

DS2202 HIL I/O Board

# 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.

© 2005 - 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	5
Introduction to the Features of the DS2202	7
Functional Units of the DS2202	7
Features of the DS2202	9
Integration of the DS2202 in a Modular System	10
Multi I/O Interface	13
ADC Unit	14
DAC Unit	16
Bit I/O Unit	18
PWM Signal Measurement	22
PWM Signal Generation	25
Frequency Measurement (F2D)	30
Square-Wave Signal Generation (D2F)	32
Serial Interface	37
Basics on the Serial Interface	37
Comparing RS232 and RS422	39
Specifying the Baud Rate of the Serial Interface	40
Software FIFO Buffer	41
CAN Support	43
Setting Up a CAN Controller	44
Initializing the CAN Controller	44
CAN Transceiver Types	46
Defining CAN Messages	50
Implementing a CAN Interrupt	52
Using RX Service Support	52
Removing a CAN Controller (Go Bus Off)	54
Getting CAN Status Information	55
Using the RTI CAN MultiMessage Blockset	57
Basics on the RTI CAN MultiMessage Blockset	57
Basics on Working with CAN FD	62

Basics on Working with a J1939-Compliant DBC File.....	67
Transmitting and Receiving CAN Messages.....	73
Manipulating Signals to be Transmitted.....	76
CAN Signal Mapping.....	80
CAN Signal Mapping.....	80
<b>Interrupt Handling and DS2202 Interrupts</b>	<b>81</b>
Basics of Interrupt Handling.....	81
Basics of DS2202 Interrupts.....	82
<b>Limitations</b>	<b>85</b>
Quantization Effects.....	85
Conflicting I/O Features.....	86
Limited Number of CAN Messages.....	89
Limitations with RTICANMM.....	91
Limitations with J1939-Support.....	95
<b>Index</b>	<b>97</b>





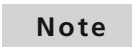



# About This Document

## Content

This document provides feature-oriented access to the reference information you need to implement the functions provided by the DS2202 HIL I/O Board. For an overview of all features, refer to [Introduction to the Features of the DS2202](#) on page 7.

## Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
	Indicates a hazard that, if not avoided, could result in property damage.
	Indicates important information that you should take into account to avoid malfunctions.
	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 the DS2202

---

## Introduction

The DS2202 HIL I/O Board is designed to simulate and measure automotive signals.

---

## Where to go from here

## Information in this section

<a href="#">Functional Units of the DS2202.....</a>	<a href="#">7</a>
Provides an overview of the functional units of the DS2202.	
<a href="#">Features of the DS2202.....</a>	<a href="#">9</a>
Provides an overview of the features of the DS2202.	
<a href="#">Integration of the DS2202 in a Modular System.....</a>	<a href="#">10</a>
As an I/O board, the DS2202 is always part of a PHS-bus-based system.	

## Functional Units of the DS2202

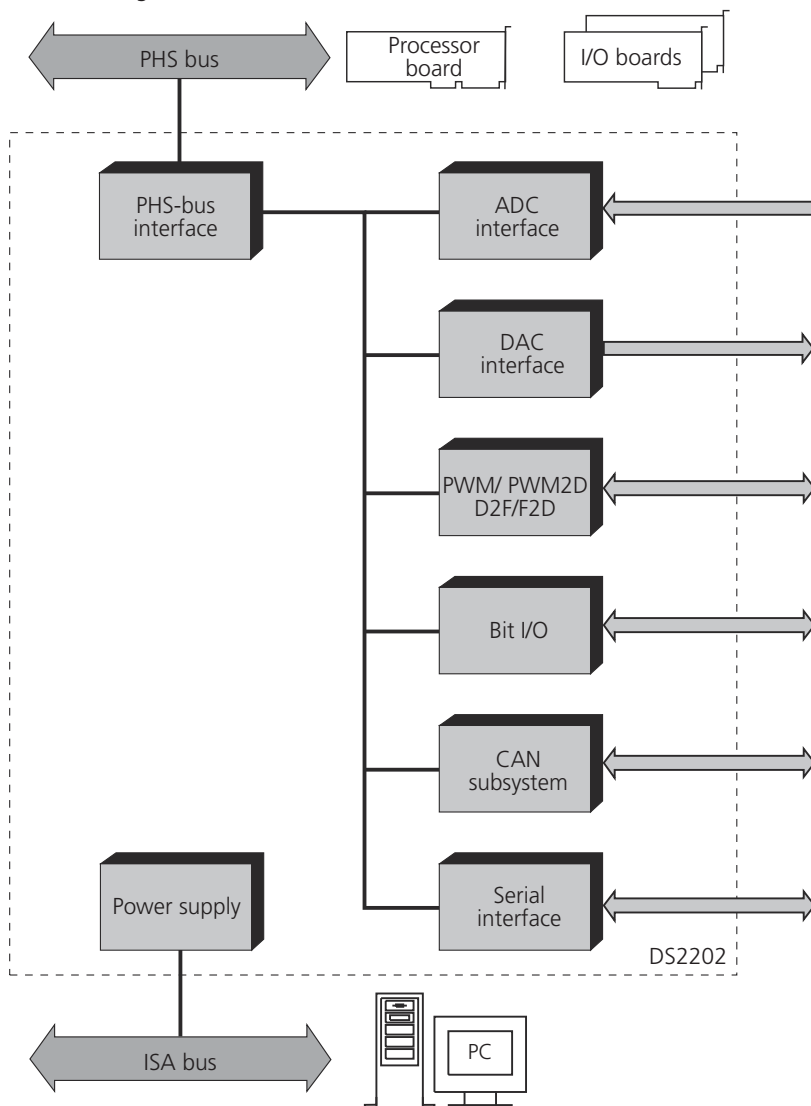
---

## Introduction

The DS2202 HIL I/O Board provides different functional units.

**Overview of functional units**

The following illustration shows the functional units of the DS2202.



**Legend** The illustration uses the following abbreviations:

ADC	Analog/digital converter
CAN	Controller area network
DAC	Digital/analog converter
PWM	Pulse width modulation
PWM2D	PWM signal measurement
F2D	Square-wave signal measurement
D2F	Square-wave signal generation



<b>Multi I/O interface</b>	The DS2202 contains a multi I/O interface that provides a typical set of I/O functions, including A/D conversion, D/A conversion, digital I/O, PWM signal generation and measurement, and others.
<b>Communication interfaces</b>	<p>The DS2202 provides the following communication interfaces:</p> <ul style="list-style-type: none"> <li>▪ A standard serial interface (RS232, RS422 based on a Texas Instruments TL16C550 UART)</li> <li>▪ A CAN subsystem that is based on the STMicroelectronics ST10F269 microcontroller. It provides access to two CAN buses.</li> </ul>
<b>Voltage systems</b>	The DS2202 supports two independent voltage systems in the voltage range 5 V ... 60 V. The voltage must be supplied externally.
<b>Related topics</b>	<p>References</p> <div> <a href="#">Features of the DS2202</a> ..... 9         </div>

## Features of the DS2202

<b>Introduction</b>	The DS2202 provides the following features, summarized in alphabetical order:
<b>A/D conversion</b>	The ADC unit provides 16 unipolar A/D channels with 14-bit resolution and 20 µs conversion time for all channels. Refer to <a href="#">ADC Unit</a> on page 14.
<b>Bit I/O</b>	The bit I/O unit provides 38 discrete input channels and 16 discrete output channels. Refer to <a href="#">Bit I/O Unit</a> on page 18.
<b>CAN support</b>	The CAN support serves two CAN controllers that meet the CAN 2.0A (11-bit identifier) and CAN 2.0B (29-bit identifier) specifications. Refer to <a href="#">CAN Support</a> on page 43.
<b>D/A conversion</b>	The DAC unit provides 20 unipolar D/A output channels with 12-bit resolution and 20 µs full-scale settling time to 1 LSB. Refer to <a href="#">DAC Unit</a> on page 16.

<b>Frequency measurement</b>	For frequency measurement, 24 channels are available with a resolution of 16 bit. Refer to <a href="#">Frequency Measurement (F2D)</a> on page 30.
<b>Interrupt control</b>	The DS2202 provides interrupts for the serial interface and the CAN subsystem. Refer to <a href="#">Interrupt Handling and DS2202 Interrupts</a> on page 81.
<b>PWM signal generation</b>	9 output channels with run-time-adjustable frequencies and duty cycles are available for PWM signal generation. Refer to <a href="#">PWM Signal Generation</a> on page 25.
<b>PWM signal measurement</b>	24 input channels are available for PWM signal measurement. Refer to <a href="#">PWM Signal Measurement</a> on page 22.
<b>Serial interface</b>	The serial interface is based on the standard UART TL16C550C from Texas Instruments. The interface can be used in RS232 or RS422 mode. Refer to <a href="#">Serial Interface</a> on page 37.
<b>Square-wave signal generation</b>	9 output channels are available with a resolution of 16 bit for square-wave signal generation. Refer to <a href="#">Square-Wave Signal Generation (D2F)</a> on page 32.
	<div> <b>Note</b> <p>Some I/O features of the DS2202 conflict with other I/O features of the board. See <a href="#">Conflicting I/O Features</a> on page 86.</p> </div>
<b>Limitations</b>	There are some limitations when you work with the DS2202. See <a href="#">Limitations</a> on page 85.
<b>Related topics</b>	<b>References</b> <div> <a href="#">Functional Units of the DS2202</a>..... 7 </div>

## Integration of the DS2202 in a Modular System

<b>Introduction</b>	As an I/O board, the DS2202 is always part of a PHS-bus-based system. While the DS2202 measures and simulates the signals required, the processor board
---------------------	---

performs the calculation of the real-time model. That is, applications using DS2202 I/O features are implemented on the processor board.

---

**PHS bus**

Communication between the processor board and the I/O board is performed via the peripheral high-speed bus: That is the PHS++ bus, which is called “PHS bus” in the documentation.

**Partitioning the PHS bus with the DS802** With the DS802 PHS Link Board you can spatially partition the PHS bus by arranging the I/O boards in several expansion boxes.

The DS802 can be used in combination with many types of available dSPACE I/O boards. However, some I/O boards and some functionalities of specific I/O boards are not supported.

The I/O board support depends on the dSPACE software release which you use. For a list of supported I/O boards, refer to [DS802 Data Sheet \(PHS Bus System Hardware Reference !\[\]\(cbe2492b119e39e02a1dab2af4a4b296\_img.jpg\)](#)).



# Multi I/O Interface

## Where to go from here

## Information in this section

### [ADC Unit..... 14](#)

The ADC unit consists of a successive approximation register (SAR) A/D converter with a 16:1 input multiplexer that provides 16 input channels (ADC1 ... ADC16), with 14-bit resolution each, 20  $\mu$ s conversion time for all channels, and one integrated sample/hold for all channels.

### [DAC Unit..... 16](#)

The DAC unit provides 20 unipolar D/A output channels (DAC1 ... DAC20) with 12-bit resolution (fully monotonic) and 20  $\mu$ s full-scale settling time to 1 LSB.

### [Bit I/O Unit..... 18](#)

The bit I/O unit provides 38 discrete digital input channels, and 16 discrete digital output channels.

### [PWM Signal Measurement..... 22](#)

The DS2202 provides 24 independent PWM channels to analyze the duty cycle, frequency, and low and high periods of PWM signals. You can measure the duty cycle within 0 ... 100% for PWM frequencies in the range 0.01 Hz ... 100 kHz, at a resolution of 16 bit.

### [PWM Signal Generation..... 25](#)

The DS2202 provides 9 independent output channels for the generation of PWM signals.

### [Frequency Measurement \(F2D\)..... 30](#)

The DS2202 provides 24 independent input channels for the frequency measurement of square-wave signals.

### [Square-Wave Signal Generation \(D2F\)..... 32](#)

The DS2202 provides 9 independent output channels for the generation of square-wave signals with variable frequencies.

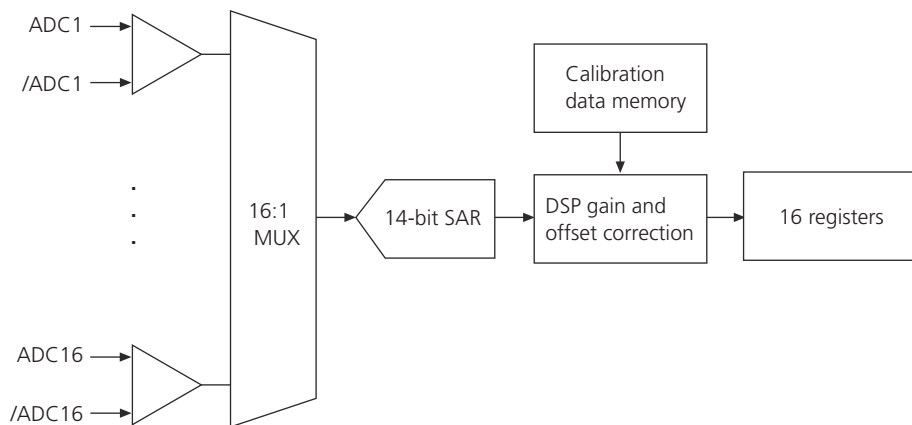
## ADC Unit

### Introduction

The ADC unit consists of a successive approximation register (SAR) A/D converter with a 16:1 input multiplexer that provides 16 input channels (ADC1 ... ADC16), with 14-bit resolution each, 20  $\mu$ s conversion time for all channels, and one integrated sample/hold for all channels. All channels are differential and can measure unipolar signals with 0 ... 60 V input span, lowpass filters (1<sup>st</sup> order/–3 dB at 80 kHz), 1 M $\Omega$  input impedance to system ground, and continuous  $\pm 75$  V DC overvoltage protection. The input channels can be read individually or blockwise. The control logic allows conversion of the first 4, 8, 12 or 16 channels (starting from channel 1).

### Block diagram

The illustration shows a simplified block diagram of the ADC unit.



**Differential input channels** ADC input channels are *differential input channels*, i.e., the ADC unit measures the voltage difference of  $(ADCx - \overline{ADCx})$ . The  $\overline{ADCx}$  channels function as individual ground sense lines for each input channel. They must be connected to the ground of your system near the sensor or to GND at the DS2202 connector, for all ADC channels used.

### RTI/RTLib support

You can use the ADC unit via RTI and RTLib. For details, refer to:

- RTI: [ADC Unit \(DS2202 RTI Reference !\[\]\(3168ddc4389f6b417dd71f084513be9c\_img.jpg\)](#))
- RTLib: [ADC Unit \(DS2202 RTLib Reference !\[\]\(17332056424eb04f01463711418ba65a\_img.jpg\)](#))

### Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2202 RTLib Reference !\[\]\(ab4e2b3fc7e7887b7a72f548aa6f5e60\_img.jpg\)](#)).

## I/O mapping

The following table shows the mapping of the A/D channels to the corresponding I/O pins of I/O connector P1, as used in RTI and RTLib.

A/D Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range
1	ADC1	P1	65	P1B	28	14-bit ADC	$(\text{ADC1} - \overline{\text{ADC1}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC1}}$	P1	67	P1B	12		
2	ADC2	P1	66	P1A	28	14-bit ADC	$(\text{ADC2} - \overline{\text{ADC2}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC2}}$	P1	68	P1A	12		
3	ADC3	P1	69	P1B	45	14-bit ADC	$(\text{ADC3} - \overline{\text{ADC3}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC3}}$	P1	71	P1B	29		
4	ADC4	P1	70	P1A	45	14-bit ADC	$(\text{ADC4} - \overline{\text{ADC4}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC4}}$	P1	72	P1A	29		
5	ADC5	P1	73	P1B	13	14-bit ADC	$(\text{ADC5} - \overline{\text{ADC5}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC5}}$	P1	75	P1B	46		
6	ADC6	P1	74	P1A	13	14-bit ADC	$(\text{ADC6} - \overline{\text{ADC6}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC6}}$	P1	76	P1A	46		
7	ADC7	P1	77	P1B	30	14-bit ADC	$(\text{ADC7} - \overline{\text{ADC7}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC7}}$	P1	79	P1B	14		
8	ADC8	P1	78	P1A	30	14-bit ADC	$(\text{ADC8} - \overline{\text{ADC8}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC8}}$	P1	80	P1A	14		
9	ADC9	P1	83	P1B	31	14-bit ADC	$(\text{ADC9} - \overline{\text{ADC9}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC9}}$	P1	85	P1B	15		
10	ADC10	P1	84	P1A	31	14-bit ADC	$(\text{ADC10} - \overline{\text{ADC10}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC10}}$	P1	86	P1A	15		
11	ADC11	P1	87	P1B	48	14-bit ADC	$(\text{ADC11} - \overline{\text{ADC11}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC11}}$	P1	89	P1B	32		
12	ADC12	P1	88	P1A	48	14-bit ADC	$(\text{ADC12} - \overline{\text{ADC12}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC12}}$	P1	90	P1A	32		
13	ADC13	P1	91	P1B	16	14-bit ADC	$(\text{ADC13} - \overline{\text{ADC13}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC13}}$	P1	93	P1B	49		
14	ADC14	P1	92	P1A	16	14-bit ADC	$(\text{ADC14} - \overline{\text{ADC14}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC14}}$	P1	94	P1A	49		
15	ADC15	P1	95	P1B	33	14-bit ADC	$(\text{ADC15} - \overline{\text{ADC15}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC15}}$	P1	97	P1B	17		
16	ADC16	P1	96	P1A	33	14-bit ADC	$(\text{ADC16} - \overline{\text{ADC16}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC16}}$	P1	98	P1A	17		

## Related topics

## Basics

[Introduction to the Features of the DS2202](#) ..... 7

## References

[ADC Unit \(DS2202 RTI Reference\)](#)

[Analog Inputs \(PHS Bus System Hardware Reference\)](#)

## DAC Unit

## Introduction

The DAC unit provides 20 unipolar D/A output channels (DAC1 ... DAC20) with 12-bit resolution (fully monotonic) and 20  $\mu$ s full-scale settling time to 1 LSB.

Latched output mode is not supported, which means that changes take effect immediately. On power-up and reset, the D/A output channels are reset to zero.

The DAC unit works with an internal reference of  $V_{REF} = 10$  V, but you can apply a reference voltage  $V_{REF} = (DAC\_REF - \overline{DAC\_REF})$  in the range 5 ... 10 V.

**Note**

D/A output channels are differential outputs in the range 0 ...  $V_{REF}$ . Each output has an individual ground sense line ( $\overline{DACx}$ ) that must be connected to the ground of your system near the actuator or to GND at the DS2202 connector, for all D/A output channels used. The DAC unit controls the voltage difference of  $(DACx - \overline{DACx})$ . If the external reference is used, the same applies to the  $\overline{DAC\_REF}$  pin.

If the  $\overline{DACx}$  pins are connected to a potential other than GND, the voltage between DACx and GND pins can swing in the range -10 V ... +12 V.

The output channels have lowpass filters (1<sup>st</sup> order/-3 dB at 100 kHz). The maximum sink/source current of the D/A output channels is  $\pm 5$  mA. They are protected against short circuit to GND or any voltage within  $\pm 60$  V.

## RTI/RTLib support

You can use the DAC unit via RTI and RTLib. For details, refer to:

- RTI: [DAC Unit \(DS2202 RTI Reference\)](#)
- RTLib: [DAC Unit \(DS2202 RTLib Reference\)](#)

## Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2202 RTLib Reference\)](#).



## I/O mapping

The following table shows the mapping of the D/A output channels to the corresponding I/O pins of I/O connectors P1 and P3, as used in RTI and RTLib.

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range/Output Current
-	DAC_REF	P1	33	P1B	39	DAC external reference voltage input	$V_{REF} = (DAC\_REF - \overline{DAC\_REF}) = 5 \dots 10 \text{ V}$
	$\overline{DAC\_REF}$	P1	35	P1B	23	DAC external reference voltage sense line	
1	DAC1	P1	37	P1B	7	12-bit DAC/20 $\mu\text{s}$	$(DAC1 - \overline{DAC1}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC1}$	P1	39	P1B	40		
2	DAC2	P1	38	P1A	7	12-bit DAC/20 $\mu\text{s}$	$(DAC2 - \overline{DAC2}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC2}$	P1	40	P1A	40		
3	DAC3	P1	41	P1B	24	12-bit DAC/20 $\mu\text{s}$	$(DAC3 - \overline{DAC3}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC3}$	P1	43	P1B	8		
4	DAC4	P1	42	P1A	24	12-bit DAC/20 $\mu\text{s}$	$(DAC4 - \overline{DAC4}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC4}$	P1	44	P1A	8		
5	DAC5	P1	45	P1B	41	12-bit DAC/20 $\mu\text{s}$	$(DAC5 - \overline{DAC5}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC5}$	P1	47	P1B	25		
6	DAC6	P1	46	P1A	41	12-bit DAC/20 $\mu\text{s}$	$(DAC6 - \overline{DAC6}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC6}$	P1	48	P1A	25		
7	DAC7	P1	51	P1B	42	12-bit DAC/20 $\mu\text{s}$	$(DAC7 - \overline{DAC7}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC7}$	P1	53	P1B	26		
8	DAC8	P1	52	P1A	42	12-bit DAC/20 $\mu\text{s}$	$(DAC8 - \overline{DAC8}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC8}$	P1	54	P1A	26		
9	DAC9	P1	55	P1B	10	12-bit DAC/20 $\mu\text{s}$	$(DAC9 - \overline{DAC9}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC9}$	P1	57	P1B	43		
10	DAC10	P1	56	P1A	10	12-bit DAC/20 $\mu\text{s}$	$(DAC10 - \overline{DAC10}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC10}$	P1	58	P1A	43		
11	DAC11	P1	59	P1B	27	12-bit DAC/20 $\mu\text{s}$	$(DAC11 - \overline{DAC11}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC11}$	P1	61	P1B	11		
12	DAC12	P1	60	P1A	27	12-bit DAC/20 $\mu\text{s}$	$(DAC12 - \overline{DAC12}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC12}$	P1	62	P1A	11		
13	DAC13	P3	7	-	-	12-bit DAC/20 $\mu\text{s}$	$(DAC13 - \overline{DAC13}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC13}$	P3	40	-	-		
14	DAC14	P3	24	-	-	12-bit DAC/20 $\mu\text{s}$	$(DAC14 - \overline{DAC14}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC14}$	P3	8	-	-		
15	DAC15	P3	41	-	-	12-bit DAC/20 $\mu\text{s}$	$(DAC15 - \overline{DAC15}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC15}$	P3	25	-	-		
16	DAC16	P3	9	-	-	12-bit DAC/20 $\mu\text{s}$	$(DAC16 - \overline{DAC16}) = 0 \dots V_{REF}; \pm 5 \text{ mA}$
	$\overline{DAC16}$	P3	42	-	-		

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range/Output Current
17	DAC17	P3	26	–	–	12-bit DAC/20 $\mu$ s	$(\text{DAC17} - \overline{\text{DAC17}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC17}}$	P3	10	–	–		
18	DAC18	P3	43	–	–	12-bit DAC/20 $\mu$ s	$(\text{DAC18} - \overline{\text{DAC18}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC18}}$	P3	27	–	–		
19	DAC19	P3	11	–	–	12-bit DAC/20 $\mu$ s	$(\text{DAC19} - \overline{\text{DAC19}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC19}}$	P3	44	–	–		
20	DAC20	P3	28	–	–	12-bit DAC/20 $\mu$ s	$(\text{DAC20} - \overline{\text{DAC20}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC20}}$	P3	12	–	–		

## Related topics

### Basics

[Introduction to the Features of the DS2202..... 7](#)

### References

[Analog Outputs \(DAC\) \(PHS Bus System Hardware Reference !\[\]\(a870788d6ed9b8fd294b7654a8c8526b\_img.jpg\)\)](#)  
[DAC Unit \(DS2202 RTI Reference !\[\]\(18065afa4ef6662bca9f3f6088f7de30\_img.jpg\)\)](#)  
[Power Supply Outputs \(PHS Bus System Hardware Reference !\[\]\(b985170eefb48b9b3ef593e79310e8f5\_img.jpg\)\)](#)

## Bit I/O Unit

### Introduction

The bit I/O unit provides 38 discrete digital input channels, and 16 discrete digital output channels.

### Digital input channels

The input channels are 12/42 V compatible (fully operational up to 60 V). After software initialization, the input threshold is set to 2.5 V. You can set the input threshold in the range 1.0 ... 23.8 V via RTI/RTLib. The input channels have a hysteresis voltage of 0.2 V. This value is fixed and cannot be changed. For example, if the input threshold is 2.5 V, the input signal must exceed 2.6 V to be detected as high, and must fall below 2.4 V to be detected as low.

### Digital output channels

The output channels have push-pull drivers running from an external source (two independent VBAT rails, each of which can be used in the range of 5 ... 60 V). The maximum output current per channel is  $\pm 50 \text{ mA}$ . Self-resetting multifuses protect against short circuits to GND and VBAT or any voltage between GND and + 60 V.

**Note**

Multifuses are designed for occasional faults only. They are therefore not suitable for failure simulation.

The output channels are in high impedance state after reset and power-up.

You can enable or disable each output channel individually, via RTI/RTLib. A disabled output channel is in high impedance state.

You can set the supply rail for each output channel individually (low side, high side VBAT1, or high side VBAT2) using RTI/RTLib.

**Note**

- Before operating the digital output channels, you must connect an external power supply ( $V_{\text{Bat}}$ ) to at least one of the two VBAT supply rails.
- To ensure push-pull driver functionality, the low side switch (LOW) must be set.

**RTI/RTLib support**

You can use the bit I/O unit via RTI and RTLib. For details, refer to:

- RTI: [Bit I/O Unit \(DS2202 RTI Reference !\[\]\(d27edc55493507da2f9b8c7a52b3b96f\_img.jpg\)](#))
- RTLib: [Bit I/O Unit \(DS2202 RTLib Reference !\[\]\(9bf7a72a60a57323fa980b9b0338593f\_img.jpg\)](#))

**Execution times**

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2202 RTLib Reference !\[\]\(0d5ec72f61334709c3fc9450209b754f\_img.jpg\)](#)).

**I/O mapping**

The following table shows the mapping of the channel numbers to the corresponding I/O pins of I/O connectors P1 and P2, as used in RTI and RTLib. Some I/O features of the DS2202 conflict with each other. For details, see [Conflicting I/O Features](#) on page 86.

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range
Digital Input Channels							
1	DIG_IN1 (PWM_IN9)	P1	3	P1B	34	Digital input (shared with PWM signal measurement)	12/42 V compatible
2	DIG_IN2 (PWM_IN10)	P1	4	P1A	34	Digital input (shared with PWM signal measurement)	12/42 V compatible
3	DIG_IN3 (PWM_IN11)	P1	5	P1B	18	Digital input (shared with PWM signal measurement)	12/42 V compatible
4	DIG_IN4 (PWM_IN12)	P1	6	P1A	18	Digital input (shared with PWM signal measurement)	12/42 V compatible
5	DIG_IN5 (PWM_IN13)	P1	7	P1B	2	Digital input (shared with PWM signal measurement)	12/42 V compatible

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range
6	DIG_IN6 (PWM_IN14)	P1	8	P1A	2	Digital input (shared with PWM signal measurement)	12/42 V compatible
7	DIG_IN7 (PWM_IN15)	P1	9	P1B	35	Digital input (shared with PWM signal measurement)	12/42 V compatible
8	DIG_IN8 (PWM_IN16)	P1	10	P1A	35	Digital input (shared with PWM signal measurement)	12/42 V compatible
9	DIG_IN9 (PWM_IN17)	P1	11	P1B	19	Digital input (shared with PWM signal measurement)	12/42 V compatible
10	DIG_IN10 (PWM_IN18)	P1	12	P1A	19	Digital input (shared with PWM signal measurement)	12/42 V compatible
11	DIG_IN11 (PWM_IN19)	P1	13	P1B	3	Digital input (shared with PWM signal measurement)	12/42 V compatible
12	DIG_IN12 (PWM_IN20)	P1	14	P1A	3	Digital input (shared with PWM signal measurement)	12/42 V compatible
13	DIG_IN13 (PWM_IN21)	P1	15	P1B	36	Digital input (shared with PWM signal measurement)	12/42 V compatible
14	DIG_IN14 (PWM_IN22)	P1	16	P1A	36	Digital input (shared with PWM signal measurement)	12/42 V compatible
15	DIG_IN15 (PWM_IN23)	P1	17	P1B	20	Digital input (shared with PWM signal measurement)	12/42 V compatible
16	DIG_IN16 (PWM_IN24)	P1	18	P1A	20	Digital input (shared with PWM signal measurement)	12/42 V compatible
17	DIG_IN17 (PWM_IN1)	P1	21	P1B	37	Digital input (shared with PWM signal measurement)	12/42 V compatible
18	DIG_IN18 (PWM_IN2)	P1	22	P1A	37	Digital input (shared with PWM signal measurement)	12/42 V compatible
19	DIG_IN19 (PWM_IN3)	P1	23	P1B	21	Digital input (shared with PWM signal measurement)	12/42 V compatible
20	DIG_IN20 (PWM_IN4)	P1	24	P1A	21	Digital input (shared with PWM signal measurement)	12/42 V compatible
21	DIG_IN21 (PWM_IN5)	P1	25	P1B	5	Digital input (shared with PWM signal measurement)	12/42 V compatible
22	DIG_IN22 (PWM_IN6)	P1	26	P1A	5	Digital input (shared with PWM signal measurement)	12/42 V compatible
23	DIG_IN23 (PWM_IN7)	P1	27	P1B	38	Digital input (shared with PWM signal measurement)	12/42 V compatible
24	DIG_IN24 (PWM_IN8)	P1	28	P1A	38	Digital input (shared with PWM signal measurement)	12/42 V compatible
25	DIG_IN25	P2	19	P2B	4	Digital input	12/42 V compatible
26	DIG_IN26	P2	20	P2A	4	Digital input	12/42 V compatible
27	DIG_IN27	P2	23	P2B	21	Digital input	12/42 V compatible
28	DIG_IN28	P2	24	P2A	21	Digital input	12/42 V compatible
29	DIG_IN29	P2	27	P2B	38	Digital input	12/42 V compatible
30	DIG_IN30	P2	28	P2A	38	Digital input	12/42 V compatible
31	DIG_IN31	P1	34	P1B	39	Digital input	12/42 V compatible

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range
32	DIG_IN32	P1	36	P1A	23	Digital input	12/42 V compatible
33	DIG_IN33	P2	17	P2B	20	Digital input	12/42 V compatible
34	DIG_IN34	P2	18	P2A	20	Digital input	12/42 V compatible
35	DIG_IN35	P2	21	P2B	37	Digital input	12/42 V compatible
36	DIG_IN36	P2	22	P2A	37	Digital input	12/42 V compatible
37	DIG_IN37	P2	25	P2B	5	Digital input	12/42 V compatible
38	DIG_IN38	P2	26	P2A	5	Digital input	12/42 V compatible
<b>Digital Output Channels</b>							
1	DIG_OUT1	P2	32	P2A	6	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
2	DIG_OUT2	P2	34	P2A	39	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
3	DIG_OUT3	P2	36	P2A	23	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
4	DIG_OUT4	P2	38	P2A	7	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
5	DIG_OUT5	P2	40	P2A	40	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
6	DIG_OUT6	P2	42	P2A	24	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
7	DIG_OUT7	P2	44	P2A	8	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
8	DIG_OUT8	P2	46	P2A	41	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
9	DIG_OUT9	P2	50	P2A	9	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
10	DIG_OUT10	P2	52	P2A	42	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
11	DIG_OUT11	P2	54	P2A	26	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
12	DIG_OUT12	P2	56	P2A	10	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
13	DIG_OUT13	P2	58	P2A	43	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
14	DIG_OUT14	P2	60	P2A	27	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
15	DIG_OUT15	P2	62	P2A	11	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
16	DIG_OUT16	P2	64	P2A	44	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA

## Related topics

### Basics

[Introduction to the Features of the DS2202.....7](#)

### References

[Bit I/O Unit \(DS2202 RTI Reference !\[\]\(0aff635c4179ba9e710b00f4b01d3b20\_img.jpg\)\)](#)  
[Digital I/O Set Up \(DS2202 RTI Reference !\[\]\(29658d981ebdf5edc259074cbf6110e0\_img.jpg\)\)](#)  
[Digital Inputs \(PHS Bus System Hardware Reference !\[\]\(9b3d169a802e50e3425ebff869ff6250\_img.jpg\)\)](#)  
[Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(510c3e621c59b50959bed0883f15fd7c\_img.jpg\)\)](#)

## PWM Signal Measurement

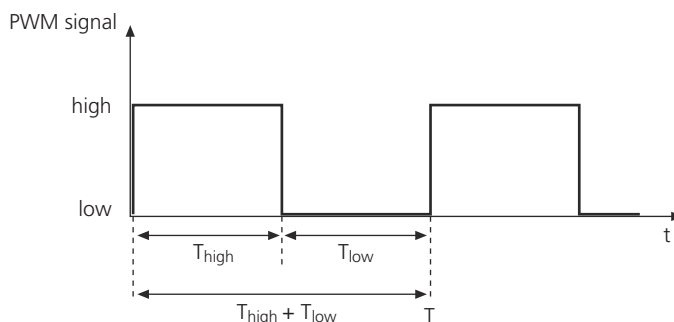
### Introduction

The DS2202 provides 24 independent PWM channels to analyze the duty cycle, frequency, and low and high periods of PWM signals.

You can measure the duty cycle within 0 ... 100% for PWM frequencies in the range 0.01 Hz ... 100 kHz, at a resolution of 16 bit.

### Measuring the frequency and duty cycle of a PWM signal

The DS2202 measures the values of the  $T_{low}$  and  $T_{high}$  periods of a PWM signal. See the following illustration:



The DS2202 uses the values of  $T_{low}$  and  $T_{high}$  to calculate the frequency and the duty cycle:

- Frequency =  $1 / (T_{low} + T_{high})$
- Duty cycle =  $T_{high} / (T_{low} + T_{high})$

### Measurement range

Depending on the measurement range, the periods can be measured within the following limits and resolutions:

Range	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
1	200 ns	10 $\mu$ s	3.27 ms	50 ns
2	400 ns	10 $\mu$ s	6.55 ms	100 ns
3	800 ns	10 $\mu$ s	13.1 ms	200 ns
4	1.6 $\mu$ s	10 $\mu$ s	26.2 ms	400 ns
5	3.2 $\mu$ s	10 $\mu$ s	52.4 ms	800 ns
6	6.4 $\mu$ s	10 $\mu$ s	105 ms	1.6 $\mu$ s
7	12.8 $\mu$ s	12.8 $\mu$ s	210 ms	3.2 $\mu$ s
8	25.6 $\mu$ s	25.6 $\mu$ s	419 ms	6.4 $\mu$ s
9	51.2 $\mu$ s	51.2 $\mu$ s	839 ms	12.8 $\mu$ s
10	102 $\mu$ s	102 $\mu$ s	1.68 s	25.6 $\mu$ s
11	205 $\mu$ s	205 $\mu$ s	3.36 s	51.2 $\mu$ s
12	410 $\mu$ s	410 $\mu$ s	6.71 s	103 $\mu$ s
13	819 $\mu$ s	819 $\mu$ s	13.4 s	205 $\mu$ s

Range	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
14	1.64 ms	1.64 ms	26.8 s	410 $\mu$ s
15	3.28 ms	3.28 ms	53.6 s	820 $\mu$ s
16	6.55 ms	6.55 ms	107.3 s	1.64 ms

**Note**

For optimum measurements, you have to select the expected period range of the PWM signal to be measured.

Due to quantization effects, you will encounter considerable deviations between the input PWM period  $T_P$  and the measured PWM period, especially for higher PWM frequencies. To avoid poor frequency resolution, you should therefore select the frequency range with the best possible resolution (resolution values as small as possible). For details, refer to [Quantization Effects](#) on page 85.

The maximum period applies to  $0\% < \text{duty cycle} < 100\%$ .

If these ranges are exceeded, the measurement will be faulty. For values outside the practical period range, some restrictions apply:

Range	Restriction
PWM period < theoretical minimum period	No precise measurement (undersampling). Some high or low periods may not be recognized.
PWM period < 10 $\mu$ s, $T_{\text{high}}$ or $T_{\text{low}}$ < 3 $\mu$ s	Pulses that you want to measure may be missing.
Max. period < PWM period < 2 · max. period	PWM signal measurement with restricted duty cycle.
PWM period > 2 · maximum period	PWM signal measurement with duty cycle alternating between "0" and "1".

The duty cycle values 0 (input constant low) and 1 (input constant high) are measured properly.

**Note****Signal periods and resolution**

Each high period and each low period of the measured signal must be longer (not equal) than the resolution to avoid missing pulses.

**Update mode**

The DS2202 supports two update modes that describe the time interval when the measured values are updated:

**Asynchronous mode** The measured values are updated at the end of each  $T_{\text{high}}$  and  $T_{\text{low}}$  period of the PWM signal. The update is asynchronous to the period.

**Synchronous mode** The measured values are updated at the end of each  $T_{\text{low}}$  period of the PWM signal only. The update is synchronous to the period.

**Input voltage**

The input channels are 12/42 V compatible (fully operational up to 60 V). After software initialization, the input threshold is set to 2.5 V. You can set the input threshold in the range 1 V... 23.8 V via RTI/RTLib. The inputs have a hysteresis voltage of 0.2 V.

**RTI/RTLib support**

You can measure PWM signals via RTI and RTLib. For details, refer to:

- RTI: [PWM Signal Measurement \(DS2202 RTI Reference !\[\]\(448bd415caa8b52d2aeb4d58499267b2\_img.jpg\)](#))
- RTLib: [PWM Signal Measurement \(DS2202 RTLib Reference !\[\]\(23be4c52910c50d5908bb101588c4f4e\_img.jpg\)](#))

**Execution times**

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2202 RTLib Reference !\[\]\(17acf1afa8cdf0b67c53d4865a5ed469\_img.jpg\)](#)).

**I/O mapping**

The following table shows the mapping of the PWM channels to the corresponding I/O pins, as used in RTI and RTLib.

**Note**

Some DS2202 I/O features conflict with each other, see [Conflicting I/O Features](#) on page 86.




Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage
1	PWM_IN1 (DIG_IN17)	P1	21	P1B	37	PWM signal measurement (shared with digital input)	12/42 V compatible
2	PWM_IN2 (DIG_IN18)	P1	22	P1A	37	PWM signal measurement (shared with digital input)	12/42 V compatible
3	PWM_IN3 (DIG_IN19)	P1	23	P1B	21	PWM signal measurement (shared with digital input)	12/42 V compatible
4	PWM_IN4 (DIG_IN20)	P1	24	P1A	21	PWM signal measurement (shared with digital input)	12/42 V compatible
5	PWM_IN5 (DIG_IN21)	P1	25	P1B	5	PWM signal measurement (shared with digital input)	12/42 V compatible
6	PWM_IN6 (DIG_IN22)	P1	26	P1A	5	PWM signal measurement (shared with digital input)	12/42 V compatible
7	PWM_IN7 (DIG_IN23)	P1	27	P1B	38	PWM signal measurement (shared with digital input)	12/42 V compatible
8	PWM_IN8 (DIG_IN24)	P1	28	P1A	38	PWM signal measurement (shared with digital input)	12/42 V compatible
9	PWM_IN9 (DIG_IN1)	P1	3	P1B	34	PWM signal measurement (shared with digital input)	12/42 V compatible
10	PWM_IN10 (DIG_IN2)	P1	4	P1A	34	PWM signal measurement (shared with digital input)	12/42 V compatible
11	PWM_IN11 (DIG_IN3)	P1	5	P1B	18	PWM signal measurement (shared with digital input)	12/42 V compatible



Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage
12	PWM_IN12 (DIG_IN4)	P1	6	P1A	18	PWM signal measurement (shared with digital input)	12/42 V compatible
13	PWM_IN13 (DIG_IN5)	P1	7	P1B	2	PWM signal measurement (shared with digital input)	12/42 V compatible
14	PWM_IN14 (DIG_IN6)	P1	8	P1A	2	PWM signal measurement (shared with digital input)	12/42 V compatible
15	PWM_IN15 (DIG_IN7)	P1	9	P1B	35	PWM signal measurement (shared with digital input)	12/42 V compatible
16	PWM_IN16 (DIG_IN8)	P1	10	P1A	35	PWM signal measurement (shared with digital input)	12/42 V compatible
17	PWM_IN17 (DIG_IN9)	P1	11	P1B	19	PWM signal measurement (shared with digital input)	12/42 V compatible
18	PWM_IN18 (DIG_IN10)	P1	12	P1A	19	PWM signal measurement (shared with digital input)	12/42 V compatible
19	PWM_IN19 (DIG_IN11)	P1	13	P1B	3	PWM signal measurement (shared with digital input)	12/42 V compatible
20	PWM_IN20 (DIG_IN12)	P1	14	P1A	3	PWM signal measurement (shared with digital input)	12/42 V compatible
21	PWM_IN21 (DIG_IN13)	P1	15	P1B	36	PWM signal measurement (shared with digital input)	12/42 V compatible
22	PWM_IN22 (DIG_IN14)	P1	16	P1A	36	PWM signal measurement (shared with digital input)	12/42 V compatible
23	PWM_IN23 (DIG_IN15)	P1	17	P1B	20	PWM signal measurement (shared with digital input)	12/42 V compatible
24	PWM_IN24 (DIG_IN16)	P1	18	P1A	20	PWM signal measurement (shared with digital input)	12/42 V compatible

## Related topics

## References

<a href="#">Digital I/O Set Up (DS2202 RTI Reference </a> )	
<a href="#">Digital Inputs (PHS Bus System Hardware Reference </a> )	
<a href="#">PWM Signal Generation.....</a>	<a href="#">25</a>
<a href="#">PWM Signal Measurement (DS2202 RTI Reference </a> )	

# PWM Signal Generation

## Introduction

The DS2202 provides 9 independent output channels for the generation of PWM signals. PWM signals are square-wave signals with run-time adjustable frequency and duty cycle.

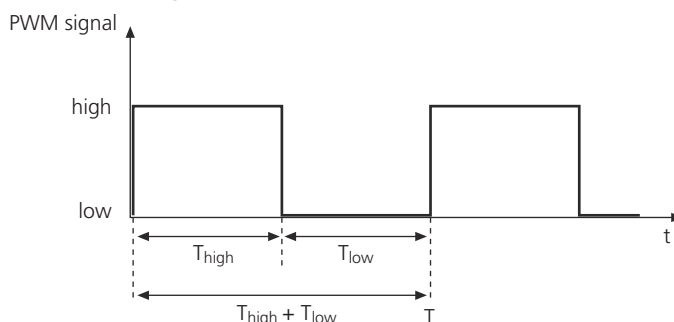
## Generating a PWM signal

To generate a PWM signal, you can specify the signal's duty cycle in the range 0 ... 100% for PWM frequencies in the range 0.01 Hz ... 100 kHz. The duty cycle values 0 and 1 result in a constant low or constant high output signal.

The duty cycle of a PWM signal is defined like this:

$$\text{Duty cycle} = T_{\text{high}} / (T_{\text{low}} + T_{\text{high}})$$

See the following illustration:



**Asymmetric PWM pulses** The pulses of the PWM signals generated by the DS2202 start at the beginning of the corresponding PWM period (asymmetric PWM pulses). With the DS2202, you cannot generate PWM pulses that are centered around the middle of the corresponding PWM period (symmetric PWM pulses).

**PWM output signals not synchronous** If you use several output channels of the DS2202 for PWM signal generation, the generated PWM signals are not synchronous.

## Limits and resolution

Depending on the selected range, the DS2202 can generate PWM signals within the following limits and resolutions:

Range	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
1	200 ns	10 µs	3.27 ms	50 ns
2	400 ns	10 µs	6.55 ms	100 ns
3	800 ns	10 µs	13.1 ms	200 ns
4	1.6 µs	10 µs	26.2 ms	400 ns
5	3.2 µs	10 µs	52.4 ms	800 ns
6	6.4 µs	10 µs	105 ms	1.6 µs
7	12.8 µs	12.8 µs	210 ms	3.2 µs
8	25.6 µs	25.6 µs	419 ms	6.4 µs
9	51.2 µs	51.2 µs	839 ms	12.8 µs
10	102 µs	102 µs	1.68 s	25.6 µs
11	205 µs	205 µs	3.36 s	51.2 µs
12	410 µs	410 µs	6.71 s	103 µs
13	819 µs	819 µs	13.4 s	205 µs

Range	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
14	1.64 ms	1.64 ms	26.8 s	410 $\mu$ s
15	3.28 ms	3.28 ms	53.6 s	820 $\mu$ s
16	6.55 ms	6.55 ms	107.3 s	1.64 ms

**Note**

Due to quantization effects, you may encounter considerable deviations between the desired PWM period TP and the generated PWM period, especially for higher PWM frequencies. To avoid poor frequency resolution, you should therefore select the frequency range with the best possible resolution (resolution values as small as possible). For details, refer to [Quantization Effects](#) on page 85.

The maximum period applies for generating signals with a 0 ... 100% duty cycle. If these ranges are exceeded, PWM signal generation will be faulty. For values outside these ranges, note the following restrictions:

Range	Restriction
PWM period < theoretical minimum period	No PWM signal generation. Signal is constantly high or low.
Theoretical min. period < PWM period < 10 $\mu$ s $T_{\text{high}}$ or $T_{\text{low}}$ < 3 $\mu$ s	PWM signal will be distorted or pulses lost due to limited switching speed of the output circuits.
Max. period < PWM period < 2 · max. period	PWM signal with restricted duty cycle

**Output voltage**

The output channels have push-pull drivers running from an external source (two independent VBAT rails, each of which can operate in the range of 5 ... 60 V). The maximum output current per channel is  $\pm 50$  mA. Self-resetting multifuses protect against short circuits to GND and VBAT or any voltage between GND and + 60 V.

**Note**

- Before operating the digital output channels, you must connect an external power supply ( $V_{\text{Bat}}$ ) to at least one of the two VBAT supply rails.
- Multifuses are designed for occasional faults only. They are therefore not suitable for failure simulation.

The output channels are in high impedance state after reset and power-up.

You can enable or disable each output channel individually, using RTI/RTLlib. A disabled output is in high impedance state.

You can set the supply rail for each output individually using RTI/RTLib (low side, high side VBAT1, or high side VBAT2).

**Note**

To ensure push-pull driver functionality, the low side switch (LOW) must be set.

---

**Update mode**

The DS2202 supports two update modes for you to specify when new values for the duty cycle are set.

**Asynchronous mode** The new values are updated immediately. An update can happen anywhere during the PWM period.

**Note**

For PWM signal generation with *asynchronous* update, a high or low pulse is cut off when the new  $T_{high}$  or  $T_{low}$  value:

- Is shorter than the current one  
and
- Exceeds the time which has elapsed in the current  $T_{high}$  or  $T_{low}$  period, respectively.

As a result, the PWM period is not constant during update (i.e., actual  $T_{high} + T_{low}$ ). If this is not desired, select the *synchronous* update mode instead.

**Synchronous mode** Update of the input parameters is performed synchronously to the period. Changes take effect in the next period (at each rising edge).

**Note**

For PWM signal generation with *synchronous* update, the output period should be constant. It is constant if  $T = T_{high} + T_{low}$  is constant. If you change the period during run time, synchronous PWM update cannot be ensured.

---

**RTI/RTLib support**

You can generate PWM signals via RTI and RTLib. For details, refer to:

- RTI: [PWM Signal Generation \(DS2202 RTI Reference !\[\]\(3292f5442e3b4027aa0bb60988f9fc82\_img.jpg\)](#))
- RTLib: [PWM Signal Generation \(DS2202 RTLib Reference !\[\]\(705a9285b25462d2e675759828e0d2ed\_img.jpg\)](#))

---

**Execution times**

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2202 RTLib Reference !\[\]\(cbd8541a32dfc32f356f5c6c994b0a21\_img.jpg\)](#)).

**I/O mapping**

The following table shows the mapping of the channel numbers to the corresponding I/O pins of I/O connectors P2 and P3, as used in RTI and RTLib.

**Note**

Some I/O features of the DS2202 conflict with each other. For details, see [Conflicting I/O Features](#) on page 86.

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage
1	PWM_OUT1	P2	31	P2B	6	PWM signal generation (shared with square-wave signal generation)	0.4 V ... (VBAT(x) – 1.2 V); $\pm 50$ mA
2	PWM_OUT2	P2	33	P2B	39	PWM signal generation (shared with square-wave signal generation)	0.4 V ... (VBAT(x) – 1.2 V); $\pm 50$ mA
3	PWM_OUT3	P2	35	P2B	23	PWM signal generation (shared with square-wave signal generation)	0.4 V ... (VBAT(x) – 1.2 V); $\pm 50$ mA
4	PWM_OUT4	P2	37	P2B	7	PWM signal generation (shared with square-wave signal generation)	0.4 V ... (VBAT(x) – 1.2 V); $\pm 50$ mA
5	PWM_OUT5	P2	39	P2B	40	PWM signal generation (shared with square-wave signal generation)	0.4 V ... (VBAT(x) – 1.2 V); $\pm 50$ mA
6	PWM_OUT6	P2	41	P2B	24	PWM signal generation (shared with square-wave signal generation)	0.4 V ... (VBAT(x) – 1.2 V); $\pm 50$ mA
7	PWM_OUT7	P3	16	–	–	PWM signal generation (shared with square-wave signal generation)	0.4 V ... (VBAT(x) – 1.2 V); $\pm 50$ mA
8	PWM_OUT8	P3	49	–	–	PWM signal generation (shared with square-wave signal generation)	0.4 V ... (VBAT(x) – 1.2 V); $\pm 50$ mA
9	PWM_OUT9	P3	33	–	–	PWM signal generation (shared with square-wave signal generation)	0.4 V ... (VBAT(x) – 1.2 V); $\pm 50$ mA

**Related topics****References**

[Digital I/O Set Up \(DS2202 RTI Reference !\[\]\(0b5e7e25e8775f7e7e80906ada4f0021\_img.jpg\)](#))  
[Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(740312fd467f47b04cab841ab3868d83\_img.jpg\)](#))  
[PWM Signal Generation \(DS2202 RTI Reference !\[\]\(dbb8da2687e90ededffd3484b6b666cf\_img.jpg\)](#))  
[PWM Signal Measurement](#)..... 22

## Frequency Measurement (F2D)

### Introduction

The DS2202 provides 24 independent input channels for the frequency measurement of square-wave signals.

You can measure frequencies in the range 0.3 mHz ... 100 kHz.

### Frequency ranges

To get the best resolution of the measured square-wave signal, you should always use the frequency range with the lowest possible range number. You can measure frequencies in the following ranges:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	100 kHz	50 ns
2	4.77 Hz	100 kHz	100 ns
3	2.39 Hz	100 kHz	200 ns
4	1.20 Hz	100 kHz	400 ns
5	0.60 Hz	100 kHz	800 ns
6	0.30 Hz	100 kHz	1.6 $\mu$ s
7	0.15 Hz	100 kHz	3.2 $\mu$ s
8	75 mHz	78.12 kHz	6.4 $\mu$ s
9	38 mHz	39.06 kHz	12.8 $\mu$ s
10	19 mHz	19.53 kHz	25.6 $\mu$ s
11	10 mHz	9.76 kHz	51.2 $\mu$ s
12	5.0 mHz	4.88 kHz	103 $\mu$ s
13	2.5 mHz	2.44 kHz	205 $\mu$ s
14	1.2 mHz	1.22 kHz	410 $\mu$ s
15	0.6 mHz	610.35 Hz	820 $\mu$ s
16	0.3 mHz	305.17 Hz	1.64 ms

If these ranges are exceeded, the measurement will be faulty. For values outside the frequency ranges, note the following restrictions:

Range	Restriction
Frequency < $f_{\min}$	The measurement returns a frequency of 0 Hz.
Frequency > $f_{\max}$	Faulty measurement because of quantization problems, refer to <a href="#">Quantization Effects</a> on page 85.

#### Note

##### Signal periods and resolution

Each high period and each low period of the measured signal must be longer (not equal) than the resolution to avoid missing pulses.

**Input voltage** The input channels are 12/42 V compatible (fully operational up to 60 V). The default input threshold value is 2.5 V. You can set the input threshold for all digital input channels in the range 1 ... 23.8 V using RTI/RTLib. The inputs have a hysteresis voltage of 0.2 V.

**RTI/RTLib support** You can measure square-wave signal frequencies via RTI and RTLib. For details, refer to:

- RTI: [Frequency Measurement \(DS2202 RTI Reference !\[\]\(d84e7ea36f695d92cb39ec32c307ac93\_img.jpg\)](#))
- RTLib: [Frequency Measurement \(DS2202 RTLib Reference !\[\]\(db9b0c6fa4ac1078c53d7f74438ad75d\_img.jpg\)](#))

**Execution times** The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2202 RTLib Reference !\[\]\(feabb98897b440bc8695a03336a6e2df\_img.jpg\)](#)).

**I/O mapping** The following table shows the mapping of the channel numbers to the corresponding I/O pins of I/O connector P1, as used in RTI and RTLib.

**Note**

Some I/O features of the DS2202 conflict with each other. For details, see [Conflicting I/O Features](#) on page 86.

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage
1	PWM_IN1 (DIG_IN17)	P1	21	P1B	37	Frequency measurement (shared with digital input)	12/42 V compatible
2	PWM_IN2 (DIG_IN18)	P1	22	P1A	37	Frequency measurement (shared with digital input)	12/42 V compatible
3	PWM_IN3 (DIG_IN19)	P1	23	P1B	21	Frequency measurement (shared with digital input)	12/42 V compatible
4	PWM_IN4 (DIG_IN20)	P1	24	P1A	21	Frequency measurement (shared with digital input)	12/42 V compatible
5	PWM_IN5 (DIG_IN21)	P1	25	P1B	5	Frequency measurement (shared with digital input)	12/42 V compatible
6	PWM_IN6 (DIG_IN22)	P1	26	P1A	5	Frequency measurement (shared with digital input)	12/42 V compatible
7	PWM_IN7 (DIG_IN23)	P1	27	P1B	38	Frequency measurement (shared with digital input)	12/42 V compatible
8	PWM_IN8 (DIG_IN24)	P1	28	P1A	38	Frequency measurement (shared with digital input)	12/42 V compatible
9	PWM_IN9 (DIG_IN1)	P1	3	P1B	34	Frequency measurement (shared with digital input)	12/42 V compatible
10	PWM_IN10 (DIG_IN2)	P1	4	P1A	34	Frequency measurement (shared with digital input)	12/42 V compatible

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage
11	PWM_IN11 (DIG_IN3)	P1	5	P1B	18	Frequency measurement (shared with digital input)	12/42 V compatible
12	PWM_IN12 (DIG_IN4)	P1	6	P1A	18	Frequency measurement (shared with digital input)	12/42 V compatible
13	PWM_IN13 (DIG_IN5)	P1	7	P1B	2	Frequency measurement (shared with digital input)	12/42 V compatible
14	PWM_IN14 (DIG_IN6)	P1	8	P1A	2	Frequency measurement (shared with digital input)	12/42 V compatible
15	PWM_IN15 (DIG_IN7)	P1	9	P1B	35	Frequency measurement (shared with digital input)	12/42 V compatible
16	PWM_IN16 (DIG_IN8)	P1	10	P1A	35	Frequency measurement (shared with digital input)	12/42 V compatible
17	PWM_IN17 (DIG_IN9)	P1	11	P1B	19	Frequency measurement (shared with digital input)	12/42 V compatible
18	PWM_IN18 (DIG_IN10)	P1	12	P1A	19	Frequency measurement (shared with digital input)	12/42 V compatible
19	PWM_IN19 (DIG_IN11)	P1	13	P1B	3	Frequency measurement (shared with digital input)	12/42 V compatible
20	PWM_IN20 (DIG_IN12)	P1	14	P1A	3	Frequency measurement (shared with digital input)	12/42 V compatible
21	PWM_IN21 (DIG_IN13)	P1	15	P1B	36	Frequency measurement (shared with digital input)	12/42 V compatible
22	PWM_IN22 (DIG_IN14)	P1	16	P1A	36	Frequency measurement (shared with digital input)	12/42 V compatible
23	PWM_IN23 (DIG_IN15)	P1	17	P1B	20	Frequency measurement (shared with digital input)	12/42 V compatible
24	PWM_IN24 (DIG_IN16)	P1	18	P1A	20	Frequency measurement (shared with digital input)	12/42 V compatible

## Related topics

## References

[Digital I/O Set Up \(DS2202 RTI Reference !\[\]\(6605b201d6f14d9b3bcb8ab5f274d107\_img.jpg\)\)](#)  
[Digital Inputs \(PHS Bus System Hardware Reference !\[\]\(f4056bb2e5acf0a782fb9d812dad489d\_img.jpg\)\)](#)  
[Frequency Measurement \(DS2202 RTI Reference !\[\]\(23e633620764e54910cf7e601ab387fc\_img.jpg\)\)](#)  
[Square-Wave Signal Generation \(D2F\).....32](#)

# Square-Wave Signal Generation (D2F)

## Introduction

The DS2202 provides 9 independent output channels for the generation of square-wave signals with variable frequencies.

You can generate frequencies in the range 0.3 mHz ... 100 kHz.



**Frequency ranges**

To get the best resolution for the signal to be generated, always use the frequency range with the lowest possible range number. You can generate frequencies in the following ranges:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	100 kHz	100 ns
2	4.77 Hz	100 kHz	200 ns
3	2.39 Hz	100 kHz	400 ns
4	1.20 Hz	100 kHz	800 ns
5	0.60 Hz	100 kHz	1.6 $\mu$ s
6	0.30 Hz	100 kHz	3.2 $\mu$ s
7	0.15 Hz	100 kHz	6.4 $\mu$ s
8	75 mHz	78.12 kHz	12.8 $\mu$ s
9	38 mHz	39.06 kHz	25.6 $\mu$ s
10	19 mHz	19.53 kHz	51.2 $\mu$ s
11	10 mHz	9.76 kHz	103 $\mu$ s
12	5.0 mHz	4.88 kHz	205 $\mu$ s
13	2.5 mHz	2.44 kHz	410 $\mu$ s
14	1.2 mHz	1.22 kHz	820 $\mu$ s
15	0.6 mHz	610.35 Hz	1.64 ms
16	0.3 mHz	305.17 Hz	3.28 ms

If these ranges are exceeded, square-wave signal generation will be faulty. For values outside these ranges, note the following restrictions:

Range	Restriction
Frequency < $f_{\min}$	The frequency is set to 0 Hz.
Frequency > $f_{\max}$	The frequency saturates to $f_{\max}$ .

**Output voltage**

The output channels have push-pull drivers running from an external source (2 independent VBAT rails, each of which can operate in the range 5 ... 60 V). The maximum output current per channel is  $\pm 50$  mA. Self-resetting multifuses protect against short circuits to GND and VBAT or any voltage between GND and +60 V.

**Note**

- Before operating the digital output channels, you must connect an external power supply ( $V_{\text{Bat}}$ ) to at least one of the two VBAT supply rails.
- Multifuses are designed for occasional faults only. They are therefore not suitable for failure simulation.

The output channels are in high impedance state after reset and power-up.

You can enable or disable each output channel individually, using RTI/RTLib. A disabled output is in high impedance state.

You can set the supply rail for each output individually using RTI/RTLib (low side, high side VBAT1, or high side VBAT2).

#### Note

To ensure push-pull driver functionality, the low side switch (LOW) must be set.

### RTI/RTLib support

You can generate square-wave signals via RTI and RTLib. For details, refer to:

- RTI: [Square-Wave Signal Generation \(DS2202 RTI Reference !\[\]\(c6a8736a601a632e2c96605cf66055ed\_img.jpg\)](#))
- RTLib: [Square-Wave Signal Generation \(DS2202 RTLib Reference !\[\]\(64ef2b19d70b31fbbfce0e0e2aa3d7b4\_img.jpg\)](#))

### Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2202 RTLib Reference !\[\]\(6059a5aa8b4ca7bb793408023d6c6e42\_img.jpg\)](#)).

### I/O mapping

The following table shows the mapping of the channel numbers to the corresponding I/O pins of I/O connectors P2 and P3, as used in RTI and RTLib.

#### Note

Some I/O features of the DS2202 conflict with each other. For details, see [Conflicting I/O Features](#) on page 86.

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage
1	PWM_OUT1	P2	31	P2B	6	Square-wave signal generation (shared with PWM signal generation)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
2	PWM_OUT2	P2	33	P2B	39	Square-wave signal generation (shared with PWM signal generation)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
3	PWM_OUT3	P2	35	P2B	23	Square-wave signal generation (shared with PWM signal generation)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
4	PWM_OUT4	P2	37	P2B	7	Square-wave signal generation (shared with PWM signal generation)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
5	PWM_OUT5	P2	39	P2B	40	Square-wave signal generation (shared with PWM signal generation)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
6	PWM_OUT6	P2	41	P2B	24	Square-wave signal generation (shared with PWM signal generation)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
7	PWM_OUT7	P3	16	–	–	Square-wave signal generation (shared with PWM signal generation)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
8	PWM_OUT8	P3	49	–	–	Square-wave signal generation (shared with PWM signal generation)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA

Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage
9	PWM_OUT9	P3	33	–	–	Square-wave signal generation (shared with PWM signal generation)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA

## Related topics

## References

[Digital I/O Set Up \(DS2202 RTI Reference !\[\]\(cbe80b694ebd74fcfe136a095b608235\_img.jpg\)\)](#)  
[Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(27df6be88af07602ea392719b144fe7f\_img.jpg\)\)](#)  
[Frequency Measurement \(F2D\)..... 30](#)  
[Square-Wave Signal Generation \(DS2202 RTI Reference !\[\]\(96f0a292e266dbee33329d5ab59a28c7\_img.jpg\)\)](#)



# Serial Interface

## Where to go from here

## Information in this section

<a href="#">Basics on the Serial Interface.....</a>	<a href="#">37</a>
Provides information on board's <i>universal asynchronous receiver and transmitter</i> (UART) for performing serial asynchronous communication with external devices.	
<a href="#">Comparing RS232 and RS422.....</a>	<a href="#">39</a>
The DS2202 allows you to use the RS232 or RS422 transceiver mode.	
<a href="#">Specifying the Baud Rate of the Serial Interface.....</a>	<a href="#">40</a>
Provides information on the baud rate that you can specify for the board's serial interface.	
<a href="#">Software FIFO Buffer.....</a>	<a href="#">41</a>
The serial interface features a memory section (software FIFO buffer) providing the UART with additional space for data storage. The buffer stores data that will be written to (transmit buffer) or was read by (receive buffer) the UART.	

## Basics on the Serial Interface

### UART

The board contains a universal asynchronous receiver and transmitter (UART) for performing serial asynchronous communication with external devices.

The UART interface is based on a 16C550C-compatible communication element (TL16C550C from Texas Instruments). It is driven by a 16 MHz oscillator. For more information on the TL16C550C, refer to <http://www.ti.com>. The UART can be used in the RS232 or RS422 transceiver mode with the following characteristics:

- Selectable transceiver mode (RS232, RS422). Depending on the selected transceiver mode, the I/O board can be connected to only one external serial

communication device, or to a network of devices. For details, see [Comparing RS232 and RS422](#) on page 39.

- Baud rates of up to
  - 115.2 kBd (RS232)
  - 1 MBd (RS422)

For details, see [Specifying the Baud Rate of the Serial Interface](#) on page 40.

- Selectable number of data bits, parity bit and stop bits
- Software FIFO buffer of selectable size. For details, see [Software FIFO Buffer](#) on page 41.

### Serial data transfer

Data transfer is initiated by a start bit. Starting with the least significant bit (LSB), a selectable number of data bits (5 ... 8) is transferred, followed by an optional parity bit. You can select between different parity modes (no, even, odd parity, and parity bit forced to a logical 0 or 1). 1, 1.5 or 2 stop bits follow.

### UART interrupt

The UART provides one hardware interrupt. Using RTI, this interrupt is extended to the following 4 subinterrupts:

- Interrupt triggered when the number of bytes in the receive buffer reaches a specified threshold
- Interrupt triggered when the transmit buffer is empty
- Line status interrupt
- Modem status interrupt

For information on the interrupt handling, see [Basics of DS2202 Interrupts](#) on page 82.

### RTI/RTLib support

You can access the serial interface via RTI and RTLib. For details, see

- RTI: [Serial Interface \(DS2202 RTI Reference !\[\]\(e615ca91639aee4263e67e1cc9ac86eb\_img.jpg\)](#))
- RTLib: [Serial Interface Communication \(DS2202 RTLib Reference !\[\]\(ff4f02ee9868b6fc73231e11f2af1336\_img.jpg\)](#))

### I/O mapping

The following table shows pins used by the serial interface.

Connector Pin	Sub-D Pin	Signal	Description
<b>RS232 mode</b>			
P2 53	P2B 26	TXD	RS232 transmit
P2 57	P2B 43	RXD	RS232 receive
<b>RS422 mode</b>			
P2 53	P2B 26	TXD	RS422 transmit
P2 55	P2B 10	$\overline{\text{TXD}}$	
P2 57	P2B 43	RXD	RS422 receive
P2 59	P2B 27	$\overline{\text{RXD}}$	

**Note**

The board provides only one serial interface. You can choose between RS232 and RS422 only.

## Comparing RS232 and RS422

### Introduction

The DS2202 allows you to use the RS232 or RS422 transceiver mode.

### RS232 transceiver mode

In RS232 transceiver mode, one transmitter and one receiver are supported at each data transmission line (point-to-point connection). The RS232 transceiver mode is a single-ended data transfer mode: Signals are represented by voltage levels with respect to ground. There is one wire for each signal.

**Data signals and control signals** In RS232 transceiver mode, the TXD signal provides the data to be transmitted. The RXD signal provides the received data.

**Cable length and baud rate** Due to the single-ended mode, noise signals strongly affect data transfer in an RS232 network. The maximum distance and baud rate between transmitter and receiver are therefore limited. The cable length also limits the maximum baud rate (meets EIA-232-E and V.28 specifications).

### RS422 transceiver mode

The RS422 transceiver mode is a balanced differential data transfer mode: Each signal is transmitted together with the corresponding inverted signal. For example, the data transmission signals TXD and  $\overline{\text{TXD}}$  represent a pair of balanced differential inputs.

**Data signals and control signals** In RS422 transceiver mode, the TXD and  $\overline{\text{TXD}}$  signals provide the data to be transmitted. The RXD and  $\overline{\text{RXD}}$  signals provide the received data.

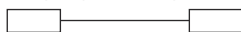
**Cable length and baud rate** Since the RS422 transceiver modes use differential signals, the effects of induced noise signals that appear as common mode voltages on a network are reduced. Compared to the RS232 transceiver mode, higher baud rates between transmitters and receivers are therefore possible. However, the cable length limits the maximum baud rate: As a rule of thumb, the baud rate (in baud) multiplied by the cable length (in meters) should not exceed  $10^8$ .

**RS422 networks** In RS422 networks, data is sent by one transmitter and received by up to 10 receivers. Two twisted pair cables – each providing two transmission lines – are usually used (unidirectional connections) for transmission

and reception of data: one twisted pair cable for the transmitted data (TXD and  $\overline{\text{TXD}}$ ), the other for the received data (RXD and  $\overline{\text{RXD}}$ ).

**Topologies of RS422 networks** In RS422 networks, you can implement different topologies such as

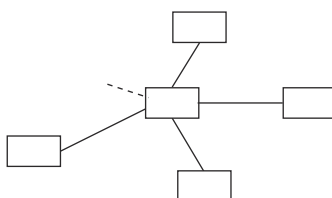
- Simple point-to-point connections



- Daisy-chain connections



- Backbone connections



## Related topics

### Basics

Serial Interface.....	37
Software FIFO Buffer.....	41

## Specifying the Baud Rate of the Serial Interface

### Oscillator frequency

The serial interface of the DS2202 is driven by an oscillator with a frequency  $f_{\text{OSC}} = 16 \text{ MHz}$ .

### Baud rate range

Depending on the selected transceiver mode, you can specify the baud rate for serial communication with the DS2202 in the following range:

Mode	Baud Rate Range
RS232	300 ... 115,200 baud
RS422	300 ... 1,000,000 baud

### Available baud rates

You can specify any baud rate in the range listed above using RTI and RTLlib. However, the baud rate used by the board is a fraction of the oscillator frequency  $f_{\text{OSC}}$ . The available baud rates can be calculated according to

$$f = f_{\text{OSC}} / (16 \cdot n),$$

where  $n$  is a positive integer within the range 1 ... 65535.



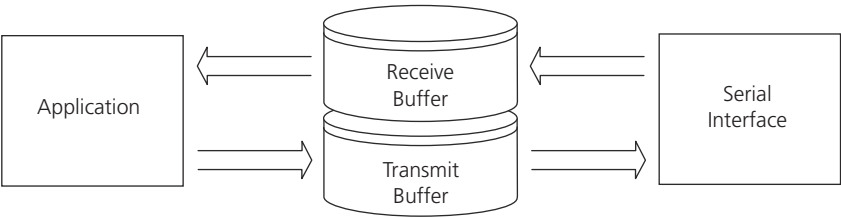
When you specify a baud rate in RTI or RTLib, the closest available baud rate is actually used for serial communication. For example, if you specify 70,000 baud as the baud rate, the baud rate used is 71,429 baud.

# Software FIFO Buffer

## Introduction

The serial interface features a memory section (software FIFO buffer) of selectable size providing the UART with additional space for data storage. The buffer stores data that will be written to (transmit buffer) or was read by (receive buffer) the UART.

The following illustration shows the buffer principle:



## Transmit buffer

Data to be transmitted usually is sent immediately.

Data that cannot be transmitted immediately is buffered in the transmit buffer (TX SW FIFO). The buffer cannot be overwritten: If an overflow of TX SW FIFO occurs, you can specify either to discard all new data, or to write as much data as possible to the buffer.

## Receive buffer

Data that is received via the serial interface is first copied to the UART FIFO buffer. When the specified number of bytes is received:

- The UART generates an interrupt.
- The bytes are moved to the receive buffer (RX SW FIFO).

If an overflow of the RX SW FIFO occurs, either old data can be overwritten, or new data discarded.

## Related topics

Basics	
Comparing RS232 and RS422.....	39



# 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

<a href="#">Setting Up a CAN Controller.....</a>	<a href="#">44</a>
Explains how to set up a CAN controller to use a dSPACE board with CAN bus interface.	
<a href="#">Using the RTI CAN MultiMessage Blockset.....</a>	<a href="#">57</a>
Provides information on using the RTI CAN MultiMessage Blockset.	
<a href="#">CAN Signal Mapping.....</a>	<a href="#">80</a>
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

<a href="#">Limited Number of CAN Messages.....</a>	<a href="#">89</a>
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.....44](#)

The CAN controller performs serial communication according to the CAN protocol. You must configure the CAN controller according to the application.

### [CAN Transceiver Types.....46](#)

The way in which CAN messages are transmitted on a CAN bus depends on the CAN transceiver used.

### [Defining CAN Messages.....50](#)

The dSPACE CAN software lets you easily define CAN messages to be transmitted or received.

### [Implementing a CAN Interrupt.....52](#)

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.....52](#)

RTI CAN Blockset provides two concepts for receiving CAN messages.

### [Removing a CAN Controller \(Go Bus Off\).....54](#)

You can remove the CAN controller that is being used from the bus when you use several CAN controllers.

### [Getting CAN Status Information.....55](#)

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.

## 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.

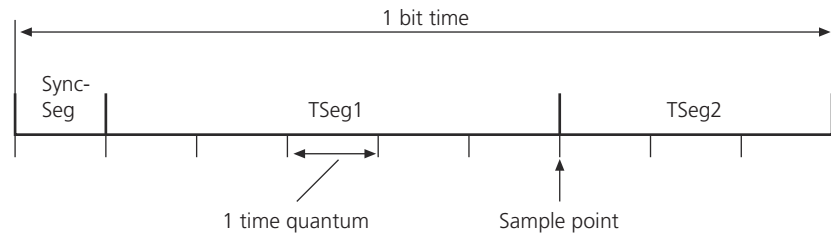
**Advanced configuration (bit timing parameters)**

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), 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<xxxx>\_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 !\[\]\(d0a1791f26d167e866e44ebbf83efebe\_img.jpg\)](#)).

**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 !\[\]\(eafc244b53721dd1ec133f0772f70fc7\_img.jpg\)](#)).

#### 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 !\[\]\(d328bb1c8b293dce97ce8ae48fe06a23\_img.jpg\)](#)).
- 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 !\[\]\(de0615d88b2098828c20ab3d39ea2ef6\_img.jpg\)](#)).

#### Related topics

##### Basics

[Gatewaying Messages Between CAN Buses \(RTI CAN Blockset Reference !\[\]\(d5d7044e5caf6907399af2dced8d6ff8\_img.jpg\)](#))

## 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 !\[\]\(9cc0308e647881098efb3200229312e5\_img.jpg\)](#)).
- To restart data transfer, use the RTICAN Bus Off Recovery block. Refer to [RTICAN Bus Off Recovery \(RTI CAN Blockset Reference !\[\]\(6f6b04114cbaf59a77679a24bfdd4e7d\_img.jpg\)](#)).

**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 !\[\]\(3d8c13c92b853674f749aac6fa869926\_img.jpg\)](#)).

---

**Related topics****References**

[RTICAN Status \(RTI CAN Blockset Reference !\[\]\(fa6f3af6bfa46c5d4a2d362681095beb\_img.jpg\)](#))



# 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..... 57](#)

Gives you an overview of the features of the RTI CAN MultiMessage Blockset.

### [Basics on Working with CAN FD..... 62](#)

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..... 67](#)

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..... 73](#)

Large CAN message bundles can be managed by a single Simulink block provided by the RTI CAN MultiMessage Blockset.

### [Manipulating Signals to be Transmitted..... 76](#)

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 62.

#### 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 !\[\]\(5361750c22c4e047a52f4eac1ec2d4cc\_img.jpg\)](#)).

### 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

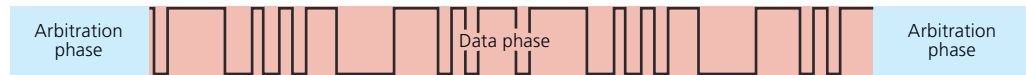
CAN FD stands for *CAN with Flexible Data Rate*. The CAN FD protocol is based on the CAN protocol as specified in ISO 11898-1. Compared with the classic CAN protocol, CAN FD comes with an increased bandwidth for the serial communication. The improvement is based on two factors:

- 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.

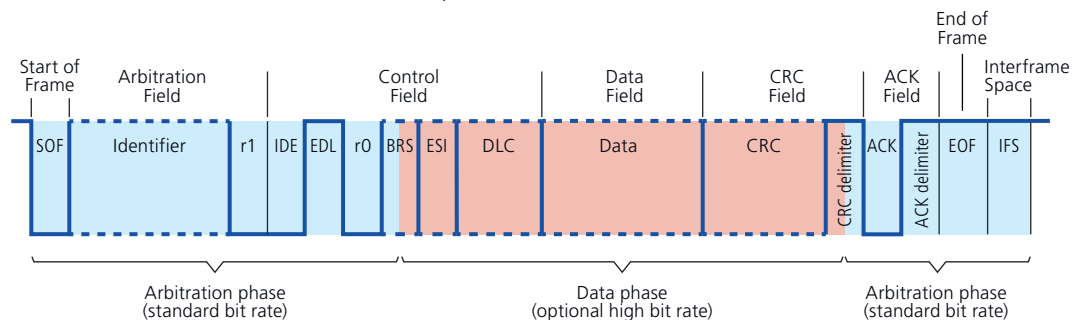
The RTI CAN MultiMessage Blockset supports both CAN FD protocols.

**CAN FD message characteristics**

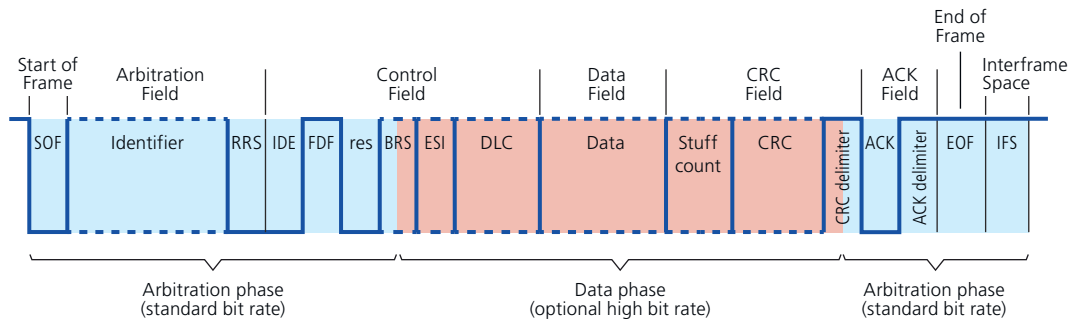
In principle, CAN FD messages and CAN messages consist of the same elements and have the same message structure.

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 !\[\]\(c6a8736a601a632e2c96605cf66055ed\_img.jpg\)](#))
- MicroAutoBox II: [DS4342 Connections in Different Topologies \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(64ef2b19d70b31fbbfce0e0e2aa3d7b4\_img.jpg\)](#))

---

### 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 !\[\]\(9c2e8d1b5bd77cb5c9f83b7a9cff79fd\_img.jpg\)](#))

## Basics on Working with a J1939-Compliant DBC File

### Introduction

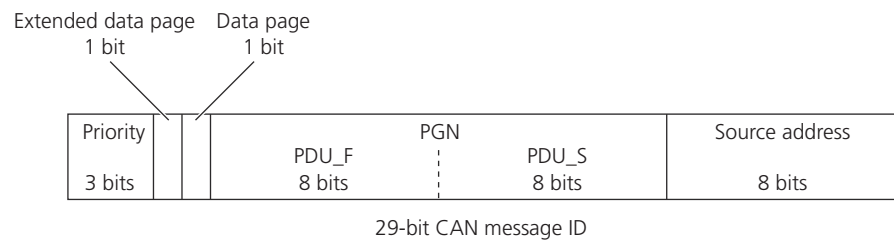
J1939 is a vehicle CAN bus standard defined by the Society of Automotive Engineers (SAE). It is used for communication in heavy-duty vehicles, for example, for communication between a tractor and its trailer.

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.

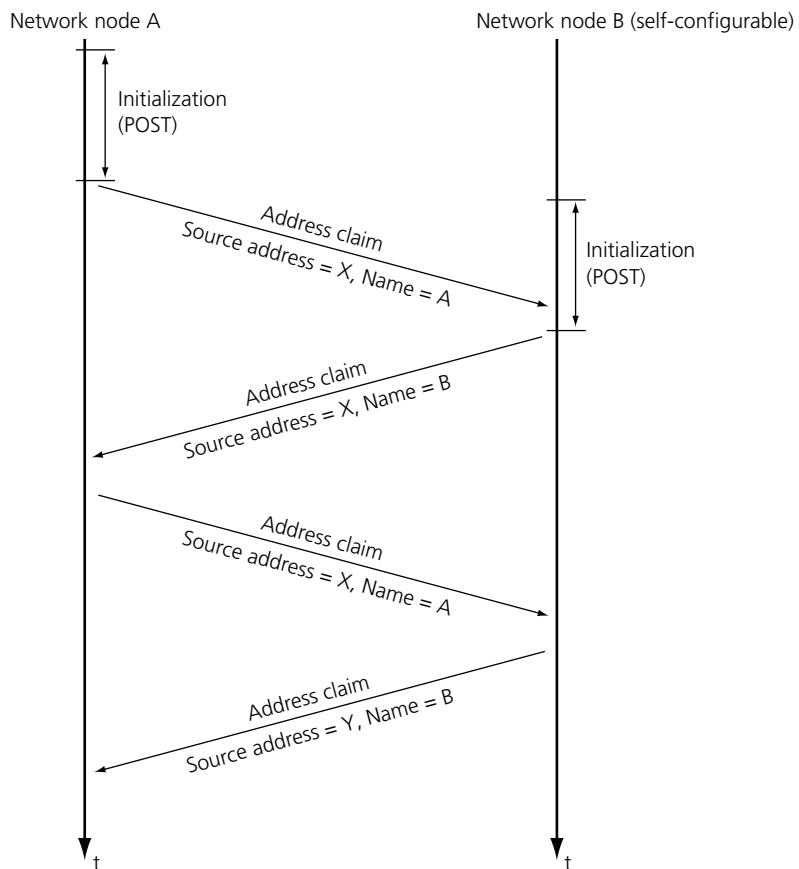
**Name** The J1939 standard defines a 64-bit *name* to identify each network node. The name indicates the main function of the network node with the associated address and provides information on the manufacturer.

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.

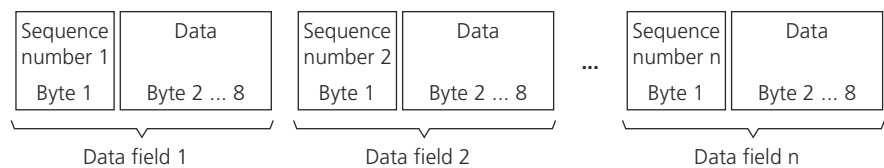
**Note**

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 !\[\]\(c3d993ca47bfe2a953c700506ce31fa0\_img.jpg\)\)](#)

# 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 !\[\]\(faf942dc3e59ce8eb64b4ac481eca7e0\_img.jpg\)\)](#). 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 )

## Manipulating Signals to be Transmitted

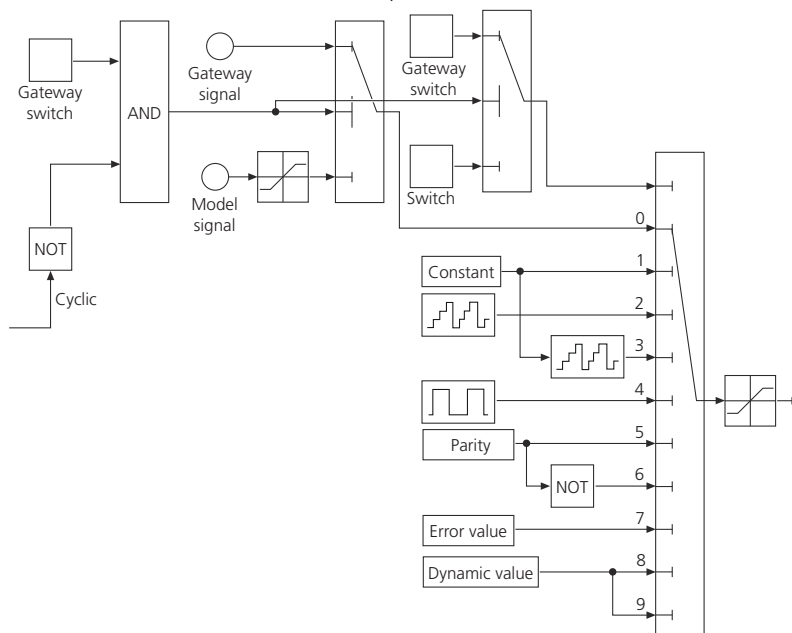
## Introduction

All the signals of all the RX and TX messages (see [Defining RX messages and TX messages](#) on page 74) automatically get corresponding entries in the generated TRC file. This allows you to analyze them (signals of RX messages) or change their values (signals of TX messages) with the Bus Navigator in ControlDesk.


### Manipulating signals to be transmitted

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

The illustration below visualizes the options.



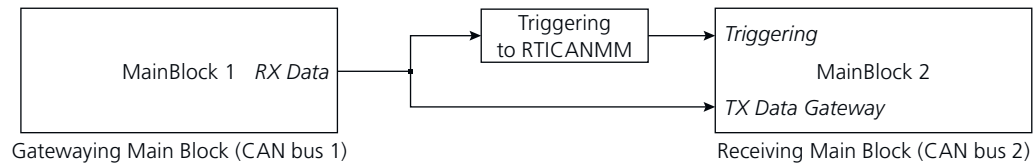
You can switch between these options in ControlDesk.


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

Because additional code has to be generated, TX model signals reduce performance. For optimum performance, you should specify as few TX model signals as possible.

You can specify TX model signals on the [Model Signals \(TX\) Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference )

**Gateway signal** A signal to be manipulated before it is exchanged between two CAN buses. Gateway signals have to be gatewayed via two RTICANMM MainBlocks. You have to specify gateway signals for the receiving MainBlock.




MainBlock1 gateway 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

### I/O signals

The following table lists the CAN signals of the DS2202 and the mapping of these signals to RTI blocks and RTLib functions.

The table also provides the mapping of the I/O signals to the I/O pins on the DS2202 and on the Sub-D connectors (P1A, P1B, P2A, P2B).

Signal	Channel/Bit Numbers of Related RTI Blocks/RTLib Functions				I/O Pin on ...	
	Related RTI Block(s)	Ch/Bit (RTI)	Related RTLib Functions	Ch/Bit (RTLib)	DS2202	Adap. Cable
<b>CAN Support</b>						
<ul style="list-style-type: none"> <li>CANxL: CAN dominant low</li> <li>CANxH: CAN dominant high</li> <li>Electrical characteristics: see <a href="#">CAN Bus Interface (PHS Bus System Hardware Reference [1])</a></li> </ul>						
CAN1L	<ul style="list-style-type: none"> <li>RTICAN CONTROLLER SETUP</li> <li>RTICANMM ControllerSetup</li> </ul>	CAN1	Slave CAN Access Functions	CAN1	P2 83	P2B 31
CAN1H					P2 81	P2B 47
GND					P2 79, 85	P2B 14, 15
CAN2L		CAN2		CAN2	P2 84	P2A 31
CAN2H					P2 82	P2A 47
GND					P2 80, 86	P2A 14, 15

### Related topics

#### Basics

[CAN Support](#)..... 43

#### References

[CAN Bus Interface \(PHS Bus System Hardware Reference \[1\]\)](#)  
[RTICAN CONTROLLER SETUP \(RTI CAN Blockset Reference \[1\]\)](#)  
[RTICANMM ControllerSetup \(RTI CAN MultiMessage Blockset Reference \[1\]\)](#)  
[Slave CAN Access Functions \(DS2202 RTLib Reference \[1\]\)](#)



# Interrupt Handling and DS2202 Interrupts

Where to go from here

Information in this section

Basics of Interrupt Handling.....	81
Explains how interrupts are handled with RTI or RTLib.	
Basics of DS2202 Interrupts.....	82
Provides information on the interrupts of the board.	

## Basics of Interrupt Handling

Introduction

Interrupts from the DS2202 are sent to the master interrupt controller of the connected processor board via the interrupt lines of the PHS bus.

Interrupt lines of the PHS bus

Each dSPACE I/O board is connected to a dSPACE processor board via the PHS bus. The PHS bus provides 8 interrupt lines, enabling the I/O boards to generate interrupts on the processor board. Interrupts from the I/O board are sent to the master interrupt controller of the connected processor board via the interrupt lines of the PHS bus.

The 8 interrupt lines are connected to the inputs of the master interrupt controller on the processor board. I/O boards supporting interrupts each have a slave interrupt controller unit (ICU) with 8 interrupt inputs. Assignment of slave interrupt controller outputs to PHS-bus interrupt lines is programmable. Only one I/O board can use the same PHS-bus interrupt line at a time. The configuration consisting of a cascaded master and slave interrupt controllers permits a total of 64 interrupts in a system.

**Interrupt handling  
with RTI/RTLib**

Interrupt handling varies depending on whether you use RTI blocks or RTLib functions for your application:

**Interrupt-driven subsystems in RTI** You can use interrupts or subinterrupts to trigger interrupt-driven subsystems in your Simulink model. When the task in one system has finished, the interrupt handler automatically creates an "End of interrupt" (EOI) message to indicate the state to other units. Refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(99f58673407353e96a019fbca558fd72\_img.jpg\)](#)).

**Handcoded models** If you use handcoded models, you have to program the interrupt handling yourself. RTLib provides the interrupt handlers and functions required.

**Related topics****Basics**

[Interrupt Controller \(DS1006 Features !\[\]\(de95854c7ee024cfadc48187bbb781b2\_img.jpg\)](#))

[Interrupt Controller \(DS1007 Features !\[\]\(3211b5d1d968fc1665909b34f9f16010\_img.jpg\)](#))

## Basics of DS2202 Interrupts

**Interrupts**

The DS2202 provides access to the following interrupts:

Interrupt Type	Description
CAN interrupt	Interrupt from the CAN subsystem. Subinterrupts for message events or CAN bus events can be defined.
Serial interface interrupt	Interrupt from the serial interface (UART). Subinterrupt handlers permit handling of all interrupts that the serial interface can generate.

**RTI/RTLib support**

You can access the interrupts via RTI and RTLib.

**CAN interrupt** Refer to:

- [RTICAN Interrupt \(RTI CAN Blockset Reference !\[\]\(3dc92c626ede9fa1b47e2e010104b5c4\_img.jpg\)](#))
- [Slave CAN Access Functions \(DS2202 RTLib Reference !\[\]\(71e9a2c5583c3d2a2fe005f4239e5d39\_img.jpg\)](#))

**Serial interface interrupt** Refer to:

- [DS2202SER\\_INT\\_Bx\\_Iy \(DS2202 RTI Reference !\[\]\(6c117786eacd86d9626685ebfb559b77\_img.jpg\)](#))
- [Serial Interface Communication \(DS2202 RTLib Reference !\[\]\(a2437798f31357d2bea910e1270385bf\_img.jpg\)](#))

## Related topics

## References

[DS2202SER\\_INT\\_Bx\\_ly](#) (DS2202 RTI Reference )  
[RTICAN Interrupt](#) (RTI CAN Blockset Reference )  
[Serial Interface Communication](#) (DS2202 RTLib Reference )  
[Slave CAN Access Functions](#) (DS2202 RTLib Reference )



# Limitations

## Where to go from here

## Information in this section

<a href="#">Quantization Effects.....</a>	<a href="#">85</a>
Quantization effects occur in signal generation or measurement.	
<a href="#">Conflicting I/O Features.....</a>	<a href="#">86</a>
Conflicting I/O features are I/O features that share the same board resources.	
<a href="#">Limited Number of CAN Messages.....</a>	<a href="#">89</a>
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>	<a href="#">91</a>
There are a number of general limitations with RTICANMM.	
<a href="#">Limitations with J1939-Support.....</a>	<a href="#">95</a>
There are a number of limitations regarding the J1939 support of RTICANMM.	

## Quantization Effects

### Introduction

Signal generation and measurement are only feasible within the limits of the resolution of the timing I/O unit. The limited resolution causes quantization errors that increase with increasing frequencies.

When performing square-wave signal generation (D2F), for example, you will encounter considerable deviations between the desired frequency and the generated frequency, especially for higher frequencies. The (quantized) generated signal frequencies can be calculated according to the following equation:

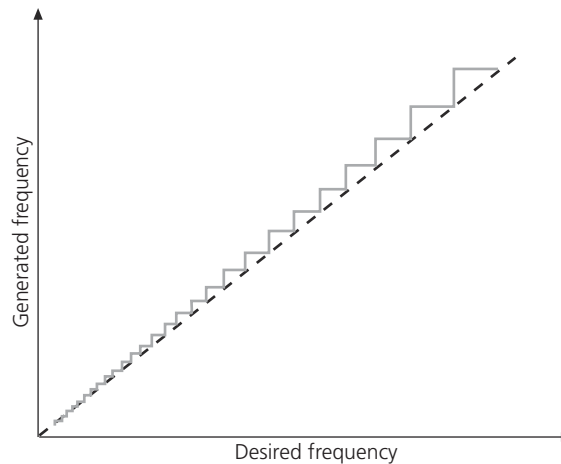
$$f = \frac{1}{n \cdot R}$$

where  $R$  is the resolution (in s), and  $n$  is a positive integer.

### Example

For example, if you select range 16 (0.3 mHz ... 305.17 Hz) and 130 Hz is specified as the desired frequency for D2F, a frequency of 152.59 Hz is actually generated.

The following illustration shows the increasing quantization effects for increasing desired frequencies:



You should therefore select the range with the best possible resolution.

## Conflicting I/O Features

### Types of I/O conflicts

There are I/O features that share the same board resources.

**Conflicts concerning single I/O channels** There are conflicts that concern single channels of an I/O feature. The dSPACE board provides only a limited number of I/O pins. The same pins can be shared by different I/O features. However, a pin can serve as the I/O channel for only one feature at a time.

**Conflicts concerning an I/O feature as a whole** There are conflicts that concern the use of an I/O feature as a whole. Suppose two I/O features of the dSPACE board use the same on-board timer device. In this case, only one of the two I/O features can be used at a time. The other feature is completely blocked.

**Conflicts concerning signal inputs** If an I/O pin of a signal input is shared, this pin can serve more than one feature at a time. For example, you can use a DIG\_INx pin to get the status of a signal and to measure the duty cycle or period.




**Conflicts for the DS2202**

The following tables list the I/O features of the DS2202 that conflict with other I/O features, and the related RTI blocks/RTLib functions.

- Conflicts for the multi I/O interface
  - [Conflicts for digital inputs](#) on page 87
  - [Conflicts for PWM signal measurement \(PWM2D\)](#) on page 87
  - [Conflicts for square-wave signal measurement \(F2D\)](#) on page 88
  - [Conflicts for PWM signal generation](#) on page 88
  - [Conflicts for square-wave signal generation \(D2F\)](#) on page 89
- Conflicts for the serial interface: see [Conflicts for the serial interface](#) on page 89

**Conflicts for digital inputs**

The following I/O features of the DS2202 conflict with digital inputs provided by the multi I/O interface:

Digital Inputs *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 16	Ch 1 ... ch 16	DIG_IN1 ... DIG_IN16	<ul style="list-style-type: none"><li>▪ Square-wave signal measurement (F2D)</li><li>▪ PWM Signal Measurement (PWM2D)</li></ul>	Ch 9 ... ch 24 Ch 9 ... ch 24	Ch 9 ... ch 24 Ch 9 ... ch 24
Ch 17 ... ch 24	Ch 17 ... ch 24	DIG_IN17 ... DIG_IN24	<ul style="list-style-type: none"><li>▪ Square-wave signal measurement (F2D)</li><li>▪ PWM Signal Measurement (PWM2D)</li></ul>	Ch 1 ... ch 8 Ch 1 ... ch 8	Ch 1 ... ch 8 Ch 1 ... ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"><li>▪ DS2202BIT_IN16_Bx_Gy, DS2202BIT_IN_Bx_Cy</li><li>▪ See <a href="#">Bit I/O Unit (DS2202 RTLib Reference </a>)</li></ul>			**) Related RTI blocks and RTLib functions: Square-wave signal measurement (F2D): <ul style="list-style-type: none"><li>▪ DS2202F2D_Bx_Cy</li><li>▪ See <a href="#">Frequency Measurement (DS2202 RTLib Reference </a>)</li></ul> PWM2D: <ul style="list-style-type: none"><li>▪ DS2202PWM2D_Bx_Cy</li><li>▪ See <a href="#">PWM Signal Measurement (DS2202 RTLib Reference </a>)</li></ul>		

**Conflicts for PWM signal measurement (PWM2D)**




The following I/O features of the DS2202 conflict with PWM signal measurement provided by the multi I/O interface:

PWM Signal Measurement *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 8	Ch 1 ... ch 8	PWM_IN1 ... PWM_IN8	<ul style="list-style-type: none"><li>▪ Square-wave signal measurement (F2D)</li><li>▪ Digital input</li></ul>	Ch 1 ... ch 8  Ch 17 ... ch 24	Ch 1 ... ch 8  Ch 17 ... ch 24

PWM Signal Measurement *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Ch 17 ... ch 24	Ch 17 ... ch 24	PWM_IN9 ... PWM_IN24	<ul style="list-style-type: none"> <li>Square-wave signal measurement (F2D)</li> <li>Digital input</li> </ul>	Ch 9 ... ch 24  Ch 1 ... ch 16	Ch 9 ... ch 24  Ch 1 ... ch 16
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> <li>DS2202PWM2D_Bx_Cy</li> <li>See <a href="#">PWM Signal Measurement (DS2202 RTLib Reference)</a></li> </ul>			**) Related RTI blocks and RTLib functions: Square-wave signal measurement (F2D): <ul style="list-style-type: none"> <li>DS2202F2D_Bx_Cy</li> <li>See <a href="#">Frequency Measurement (DS2202 RTLib Reference)</a></li> </ul> Digital input: <ul style="list-style-type: none"> <li>DS2202BIT_IN16_Bx_Gy, DS2202BIT_IN_Bx_Cy</li> <li>See <a href="#">Bit I/O Unit (DS2202 RTLib Reference)</a></li> </ul>		



### Conflicts for square-wave signal measurement (F2D)

The following I/O features of the DS2202 conflict with square-wave signal measurement provided by the multi I/O interface:

Square-Wave Signal Measurement *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 8	Ch 1 ... ch 8	PWM_IN1 ... PWM_IN8	<ul style="list-style-type: none"><li>▪ Square-wave signal measurement (F2D)</li><li>▪ Digital input</li></ul>	Ch 1 ... ch 8  Ch 17 ... ch 24	Ch 1 ... ch 8  Ch 17 ... ch 24
Ch 17 ... ch 24	Ch 17 ... ch 24	PWM_IN9 ... PWM_IN24	<ul style="list-style-type: none"><li>▪ Square-wave signal measurement (F2D)</li><li>▪ Digital input</li></ul>	Ch 9 ... ch 24  Ch 1 ... ch 16	Ch 9 ... ch 24  Ch 1 ... ch 16
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"><li>▪ DS2202F2D_Bx_Cy</li><li>▪ See <a href="#">Frequency Measurement (DS2202 RTLib Reference</a> </li></ul>			**) Related RTI blocks and RTLib functions: PWM2D: <ul style="list-style-type: none"><li>▪ DS2202PWM2D_Bx_Cy</li><li>▪ See <a href="#">PWM Signal Measurement (DS2202 RTLib Reference</a> </li></ul> Digital input: <ul style="list-style-type: none"><li>▪ DS2202BIT_IN16_Bx_Gy, DS2202BIT_IN_Bx_Cy</li><li>▪ See <a href="#">Bit I/O Unit (DS2202 RTLib Reference</a> </li></ul>		

### Conflicts for PWM signal generation



The following I/O features of the DS2202 conflict with PWM signal generation provided by the multi I/O interface:

PWM Signal Generation *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)		Ch (RTI)	Ch (RTLib)	
Conflicts Concerning Single Channels					
Ch 1 ... ch 9	Ch 1 ... ch 9	PWM_OUT1 ... PWM_OUT9	Square-wave signal generation (D2F)	Ch 1 ... ch 9	Ch 1 ... ch 9
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"><li>▪ DS2202PWM_Bx_Cy</li><li>▪ See <a href="#">PWM Signal Generation (DS2202 RTLib Reference </a>)</li></ul>			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"><li>▪ DS2202D2F_Bx_Cy</li><li>▪ See <a href="#">Square-Wave Signal Generation (DS2202 RTLib Reference </a>)</li></ul>		



**Conflicts for square-wave signal generation (D2F)**

The following I/O features of the DS2202 conflict with square-wave signal generation provided by the multi I/O interface:

Square-Wave Signal Generation *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	Ch (RTLib)
Ch (RTI)	Ch (RTLib)				
<b>Conflicts Concerning Single Channels</b>					
Ch 1 ... ch 9	Ch 1 ... ch 9	PWM_OUT1 ... PWM_OUT9	PWM signal generation	Ch 1 ... ch 9	Ch 1 ... ch 9
*) Related RTI blocks and RTLib functions: ▪ DS2202D2F_Bx_Cy ▪ See <a href="#">Square-Wave Signal Generation (DS2202 RTLib Reference </a> )			**) Related RTI blocks and RTLib functions: ▪ DS2202PWM_Bx_Cy ▪ See <a href="#">PWM Signal Generation (DS2202 RTLib Reference </a> )		

**Conflicts for the serial interface**

The DS2202 supports only one serial interface. It can be configured to either RS232 or RS422 mode.

**Related topics****Basics**

[Introduction to the Features of the DS2202..... 7](#)

## 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  $n_{\text{sum}}$ :

$$n_{\text{sum}} = n_{\text{TX}} + n_{\text{RX}} + n_{\text{RQ}} + n_{\text{RM}}$$

**Maximum number of CAN messages**

The sum of the above messages  $n_{\text{sum}}$  in one application must always be smaller than or equal to the maximum number of CAN messages  $n_{\text{max}}$ :

$$n_{\text{sum}} \leq n_{\text{max}} ; n_{\text{RM}} \leq 10$$

$n_{\text{max}}$  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  $n_{\text{max}}$  is listed in the table below:

Platform	$n_{\text{max}}$ with RTLib	$n_{\text{max}}$ 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  $n_{\text{RX}}$  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 52.

**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.

## Limitations with J1939-Support

### Limitations

The following limitations apply to the J1939 support of the RTI CAN MultiMessage Blockset:

- The J1939 support for the RTI CAN MultiMessage Blockset requires a separate license.
- To use J1939, you must provide a J1939-compliant DBC file.
- Though most messages are already defined in the J1939 standard, you must specify the required messages in your DBC file.
- When you gateway messages, J1939 network management (address claiming) is not supported. This limitation applies to gatewaying via RTICANMM MainBlocks and via RTICANMM Gateway block.
- When you gateway J1939 messages via an RTICANMM Gateway block, multipacket messages cannot be added to the filter list. This means that J1939 messages longer than 8 bytes cannot be excluded from being gatewayed.
- For J1939 messages, the CRC option is limited to the first eight bytes.
- For J1939 messages, the custom code option is limited to the first eight bytes.
- Peer-to-peer communication for J1939 messages longer than 8 bytes via RTS/CTS is supported only for receiving network nodes whose simulation type is set to 'simulated' or 'external'.
- CAN messages with extended identifier format and also J1939 messages use a 29-bit message identifier. Because the RTI CAN MultiMessage Blockset cannot differentiate between the two message types on the CAN bus, working with extended CAN messages and J1939 messages on the same bus is not supported.
- For J1939 messages, the manipulation of the PGN is not supported.





**A**

ADC unit  
DS2202 14

**B**

bit I/O unit  
DS2202 18

**C**

CAN  
channel 44  
fault-tolerant transceiver 48  
interrupts 52  
physical layer 46  
service 55  
setup 44  
status information 55  
CAN FD 62  
CAN support 43  
Common Program Data folder 6  
comparing transceiver modes 39

**D**

DAC unit  
DS2202 16  
data file support 51  
defining CAN messages 50  
Documents folder 6  
DS2202 7  
ADC unit 14  
characteristics 14  
I/O mapping 15  
bit I/O unit 18  
characteristics 18  
digital inputs 18  
digital outputs 18  
I/O mapping 19  
DAC unit 16  
characteristics 16  
I/O mapping 17  
duty cycle  
measuring 22  
feature summary 9  
frequency measurement 30  
characteristics 30  
I/O mapping 31  
I/O mapping  
ADC unit 15  
bit I/O unit 19  
DAC unit 17  
frequency measurement 31  
PWM signal generation 29  
PWM signal measurement 24  
square-wave signal generation 34  
interrupts 81  
introduction to the features 7  
limitations 85  
measuring duty cycle 22

PHS++ bus 10  
PWM signal generation 25  
characteristics 25  
I/O mapping 29  
PWM signal measurement 22  
characteristics 22  
I/O mapping 24  
square-wave signal generation 32  
characteristics 32  
I/O mapping 34  
DS802  
partitioning PHS bus 11

**F**

feature summary  
DS2202 9  
frequency measurement  
DS2202 30

**I**

interrupts  
DS2202 81  
ISO11898 47  
ISO11898-6 48

**J**

J1939  
broadcast/peer-to-peer messages 67  
limitations 95  
working with J1939-compliant DBC files 67

**L**

limitations  
DS2202 85  
J1939 95  
RTI CAN MultiMessage Blockset 91  
Local Program Data folder 6

**M**

messages  
defining CAN messages 50  
delay time 50  
multiple 51  
multiple data files 51  
multiple message support 51

**P**

partitioning PHS bus with DS802 11  
PHS bus  
interrupt lines 81  
piggyback support 49  
PWM signal generation  
DS2202 25  
PWM signal measurement  
DS2202 22

**Q**

quantization effects 85

**R**

receive buffer 41  
RS485 48  
RTI CAN MultiMessage Blockset  
limitations 91  
supported platforms 57  
RTICANMM  
J1939 95  
RX service support 52  
RX SW FIFO 41

**S**

serial interface 37  
baud rates 40  
cable length/baud rate (RS232) 39  
cable length/baud rate (RS422) 39  
oscillator frequency 40  
RS232 transceiver mode 39  
RS422 networks 39  
RS422 topologies 40  
RS422 transceiver mode 39  
square-wave signal generation  
DS2202 32

**T**

TJA1041  
limitations 49  
TJA1054  
transceiver 49  
transceiver  
TJA1054 49  
transmit buffer 41  
TX SW FIFO 41

**U**

UART 37

**W**

working with CAN FD 62

