from a connected USB mass storage device to the local memory of the board and start the application.

**Requirements on the USB mass storage device** Any standard USB 2.0 or 3.0 mass storage device can be used at the USB 2.0 port, such as a USB memory stick or an external USB hard drive with or without separate power supply. The USB device must be formatted with the Microsoft FAT32 file system and must be connected directly to your hardware.

**Note**

A connection via a USB hub is not supported.

USB mass storage devices differ according to their rates for writing data. It is recommended to use a fast device for good performance.

The maximum supported file system size is 32 GB. Using a file system with a size greater than 32 GB might work but is neither recommended nor supported.

If you also want to use flight recording via a USB device, you have to use the same USB mass storage device from which you loaded the application.

**Preparing the USB mass storage device** Before you can run an application from a USB mass storage device, you have to prepare the device by using your host PC.

To run an application from a USB mass storage device, you have to create the following two folders on the device:

- **autostart**

You have to copy the application that you want to start after power-up to this folder.

**Note**

- Only one application file is allowed in this folder. If multiple applications are stored in this folder, the board will not load any of them.
- If an application is stored in the board's flash memory, the application on the USB device will be ignored. You have to clear the flash memory so you can use the USB device for loading an application.

- **applications**

You can copy the applications that you want to start manually to this folder.

If the USB mass storage device is connected to the board, you can create an image of the currently running application. The image file will then be stored in this folder as well.

After you prepared the USB mass storage device, you can plug the USB device into your board. Then, you can use the board's web interface to manage the applications.

Flight recording data will be stored directly in the root folder of your USB mass storage device.

**Managing applications on the USB device** Enter the board's IP address in any web browser to open the board's web interface, which provides the **USB Management** page. You can use the commands on this page to manage the applications on the USB device without disconnecting it from the board.

For general information on the web interface, refer to [Basics on the Web Interface \(MicroLabBox Hardware Installation and Configuration !\[\]\(df47d6bec273bbb8b349135fff3a20f7\_img.jpg\)](#)).

The USB Management page provides the following commands:

- **Create Application Image**

With this command, you can create an application image from the currently running application and store it to the **applications** folder. The file name consists of the application name, the current date and time and the suffix *img*. In contrast to applications that were generated by using MATLAB/Simulink (with the suffix *rta*), an image file does not contain any dynamic state information that might have existed at the time the image file was captured.

- **Manage USB Applications**

You can use this command to display an overview of the application files stored in the **autostart** and **applications** folders on the USB device. Each entry provides the **Delete** command to delete the selected application from the USB device and the **Start** command to start it manually.

#### Note

- Files are deleted immediately, without confirmation.
- Before you can load and start an application, any running application must be unloaded from the RAM. You can use the **Unload Application** command for this. You can start applications only in the running state, not in the stopped state.

You can use the **Copy to Autostart** command to copy the selected application file to the **autostart** folder. Any existing application files in the **autostart** folder are renamed, so that they will no longer be recognized as executable applications. The copied application can be started manually by using the **Start** command or after a reboot of the board.

- **List USB Applications**

You can use this command to display an overview of the application files stored on the USB device.

- **Unload Application**

You can use this command to stop and unload the running application.

- **Delete USB Autostart**

You can use this command to delete any files stored in the **autostart** folder of the USB device. This is primarily required to delete the copies of application files that were created when copying an application file from the **applications** folder to the **autostart** folder.

#### Note

The executable application will also be deleted.

## Related topics

## References

[Clear Flash \(ControlDesk Platform Management\)](#) 



# Interfaces

## Introduction

MicroLabBox has several interfaces for data communication with the host PC and external devices.

## Where to go from here

### Information in this section

Host Interface.....	25
Ethernet I/O Interface.....	26
USB Interface.....	27
PGI Interface.....	28
JTAG Interface.....	28

### Information in other sections

Communication Interfaces (MicroLabBox Hardware Installation and Configuration )

## Host Interface

### Basics

MicroLabBox uses a standard Ethernet connection with TCP/IP protocol as its host interface.

The host interface is used to:

- Connect the board to a host tool such as ControlDesk.
- Download a real-time application to the board.

- Transfer runtime data to and from the host PC.
- Display and configure the board via its web interface.

By entering `http://<IPAddress>` in a standard web browser, you can open the web interface for configuring and managing the board.

This tool provides the following functions:

- Specifying the board's network configuration.
- Configuring the Ethernet switch.
- Managing the onboard flash memory.
- Managing applications on a connected USB mass storage device.
- Generating a system status report to provide information to dSPACE Support.
- Displaying host messages generated by the board without ControlDesk connected.
- Rebooting the system in different modes.

For further information, refer to [Basics on the Web Interface \(MicroLabBox Hardware Installation and Configuration !\[\]\(339a16584d5da0f0a3ca4e9ec17bf6a1\_img.jpg\)](#)).

## Ethernet I/O Interface

### Basics

MicroLabBox provides three RJ45 connectors for Ethernet communication. The connectors are internally managed by an Ethernet switch. Each of the connectors is therefore equivalent. While one connector is always to be used for the host PC communication, the other two connectors can be individually configured for I/O communication or host PC communication.

### Field of application

You can use the Ethernet I/O Interface to perform communication between the real-time application and any external device that also provides an Ethernet interface using TCP/IP or UDP/IP protocol, such as another dSPACE board or a calibration device. The RTI Ethernet Blockset lets you implement the communication in a Simulink® model. For further information, refer to [RTI Ethernet Blockset Reference !\[\]\(c50c8b7b2cc2cf9ff925edec0ee94c0d\_img.jpg\)](#).

### Data transfer

The Ethernet I/O Interface allows Gigabit connections and supports the TCP/IP and UDP/IP protocols. This interface is optimized to provide low data transfer latencies and high data transfer rates.

**Supported features and limitations** If you want to implement network-based communication, you should have a basic knowledge of IP-based networks. Here is a list of some basic features and limitations.

The following general features are supported:

- UDP/IP protocol and TCP/IP protocol
- 10/100/1000 Mb data rates

- Half duplex/Full duplex modes
  - Up to 256 sockets
  - Auto negotiation with Auto-MDIX
  - DHCP
  - Gateway configuration
  - Local loopback
  - Interrupt generation when data has been received
  - IP fragmentation
  - UDP-specific features:
    - Broadcast
    - Dynamic change of destination addresses
  - TCP-specific features:
    - Client and server mode
- TCP-specific limitation:
- A server socket cannot accept more than one client socket to be connected to.

### Specifying the IP addresses

Because the IP protocol is used, you must specify the IP addresses of the Ethernet I/O Interface (source) and the target device (e.g., ECU).

Using the RTI Ethernet Blockset, the ETHERNET\_SETUP\_BLx block provides the relevant IP address parameters, refer to [ETHERNET\\_SETUP\\_BLx \(RTI Ethernet Blockset Reference\)](#).

### Related topics

#### References

[RTI Ethernet Blockset Reference](#)

## USB Interface

### Introduction

MicroLabBox provides a USB 2.0 connector with two ports. The USB port (A) can be used for flight recording and booting an application. A second USB port (B) is reserved for future use.

For detailed information on the USB Flight Recorder, refer to [USB Flight Recorder](#) on page 75.

For detailed information on booting from a USB device, refer to [Running an Application from a USB Mass Storage Device](#) on page 22.

## PGI Interface

---

### Introduction

The Programmable Generic Interface (PGI) can be used by a dSPACE solution that allows you to connect sensors and actuators with different interfaces and protocols to your system. The PGI solution consists of hardware and software. For further information, contact dSPACE Support.

## JTAG Interface

---

### Introduction

The FPGA JTAG interface is prepared for connecting a debugger to the user programmable I/O FPGA.

With dSPACE Release 2015-A, the connector provides no functionality.

# Basic Features

---

<b>Introduction</b>	MicroLabBox provides basic features for implementing tasks and accessing board devices.
---------------------	---

---

## Where to go from here

## Information in this section

Interrupt Handling.....	29
Trigger Signals.....	31
Timer Services.....	34
Sensor Supply.....	35
Buzzer.....	36
LED Control.....	37

## Interrupt Handling

---

<b>Introduction</b>	MicroLabBox provides interrupts for controlling the execution of interrupt service routines on different levels of your program code.
---------------------	---

---

### Generating interrupts for controlling the task execution

The interrupt controller of MicroLabBox's DS1202 Board handles various interrupts (level-triggered or edge-triggered), for example, timer interrupts. Each processor core has its own interrupt controller. The interrupts can be masked. You can globally enable and disable interrupt generation. The interrupts are prioritized.

With RTI, you can easily implement interrupt-driven subsystems by means of specific interrupt blocks provided in the RTI library. You can use these blocks to

receive interrupts from I/O boards. If you create a handcoded model, you can use RTLib functions to handle interrupts.

---

### Generating interrupts by I/O features

If an I/O feature provides execution states that might be relevant for starting another function or synchronizing with other functions, you can configure the feature to generate interrupts. All interrupt types of one I/O feature can be combined. The interrupt generation is a channel-specific configuration of an I/O feature. Other channels of the same feature type can be configured differently.

The following I/O features provide interrupt generation:

- ADC Class 1  
Interrupts can be generated at:
  - Start of a conversion burst
  - End of a conversion burst / Data ready
  - Channel failure
- Bit I/O Class 1, PWM Signal Generation and PWM Signal Measurement  
Interrupts can be generated at:
  - Rising edge
  - Falling edge
- Pulse Width Measurement  
Interrupts can be generated at:
  - Data ready
- Serial Peripheral Interface (SPI)  
Interrupts can be generated at:
  - End of SPI cycle
- Serial interface  
Interrupts can be generated at:
  - Message has been sent
  - Message has been received
  - Bus is in line status or modem status
- CAN interface  
Interrupts can be generated at:
  - Message has been sent
  - Message has been received
  - Bus is in off state

The generated interrupts can be used to trigger an interrupt service routine or a subsystem in your Simulink model.

---

### Configuring interrupt generation

Interrupts and trigger signals are handled as events of an I/O feature. For information on the common configuration settings for events, refer to *Configuring events* in [Trigger Signals](#) on page 31.



**RTI/RTLib support**

You can implement the interrupt handling via RTI and RTLib.

For details, see

- RTI:

For specifying the interrupt generation on the Event pages of the following RTI blocks:

- [ADC\\_CLASS1\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_BIT\\_IN\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_BIT\\_OUT\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_PWM\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_PWM2D\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_PW2D\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_SPI\\_SETUP\\_BLx](#) (MicroLabBox RTI Reference )

For making interrupts available as trigger sources:



- [ADC\\_CLASS1\\_HWINT\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_HWINT\\_BLx](#) (MicroLabBox RTI Reference )

For further information, refer to [Interrupts](#) (MicroLabBox RTI Reference )

- RTLib:

For specifying the interrupt generation of I/O features, except for the serial interface and the CAN interface, you have to use the following RTLib functions:

- `xxxx_setInterruptMode` to specify the event react to. (xxxx must be replaced by the related I/O feature, see the list of the supported I/O features above.)
- `xxxx_setIoIntVector` to register the interrupt service routine to be triggered by the interrupt.
- `xxxx_disableInterrupts` and `xxxx_enableInterrupts` to stop and restart the generation of interrupts.

For further information, refer to [Interrupt Handling](#) (MicroLabBox RTLib Reference ) and [Implementing I/O Functions](#) (MicroLabBox RTLib Reference )

## Trigger Signals

**Introduction**

MicroLabBox provides 16 trigger lines for triggering other I/O features.

**Basics on trigger lines**

The trigger feature of MicroLabBox supports the same events that you can use for interrupt generation. If an I/O feature provides execution states that might be relevant for starting another function or synchronizing with other functions, you can configure the feature to generate trigger signals. You can combine up to two trigger types for one I/O feature. The trigger signal generation is a

channel-specific configuration of an I/O feature. Other channels of the same feature type can be configured differently.

In contrast to interrupts, which are handled by the real-time processor, a trigger is a pulse signal generated on an internal trigger line of the I/O FPGA. For each trigger signal that is to be generated, you have to choose one of the 16 trigger lines to be used for transmitting the signal. A trigger line can have only one source, but multiple consumers.

A trigger signal must be connected to internal consumers, for example, to start an A/D conversion. By connecting a trigger signal to a pulse signal generator, you can trigger an external device via the related digital output channel.

---

### Generating trigger signals

The following I/O features provide trigger signal generation:

- ADC Class 1  
Trigger signals can be generated at:
  - Start of a conversion burst
  - End of a conversion burst / Data ready
  - Channel failure
- Bit I/O Class 1, PWM Signal Generation and PWM Signal Measurement  
Trigger signals can be generated at:
  - Rising edge
  - Falling edge
- Pulse Width Measurement  
Trigger signals can be generated at:
  - Data ready
- Serial Peripheral Interface (SPI)  
Trigger signals can be generated at:
  - End of SPI cycle

---

### Consuming trigger signals

To use a trigger signal for triggering an I/O feature of the board, the consumer has to be specified to listen to the relevant trigger line.

The following I/O features provide trigger signal consumption:

- ADC Class 1  
Trigger signals can be used as trigger sources for:
  - Starting a burst of conversions
  - Starting a conversion
- Pulse Signal Generation  
Trigger signals can be used as trigger sources for:
  - Generating a pulse signal

The triggered pulse signal can then be used to trigger an external device connected to the digital output channel specified for pulse signal generation.



## Configuring events

Interrupts and trigger signals are handled as events of an I/O feature. You can specify the behavior of a specified event. The event configuration is applied to both event types. You cannot configure interrupts and trigger signals separately.

**Downsampling an event** If you do not want to generate an interrupt or trigger signal at each specified event, for example, at each rising edge of an input signal, you can specify a downsampling factor to reduce the number of generated interrupts and trigger signals. At maximum, you can specify to react only to every 256<sup>th</sup> event.

**Delaying the reaction to an event** If you do not want to generate an interrupt or trigger signal directly at the occurrence of an event, you can specify a time interval to delay the generation. At maximum, you can specify a time delay of 1.34 seconds.

### Note

These event settings are only available for:

- Bit I/O
- PWM signal generation
- PWM signal measurement
- Pulse width measurement (only event delay available)




## RTI/RTLib support

You can implement the trigger handling via RTI and RTLib.



For details, see

### ▪ RTI:

For specifying the trigger generation on the Event pages of the following RTI blocks:

- [ADC\\_CLASS1\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_BIT\\_IN\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_BIT\\_OUT\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_PWM\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_PWM2D\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_PW2D\\_BLx](#) (MicroLabBox RTI Reference )
- [DIO\\_CLASS1\\_SPI\\_SETUP\\_BLx](#) (MicroLabBox RTI Reference )

For specifying the trigger use for the following RTI blocks:

- [ADC\\_CLASS1\\_BLx](#) (MicroLabBox RTI Reference ) - Single Conv page and Burst Conv page
- [DIO\\_CLASS1\\_PULSE\\_BLx](#) (MicroLabBox RTI Reference ) - Trigger page

### ▪ RTLib:

For specifying the trigger generation, you have to use the following RTLib functions:

- `xxxx_setTriggerMode` to specify the event to react to. (xxxx must be replaced by the related I/O feature, see the list of the supported I/O features above.)

- `xxxx_setTriggerLineOut` to specify the trigger line to be used for transmitting the generated trigger signal.
  - `xxxx_setTriggerLineOutStatus` to set the status of the trigger line.
- For specifying the trigger use, you have to use the following RTLib function:
- `xxxx_setTriggerLineIn` to specify the trigger line to listen to.
- For further information, refer to [Implementing I/O Functions \(MicroLabBox RTLib Reference !\[\]\(467d80e979964f7f8c752fb22248b5b7\_img.jpg\)](#)).

## Timer Services

<b>Introduction</b>	MicroLabBox provides various timers and a time stamp counter.
<b>Timer clocks</b>	The clocks of the time-stamp counter and the timers A, B, and D are derived from the bus clock (BCLK) of 100 MHz that equals a timer period of 10 ns. Each processor core of MicroLabBox provides its own timers.
<b>Timer A and Timer D</b>	<p>Timer A and Timer D are used in the same way. They are usually used for periodic timer events such as the timer-driven tasks of an application. The timers are 32-bit down counters that generate an interrupt whenever they reach zero. After generating an interrupt, the timers are reloaded automatically.</p> <p>Timers A and D are driven by <math>BCLK/2</math>, which equals to a timer period of 20 ns.</p> <p>In a multicore system, the interrupts of Timer A can be forwarded to other processor cores of a MicroLabBox. The interrupts of Timer D cannot be forwarded.</p>
<b>Timer B</b>	<p>Timer B can be used for periodic or asynchronous events. It is a 32-bit up counter with a prescaler and programmable compare value. Timer B generates an interrupt when it reaches the compare value. After generating an interrupt, the counter continues counting. To generate the next interrupt, the compare value has to be set to the next desired time.</p> <p>The prescaler is programmable in power-of-two steps (4 ... 512) so that Timer B can be driven by <math>BCLK/4</math> ... <math>BCLK/512</math> (40 ns ... 5120 ns). If you use the interrupts via RTI blocks, the prescaler is always set to the highest resolution (<math>BCLK/4</math>).</p> <p>In a multicore system, the interrupts of Timer B can be forwarded to other processor cores of MicroLabBox.</p>
<b>Time-Stamp Counter</b>	The Time-Stamp Counter is a 64-bit counter based on the CPU-internal time-base register. The counter is driven by a 25 MHz clock and has therefore a

resolution of 40 ns. Each CPU core has its own 64-bit time-base register. Both registers are synchronized. You can either read the entire 64 bits of the register or the lower 32 bits only.

**Time interval measurement** The Time-Stamp Counter can be used for determining absolute points in time as well as for measuring intervals. For details, refer to [Time Interval Measurement \(MicroLabBox RTLib Reference !\[\]\(c507f772dba2b921f86777f01218e570\_img.jpg\)](#)).

**Time-stamping** The Time-Stamp Counter provides the time base for time-stamping. Time-stamping supplements data points. This means that the plots are not distorted even if data points are sampled at irregular intervals, for example, when asynchronous tasks are simulated.

#### Tip

You can always use the RTLib's Time Stamping module to read the current system time. The Time Stamping module automatically accesses the correct time base. For details on the Time Stamping module, refer to [Time-Stamping \(MicroLabBox RTLib Reference !\[\]\(a03a7eb2f4046e1d3c76772003e549ea\_img.jpg\)](#)).

In a multicore system, time-stamping is used to produce a global time base for all the connected processor cores of a MicroLabBox. For more information, refer to [Synchronization Aspects \(DS1007 Features !\[\]\(cbe2492b119e39e02a1dab2af4a4b296\_img.jpg\)](#)).

## RTI/RTLib support

You can implement the timer services via RTLib. With RTI, you use the timer services of MATLAB/Simulink and the RTI Timer Interrupt block.

For details, see

- RTI:
  - [Timer Interrupt Block \(RTI and RTI-MP Implementation Reference !\[\]\(d27edc55493507da2f9b8c7a52b3b96f\_img.jpg\)](#))
- RTLib:
  - [Timer A \(MicroLabBox RTLib Reference !\[\]\(9bf7a72a60a57323fa980b9b0338593f\_img.jpg\)](#))
  - [Timer B \(MicroLabBox RTLib Reference !\[\]\(4b60241e906ef61007ada3e521a0c6a3\_img.jpg\)](#))
  - [Timer D \(MicroLabBox RTLib Reference !\[\]\(5c2af0230acb459edf1f07c643964277\_img.jpg\)](#))
  - [Time Interval Measurement \(MicroLabBox RTLib Reference !\[\]\(5830b3ccd9bca4967fbf16381746f93d\_img.jpg\)](#))
  - [Time-Stamping \(MicroLabBox RTLib Reference !\[\]\(880cb2800aa1f40e4b440b7f1a01127d\_img.jpg\)](#))

## Sensor Supply

### Introduction

MicroLabBox provides two sensor supply outputs with different characteristics.

### Sensor supply output with fixed output voltage

One sensor supply provides a fixed output voltage of 12 V DC. The output is permanently available while MicroLabBox is switched on.

---

**Sensor supply output with adjustable output voltage**

The second sensor supply provides an adjustable voltage in the range 2 ... 20 V DC. This output is controlled by your real-time application in its initialization phase. When the real-time application is not accessing the sensor supply, no voltage output is applied.

**Note**

The sensor supply remains activated if the real-time application that has initialized the sensor supply has been unloaded.

---

**RTI/RTLib support**

You can access the adjustable sensor supply via RTI and RTLib.

For details, refer to:

- RTI:  
[DS1202\\_SENSOR\\_SUPPLY](#) (MicroLabBox RTI Reference )
- RTLib:  
[Sensor Supply](#) (MicroLabBox RTLib Reference )

---

**Related topics****References**

[Sensor Supply Outputs](#) (MicroLabBox Hardware Installation and Configuration )

## Buzzer

---

**Introduction**

MicroLabBox provides a buzzer for generating acoustic signals.

---

**Generating an acoustic signal**

The buzzer of MicroLabBox can be controlled via a real-time application.

The following settings are available to configure the sound:



- Frequency of a beep
- Duration of one beep
- Number of beeps
- Time interval between two beeps

---

**RTI/RTLib support**

You can access the buzzer via RTI and RTLib.

For details, refer to:

- RTI:  
[BUZZER](#) (MicroLabBox RTI Reference )
- RTLib:  
[Buzzer Control](#) (MicroLabBox RTLib Reference )

## LED Control

---

**Introduction**

MicroLabBox provides four customizable LEDs.

---

**Controlling the customizable LEDs**

The customizable LEDs of MicroLabBox can be controlled via a real-time application.

For each LED, you can separately configure its color in RGB format.

---

**RTI/RTLib support**

You can access the customizable LEDs via RTI and RTLib.

For details, refer to:

- RTI:  
[LED\\_BLx](#) (MicroLabBox RTI Reference )
- RTLib:  
[LED Control](#) (MicroLabBox RTLib Reference )



# I/O Features

---

## Introduction

MicroLabBox supports the following I/O features

---

## Where to go from here

## Information in this section

<a href="#">A/D Conversion.....</a>	<a href="#">40</a>
<a href="#">D/A Conversion.....</a>	<a href="#">52</a>
<a href="#">Bit I/O.....</a>	<a href="#">54</a>
<a href="#">Timing I/O.....</a>	<a href="#">58</a>

## Information in other sections

<a href="#">Serial Interface.....</a>	<a href="#">81</a>
MicroLabBox provides a serial interface that can be used to implement a serial communication.	
<a href="#">Serial Peripheral Interface.....</a>	<a href="#">83</a>
MicroLabBox provides serial peripheral interfaces that can be used to perform high-speed synchronous communication with external devices.	

## A/D Conversion

**Introduction** MicroLabBox provides two A/D conversion units with different characteristics.

**Where to go from here**

**Information in this section**

ADC Class 1.....	40
ADC Class 2.....	50

## ADC Class 1

**Introduction** The ADC Class 1 unit provides A/D converters that can be configured for various use cases. The features are described in detail.

**Characteristics**

The ADC Class 1 unit consists of 24 independent A/D converters equipped with differential inputs.

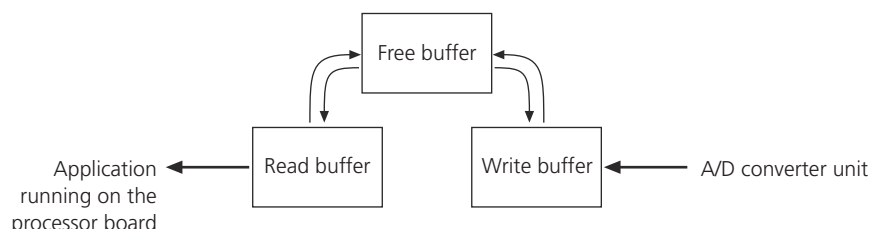
Each of the 24 A/D conversion channels provides:

- 16-bit resolution
- A sample rate of 1 MSPS (maximum conversion time of 1  $\mu$ s)
- An input voltage range of -10 V ... +10 V
- Burst mode for sampling a data set of up to 8192 analog values per burst:
  - Triggered sample mode with selectable trigger sources for starting the bursts and A/D conversions (see [Burst-triggered sample mode](#) on page 43)
  - Continuous sample mode with automatically started successive bursts and A/D conversions (see [Burst continuous sample mode](#) on page 43)
- Single A/D conversion mode to use the channel as a standard A/D converter without utilizing its burst capability (see [Single conversion mode](#) on page 44)
- Selectable sources for triggering A/D conversions, for example, trigger line, channel timer or software trigger (see [Trigger signals](#) on page 42)
- Swinging buffers for decoupling the conversion process from the read process (see [Swinging Buffer](#) on page 41)
- Three independent hardware interrupts associated with the A/D conversion state. For information on ADC Class 1 interrupt handling, see [Interrupts provided by the ADC Class 1 unit](#) on page 46.



## Swinging Buffer

Each A/D conversion channel features a swinging buffer for decoupling the write buffer and the read buffer.



**Swinging buffer principle** The swinging buffer, comprising a write, a free, and a read buffer, passes all conversion results from the A/D converter unit to the application running on MicroLabBox. The buffers can change places, as indicated by the arrows between them in the illustration above. The events which trigger the buffer exchange are described below. The number of temporarily stored conversion results, the burst size, can be uniformly specified in the range 1 ... 8192.

**Write buffer** The A/D converter unit writes the conversion results to the write buffer until it is filled to the specified burst size. When this event occurs, the write buffer changes places with the free buffer, which is then filled with new conversion results.

**Read buffer** The application running on MicroLabBox reads the conversion results from the read buffer. Before the conversion results are transferred, the read function used in the application requests a read buffer.

Depending on the read method used by the application,

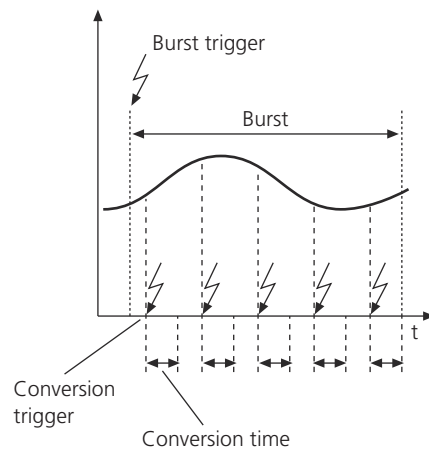
- The current conversion results in the read buffer are transferred immediately.
- Or
- The A/D conversion function waits until the buffer control unit exchanges the read buffer with the free buffer containing new conversion results.

For more information on the read methods, refer to [Methods of reading conversion results](#) on page 44.

If the application does not read the conversion results fast enough, the free buffer, containing new results, might be overwritten by the write buffer before becoming the read buffer. In this case, a data lost interrupt is generated on the channel (if the generation of the data lost interrupt was initialized beforehand).

## Trigger signals

To configure an A/D converter channel of the ADC Class 1 unit, up to two kinds of triggers must be selected.



**Burst trigger** A burst trigger initiates a sequence of A/D conversions, called a conversion burst. You select the burst trigger by assigning a burst trigger source, such as a trigger line to a converter channel.

**Conversion trigger** The conversion trigger starts every A/D conversion with one conversion result per trigger. You select the conversion trigger by assigning a conversion trigger source, such as the channel timer to a converter channel.

**Starting a conversion burst** You need a burst trigger and a conversion trigger for every burst. One burst trigger has to initiate the burst. Then the conversion triggers have to start the A/D conversions. The burst ends, controlled by the buffer control unit, when the number of A/D conversion results in the write buffer has reached the specified burst size.

**Trigger control** When a burst is running, additional burst triggers are ignored. When no burst is running, conversion triggers are ignored.

If a conversion is in progress and has not completed before another conversion trigger occurs, this trigger is ignored, and a conversion trigger overflow interrupt is generated (if the generation of the conversion trigger overflow interrupt was initialized beforehand). For more information, refer to [Interrupts provided by the ADC Class 1 unit](#) on page 46.

**Trigger sources** An ADC Class 1 conversion channel can react to the following trigger sources:

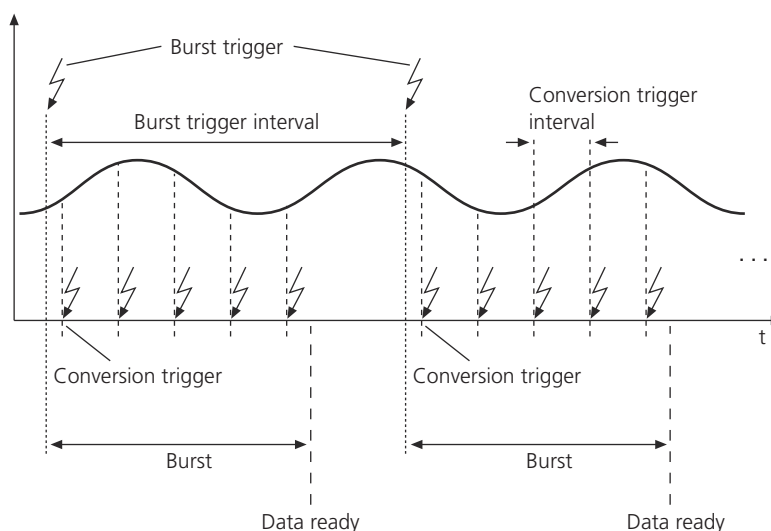
- Trigger line
- Individual channel timer for each channel
- Software trigger (using RTLib) and sample base rate (using RTI).

**Trigger line** The trigger line that you specify as trigger source must be connected to another I/O feature in your real-time application that generates a trigger signal. For further information, refer to [Trigger Signals](#) on page 31.

For detailed information on the electrical characteristics, refer to [Data Sheet \(MicroLabBox Hardware Installation and Configuration\)](#).

**Burst-triggered sample mode**

Each burst is started by the trigger event according to the selected burst trigger source. The burst is finished when the specified number of A/D conversions was executed. Then the conversion channel waits for the next burst trigger before starting a new burst conversion.



The burst conversion trigger and the first conversion trigger in the burst do not need a time delay.

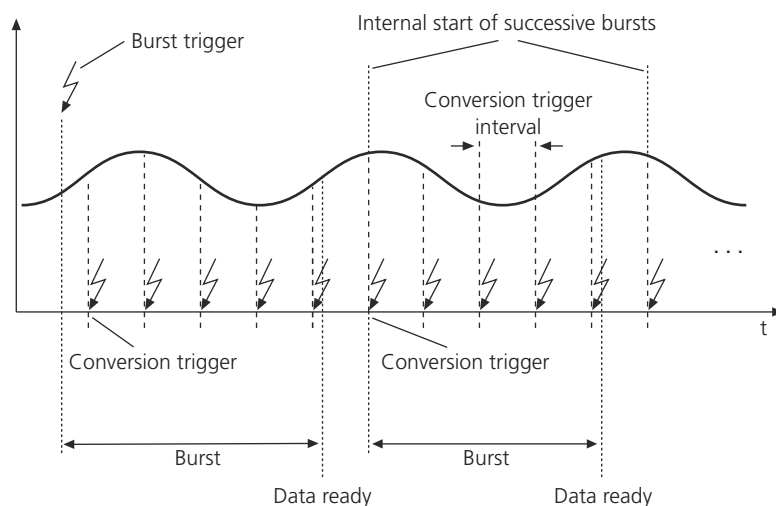
The conversion results can be read after a burst has completely finished or was stopped before the last A/D conversion was executed. There are three methods of reading the conversion results. For details, refer to [Methods of reading conversion results](#) on page 44.

**Burst continuous sample mode**

Only the first burst must be started by a trigger event according to the selected burst trigger source. Successive bursts are started automatically. After a burst is finished, the next burst is started immediately after the last A/D conversion of the previous burst. No further burst triggers are required.

Using the RTI Blockset, the initial burst trigger is performed internally if the continuous sample mode is set.

The A/D conversions within the bursts are performed according to the selected conversion trigger source.

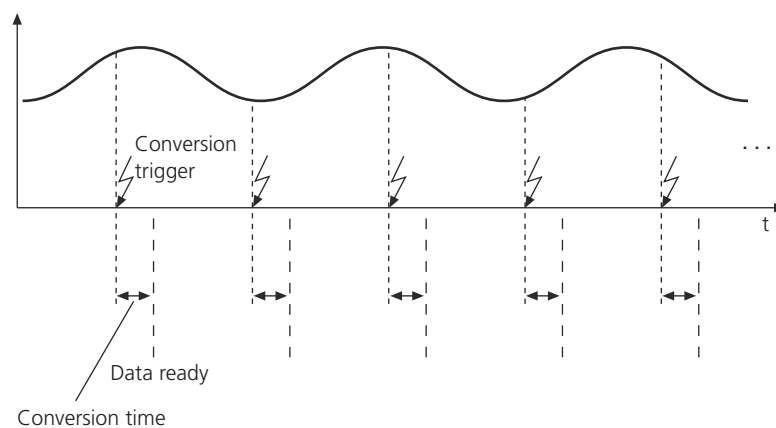


The first burst conversion trigger and the first conversion trigger do not need a time delay.

The conversion results can be read after a burst has completely finished or was stopped before the last A/D conversion was executed. There are three methods of reading the conversion results. For details, refer to [Methods of reading conversion results](#) on page 44.

### Single conversion mode

In single conversion mode, the burst capabilities of an ADC Class 1 conversion channel are not used. The conversion channels work as standard A/D converters for converting a single value after receiving a conversion trigger. Each trigger event produces one conversion result which can be read immediately after its conversion time.



### Methods of reading conversion results

The RTI Blockset and the RTLib functions offer different methods of reading the A/D conversion results.

**Using the polling method** The first method to read new conversion results is to use read functions, which causes the application to wait until new conversion results are available. This is the polling method.

- RTI Blockset

The RTI Blockset uses this method in the configurations shown in the table below.

- RTLib does not offer polling read functions.

To avoid reading the current conversion results repeatedly until a new read buffer is available, the RTLib alternatively offers the `AdcCl1AnalogIn_isDataReady` function, which only queries if a new buffer is available but does not read the conversion results.

**Using the data ready interrupt** The second method of reading new conversion results is to use the data ready interrupt indicating that the conversion has finished.

Using the data ready interrupt is the most efficient way to read new conversion results. This method is recommended for use in applications that require fast response.

The data ready interrupt indicates that a single conversion or a burst of A/D conversions has finished and the new conversion results can be read using a non-polling read method. The interrupt must be made available by adding an `ADC_CLASS1_HWINT_BLx` block to your Simulink model. The `AdcCl1AnalogIn_setInterruptMode` function enables the interrupt in your handcoded C application.

The RTI block which reads the conversion results and other functions you want to react to the interrupt must be embedded in a subsystem driven by the data ready interrupt. In handcoded C applications, the reading functions and other functions are generally placed in the interrupt service routine. For details on the MicroLabBox interrupts, refer to [Interrupt Handling](#) on page 29.

**Using the non-polling method** The third method is to read conversion results immediately without waiting for a finished conversion. You can also read old conversion results with this non-polling method.

This reading method uses read functions which do not poll internally for the availability of a read buffer with new conversion results. The current read buffer is read instead. The required information on whether a buffer was read repeatedly or a buffer with new conversion results was read is indicated by an RTI block output or a flag as a parameter of the RTLib function.

- RTI Blockset

The RTI Blockset uses this method in the configurations shown in the table below.

- RTLib offers the following functions which read the current buffer and provide the buffer state:

- `AdcCl1AnalogIn_read`  
followed by
- `AdcCl1AnalogIn_getSingleValue`
- `AdcCl1AnalogIn_getBurstCurrent`

**Read methods used with RTLib and RTI** With RTLib, you can implement a read method by using the appropriate RTLib functions. With RTI, the specified block settings determine the read method. Because there is no visible hint in the block's dialog, the following table shows you which settings result in which read method.

Conversion Mode	Burst Trigger	Conversion Trigger	Read Method
Single conversion	(Continuous)	Sample base rate (ADC)	Polling
		Sample base rate (ADCSTART)	Non-polling
		Trigger line	Non-polling
Burst conversion	Sample base rate (ADC)	Timer	Polling
		Trigger line	Non-polling
		Sample base rate (ADCSTART)	Non-polling
		Trigger line	Non-polling
		Sample base rate (ADC)	Non-polling
		Sample base rate (ADCSTART)	Non-polling
	Continuous	Timer	Non-polling
		Trigger line	Non-polling
		Sample base rate (ADC)	Non-polling
		Sample base rate (ADCSTART)	Non-polling
		Timer	Non-polling
		Trigger line	Non-polling

- Sample base rate (ADC) means that the ADC\_CLASS1\_BLx block triggers the conversion start and burst start.
- Sample base rate (ADCSTART) means that the ADC\_CLASS1\_START\_BLx block triggers the conversion start and burst start.

### Interrupts provided by the ADC Class 1 unit

The hardware interrupts from the ADC Class 1 unit are used to trigger interrupt-driven tasks which are executed on MicroLabBox.

An interrupt must be enabled to be used. The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide\)](#).

The interrupt which is to trigger the subsystem must be made available using the ADC\_CLASS1\_HWINT\_BLx block. The appropriate channel number and the respective interrupt specification in the block's dialog have to be selected. The model must contain an ADC\_CLASS1\_BLx block using the specified channel.

#### Note

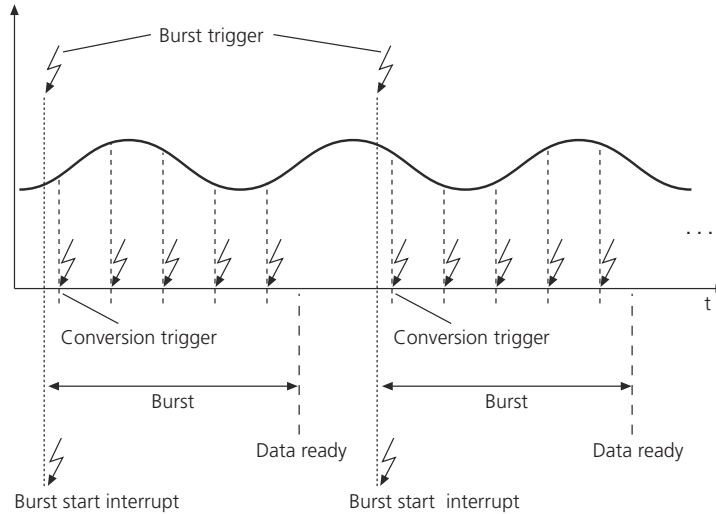
You need a separate ADC\_CLASS1\_HWINT\_BLx block for each interrupt that you want to use on a conversion channel.

**Burst start interrupt** The burst start interrupt is provided by the A/D converters, one per channel.

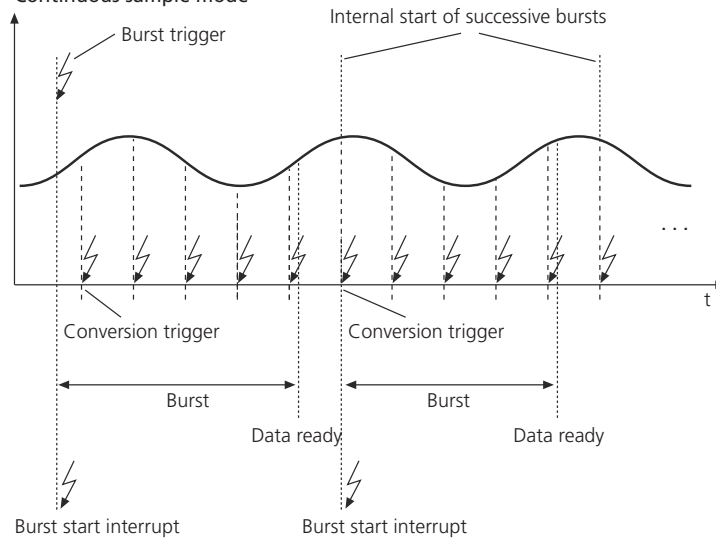
For burst conversion mode, the interrupt shows that a conversion burst has started.

For single conversion mode, the interrupt shows that a conversion has started.

#### Triggered sample mode



#### Continuous sample mode

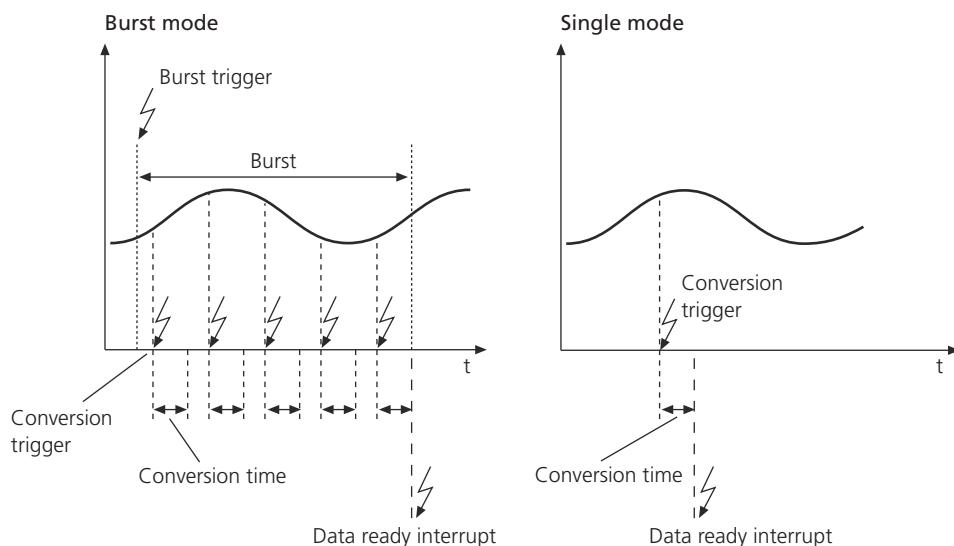


The first burst conversion trigger and the first conversion trigger do not need a time delay.

**Data ready interrupt** The data ready interrupt is provided by the A/D converters, one per channel.

For burst conversion mode, the interrupt shows that a conversion burst has finished and the new conversion results are available. If a burst is terminated before it has reached the number of specified conversions, the interrupt is generated when the current conversion has finished and the incomplete conversion results are available.

For single conversion mode, the interrupt shows that a conversion has finished and the new conversion result is available.



**Failure interrupt** The failure interrupt is provided by the A/D converters, one per channel.

The interrupt shows that a failure occurred. To get detailed information on the failure, read the failure information by using the `ADC_CLASS1_FAILURE_BLx` block in your model.

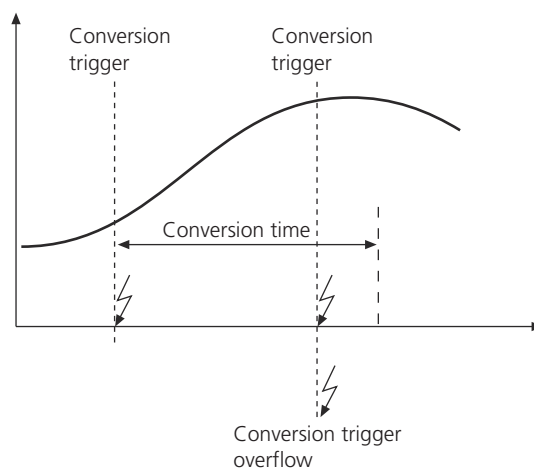
The following failure types can be detected:

- **No burst start**  
This failure shows that a burst trigger is missing. The start of a conversion within a burst was requested but the burst was not already started.
- **Data lost**  
This failure shows that a filled free buffer in the swinging buffer was overwritten by new conversion results before the old conversion results were read. This condition occurs if conversion bursts are started more frequently than the buffers are read. For more information on the swinging buffer, refer to [Swinging Buffer](#) on page 41.
- **Store error**  
This failure shows that a value could not be stored.
- **Burst trigger overflow**  
This failure shows that a burst trigger was received before the preceding burst finished. The burst trigger is ignored.



- Conversion trigger overflow

This failure shows that a conversion trigger was received before the preceding conversion finished. The conversion trigger is ignored.



## RTI/RTLib support

You can access the ADC Class 1 unit via RTI and RTLib.

For details, see:

- RTI:

[A/D Conversion \(MicroLabBox RTI Reference !\[\]\(17413706fd4997a1a4bdf85c6864eee1\_img.jpg\)](#))

- ADC\_CLASS1\_BLx
- ADC\_CLASS1\_START\_BLx
- ADC\_CLASS1\_FAILURE\_BLx
- ADC\_CLASS1\_HWINT\_BLx

- RTLib:

[Analog In Class 1 \(MicroLabBox RTLib Reference !\[\]\(d3102649f02e825ddb76dc3de0190154\_img.jpg\)](#))

## I/O mapping

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Analog I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(c045a398c48fcb47adf237d338b1b391\_img.jpg\)](#))
- [Analog In and Analog Out Connectors \(BNC\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(6ea471090ba6b2c70129dc83eb6e6a11\_img.jpg\)](#))
- [Analog In Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(943b1c41f252b081e01aba2e7830f1c9\_img.jpg\)](#))

For detailed information on the channel characteristics, refer to [Analog Class 1 Inputs \(MicroLabBox Hardware Installation and Configuration !\[\]\(3342c215b2a8b663596a81468d5dc314\_img.jpg\)](#)).

**Related topics****Basics**

[Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(99f58673407353e96a019fbca558fd72\_img.jpg\)\)](#)

## ADC Class 2

**Introduction**

The ADC Class 2 unit provides free-running A/D converters. The features are described in detail.

**Characteristics**

The ADC Class 2 unit consists of 8 parallel A/D converters. They are equipped with differential inputs.

Each of the 8 A/D conversion channels provides:

- 14-bit resolution
- A fixed sample rate of 10 MSPS (conversion time of 100 ns)
- An input voltage range of -10 ... +10 V

**Conversion behavior**

The incoming signals are continuously converted to the interval of the sample rate and can be read from a register.

**RTI/RTLib support**

You can access the ADC Class 2 unit via RTI and RTLib.

For details, see:

- RTI:
  - [A/D Conversion \(MicroLabBox RTI Reference !\[\]\(3dc92c626ede9fa1b47e2e010104b5c4\_img.jpg\)\)](#)
    - ADC\_CLASS2\_BLx
- RTLib:
  - [Analog In Class 2 \(MicroLabBox RTLib Reference !\[\]\(71e9a2c5583c3d2a2fe005f4239e5d39\_img.jpg\)\)](#)

**I/O mapping**

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Analog I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(0d11e49c561fa84a6677bf9d4d629be0\_img.jpg\)\)](#)
- [Analog In and Analog Out Connectors \(BNC\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(ef9c29ef42528ebedc716879fbee1e10\_img.jpg\)\)](#)

- [Analog In Class 2 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(6302aad5aed157b291fddf37b4870784\_img.jpg\)](#))

For detailed information on the channel characteristics, refer to [Analog Class 2 Inputs \(MicroLabBox Hardware Installation and Configuration !\[\]\(c507f772dba2b921f86777f01218e570\_img.jpg\)](#)).

## D/A Conversion

### Introduction

MicroLabBox provides one D/A conversion unit.

## DAC Class 1

### Introduction

The DAC Class 1 unit provides D/A converters. The configurable features are described in detail.

### Characteristics

The DAC Class 1 unit consists of 16 D/A converters equipped with single-ended outputs.

Each of the 16 D/A conversion channels provides:

- 16-bit resolution
- A maximum settling time (to 1 LSB) of 1  $\mu$ s
- An output voltage range of -10 V ... +10 V

### Synchronous update

An instantiated DAC Class 1 unit can handle up to 16 output channels. The channel signals that you specified for one unit are synchronously output.

### RTI/RTLib support

You can access the DAC Class 1 unit via RTI and RTLib.

For details, see:

- RTI: [D/A Conversion \(MicroLabBox RTI Reference !\[\]\(3292f5442e3b4027aa0bb60988f9fc82\_img.jpg\)](#))
  - DAC\_CLASS1\_BLx
- RTLib: [Analog Out Class 1 \(MicroLabBox RTLib Reference !\[\]\(705a9285b25462d2e675759828e0d2ed\_img.jpg\)](#))

### I/O mapping

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Analog I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(dfca7b7ba04af5e0fd8f631088951778\_img.jpg\)](#))
- [Analog In and Analog Out Connectors \(BNC\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(bf1925d4ff0ffe3d685948333cd87bdf\_img.jpg\)](#))
- [Analog Out Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(9b90727c7cda6f1230465e201d8421c6\_img.jpg\)](#))

For detailed information on the channel characteristics, refer to [Analog Class 1 Outputs \(MicroLabBox Hardware Installation and Configuration !\[\]\(8af806fb1314382d09bc5ec5b767526c\_img.jpg\)\)](#).

## Bit I/O

### Introduction

MicroLabBox provides two Bit I/O units with different characteristics.

### Where to go from here

### Information in this section

Bit I/O (DIO Class 1).....	54
Bit I/O (DIO Class 2).....	56

## Bit I/O (DIO Class 1)

### Introduction

The Bit I/O feature is based on the DIO Class 1 unit providing bitwise access to the single-ended digital I/O channels.

### Characteristics

The Bit I/O feature of the DIO Class 1 unit consists of single-ended digital I/O channels:

- 48 bidirectional channels that can be configured as input or output
- Grouped on three ports with 16 channels each
- Generation of interrupts and trigger lines started by the specified edge type. For information, see [Interrupts and trigger lines provided by the Bit I/O feature](#) on page 55


The following characteristics are relevant to the channels used as inputs:

- You can read the value from a single channel or from all the channels (up to one complete port) specified for the current unit.
- You can specify a noise filter. Signals that are shorter than the specified time interval are ignored.

The following characteristics are relevant to the channels used as outputs:

- You can specify the output voltage to:
  - 2.5 V
  - 3.3 V
  - 5.0 V

The assignment of I/O channels can only be done within one port. This ensures data consistency.

After power-up when no application is running, all outputs are set to the high impedance state (tristate). For further information, refer to [Digital Class 2 I/O \(Bidirectional\)](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

## Interrupts and trigger lines provided by the Bit I/O feature

The hardware interrupts from the Bit I/O feature are used to trigger interrupt-driven tasks which are executed on MicroLabBox. Trigger lines are used to transmit a trigger signal to an internal or external consumer of the trigger. You can configure to generate either interrupts or trigger signals, or both.

**Event types** You can specify the following event types which are to start the generation of an interrupt or trigger signal:

- Rising edge
- Falling edge
- Both edges

**Handling interrupts** The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(83f22ed94ec5517769dd76d702c6bfd8\_img.jpg\)](#)).

The interrupt which is to trigger the subsystem must be made available using the DIO\_CLASS1\_HWINT\_BLx block. The appropriate function identifier, port number, channel number and the event type in the block's dialog have to be selected. The model must contain a DIO\_CLASS1\_BIT\_IN\_BLx or DIO\_CLASS1\_BIT\_OUT\_BLx block with the same port and channel specified for interrupt generation and the same event type.

### Note

You need a separate DIO\_CLASS1\_HWINT\_BLx block for each interrupt that you want to use on a digital I/O channel.

**Handling trigger lines** The functions you want to trigger by trigger line must be configured to listen to the specified trigger line.

The following I/O functions can be triggered by trigger line:

- ADC Class 1
- Pulse Signal Generation (DIO Class 1)
  - With the generated pulse signal, you can trigger any external device.
- EnDat Interface
- SSI Interface

For further information, refer to [Interrupt Handling](#) on page 29 and [Trigger Signals](#) on page 31.

## RTI/RTLib support

You can access the Bit I/O feature of the DIO Class 1 unit via RTI and RTLib.

For details, see:

- RTI:
  - [Bit I/O \(MicroLabBox RTI Reference !\[\]\(307ad7be8dd8053938b04a332782a8a1\_img.jpg\)](#))
  - DIO\_CLASS1\_BIT\_IN\_BLx
  - DIO\_CLASS1\_BIT\_OUT\_BLx
  - DIO\_CLASS1\_HWINT\_BLx

- RTLib:
  - [Bit I/O \(MicroLabBox RTLib Reference !\[\]\(cd3e54d951a9fb854f48e4697cf550f9\_img.jpg\)\)](#)
  - Digital In Class 1
  - Digital Out Class 1

## I/O mapping

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(448bd415caa8b52d2aeb4d58499267b2\_img.jpg\)\)](#)  
DIO1 ch 1 ... DIO1 ch 32
- [Digital I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(23be4c52910c50d5908bb101588c4f4e\_img.jpg\)\)](#)  
DIO1 ch 33 ... DIO1 ch 48
- [Digital I/O Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(5dc449795a3a9c8d29c257423584cf78\_img.jpg\)\)](#)  
DIO1 ch 1 ... DIO1 ch 48

For detailed information on the channel characteristics, refer to:

- [Digital Class 1 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(a9a7cf821bf949be41db724492f295be\_img.jpg\)\)](#)

## Related topics

### Basics

[Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(e1c624d4757f08486e89482c18364c17\_img.jpg\)\)](#)

# Bit I/O (DIO Class 2)

## Introduction

The Bit I/O feature is based on the DIO Class 2 unit providing bitwise access to the differential digital I/O channels.

## Characteristics

The Bit I/O feature of the DIO Class 2 unit consists of differential digital I/O channels:

- 12 bidirectional channels that can be configured as input or output.

## RTI/RTLib support

The Bit I/O feature of the DIO Class 2 unit is not supported by RTI and RTLib.





The I/O channels of the DIO Class 2 unit are also used by some features of the electric motor control support, see [Electric Motor Control](#) on page 91.

---

## I/O mapping

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O B Connector \(Sub-D\)](#) ([MicroLabBox Hardware Installation and Configuration](#) )
- [Digital I/O Class 2 Connectors \(Spring-Cage\)](#) ([MicroLabBox Hardware Installation and Configuration](#) )

For detailed information on the channel characteristics, refer to:

- [Digital Class 2 I/O \(Bidirectional\)](#) ([MicroLabBox Hardware Installation and Configuration](#) )

## Timing I/O

### Introduction

MicroLabBox provides different features to generate and measure signal patterns.

### Where to go from here

#### Information in this section

PWM Signal Generation (DIO Class 1).....	58
PWM Signal Measurement (DIO Class 1).....	62
Pulse Signal Generation (DIO Class 1).....	65
Pulse Width Measurement (DIO Class 1).....	67

#### Information in other sections

Multichannel PWM Signal Generation (DIO Class 1).....	112
Block-Commutated PWM Signal Generation (DIO Class 1).....	108

## PWM Signal Generation (DIO Class 1)

### Introduction

The PWM signal generation feature is based on the DIO Class 1 unit providing access to the single-ended digital I/O channels.

### Characteristics

This I/O feature is used to generate pulse-width modulated (PWM) signals on one digital output channel.

- 48 bidirectional channels can be configured as output for PWM signal generation.  
The channels are grouped on three ports with 16 channels each.
- Adjustable voltage level for the output: 2.5 V, 3.3 V, 5.0 V
- Adjustable PWM period in the range 100 ns ... 1.34 s
- Adjustable duty cycle
- Generation of interrupts and trigger lines as events started by the specified edge type
- Configuration of the event behavior via downsampling and delay, see [Interrupts and trigger lines provided by the PWM signal generation feature](#) on page 59.
- Adjustable update mode, see [Update mode](#) on page 60.

- Possibility to invert the output signal, see [Inverting mode](#) on page 61.
- Adjustable delay between the occurrence of a rising edge and the generation of a signal (rising edge delay, only supported by RTLib)

---

**PWM period range**

The PWM period can be specified in the range 100 ns ... 1.34 s.  
The value can be specified in steps of 10 ns.

---

**Interrupts and trigger lines provided by the PWM signal generation feature**

The hardware interrupts from the PWM signal generation feature are used to trigger interrupt-driven tasks which are executed on MicroLabBox. Trigger lines are used to transmit a trigger signal to an internal or external consumer of the trigger. You can configure to generate either interrupts or trigger signals, or both.

**Event types** You can specify the following event types which has to start the generation of an interrupt or trigger signal:

- Rising edge
- Falling edge
- Both edges

**Handling interrupts** The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide\)](#). The interrupt which is to trigger the subsystem must be made available using the DIO\_CLASS1\_HWINT\_BLx block. The appropriate function identifier, port number, channel number and the event type in the block's dialog have to be selected. The model must contain a DIO\_CLASS1\_PWM\_BLx block with the same port and channel specified for interrupt generation and the same event type.

**Note**

You need a separate DIO\_CLASS1\_HWINT\_BLx block for each interrupt that you want to use on a digital I/O channel.

**Handling trigger lines** The functions you want to trigger by trigger line must be configured to listen to the specified trigger line.

The following I/O functions can be triggered by trigger line:

- ADC Class 1
- Pulse Signal Generation (DIO Class 1)  
With the generated pulse signal, you can trigger any external device.
- EnDat Interface
- SSI Interface

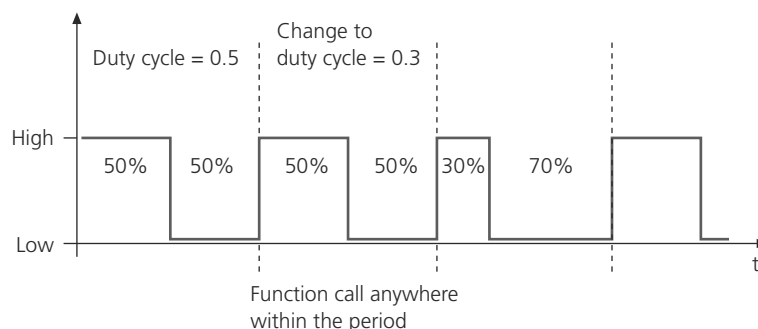
For further information, refer to [Interrupt Handling](#) on page 29 and [Trigger Signals](#) on page 31.

### Update mode

A PWM signal is specified by its period and duty cycle. To change the signal during run time, you can specify new values for the period and the duty cycle.

**Synchronous update mode** New values for the period and/or the duty cycle are updated at the next rising edge of the PWM output signal. The update is synchronous for constant period values only.

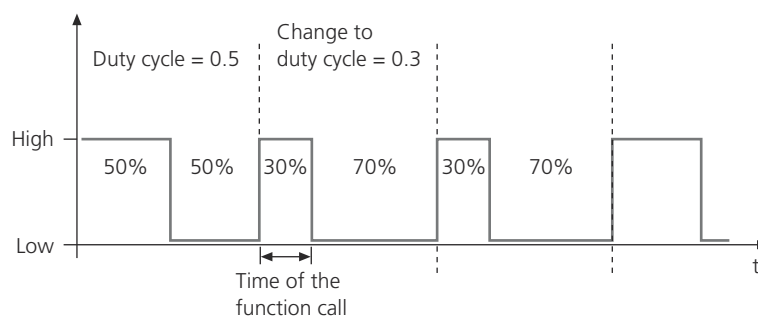
The following figure shows an example of how the duty cycle is updated in synchronous update mode.



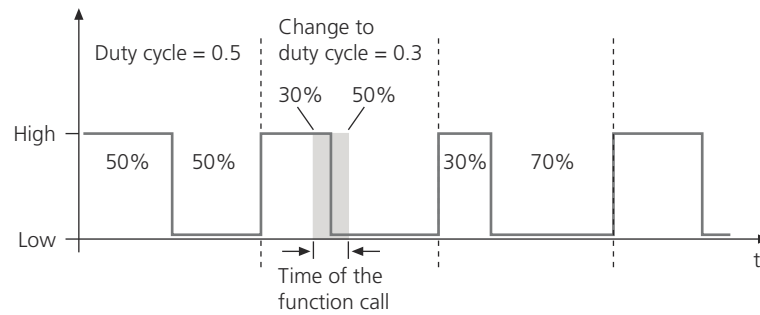
**Asynchronous update mode** New values for the period and/or the duty cycle are updated immediately. The update is asynchronous to the period. This can result in period and/or duty cycle values that differ from the old *and* the new values for one period.

If you want to extend the current duty cycle, the update is immediately executed when you call the update during high level. New values during low level are updated at the next period.

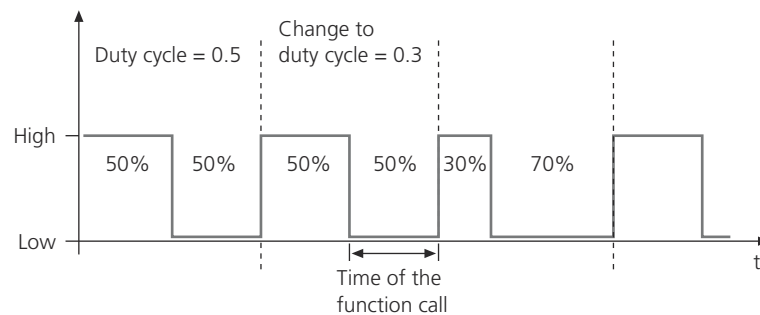
If you want to reduce the current duty cycle, the update depends also on the position within the period. The following figures show how the duty cycle is updated in asynchronous update mode.



If the function call for updating is executed during the high level before the end of the new high level has been reached, the update takes place within the same period.



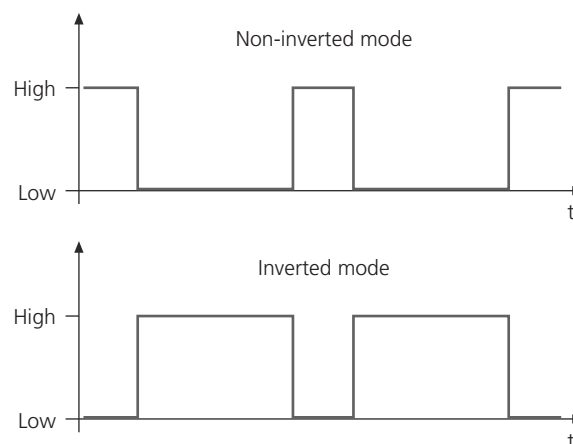
If the function call for updating is executed between the end of the old and the end of the new high level, the update results in a duty cycle between the old and the new value in the current period and a fully updated duty cycle in the next period.



If the function call for updating is executed during the low level, the update takes place in the next period.

### Inverting mode

The generation of a PWM signal starts with a rising edge by default. With the inverting mode, you can configure to invert the output signal.



**RTI/RTLib support**

You can access the PWM signal generation feature of the DIO Class 1 unit via RTI and RTLib.

For details, see:

- RTI:  
[PWM Signal Generation \(MicroLabBox RTI Reference !\[\]\(065aacad479feea1b3f501fa02b79a7a\_img.jpg\)\)](#)
  - DIO\_CLASS1\_PWM\_BLx
- RTLib:  
[PWM Signal Generation \(PWM Out\) \(MicroLabBox RTLib Reference !\[\]\(f90d8b6badff022f4fa9e71b17a20969\_img.jpg\)\)](#)

**I/O mapping**

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(661ad2fdbe8fa1392f2b194cfa45d124\_img.jpg\)\)](#)  
DIO1 ch 1 ... DIO1 ch 32
- [Digital I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(4193cdf1061c98ac39c3073e7f9019f2\_img.jpg\)\)](#)  
DIO1 ch 33 ... DIO1 ch 48
- [Digital I/O Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(4caf182c2ec1a7bf8758f380863453a1\_img.jpg\)\)](#)  
DIO1 ch 1 ... DIO1 ch 48

For detailed information on the channel characteristics, refer to [Digital Class 1 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(73002692dd5e7a64e60946be3158e719\_img.jpg\)\)](#).

**Related topics****Basics**

<a href="#">Block-Commutated PWM Signal Generation (DIO Class 1).....</a>	<a href="#">108</a>
<a href="#">Multichannel PWM Signal Generation (DIO Class 1).....</a>	<a href="#">112</a>
<a href="#">Tasks Driven by Interrupt Blocks (RTI and RTI-MP Implementation Guide )</a>	

## PWM Signal Measurement (DIO Class 1)

**Introduction**

The PWM signal measurement feature is based on the DIO Class 1 unit providing access to the single-ended digital I/O channels.

## Characteristics

This I/O feature is used to measure the frequency and duty cycle of pulse-width modulated (PWM) signals on one digital input channel.

- 48 bidirectional channels can be configured as input for PWM signal measurement.

The channels are grouped on three ports with 16 channels each.

- Generation of interrupts and trigger lines as events started by the specified edge type.
- Configuration of the event behavior via downsampling and delay, see [Interrupts and trigger lines provided by the PWM signal measurement feature](#) on page 63.
- Adjustable timeout, see [Timeout](#) on page 64

## Measurement range

You can measure frequencies in the range 0.5 Hz ... 10 MHz with a resolution of 10 ns.

### Note

At the upper frequency limit, the duty cycle can only be approximately measured because of the limited time resolution.

If the frequency of the input signal is lower than the minimum value, a frequency of 0.0 Hz is returned. Then, the duty cycle is set to 0.0 or 1.0 according to the current signal level.

## Interrupts and trigger lines provided by the PWM signal measurement feature

The hardware interrupts from the PWM signal measurement feature are used to trigger interrupt-driven tasks which are executed on MicroLabBox. Trigger lines are used to transmit a trigger signal to an internal or external consumer of the trigger. You can configure to generate either interrupts or trigger signals, or both.

**Event types** You can specify the following event types which has to start the generation of an interrupt or trigger signal:

- Rising edge
- Falling edge
- Both edges

**Handling interrupts** The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide\)](#). The interrupt which is to trigger the subsystem must be made available using the DIO\_CLASS1\_HWINT\_BLx block. The appropriate function identifier, port number, channel number and the event type in the block's dialog have to be selected. The model must contain a DIO\_CLASS1\_PWM2D\_BLx block with the same port and channel specified for interrupt generation and the same event type.

**Note**

You need a separate `DIO_CLASS1_HWINT_BLx` block for each interrupt that you want to use on a digital I/O channel.

**Handling trigger lines** The functions you want to trigger by trigger line must be configured to listen to the specified trigger line.

The following I/O functions can be triggered by trigger line:

- ADC Class 1
- Pulse Signal Generation (DIO Class 1)  
With the generated pulse signal, you can trigger any external device.
- EnDat Interface
- SSI Interface

For further information, refer to [Interrupt Handling](#) on page 29 and [Trigger Signals](#) on page 31.

**Timeout**

For the minimum measuring value of 0.5 Hz, the period will last 1 second. This is the time that you have to wait for a new value and that is required to detect that the frequency is lower than the lower limit of the measuring range.

If you do not need the full measuring range, you can specify a timeout value lower than 1 second to detect deviations of the expected frequency range faster.

You can specify a timeout value in the range 50 µs ... 1.0 s.

The value can be specified in steps of 10 ns.

**RTI/RTLib support**

You can access the PWM signal measurement feature of the DIO Class 1 unit via RTI and RTLib.

For details, see:

- RTI:  
[PWM Signal Measurement \(MicroLabBox RTI Reference !\[\]\(e04a2df4a948cc496cda3a868d1e74be\_img.jpg\)\)](#)
  - `DIO_CLASS1_PWM2D_BLx`
- RTLib:  
[PWM Signal Measurement \(PWM In\) \(MicroLabBox RTLib Reference !\[\]\(c975569833cee78db62fbb5425c2a66b\_img.jpg\)\)](#)

**I/O mapping**

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.



For a detailed connector pinout, refer to:

- [Digital I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(c8dce68b26731c7aa5915072fc9d68dd\_img.jpg\)](#))

DIO1 ch 1 ... DIO1 ch 32

- [Digital I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(f15d3c54be60b4fd0ce1da9fb3f67256\_img.jpg\)](#))

DIO1 ch 33 ... DIO1 ch 48

- [Digital I/O Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(b1b781be830eb908d845c527ab08d5f8\_img.jpg\)](#))

DIO1 ch 1 ... DIO1 ch 48

For detailed information on the channel characteristics, refer to [Digital Class 1 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(17413706fd4997a1a4bdf85c6864eee1\_img.jpg\)](#)).

## Related topics

### Basics

[Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(d3102649f02e825ddb76dc3de0190154\_img.jpg\)](#))

# Pulse Signal Generation (DIO Class 1)

## Introduction

The pulse signal generation feature is based on the DIO Class 1 unit providing access to the single-ended digital I/O channels.

## Characteristics

This I/O feature is used to generate a pulse signal on one digital output channel.

- 48 bidirectional channels can be configured as output for pulse signal generation.

The channels are grouped on three ports with 16 channels each.

- Adjustable pulse width
- Adjustable voltage level for the output: 2.5 V, 3.3 V, 5.0 V
- Pulse generation controlled by software or trigger line, see Trigger conditions for generating a pulse signal.
- Possibility to invert the output signal, see [Inverting mode](#) on page 66.

## Pulse width range

The pulse width can be specified in the range 100 ns ... 0.5 s.

The value can be specified in steps of 10 ns.

## Trigger conditions for generating a pulse signal

The generation of a pulse signal can be controlled by software or by a trigger line.

**Triggering by software** The software trigger is able to react on the output signal of another I/O feature, for example, a generated PWM signal.

Using RTI, you can enable a trigger port that can be configured for the following event types:

- Rising edge
- Falling edge
- Both edges
- Function call

With RTLib, you can implement the pulse generation controlled by any condition in your application.

**Triggering by trigger line** You can configure the pulse signal generation feature to listen to a trigger line. There must be another I/O feature in your real-time application that supports generating and transmitting trigger signals via trigger line. With the generated pulse signal, you can forward the trigger to any external device connected to the output channel of the pulse signal generation feature.

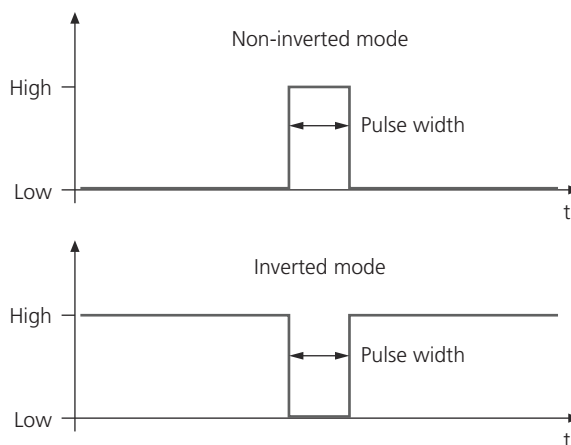
The following I/O functions can generate a signal on a trigger line:

- ADC Class 1
- Bit I/O Class 1
- PWM Signal Generation (DIO Class 1)
- PWM Signal Measurement (DIO Class 1)
- Pulse Signal Measurement (DIO Class 1)
- Serial Peripheral Interface (DIO Class 1)

For further information, refer to [Trigger Signals](#) on page 31.

## Inverting mode

The generation of a pulse signal starts with a rising edge by default. The pulse width specifies the duration of the high time of the signal. If you invert the signal, the pulse width specifies the duration of the low time of the signal, while the rest of the output signal is set to high level.



**RTI/RTLib support**

You can access the pulse signal generation feature of the DIO Class 1 unit via RTI and RTLib.

For details, see:

- RTI:  
[Pulse Generation \(MicroLabBox RTI Reference !\[\]\(aca6fcc8bd95e8255b9ea1b1d08ef300\_img.jpg\)\)](#)
  - DIO\_CLASS1\_PULSE\_BLx
- RTLib:  
[Pulse Signal Generation \(Pulse Out\) \(MicroLabBox RTLib Reference !\[\]\(0083087c61cec498ac803a4aec5bb1bd\_img.jpg\)\)](#)

**I/O mapping**

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(e492b5d52ab457a7a3c2826c4091dfee\_img.jpg\)\)](#)  
DIO1 ch 1 ... DIO1 ch 32
- [Digital I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(1d9440fab1f214291ce1c26a75f9c2cd\_img.jpg\)\)](#)  
DIO1 ch 33 ... DIO1 ch 48
- [Digital I/O Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(6be2e1cb461308cfbb51376f893366b1\_img.jpg\)\)](#)  
DIO1 ch 1 ... DIO1 ch 48

For detailed information on the channel characteristics, refer to [Digital Class 1 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(5361750c22c4e047a52f4eac1ec2d4cc\_img.jpg\)\)](#).

## Pulse Width Measurement (DIO Class 1)

**Introduction**

The pulse width measurement feature is based on the DIO Class 1 unit providing access to the single-ended digital I/O channels.

**Characteristics**

This I/O feature is used to measure the width of a pulse signal on one digital input channel.

- 48 bidirectional channels can be configured as input for pulse width measurement.  
The channels are grouped on three ports with 16 channels each.
- Generation of interrupts and trigger lines as events started by the *data ready* flag.

- Configuration of the event behavior via delay, see [Interrupts and trigger lines provided by the pulse width measurement feature](#) on page 68.
- Adjustable edge polarity, see [Edge polarity](#) on page 69.
- Adjustable maximum pulse width, see [Measurement range](#) on page 68.

---

## Measurement range

You can measure a pulse width in the range 50 ns ... 1.34 s with a resolution of 10 ns.

The pulse width can only be correctly measured if both the high time and the low time are at least 50 ns.

If the pulse width of the square-wave input signal is lower than the minimum value, the resulting pulse width might be erroneous.

If the maximum pulse width is exceeded, the resulting pulse width is the maximum value of the data type used (float64).

**Decreasing the maximum pulse width** The maximum waiting time interval is given by the upper limit of the pulse width range of 1.34 s. This is the time that you have to wait for a new value and that is required to detect that the pulse width is higher than the upper limit of the measuring range.

If you do not need the full measuring range, you can specify an individual maximum pulse width to detect an overflow faster.

You can specify a maximum pulse width in the range 50 µs ... 1.34 s.

The value can be specified in steps of 10 ns.

---

## Interrupts and trigger lines provided by the pulse width measurement feature

The hardware interrupts from the pulse width measurement feature are used to trigger interrupt-driven tasks which are executed on MicroLabBox. Trigger lines are used to transmit a trigger signal to an internal or external consumer of the trigger. You can configure to generate either interrupts or trigger signals, or both.

**Event types** You can specify the following event type which is to start the generation of an interrupt or trigger signal:

- Data ready

**Handling interrupts** The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(248b91fcdac4810ffd15cf33fb6aec6f\_img.jpg\)](#)).

The interrupt which is to trigger the subsystem must be made available using the DIO\_CLASS1\_HWINT\_BLx block. The appropriate function identifier, port number, channel number and the event type in the block's dialog have to be selected. The model must contain a DIO\_CLASS1\_PW2D\_BLx block with the same port and channel specified for interrupt generation and the same event type.

**Note**

You need a separate DIO\_CLASS1\_HWINT\_BLx block for each interrupt that you want to use on a digital I/O channel.

**Handling trigger lines** The functions you want to trigger by trigger line must be configured to listen to the specified trigger line.

The following I/O functions can be triggered by trigger line:

- ADC Class 1
- Pulse Signal Generation (DIO Class 1)  
With the generated pulse signal, you can trigger any external device.
- EnDat Interface
- SSI Interface

For further information, refer to [Interrupt Handling](#) on page 29 and [Trigger Signals](#) on page 31.

**Edge polarity**

You can specify whether the high level section or the low level section of the input signal is to be used for measuring the pulse width.

**RTI/RTLlib support**

You can access the pulse width measurement feature of the DIO Class 1 unit via RTI and RTLlib.

For details, see:

- RTI:  
[Pulse Width Measurement \(MicroLabBox RTI Reference !\[\]\(4695f05050b0d393767d0512587d4e50\_img.jpg\)](#))
  - DIO\_CLASS1\_PW2D\_BLx
- RTLlib:  
[Pulse Width Measurement \(Pulse In\) \(MicroLabBox RTLlib Reference !\[\]\(e6380cce6342e403c00cb7c9feb7e762\_img.jpg\)](#))

**I/O mapping**

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(15cb01d00100e773a50f80002909e9a5\_img.jpg\)](#))  
DIO1 ch 1 ... DIO1 ch 32
- [Digital I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(d8d739ecb1ddc47a32a7c3d18f26efef\_img.jpg\)](#))  
DIO1 ch 33 ... DIO1 ch 48

- [Digital I/O Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(849840539e55921a3851a4ff96d7400d\_img.jpg\)\)](#)

DIO1 ch 1 ... DIO1 ch 48

For detailed information on the channel characteristics, refer to [Digital Class 1 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(eafc244b53721dd1ec133f0772f70fc7\_img.jpg\)\)](#).

---

## Related topics

### Basics

[Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(5a132f13505a6571904d622757b7a8f0\_img.jpg\)\)](#)

# Nonvolatile Data Handling (NVDATA)

Where to go from here

Information in this section

<a href="#">General Information on Handling Nonvolatile Data.....</a>	<a href="#">71</a>
Gives you feature details and information on how to access the board's nonvolatile memory.	
<a href="#">Using the Web Interface for Nonvolatile Data Handling.....</a>	<a href="#">73</a>
The web interface of your MicroLabBox provides a configuration page which lets you manage the data sets stored in the board's nonvolatile memory.	

## General Information on Handling Nonvolatile Data

Introduction

Gives you details about the features and information on how to access the board's nonvolatile memory.

Characteristics

- The nonvolatile data handling feature lets you access the board's nonvolatile memory via a real-time application. The values that you want to write to the memory or that you want to read from it must be provided by a data set.
- Up to 64 data sets can be handled.
  - Up to 64 elements can be contained in one data set.  
Only elements of the same data type can be used in a data set.
  - The maximum memory size available for nonvolatile data is 64 kB.
  - The data transfer is protected by a double buffer mechanism.
  - The data rate for writing a data set is limited to 10 MB/s.

**Note**

The board's nonvolatile memory is a global memory. The data sets can be therefore accessed by several applications running on the hardware. This might cause conflicts, e.g., if an application tries to create a data set with a name that already exists but with a different data type.

To manage the nonvolatile data without running a real-time application, you can use the board's web interface, refer to [Using the Web Interface for Nonvolatile Data Handling](#) on page 73.

**Supported data types**

The elements of a data set must be of the same data type. The following data types can be specified:

Data Type	Meaning
Int8	8-bit integer values Allocates 1 byte
UInt8	8-bit integer values (unsigned) Allocates 1 byte
Int16	16-bit integer values Allocates 2 bytes
UInt16	16-bit integer values (unsigned) Allocates 2 bytes
Int32	32-bit integer values Allocates 4 bytes
UInt32	32-bit integer values (unsigned) Allocates 4 bytes
Single (Float32)	32-bit float values Allocates 4 bytes
Double (Float64)	64-bit float values Allocates 8 bytes

**RTI/RTLib support**

You can access the board's nonvolatile memory via RTI and RTLib.

For details, refer to:

- RTI: [Nonvolatile Data Handling \(NVDATA\) \(MicroLabBox RTI Reference !\[\]\(4decd7f4d36b8b21e9f05326cc7983ef\_img.jpg\)](#))
  - NVDATA\_READ\_BLx
  - NVDATA\_WRITE\_BLx
- RTLib: [Nonvolatile Data Handling \(NVDATA\) \(MicroLabBox RTLib Reference !\[\]\(c3e0af516d5b5e8e8267fd350d6c692b\_img.jpg\)](#))




## Using the Web Interface for Nonvolatile Data Handling

### Introduction


The web interface of your MicroLabBox provides a configuration page that lets you manage the data sets stored in the board's nonvolatile memory. You can view, export, rename, or delete the data sets. You can also format the entire file system to restore its initial state.

### Data set overview

If you click NVDATA in the main menu, the NVDATA Management page opens and the currently stored data sets are displayed with their names, data types, and the number of contained elements. You can delete, view the details of and download each data set to your computer via the relevant commands.



**DS1202**  
**NVDATA Management**



[MAIN](#) [CONFIGURATION](#) [FLASH](#) [USB](#) [NVDATA](#) [SUPPORT](#) [MESSAGES](#) [REBOOT](#)

Deleting or renaming data sets is disabled while an application is running.

**Data sets in NVDATA file system:**

Name	Type	Size	Delete	View	Download
Seat_Position	DOUBLE	16	<a href="#">Delete</a>	<a href="#">View</a>	<a href="#">Download</a>
Mileage	DOUBLE	1	<a href="#">Delete</a>	<a href="#">View</a>	<a href="#">Download</a>
Error_History	UINT32	32	<a href="#">Delete</a>	<a href="#">View</a>	<a href="#">Download</a>

#### Note

- If a real-time application is currently running on the board, deleting a data set, renaming a data set, and formatting the file system is not possible. Before you can execute one of these functions, the real-time application must be stopped and unloaded.
- The current view on the board's nonvolatile memory is only a snapshot. If a real-time application is currently running, the data might be modified while you are looking at it. Refresh the page to update its contents.

### Renaming data sets

You can edit the data set name by clicking the displayed name. To save the modified name, press Enter.

The supported characters for a data set name are [A-Z], [a-z], [0-9], and the underscore '\_'. Whitespaces or special characters are not supported.

### Deleting data sets

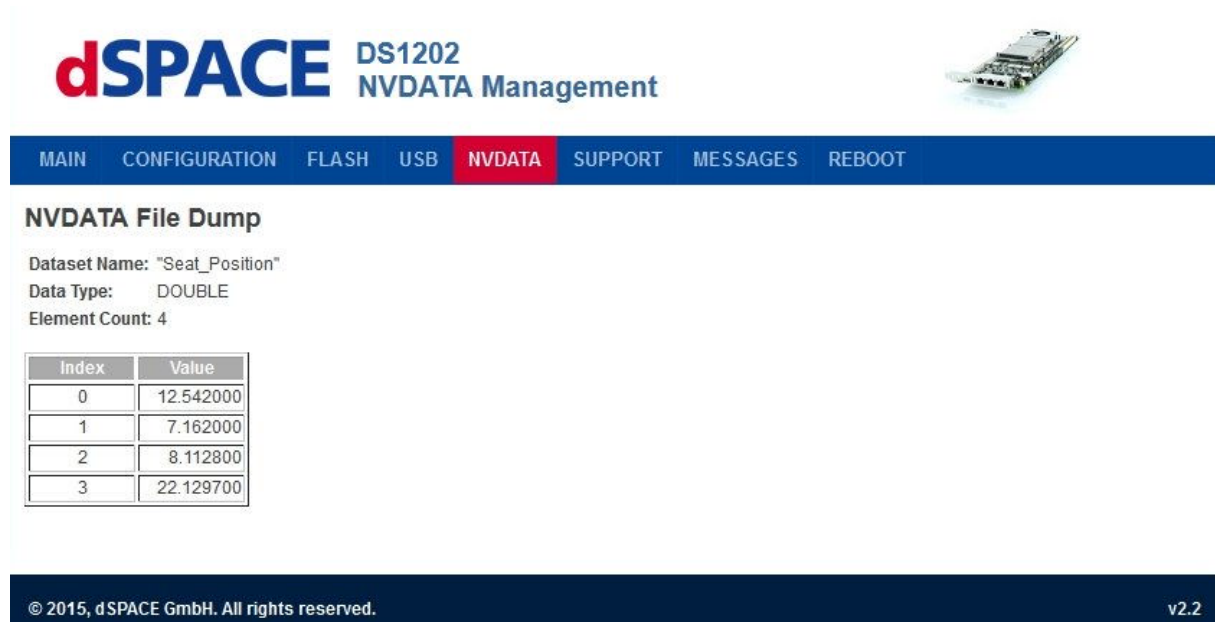
You can delete a data set by clicking its related Delete button.

The data set is deleted without further confirmation. It is not possible to recover the data.

### Viewing data set details

You can view the elements contained in a data set by clicking its related **View** link. The contents of the data set are then displayed.

When you view a data set while an application is running, a temporary snapshot of the data set is displayed.



**dSPACE DS1202 NVDATA Management**

MAIN CONFIGURATION FLASH USB **NVDATA** SUPPORT MESSAGES REBOOT

### NVDATA File Dump

Dataset Name: "Seat\_Position"  
Data Type: DOUBLE  
Element Count: 4

Index	Value
0	12.542000
1	7.162000
2	8.112800
3	22.129700

© 2015, dSPACE GmbH. All rights reserved. v2.2

### Exporting data sets

You can export a data set by using its related **Download** link. Right-click the link to open the context menu and select the **Save target as** command.

The data set is downloaded and saved as a text file in the comma-separated CSV format.

#### Note

Floating-point values are represented with the decimal point. If you want to import the CSV file into a spreadsheet application, another character might be expected, such as the decimal comma. You can replace the characters using any standard text editor.

### Formatting the NVDATA file system

You can format the board's nonvolatile memory to its initial state by clicking the **Format File System** button. All data sets are deleted.

# USB Flight Recorder

**Purpose** With the USB Flight Recorder, you can perform long-term data acquisition. The values of selectable variables are written to the connected USB mass storage device during simulation.

## Where to go from here

### Information in this section

[Basics on USB Flight Recorder..... 75](#)  
[Handling the Data of the USB Flight Recorder..... 79](#)

### Information in other sections

[Running an Application from a USB Mass Storage Device..... 22](#)  
[RTI USB Flight Recorder Blockset Reference](#)  
 Provides concise information on the RTI USB Flight Recorder Blockset.  
[USB Flight Recorder RTLib Reference](#)  
 Provides detailed descriptions of the C functions needed to program RTI-specific Simulink S-functions or implement your real-time models manually via C programs (handcoding).

## Basics on USB Flight Recorder

### General information

The USB Flight Recorder is used to store time histories of real-time variables. The values of real-time variables are written to an externally connected USB mass storage device during real-time simulation.

A maximum of 250 different real-time variables can be recorded.

The recorded data is written to a file in the root directory of the USB mass storage device. The file name is automatically generated and contains the name of the real-time application, the creation date and the creation time.

On multicore platforms such as MicroLabBox, the USB Flight Recorder is separately configured for each real-time application. A separate sequence of output files will be generated for each application.

The default maximum size of a single file is 32 MB. With RTLlib, you can specify a maximum file size of up to 256 MB. If more than the specified maximum file size is recorded during one simulation, the data is split into several files. Consecutive files are created until the storage capacity of the USB device has been reached. Then the captured data is discarded or older files are overwritten, according to your setting (refer to [Memory overwrite mode](#) on page 76).

After the simulation has finished, the recorded data can be read out by the host PC, refer to [Handling the Data of the USB Flight Recorder](#) on page 79.

Before you power down the board, you have to stop flight recording. For information how to terminate a flight recorder session, refer to [Avoiding data loss](#) on page 77.

---

### Memory overwrite mode

You can use RTI or RTLlib functions to define how to handle data when the USB mass storage device for flight recording is full:

**Discard new data (blocked mode)** When the USB mass storage device for flight recording is full, no further data is recorded.

The flight recording session is stopped, but the real-time application continues to run.

**Replace old data (overwrite mode)** When the USB mass storage device for flight recording is full, the oldest files are replaced.

#### Note

On multicore platforms such as MicroLabBox, all real-time applications must use the same memory overwrite mode.

---

### Requirements on the USB mass storage device

Any standard USB 2.0 mass storage device can be used, such as a USB memory stick or an external USB hard drive with or without separate power supply. The USB device must be formatted with the Microsoft FAT32 file system and must be directly connected to your hardware.

#### Note

A connection via a USB hub is not supported.

USB mass storage devices differ according to their rates for writing data. It is recommended to use a fast device for good performance.

The maximum supported file system size is 32 GB. Using a file system with a size greater than 32 GB might work but is neither recommended nor supported.

If you use an external USB hard drive with more than one partition, the flight recorder data is stored only in the first partition.

## Avoiding data loss

The Windows FAT32 file system is not designed to operate in a fail-safe manner. For example, removing a USB memory stick while data is written to it can result in corrupted data.

### Note

#### Risk of data loss

While a USB Flight Recorder session is active:

- Do not unplug the USB device from your hardware.
- Do not switch off your hardware.

You can recognize an active USB Flight Recorder session by the green flashing USB status LED.

To safely remove the USB device while an application is running, follow the instructions in this section.

To avoid partial data loss or corruption of the recorded data, you have to take the following precautions.

**Removing the USB device while an application is running** To safely remove the USB device while an application is running, apply one of the following methods:

- Stop the real-time application.

The USB device can be safely removed as soon as the real-time application is stopped, for example, by using ControlDesk.

- Eject by button.

MicroLabBox provides a USB eject button near the USB connectors that you can use for unmounting the USB device. The unmount procedure is started when you press the button. You can remove the USB device when the USB status LED is off. A host message is also generated.

The USB eject button is placed on the rear panel that is identical for each MicroLabBox variant. For more information on the USB eject button, refer to, e.g., [Housing Components of the MicroLabBox BNC Variant \(MicroLabBox Hardware Installation and Configuration !\[\]\(e50091943b385fe16d3277389202856f\_img.jpg\)](#)).

- Eject by user-defined signal.

You can use the USB\_FLIGHT\_REC\_EJECT block in your Simulink model or the **dsflrec\_usb\_eject** command in your handcoded application to unmount the USB device as a reaction to a signal, for example, a specific model variable.

To restart the USB Flight Recorder, you have to remove the USB device and reconnect it to your hardware.

**Accessing USB Flight Recorder files via FTP**

You can use any standard FTP client to retrieve USB Flight Recorder files without disconnecting the USB device from your hardware. The USB Flight Recorder files are stored in <FTP\_Root>\usb. When the real-time application is not running, reading USB Flight Recorder files via FTP is safe, otherwise the following limitations apply:

- An FTP connection generates a CPU load that might result in partial data loss if the USB flight recording load is high.
- If a file is read via FTP while a flight recording session is running, incomplete data will be retrieved if the flight recorder data is written in overwrite mode.
- Any data capture session running via Ethernet might be disturbed by an FTP connection because the network bandwidth is shared.

**Note**

Do not download flight recorder data while the real-time application is running.

**USB status LED**

The status LED of the USB connector displays the current status of the USB device and the flight recorder.

LED Status	Meaning
Off	No USB device is connected.
Green	USB device is connected and flight recorder is not running.
Green blinking	USB device is connected and flight recorder is running.
Orange	USB device is full and the active flight recorder is specified not to overwrite old files.
Red	Write error when accessing the USB device, for example, if the device was removed while the flight recorder was running.

**Time base**

In the flight recorder, data captures are stored together with time stamps. Time stamps are measured in seconds relative to the time base 01/01/1970. Time stamps are interpreted appropriately by MATLAB or dSPACE experiment software. You can change the time base using M-program code. For an example, refer to [MAT File Format for the USB Flight Recorder \(RTI USB Flight Recorder Blockset Reference\)](#).

Each entry is stored together with a time stamp indicating an absolute date and time value with a resolution of 10.24  $\mu$ s.

**Startup behavior and maximum data rate**

The maximum data rate per application depends on the real-time platform, the USB mass storage device and the number of running applications.

**Using MicroLabBox** A maximum data rate of 8 MB/s is possible without data loss. If separate real-time applications are running on the multicore board, the maximum data rate is totally 8 MB/s.

### Limitations using the USB Flight Recorder

**Stopping the simulation** Do not stop the simulation during recording, for example, by switching the simulation state from *Run* to *Stop*, and then to *Run* again. If the data is not continuously recorded, time-stamping might be corrupted.

**Using a USB mass storage device** There are some limitations when working with a USB mass storage device:

- It is recommended to use a separate USB mass storage device for flight recording. Other files in the root folder of the device will be deleted by the USB Flight Recorder.
- Do not use a USB hub. The device must be directly connected to your hardware.
- If you remove the USB device while data is written, data loss might not be the only problem. In rare cases, the USB driver of the board fails to detect a reconnected USB device. To solve the problem, you have to restart the board.

### RTI/RTLib support

**Using RTI** You can use the RTI blocks from the RTI USB Flight Recorder blockset to write flight recorder data to the USB mass storage device, refer to [Components of the RTI USB Flight Recorder Blockset \(RTI USB Flight Recorder Blockset Reference\)](#).

**Using RTLib** You can use the `dsflrec_usb` RTLib functions to write flight recorder data to the USB mass storage device, refer to [USB Flight Recorder \(USB Flight Recorder RTLib Reference\)](#).

## Handling the Data of the USB Flight Recorder

### Introduction

The data recorded by the USB Flight Recorder can be handled via ControlDesk.

### Loading data to the host PC

After the simulation has finished, the recorded data can be downloaded to the host PC.

If the USB device is connected to your hardware, you can use ControlDesk and its specific functions for USB Flight Recorder handling to access the recorded data. You can select several binary files to download and convert them to CSV or MAT file format and to delete the binary files.

For further information, refer to [How to Upload Flight Recorder Data Written to a USB Mass Storage Device \(ControlDesk Measurement and Recording\)](#).

Alternatively, you can use an FTP client or the File Explorer to download data from the USB mass storage device. Use `ftp://<IP_Address>` to connect to your real-time hardware.


#### Note

Do not delete any files on the USB mass storage device while a flight recorder session is still running.  
For further information, refer to *Accessing USB Flight Recorder files via FTP* in [Basics on USB Flight Recorder](#) on page 75.

If the USB device is directly connected to your PC, you can use ControlDesk's functions to load and convert a single binary file. You can also use a standard file manager, for example, the File Explorer, to copy the recorded binary files to a local drive or to delete them from the USB device.

#### Note

See the section *Avoiding data loss* in [Basics on USB Flight Recorder](#) on page 75 for information on how to safely remove the USB device from your hardware.

For the handling of a great amount of binary files on the USB mass storage device, or if there is no ControlDesk installed on the PC used for postprocessing the flight recorder data, you can use a command line tool for merging, extracting and converting several binary files, refer to [Merging, Extracting and Converting BIN Files of a Flight Recorder](#) ([ControlDesk Measurement and Recording](#) ).

## Related topics

### Basics

[Basics on USB Flight Recorder](#)..... 75



# Serial Interface

## Introduction

MicroLabBox provides a serial interface that can be used to implement a serial communication.

## Serial Interface of MicroLabBox

### UART characteristics

MicroLabBox is equipped with two serial interfaces (UART) to communicate with RS232, RS422, and RS485 devices. The two interfaces are both connected to the RS232 (422/485) Sub-D connector of the board.

The data transfer rates depend on the bus protocol used.

Protocol	Transfer Data Rate
RS232	50 Bd ... 230.4 kBd
RS422/485	50 Bd ... 10 MBd

### RTI/RTLib support

You can access the serial interface via RTI and RTLib.

For details, see:

- RTI:

[Serial Interface \(MicroLabBox RTI Reference !\[\]\(95b425611cbd2b8716a140cf67c81822\_img.jpg\)](#))

- DS1202SER\_SETUP\_Cx
- DS1202SER\_RX\_Cx
- DS1202SER\_TX\_Cx
- DS1202SER\_STAT\_Cx
- DS1202SER\_INT\_Cx\_Iy
- DS1202SER\_INT\_REC\_LEV\_Cx

- RTLib:

[Serial Interface Communication \(MicroLabBox RTLib Reference !\[\]\(3342c215b2a8b663596a81468d5dc314\_img.jpg\)](#))

### I/O mapping

For a detailed connector pinout, refer to:

- [RS232 \(422/485\) Connector \(Sub-D\)](#) (MicroLabBox Hardware Installation and Configuration )

For detailed information on the channel characteristics, refer to:

- [Communication Interfaces](#) (MicroLabBox Hardware Installation and Configuration )

### Related topics

#### References

[Serial Interface](#) (MicroLabBox RTI Reference )  
[Serial Interface Communication](#) (MicroLabBox RTLib Reference )

# Serial Peripheral Interface

## Introduction

MicroLabBox provides serial peripheral interfaces (SPI) that can be used to perform high-speed synchronous communication with devices connected to MicroLabBox, such as an A/D converter.

## Serial Peripheral Interface (DIO Class 1)

### Introduction

The serial peripheral interfaces (SPI) of the DIO Class 1 unit provide high-speed synchronous communication with devices connected to MicroLabBox, such as an A/D converter.

### Characteristics

The DIO Class 1 unit of MicroLabBox provides up to four serial peripheral interfaces (SPI).

The SPI transfers serial bit streams of selectable length and transfer rate from and to external devices. The basic transfer rate for serial data transmission is defined via the clock signal (CLK). This triggers the data transfer between the SPI and a connected external device.

#### Note

- The SPI can be processed only in master mode. It cannot respond to any externally initiated serial transfers.
- The specified output voltage must suit to the connected peripherals.

The SPI supports the following features:

- Up to four chip select channels can be configured.  
By using a decoder, you can select up to 15 independent peripherals.
- Up to 64 chip select cycle configurations per SPI unit can be configured.  
During run time, you only have to reference the specified number of a cycle configuration for transmitting or receiving SPI data.

A chip select cycle configuration is used to specify the following characteristics of an SPI transmission:

- Transfer length by specifying the number of words (1 ... 64) and bits per word (1 ... 128). The transfer length must not exceed 4096 bits.

$$\text{Len} = \text{floor}((\text{BitsPerWord} + 31) / 32) \cdot 32 \cdot \text{Words}$$

Data received by the SPI is stored temporarily in a FIFO buffer of the I/O FPGA. Buffer overflows are indicated by a status information and cause old data to be overwritten.

- Bit direction in a word
- Period of the clock signal defined by the specified baud rate in the range 5 kBd ... 2.5 MBd.
- Polarity and phase of the clock signal (SPI mode)
- Timing behavior of the transmission by specifying the time before and after transfer, the minimum time between two chip select cycles, and the time between words. For detailed information on the timing parameters, see [Timing behavior](#) on page 85.
- Optional, generation of an *end of cycle* event generating an interrupt and/or a trigger signal, see [Interrupts and trigger lines provided by the serial peripheral interface \(SPI\) feature](#) on page 84.

### Interrupts and trigger lines provided by the serial peripheral interface (SPI) feature

The hardware interrupts from the SPI feature are used to trigger interrupt-driven tasks which are executed on MicroLabBox. Trigger lines are used to transmit a trigger signal to an internal or external consumer of the trigger. You can configure to generate either interrupts or trigger signals, or both.

**Event types** You can specify the following event type which has to start the generation of an interrupt or trigger signal:

- End of SPI cycle

**Handling interrupts** The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide\)](#).

The interrupt which is to trigger the subsystem must be made available using the DIO\_CLASS1\_HWINT\_BLx block. The appropriate function identifier, port number, channel number and the event type in the block's dialog have to be selected. The model must contain the DIO\_CLASS1\_SPI\_SETUP\_BLx block with the same port and channel specified for interrupt generation and the same event type.

#### Note

You need a separate DIO\_CLASS1\_HWINT\_BLx block for each interrupt that you want to use on a digital I/O channel.

**Handling trigger lines** The functions you want to trigger by trigger line must be configured to listen to the specified trigger line.

The following I/O functions can be triggered by trigger line:

- ADC Class 1
- Pulse Signal Generation (DIO Class 1)  
With the generated pulse signal, you can trigger any external device.
- EnDat Interface
- SSI Interface

For further information, refer to [Interrupt Handling](#) on page 29 and [Trigger Signals](#) on page 31.

## Timing behavior

The timing behavior mainly depends on the specified **Time between data words** parameter (**TimeBetweenWords** parameter in RTLib). If an SPI cycle consists of several words, you can specify time during which the CLK signal pauses between two subsequent words.

- If you specify 0 for the time between data words, the transfer of each word of an SPI cycle follows the same timing parameters.
- If you specify a value in the range 20 ns ... 800 µs for the time between data words, the other timing parameters only affect the timing behavior of the beginning and end of an entire SPI cycle. The chip select signal is not switched to the inactive state to mark the end of a word in the SPI cycle.

The other timing parameters are (name of the parameters in RTI / RTLib):

- Time before transfer / TimeBeforeTransfer  
Specifies the time between the point at which the CS signal is set to active (beginning of a cycle or beginning of a word) and the first period of the following CLK signal. The relevant edge of the CLK signal depends on the specified clock polarity and clock phase.
- Time after transfer / TimeAfterTransfer  
Specifies the time between the last period of the CLK signal and the point at which the CS signal is set to inactive (end of a cycle or end of a word). The relevant edge of the CLK signal depends on the specified clock polarity and clock phase.
- Time between chip select cycles / CSInactiveTime  
Specifies the minimum time between to cycles or two subsequent words during which the chip select signal is set to onactive.

The following table lists the timing behavior. see below.

The timing parameters can be specified with a maximum value of 800 µs. The value range is internally separated into 12 intervals that are automatically assigned to the specified value. Each interval provides a different step size that is used to saturate a specified value to its next available value.

Interval Number	Lower Limit of Interval	Step Size
1	0 ns	10 ns
2	640 ns	20 ns
3	1.28 µs	40 ns
4	2.56 µs	80 ns

Interval Number	Lower Limit of Interval	Step Size
5	5.12 $\mu$ s	160 ns
6	10.24 $\mu$ s	320 ns
7	20.48 $\mu$ s	640 ns
8	40.96 $\mu$ s	1.28 $\mu$ s
9	81.92 $\mu$ s	2.56 $\mu$ s
10	163.84 $\mu$ s	5.12 $\mu$ s
11	327.68 $\mu$ s	10.24 $\mu$ s
12	655.36 $\mu$ s	20.48 $\mu$ s

**Note**

All four chip select signals of an SPI interface are synchronously switched. But if you decode them for more than four slave devices, the signals might not arrive synchronously at the decoder inputs. The decoder therefore might produce spikes at its outputs, which you should suppress by using an appropriate filter circuit.

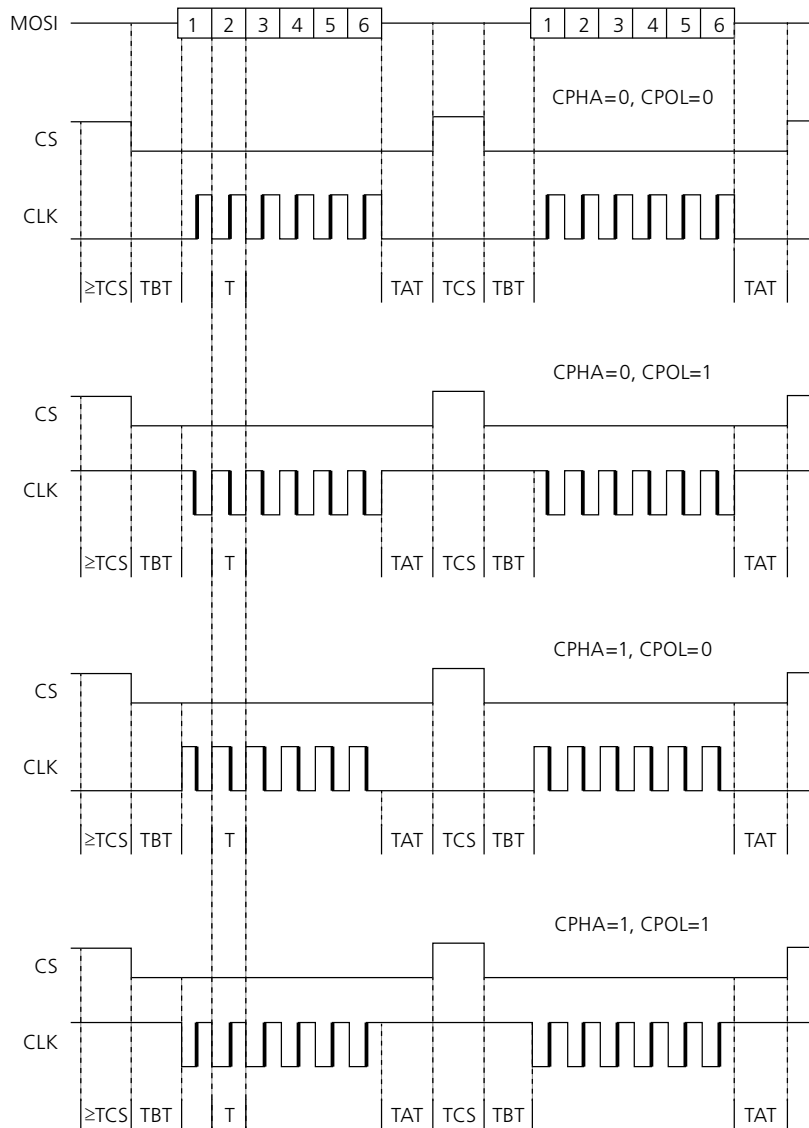
**Examples of the timing behavior**

The SPI cycle consists of two words with six bits each. The polarity of the chip select signal is low active.

The illustrations contain different signal shapes showing the possible combinations of the SPI clock polarity and SPI clock phase. You can see how the timing parameters are considered for the generated outputs.

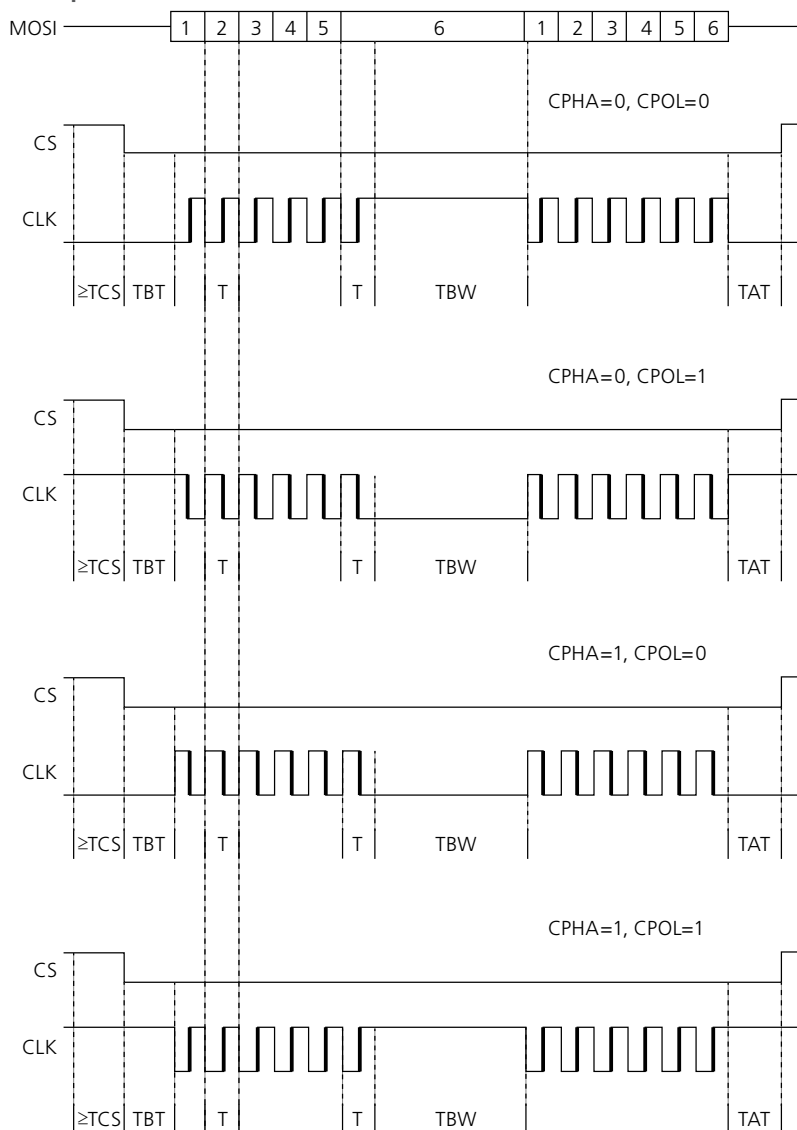
The following abbreviations are used:

Abbreviation	Meaning
T	Period of the SPI clock signal
TBT	Time before transfer
TAT	Time after transfer
TCS	Time chip select signal inactive
TBW	Time between data words
CLK	SPI clock
CPHA	SPI clock phase
CPOL	SPI clock polarity

**Example with time between data words set to 0**

The DIO Class 1 unit writes data to MOSI at the thin edges and reads data from MISO at the bold edges. When no data is transmitted, the MOSI channel is set to High-Z.

**Example with time between data words set to >0**



The DIO Class 1 unit writes data to MOSI at the thin edges and reads data from MISO at the bold edges. Between the words, the last data bit is held on the MOSI channel. When no data is transmitted, the MOSI channel is set to High-Z.

**RTI/RTLib support**

You can access the serial peripheral interface feature of the DIO Class 1 unit via RTI and RTLib.



For details, see:

- RTI:
  - [Serial Peripheral Interface \(MicroLabBox RTI Reference !\[\]\(c8dce68b26731c7aa5915072fc9d68dd\_img.jpg\)](#))
  - DIO\_CLASS1\_SPI\_SETUP\_BLx
  - DIO\_CLASS1\_SPI\_CYCLE\_SETUP\_BLx
  - DIO\_CLASS1\_SPI\_TX\_BLx
  - DIO\_CLASS1\_SPI\_RX\_BLx
- RTLib:
  - [Serial Peripheral Interface \(SPI\) \(MicroLabBox RTLib Reference !\[\]\(76b3245de86167eba9fcdc9cc9f32aa4\_img.jpg\)](#))

## I/O mapping

The following table shows the order of the signals. You have to specify the digital input channel (MISO) and the first digital output channel (CLK). At least two subsequent output channels are allocated for the MOSI and CS1 signals. Three further output channels are optionally allocated for the CS2, CS3 and CS4 signals.

The digital input channel can be configured as trigger source for the generation of an interrupt and/or a trigger signal. For the digital output channels, you can configure the electrical interface.

Signal	Channel	Description
MISO	ChannelIn <sup>1)</sup>	Master In, Slave Out (Data Out at the connected device)
CLK	ChannelOut	SPI clock (also known as Serial clock)
MOSI	ChannelOut + 1	Master Out, Slave In (Data In at the connected device) Automatically reserved related to the specified first output channel for the CLK signal.
CS1	ChannelOut + 2	Chip Select 1 (also known as Slave Select) A chip select channel is used to address a certain SPI slave. Automatically reserved related to the specified first output channel for the CLK signal.
CS2	ChannelOut + 3	Chip Select 2 (optional)
CS3	ChannelOut + 4	Chip Select 3 (optional)
CS4	ChannelOut + 5	Chip Select 4 (optional)

<sup>1)</sup> When using RTI, you have to specify only the first channel to be used for the SPI interface. This will be the MISO channel. All other channels are automatically allocated in the order shown.

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also

printed on the housing of MicroLabBox. You have to consider only the connector panel type.

The number of required channels must not exceed the number of available channels on the selected port.

For a detailed connector pinout, refer to:

- [Digital I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(cf5be311f7b2821912d8009884508fa2\_img.jpg\)](#))

DIO1 ch 1 ... DIO1 ch 32

- [Digital I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(4e333a6106fc298d0ae6dff272a736ef\_img.jpg\)](#))

DIO1 ch 33 ... DIO1 ch 48

- [Digital I/O Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(97faa0168e491544be255cfcab218e9b\_img.jpg\)](#))

DIO1 ch 1 ... DIO1 ch 48

For detailed information on the channel characteristics, refer to:

- [Digital Class 1 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(b6d55d0b173caf9b2505126db01e6158\_img.jpg\)](#))

---

### Related topics

#### Basics

[Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(f60b7a900783ac3fd531bfd9c111be6d\_img.jpg\)](#))

# Electric Motor Control

**Introduction** MicroLabBox provides specific I/O features used for electric motor control.

Where to go from here	Information in this section
	<div><div>Basics on Electric Motor Control.....91</div><div>Hall Sensor Interface.....95</div><div>Incremental Encoder Interface.....98</div><div>Resolver Interface.....101</div><div>EnDat Interface.....103</div><div>SSI Interface.....105</div><div>Block-Commutated PWM Signal Generation (DIO Class 1).....108</div><div>Multichannel PWM Signal Generation (DIO Class 1).....112</div></div>
	<div><div>Information in other sections</div><div><div>RTI Electric Motor Control Blockset Reference</div><div>Provides concise information on the blocks of the RTI Electric Motor Control Blockset.</div></div></div>

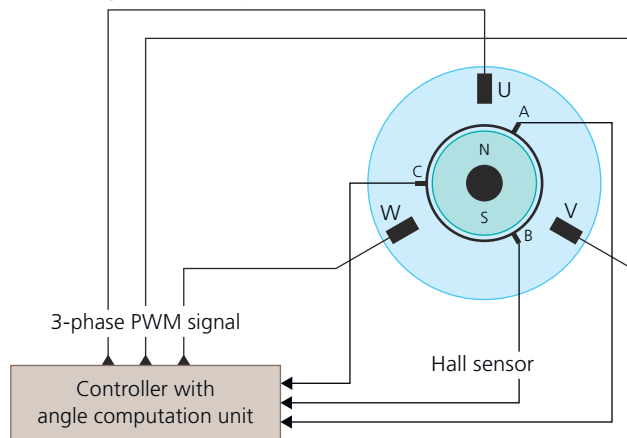
## Basics on Electric Motor Control

**Introduction** MicroLabBox provides a specialized functionality for controlling electric motors.

## General information

The electric motor control feature allows you to implement a controller for an electric motor. You can process input signals from sensors and generate output signals for actors typically used with electric motors. The main component of the controller is the angle computation unit. The calculation of the motor position and speed can be done for electric motors with different numbers of phases and pole pairs. The number of sensor pole pairs can also be configured, so that the resulting electrical angle position can be directly used for the commutation of the motor.

The figure below shows an abstract example of an electric motor with one pole pair, whose rotational position is measured via Hall sensors. The sensor information is used to control the 3-phase PWM signal for increasing or decreasing the motor speed to the nominal value.



**Characteristics** The main characteristics of the electric motor control feature are:

- Up to two controllers can be instantiated.
- Electric motors with up to 6 phases and 16 pole pairs are supported.
- When you use the RTI Electric Motor Control Blockset with the block-commutated PWM signal generation, the number of phases is restricted to 3.
- Up to two sensor units per controller
- Multichannel PWM and block-commutated PWM with a variety of settings are supported as output signals.
- Generation of events triggered by the positions of the sensors

You can specify to generate the following event types:

- Interrupts
  - Interrupt generation for up to four absolute positions or for a periodic event (relative position)
- Trigger signals

You can combine interrupts and trigger signals.

## Interrupts and trigger lines provided by the electric motor control feature

The hardware interrupts from the electric motor control feature are used to trigger interrupt-driven tasks which are executed on MicroLabBox. Trigger lines are used to transmit a trigger signal to an internal or external consumer of the trigger. You can configure to generate either interrupts or trigger signals, or both.

**Event types** You can specify the following event types which has to start the generation of an interrupt or trigger signal:

- Absolute position 1 ... 4
- Relative position by specifying an angle interval and optionally an offset

You can configure whether the event types are based on the electrical or mechanical position.

**Handling interrupts** The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide\)](#). The interrupt which is to trigger the subsystem must be made available using the DIO\_CLASS1\_HWINT\_BLx block. The appropriate function identifier and unit number and the event type in the block's dialog have to be selected. The model must contain an EMC\_MOTOR\_SETUP\_BLx block with the same unit number specified for interrupt generation and the same event type.

### Note

You need a separate DIO\_CLASS1\_HWINT\_BLx block for each interrupt type that you want to use for motor control.

**Handling trigger lines** The functions you want to trigger by trigger line must be configured to listen to the specified trigger line.

The following I/O functions can be triggered by trigger line:

- ADC Class 1
- Pulse Signal Generation (DIO Class 1)
  - With the generated pulse signal, you can trigger any external device.
- EnDat Interface
- SSI Interface

For further information, refer to [Interrupt Handling](#) on page 29 and [Trigger Signals](#) on page 31.

## Sensor inputs

Each controller can be connected with up to two sensor units. The controller handles the two sensor interfaces A and B differently.

- Supported sensor types at sensor interface A are:
  - Hall sensor
    - A Hall sensor can detect the position of a motor immediately after power on. However, the precision of a Hall sensor is very low. For detailed information, refer to [Hall Sensor Interface](#) on page 95.

- Resolver sensor

A resolver sensor can detect the position of a motor immediately after power on with high precision. For detailed information, refer to [Resolver Interface](#) on page 101.

- The supported sensor types at sensor interface B are:

- Incremental encoder

An incremental encoder has a higher precision than a Hall sensor, but it requires up to one revolution of the motor to measure its absolute position. For detailed information, refer to [Incremental Encoder Interface](#) on page 98.

- Absolute encoder connected to the EnDat or SSI interface

An absolute encoder is able to immediately measure the absolute position of a motor. For detailed information, refer to [EnDat Interface](#) on page 103 and [SSI Interface](#) on page 105.

Often, a combination of a Hall sensor unit and an incremental encoder unit is used for the two sensor inputs. The angle computation unit then uses the position from the Hall sensor only as long as no valid position from the higher precision incremental encoder sensor is available.

If you use a resolver sensor, a second sensor at the sensor interface B is usually not required.

If you use an absolute encoder, a second sensor at the sensor interface A is usually not required.

### Validity check

The angle computation unit checks the validity of the measured electrical positions and mechanical positions. The result depends on the number of pole pairs, the sensors used and the occurrence of the index signal of an incremental encoder.

Sensor A	Sensor B	Electrical Position Valid	Mechanical Position Valid
Hall sensor with 1 pole pair	None	✓	✓
Hall sensor with > 1 pole pair	None	✓	–
Hall sensor with 1 pole pair	Incremental encoder	✓	✓
Hall sensor with > 1 pole pair	Incremental encoder	✓	✓ (after first index)
None	Incremental encoder	✓ (after first index)	✓ (after first index)
None	Absolute encoder	✓	✓
Resolver sensor with 1 pole pair	None	✓	✓
Resolver sensor with > 1 pole pair	None	✓	–

### Output signals

To control the electric motor, there must be output channels that react to the sensor inputs. For the signal generation, MicroLabBox provides different types of PWM signal generation.

- Multichannel PWM signal generation

The multichannel PWM signal generation feature is usually used for sinus-commutated output signals. For detailed information, refer to [Multichannel PWM Signal Generation \(DIO Class 1\)](#) on page 112.

- Block-commutated PWM signal generation

The block-commutated PWM signal generation feature provides the settings for block-commutated output signals. For detailed information, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\)](#) on page 108.

## RTI/RTLib support

You can access the electric motor control feature via RTI and RTLib.


### RTI support

#### Note

To use the *RTI Electric Motor Control Blockset*, a separate license is required.

For further information on the RTI blockset, refer to [RTI Electric Motor Control Blockset Reference](#) .

**RTLib support** The RTLib functions for electric motor control are contained in MicroLabBox's RTLib.

For further information, refer to [Electric Motor Control Functions \(MicroLabBox RTLib Reference\)](#) .

## Related topics

### Basics

[Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide\)](#) 

# Hall Sensor Interface

## Introduction

The Hall sensor interface provides access to Hall sensors that react to the magnetic field of the rotor or stator of a motor to determine the motor position.

## Characteristics

A Hall sensor unit usually consists of three inputs, called A, B and C.

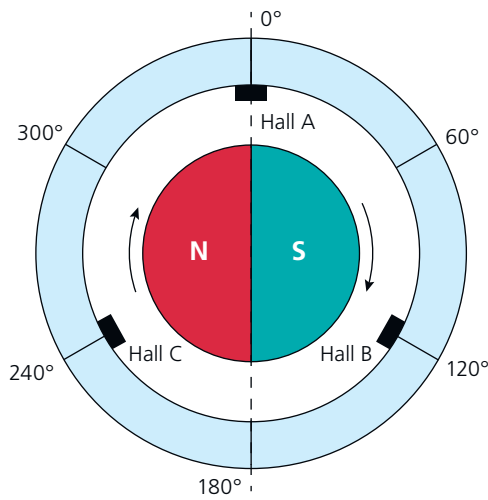
The Hall sensor feature is able to allocate digital input channels of DIO Class 1 type and DIO Class 2 type. The channels of one sensor unit must belong to the same channel class and the same port.

The configuration settings of a Hall sensor are:

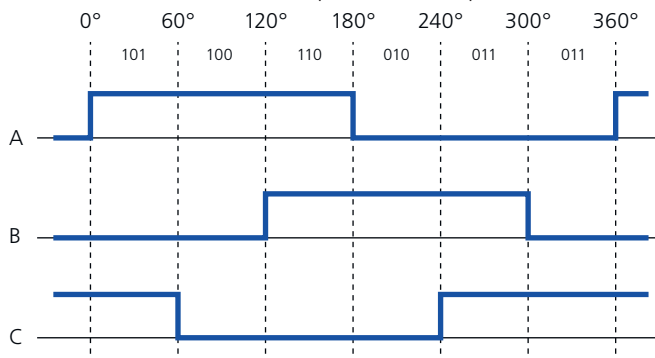
- Definition of sectors for each sensor input by specifying the angular position of a sensor's rising edge and falling edge

- Angular offset of the sectors to adjust the position of the sensor unit
- Noise filter for suppressing non-relevant signal deviations
- Validity check of the measured input signals
- Reversing the orientation to adjust the rotational direction with the direction required by the controller.

#### Example of a Hall sensor configuration



Each sector with a size of 60° provides a unique identifier.



A position is not valid, for example, if all three inputs return 0 or 1.

The configuration of the Hall sensor shown in the figure is described in the following table.

Sensor Input	Rising Edge	Falling Edge
Hall A	0°	180°
Hall B	120°	300°
Hall C	240°	60°



The Hall sensor unit returns the angle value of the current sector's middle.

Sector	Return Value
0°-60°	30°
60°-120°	90°
120°-180°	150°
180°-240°	210°
240°-300°	270°
300°-360°	330°

### RTI/RTLib support

You can access the Hall sensor interface via RTI and RTLib.

For details, see:

- RTI:
  - EMC\_HALL\_BLx
- RTLib:
  - [Hall Sensor Interface \(MicroLabBox RTLib Reference !\[\]\(b1b781be830eb908d845c527ab08d5f8\_img.jpg\)](#))

### I/O mapping

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(039cd6b2e7148ba5690aa619b922c426\_img.jpg\)](#))
- DIO1 ch 1 ... DIO1 ch 32
- [Digital I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(8b9db310e3bd56ffa44f3d5130ea99e2\_img.jpg\)](#))
- DIO1 ch 33 ... DIO1 ch 48
- [Digital I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(49f66b396e80c47181c1b6b90370748d\_img.jpg\)](#))
- DIO2 ch 1 ... DIO2 ch 12
- [Digital I/O Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(f186cdc5336a7be142e8eda07f4bdfc8\_img.jpg\)](#))
- DIO1 ch 1 ... DIO1 ch 48
- [Digital I/O Class 2 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(8d86d9d4a1b60f6ddfe06edafff75620\_img.jpg\)](#))
- DIO2 ch 1 ... DIO2 ch 12

For detailed information on the channel characteristics, refer to

- [Digital Class 1 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(896151ec231b70900e969d67696ca48d\_img.jpg\)](#))
- [Digital Class 2 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(a852c5461f8be0331350e2cc706daa68\_img.jpg\)](#))

## Incremental Encoder Interface

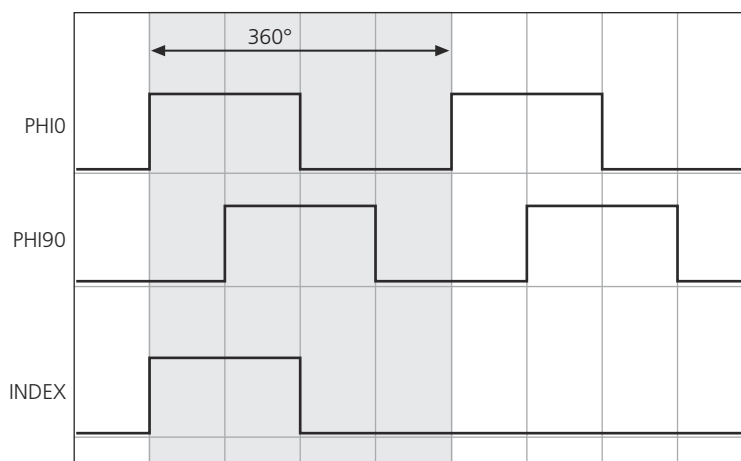
### Introduction

The incremental encoder interface provides access to incremental encoders that determine the motor position.

### Basics on incremental encoders

Incremental encoders provide the two encoder signals PHI0 and PHI90 and the optional index signal. The encoder signals PHI0 and PHI90 have a phase shift of 90°.

The following illustration shows the shape of the PHI0 and PHI90 digital signals together with the optional index signal. It also shows the 4-fold subdivision of an encoder line. The gray area represents one encoder line (360° means one period). For the number of encoder lines per rotation, refer to the encoder user documentation.



A position counter stores the current value. The optional index signal can be used to reset the position counter. The count direction depends on the encoder's rotation direction.

### Characteristics

Each encoder interface can handle positions in the range  $-2^{21} \dots +2^{21}-0.25$  lines (-2,097,152.0 ... +2,097,151.75), including the 4-fold subdivision.

The incremental encoder feature is able to allocate digital input channels of DIO Class 1 type and DIO Class 2 type. Up to six incremental encoder units can be instantiated.

The channels of one incremental encoder unit must belong to the same channel class and the same port. The two (PHI0, PHI90) or three (PHI0, PHI90, IDX) input channels are automatically allocated in subsequent order.

**Note**

- For differential incremental encoders, there are only 12 DIO Class 2 channels available. You can therefore use either up to four differential encoders with index detection or six differential encoders without index detection.
- The total number of encoders must not exceed six. For example, if you have specified four single-ended encoders using DIO Class 1 channels, you can additionally specify two differential encoders using DIO Class 2 channels.

The configuration settings of an incremental encoder are:

- Number of encoder lines the connected incremental encoder supports.  
The maximum number of encoder lines is  $2^{21}$ .
- Individual values for the minimum and maximum positions to adapt the length of the position counter to the resolution of the connected encoder

If the position counter's value reaches the maximum position count value, the minimum position count value is written to the position counter with the next increment. If the position counter's value reaches the minimum position count value, the maximum position count value is written to the position counter with the next decrement.

**Note**

The last line of the connected encoder indicates the next revolution. A revolution ends with **LineMax** - 0.25.

For example, if you have an encoder with 3600 lines and you want to start with zero as minimum position count value, you have to specify 3599.75 as the maximum position count value.

- Index mode to provide a third input channel for the index signal.

You can set the index mode to:

- **No index signal used:** the index pulses are ignored.
- **Reset the counter once:** the counter is reset only after the first index detection.
- **Reset the counter continuously:** the counter is reset after each index detection.

The counter is reset at the first rising edge of the PHI0 or PHI90 signals within the active time of the IDX signal. There must be an inactive time of the IDX signal before you can use the counter reset again. In gated mode, the PHI0 and PHI90 signals must both be high within the active time of the IDX signal to reset the counter.

If the position counter is reset, it starts with the specified index position that must be within the specified minimum and maximum positions.

- **Gated mode for the index signal**

In gated mode, the index signal is ignored unless PHI0 and PHI90 signals are high at the same time.

- Input filter to specify the minimum pulse width of a signal to be used for processing in the range 0 ... 10e-3 s
- Number of data captures used for calculating the speed
- Output flag showing that the index position was passed



The return value of an incremental encoder is the position in encoder lines and the mechanical position as an angle of the motor revolution. The range depends on the encoder settings. The calculated speed can be in the range -10.0e6 ... 10.0e6 lines/s.

---

### RTI/RTLib support

You can access the incremental encoder interface via RTI and RTLib.

For details, see:






- RTI:
  - [EMC\\_ENCODER\\_BLx](#) (RTI Electric Motor Control Blockset Reference )
- RTLib:
  - [Incremental Encoder Interface](#) (MicroLabBox RTLib Reference )

---

### I/O mapping

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O A Connector \(Sub-D\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO1 ch 1 ... DIO1 ch 32
- [Digital I/O B Connector \(Sub-D\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO1 ch 33 ... DIO1 ch 48
- [Digital I/O B Connector \(Sub-D\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO2 ch 1 ... DIO2 ch 12
- [Digital I/O Class 1 Connectors \(Spring-Cage\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO1 ch 1 ... DIO1 ch 48
- [Digital I/O Class 2 Connectors \(Spring-Cage\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO2 ch 1 ... DIO2 ch 12

For detailed information on the channel characteristics, refer to

- [Digital Class 1 I/O \(Bidirectional\)](#) (MicroLabBox Hardware Installation and Configuration )
- [Digital Class 2 I/O \(Bidirectional\)](#) (MicroLabBox Hardware Installation and Configuration )

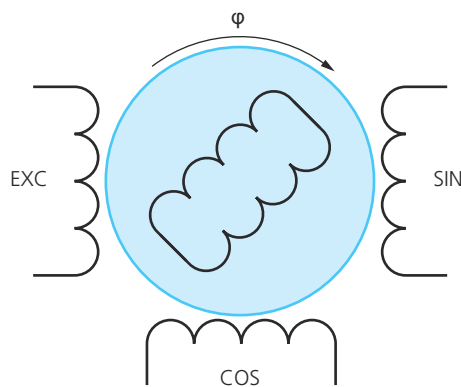
# Resolver Interface

## Introduction

The resolver interface provides access to resolvers that determine the motor position.

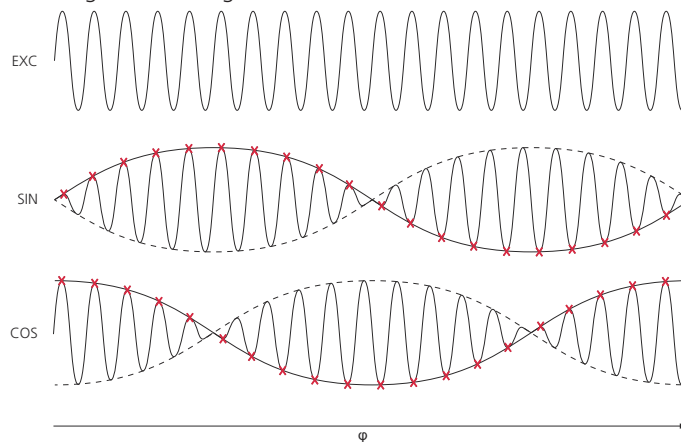
## Basics on resolvers

The resolver interface of MicroLabBox supports only resolvers with an excited single coil. The sinusoidal signal of the excited coil causes position-dependent induction signals in the non-excited coils. The non-excited coils are orthogonally arranged. The resulting induction signals are therefore a sine signal and a cosine signal.



The resolver interface manages the generation of the excitation signal and the processing of the induced signals. It provides the electrical rotor position. The position of the motor to be controlled is calculated by the angle computation unit according to its configuration, such as the number of its pole pairs.

The following illustration shows the shape of the excitation signal EXC and the resulting induction signals SIN and COS for one revolution.



## Characteristics

MicroLabBox provides two resolver interfaces. The resolver interfaces can be separately adapted to the characteristics of the connected resolver sensors.

The configuration settings of a resolver interface are:

- Excitation frequency adjustable in the range 2 ... 20 kHz in steps of 250 Hz
- Excitation voltage adjustable to 3 V<sub>RMS</sub>, 7 V<sub>RMS</sub>, or 10 V<sub>RMS</sub>
- Induction voltage adjustable to 1.5 V<sub>RMS</sub>, 3.5 V<sub>RMS</sub>, or 5 V<sub>RMS</sub>, supporting the typical transformation ratio of 0.5
- Rotational speed adjustable to 7500 rpm, 30,000 rpm, 60,000 rpm, or 150,000 rpm. By decreasing the resolution of the speed measurement from 16 bit to 14 bit, 12 bit or 10 bit, the speed range increases.
- Angular offset of the sensor to adjust its position
- Reversing the orientation to adjust the rotational direction with the direction required by the controller

#### Resolver status information

The resolver interface can detect errors for the connected resolver hardware. The resolver status information can contain the following errors.



Error	Meaning
Configuration parity error	The register containing the configuration data of the resolver is corrupted.
Phase lock	The difference between the phase of the excitation frequency and the phase of the sine and cosine signals exceeds the phase lock range, for example, caused by a <i>Velocity too high</i> error.
Velocity too high	The measured rotational speed exceeds the specified maximum speed. Specify a higher speed range.
Loss of tracking	The resolver interface cannot evaluate the position. The difference between the measured position and the internally calculated reference position of the resolver selftest exceeds the tolerance value. The tolerance value depends on the resolution of the speed range.  <div style="background-color: #f0f0f0; padding: 5px;"> 10 bit --&gt; 12.5° max. tolerance  12 bit --&gt; 5.0° max. tolerance  14 bit --&gt; 2.5° max. tolerance  16 bit --&gt; 2.5° max. tolerance </div>
Degradation of signal mismatch <sup>1)</sup>	The resolver selftest has detected that the amplitudes of the sine and cosine signals differ too much.
Degradation of signal overrange <sup>1)</sup>	The sine and cosine input signals exceed the tolerance values for the specified voltage level.
Inputs loss of signal	The captured values for the sine and cosine input signals are below the lower limit of the specified voltage level, for example, caused by an interrupted or disconnected signal.
Inputs clipped <sup>1)</sup>	The input signal of the sine or cosine signal exceeds the upper limit of the resolver interface. Before this error occurs, the protection circuit of the resolver interface will be activated.

<sup>1)</sup> Not supported by RTI.

**RTI/RTLib support**

You can access the resolver interface via RTI and RTLib.

For details, see:

- RTI:
  - [EMC\\_RESOLVER\\_BLx](#) (RTI Electric Motor Control Blockset Reference )
- RTLib:
  - [Resolver Interface](#) (MicroLabBox RTLib Reference )



**I/O mapping**


A resolver interface is connected to one resolver connector on the front or top panel. The signals of resolver interface 1 belong to the Resolver 1 connector, the signals of resolver interface 2 belong to the Resolver 2 connector.

Each interface provides six signals:

- 2 differential analog output signals for EXC and  $\overline{\text{EXC}}$
- 4 differential analog input signals for SIN,  $\overline{\text{SIN}}$ , COS and  $\overline{\text{COS}}$

For a detailed connector pinout, refer to:

- [Resolver Connectors \(Sub-D\)](#) (MicroLabBox Hardware Installation and Configuration )
- [Resolver Connectors \(Spring-Cage\)](#) (MicroLabBox Hardware Installation and Configuration )

For a detailed information on the channel characteristics, refer to [Resolver Interfaces](#) (MicroLabBox Hardware Installation and Configuration )

## EnDat Interface

**Introduction**

The EnDat interface provides access to absolute encoders that determine the mechanical motor position.

**Basics on absolute encoders**

There is a great variety of absolute encoders that support the EnDat protocol. The encoders use a serial transmission protocol. Absolute encoders connected to an EnDat interface handle the two encoder signals *Clock* and *Data*.

The data transmission is driven by the clock output signal. The data signal is used to get the measured data from the connected encoder but also to write commands and data to the encoder.

The EnDat protocol provides diagnostic capabilities, such as a CRC error check. This lets you evaluate the validity of the transmitted values.

The EnDat interface supports single-turn encoders and multi-turn encoders. A single-turn encoder provides the absolute position for one revolution, a multi-turn encoder also provides the number of the current revolution.

## Characteristics

The EnDat interface is able to allocate digital channels of DIO Class 2 type. Up to two EnDat interface units can be instantiated.

The two channels of an EnDat interface unit are automatically allocated in subsequent order.

The configuration settings of an EnDat interface are:

- Type of the connected encoder: Single-turn encoder or multi-turn encoder
- Resolution of the measured position value, i.e., the number of bits provided by the connected encoder to represent the position value. A maximum value of 28 bits is supported.
- For single-turn encoders, you can explicitly specify the positions per revolution (also called *number of steps*), if the value differs from the typical value of  $2^{\text{ResolutionPerRevolution}}$ .
- For multi-turn encoders, you must specify the resolution of the revolution counter, i.e., the number of bits provided by the connected encoder to represent the revolution number. The maximum number of revolutions that can be counted is implicitly defined by this. A maximum value of 28 bits is supported.
- Frequency of the clock signal from 100 kHz to 16 MHz.

When you are using a clock frequency higher than 8 MHz, the maximum cable length that you can use decreases.

The default waiting time between two data transmissions (also called recovery time) is in the range 10 ... 30  $\mu\text{s}$ . If you specify a clock frequency of 1 MHz or higher for an EnDat 2.2 sensor, the sensor is automatically configured to the shorter waiting time of 1.25 ... 3.75  $\mu\text{s}$ . Otherwise, the waiting time might take longer than the time for data transmission. This behavior corresponds to the EnDat 2.2 specification.

### Note

The configuration of the shorter waiting time is stored in the sensor's non-volatile memory. After its activation, EnDat 2.2 commands cannot be used with a clock frequency less than 1 MHz.

If you want to use the sensor with another hardware system with a clock frequency less than 1 MHz, you have to clear the shorter waiting time configuration. To do so, specify a clock frequency less than 1 MHz and start the real-time application with the connected sensor once.

- Measurement can be adapted to a reverse direction.
- The measured position value can be modified by a position offset.
- Data transmission can be performed continuously or triggered by an event connected to a trigger line.
- Evaluation of diagnostic information, such as CRC checksum error, configuration status, transmission status, and the error bit F1.





You can specify the number of CRC errors to ignore before a CRC checksum error becomes effective. When you use RTI, this error limit is set to 1 and cannot be modified. This means there have to be two successive CRC errors to trigger a CRC checksum error in the transmission error information. When you use RTLib, the error limit can be specified in the range 0 ... 255.

If the measured data is qualified as invalid, the last valid values are kept.

### RTI/RTLib support

You can access an absolute encoder connected to the EnDat interface via RTI and RTLib.



For details, refer to:

- RTI:
  - [EMC\\_ENDAT\\_BLx](#) (RTI Electric Motor Control Blockset Reference )
- RTLib:
  - [EnDat Interface](#) (MicroLabBox RTLib Reference )

### I/O mapping

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O B Connector \(Sub-D\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO2 ch 1 ... DIO2 ch 12
- [Digital I/O Class 2 Connectors \(Spring-Cage\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO2 ch 1 ... DIO2 ch 12

For detailed information on the channel characteristics, refer to

- [Digital Class 2 I/O \(Bidirectional\)](#) (MicroLabBox Hardware Installation and Configuration )

## SSI Interface

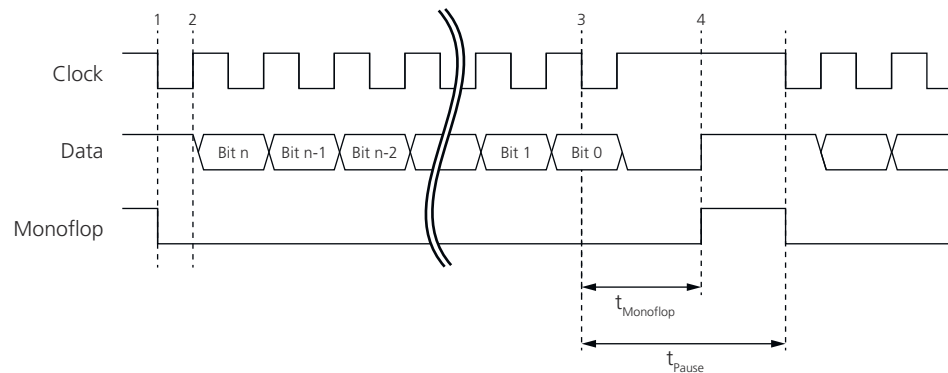
### Introduction

The synchronous serial interface (SSI) provides access to absolute encoders that determine the mechanical motor position.

### Basics on absolute encoders supporting the SSI protocol

There is a great variety of absolute encoders that support the SSI protocol. The encoders use a serial transmission protocol. Absolute encoders connected to an SSI interface handle the two encoder signals *Clock* and *Data*.

The data transmission is driven by the clock output signal. The data signal is used to get the measured data from the connected encoder. The transmission is controlled via a monoflop. The first falling edge of the clock signal triggers the monoflop (position 1 in the diagram). With the first rising edge of the clock signal, the transmission starts with the most significant bit (position 2). When all bits are transmitted (position 3), the clock signal remains on its high level for the specified pause duration. If the monoflop is no longer retriggered by falling edges of the clock signal, it falls back after a monoflop-specific waiting time (position 4). As long as the monoflop is inactive, the register of the sensor is updated with the measured position.



Some SSI encoders provide additional bits, for example, a parity bit or CRC error bits for diagnostic purposes. The position of these bits within the bitstream and the evaluation is not standardized. The transmission of the position data might also differ between various manufacturers.

#### Note

It is strongly recommended to read the encoder's user documentation to get the correct configuration settings for the SSI interface.

The SSI interface supports single-turn encoders and multi-turn encoders. A single-turn encoder provides the absolute position for one revolution, a multi-turn encoder also provides the number of the current revolution.

### Characteristics

The SSI interface is able to allocate digital channels of DIO Class 2 type. Up to two SSI interface units can be instantiated.

The two channels of an SSI interface unit are automatically allocated in subsequent order.

The configuration settings of an SSI interface are:

- Type of the connected encoder: Single-turn encoder or multi-turn encoder
- Resolution of the measured position value, i.e., the number of bits provided by the connected encoder to represent the position value. A maximum value of 28 bits is supported.

- For single-turn encoders, you can specify the positions per revolution (also called *number of steps*), if the value differs from the typical value of  $2^{\text{ResolutionPerRevolution}}$ .
- For multi-turn encoders, you can specify the resolution of the revolution counter, i.e., the number of bits provided by the connected encoder to represent the revolution number. The maximum number of revolutions that can be counted is implicitly defined by this. A maximum value of 28 bits is supported.
- Index of the first bit in the bitstream providing the position data.
- Frequency of the clock signal from 100 kHz to 2 MHz.  
The maximum frequency that you can use depends on the encoder and the required cable length. For details, refer to the encoder's user documentation.
- The pause time between two transmissions.
- Coding scheme of the connected encoder: Gray-coded or binary-coded
- Measurement can be adapted to a reverse direction.
- The measured position value can be modified by a position offset.
- Data transmission can be performed continuously or triggered by an event connected to a trigger line.
- Evaluation of connection errors.
- Evaluation of transmission errors to check the validity of the measured values, if the connected encoder supports repeated reading.

## Handling errors

The SSI interface analyzes the *Data* signal, which can be used to detect connection errors and transmission errors. If one of these errors become effective, the position is set to invalid and an error flag is set.

### ▪ Connection error

The detection of a connection error is always active. You cannot configure it. A connection error might occur due to the following reasons:

- There is no physical connection to the encoder.
- The connected encoder does not respond within the timeout limit. The timeout limit has been reached if there is no rising edge of the Data signal after the specified pause time of the Clock signal.  
This can happen if the specified pause time is too short according to the encoder's specification.

### ▪ Transmission error

The detection of a transmission error is a switchable feature of the SSI interface. By reading the position data twice, data consistency is verified.

#### Note

- Using the data verification feature reduces the effective bandwidth of the transmission.
- If there are problems with this feature, refer to the encoder's user documentation for information whether the encoder supports repeated reading.

You can configure how many successive transmission errors to ignore before a transmission error becomes effective, i.e., before the measurement state is set to invalid.



Independently of the number of reported transmission errors, the total number of transmission errors since application download is logged.

---

### RTI/RTLib support

You can access an absolute encoder connected to the SSI interface via RTI and RTLib.

For details, refer to:



- RTI:
  - [EMC\\_SSI\\_BLx](#) (RTI Electric Motor Control Blockset Reference )
- RTLib:
  - [SSI Interface](#) (MicroLabBox RTLib Reference )

---

### I/O mapping

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O B Connector \(Sub-D\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO2 ch 1 ... DIO2 ch 12
- [Digital I/O Class 2 Connectors \(Spring-Cage\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO2 ch 1 ... DIO2 ch 12

For detailed information on the channel characteristics, refer to:

- [Digital Class 2 I/O \(Bidirectional\)](#) (MicroLabBox Hardware Installation and Configuration )

## Block-Commutated PWM Signal Generation (DIO Class 1)

---

### Introduction

The block-commutated PWM signal generation feature (BCPWM) is based on the DIO Class 1 unit providing access to the single-ended digital I/O channels. It is intended to be used to generate the output signals for any kind of electric motors, especially for brushless DC (BLDC) motors.

---

### Characteristics

This I/O feature is used to generate angle-dependent signal patterns on multiple digital output channels for controlling electric motors. You can specify up to two BCPWM units for signal generation.

- Up to 12 channels can be configured as outputs for each BCPWM unit.

Each BCPWM unit requires at least two channels using RTLib. Using RTI always six channels are allocated for a BCPWM unit.

- The unit can get the current electrical motor position from an angle computation unit or directly from an input available in your real-time application.
- Adjustable configuration of the motor by specifying the number of sectors in the range 2 ... 12 and the position of the sectors. Using RTI, the number of sectors is always set to 6.
- Individual signal patterns for each specified sector, refer to [Supported signal patterns](#) on page 109.
- Adjustable voltage level for the outputs: 2.5 V, 3.3 V, 5.0 V.
- Adjustable signal pattern for a stationary sector.
- Generation of interrupts and trigger signals as events started by the specified event type.
- Configuration of the event behavior via downsampling and delay, refer to [Interrupts and trigger lines provided by the block-commutated PWM signal generation](#) on page 111.
- Adjustable update mode, refer to [Update mode](#) on page 110.
- Adjustable delay between falling and rising edges of a signal pair (dead time) in the range 0 ... 655  $\mu$ s, refer to [Specifying dead time](#) on page 110.
- Adjustable duty cycle.
- Setting to enable the high-impedance state of all the relevant outputs.

In addition to the angle-dependent signal generation, you can specify position-independent signal generation with the following characteristics:

- Up to six specific pattern, which can be separately triggered by an input port.
- Individual signal patterns for each specified specific pattern, refer to [Supported signal patterns](#) on page 109.

---

#### PWM period range

The PWM period can be specified in the range 100 ns ... 1.34 s.  
The value can be specified in steps of 10 ns.

---

#### Supported signal patterns

While the output signals of the stationary sector are to be configured as constantly high or low, the output signals of the other sectors can be separately configured with any kind of signal pattern.

The following signal patterns are supported:

- Low level  
Constant low signal level in the related sector
- High level  
Constant high signal level in the related sector
- PWM  
PWM signal specified by period and duty cycle with a non-inverted output in the related sector

- Inverted PWM

PWM signal specified by period and duty cycle with an inverted output in the related sector

- Complementary PWM

PWM signal specified by period and (1 - duty cycle) with a non-inverted output in the related sector

- Complementary inverted PWM

PWM signal specified by period and (1 - duty cycle) with an inverted output in the related sector

## Update mode

During run time, you can specify new values for the period and the duty cycle. With the update mode, you can specify the timing behavior of the update.

### Update at the middle of the high pulses for block-commutated PWM signal generation

New values for the period and/or the duty cycle are updated at the middle of the high pulses of the PWM output signals.

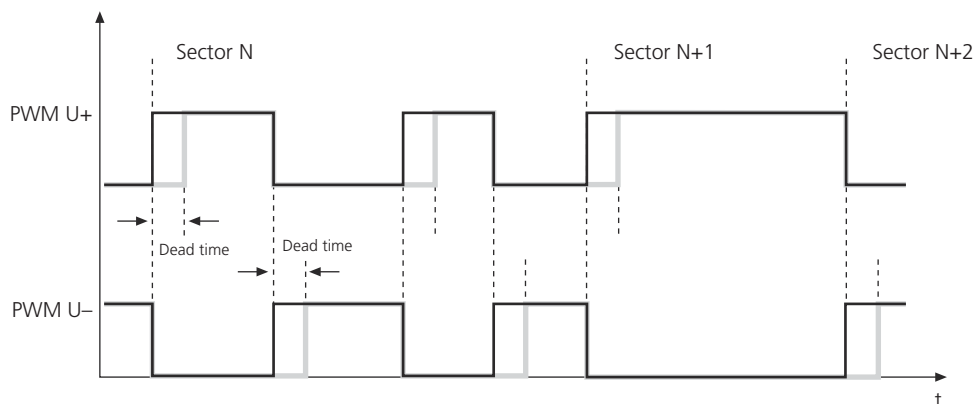
### Update at the start of a period for block-commutated PWM signal generation

New values for the period and/or the duty cycle are updated at the start of a period (middle of the low pulses) of the PWM output signals.

You can combine the two full-cycle update modes. Then, new values are updated at the start of a period *and* in the middle of the high pulses, whichever occurs first. In this half-cycle update mode, changes take effect faster.

## Specifying dead time

With the Dead time option of the EMC\_BC\_PWM\_BLx block or the `RisingEdgeDelay` parameter of the `DioC11BcPwmOut_setRisingEdgeDelay` RTLib function, you can define the timing relationship between the output signals and their corresponding inverted output signals. The specified dead time is valid for all the channels of the current block-commutated PWM group. Dead times are used, for example, to avoid ripple currents or prevent shoot-through currents on the phase drivers. The high times of the inverted and non-inverted output signals are reduced according to the delay of their rising edges. The following illustration shows an example of a resulting signal shape.



The dead time can be specified in the range 0 ... 655  $\mu$ s. If the high time of the inverted or non-inverted signal is less than the specified dead time, the signal remains in low state.

### Interrupts and trigger lines provided by the block-commutated PWM signal generation

The hardware interrupts from the block-commutated PWM signal generation feature are used to trigger interrupt-driven tasks which are executed on MicroLabBox. Trigger lines are used to transmit a trigger signal to an internal or external consumer of the trigger. You can configure to generate either interrupts or trigger signals, or both.

**Event types** You can specify the following event types which has to start the generation of an interrupt or trigger signal. The available event types depend on the specified alignment mode.

- Beginning of the period
- Middle of the period
- Beginning and middle of the period

**Handling interrupts** The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide\)](#). The interrupt which is to trigger the subsystem must be made available using the DIO\_CLASS1\_HWINT\_BLx block. The appropriate function identifier, port number, channel number and the event type in the block's dialog have to be selected. The model must contain an [EMC\\_BC\\_PWM\\_BLx \(RTI Electric Motor Control Blockset Reference\)](#) block with the same port and channel specified for interrupt generation and the same event type.

#### Note

You need a separate DIO\_CLASS1\_HWINT\_BLx block for each interrupt that you want to use on a digital I/O channel.

**Handling trigger lines** The functions you want to trigger by trigger line must be configured to listen to the specified trigger line.

The following I/O functions can be triggered by trigger line:



- ADC Class 1
- Pulse Signal Generation (DIO Class 1)
  - With the generated pulse signal, you can trigger any external device.
- EnDat Interface
- SSI Interface

For further information, refer to [Interrupt Handling](#) on page 29 and [Trigger Signals](#) on page 31.

### RTI/RTLib support

You can access the block-commutated PWM signal generation feature of the DIO Class 1 unit via RTI and RTLib.




For details, see:


- RTI:
  - [EMC\\_BC\\_PWM\\_BLx](#) (RTI Electric Motor Control Blockset Reference )
- RTLib:
  - [Block-Commutated PWM Generation](#) (MicroLabBox RTLib Reference )

## I/O mapping

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.

For a detailed connector pinout, refer to:

- [Digital I/O A Connector \(Sub-D\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO1 ch 1 ... DIO1 ch 32
- [Digital I/O B Connector \(Sub-D\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO1 ch 33 ... DIO1 ch 48
- [Digital I/O Class 1 Connectors \(Spring-Cage\)](#) (MicroLabBox Hardware Installation and Configuration )  
DIO1 ch 1 ... DIO1 ch 48

For detailed information on the channel characteristics, refer to [Digital Class 1 I/O \(Bidirectional\)](#) (MicroLabBox Hardware Installation and Configuration )

## Related topics

### Basics

<a href="#">Multichannel PWM Signal Generation (DIO Class 1)</a> .....	112
<a href="#">PWM Signal Generation (DIO Class 1)</a> .....	58
<a href="#">Tasks Driven by Interrupt Blocks (RTI and RTI-MP Implementation Guide )</a>	

# Multichannel PWM Signal Generation (DIO Class 1)

## Introduction

The multichannel PWM signal generation feature is based on the DIO Class 1 unit providing access to the single-ended digital I/O channels. It is intended to be used to generate the output signals for electric motor control.

## Characteristics

The multichannel PWM signal generation feature can be flexibly configured for any number of channels within the limits of available output channels. You can use it to generate PWM signals with up to 16 phases according to your needs.



- 16 digital channels can be configured as output for multichannel PWM signal generation.
- Adjustable duty cycle.
- Adjustable voltage level for the output: 2.5 V, 3.3 V, 5.0 V.
- Adjustable PWM period in the range 100 ns ... 1.34 s.
- Adjustable alignment mode, refer to [Alignment mode](#) on page 113.
- Generation of interrupts and trigger signals as events started by the specified event type.
- Configuration of the event behavior via downsampling and delay, refer to [Interrupts and trigger lines provided by the multichannel PWM signal generation feature](#) on page 119.
- Adjustable update mode, refer to [Update mode](#) on page 115.
- Possibility to invert the output signals, refer to [Generation of inverted signals](#) on page 117.
- Adjustable delay between falling and rising edges of a channel pair (dead time) in the range 0 ... 655  $\mu$ s, refer to [Specifying dead time](#) on page 118.
- Setting to enable the high-impedance state of all the relevant outputs.

**PWM period range**

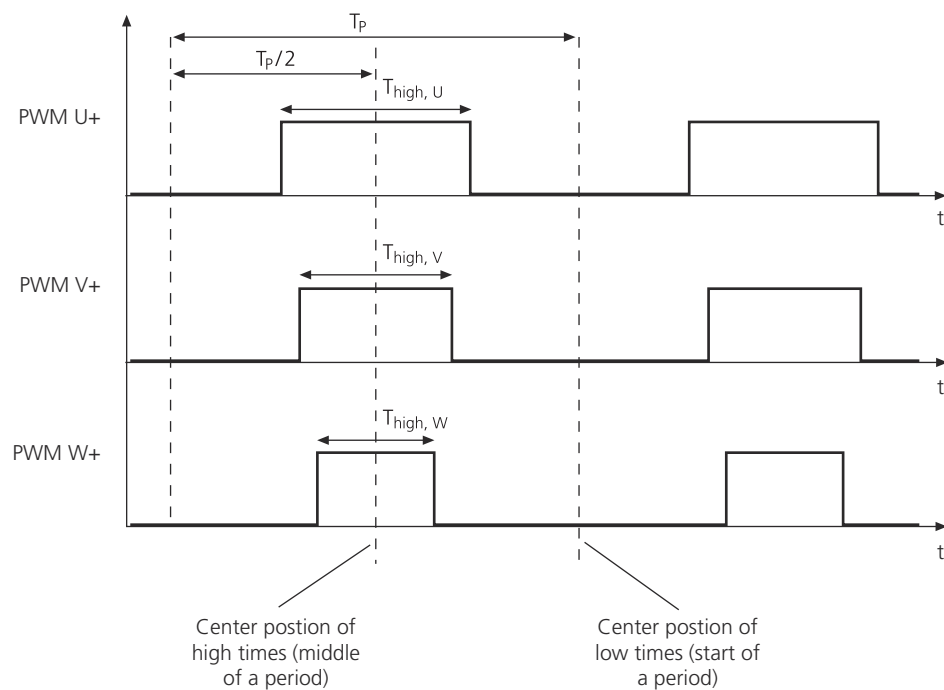
The PWM period can be specified in the range 100 ns ... 1.34 s.

The value can be specified in steps of 10 ns.

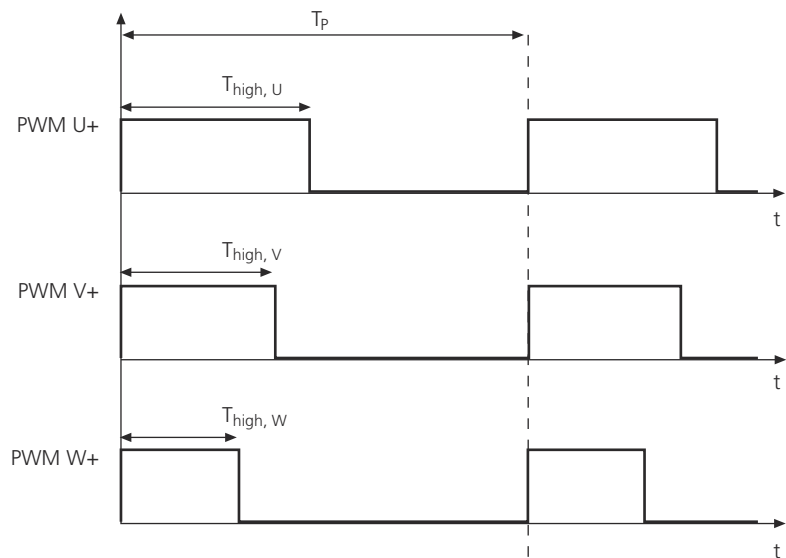
**Alignment mode**

For multi-channel PWM signal generation, you can configure how the signals are aligned to each other. You can choose between center-aligned signal generation and edge-aligned signal generation.

**Signal characteristics of a center-aligned PWM signal** The following example shows a center-aligned 3-phase PWM signal. The signals are aligned to the middle of their periods, which is also the middle of their high pulses.



**Signal characteristics of an edge-aligned PWM signal** The following example shows an edge-aligned 3-phase PWM signal. The signals are aligned to their rising edges.



**Settings that depend on the alignment mode** Depending on the specified alignment mode, not all the configurations are available for some settings. The following table gives you an overview.

Alignment	Update Mode
Edge-aligned	<ul style="list-style-type: none"> <li>At the start of a period</li> </ul>
Center-aligned	<ul style="list-style-type: none"> <li>At the start of a period and/or</li> <li>at the middle of a period</li> </ul>

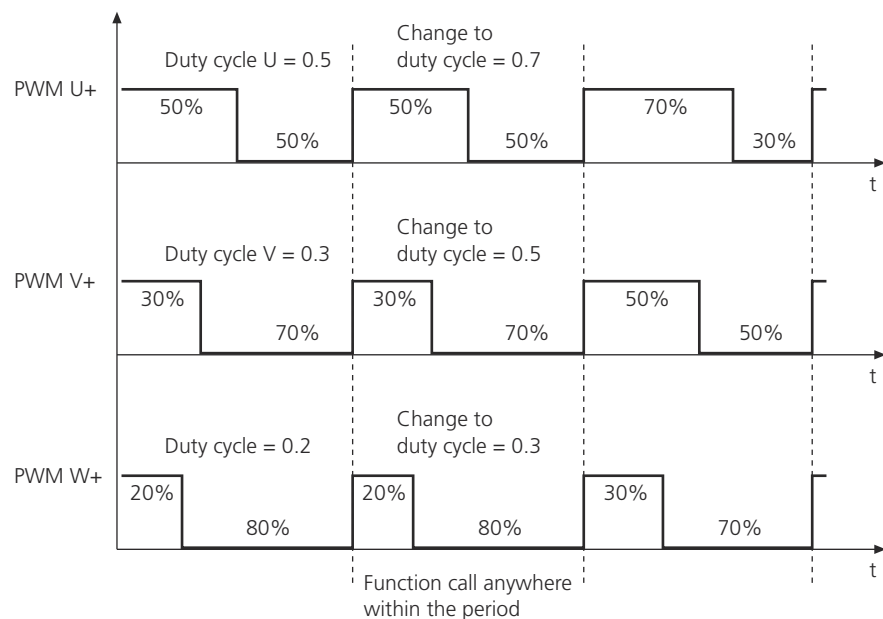
### Update mode

During run time, you can specify new values for the period and the duty cycles. With the update mode, you can specify the timing behavior of the update.

The duty cycles have to be specified for the non-inverted channels. If you have enabled the generation of inverted signals, their duty cycles are automatically adjusted.

**Synchronous update for edge-aligned multi-channel PWM signal generation** New values for the period and/or the duty cycles are updated at the next rising edge of the PWM output signal.

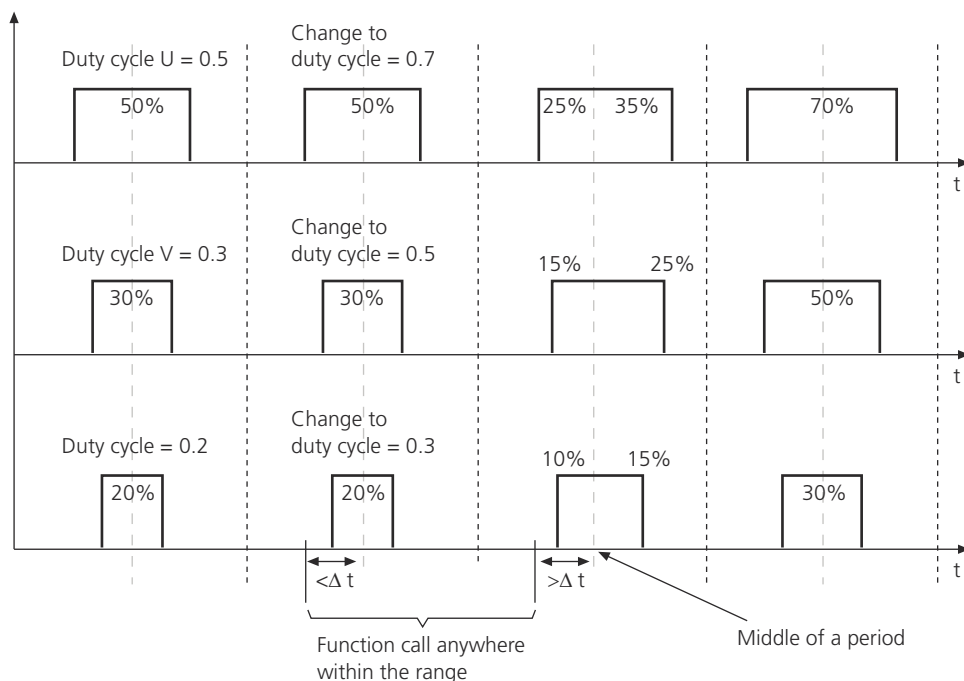
The following figure shows an example of how the duty cycles of a 3-phase PWM are updated in synchronous update mode.



**Update at the middle of the high pulses for center-aligned multi-channel PWM signal generation** New values for the period and/or the duty cycles are updated at the middle of the high pulses of the PWM output signals.

The following figure shows an example of how the duty cycles of a 3-phase PWM are updated in the *CENTER* update mode.

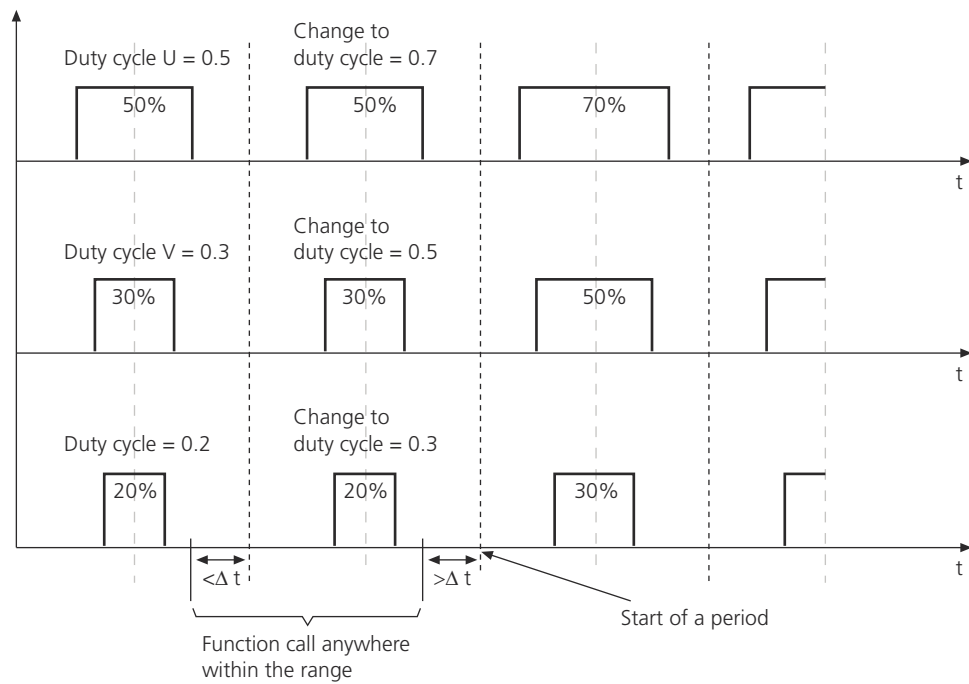
The update function must be called before the middle of the following high pulses ( $\Delta t$ ) so the new values are considered in the second half of the current high pulses.



**Update at the start of a period for center-aligned multi-channel PWM signal generation** New values for the period and/or the duty cycles are updated at the start of a period (middle of the low pulses) of the PWM output signals.

The following figure shows an example of how the duty cycles of a 3-phase PWM are updated in the *START* update mode.

The update function must be called before the middle of the low pulses ( $\Delta t$ ; start of the period) so the new values are considered in the following period.

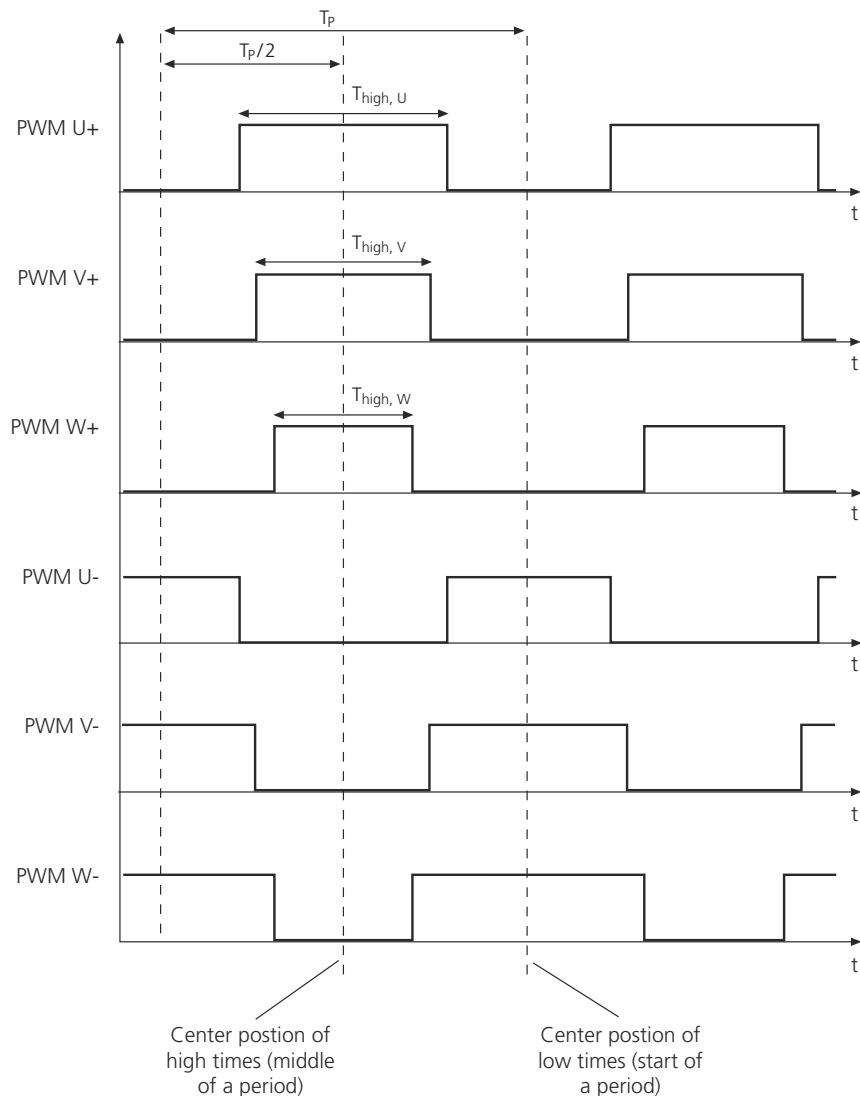


In center-aligned mode, you can combine the two full-cycle update modes. Then, new values are updated at the start of a period *and* in the middle of the high pulses, whichever occurs first. In this half-cycle update mode, changes take effect faster.

#### Generation of inverted signals

You can specify the generation of inverted signals for center-aligned PWM signal generation. The number of required channels is then automatically doubled. The number of required channels must not exceed the number of available channels.

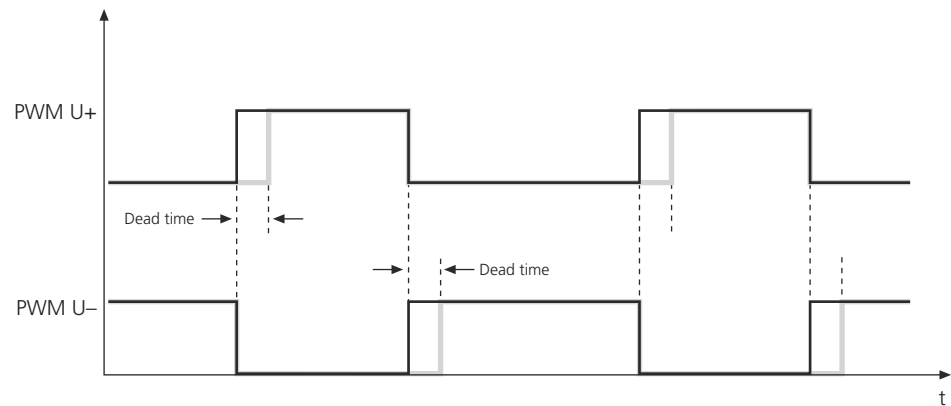
**Signal characteristics of a center-aligned PWM signal with inverted signals** The following example shows a center-aligned 3-phase PWM signal generation with inverted signals.



### Specifying dead time

With the Dead time option of the EMC\_MC\_PWM\_BLx block or the **RisingEdgeDelay** parameter of the **DioC11MultiPwmOut\_setRisingEdgeDelay** RTLib function, you can define the timing relationship between the PWM output signals and their corresponding inverted output signals for center-aligned PWM signal generation. The specified dead time is valid for all the channels of the current multi-channel PWM group. Dead times are used, for example, to avoid ripple currents or prevent shoot-

through currents on the phase drivers. The high times of the inverted and non-inverted output signals are reduced according to the delay of their rising edges. The following illustration shows the resulting signal shape of a center-aligned multi-channel PWM with a specified dead time.



The dead time can be specified in the range 0 ... 655  $\mu$ s. If the high time of the inverted or non-inverted signal is less than the specified dead time, the signal remains in low state.

#### Interrupts and trigger lines provided by the multichannel PWM signal generation feature

The hardware interrupts from the multichannel PWM signal generation feature are used to trigger interrupt-driven tasks which are executed on MicroLabBox. Trigger lines are used to transmit a trigger signal to an internal or external consumer of the trigger. You can configure to generate either interrupts or trigger signals, or both.

**Event types** You can specify the following event types which has to start the generation of an interrupt or trigger signal. The available event types depend on the specified alignment mode.

- Beginning of the period
- Middle of the period
- Beginning and middle of the period

**Handling interrupts** The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide\)](#).

The interrupt which is to trigger the subsystem must be made available using the DIO\_CLASS1\_HWINT\_BLx block. The appropriate function identifier, port number, channel number and the event type in the block's dialog have to be selected. The model must contain an [EMC\\_MC\\_PWM\\_BLx \(RTI Electric Motor Control Blockset Reference\)](#) block with the same port and channel specified for interrupt generation and the same event type.

**Note**

You need a separate `DIO_CLASS1_HWINT_BLx` block for each interrupt that you want to use on a digital I/O channel.

**Handling trigger lines** The functions you want to trigger by trigger line must be configured to listen to the specified trigger line.

The following I/O functions can be triggered by trigger line:



- ADC Class 1
- Pulse Signal Generation (DIO Class 1)  
With the generated pulse signal, you can trigger any external device.
- EnDat Interface
- SSI Interface

For further information, refer to [Interrupt Handling](#) on page 29 and [Trigger Signals](#) on page 31.

**RTI/RTLib support**

You can access the multichannel PWM signal generation feature of the DIO Class 1 unit via RTI and RTLib.

For details, see:

- RTI:
  - [EMC\\_MC\\_PWM\\_BLx](#) (RTI Electric Motor Control Blockset Reference )
- RTLib:
  - [Multichannel PWM Signal Generation](#) (MicroLabBox RTLib Reference )

**I/O mapping**

The order of the channels is as follows, if you have specified to also generate the inverted signals.

Signal
Non-inverted signal 1
...
Non-inverted signal n
Inverted signal 1
...
Inverted signal n

MicroLabBox has a static mapping between I/O signals and I/O pins. When you use RTI, the block dialog displays the related signal name. The signal name is also printed on the housing of MicroLabBox. You have to consider only the connector panel type.



For a detailed connector pinout, refer to:


- [Digital I/O A Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(c8dce68b26731c7aa5915072fc9d68dd\_img.jpg\)](#))
- DIO1 ch 1 ... DIO1 ch 32
- [Digital I/O B Connector \(Sub-D\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(76b3245de86167eba9fcdc9cc9f32aa4\_img.jpg\)](#))
- DIO1 ch 33 ... DIO1 ch 48
- [Digital I/O Class 1 Connectors \(Spring-Cage\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(13db7587f50867332e5bedc6a161739d\_img.jpg\)](#))
- DIO1 ch 1 ... DIO1 ch 48

For detailed information on the channel characteristics, refer to

- [Digital Class 1 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(f15d3c54be60b4fd0ce1da9fb3f67256\_img.jpg\)](#))
- [Digital Class 2 I/O \(Bidirectional\) \(MicroLabBox Hardware Installation and Configuration !\[\]\(7bf135d42c40a6430c927b2fd03d7659\_img.jpg\)](#))

Related topics

Basics

<a href="#">Block-Commutated PWM Signal Generation (DIO Class 1).....</a>	<a href="#">108</a>
<a href="#">PWM Signal Generation (DIO Class 1).....</a>	<a href="#">58</a>
<a href="#">Tasks Driven by Interrupt Blocks (RTI and RTI-MP Implementation Guide </a> )	



# CAN Support

---

## Introduction

The following topics provide all the information required for working with dSPACE CAN boards.

---

## Where to go from here

### Information in this section

- [Setting Up a CAN Controller.....](#) 124  
Explains how to set up a CAN controller to use a dSPACE board with CAN bus interface.
- [Using the RTI CAN MultiMessage Blockset.....](#) 138  
Provides information on using the RTI CAN MultiMessage Blockset.
- [CAN Signal Mapping.....](#) 161  
Lists the CAN signals of the dSPACE real-time hardware and the mapping of these signals to RTI blocks and RTLib functions.

### Information in other sections

- [Limited Number of CAN Messages.....](#) 167  
When you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions, the number of CAN messages in an application is limited.

# Setting Up a CAN Controller

## Introduction

To use a dSPACE board with CAN bus interface, you have to set up the CAN controller.

## Where to go from here

## Information in this section

### [Initializing the CAN Controller..... 124](#)

The CAN controller performs serial communication according to the CAN protocol. You must configure the CAN controller according to the application.

### [CAN Transceiver Types..... 126](#)

The way in which CAN messages are transmitted on a CAN bus depends on the CAN transceiver used.

### [Defining CAN Messages..... 130](#)

The dSPACE CAN software lets you easily define CAN messages to be transmitted or received.

### [Implementing a CAN Interrupt..... 132](#)

The CAN controller is responsible for generating interrupts to the master processor. You can specify the events on which these interrupts are generated.

### [Using RX Service Support..... 132](#)

RTI CAN Blockset provides two concepts for receiving CAN messages.

### [Removing a CAN Controller \(Go Bus Off\)..... 134](#)

You can remove the CAN controller that is being used from the bus when you use several CAN controllers.

### [Getting CAN Status Information..... 135](#)

You can use the Error Management Logic (EML) of a CAN controller to get error and status information on the CAN bus and the controller.

### [CAN Partial Networking..... 136](#)

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

## Initializing the CAN Controller

## Introduction

The CAN controller performs serial communication according to the CAN protocol. You can take control of or communicate with other members of a CAN

bus via the controller. This means you must configure the CAN controller — called the CAN channel — according to the application.

### Standard configuration

You must specify the baud rate for the CAN application and the sample mode:

Sample Mode	Description
1-sample mode	(supported by all dSPACE CAN boards) The controller samples a bit once to determine if it is dominant or recessive.
3-sample mode	(supported by the DS4302 only) The controller samples a bit three times and uses the majority to determine if it is dominant or recessive.

The required bit timing parameters are automatically calculated by the dSPACE CAN software.

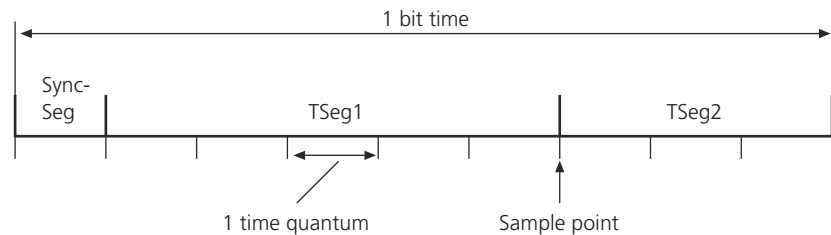
### Advanced configuration (bit timing parameters)

The bits of a CAN message are transmitted in consecutive bit times. According to the CAN specification, a bit time consists of two programmable time segments and a synchronization segment:

**TSeg1** Timing segment 1. The time before the sample point.

**TSeg2** Timing segment 2. The time after the sample point.

**SyncSeg** Used to synchronize the various bus members (nodes).



The following parameters are also part of the advanced configuration:

**SP** Sample point. Defines the point in time at which the bus voltage level (CAN-H, CAN-L) is read and interpreted as a bit value.

**SJW** Synchronization jump width. Defines how far the CAN controller can shift the location of the sample point to synchronize itself to the other bus members.

**BRP** Baud rate prescaler value. The BRP defines the length of one time quantum.

**SMPL** Sample mode. Either 1-sample or 3-sample mode. Applicable to the DS4302 only.

Except for the SyncSeg parameter, you must define all these parameters via the values of the bit timing registers (BTR0, BTR1), located on the CAN controller.

**Note**

Setting up bit timing parameters requires advanced knowledge of the CAN controller hardware and the CAN bus hardware.

**RTI support**

You initialize a CAN controller with the RTICAN CONTROLLER SETUP block.

Refer to [RTICAN CONTROLLER SETUP \(RTI CAN Blockset Reference !\[\]\(950a62bbddad88d64435fd35607dfc42\_img.jpg\)\)](#).

**Related topics****References**

[RTICAN CONTROLLER SETUP \(RTI CAN Blockset Reference !\[\]\(e1d6102fe77919492c04879c8450f1f5\_img.jpg\)\)](#)

## CAN Transceiver Types

**Introduction**

To communicate with other bus members in a CAN bus, each bus member is equipped with a CAN transceiver. The transceiver defines the type of wire used for the bus (coaxial, two-wire line, or fiber-optic cables), the voltage level, and the pulse forms used for 0-bit and 1-bit values. The way in which CAN messages are transmitted on a CAN bus therefore significantly depends on the CAN transceiver used.

**Note**

Make sure that the CAN transceiver type used on the CAN bus matches the type on the dSPACE board you use to connect to the bus.

**Terminating the CAN bus**

Depending on the CAN transceiver type, you must terminate each CAN bus with resistors at both ends of the bus.

**Note**

Failure to terminate the bus will cause bit errors due to reflections. These reflections can be detected with an oscilloscope.

**Supported transceivers**

The following table lists dSPACE hardware and the supported transceivers:

dSPACE Hardware	Transceiver Type
<ul style="list-style-type: none"> <li>DS2202</li> <li>DS2210</li> <li>DS2211</li> </ul>	ISO11898
DS4302	<p>The following transceiver types are supported:</p> <ul style="list-style-type: none"> <li>ISO11898</li> <li>RS485</li> <li>C252</li> <li>Piggyback<sup>1)</sup></li> </ul> <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p><b>Note</b></p> <p>The RTI CAN Blockset does not support transceiver types with different modes, for example single-wire and two-wire operation. Nevertheless, such transceiver types can be applied to the DS4302, but additional user-written S-functions are required to implement the communication between the piggyback module and the CAN controller.</p> </div>
MicroAutoBox II	<p>The following transceiver types are supported:</p> <ul style="list-style-type: none"> <li>ISO11898</li> <li>ISO11898-6<sup>2)</sup>, <sup>3)</sup></li> </ul>
MicroLabBox	<p>The following transceiver types are supported:</p> <ul style="list-style-type: none"> <li>ISO11898</li> <li>ISO11898-6<sup>2)</sup></li> </ul>

<sup>1)</sup> If none of the above transceivers matches your application or if a TJA1041 transceiver is used, "piggyback" must be selected as the transceiver type.

<sup>2)</sup> Selecting the ISO11898-6 transceiver type is required to perform partial networking.

<sup>3)</sup> Supported only by MicroAutoBox II with DS1513 I/O board.

**ISO11898 transceiver**

ISO11898 defines a high-speed CAN bus that supports baud rates of up to 1 MBd. This is the most commonly used transceiver, especially for the engine management electronics in automobiles.

**CAN-H, CAN-L** ISO11898 defines two voltage levels:

Level	Description
CAN-H	High if the bit is dominant (3.5 V), floating (2.5 V) if the bit is recessive.
CAN-L	Low if the bit is dominant (1.0 V), floating (2.5 V) if the bit is recessive.

**Termination** To terminate the CAN bus lines, ISO11898 requires a 120-Ω resistor at both ends of the bus.

**ISO11898-6 transceiver**

High-speed transceiver that supports partial networking.

**Termination** To terminate the CAN bus lines, ISO11898-6 requires a 120-Ω resistor at both ends of the bus.

**Note**

There are some limitations when you use the optional ISO11898-6 transceiver:

- No wake-up interrupt is implemented.
- Partial networking is supported only for the following baud rates:
  - 125 kbit/s
  - 250 kbit/s
  - 500 kbit/s
  - 1000 kbit/s

Other baud rates can be used for normal CAN operation, but detecting wake-up messages for partial networking is supported only for the baud rates listed above.

- You have to enable Automatic Wake Up on the Parameters Page (RTI<xxx>\_ISO11898\_6\_SST) before you build the model. You cannot enable automatic wake-up during run time.
- If the transceiver is in *power on / listen only* mode, the CAN controller does not send an acknowledge message to the transmitter. The transmitter therefore continues to send the message until it receives the acknowledge signal. This might result in a task overrun if an RX interrupt is configured for the CAN controller.
- If the transceiver is in *power on / listen only* mode, it is not able to send CAN messages. Automatic wake-up is not possible if the transceiver is in *power on / listen only* mode. Because no message is sent on the CAN bus by the transceiver in *power on / listen only* mode, CAN arbitration fails. The CAN controller changes to the BUS OFF state. It is not possible to set the BUS state automatically to BUS ON via an interrupt, because the reason for the BUS OFF state still remains. You must set the CAN controller to BUS ON after you have switched the transceiver state to normal, standby, or sleep mode.

**RS485 transceiver**

The RS485 transceiver supports baud rates of up to 500 kD. It is often used in the automotive industry. A CAN bus using this transceiver can connect up to 25 CAN nodes.

**Termination** To terminate the CAN bus lines, a 120-Ω resistor must be used at both ends of the CAN bus.

**C252 fault-tolerant transceiver**


The C252 fault-tolerant transceiver supports baud rates of up to 125 kD. Its main feature is on-chip error management, which allows the CAN bus to continue operating even if errors such as short circuits between the bus lines occur.



When this transceiver is used, the CAN bus can interconnect nodes that are widely distributed. You can switch the C252 transceiver between sleep and normal (awake) mode.

**Termination** There are two ways to terminate the CAN bus lines: Use a 10 k $\Omega$  resistor for many connected bus members, or a 1.6 k $\Omega$  resistor if the number of bus members is equal to or less than five. The termination resistors are located between CAN-L and RTL and CAN-H and RTH (refer also to the "PCA82C252 Fault-tolerant Transceiver Data Sheet" issued by Philips Semiconductors).

**Note**

The TJA1054 transceiver is pin and downward compatible with the C252 transceiver. If the TJA1054 transceiver is on board the DS4302 and you want to use the fault-tolerant transceiver functionality, select "C252" in the RTI CAN CONTROLLER SETUP block. Refer to [Unit Page \(RTICAN CONTROLLER SETUP\)](#) (RTI CAN Blockset Reference .

---

**Custom transceivers**

The DS4302 allows you to mount up to four customization modules to use transceivers that are not on the DS4302.

**Connecting customization modules** For instructions on connecting customization modules, refer to [Customization Modules \(PHS Bus System Hardware Reference !\[\]\(003082e50e3009141f59bd5df831749f\_img.jpg\)](#).

**Optional TJA1041 transceiver** dSPACE provides the optional TJA1041 that you can use as a custom transceiver for the DS4302. For a detailed description of the transceiver and the available transceiver modes, refer to the data sheet of the TJA1041 transceiver.

For details on the RTI support for the TJA1041 transceiver, refer to [TJA1041 Support Blocks \(RTI CAN Blockset Reference !\[\]\(529949c2c3dadbaa4e538e8c643454bc\_img.jpg\)\)](#).

#### Note

There are some limitations when you use the optional TJA1041 transceiver:

- No wake-up interrupt is implemented.
- You have to enable Automatic Wake Up in the [DS4302\\_TJA1041\\_SST \(RTI CAN Blockset Reference !\[\]\(cf5be311f7b2821912d8009884508fa2\_img.jpg\)\)](#) block before you build the model. You cannot enable automatic wake-up during run time.
- If the transceiver is in *power on / listen only* mode, the CAN controller does not send an acknowledge message to the transmitter. The transmitter therefore continues to send the message until it receives the acknowledge signal. This might cause a task overrun if an RX interrupt is configured for the CAN controller.
- If the transceiver is in *power on / listen only* mode, it is not able to send CAN messages. Automatic wake-up is not possible if the transceiver is in *power on / listen only* mode. Because no message is sent on the CAN bus by the transceiver in *power on / listen only* mode, CAN arbitration fails. The CAN controller changes to the BUS OFF state. It is not possible to set the BUS state automatically to BUS ON via an interrupt, because the reason for the BUS OFF state still remains. You must set the CAN controller to BUS ON after you have switched the transceiver state to normal, standby, or sleep mode.

## Defining CAN Messages

### Introduction

The dSPACE CAN software lets you easily define CAN messages to be transmitted or received.

### Message types

You can define a message as a TX, RX, RQ, or RM message:

Message Type	Description
Transmit (TX)	This message is transmitted with a specific identifier. A TX message contains up to 8 bytes of data.
Receive (RX)	This message is <i>not</i> transmitted over the bus. An RX message is used only to define how the CAN controller processes a received message. An RX message transfers the incoming data from the CAN controller to the master processor.
Request (RQ)	First part of a <i>remote transmission request</i> <sup>1)</sup> . An RQ message is transmitted with a specific identifier to request data. An RQ message does not contain data.

Message Type	Description
Remote (RM)	Second part of a <i>remote transmission request</i> <sup>1)</sup> . An RM message is a TX message that is sent only if the CAN controller has received a corresponding RQ message. The RM message contains the data requested by the RQ message.

<sup>1)</sup> With RTI CAN Blockset, the remote transmission request is divided into an RQ message and an RM message. The meanings of the words “remote” and “request” used in this document do not correspond to those used in CAN specifications.

### Message configuration

With RTI CAN Blockset, you have to implement one message block for each message. To define a message to be transmitted, for example, you must implement an RTICAN Transmit (TX) block.

**Message configuration by hand** You can perform message configuration by hand. In this case, you must specify the message identifier and identifier format (STD, XTD), the length of the data field, and the signals for each message. You also have to specify the start bit and length of each signal.

**Message configuration from data file (data file support)** You can load a data file containing the configuration of one or more messages. Then you can assign a message defined in the data file to a message block. Refer to [Configuring CAN Messages via Data Files \(RTI CAN Blockset Reference !\[\]\(cbe2492b119e39e02a1dab2af4a4b296\_img.jpg\)](#)).

### Multiple message access





Multiple message access allows you to place several RX or TX blocks with the same identifier and identifier format in one model. You can decode the signals of an RX message in several ways, or place TX blocks in several enabled subsystems to send data in various ways.

### Delay time for message transmission

To distribute messages over time and avoid message bursts, you can specify delay times. A message is sent after the delay time. The delay time is a multiple of the time needed to send a CAN message at a given baud rate and identifier format. You can only enter a factor to increase or decrease the delay time.

### RTI support

With RTI CAN Blockset, you have to implement one message block for each message. Refer to:

Message Type	RTI Block
Transmit (TX)	<a href="#">RTICAN Transmit (TX) (RTI CAN Blockset Reference </a> )
Receive (RX)	<a href="#">RTICAN Receive (RX) (RTI CAN Blockset Reference </a> )
Request (RQ)	<a href="#">RTICAN Request (RQ) (RTI CAN Blockset Reference </a> )
Remote (RM)	<a href="#">RTICAN Remote (RM) (RTI CAN Blockset Reference </a> )

**Related topics****Basics**

[Configuring CAN Messages via Data Files \(RTI CAN Blockset Reference !\[\]\(dfbd6b3763a6d1d9afaa974f64e2e4b5\_img.jpg\)](#))

## Implementing a CAN Interrupt

**Introduction**

The CAN controller transmits and receives messages and handles error management. It is also responsible for generating interrupts to the master processor. You can specify the events on which these interrupts are generated.

A special Bus Failure interrupt and a wake-up interrupt are available for the DS4302.

**RTI support**

You can implement a CAN interrupt with the RTICAN Interrupt block. Refer to [RTICAN Interrupt \(RTI CAN Blockset Reference !\[\]\(aa53ad6fea213b8b2226d3077e30533a\_img.jpg\)](#)).

**Related topics****References**

[RTICAN Interrupt \(RTI CAN Blockset Reference !\[\]\(fe3aebe81acea8d45108cd2768939da7\_img.jpg\)](#))

## Using RX Service Support

**Concepts for receiving CAN messages**

When CAN messages are received, RX blocks access the DPMEM between the master processor and the slave processor.

RTI CAN Blockset provides two concepts for receiving CAN messages:

- Common receive concept
- RX service receive concept

**Common receive concept**

According to the common receive concept, one data object is created in the DPMEM for each received CAN message. Due to the limited DPMEM size, the number of RX blocks you can use in a model is limited to 100 (200 for the DS4302).

**RX service receive concept**


When you enable RX service support, one data object is created in the DPMEM for all received CAN messages, and memory on the master processor is used to

receive CAN messages. The RX service fills this memory with new CAN data. This concept improves run-time performance.

#### Tip

In contrast to the common receive concept, the number of RX blocks for which RX service support is enabled is unlimited.

**Specifying a message filter** When you use RX service, you have to specify a filter to select the messages to receive via RX service. To define the filter, you have to set up a bitmap that represents the message. Each bit position can be assigned 0 (must be matched), 1 (must be matched), or X (don't care). A message is received via RX service only if it matches the bitmap.

You can define the message filter on the [RX Service Page \(RTICAN CONTROLLER SETUP\)](#) (RTI CAN Blockset Reference )

**Specifying the queue size** When you use RX service, you have to specify the maximum number of CAN messages that you expect to receive in a sample step. The memory allocated on the master processor used to queue CAN messages is calculated from the specified maximum number of CAN messages.

#### Note

If more CAN messages than the specified Queue size are received in a sample step, the oldest CAN messages are lost. You should therefore specify the queue size so that no CAN messages are lost.

*Example:*

A CAN controller is configured to use the baud rate 500 kBd. The slowest RX block assigned to this CAN controller is sampled every 10 ms. At the specified baud rate, a maximum of about 46 CAN messages (STD format) might be received during two consecutive sample steps. To ensure that no CAN message is lost, set the queue size to 46.

**Triggering an interrupt when a message is received via RX service** You can let an interrupt be triggered when a message is received via RX service.

#### Note

You cannot let an interrupt be triggered when a message *with a specific ID* is received. An interrupt is triggered each time a message is received via RX service.

You can define the interrupt on the [Unit Page \(RTI CAN Interrupt\)](#) (RTI CAN Blockset Reference )

**Precondition for gatewaying messages** Enabling the RX service is a precondition for *gatewaying messages* between CAN controllers.

Refer to [Gatewaying Messages Between CAN Buses \(RTI CAN Blockset Reference\)](#).

**Precondition for the TX loop back feature** RX service allows you to use the *TX loop back feature*. The feature lets you observe whether message transfer over the bus was successful.

You can enable TX loop back on the [Options Page \(RTICAN Transmit \(TX\)\) \(RTI CAN Blockset Reference\)](#).

#### Enabling RX service support

You have to enable RX service support for each CAN controller and for each RX block.

#### RTI support

- For a CAN controller, you enable the RX service on the RX Service page of the RTICAN CONTROLLER SETUP block. Refer to [RX Service Page \(RTICAN CONTROLLER SETUP\) \(RTI CAN Blockset Reference\)](#).
- For an RX block, you enable the RX service on the Options page of the RTICAN Receive (RX) block of the RTICAN CONTROLLER. Refer to [Options Page \(RTICAN Receive \(RX\)\) \(RTI CAN Blockset Reference\)](#).

#### Related topics

##### Basics

[Gatewaying Messages Between CAN Buses \(RTI CAN Blockset Reference\)](#)

## Removing a CAN Controller (Go Bus Off)

#### Introduction

If you use several CAN controllers, you can remove the one currently in use from the bus. Data transfer from the master to the slave processor is then stopped. You can select the CAN controller you want to remove from the bus via the RTICAN Go Bus Off block.

You can restart data transfer with another CAN controller or the same one with the RTICAN Bus Off Recovery block.

#### RTI support

- To remove a CAN controller from the bus, use the RTICAN Go Bus Off block. Refer to [RTICAN Go Bus Off \(RTI CAN Blockset Reference\)](#).
- To restart data transfer, use the RTICAN Bus Off Recovery block. Refer to [RTICAN Bus Off Recovery \(RTI CAN Blockset Reference\)](#).

**Related topics****References**

[RTICAN Bus Off Recovery \(RTI CAN Blockset Reference !\[\]\(feabb98897b440bc8695a03336a6e2df\_img.jpg\)\)](#)  
[RTICAN Go Bus Off \(RTI CAN Blockset Reference !\[\]\(c7f935293d8062fa748ed86b74d28761\_img.jpg\)\)](#)

## Getting CAN Status Information

**Introduction**

You can use the Error Management Logic (EML) of a CAN controller to get error and status information on the CAN bus and the controller. Errors occur, for example, if a CAN controller fails to transmit a message successfully.

**CAN controller status information**

The controller's EML has two counters: the Receive Error counter and the Transmit Error counter. According to their values, the EML can set the CAN controller to one of the following states:

Counter Value	Error State	Description
Each counter value < 128	Error active	The CAN controller is active. Before turning to the error passive state, the controller sets an error warn (EWRN) bit if one of the counter values is $\geq 96$ .
At least one counter value $\geq 128$	Error passive	The CAN controller is still active. The CAN controller can recover from this state itself.
Transmit Error counter value $\geq 256$	Bus off	The CAN controller disconnects itself from the bus. To recover, an external action is required (bus off recovery).

**CAN bus status information**

You can get the following CAN bus status information:

Number of ...	Description
Stuff bit errors	Each time more than 5 equal bits in a sequence occurred in a part of a received message where this is not allowed, the appropriate counter is incremented.
Form errors	Each time the format of a received message deviates from the fixed format, the appropriate counter is incremented.
Acknowledge errors	Each time a message sent by the CAN controller is not acknowledged, the appropriate counter is incremented.
Bit 0 errors	Each time the CAN controller tries to send a dominant bit level and a recessive bus level is detected instead, the appropriate counter is incremented. During bus off recovery, the counter is incremented each time a sequence of 11 recessive bits is detected. This enables the controller to monitor the bus off recovery sequence, indicating that the bus is not permanently disturbed.
Bit 1 errors	Each time the CAN controller tries to send a recessive bit level and a dominant bus level is detected instead, the appropriate counter is incremented.

Number of ...	Description
Cyclic redundancy check (CRC) errors	Each time the CRC checksum of the received message is incorrect, the appropriate counter is incremented. The EML also checks the CRC checksum of each message (see <a href="#">Message fields (RTI CAN Blockset Reference)</a> )).
Lost RX messages	Each time a message cannot be stored in the buffer of the CAN controller, the message is lost and an <i>RX lost error</i> is detected.
Successfully received RX messages	Each time an RX message is received successfully, the appropriate counter is incremented.
Successfully sent TX messages	Each time a TX message is sent successfully, the appropriate counter is incremented.
(DS4302 only) Status of fault tolerant receiver	The error state of the fault tolerant receiver is reported.
(DS4302 only) Fault tolerant transceiver	The value of the output is increased if a CAN bus events occurs.

**RTI support**

To get status information, use the RTICAN Status block. Refer to [RTICAN Status \(RTI CAN Blockset Reference\)](#).

**Related topics****References**

[CAN Service Functions \(MicroLabBox RTLib Reference\)](#)  
[RTICAN Status \(RTI CAN Blockset Reference\)](#)

## CAN Partial Networking

**Introduction**

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

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

- MicroAutoBox II equipped with the DS1513 I/O board
- MicroLabBox

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

- Filtering message IDs: You can define a message filter to select the messages to use as wake-up messages. The filter uses a bitmask which represents the



message. A message passes the filter and is used as wake-up message only if it matches the bitmask.

- Filtering message data: You can mask the data bytes of incoming wake-up messages to determine whether they are valid wake-up messages.

**Switching the CAN transceiver to sleep mode** The CAN transceiver of the dSPACE real-time hardware is switched to sleep mode via the real-time application.

#### Tip

(MicroAutoBox II only) You can stop and power down the MicroAutoBox II with the DS1401\_POWER\_DOWN block from the DS1401 MicroAutoBox II Base Board II library. To set MicroAutoBox II to sleep mode, KL15 must be disconnected from the power supply. For further information, refer to [DS1401\\_POWER\\_DOWN \(MicroAutoBox II RTI Reference !\[\]\(d66ff64371a51729ac8c1cdaa685ba6f\_img.jpg\)](#)). This is not possible for MicroLabBox.

**Waking up dSPACE real-time hardware** You can use partial networking messages to wake up dSPACE real-time hardware after its CAN transceiver is switched to sleep mode.

- (Relevant for MicroAutoBox II only) If a MicroAutoBox II's CAN transceiver is woken up via a partial networking message, the MicroAutoBox II behaves as if it was powered up manually. Depending on where the real-time application is loaded (flash memory or RAM), the MicroAutoBox II starts the application or waits for further input.
- (Relevant for MicroLabBox only) Unlike MicroAutoBox II, MicroLabBox cannot be powered down and then woken up via partial networking messages. However, the CAN transceiver of MicroLabBox can be set to sleep mode, and then woken up via partial networking messages later on.

**RTI support** Refer to [Partial Networking Page \(RTICAN CONTROLLER SETUP\) \(RTI CAN Blockset Reference !\[\]\(17413706fd4997a1a4bdf85c6864eee1\_img.jpg\)](#)).

## Related topics

## References

[Partial Networking Page \(RTICAN CONTROLLER SETUP\) \(RTI CAN Blockset Reference !\[\]\(d3102649f02e825ddb76dc3de0190154\_img.jpg\)](#))

# Using the RTI CAN MultiMessage Blockset

## Introduction

The RTI CAN MultiMessage Blockset is a Simulink blockset for efficient and dynamic handling of complex CAN setups in hardware-in-the-loop (HIL) applications.

## Where to go from here

## Information in this section

### [Basics on the RTI CAN MultiMessage Blockset..... 138](#)

Gives you an overview of the features of the RTI CAN MultiMessage Blockset.

### [Basics on Working with CAN FD..... 143](#)

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

### [Basics on Working with a J1939-Compliant DBC File..... 148](#)

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

### [Transmitting and Receiving CAN Messages..... 154](#)

Large CAN message bundles can be managed by a single Simulink block provided by the RTI CAN MultiMessage Blockset.

### [Manipulating Signals to be Transmitted..... 157](#)

You can analyze signals of RX messages or change the values of signals of TX messages in the experiment software.

## Basics on the RTI CAN MultiMessage Blockset

## Introduction

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

## Supported dSPACE platforms

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

- SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board

- MicroAutoBox II
- MicroLabBox
- PHS-bus-based systems (DS1006 or DS1007 modular systems) containing one of the following I/O boards:
  - DS2202 HIL I/O Board
  - DS2210 HIL I/O Board
  - DS2211 HIL I/O Board
  - DS4302 CAN Interface Board
  - DS4505 Interface Board, if the DS4505 is equipped with DS4342 CAN FD Interface Modules

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

#### Note

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

#### Managing large CAN message bundles

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

#### Support of CAN FD protocol

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

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

For more information, refer to [Basics on Working with CAN FD](#) on page 143.

#### Support of AUTOSAR features

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

#### Manipulating signals to be transmitted

The RTI CAN MultiMessage Blockset provides several options to manipulate the values of signals before they are transmitted. You can switch between them with the Bus Navigator in ControlDesk via entries in the generated TRC file. In

addition, you can calculate checksum, parity, toggle, counter, and mode counter values.

### Updating a model

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

#### Tip

You do not have to recreate the S-function for the RTI CAN MultiMessage Blockset if you switch from one processor board to another, e.g., from a DS1006 to a DS1007, and vice versa.

When you switch to or from a MicroAutoBox II or MicroLabBox, you only have to recreate the ControllerSetup block. You also have to recreate the ControllerSetup block if you change the controller name.

### Modifying model parameters during run time



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

(Relevant only for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board) For information on where to find the signals of the CAN bus in the TRC file, refer to [Available TRC File Variable Entries and Their Locations in the TRC File \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(aa53ad6fea213b8b2226d3077e30533a\_img.jpg\)](#)).

### User-defined variables

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

### Working with variants of CAN communication

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

### Gatewaying messages between CAN buses

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

<b>Online modification of gateway exclude list</b>	You can modify the exclude list of RTICANMM Gateways during run time, i.e., specify messages not to be gatewayed.
<b>Dynamic message triggering</b>	You can modify the cycle times and initiate a spontaneous transmission (interactive or model-based) during run time.
<b>Defining free raw messages</b>	<p>You can define free raw messages as additional messages that are independent of the database file. Once they are defined, you can use them like standard database messages and specify various options, for example:</p> <ul style="list-style-type: none"> <li>▪ Trigger options</li> <li>▪ Ports and displays</li> <li>▪ Message ID and length adjustable during run time</li> </ul> <p>The following features are not supported:</p> <ul style="list-style-type: none"> <li>▪ Checksum generation</li> <li>▪ Custom signal manipulation</li> </ul>
<b>Capturing messages</b>	<p>You can process the raw data of messages whose IDs match a given filter via the capture messages feature. This can be a specific ID, a range of IDs, or even all IDs. Optionally, you can exclude the messages contained in the database file.</p> <p>The captured messages can be made available as outports of the MainBlock or in the TRC file.</p>
<b>CAN partial networking</b>	With CAN partial networking, you can set selected ECUs in a network to sleep mode or shut them down if they do not have to run continuously. Wake-up messages then activate specific ECUs as and when required, and for as long as required.

**Note**

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

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

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

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

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

For more information, refer to [Partial Networking Page \(RTICANMM ControllerSetup\)](#) (RTI CAN MultiMessage Blockset Reference .

---

#### TRC file entries with initial data

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

---

#### Visualization with the Bus Navigator

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

---

#### RTI CAN Blockset and RTI CAN MultiMessage Blockset

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

---

#### Further information on the RTI CAN MultiMessage Blockset

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

- [RTI CAN MultiMessage Blockset Reference](#) 

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

- [RTI CAN MultiMessage Blockset Tutorial](#) 

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

## Basics on Working with CAN FD

---

### Introduction

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

---

### Basics on CAN FD

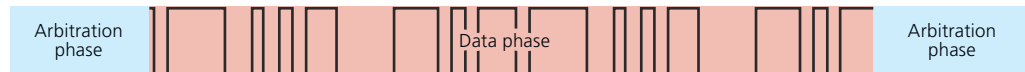
CAN FD stands for *CAN with Flexible Data Rate*. The CAN FD protocol is based on the CAN protocol as specified in ISO 11898-1. Compared with the classic CAN protocol, CAN FD comes with an increased bandwidth for the serial communication. The improvement is based on two factors:

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

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

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

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

**Classic CAN message****CAN FD message using a higher bit rate****CAN FD message with longer payload using a higher bit rate****CAN FD protocols**

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

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

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

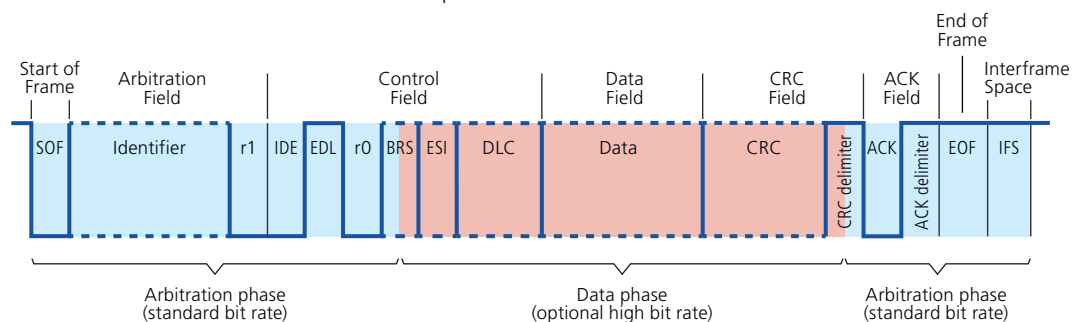
The RTI CAN MultiMessage Blockset supports both CAN FD protocols.

**CAN FD message characteristics**

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

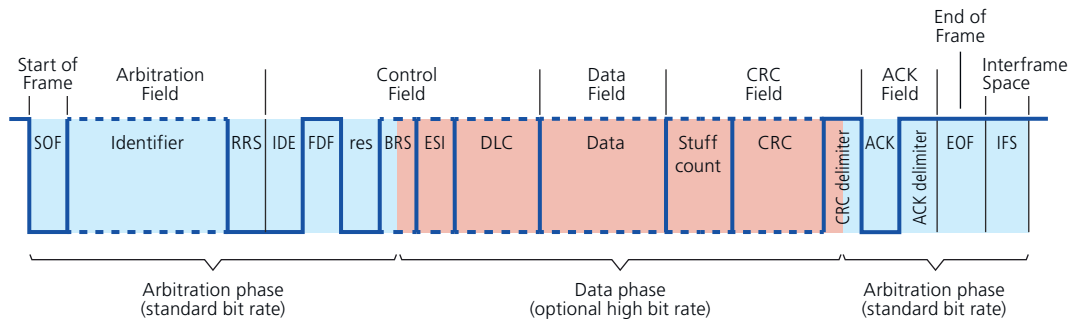
For an overview of the fields of a CAN FD message and the message components for each of the two CAN FD protocols, refer to the following illustration:

- Non-ISO CAN FD protocol:





■ ISO CAN FD protocol:



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

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

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

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

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

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



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

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

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

#### Activating CAN FD mode in the RTI CAN MultiMessage Blockset

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

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

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

#### Supported database file types

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

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

**CanFrameTxBehavior and CanFrameRxBehavior attributes** In AUTOSAR and FIBEX files, the `CanFrameTxBehavior` and/or `CanFrameRxBehavior`

attributes of a message can be defined to specify whether the message is to be treated as a CAN FD message or classic CAN 2.0 message. The RTI CAN MultiMessage Blockset evaluates the attribute as follows:

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

---

### Supported dSPACE platforms

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

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

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

- PHS-bus-based system with DS4505: [DS4342 Connections in Different Topologies \(PHS Bus System Hardware Reference !\[\]\(67ff022fd78f943b679992c2874bbfd1\_img.jpg\)](#))
- MicroAutoBox II: [DS4342 Connections in Different Topologies \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(042ea11c58a77088d3dd7150909adec0\_img.jpg\)](#))

---

### Working with CAN messages and CAN FD messages

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

---

### Related topics

#### References

[Setup Page \(RTICANMM ControllerSetup\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(0d5ec72f61334709c3fc9450209b754f\_img.jpg\)](#))

## Basics on Working with a J1939-Compliant DBC File

### Introduction

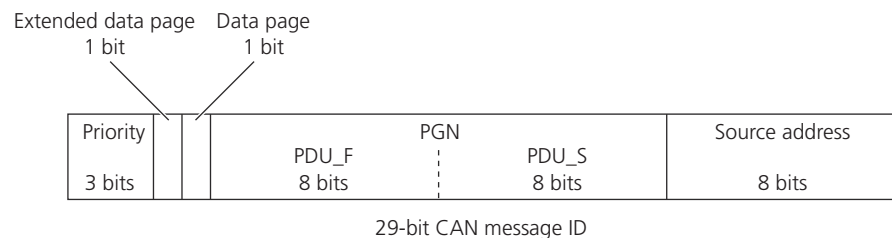
J1939 is a vehicle CAN bus standard defined by the Society of Automotive Engineers (SAE). It is used for communication in heavy-duty vehicles, for example, for communication between a tractor and its trailer.

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

### Broadcast and peer-to-peer communication

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

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



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

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

The first 8 bits of the PGN represent the *PDU\_F* (Protocol Data Unit format). The *PDU\_F* value specifies the communication mode of the message (peer-to-peer or broadcast). The interpretation of the *PDU\_S* value (PDU-specific) depends on the *PDU\_F* value. For messages with a *PDU\_F* < 240 (peer-to-peer communication, also called PDU1 messages), *PDU\_S* is not relevant for the PGN, but contains the destination address of the network node that receives the message. For

messages with a PDU\_F  $\geq 240$  (broadcast messages, also called PDU2 messages), PDU\_S specifies the second 8 bits of the PGN and represents the group extension. A group extension is used to increase the number of messages that can be broadcast in the network.

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

### Message attributes in J1939-compliant DBC files

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

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

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

### Source/destination mapping

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

#### Tip

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

### Container and instance messages

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

**Container message** A J1939 message defined by its PGN (and Data Page bit). A container can receive all the messages with its PGN in general. If several messages are received in one sampling step, only the last received message is

held in the container. Container messages can be useful, for example, when you configure a gateway with message manipulation.

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

#### Note

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

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

## Network management

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

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

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

The J1939 standard reserves the following addresses:

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

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

#### Note

The RTI CAN MultiMessage Blockset interprets attributes in the DBC file like this:

- In a DBC file created with CANalyzer 5.1 or earlier, the *name* network node attributes and the *J1939PGSrc* and *J1939PGDest* message attributes are read in. The *J1939PGSrc* attribute is interpreted as the address of the node that sends the message, the *J1939PGDest* attribute is interpreted as the address of the node that receives the message.
- In a DBC file created with CANalyzer 5.2 or later, the *name* and *NMStationAddress* network node attributes are read in. The *NMStationAddress* attribute is interpreted as the network node address.

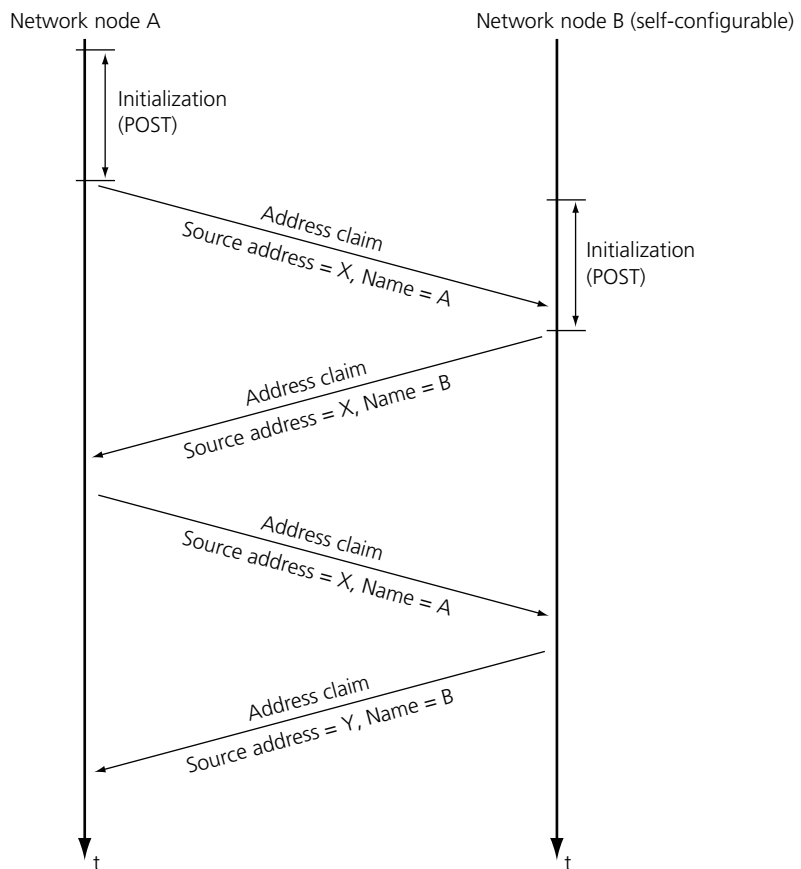
**Name** The J1939 standard defines a 64-bit *name* to identify each network node. The name indicates the main function of the network node with the associated address and provides information on the manufacturer.

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

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

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

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



The following steps are performed in the address claiming procedure:

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

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



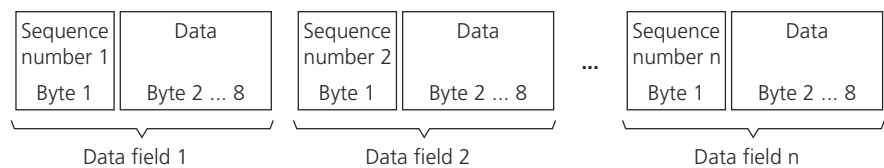
**Note**

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

**Messages > 8 bytes (message packaging)**

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

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



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

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

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

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

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

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

## Related topics

### Basics

[Lesson 13 \(Advanced\): Working with a J1939-Compliant DBC File \(RTI CAN MultiMessage Blockset Tutorial !\[\]\(0f848bbd71cef6b345273b16f905912a\_img.jpg\)\)](#)

# Transmitting and Receiving CAN Messages

## Introduction

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

## Defining CAN communication

To define the CAN communication of a CAN controller, you can specify a DBC, MAT, FIBEX, or AUTOSAR system description file as the database file on the [General Settings Page \(RTICANMM MainBlock\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(6059a5aa8b4ca7bb793408023d6c6e42\_img.jpg\)\)](#). You can also define CAN communication without using a database file.

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

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

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

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

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

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

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

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

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

---

### Defining RX messages and TX messages

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

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

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

---

### Triggering TX message transmission


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

For details, refer to [Triggering Options Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ).

---

### Triggering reactions to the reception of RX messages

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


For details, refer to [Raw Data Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ).

---

### Working with raw data

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

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

For details, refer to [Raw Data Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ).

## Implementing checksum algorithms

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

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

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

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

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

For details, refer to [Checksum Definition Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ).

## Gatewaying messages

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

You can gateway CAN messages in two ways:

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

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

For details, refer to [RTICANMM Gateway](#) (RTI CAN MultiMessage Blockset Reference ).

## Related topics

## References

[General Settings Page \(RTICANMM MainBlock\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(bd1a142de767a21e5362c595f844a4ff\_img.jpg\)](#))

# Manipulating Signals to be Transmitted

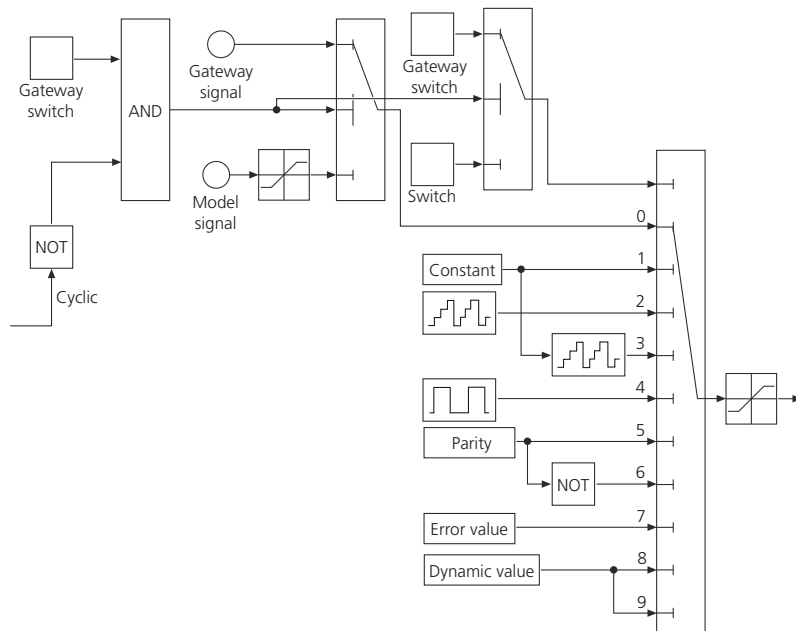
## Introduction

All the signals of all the RX and TX messages (see [Defining RX messages and TX messages](#) on page 155) automatically get corresponding entries in the generated TRC file. This allows you to analyze them (signals of RX messages) or change their values (signals of TX messages) with the Bus Navigator in ControlDesk.

## Manipulating signals to be transmitted

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

The illustration below visualizes the options.



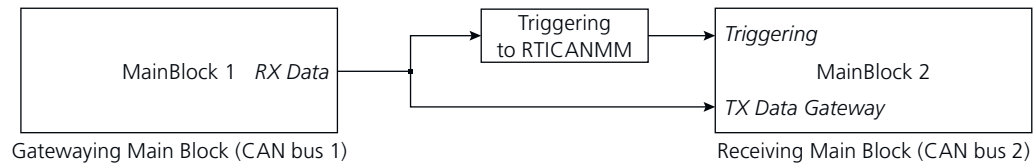
You can switch between these options in ControlDesk.

**TX model signal** A signal of a TX message whose value can be changed from within the model. By default, the values of TX model signals cannot be changed in ControlDesk. If you also want to manipulate TX model signals from ControlDesk, you have to select them on the [Input Manipulation Page \(RTICANMM MainBlock\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(bd3b31712ad9bab5a241210fa6925cdd\_img.jpg\)](#)).

Because additional code has to be generated, TX model signals reduce performance. For optimum performance, you should specify as few TX model signals as possible.

You can specify TX model signals on the [Model Signals \(TX\) Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ).

**Gateway signal** A signal to be manipulated before it is exchanged between two CAN buses. Gateway signals have to be gatewayed via two RTICANMM MainBlocks. You have to specify gateway signals for the receiving MainBlock.



MainBlock1 gatewayes messages and their signals to MainBlock2. Specifying gateway signals for MainBlock2 adds a TX Data Gateway input to it. The specified gateway signals are transmitted via CAN bus 2 with the signal values received from MainBlock1 when triggered by the messages received from MainBlock1. You therefore have to specify triggered message transmission for the messages of the gateway signals on the [Message Cyclic Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ). In addition, you can enable signal value switching for gateway signals during run time, for example, to transmit the signal constant value instead of the gateway value.

#### Note

Implementing gateway signals at least doubles the number of block inputs and therefore reduces performance. If you want to exchange messages between CAN controllers and do not want to perform signal manipulation, you should use the RTICANMM Gateway block instead.

You can specify gateway signals on the pages located in the [Gateway Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ).

**Toggle signal** A 1-bit signal that can be used, for example, to indicate whether CAN messages are transmitted. If a CAN message is transmitted, the toggle signal value alternates between 0 and 1. Otherwise, the toggle signal value remains constant.

You can specify toggle signals on the [Toggle Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ).

**Parity signal** A signal that a parity bit is appended to. You can specify one or more signals of a TX message as parity signals. A parity bit is calculated for the specified signals according to whether even or odd parity is selected. The bit is appended to the signal and the TX message is transmitted with the parity signal.

You can specify parity signals on the [Parity Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) )

**Counter signal** A signal of a TX message that is used to check for correct message transmission or to trigger the transmission of signals in a message.

- **Behavior of counter signals**

The value of a counter signal changes with every message transmission. You can specify the counter start, step, and stop values, etc. Each time the counter reaches the stop value, it turns around to the counter start value.

- **Use of counter signals**

You can specify counter signals to check for correct transmission of a message or trigger the transmission of signals in a message.

- *Checking correct message transmission:* The receiver of a message expects a certain counter signal value. For example, if the transmission of a message was stopped for two transmissions, the expected counter signal value and the real counter signal value can differ, which indicates an error. Counter signals used in this way are often called alive counters.

- *Triggering the transmission of signals:* You can trigger the transmission of signals if you specify a mode signal as a counter signal. The signal value of the mode signal triggers the transmission of mode-dependent signals. Because the counter signal value changes with every message transmission, this triggers the transmission of the mode-dependent signals. By using counter signals in this way, you can work with signals which use the same bytes of a message. Counter signals used in this way are often called mode counters.

- **Using counter signals in ControlDesk**

In ControlDesk, you can transmit the signal's constant, counter, or increment counter value. The increment counter value is the counter value incremented by the signal's constant value.

You can specify counter signals on the [Counter Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) )

**Error value** A static signal value that indicates an error. You can specify an error value for each signal to be transmitted. Alternatively, error values can be defined in a database file. In ControlDesk, you can switch to transmit the signal's error value, constant value, etc. However, you cannot change the error value during run time. In ControlDesk, you can use a Variable Array (MultiState LED value cell type) to indicate if the error value is transmitted.

You can specify error values on the [Signal Errors Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) )

**Dynamic value** A signal value that is transmitted for a defined number of times.

- *Behavior of dynamic values:* You can specify to use a dynamic value for each signal to be transmitted. In ControlDesk, you can specify to transmit the signal's constant value or dynamic value. If you switch to the dynamic value, it is transmitted for a defined number of times (countdown value). Then signal manipulation automatically switches back to the signal manipulation option used before the dynamic value.

- *Example of using dynamic values:* Suppose you specify a dynamic value of 8 and a countdown value of 3. If you switch to the dynamic value of the signal, the signal value 8 is sent the next 3 times the TX message is transmitted. Then the signal manipulation option is reset.

You can specify dynamic values on the pages located in the [Dynamic Signal Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).



# CAN Signal Mapping

## Introduction

Lists the CAN signals of the dSPACE real-time hardware and the mapping of these signals to RTI blocks and RTLib functions.

## CAN Signal Mapping

### Introduction

The CAN subsystem of MicroLabBox provides CAN interfaces that meet the ISO/DIS 11898 specifications.

### I/O mapping

The following table shows the pin numbering of the CAN\_TP1 module.

Controller/Channel	Signal	CAN Connector Pin (Sub-D Connector)
1	CAN 1 low in/out (CAN-L)	2
	CAN 1 high in/out (CAN-H)	7
	GND	3
2	CAN 2 low in/out (CAN-L)	4
	CAN 2 high in/out (CAN-H)	8
	GND	3

### Related RTI blocks

- RTICAN CONTROLLER SETUP
- RTICANMM ControllerSetup

### Related RTLib functions

Slave CAN Access Functions



# FPGA Support


Introduction	The I/O FPGA Type 1 unit of MicroLabBox provides an I/O FPGA module that is used for the standard I/O features of MicroLabBox or for custom FPGA applications.
--------------	--

Where to go from here


Information in this section

<a href="#">General Information on FPGA Support.....</a>	<a href="#">163</a>
Provides information on hardware and software components relevant for FPGA support.	
<a href="#">Accessing the I/O FPGA Type 1 Unit.....</a>	<a href="#">165</a>
There are some FPGA access functions which are required if you want to integrate an FPGA application into a handcoded processor application.	

Information in other sections

<a href="#">RTI Block Settings for the DS1202 FPGA I/O Type 1 (RTI FPGA Programming Blockset - FPGA Interface Reference </a> )
The block dialogs provide hardware-specific settings after you load one of the DS1202 FPGA I/O Type 1 frameworks.

## General Information on FPGA Support

Hardware components	The DS1302 I/O Board of MicroLabBox provides an I/O FPGA module. For information on the I/O FPGA module, refer to <a href="#">General Data (MicroLabBox Hardware Installation and Configuration </a> ).
---------------------	---

The I/O FPGA is directly connected to the MicroLabBox processor board (DS1202) via local bus. The I/O FPGA is connected to the I/O board's analog and digital I/O channels.

## Software components

The I/O FPGA module is used by the standard I/O features of MicroLabBox or optionally by custom FPGA applications.

**Standard FPGA application** The standard FPGA application provides the I/O features of MicroLabBox that you can access via the RTI1202 Blockset and the RTI Electric Motor Control (EMC) Blockset, and via the I/O functions provided by RTLib. This FPGA application is automatically programmed to the FPGA if the real-time application that you load to the hardware does not contain a custom FPGA application.

**Custom FPGA application** For generating a custom FPGA application, you can use the RTI FPGA Programming Blockset.

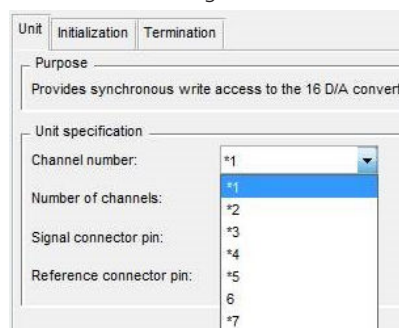
The FPGA Interface sublibrary provides the following two frameworks for MicroLabBox:

- **DS1202 FPGA I/O Type 1**

If you use this framework for a custom FPGA application, you cannot use RTI blocks and RTLib functions for the standard I/O features in the same real-time application. Only the custom FPGA application will be programmed to the FPGA when you download the real-time application.

- **DS1202 FPGA I/O Type 1 (Flexible I/O)**

If you use this framework for a custom FPGA application, you can use RTI blocks and RTLib functions for the standard I/O features in the same real-time application. However, you must not access the same channel with the custom FPGA application and the standard FPGA application. After you build the custom FPGA application, the I/O channels used by the custom FPGA application are marked by an asterisk in the RTI1202 and RTI Electric Motor Control block dialogs.



For more information on using the flexible I/O mode, refer to [Features of the MicroLabBox Flexible I/O Framework \(RTI FPGA Programming Blockset Guide\)](#).

For more information on the frameworks, refer to [RTI Block Settings for the DS1202 FPGA I/O Type 1 \(RTI FPGA Programming Blockset - FPGA Interface Reference !\[\]\(8af806fb1314382d09bc5ec5b767526c\_img.jpg\)](#)).

**Processor application** The RTI FPGA Programming Blockset also provides the Processor Interface sublibrary to implement the communication on the processor side.

If you want to integrate an FPGA application into a handcoded processor application, you have to use the `IoFpga_xxx` functions from RTLib to implement the communication on the processor side. For more information, refer to [Accessing the I/O FPGA Type 1 Unit](#) on page 165.

## Related topics

### Basics

[RTI FPGA Programming Blockset Guide](#)

# Accessing the I/O FPGA Type 1 Unit

## Introduction

There are some FPGA access functions which are required if you want to integrate an FPGA application into a handcoded processor application.

## General information on the access functions

**Module initialization** Before the initialization of the I/O FPGA Type 1 module, the processor board must be initialized and an FPGA application must be running. Because the module initialization is terminated if there is no FPGA application running, one must be programmed beforehand. Module initialization succeeds only if the FPGA application is compatible with the processor application.

**Identification** To avoid hardware damage, the components used (processor application, FPGA application, FPGA framework) must be compatible with each other. To check their compatibility, their identifiers can be read and compared using the identification functions.

**Interrupt handling** The FPGA Type 1 module provides 32 interrupt channels which you can handle using the RTLib interrupt functions, refer to [Interrupt Functions \(MicroLabBox RTLib Reference !\[\]\(0fb13ad0bfa3d86868cdd3883e5665b3\_img.jpg\)](#)).

**Data exchange** With the RTLib, you can implement the processor's read and write access to the data storage that is defined in the FPGA framework. Data exchange with the FPGA application requires data type conversion, which is configured by the `scaling` and `mode` parameters.

The FPGA Type 1 module provides the following channel types and channel numbers:

- 32 Buffer In channels
- 32 Buffer Out channels

- 256 Register In channels
- 256 Register Out channels

The maximum number of elements in a buffer is specified in the FPGA framework. You can use buffers with a data width of 32 bits or 64 bits. The configuration of a buffer is valid for all its elements. Buffers are accessed sequentially from the FPGA.

You can also use registers with a data width of 32 bits or 64 bits. You can specify groups of registers, which can be accessed synchronously by the FPGA. Each register in a group is configured separately.

**Programming** The FPGA application that you build must be programmed to the FPGA. This is automatically done by loading the generated real-time application to MicroLabBox. Whether the real-time application containing the FPGA application is loaded to the RAM or flash memory of the board depends on the load settings of the real-time application.

---

#### RTLib and RTI FPGA Programming Blockset

These access functions are available via RTLib, refer to [FPGA Module Access Functions \(MicroLabBox RTLib Reference !\[\]\(5a132f13505a6571904d622757b7a8f0\_img.jpg\)](#)).

If you have installed the RTI FPGA Programming Blockset, you can also use its **Processor Interface** sublibrary to implement the FPGA access functions, for example, the data access between the processor model and the FPGA model. With the **PROC\_SETUP\_BL** block, you can manage your FPGA applications. Internally, these RTI blocks use the RTLib functions for the I/O FPGA Type 1 unit.

# Limitations

**Introduction** The following limitations apply to working with MicroLabBox.

Where to go from here	Information in this section
	<a href="#">Details on the Limitations for MicroLabBox.....</a> 167 There are limitations concerning implementation features.
	<a href="#">Limited Number of CAN Messages.....</a> 167 When you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions, the number of CAN messages in an application is limited.
	<a href="#">Limitations with RTICANMM.....</a> 169 There are a number of general limitations with RTICANMM.

## Details on the Limitations for MicroLabBox

**Unsupported features** MicroLabBox does not support the bypassing services.

## Limited Number of CAN Messages

**Limitation** When you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions, the number of CAN messages in an application is limited.

This applies to the following message types:

- Transmit (TX) messages
- Receive (RX) messages
- Request (RQ) messages

An RQ message and the corresponding RX message are interpreted as a single (RQ) message. You cannot enable RX service support for the corresponding RX message.

- Remote (RM) messages

The sum of these messages is  $n_{sum}$ :

$$n_{sum} = n_{TX} + n_{RX} + n_{RQ} + n_{RM}$$

### Maximum number of CAN messages

The sum of the above messages  $n_{sum}$  in one application must always be smaller than or equal to the maximum number of CAN messages  $n_{max}$ :

$$n_{sum} \leq n_{max} ; n_{RM} \leq 10$$

$n_{max}$  in one application depends on:

- Whether you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions.
- Whether you use RX service support.

The maximum number of CAN messages  $n_{max}$  is listed in the table below:

Platform	$n_{max}$ with RTLib	$n_{max}$ with RTI CAN Blockset							
		RX Service Support Disabled				RX Service Support Enabled			
		1 <sup>1)</sup>	2 <sup>1)</sup>	3 <sup>1)</sup>	4 <sup>1)</sup>	1 <sup>1)</sup>	2 <sup>1)</sup>	3 <sup>1)</sup>	4 <sup>1)</sup>
DS2202 (2 CAN controllers)	100	98	96	–	–	96 <sup>2)</sup>	92 <sup>2)</sup>	–	–
DS2210 (2 CAN controllers)	100	98	96	–	–	96 <sup>2)</sup>	92 <sup>2)</sup>	–	–
DS2211 (2 CAN controllers)	100	98	96	–	–	96 <sup>2)</sup>	92 <sup>2)</sup>	–	–
MicroAutoBox II <sup>3)</sup> (2 CAN controllers per CAN_Type1)	100	98	96	–	–	96 <sup>2)</sup>	92 <sup>2)</sup>	–	–
MicroLabBox (2 CAN controllers)	100	98	96	–	–	96 <sup>2)</sup>	92 <sup>2)</sup>	–	–
DS4302 (4 CAN controllers)	200	198	196	194	192	196 <sup>2)</sup>	192 <sup>2)</sup>	188 <sup>2)</sup>	184 <sup>2)</sup>

<sup>1)</sup> Number of CAN controllers used in the application

<sup>2)</sup> It is assumed that RX service support is enabled for all the CAN controllers used, and that both CAN message identifier formats (STD, XTD) are used.


<sup>3)</sup> Depending on the variant, the MicroAutoBox II contains up to three CAN\_Type1 modules, each with 2 CAN controllers. The values in the list apply to a single CAN\_Type1 module.



### Ways to implement more CAN messages

There are two ways to implement more CAN messages in an application.

**Using RX service support** If you use RTI CAN Blockset's RX service support, the number of receive (RX) messages  $n_{RX}$  in the equations above applies only to RTICAN Receive (RX) blocks for which RX service support is not enabled. The number of RTICAN Receive (RX) blocks for which RX service support is enabled is unlimited. Refer to [Using RX Service Support](#) on page 132.

**Using the RTI CAN MultiMessage Blockset** To implement more CAN messages in an application, you can also use the RTI CAN MultiMessage Blockset. Refer to the [RTI CAN MultiMessage Blockset Tutorial](#) .

### Maximum number of CAN subinterrupts

The number of available CAN subinterrupts you can implement in an application is limited:


Platform	Available CAN Subinterrupts
DS2202 (2 CAN controllers)	15
DS2210 (2 CAN controllers)	15
DS2211 (2 CAN controllers)	15
MicroAutoBox II <sup>1)</sup> (2 CAN controllers per CAN_Type1)	15
MicroLabBox (2 CAN controllers)	15
DS4302 (4 CAN controllers)	31

<sup>1)</sup> Depending on the variant, the MicroAutoBox II contains up to 3 CAN\_Type1 modules, each with 2 CAN controllers. The values in the list apply to a single CAN\_Type1 module.

## Limitations with RTICANMM

### RTI CAN MultiMessage Blockset

The following limitations apply to the RTI CAN MultiMessage Blockset:

- The configuration file supports only messages whose name does not begin with an underscore.
- Do not use the RTI CAN MultiMessage Blockset and the RTI CAN Blockset for the same CAN controller.
- Do not use the RTI CAN MultiMessage Blockset in enabled subsystems, triggered subsystems, configurable subsystems, or function-call subsystems. As an alternative, you can disable the entire RTI CAN MultiMessage Blockset by switching the CAN controller variant, or by setting the GlobalEnable triggering option. This option is available on the [Triggering Options Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).
- Do not run the RTI CAN MultiMessage Blockset in a separate task.

- Do not copy blocks of the RTI CAN MultiMessage Blockset. To add further blocks of the RTI CAN MultiMessage Blockset to a model, always take them directly from the `rticanmm.lib` library. To transfer settings between two MainBlocks or between two Gateway blocks, invoke the Save Settings and Load Settings commands from the Settings menu (refer to RTICANMM MainBlock or [RTICANMM Gateway \(RTI CAN MultiMessage Blockset Reference\)](#))).
- The RTI CAN MultiMessage Blockset is not included in the RTI update mechanism and is not updated when you open a model with an older version. To update the RTI CAN MultiMessage Blockset, invoke Create S-Function for All RTICANMM Blocks from the Options menu of the [RTICANMM GeneralSetup \(RTI CAN MultiMessage Blockset Reference\)](#)).

As an alternative, you can create new S-functions for all RTICANMM blocks manually (use the following order):

1. [RTICANMM GeneralSetup \(RTI CAN MultiMessage Blockset Reference\)](#)
  2. [RTICANMM ControllerSetup \(RTI CAN MultiMessage Blockset Reference\)](#)
  3. [RTICANMM MainBlock \(RTI CAN MultiMessage Blockset Reference\)](#)
  4. [RTICANMM Gateway \(RTI CAN MultiMessage Blockset Reference\)](#)
- Model path names with multi-byte character encodings are not supported.
  - Mode signals with opaque byte order format that are longer than 8 bits are not supported.
  - The RTI CAN MultiMessage Blockset generates data structures on the basis of the relevant element names specified in the database file. The length of an element name is limited to 56 characters. If an element name exceeds this limit, the RTI CAN MultiMessage Blockset shortens the name to 56 characters, using a checksum to ensure name uniqueness, and makes an entry in the log file.

The following list shows the element types whose maximum name length must not exceed 56 characters:

- Messages
- Signals
- UpdateBit signals
- Mode signals
- Nodes
- Simulink can store design data that your model uses in a data dictionary as a persistent repository. Data dictionaries are not supported by the RTI CAN MultiMessage Blockset.

---

**FIBEX 3.1, FIBEX 4.1,  
FIBEX 4.1.1, or FIBEX 4.1.2 file  
as the database**

The RTI CAN MultiMessage Blockset does not support multiple computation methods for signals. If several CompuMethods are defined for a signal in the FIBEX file, the RTI CAN MultiMessage Blockset uses the first linear computation method it finds for the signal.

**MAT file as the database**

In the RTI CAN MultiMessage Blockset, the length of signal names is restricted to 32 characters. However, MATLAB allows longer signal names. When MATLAB entries are mapped to the signals in RTICANMM, the signal names are truncated at the end and supplemented by a consecutive number, if necessary. To ensure that unchanged signal names are used in the RTI CAN MultiMessage Blockset, the signal names in the Simulink model must not exceed 32 characters.

**AUTOSAR system description file as the database**

- The RTI CAN MultiMessage Blockset does not support the following features that can be defined in an AUTOSAR 3.2.2, 4.0.3, 4.1.1, 4.1.2, 4.2.1, 4.2.2, 4.3.0, 4.3.1, 4.4.0, or AUTOSAR Classic R19-11 or R20-11 system description file:
  - Partial networking (There are some exceptions: Partial networking is supported for the MicroAutoBox II equipped with a DS1513 I/O Board, the MicroLabBox, and dSPACE hardware that is equipped with DS4342 CAN FD Interface Modules.)
  - Unit groups
  - Segment positions for MultiplexedIPdus
  - End-to-end protection for ISignalGroups
- The RTI CAN MultiMessage Blockset does not support the new features of AUTOSAR Release 4.4.0 and AUTOSAR Classic Platform Release R19-11 and R20-11.
- When you work with an AUTOSAR ECU Extract as the database, the RTI CAN MultiMessage Blockset does not support frames with multiplexed IPDUs whose PDUs are only partially included (e.g., the imported ECU Extract contains only their dynamic parts while their static parts are contained in another ECU Extract).

**Limitations for container IPDUs**

- The RTI CAN MultiMessage Blockset does not support nested container IPDUs.
- For contained IPDUs that are included in container IPDUs with a dynamic container layout, the RTI CAN MultiMessage Blockset does not support the long header type. For the **ContainerIpduHeaderType** AUTOSAR attribute, the RTI CAN MultiMessage Blockset supports only the **SHORT\_HEADER** value.
- For the **ContainedIpduCollectionSemantics** AUTOSAR attribute, the RTI CAN MultiMessage Blockset supports the **QUEUED** and **LAST\_IS\_BEST** values. However, when a container IPDU with a queued semantics is received that contains multiple instances of a contained IPDU, only the last received instance is displayed.
- For the **RxAcceptContainedIpdu** AUTOSAR attribute, the RTI CAN MultiMessage Blockset does not support the **ACCEPT\_CONFIGURED** value for container IPDUs, which allows only a certain set of contained IPDUs in a container IPDU.
- The RTI CAN MultiMessage Blockset supports TX message length manipulation (static and dynamic length manipulation) only for contained IPDUs that are included in container IPDUs with a dynamic container layout, not for contained IPDUs included in container IPDUs with a static container layout and also not for dynamic and static container IPDUs.

- The RTI CAN MultiMessage Blockset lets you manipulate the length of a contained IPDU that is included in container IPDUs with a dynamic container layout as long as the IPDU has not yet been written to a container IPDU. Once a contained IPDU is written to its container IPDU, the length manipulation options no longer have any effect on the instance of the contained IPDU that is currently triggered and written to the container IPDU. But the length manipulation options take effect again when the contained IPDU is triggered the next time. Length manipulation is not supported for contained IPDUs that are included in container IPDUs with a static container layout.
- The RTI CAN MultiMessage Blockset supports TX message ID manipulation only for contained IPDUs that are included in container IPDUs with a dynamic container layout, not for contained IPDUs that are included in container IPDUs with a static container layout and also not for dynamic and static container IPDUs. By activating the TX message ID manipulation option for contained IPDUs in dynamic container IPDUs, you actually manipulate the `SHORT_HEADER` of the contained IPDUs.
- The RTI CAN MultiMessage Blockset supports neither TX signal manipulation nor gateway signal manipulation for container IPDU signals.
- When you gateway messages using the RTICANMM Gateway block, you cannot exclude contained IPDUs from being gatewayed. Excluding container IPDUs is possible.

---

#### Limitations for secure onboard communication

- The RTI CAN MultiMessage Blockset does not support counters as freshness values. Only time stamp values can be used as freshness values.
- Cryptographic IPDUs are not displayed on the dialog pages of the RTICANMM MainBlock.
- The RTI CAN MultiMessage Blockset supports secured PDU headers only for container IPDUs with a dynamic container layout. For all other IPDU types, secured PDU headers are not supported.

---

#### Limitations for global time synchronization

- The RTI CAN MultiMessage Blockset does not support the simulation of a global time master.
- The RTI CAN MultiMessage Blockset does not support offset GTS messages (offset synchronization messages (OFS messages) and offset adjustment messages (OFNS messages)).
- GTS messages are not displayed on the Checksum Messages Page (RTICANMM MainBlock). In the case of secured GTS messages, a predefined checksum algorithm is used if the GTS manipulation option is selected on the Signal Default Manipulation Page (RTICANMM MainBlock) for the SyncSecuredCRC and FupSecuredCRC signals.
- The RTI CAN MultiMessage Blockset does not support switching between the secured and the unsecured GTS message types at run time, i.e., you cannot switch from a CRC-secured SYNC and FUP message pair to an unsecured message pair, or vice versa.
- If multiple time slaves are defined for a GTS message, only the highest `FupTimeout` value is imported and can be used during run time.

- Only valid pairs of SYNC and FUP messages can update the time in a time base manager instance. SYNC and FUP messages form a valid pair if they meet the following conditions:
  - Both messages use the same CAN identifier and the same ID format.
  - Both messages use the same time domain identifier.
  - Both messages must be CRC-secured or both must be unsecured.
- For signals of GTS messages, the RTI CAN MultiMessage Blockset only supports Global time synchronization and Constant as TX signal manipulation options, where Global time synchronization is set as default option. Other TX signal manipulation options are not supported for signals of GTS messages.
- The RTI CAN MultiMessage Blockset does not support gateway signal manipulation for signals of GTS messages.
- For the `crcValidated` AUTOSAR attribute, the RTI CAN MultiMessage Blockset does not support the following values:
  - `crcIgnored`
  - `crcOptional`
- Clearing the Use specific data types checkbox on the Code Options Page (RTICANMM MainBlock) of the RTICANMM MainBlock has no effect on GTS messages. GTS messages always use specific data types.

---

#### Visualization with the Bus Navigator

The current version of the RTI CAN MultiMessage Blockset supports visualization with the Bus Navigator in ControlDesk 4.2.1 or later. You cannot work with earlier versions of ControlDesk in connection with applications created with the current version of the RTI CAN MultiMessage Blockset.



# Appendix

Where to go from here

Information in this section

<a href="#">Troubleshooting.....</a>	<a href="#">175</a>
Gives you information on known problems and their solutions.	
<a href="#">Glossary.....</a>	<a href="#">176</a>
The glossary briefly explains the most important expressions and naming conventions used in the MicroLabBox documentation.	

## Troubleshooting

Known problem reports

To stay up-to-date with information on possible problems, you should periodically check the known problem reports at <http://www.dspace.com/go/ProblemReports>.

## Glossary

---

**ACU** Angle Computation Unit

A controller unit that handles the input and output signals for electric motor control.

**ADC** Analog/digital converter

**BCPWM** Block-commutated PWM signal generation (used for electric motor control)

**CAN** Controller area network

**Class 1/Class 2** Specifies the channel groups with which an I/O function can be used.

**CN** Computation node

This is the processor that provides the calculation power for your real-time model.

**DAC** Digital/analog converter

**DIO** Digital input/output

**DPMEM** Dual-port memory

**ECU** Electronic control unit

**EMC** Electric motor control

**ENC** Incremental encoder interface

**FPGA** Field Programmable Gate Array. An FPGA module is used in digital technology to allow modifications in a circuit's functionality without replacing hardware.

**HCN** Host communication node

This is a co-processor that handles the communication with the host PC as well as data streaming to the USB mass storage device, if available.

**MCPWM** Multichannel PWM signal generation

**PPC** Power PC processor

**PWM** Pulse width modulation

**RTI** Real-Time Interface

dSPACE product that provides several blocksets, which can be used with MATLAB®/Simulink®.

**RTLib** Real-Time Library

dSPACE product that provides several C function libraries for handcoding real-time applications.

**SPI** Serial peripheral interface



**A**

- A/D conversion
  - MicroLabBox 40
- ADC Class 1
  - burst start interrupt 46
  - burst trigger 42
  - characteristics 40
  - conversion trigger 42
  - data ready interrupt 47
  - failure interrupt 48
  - I/O mapping 49
  - interrupts 46
  - swinging buffer 41
  - trigger line 42
  - trigger signals 42
  - trigger sources 42
- ADC Class 2
  - characteristics 50
  - conversion behavior 50
  - I/O mapping 50
  - RTI/RTLib support 50
- appendix
  - MicroLabBox 175
- application
  - flash memory 21
  - local memory 20
- application start
  - MicroLabBox 17
- auto negotiation 26

**B**

- bit I/O
  - MicroLabBox 54
- Bit I/O (DIO Class 1) 54
  - characteristics 54
  - I/O mapping 56
  - interrupts 55
  - RTI/RTLib support 55
  - trigger lines 55
- Bit I/O (DIO Class 2) 56
  - characteristics 56
  - I/O mapping 57
  - RTI/RTLib support 56
- block-commutated PWM signal generation (DIO Class 1) 108
  - characteristics 108
  - I/O mapping 112
  - interrupts 111
  - RTI/RTLib support 111
  - trigger lines 111
- broadcast 26
- burst trigger
  - ADC Class 1 42

**C**

- CAN
  - channel 124
  - fault-tolerant transceiver 128
  - interrupts 132

- physical layer 126
- service 135
- setup 124
- status information 135
- CAN FD 143
- CAN support 123
- channel classes 11
- characteristics
  - SPI (DIO Class 1) 83
- Class 1 11
- Class 2 11
- Common Program Data folder 8
- conversion trigger
  - ADC Class 1 42

**D**

- D/A conversion
  - MicroLabBox 52
- DAC Class 1
  - characteristics 52
  - I/O mapping 52
  - RTI/RTLib support 52
  - synchronous update 52
- data file support 131
- data rates 26
- datagram size 26
- defining CAN messages 130
- DHCP 26
- DIO Class 1
  - SPI 83
- Documents folder 8

**E**

- electric motor control
  - interrupts 93
  - MicroLabBox 91
  - trigger lines 93
- EnDat interface 103
  - characteristics 104
  - I/O mapping 105
  - RTI/RTLib support 105

**F**

- failure
  - burst trigger overflow 48
  - conversion trigger overflow 48
  - data lost 48
  - no burst start 48
  - store error 48
- features of MicroLabBox 14
- firmware 19
- flash memory
  - application 21
- FPGA support 163
  - access functions 165
  - accessing I/O FPGA Type 1 165
  - hardware components 163
  - RTI support 166
  - RTLib support 166
  - software components 164

- functional units
  - MicroLabBox 9

**H**

- Hall sensor interface 95
  - characteristics 95
  - I/O mapping 97
  - RTI/RTLib support 97
- host interface 25

**I**

- I/O mapping
  - ADC Class 1 49
  - ADC Class 2 50
  - Bit I/O (DIO Class 1) 56
  - Bit I/O (DIO Class 2) 57
  - block-commutated PWM signal generation (DIO Class 1) 112
  - DAC Class 1 52
  - EnDat interface 105
  - Hall sensor interface 97
  - incremental encoder interface 100
  - multichannel PWM signal generation (DIO Class 1) 120
  - pulse signal generation (DIO Class 1) 67
  - pulse width measurement (DIO Class 1) 69
  - PWM signal generation (DIO Class 1) 62
  - PWM signal measurement (DIO Class 1) 64
  - resolver interface 103
  - serial interface 82
  - SPI (DIO Class 1) 89
  - SSI interface 108
- incremental encoder interface 98
  - characteristics 98
  - I/O mapping 100
  - RTI/RTLib support 100
- interface
  - host 25
- interrupt
  - burst start 46
  - data ready 47
  - failure 48
- interrupts
  - ADC Class 1 46
  - Bit I/O (DIO Class 1) 55
  - block-commutated PWM signal generation (DIO Class 1) 111
  - electric motor control 93
  - multichannel PWM signal generation (DIO Class 1) 119
  - pulse width measurement (DIO Class 1) 68
  - PWM signal generation (DIO Class 1) 59
  - PWM signal measurement (DIO Class 1) 63
  - SPI (DIO Class 1) 84
- IP fragmentation 26
- ISO11898 127
- ISO11898-6 128

**J**

- J1939

broadcast/peer-to-peer messages 148  
 working with J1939-compliant DBC files 148  
 JTAG interface 28

## L

limitations  
 MicroLabBox 167  
 RTI CAN MultiMessage Blockset 169  
 local memory  
 application 20  
 Local Program Data folder 8

## M

MAT format 79  
 messages  
 defining CAN messages 130  
 delay time 130  
 multiple 131  
 MicroLabBox  
 A/D conversion 40  
 appendix 175  
 application start 17  
 bit I/O 54  
 D/A conversion 52  
 electric motor control 91  
 limitations 167  
 system overview 9  
 timing I/O 58  
 multichannel PWM signal generation (DIO Class 1) 112  
 characteristics 112  
 I/O mapping 120  
 interrupts 119  
 RTI/RTLib support 120  
 trigger lines 119  
 multiple data files 131  
 multiple message support 131

## N

network topology for dual-core systems 16

## O

overview  
 MicroLabBox features 14

## P

PGI interface 28  
 piggyback support 129  
 pulse signal generation (DIO Class 1) 65  
 characteristics 65  
 I/O mapping 67  
 RTI/RTLib support 67  
 trigger lines 65  
 pulse width measurement (DIO Class 1) 67  
 characteristics 67  
 I/O mapping 69  
 interrupts 68  
 RTI/RTLib support 69

trigger lines 68  
 PWM signal generation (DIO Class 1) 58  
 characteristics 58  
 I/O mapping 62  
 interrupts 59  
 RTI/RTLib support 62  
 trigger lines 59  
 PWM signal measurement (DIO Class 1) 62  
 characteristics 63  
 I/O mapping 64  
 interrupts 63  
 RTI/RTLib support 64  
 trigger lines 63

## R

resolver interface 101  
 characteristics 101  
 I/O mapping 103  
 RTI/RTLib support 103  
 routing 26  
 RS232 81  
 RS422 81  
 RS485 81, 128  
 RTI CAN MultiMessage Blockset  
 limitations 169  
 supported platforms 138  
 RTI/RTLib support  
 Bit I/O (DIO Class 1) 55  
 Bit I/O (DIO Class 2) 56  
 block-commutated PWM signal generation (DIO Class 1) 111  
 EnDat interface 105  
 Hall sensor interface 97  
 incremental encoder interface 100  
 multichannel PWM signal generation (DIO Class 1) 120  
 pulse signal generation (DIO Class 1) 67  
 pulse width measurement (DIO Class 1) 69  
 PWM signal generation (DIO Class 1) 62  
 PWM signal measurement (DIO Class 1) 64  
 resolver interface 103  
 serial interface 81  
 SPI (DIO Class 1) 88  
 SSI interface 108  
 running application  
 USB mass storage device 22  
 RX service support 132

## S

serial interface 81  
 I/O mapping 82  
 RTI/RTLib support 81  
 serial peripheral interface  
 DIO Class 1 83  
 SPI  
 DIO Class 1 83  
 SPI (DIO Class 1)  
 characteristics 83  
 examples 86  
 I/O mapping 89

interrupts 84  
 RTI/RTLib support 88  
 timing behavior 85  
 trigger lines 84  
 SSI interface 105  
 characteristics 106  
 I/O mapping 108  
 RTI/RTLib support 108  
 swinging buffer  
 ADC Class 1 41  
 system overview  
 MicroLabBox 9

## T

timing I/O  
 MicroLabBox 58  
 TJA1041  
 limitations 129  
 TJA1054  
 transceiver 129  
 transceiver  
 TJA1054 129  
 trigger line  
 ADC Class 1 42  
 trigger lines  
 Bit I/O (DIO Class 1) 55  
 block-commutated PWM signal generation (DIO Class 1) 111  
 electric motor control 93  
 multichannel PWM signal generation (DIO Class 1) 119  
 pulse signal generation (DIO Class 1) 65  
 pulse width measurement (DIO Class 1) 68  
 PWM signal generation (DIO Class 1) 59  
 PWM signal measurement (DIO Class 1) 63  
 SPI (DIO Class 1) 84  
 trigger signals  
 ADC Class 1 42  
 trigger sources  
 ADC Class 1 42

## U

UART 81  
 USB Flight Recorder 75  
 avoiding data loss 77  
 basics 75  
 FTP connection 78  
 handling the data 79  
 limitations 79  
 max. data rate 78  
 overwrite mode 76  
 USB mass storage device 76  
 USB status LED 78  
 USB interface 27  
 USB mass storage device  
 running application 22  
 using multicore MicroLabBox 16

## W

working with CAN FD 143