

MicroAutoBox II

Features

For all variants of MicroAutoBox II

Release 2021-A – May 2021

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

This publication 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.

© 2000 - 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 Reference	9
Introduction to the Features of MicroAutoBox II	11
System Overview	12
Hardware Concept	12
Overview of Board Revisions	15
MicroAutoBox II 1401/1507	19
MicroAutoBox II 1401/1511	20
MicroAutoBox II 1401/1511/1514	21
MicroAutoBox II 1401/1513	22
MicroAutoBox II 1401/1513/1514	23
MicroAutoBox II with DS1552 Multi-I/O Module	23
MicroAutoBox II with DS1554 Engine Control I/O Module	24
Feature Support	25
Base Board's Basics	28
MicroAutoBox II Base Board	28
MicroAutoBox II Application Start	30
Boot Firmware	30
Running an Application from the Global Memory	32
Running an Application from the Flash Memory	33
MicroAutoBox II Basic Features	35
Interrupt Handling	37
Basics on Interrupt Handling	37
Word-Based Subinterrupt Handling	40
Basics on Word-Based Subinterrupt Handling	40
Basics on the Alive Mechanism	41
Basics on Subinterrupt Handling	44
Timer Services	47
Timer	47
Decrementer	48
Time Base Counter	48
Safety Functions	49
Basics on Implementing Safety Functions	49

Basics on Watchdog Handling.....	50
Basics on Challenge-Response Monitoring.....	52
Basics on Memory Integrity and Extras.....	57
Nonvolatile Data Handling.....	61
Basics on RTC RAM Access.....	61
Basics on Nonvolatile RAM Access.....	62
Basics on Flash Memory Access.....	63
Flight Recorder.....	65
Flight Recorder (Flash Memory).....	65
Basics on Flight Recorder.....	66
Using the Flight Recorder.....	67
MAT File Format for the Flight Recorder.....	68
USB Flight Recorder.....	69
Basics on USB Flight Recorder.....	70
Handling the Data of the USB Flight Recorder.....	74
Power Hold Control.....	76
Basics of Power Hold Control.....	76
Onboard Sensors.....	78
Sensor for Acceleration Measurement.....	78
Sensor for Pressure Measurement.....	81
MicroAutoBox II I/O Features	83
Information on the I/O Module Availability.....	84
Overview of the Number of Available I/O Modules.....	84
A/D Conversion.....	86
Overview of the A/D Conversion Units.....	86
ADC Unit Type 4.....	87
AIO Unit Type 1 (ADC).....	99
ADC 1552 Type 1 Unit.....	101
ADC 1552 Type 2 Unit.....	108
Bit I/O.....	111
Overview of the Bit I/O Units.....	111
Bit I/O Unit (DIO Type 3).....	112
Bit I/O Unit (DIO Type 4).....	117
Bit I/O Unit (DIO 1552 Type 1).....	122
D/A Conversion.....	126
Overview of the D/A Conversion Units.....	126
AIO Unit Type 1 (DAC).....	127

DAC Unit Type 3.....	128
DAC 1552 Type 1 Unit.....	129
Ethernet I/O Interface.....	131
Basics on the Ethernet I/O Interface.....	131
ECU Interface.....	133
General Description.....	133
Hardware.....	134
Working Modes.....	136
DPMEM Addresses Seen from the ECU.....	137
Data Type Formats.....	139
ECU Interrupts.....	140
ECU Software Porting Kit.....	142
Working with the ECU Software Porting Kit.....	142
PWM Signal Generation with a Variable Period.....	144
Overview of the Signal Generation Functionalities.....	144
PWM Generation (PWM) on the DIO Type 3 Unit.....	147
PWM Generation (PWM) on the DIO Type 4 Unit.....	151
PWM Generation (PWM) on the DIO 1552 Type 1 Unit.....	156
Square-Wave Signal Generation (FREQ) on the DIO Type 3 Unit.....	159
Square-Wave Signal Generation (FREQ) on the DIO Type 4 Unit.....	162
Square-Wave Signal Generation (FREQ) on the DIO 1552 Type 1 Unit.....	165
Multi-Channel PWM Signal Generation (MC_PWM).....	169
Basics on Multi-Channel PWM Signal Generation on the DIO Type 3.....	169
Basics on Multi-Channel PWM Signal Generation on the DIO Type 4.....	182
PWM Signal Measurement (PWM2D).....	195
Overview of the Signal Measurement Functionalities.....	195
PWM Measurement (PWM2D) on the DIO Type 3 Unit.....	197
PWM Measurement (PWM2D) on the DIO Type 4 Unit.....	202
PWM Measurement (PWM2D) on the DIO 1552 Type 1 Unit.....	205
Pulse Pattern Measurement.....	209
Frequency Measurement (F2D) on the DIO Type 3 Unit.....	209
Frequency Measurement (F2D) on the DIO Type 4 Unit.....	213
Frequency Measurement (F2D) on the DIO 1552 Type 1 Unit.....	216
Pulse Width Measurement (PW2D).....	220
Pulse Width Measurement (PW2D) on the DIO Type 3 Unit.....	220
Pulse Width Measurement (PW2D) on the DIO Type 4 Unit.....	224
Incremental Encoder Interface.....	228
Overview of the Incremental Encoder Interfaces.....	228
Basics on the Incremental Encoder Interface.....	229

Incremental Encoder Interface on the DIO Type 3 Unit.....	231
Incremental Encoder Interface on the DIO Type 4 Unit.....	236

CAN Support 241

Setting Up a CAN Controller.....	243
Initializing the CAN Controller.....	244
CAN Transceiver Types.....	245
MicroAutoBox II: Selecting the CAN Controller Frequency.....	249
Defining CAN Messages.....	250
Implementing a CAN Interrupt.....	251
Using RX Service Support.....	252
Removing a CAN Controller (Go Bus Off).....	254
Getting CAN Status Information.....	254
CAN Partial Networking.....	256
Using the RTI CAN MultiMessage Blockset.....	258
Basics on the RTI CAN MultiMessage Blockset.....	258
Basics on Working with CAN FD.....	263
Basics on Working with a J1939-Compliant DBC File.....	268
Transmitting and Receiving CAN Messages.....	274
Manipulating Signals to be Transmitted.....	277
CAN Signal Mapping.....	281
CAN Signal Mapping.....	281

LIN Support 287

LIN Basics.....	290
LIN Basics.....	290
Master/Slave Concept.....	291
Example of a LIN Bus.....	292
Fields of Application.....	294
Remaining Bus Simulation.....	294
Testing Communication Timing Constraints.....	295
Testing Against Specification Limits.....	295
Detecting the Baud Rate of a LIN Bus.....	296
Testing Diagnostic and Failure Conditions.....	297
Testing Energy-Saving Modes.....	298
LIN Bus Handling.....	300
Setting Up a LIN Bus.....	300
Basics on LIN Bus Handling.....	301
Specifying the LIN Bus Parameters.....	302

Database Files.....	302
Defining LIN Nodes.....	303
Handling Frames.....	304
Transmitting and Receiving Frames.....	304
Manipulating Frames.....	305
Controlling a LIN Bus.....	305
Switching LIN Nodes.....	306
Setting a LIN Bus to Sleep Mode.....	306
Waking Up a LIN Bus.....	306
Changing LIN Bus Parameters.....	307
Reading Status Information.....	307
Handling Schedules.....	308
Setting Up a LIN Schedule.....	308
Setting the Priority of LIN Schedules.....	309
Using Interrupts.....	309
Defining Interrupts.....	309
Using the RTI LIN MultiMessage Blockset.....	311
Basics on the RTI LIN MultiMessage Blockset.....	311
Transmitting and Receiving LIN Frames.....	313
How to Define a Checksum Algorithm.....	316
Manipulating Signals to be Transmitted.....	317
FlexRay Support	319
Basics on FlexRay Support.....	320
IP Module Support	323
Basics on IP Module Support.....	323
FPGA Support	325
General Information on the FPGA Support.....	326
Accessing the FPGA Type 1 Unit.....	327
Serial Interface	329
Basics on Serial Interface.....	329
Specifying the Baud Rate of the Serial Interface.....	331

Single Edge Nibble Transmission (SENT) Support	333
Basics of the SENT Protocol.....	334
Basics on SENT Diagnostic Information.....	337
Using the SENT Protocol on MicroAutoBox II.....	338
Implementing SENT Receivers in Simulink.....	342
Implementing SENT Receivers Using RTLib Functions.....	345
Serial Peripheral Interface	351
Serial Peripheral Interface on the DIO Type 3 Unit.....	351
Serial Peripheral Interface on the DIO Type 4 Unit.....	360
Limitations	369
Limited Number of CAN Messages.....	369
Limitations with RTICANMM.....	371
Limitations with CAN FD.....	375
Limitations with J1939-Support.....	375
Limitations of RTI LIN MultiMessage Blockset.....	376
I/O Mappings	383
I/O Mapping for MicroAutoBox II 1401/1507.....	383
I/O Mapping for MicroAutoBox II 1401/1511.....	384
I/O Mapping for MicroAutoBox II 1401/1513.....	388
I/O Mapping for MicroAutoBox II with DS1552 Multi-I/O Module.....	392
I/O Mapping for MicroAutoBox II with DS1554 Engine Control I/O Module.....	395
Appendix	397
Troubleshooting.....	397
Glossary.....	398
Index	401

About This Reference

Content





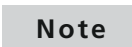


This document provides feature-oriented access to the reference information you need to implement your control models on one of the MicroAutoBox II variants:


MicroAutoBox II

- 1401/1507
- 1401/1511
- 1401/1511/1514, optionally with DS1552 Multi-I/O Module or DS1554 Engine Control I/O Module
- 1401/1513
- 1401/1513/1514, optionally with DS1552 Multi-I/O Module or DS1554 Engine Control I/O Module
- MicroAutoBox II with MicroAutoBox Embedded PC

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.

Symbol	Description
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files


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

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

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

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

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

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

Introduction to the Features of MicroAutoBox II

Field of application

MicroAutoBox II combines the advantages of a rapid control prototyping (RCP) system with those of an automotive electronic control unit (ECU). Therefore, it is ideally suited as hardware for prototyping in a vehicle.

MicroAutoBox II operates without user intervention, just like an ECU, and can be installed virtually anywhere in the vehicle. At the same time MicroAutoBox II provides all the benefits of a dSPACE real-time system. A PC or notebook can be attached temporarily for program download, data analysis and calibration.

Where to go from here

Information in this section

System Overview.....	12
Base Board's Basics.....	28
MicroAutoBox II Application Start.....	30
A real-time application can be run from the program memory and from the flash memory.	

System Overview

Introduction dSPACE provides the MicroAutoBox II in different variants. This section gives you an overview on the hardware concept and the differences of the MicroAutoBox II variants.

Where to go from here	Information in this section
	Hardware Concept..... 12
	Overview of Board Revisions..... 15
	MicroAutoBox II 1401/1507..... 19
	MicroAutoBox II 1401/1511..... 20
	MicroAutoBox II 1401/1511/1514..... 21
	MicroAutoBox II 1401/1513..... 22
	MicroAutoBox II 1401/1513/1514..... 23
	MicroAutoBox II with DS1552 Multi-I/O Module..... 23
	MicroAutoBox II with DS1554 Engine Control I/O Module..... 24
	Feature Support..... 25

Hardware Concept

Hardware components

A MicroAutoBox II system consists of two or three boards in a milled aluminum box:

- One DS1401 base board
- One or two I/O boards (DS15xx)

Optionally, the MicroAutoBox Embedded PC is integrated in a MicroAutoBox II system.

The DS1401 base board and the I/O boards are connected via an internal bus (intermodule bus).

For details on MicroAutoBox II's hardware package, refer to [Hardware \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(23d9fc146e83b5c3013cfa32c784f8d5_img.jpg\)](#)).

DS1401 base board	<p>The DS1401 base board provides the basic units of MicroAutoBox II, for example, the PowerPC processor, the memory and the onboard power supply unit. Because the PowerPC is executing the real-time application, it is called the <i>Master PPC</i>.</p>
I/O boards	<p>The I/O boards that can be combined with the DS1401 base board particularly determine the I/O functionality of the whole MicroAutoBox II system.</p> <p>MicroAutoBox II variant A MicroAutoBox II system consists of one or two I/O boards depending on the MicroAutoBox II variant you have purchased. For example, MicroAutoBox II 1401/1511/1514 consists of the DS1401 base board and the DS1511 and DS1514 I/O Boards. The following variants exist:</p> <ul style="list-style-type: none"> ▪ MicroAutoBox II 1401/1507 ▪ MicroAutoBox II 1401/1511 ▪ MicroAutoBox II 1401/1511/1514 ▪ MicroAutoBox II 1401/1513 ▪ MicroAutoBox II 1401/1513/1514 ▪ MicroAutoBox II with DS1552 Multi-I/O Module ▪ MicroAutoBox II with DS1554 Engine Control I/O Module ▪ MicroAutoBox II with MicroAutoBox Embedded PC <p>For an overview of the variant-specific features, refer to Feature Support on page 25.</p> <p>For information on the I/O boards revisions and the required boot firmware versions, refer to Overview of Board Revisions on page 15.</p>
I/O board subcomponents	<p>Depending on the I/O board type, the I/O board has specific functional subcomponents:</p> <p>CAN subsystem (Slave CAN MC) A further subsystem of the I/O board providing CAN microcontrollers with two different clock rates, is used for connection to up to two CAN buses. It is briefly named as <i>Slave CAN MC</i>. The PPC of the DS1401 is the master, whereas the CAN microcontrollers are slaves.</p> <p>FlexRay IP modules FlexRay IP modules are used for connection to a FlexRay bus. No microcontroller is used.</p>
Available I/O modules	<p>On the DS1514 I/O board you can mount an I/O module, that is a piggyback board, to enlarge the I/O capability of your MicroAutoBox II. An I/O module provides signal conditioning for the mapped I/O pins.</p> <p>DS1552 Multi-I/O Module With the DS1552 Multi-I/O Module you can extend your MicroAutoBox II with the following features by using the RTI DS1552 I/O Extension Blockset or the <i>FPGA1401Tp1 with DS1552 Multi-I/O Module</i> framework from the RTI FPGA Programming Blockset.</p>

The following features are provided by the RTI DS1552 I/O Extension Blockset:

- Analog input channels
 - [ADC 1552 Type 1 Unit](#) on page 101
 - [ADC 1552 Type 2 Unit](#) on page 108
- Analog output channels
 - [DAC 1552 Type 1 Unit](#) on page 129
- Digital input channels for signal measurement
 - [Bit I/O Unit \(DIO 1552 Type 1\)](#) on page 122
 - [PWM Measurement \(PWM2D\) on the DIO 1552 Type 1 Unit](#) on page 205
 - [Frequency Measurement \(F2D\) on the DIO 1552 Type 1 Unit](#) on page 216
- Digital output channels for signal generation
 - [Bit I/O Unit \(DIO 1552 Type 1\)](#) on page 122
 - [PWM Generation \(PWM\) on the DIO 1552 Type 1 Unit](#) on page 156
 - [Square-Wave Signal Generation \(FREQ\) on the DIO 1552 Type 1 Unit](#) on page 165
- When using the RTI FPGA Programming Blockset, you can additionally use:
 - Bidirectional digital channels
 - Sensor supply channel
 - Serial interface (RS232 and RS422/485)

For further information, refer to [Hardware Supported by the RTI FPGA Programming Blockset](#) ([RTI FPGA Programming Blockset Guide](#) ).

DS1554 Engine Control I/O Module With the DS1554 Engine Control I/O Module you can extend your MicroAutoBox II with the following features only by using the *FPGA1401Tp1 with DS1554 Engine Control Module* framework from the RTI FPGA Programming Blockset:

- Analog input channels
- Digital output channels
- Digital bidirectional channels
- Additional channels for specific engine control features, such as:
 - Crankshaft measurement
 - Camshaft measurement
 - Support of knock sensors

For further information, refer to [Hardware Supported by the RTI FPGA Programming Blockset](#) ([RTI FPGA Programming Blockset Guide](#) ).

MicroAutoBox Embedded PC

MicroAutoBox II can be enhanced with the MicroAutoBox Embedded PC. The MicroAutoBox Embedded PC is powered via the MicroAutoBox II power input connector. With the common power input, you can control the power-on and power-off behavior of the entire system. With a connected keyboard and monitor, you can use the MicroAutoBox Embedded PC as host PC.

The MicroAutoBox Embedded PC provides standard connectors for several use cases, for example:

- DVI-I connector for graphical devices
- Ethernet interface 100/1000 Mbit/s (two RJ45 connectors, one LEMO connector)
- USB connectors

MicroAutoBox Embedded DSU can be connected to MicroAutoBox Embedded PC with 6th Gen. Intel® Core™ i7-6822EQ Processor to continuously record and play back large volumes of data for testing purposes.

For more information, refer to:

- [Using MicroAutoBox Embedded PC \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(e662c6fdc679f154c0e75d901761d894_img.jpg\)\)](#)
- [Data Sheet MicroAutoBox Embedded PC with 3rd Gen. Intel® Core™ i7-3517UE Processor \(MicroAutoBox II Hardware Reference !\[\]\(e0657301a840725a62b5d9c03de7d165_img.jpg\)\)](#)
- [Data Sheet MicroAutoBox Embedded PC with 6th Gen. Intel® Core™ i7-6822EQ Processor \(MicroAutoBox II Hardware Reference !\[\]\(c84b30d7d5311af020af6bce6a2c548f_img.jpg\)\)](#)
- [Data Sheet MicroAutoBox Embedded DSU \(MicroAutoBox II Hardware Reference !\[\]\(a9333260d8ffbbfeaa1095df6db7bccd_img.jpg\)\)](#)

Related topics

Basics

[Feature Support..... 25](#)

Overview of Board Revisions

Introduction

MicroAutoBox II was first released in October 2010. The major updates of the DS1401 Base Board and the I/O boards are listed below.

DS1401 base board revisions

Tip

The board revision is printed on a type plate on the bottom of your MicroAutoBox II. You can also read the board revision on the DS1401 Properties page in your experimentation software.

These are the most important DS1401 Base Board revisions of MicroAutoBox II:

Date	Revision	Modifications	Boot Firmware Version	dSPACE Release ¹⁾
Q2/2010	22	First released version of MicroAutoBox II: <ul style="list-style-type: none"> ▪ Processor: PPC750GL ▪ CPU clock: 900 MHz 	<ul style="list-style-type: none"> ▪ 2.7 For MicroAutoBox II 1401/1507 	Using the new components requires at least Release 6.6.

Date	Revision	Modifications	Boot Firmware Version	dSPACE Release ¹⁾
Q4/2011	23	<ul style="list-style-type: none"> Memory: 16 MB Ethernet host interface Ethernet I/O interface Watchdog handling 	<ul style="list-style-type: none"> 3.0 For MicroAutoBox II 1401/1511 3.3 (System PLD version 1.4) 	7.4
		<ul style="list-style-type: none"> Challenge-response monitoring 	<ul style="list-style-type: none"> 3.3 (System PLD version 1.5) 	2016-B
		<ul style="list-style-type: none"> Memory Integrity and Extras 	<ul style="list-style-type: none"> 3.3 (System PLD version 1.6) 	2017-A
		<ul style="list-style-type: none"> Ethernet host interface and Ethernet I/O interface with GBit support 	<ul style="list-style-type: none"> 3.2 	7.2
		<ul style="list-style-type: none"> Onboard pressure sensor 	<ul style="list-style-type: none"> 3.2 (System PLD version 1.3) 	7.3
		<ul style="list-style-type: none"> Onboard acceleration sensor 	<ul style="list-style-type: none"> 3.3 (System PLD version 1.4) 	7.4
Q2/2012	25	Internal Ethernet switch	<ul style="list-style-type: none"> 3.3 (System PLD version 1.4) 	7.4
Q3/2019 (planned)	26	New Flash module	<ul style="list-style-type: none"> 3.10 (System PLD version 2.0.2, Host IF firmware version 5.0.0, Host IF PLD firmware version 6.2.0) 	2019-A

¹⁾ The hardware is delivered independently of a dSPACE Release. This column shows the first dSPACE Release that provides the required boot firmware version.

Note

The table lists the minimum boot firmware version required by the respective board revision to support the new feature. Higher boot firmware versions can be used without problems. With lower boot firmware versions, MicroAutoBox II does not work.

A real-time application for MicroAutoBox II can be executed on newer revisions, if the specified I/O is available and the boot firmware version is at least the firmware version listed above. You can possibly not use the entire memory, see the table above.

I/O board revisions

The following I/O board revisions are of interest:

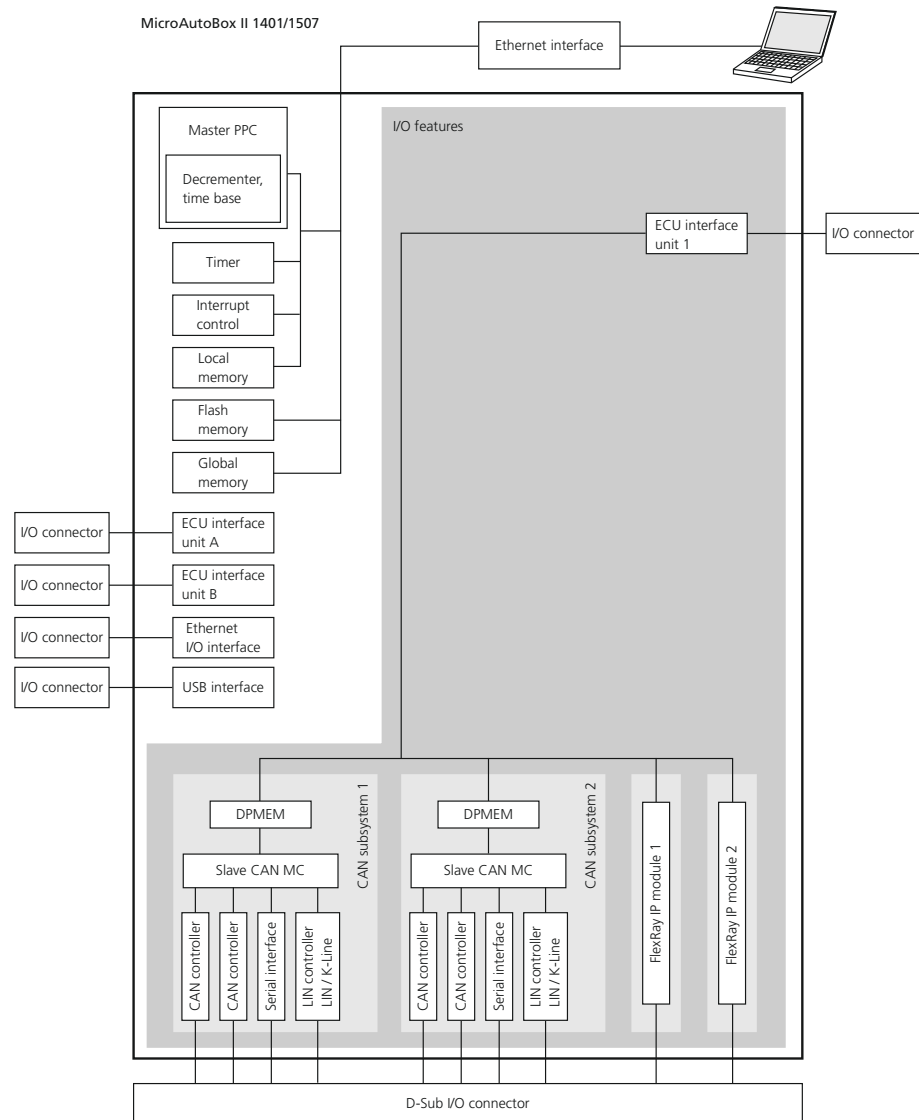
Date	Revision	Features	Boot Firmware Version	dSPACE Release
DS1507				
Q4/2005	01	<ul style="list-style-type: none"> ▪ LIN support ▪ FlexRay support ▪ 2 ECU interfaces 	not relevant	Using DS1507 requires at least Release 4.0.
DS1511				
Q4/2010	03	New I/O board providing: <ul style="list-style-type: none"> ▪ ADC Type 4 ▪ DAC Type 3 ▪ DIO Type 3 ▪ Updated CAN Type 1 	3.0.1	7.0
Q4/2011	03	New I/O features for DIO Type 3: <ul style="list-style-type: none"> ▪ Multichannel PWM signal generation ▪ SENT receiver 	not relevant (DIO Type 3 PLD version 1.3)	7.2
Q2/2012	03	New I/O feature for DIO Type 3: <ul style="list-style-type: none"> ▪ SPI master 	not relevant (DIO Type 3 PLD version 1.4)	7.3
Q4/2015	03	New I/O feature for DIO Type 3: <ul style="list-style-type: none"> ▪ Pulse width measurement (PW2D) 	not relevant (DIO Type 3 PLD version 1.5)	2015-B
Q4/2019 (planned)	07	New Flash module	not relevant (ADC Type 4 PLD version 1.2.3, DIO Type 3 PLD version 1.6.5)	2019-A
DS1513				
Q3/2013	01	New I/O board providing: <ul style="list-style-type: none"> ▪ ADC Type 4 ▪ AIO Type 1 ▪ DIO Type 4 ▪ Updated CAN Type 1 	3.3	2013-B
Q4/2015	01	New I/O feature for DIO Type 4: <ul style="list-style-type: none"> ▪ Pulse width measurement (PW2D) 	not relevant (DIO Type 4 PLD version 1.5)	2015-B
Q3/2019 (planned)	05	New Flash module	not relevant (ADC Type 4 PLD version 1.2.3, AIO Type 1 PLD version 1.1.1, DIO Type 4 PLD version 1.6.5)	2019-A

Date	Revision	Features	Boot Firmware Version	dSPACE Release
DS1514				
Q2/2015	01	New I/O board providing: <ul style="list-style-type: none">▪ Xilinx® Kintex®-7 FPGA XC7K325T▪ Support of DS1552 Multi-I/O Module	3.9	2015-A
Q2/2016	01	Support of DS1554 Engine Control I/O Module	3.9	2016-A

MicroAutoBox II 1401/1507

Functional units

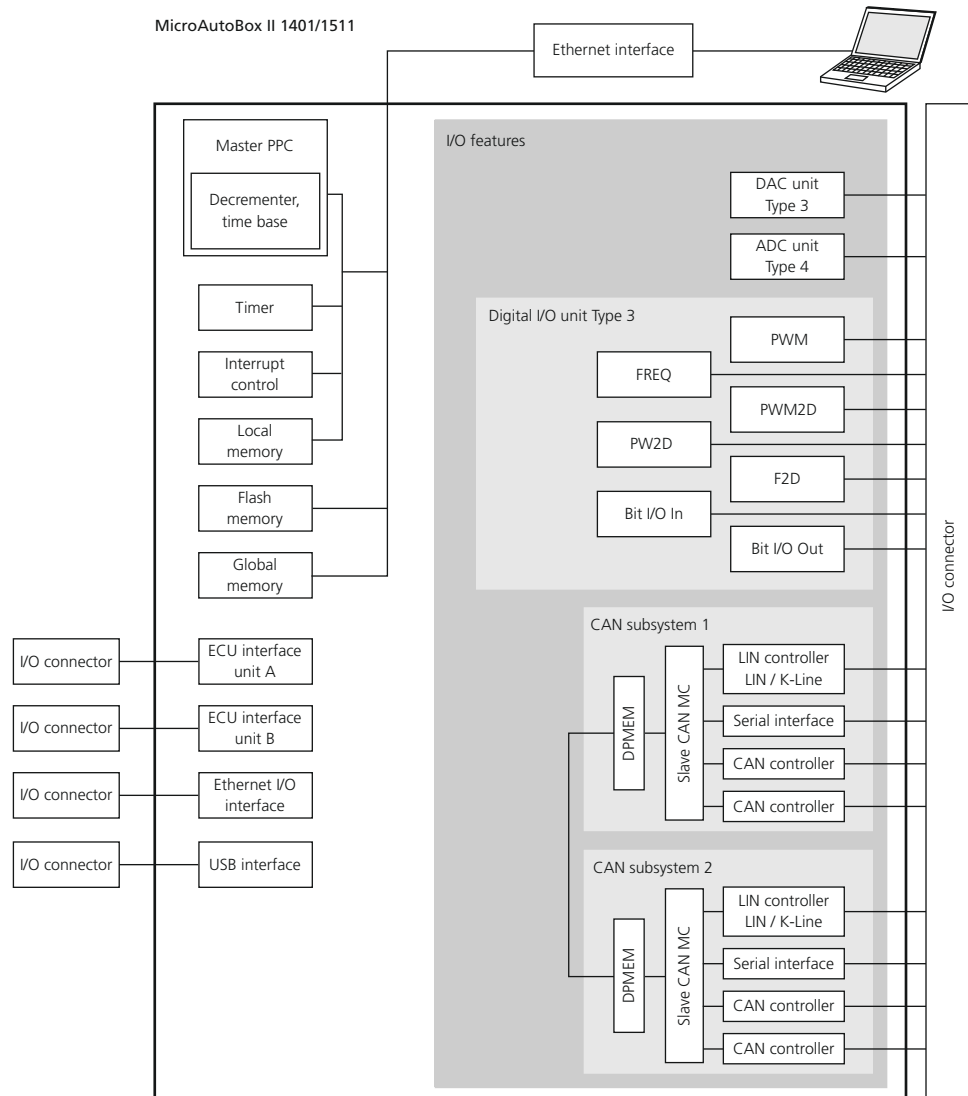
The following illustration shows the functional units of MicroAutoBox II 1401/1507:



MicroAutoBox II 1401/1511

Functional units

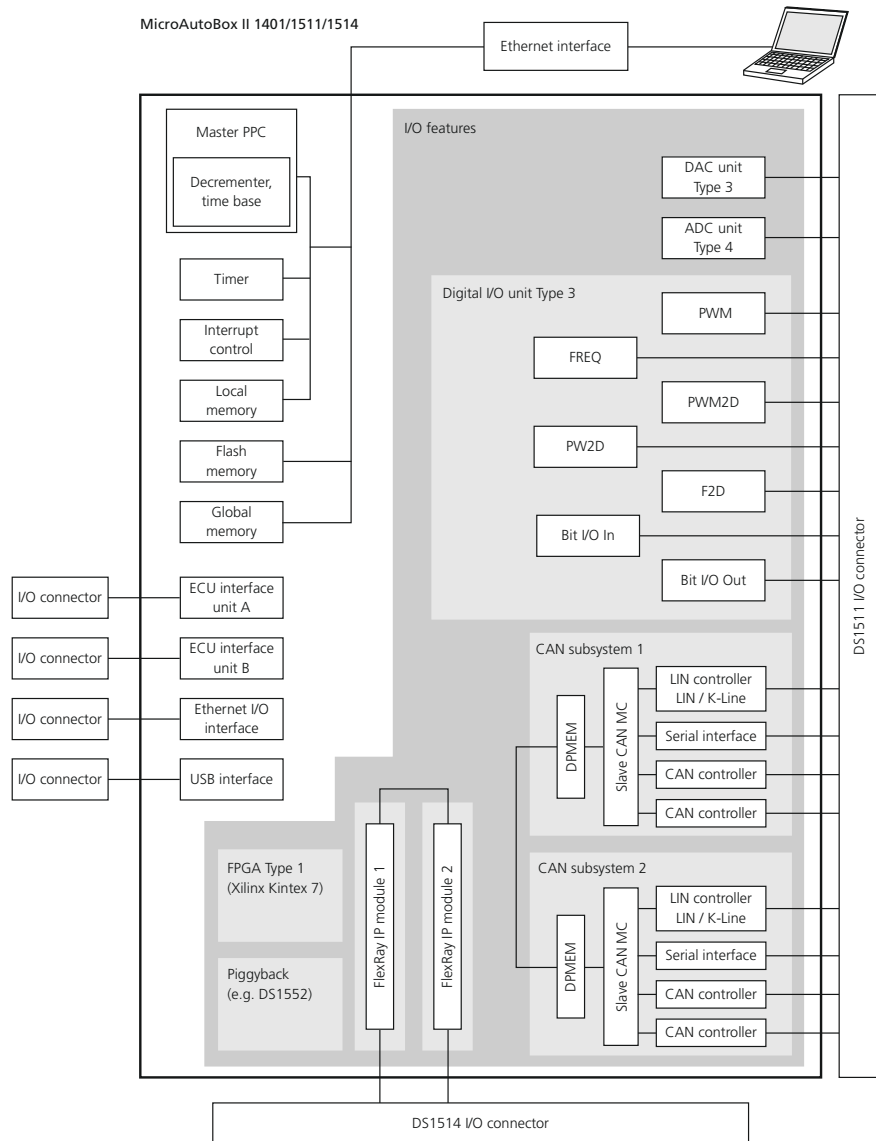
The following illustration shows the functional units of MicroAutoBox II 1401/1511.



MicroAutoBox II 1401/1511/1514

Functional units

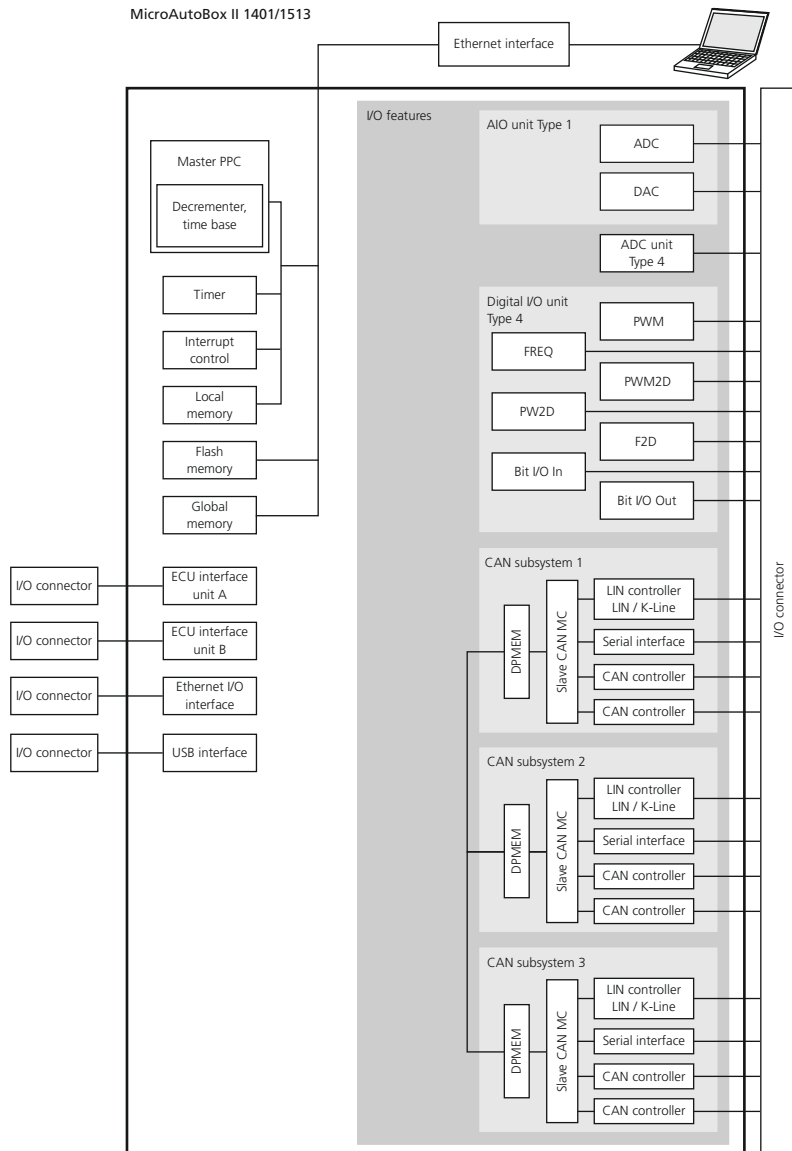
The following illustration shows the functional units of MicroAutoBox II 1401/1511/1514.



MicroAutoBox II 1401/1513

Functional units

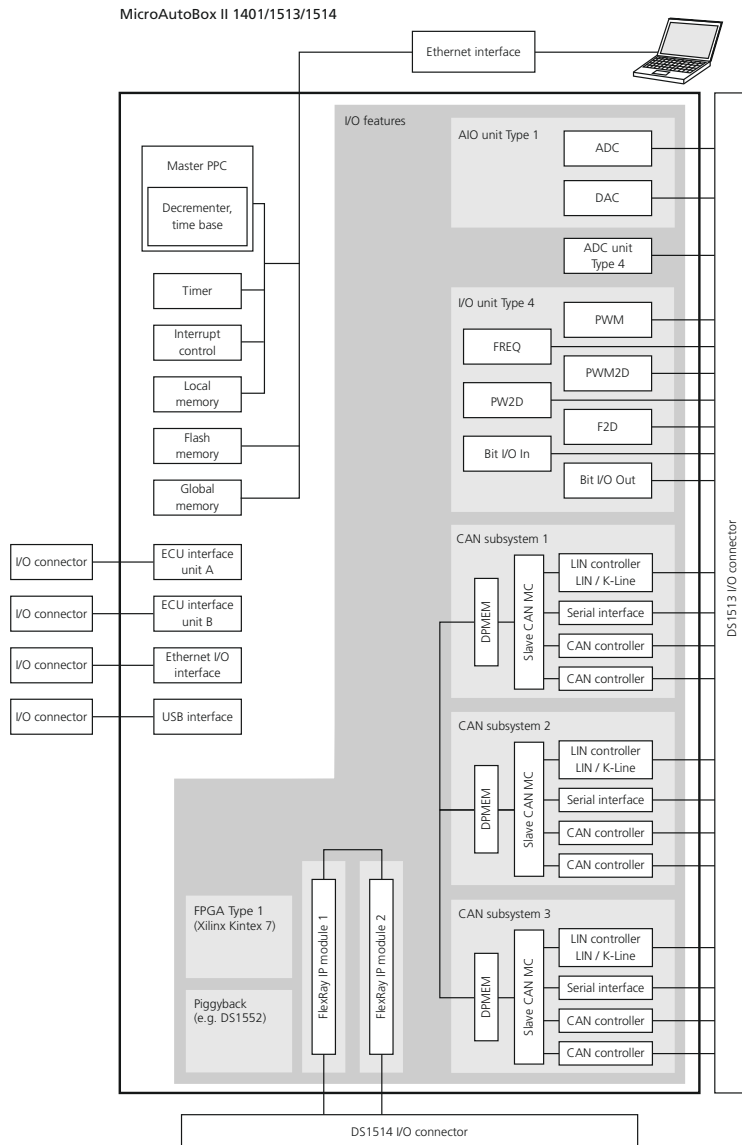
The following illustration shows the functional units of MicroAutoBox II 1401/1513.



MicroAutoBox II 1401/1513/1514

Functional units

The following illustration shows the functional units of MicroAutoBox II 1401/1513/1514.

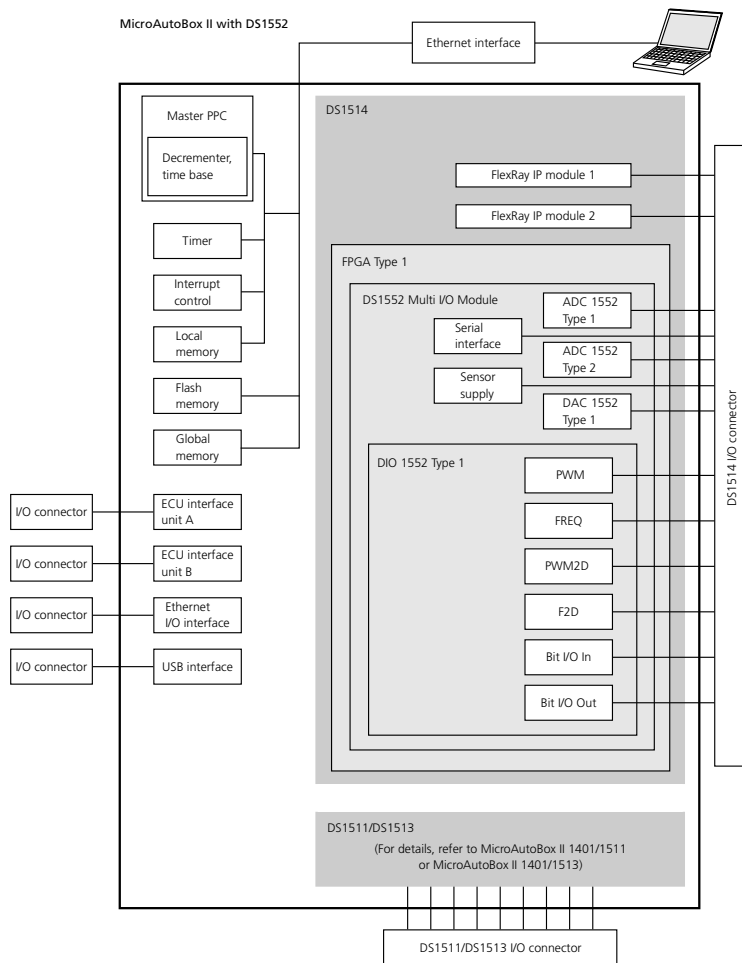


MicroAutoBox II with DS1552 Multi-I/O Module

Functional units of a DS1552 Multi-I/O Module


The DS1514 I/O board of MicroAutoBox II provides connectors to mount a DS1552 Multi-I/O Module on it. The piggyback board provides signal

conditioning for the mapped I/O pins of the DS1514's FPGA module. The following illustration shows the functional units of a DS1552 Multi-I/O Module.



The following MicroAutoBox II variants can be used with the DS1552 Multi-I/O Module:

- MicroAutoBox II 1401/1511/1514
- MicroAutoBox II 1401/1513/1514

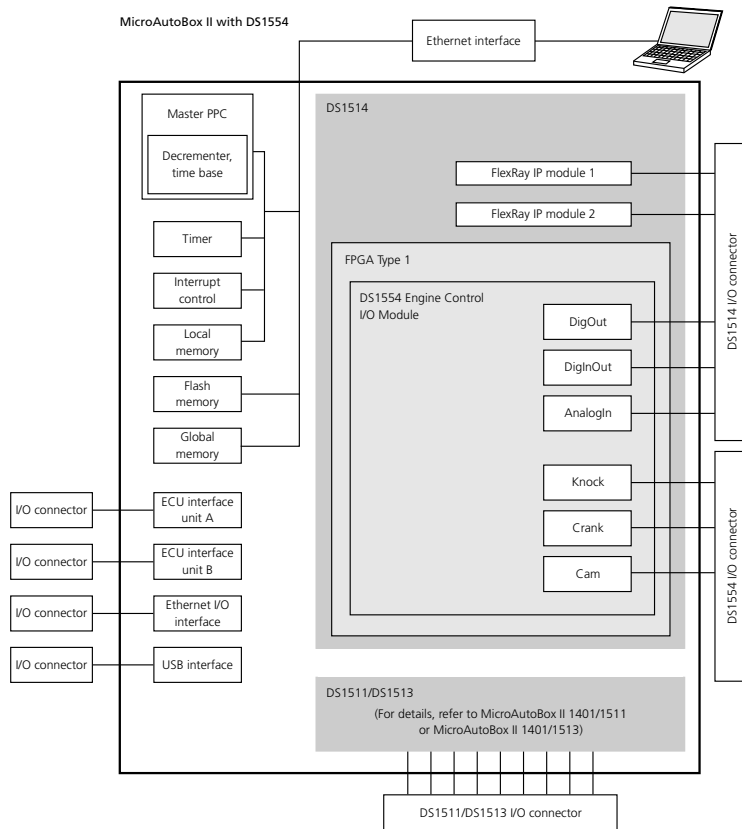
The DS1552B1 variant of the DS1552 Multi-I/O Module has the same features except for the input voltage range of its ADC 1552 Type 1 channels. For details, refer to [Data Sheet DS1552 Multi-I/O Module](#) ([MicroAutoBox II Hardware Reference](#) ).

MicroAutoBox II with DS1554 Engine Control I/O Module

Functional units of a DS1554 Engine Control I/O Module

The DS1514 I/O board of MicroAutoBox II provides connectors to mount a DS1554 Engine Control I/O Module on it. The piggyback board provides signal

conditioning for the mapped I/O pins of the DS1514's FPGA module. The following illustration shows the functional units of a DS1554 Engine Control I/O Module.



The following MicroAutoBox II variants can be used with the DS1554 Engine Control I/O Module:

- MicroAutoBox II 1401/1511/1514
- MicroAutoBox II 1401/1513/1514

For details, refer to [Data Sheet DS1554 Engine Control I/O Module](#) ([MicroAutoBox II Hardware Reference](#) ).

Feature Support

Required MicroAutoBox II hardware

The following table shows the hardware (base board and I/O boards) required for using each feature. You can also see which features are supported by each MicroAutoBox II variant. You can combine only the base boards and I/O boards that are available as MicroAutoBox II variants.

Feature	DS1401 Base Board	MicroAutoBox II with I/O Boards			
		DS1507	DS1511	DS1513	DS1514
MicroAutoBox II Basic Features					
Interrupts	✓	-	-	-	-
Watchdog ¹⁾	✓	-	-	-	-
Challenge-response monitoring ¹⁾	✓	-	-	-	-
Memory Integrity and Extras ¹⁾	✓	-	-	-	-
Onboard pressure sensor	✓	-	-	-	-
Onboard acceleration sensor	✓	-	-	-	-
Power Hold Control	✓	-	-	-	-
Nonvolatile Data Handling	✓	-	-	-	-
Flash Memory-Based Flight Recorder	✓	-	-	-	-
USB Flight Recorder ²⁾	✓	-	-	-	-
MicroAutoBox II I/O Features					
ECU Interface Unit	✓	✓	-	-	-
Ethernet I/O Interface Unit	✓	-	-	-	-
A/D Conversion	-	-	✓	✓	✓ ³⁾
Bit I/O	-	-	✓	✓	✓ ³⁾
D/A Conversion	-	-	✓	✓	✓ ³⁾
PWM Signal Generation With a Variable Period (PWM)	-	-	✓	✓	✓ ³⁾
Square-Wave Signal Generation With a Fixed Duty Cycle (FREQ)	-	-	✓	✓	✓ ³⁾
Multi-Channel PWM Signal Generation (MC_PWM)	-	-	✓	✓	-
PWM Signal Measurement (PWM2D)	-	-	✓	✓	✓ ³⁾
Frequency Measurement (F2D)	-	-	✓	✓	✓ ³⁾
Pulse Width Measurement (PW2D)	-	-	✓	✓	-
Incremental Encoder Interface	-	-	✓	✓	-
Serial Peripheral Interface (SPI)	-	-	✓	✓	-
Single Edge Nibble Transmission (SENT)	-	-	✓	✓	-
CAN Subsystem 1					
Serial Interface	-	✓	✓	✓	-
CAN Support	-	✓	✓	✓	-
LIN Support	-	✓	✓	✓	-
CAN Subsystem 2					
Serial Interface	-	✓	✓	✓	-
CAN Support	-	✓	✓	✓	-

Feature	DS1401 Base Board	MicroAutoBox II with I/O Boards			
		DS1507	DS1511	DS1513	DS1514
LIN Support	-	✓	✓	✓	-
CAN Subsystem 3					
Serial Interface	-	-	-	✓	-
CAN Support	-	-	-	✓	-
LIN Support	-	-	-	✓	-
IP Subsystem					
IP Module Support	-	✓	-	-	✓
FlexRay Support	-	✓	-	-	✓
FPGA Programming					
FPGA Support	-	-	-	-	✓

¹⁾ RTI support via RTI Watchdog Blockset.

²⁾ RTI support via RTI USB Flight Recorder Blockset.

³⁾ Requires DS1552 Multi-I/O Module.

Base Board's Basics

MicroAutoBox II Base Board

Naming convention

Since board revision DS1401-20 the DS1401 base board is denoted *MicroAutoBox II Base board*.

Processing unit

The PowerPC 750 GL microprocessor (PPC) is the main processing unit of MicroAutoBox II. Your control models will be implemented on the PPC. The PPC directly controls the ECU, the ADC and DAC units and accesses the features provided by the digital I/O and CAN subsystems. It also control the interfaces provided by the base board.

The MicroAutoBox II Base board comes with a PowerPC 750 GL, which has the following features:

- Clocked with 900 MHz
- 1 MB Level2 cache
- 16 MB Local RAM
- 16 MB FLASH memory
- 8 MB Global memory (6 MB available for host communication)
- Real-time clock

Note

MicroAutoBox II is supported since Release 6.6. Real-time applications built with earlier releases (but later than Release 4.2) can generally be loaded.

Note the following restrictions:

- Up to Release 6.4, only 8 MB memory can be used.
- Real-time applications using ECU interfaces

A real-time application built with Release 6.5 and earlier that uses the interrupts for ECU 1 and ECU 2 displays an error message if you execute it on MicroAutoBox II with DS1511, DS1513, or DS1514 I/O boards. These interrupts are not automatically forwarded to the ECU A and ECU B interrupts provided by the MicroAutoBox II base board.

You can solve the problem by:

- Using MicroAutoBox II with a DS1507 I/O board
- or
- Recompiling your real-time application

Interfaces

The MicroAutoBox II Base board has the following interfaces:

- Ethernet host interface
- Ethernet I/O interface for connecting external devices controlled by a real-time application
- Two LVDS2 interfaces (ECU interface unit A and ECU interface unit B)
- USB 2.0 interface

Since board revision DS1401-25, the MicroAutoBox II Base board provides an internal Gigabit Ethernet switch. For example, it is required to connect an ECU with DCI-GSI2 to MicroAutoBox II and to the host PC in parallel.

Onboard sensors

Since board revision DS1401-23, the MicroAutoBox II Base board provides onboard sensors for:

- Pressure measurement
- Acceleration measurement

Related topics**Basics**

[MicroAutoBox II Basic Features.....35](#)

References

[Base Board II Blockset \(MicroAutoBox II RTI Reference !\[\]\(bd3b31712ad9bab5a241210fa6925cdd_img.jpg\)\)](#)
[Basic Functionality \(MicroAutoBox II RTLib Reference !\[\]\(882be629d4a853dc90d60f084b0d185d_img.jpg\)\)](#)

MicroAutoBox II Application Start

Introduction

After power-up, MicroAutoBox II boots automatically and executes the boot firmware located in the on-board flash memory. Afterwards, the real-time application can be used.

Where to go from here

Information in this section

[Boot Firmware.....30](#)

[Running an Application from the Global Memory.....32](#)

In general, you download the application from the host PC to the MicroAutoBox II's global memory. After the download, the application is started.

[Running an Application from the Flash Memory.....33](#)

If you want an application to be started automatically after power-up, you have to load it to the MicroAutoBox II's flash memory.

Boot Firmware

Introduction


After power-up MicroAutoBox II boots from the flash memory, which holds the pre-installed boot firmware.

Characteristics of the boot firmware

The boot firmware carries out the following steps:

- It determines the I/O modules connected to the available I/O boards of the MicroAutoBox II.
- It determines whether a user-specific application exists in the global memory or the flash memory. If this is the case, the application is started. An application in the global memory takes precedence over an application in the flash memory.
- If neither the global nor the flash memory contains an application, the boot firmware only collects information about the connected MicroAutoBox II. This information is passed to the Platforms/Devices controlbar of ControlDesk, which displays the system topology.

Note**Use the latest boot firmware**

The boot firmware is available on the dSPACE Release DVD. You should regularly check the availability of new boot firmware at <http://www.dspace.com/go/firmware>. To update the firmware, use the dSPACE Firmware Manager, refer to [Firmware Manager Manual](#) .

Problems Related to the boot firmware

If MicroAutoBox II is not able to boot correctly, you have to restore the boot firmware by using the Firmware Manager.

For further information, refer to [Using the Command Line Interface for Firmware Management \(Firmware Manager Manual !\[\]\(8d0f0e0fe25b320c33272c52aec1fbca_img.jpg\)\)](#).

Further firmware components

A MicroAutoBox II consists of various hardware components. Some of them use separate firmware to provide a specific functionality.

Firmware Component	Meaning
Firmware for the main hardware components	
Boot firmware	Provides essential functions to access the hardware system connected to the host PC. It also contains functions to provide a hardware inventory and to load and start a real-time application.
Host IF firmware	Provides functions that allow the communication between the host PC and dSPACE real-time hardware that has an Ethernet-based host interface.
Host IF PLD firmware	Provides functions for the programmable logic device that controls the host interface communication.
System PLD firmware	Provides functions for the programmable logic device that offers system functions, for example, watchdog handling or accessing on-board sensors.
Firmware for additional hardware components	
AIO TYPE 1 PLD firmware	Provides functions for accessing analog input and output signals via the programmable logic device of the AIO Type 1 module. Relevant for the DS1513 I/O board of MicroAutoBox II.
ADC TYPE 4 PLD firmware	Provides functions for accessing analog input signals via the programmable logic device of the ADC Type 4 module. Relevant for the DS1511 and DS1513 I/O boards of MicroAutoBox II.
CAN TYPE 1 firmware	Provides functions for CAN communication via the CAN Type 1 module.
DIO TYPE 3 PLD firmware	Provides functions for accessing digital input and output signals via the programmable logic device of the DIO Type 3 module. Relevant for the DS1511 I/O board of MicroAutoBox II.

Firmware Component	Meaning
DIO TYPE 4 PLD firmware	Provides functions for accessing digital input and output signals via the programmable logic device of the DIO Type 4 module. Relevant for the DS1513 I/O board of MicroAutoBox II.
FPGA TYPE 1 PLD firmware	Provides basic functions for initializing and accessing the FPGA Type 1 module. Relevant for the DS1514 I/O board of MicroAutoBox II.
DS1552ExtendedIO	Provides the functionality of the DS1552 Multi-I/O Module for signal conditioning of analog and digital inputs and outputs. The firmware is loaded to the FPGA Type 1 module by using the DS1401UpdateExtIO command in a Command Prompt window. This firmware is not supported by the Firmware Manager or the Update Firmware Wizard in ControlDesk.

After updating the firmware of MicroAutoBox II, you have to turn off MicroAutoBox II. After a restart the firmware changes take effect.

Related topics

Basics

[Introduction \(Firmware Manager Manual !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)](#))

Running an Application from the Global Memory

Introduction

In general, you download the application from the host PC to the MicroAutoBox II's global memory. After the download, the application is started. This is the default behavior for both the dSPACE experiment software and RTI.

To stop and restart the application you have to reload it again, for example, via ControlDesk's Real-Time Application - Reload command.

Note

If you switch off the MicroAutoBox II, the contents of the global memory will be lost. If you download/reload an application, the previous contents of the global memory will be overwritten.

For information on handling real-time applications, refer to [Handling Real-Time Applications with ControlDesk \(DS100x, DS110x, MicroAutoBox II, MicroLabBox – Software Getting Started !\[\]\(4688aadfd656ded00cd6bdfae55089a9_img.jpg\)](#)).

Related topics**Basics**

[Troubleshooting..... 397](#)

References

[Real-Time Application - Reload \(ControlDesk Platform Management !\[\]\(d66ff64371a51729ac8c1cdaa685ba6f_img.jpg\)](#))

Running an Application from the Flash Memory

Introduction

If you want an application to be started automatically after power-up, you have to load it to the MicroAutoBox II's flash memory. This is possible via RTI, RTI-MP and the dSPACE experiment software. Autobooting allows you to use MicroAutoBox II as a stand-alone system without a connection to the host PC.

Basics

You can use up to 15 MByte of the flash memory for your application. After the download has finished, the application will be copied to the global memory, overwriting any existing application, and started automatically.

Whenever you switch on the MicroAutoBox II, the boot firmware will copy the application from the flash memory to the global memory and start it afterwards, regardless of whether the board is connected to the host PC or not.

You can stop and restart the application by resetting the MicroAutoBox II, for example, by switching off the power and turning it on again. If you use the dSPACE experiment software to stop and reload the application, it will first be reloaded from the host PC to the flash memory, and then from the flash memory to the global memory. Then the application will be started.


You can clear an application from the flash memory via the dSPACE experiment software. Flash operations can be very time consuming: clearing or reprogramming usually takes longer than 45 seconds.

Note

If you switch off the MicroAutoBox II, the contents of the global memory will be lost. If you switch on the power again, the contents of the flash memory will be copied to the global memory and started.

Related topics

Basics

Base Board's Basics.....	28
Handling Real-Time Applications with ControlDesk (DS100x, DS110x, MicroAutoBox II, MicroLabBox – Software Getting Started )	
Troubleshooting.....	397

MicroAutoBox II Basic Features

Introduction

The base board from MicroAutoBox II provides some basic features that are required for general implementation of real-time applications such as timer services or interrupt handling.

Where to go from here

Information in this section

Interrupt Handling.....	37
General information on the interrupt handling provided by the MicroAutoBox II Base board.	
Word-Based Subinterrupt Handling.....	40
Information on how to handle the subinterrupts generated by an ECU.	
Timer Services.....	47
MicroAutoBox II provides incremental and decremental timer services.	
Safety Functions.....	49
MicroAutoBox II provides functions for monitoring the execution of tasks and other functional entities in your real-time application according to safety measurements.	
Nonvolatile Data Handling.....	61
The MicroAutoBox II Base board provides memory units that can be used for nonvolatile data handling.	
Flight Recorder.....	65
MicroAutoBox II offers two different flight recorders for long-term data acquisition.	
Power Hold Control.....	76
You can control MicroAutoBox II's power hold feature via Simulink model or real-time application. This way the MicroAutoBox II is shut down after all termination processes have finished.	
Onboard Sensors.....	78
MicroAutoBox provides onboard sensors to measure the board's acceleration on three axes and the air pressure.	

Information in other sections

Hardware Concept.....	12
-----------------------	----

Interrupt Handling

Introduction

General information on the interrupt handling provided by the MicroAutoBox II Base board.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓	–	–	–	–

Basics on Interrupt Handling

Basic characteristics

The MicroAutoBox II Base board provides five basic interrupts from which three interrupts are used for the host access, the timer and the decremter. The other two interrupts are reserved.

There are 27 hardware interrupts available for the I/O boards.

Depending on the MicroAutoBox II variant used, different interrupts are available.

There are different methods on interrupt assignment for the various MicroAutoBox II units. For example, the ECU interrupt and the CAN interrupt are split into subinterrupts. These interrupts can be masked.

Interrupts can be globally enabled and disabled. Only enabled interrupt sources generate an interrupt on the base board.

Interrupts provided by Digital I/O Unit Types 3 and 4

The MicroAutoBox II Base board provides 4 interrupt lines for the communication between the base board and the DIO Type 3 module or the DIO Type 4 module.

Interrupt generation can be triggered by a rising edge, a falling edge or both. Each channel of each port can be used to generate an interrupt. The minimum pulse length to detect an interrupt is 2 μ s.

If you are using the SENT feature, the interrupt generation can be triggered by a received message. Using the SPI feature, the interrupt generation can be triggered by a completed transmission of an SPI cycle.

The 4 interrupt lines are processed in different ways:

Interrupt Line	Processing Details
1, 2	The interrupt line is processed for one single input or output channel. This does not require subinterrupt handling that would increase the latency of the interrupt service.
3	The interrupt line is processed for each channel that is specified to be connected to this interrupt line. Only <i>input</i> channels are allowed.
4	The interrupt line is processed for each channel that is specified to be connected to this interrupt line. Only <i>output</i> channels are allowed. For example, you can use the interrupt of a PWM output channel to start an A/D conversion for the current measurement.

Note

Interrupt lines 1 and 2 are not supported by RTI yet.

Depending on the available I/O board, you have to use the related RTLib functions or RTI blocks for accessing the interrupt lines.

With RTLib, you can use the `dio_tp3_single_source_int_mode_set` or `dio_tp4_single_source_int_mode_set` function to initialize interrupt lines 1 and 2, and the `dio_tp3_multi_source_int_mode_set` or `dio_tp4_multi_source_int_mode_set` function to initialize interrupt lines 3 and 4. Handling the subinterrupts of lines 3 and 4 requires the `dssint_xxx` functions, refer to [Subinterrupt Handling \(MicroAutoBox II RTLib Reference\)](#).

With RTI, the DIO_TYPE3_HWINT_Blx block or the DIO_TYPE4_HWINT_Blx block is using the interrupt lines 3 and 4.

Interrupts provided by the DS1552 Multi-I/O Module

The MicroAutoBox II Base board provides one interrupt line for the communication between the base board and the FPGA Type 1 module on the DS1514. The interrupts of the ADC 1552 Type 1 unit and of the DIO 1552 Type 1 unit are processed by the interrupt handling implemented on the FPGA Type 1 module and then transferred to the base board.

Subinterrupts of the ADC 1552 Type 1 unit The interrupt implementation of the FPGA Type 1 module provides three entries for the ADC 1552 Type 1 unit. Each entry represents one of the available interrupt types and references another interrupt array containing the interrupt states of each ADC channel. For further information, refer to [ADC 1552 Type 1 Unit](#) on page 101.

With RTLib, you can access the interrupt implementation for each of the interrupt types, refer to [ADC 1552 Type 1 Unit of the DS1552 Multi-I/O Module \(MicroAutoBox II RTLib Reference\)](#). Handling the interrupts requires the `dssint_xxx` functions, refer to [Subinterrupt Handling \(MicroAutoBox II RTLib Reference\)](#).

With RTI DS1552 I/O Extension Blockset, the ADC1552_TP1_HWINT_BLx block is using the three ADC 1552 Type 1 addresses in the interrupt implementation, refer to [ADC1552_TP1_HWINT_BLx \(MicroAutoBox II RTI Reference\)](#).

Subinterrupts of the DIO 1552 Type 1 unit The interrupt implementation of the FPGA Type 1 module provides two entries for the DIO 1552 Type 1 unit. One entry serves the interrupts of the digital input channels, the other entry serves the interrupts of the digital output channels.

Interrupt generation can be triggered by a rising edge, a falling edge or both. Each channel can be used to generate an interrupt. The minimum pulse length to detect an interrupt is 2 μ s.

With RTLib, you can access the interrupt implementation for the digital input channels and the digital output channels separately, refer to [Interrupt Handling of the DIO 1552 Type 1 Unit of the DS1552 Multi-I/O Module \(MicroAutoBox II RTLib Reference\)](#). Handling the interrupts requires the `dssint_xxx` functions, refer to [Subinterrupt Handling \(MicroAutoBox II RTLib Reference\)](#).

With RTI, the DIO1552_TP1_HWINT_BLx block is using the DIO 1552 Type 1 addresses in the interrupt implementation, refer to [DIO1552_TP1_HWINT_BLx \(MicroAutoBox II RTI Reference\)](#).

If you use the DS1552 Multi-I/O Module with the RTI FPGA Programming Blockset, the interrupts that you configured by using the FPGA_INT_BLx block are generated on the FPGA Type 1 module and then transferred to the base board. For further information on implementing interrupts in your FPGA application, refer to [How to Trigger Interrupt-Driven Processor Tasks \(RTI FPGA Programming Blockset Guide\)](#).

Interrupts provided by the DS1554 Engine Control I/O Module

You can use the DS1554 Engine Control I/O Module with the RTI FPGA Programming Blockset only. The interrupts that you configured by using the FPGA_INT_BLx block are generated on the FPGA Type 1 module and then transferred to the base board. MicroAutoBox II provides one interrupt line for the communication between the base board and the FPGA Type 1 module on the DS1514. For further information on implementing interrupts in your FPGA application, refer to [How to Trigger Interrupt-Driven Processor Tasks \(RTI FPGA Programming Blockset Guide\)](#).

ECU (sub-)interrupts

The ECU-to-PPC interrupt can be split into 16 subinterrupts via the ECU Software Porting Kit (refer to [ECU Software Porting Kit](#) on page 142).

Interrupt-driven subsystems in RTI

You can use interrupts or subinterrupts to trigger interrupt-driven subsystems of your Simulink model. For further information on interrupt-driven subsystems, refer to the Simulink user documentation.

Handcoded models

If you use handcoded models, you have to provide the interrupt handling yourself.

Word-Based Subinterrupt Handling

Introduction

Information on how to handle the subinterrupts generated by an ECU.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓	✓	–	–	–

Where to go from here

Information in this section

[Basics on Word-Based Subinterrupt Handling](#).....40

The Word-Based Subinterrupt module is designed to implement the subinterrupt features on the real-time processor (RTP) side.

[Basics on the Alive Mechanism](#).....41

Correct subinterrupt handling requires that the corresponding partner system is running ("alive").

[Basics on Subinterrupt Handling](#).....44


Provides information on the subinterrupt handling.

Basics on Word-Based Subinterrupt Handling

Introduction

The Word-Based Subinterrupt module is designed to implement the subinterrupt features on the real-time processor (RTP) side, that is your dSPACE real-time system, for example, a DS1007-based modular system or MicroAutoBox II.

Note

To implement a subinterrupt sender on the ECU without intervention to the interrupt state of the ECU, use the [ECU Software Porting Kit](#) ([MicroAutoBox II RTLib Reference](#) ).

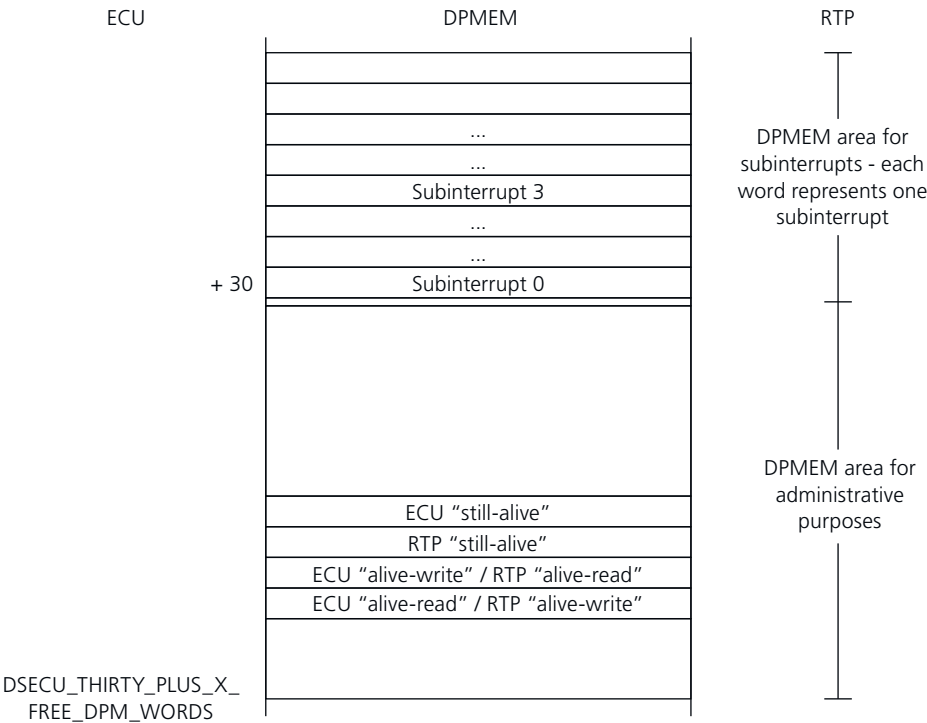
Features of the Word-Based Subinterrupt Handling module

- The Word-Based Subinterrupt Handling module provides the following features:
- It lets you trigger and handle multiple subinterrupts using a single hardware interrupt line.
 - It lets you specify multiple subinterrupts as *pending* at the receiver.
 - It lets you transmit and dispatch interrupts between an ECU and a dSPACE RCP system.
 - It lets you define interrupt senders/receivers to transmit subinterrupts.

DPMEM usage

The Word-Based Subinterrupt Handling module requires $30 + x$ (with x = number of subinterrupts to be used) successive DPMEM addresses. The first 30 words are used for the alive mechanism, version information and other administrative purposes.

The following illustration shows the usage of the reserved DPMEM areas and the location of the words used for the alive mechanism and the subinterrupt handling:



Basics on the Alive Mechanism

Introduction

Correct subinterrupt handling requires that the corresponding partner system is running ("alive").

Alive state

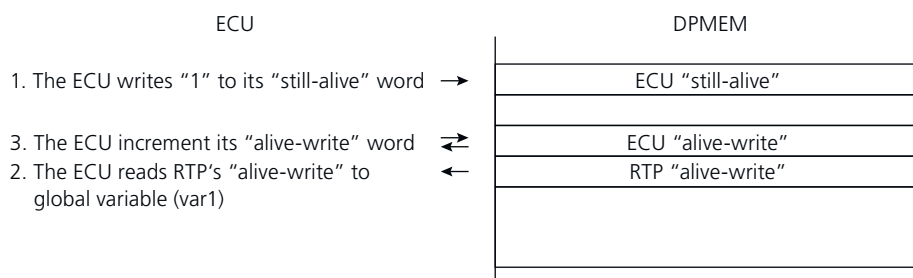
The partner system is supposed to be "alive" if the contents of the own "alive-read" word is changed. The own "alive-read" is equal to the partner's "alive-write" word. After the alive state is recognized, the "still-alive" word of the partner is evaluated.

Startup function

The alive startup function

- sets the own "still-alive" word
- reads the partner's "alive-write" word to initialize its compare value which is used to recognize changes of this word
- increments its own "alive-write" word for the partner

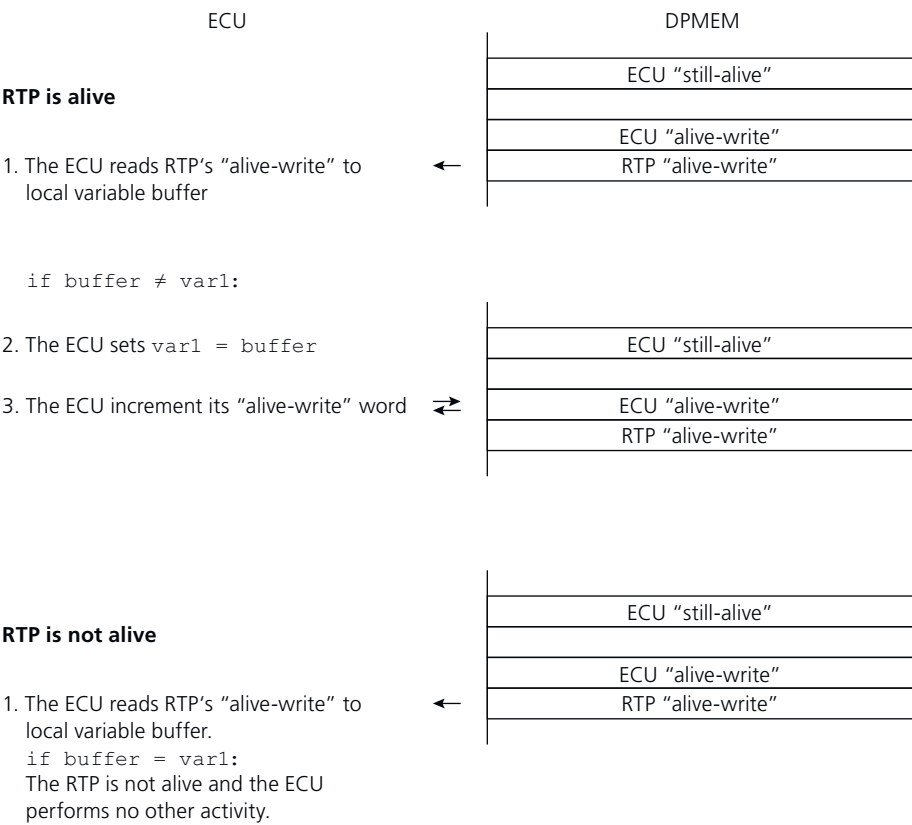
The following illustration shows the activities from the ECU side (the RTP uses its DPMEM addresses to perform corresponding actions):

**Alive function**

The alive function has to be called repetitively to compare the partner's "alive-write" word with the internal compare value. If the value has changed, the partner is recognized as alive. Only in this case, the own "alive-write" word is incremented again.

In addition to this, the internal compare value is updated to the new value. After the partner is recognized as alive, only the "still-alive" word is checked.

The following illustration shows the activities from the ECU side. The RTP uses its DPMEM addresses to perform analog actions.

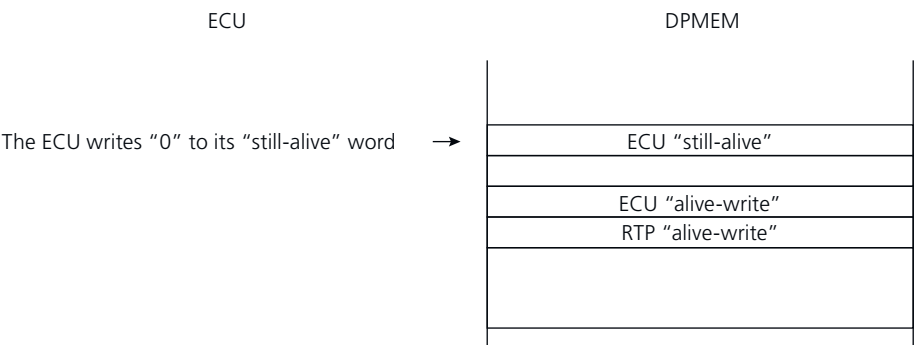


Suspend function

- If the ECU or the dSPACE RCP system intends to go to "not alive", it must
- call the suspend function which clears the "still-alive" word and
 - must not call the alive function any longer.


The alive function of the other side notices that the "still-alive" word is cleared and checks the partner's "alive-write" word for changes. This allows the system to recognize when the partner goes to the alive state again.

To revive a system you have to call the startup function once before the alive function is called repetitively again.



For detailed information on the functions, refer to [Word-Based Subinterrupt Handling \(MicroAutoBox II RTLib Reference !\[\]\(dfbd6b3763a6d1d9afaa974f64e2e4b5_img.jpg\)](#)).

Related topics

Basics	
ECU Software Porting Kit.....	142
References	
Word-Based Subinterrupt Handling (MicroAutoBox II RTLib Reference )	

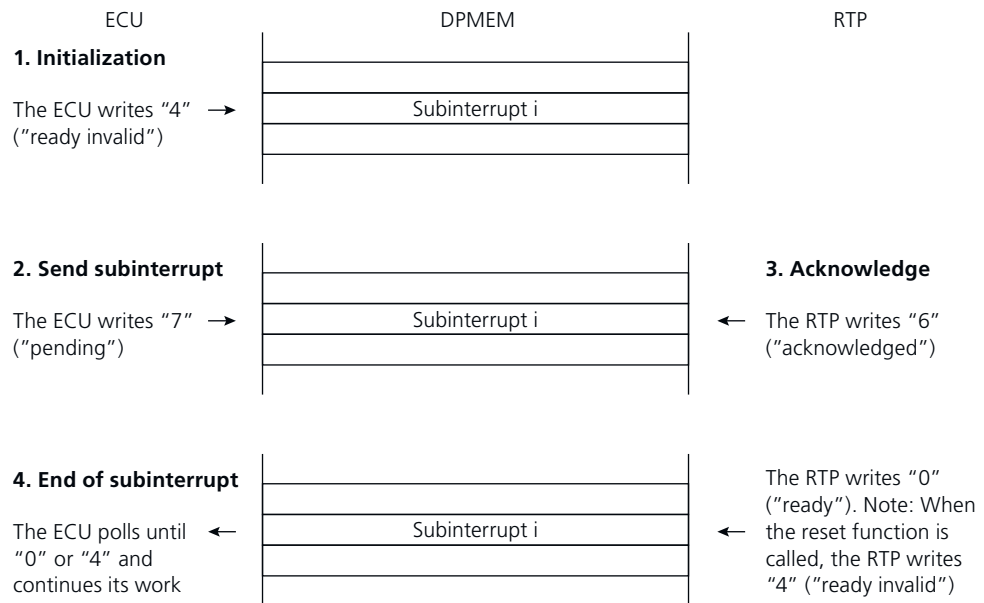
Basics on Subinterrupt Handling

Introduction

To implement word-based subinterrupt handling, you must consider the following working steps.

Subinterrupt handling overview

The following illustration shows the subinterrupt handling – for example, synchronous bypassing – in relation to the DPMEM addresses used:



Initialization

Subinterrupt sender and receiver initialize the subinterrupt words – for the defined number of subinterrupts – in the DPMEM with the value "ready invalid" (= 4).

Pending subinterrupt

If the receiver system is alive, the sender's send function sets the subinterrupt state to "pending" (= 7) and triggers the hardware interrupt.

Acknowledge

The receiver's interrupt handling performs the following steps:

- It calls the acknowledge function to acknowledge the hardware interrupt by reading the DPMEM acknowledge address.
- It sets all pending subinterrupts to the state "acknowledged" (= 6).

Decoding

The information about the acknowledged subinterrupts is passed to the *decode function* via an internal data structure. The decode function must be called repetitively to return every time the number of the subinterrupt that was acknowledged before.

If it returns `-1` there is no more subinterrupt to handle.

End of subinterrupt

After the subinterrupt task is finished, the *end-of-subinterrupt (EOSI) function* has to be called. It sets the subinterrupt state to "ready" (= 0). This is the signal for the sender that the data can be read. The function that polls for the end-of-subinterrupt returns the "valid" state.

Reset

If the receiver calls the reset function for a subinterrupt, its state is set to "ready invalid" (= 4) again as if it was pending before. The sender's poll function returns the "invalid" state and the program can react appropriately.

Related topics**References**

[ECU Software Porting Kit \(MicroAutoBox II RTLib Reference !\[\]\(10f8862fc183b400327470ea85afe9ae_img.jpg\)\)](#)

[Word-Based Subinterrupt Handling \(MicroAutoBox II RTLib Reference !\[\]\(e1d6102fe77919492c04879c8450f1f5_img.jpg\)\)](#)

Timer Services

Introduction MicroAutoBox II provides incremental and decremental timer services.

Hardware requirements Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓	–	–	–	–

Where to go from here	Information in this section
	Timer..... 47
	Decrementer..... 48
	Time Base Counter..... 48

Timer

Characteristics The DS1401 base board includes a timer that can be used asynchronously to the sample base rate. It is an 32-bit upcounter with a prescaler. When the value of the counter matches the value of the timer compare register, an interrupt will be generated.

RTI/RTLib support You can access the timer via RTI blockset and RTLib. For details, see:

- [Timer Interrupt Block \(RTI and RTI-MP Implementation Reference !\[\]\(2b376d1a92330ab09dad2665d2f89bf5_img.jpg\)](#))
- [Timer \(MicroAutoBox II RTLib Reference !\[\]\(fcaee6d397c07452e54229b176f1295d_img.jpg\)](#))

Related topics**References**

[Decrementer \(MicroAutoBox II RTLib Reference !\[\]\(3d8c13c92b853674f749aac6fa869926_img.jpg\)\)](#)
[Timer \(MicroAutoBox II RTLib Reference !\[\]\(ce455c990c00145a2dda1d9a310cb682_img.jpg\)\)](#)

Decrementer

Characteristics

The PowerPC built-in Decrementer is used for periodic timer events such as the sample base rate. The decrementer is a 32-bit downcounter generating an interrupt when the timer is down to -1. At -1 the counter is reloaded with the counter value set by `ds1401_decrementer_period_set`.

RTI/RTLib support

You can access the decrementer via RTI blockset and RTLib. For details, see:

- [Timer Interrupt Block \(RTI and RTI-MP Implementation Reference !\[\]\(4f6d8a8b127300a02d56d34d01423d15_img.jpg\)\)](#)
- [Decrementer \(MicroAutoBox II RTLib Reference !\[\]\(7e3d1ad67bf2d7a17700a66d1a313f91_img.jpg\)\)](#)

Related topics**References**

[Decrementer \(MicroAutoBox II RTLib Reference !\[\]\(4688aadfd656ded00cd6bdfae55089a9_img.jpg\)\)](#)
[ds1401_decrementer_period_set \(MicroAutoBox II RTLib Reference !\[\]\(3f972bf2e2155492661f419a89867457_img.jpg\)\)](#)
[ds1401_decrementer_read \(MicroAutoBox II RTLib Reference !\[\]\(a1a2b3c7224136cfe1cea8a4ee7f4ea8_img.jpg\)\)](#)
[ds1401_decrementer_set \(MicroAutoBox II RTLib Reference !\[\]\(087c28ab5fccf267392f2e1cdaf42f77_img.jpg\)\)](#)

Time Base Counter

Characteristics

The PowerPC built-in 64-bit time base counter is divided into two 32-bit registers – the Upper Time Base Register (TBRU) and the Lower Time Base Register (TBRL). You can use the time base registers for time measurements.

RTI/RTLib support

You can access the time base counter via RTLib only. For details, see:

- [Time Base Counter \(MicroAutoBox II RTLib Reference !\[\]\(3f0880663a5184fb79b391f4bd12984c_img.jpg\)\)](#)

Related topics**References**

[Time Base Counter \(MicroAutoBox II RTLib Reference !\[\]\(a05a1b59a958625e01d770867ed2a42e_img.jpg\)\)](#)

Safety Functions

Introduction

MicroAutoBox II provides functions for monitoring the execution of tasks and other functional entities in your real-time application according to safety measurements.

Hardware requirements

Supported MicroAutoBox II hardware:

Safety Functions	DS1401 Base Board ¹⁾	MicroAutoBox II with I/O Boards			
		DS1507	DS1511	DS1513	DS1514
Watchdog Handling	✓ ²⁾	—	—	—	—
Challenge-Response Monitoring	✓ ³⁾	—	—	—	—
Memory Integrity and Extras	✓ ⁴⁾	—	—	—	—

¹⁾ Supported as of board revision DS1401-22.

²⁾ System PLD firmware version 1.4.0 and later required.

³⁾ System PLD firmware version 1.5 and later required.

⁴⁾ System PLD firmware version 1.6 and later required.

Where to go from here

Information in this section

Basics on Implementing Safety Functions.....	49
Basics on Watchdog Handling.....	50
Basics on Challenge-Response Monitoring.....	52
Basics on Memory Integrity and Extras.....	57

Basics on Implementing Safety Functions

Introduction

MicroAutoBox II provides several features to implement real-time applications with safety aspects.

Safety measures

Your software might run into an error or your hardware, e.g., external devices connected to your real-time application, might become defective during execution. To make your application's behavior safe in such a case, you can apply various safety mechanisms.

The first safety measure is the implementation of functional reactions to errors that are known or intended. However, the real-time application might show an uncontrollable behavior if an unexpected error occurs.

The second safety measure is achieved through the watchdog handling. If the monitored task or subsystem is no longer able to retrigger its watchdog within the specified time, the real-time application can be set to a safe state and the hardware can be reset. Refer to [Basics on Watchdog Handling](#) on page 50.

The third safety measure is the challenge-response monitoring. The monitored entity, which can be a task, a subsystem, or any other functional unit, does not only have to react within the specified time but also with a specific response value. By generating the response value dynamically within the monitored entity, you can implement various use scenarios for monitoring, such as checking the execution order of functions in the monitored entity. The challenge-response monitoring is executed on a different hardware component as the real-time application to achieve a high safety level. Refer to [Basics on Challenge-Response Monitoring](#) on page 52.

The fourth safety measure is the monitoring of the supply voltage of the real-time hardware and the integrity of the memory sections used by the real-time application. Additionally, you can use custom code to trigger a failure action. Refer to [Basics on Memory Integrity and Extras](#) on page 57.

Basics on Watchdog Handling

Introduction

With a watchdog, you can monitor a task in your real-time application. You can detect whether a task has stopped or is running in an endless loop.

Specifying watchdogs

Initializing the watchdog feature enables and initializes the hardware watchdog on which all the software watchdogs are based. The hardware watchdog monitors the reactivity of your real-time application as a whole. If any task in your real-time application hangs, the background service will not service the hardware watchdog. This leads to a reaction of the hardware watchdog, for example, it will reset the hardware system.

The hardware watchdog is initialized by using the `WATCHDOG_SETUP` block or the `ds1401_wd_init` function.

To monitor a specific task in your real-time application, you have to specify a task-specific software watchdog by adding further tasks to the **WATCHDOG_SETUP** block or using the **ds1401_wd_task_monitoring_init** function.

It is guaranteed that you can initialize up to 20 software watchdogs. Memory for further software watchdogs is dynamically allocated as long as enough memory is available.

The watchdogs for the background service and the tasks can be managed individually. A watchdog's most important parameter is the time it has to wait for the monitored task to react. This is the timeout limit used to configure the watchdog timer of each watchdog separately.

It makes sense to set the timeout limit of the hardware watchdog to the lowest timeout limit of all the tasks registered with the watchdog feature. This guarantees that the requirements of the task with the shortest timeout interval are met. However, any running task might then prevent the background service from executing. If a task has an execution time greater than the timeout limit of the background task, the hardware watchdog resets the system or generates an interrupt.

Note

MicroAutoBox II and its watchdog feature are not designed to be used for safety-critical applications.

Retriggering watchdog timers

The timer of the hardware watchdog is automatically retriggered by the background service of your real-time application. The timers of the software watchdogs have to be retriggered explicitly within the tasks by using the **WATCHDOG_RETRIGGER_BLx** block or the **ds1401_wd_task_retrigger_perform** function.

To avoid retriggering a watchdog timer accidentally when you use RTLib, you have to perform it in two steps and use a special authentication code. You have to call the **ds1401_wd_task_retrigger_arm** function beforehand.

If retriggering was not executed within the specified timeout limit, the watchdog reacts by generating a machine check (MCP) interrupt to start a previously specified hook function or subsystem and/or by generating a hardware reset (HRESET) pulse to restart the entire system.

Stopping task monitoring

If you stop the real-time application, you have to stop all the running watchdogs, as otherwise they will react to the unresponding tasks and the hardware might be reset. To do so, you can disable the task monitoring using the **ds1401_wd_task_monitor_stop_perform** function.

To avoid stopping monitoring accidentally when you use RTLib, you have to perform it in two steps and use a special authentication code. You have to call the **ds1401_wd_task_monitor_stop_arm** function beforehand.

Monitoring that was stopped can be restarted by reenabling it using the `ds1401_wd_task_monitor_restart` function.

When working with RTI, the `WATCHDOG_SETUP` block internally uses these RTLib functions to start and stop task monitoring if the *simState* changes.

Handling interrupts

You can specify to generate an interrupt if a watchdog timer expired. With RTLib, you can then register a hook function that will be triggered by the interrupt and executed before the real-time application exits or the system resets. With RTI, you can trigger a subsystem with a generated watchdog interrupt.

For example, you can use the hook function or subsystem to store data or to set the outputs to their termination states.

After the execution of the triggered routine, the real-time application exits and the intermodule bus is reset, which also resets the outputs.

⚠ CAUTION

Risk of personal injury

It is not guaranteed that the hook function or subsystem is executed. There might be a fatal system crash that triggers the generation of an interrupt but prevents the hook function or subsystem from being executed.

RTI/RTLib support

You can access the watchdog feature via RTI and RTLib. Refer to:

- [RTI Watchdog Blockset Reference](#) 

The blockset also provides a demo model.

Note

The RTI blocks for the watchdog handling are available in the Watchdog sublibrary of the RTI Watchdog Blockset. To use the RTI Watchdog Blockset, a separate license is required.

- [Watchdog Handling \(MicroAutoBox II RTLib Reference\)](#) 

Basics on Challenge-Response Monitoring

Introduction

With the challenge-response monitoring feature, you can monitor an entity, such as a periodic task or a subsystem in your real-time application. The monitored entity has to react with a valid response value within a specified time.

Characteristics

The challenge-response monitoring feature provides:

- Monitoring hardware that provides up to 14 monitoring units.
- Each monitoring unit can be individually configured with the following settings:
 - Number of challenge-response pairs in the range 1 ... 16.
You can specify the values of the challenges and responses in the range 1 ... $2^{32}-1$. The challenge values and the response values must be unique.
 - Challenge cycle time in the range 2.0e-3 ... 16.384 s. The accuracy is 1 ms.
This is the time interval that is used to periodically provide a new challenge. After the challenge value of the last challenge-response pair has been provided, it restarts with the first challenge value.
 - Response timeout in the range 1.0e-3 ... (**ChallengeCycleTime**-1) s. The accuracy is 1 ms.
This is the time interval in which the monitored entity has to send back a valid response.
 - Failure counters for the current number of failures and the total number of failures. If there is no response within the specified timeout or a response with an invalid value, the counters are incremented. The current failure counter is reset each time a valid response is returned in time. It is also reset if you reload the real-time application. The total counter is only reset when you reload the real-time application or reset the hardware.
 - Failure count limit of 0 ... 255.
This is the number of successive failures which are to be ignored before an action is triggered.
 - As an action, you can specify to generate a machine check (MCP) interrupt, a hardware reset (HRESET) pulse, a combination of both, or only to report the failure.
 - MCP interrupt
The real-time application is terminating with the **exit** function that applies the specified termination functions.
 - HRESET pulse
MicroAutoBox II is reset without waiting on the termination of the real-time application.
 - Combination of MCP interrupt and HRESET pulse
The real-time application is terminating. When the monitoring unit gets a further failure, the real-time hardware is reset. The interval between generating an interrupt and generating the reset pulse is controlled by the specified challenge cycle time. It takes between one cycle time and less than two cycle times.
 - Report
The user can flexibly react to an activated failure.
 - Setting to specify a hook function for the termination phase that puts the real-time application into a safe state before it stops. Refer to [Failure actions](#) on page 55.
 - Self-test of the monitoring hardware.

Timing constraints

The configuration of the challenge-response monitoring offers some timing parameters that depend on each other.

For implementing stable monitoring, note the following rules:

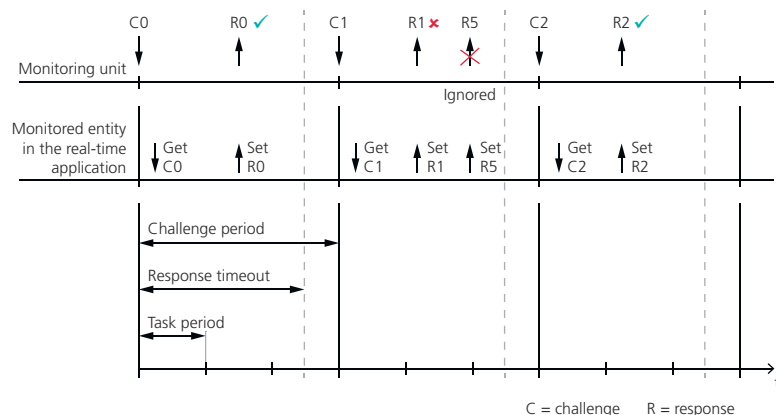
- The challenge cycle time must be:
 - At least two times longer than the sample rate of the entity to be monitored.
 - Longer than the entity requires to send a response within the response timeout.

Note, that the response timeout starts when the monitoring unit has sent the challenge value and not when the entity has received the challenge value.
- At least as long as the hook function requires to execute the termination function if you specified generating an MCP interrupt and an HRESET pulse. The time between the interrupt and the hardware reset takes at least one challenge cycle.
- The response timeout must be:
 - Less than than the challenge period.
 - Longer than the cycle time of the entity that sends the response.

At least one response is to be sent within the response timeout. If multiple responses were sent, only the first one is evaluated.
- Reading challenges and writing responses can be controlled via flags, which indicate that new challenge values are available and new response values must be returned.

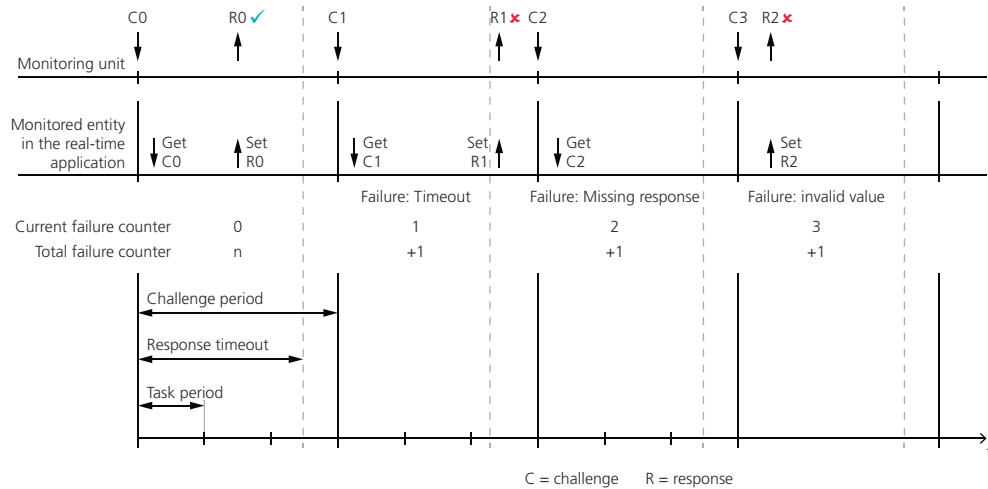
The following figures shows examples of these timing constraints.

Timing constraints are fulfilled The expected behavior of sending challenge values and receiving response values is shown in the following figure. In the second challenge period, you can see that only the first response value R1 is received and evaluated.

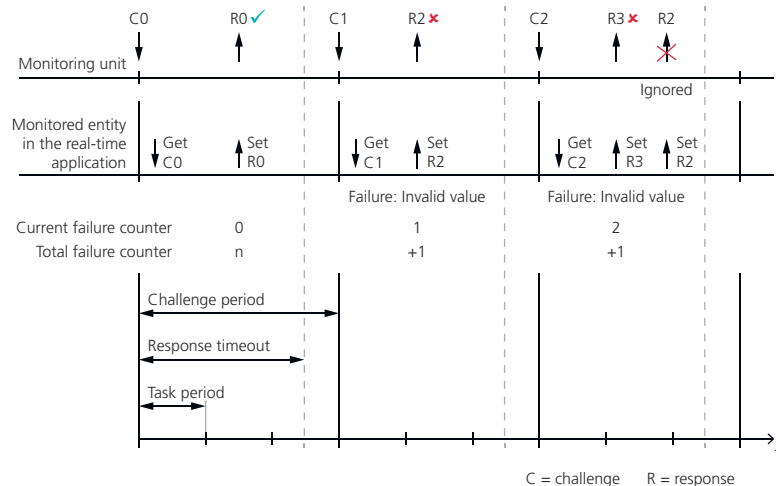


Timeout failures The following figure shows two scenarios. In the second challenge period, the response value R1 is sent within the same challenge period but the response timeout is exceeded. This leads to a failure and the failure counters are incremented. The next response value R2 exceeds not only the response timeout but also the challenge period. This leads to two failures

because the response is missing within the related challenge period and the sent response value is invalid for the current challenge period. It is presupposed, that you specified a failure count limit of at least three, otherwise the specified failure action was activated.



Invalid values The following figure shows that a failure is detected if the response value does not suit to the currently active challenge value. In the third challenge period, you can see that only the first response value R3 is received and evaluated. The failure status will not be reset by the following valid response value R2.



Failure actions

For failure handling four failure actions are available.

Generating an HRESET pulse An HRESET pulse resets the hardware. This is the most simple method to react to a failure. The real-time application immediately stops without executing any termination code. Any instantiated monitoring unit is affected by this.

A hardware reset does not reboot your hardware. Only the PowerPC processor and the intermodule bus are reset. This means, that the real-time application is

stopped and the connection to the I/O boards is refreshed. The components for the host communication are not affected. If the real-time application is stored in the Flash memory, the real-time application is automatically restarted.

Generating an interrupt If you specify to generate an interrupt, you can trigger a routine that executes individual termination code, for example, to store data or set the outputs to their termination states. With RTI, you do this by triggering a subsystem. With RTLib, you register a hook function that is to be executed when an interrupt is generated. Any instantiated monitoring unit use the same subsystem or hook function when it has to react on an interrupt.

Within the triggered subsystem or the hook function, you can get information on the monitoring unit that generated the interrupt.

After the execution of the triggered routine, the real-time application exits and the intermodule bus is reset, which also resets the outputs.

⚠ CAUTION

Risk of personal injury

It is not guaranteed that the hook function or subsystem is executed. There might be a fatal system crash that triggers the generation of an interrupt but prevents the hook function or subsystem from being executed.

Generating a combination of interrupt and HRESET pulse If you specify to generate a combination of interrupt and hardware reset, at first the real-time application executes its termination process and exits. Then the challenge-response monitoring hardware detects missing responses because of the terminated real-time application and generates an HRESET pulse. The time between the generation of the interrupt and the generation of the HRESET pulse takes the time of at least one challenge cycle and at most two challenge cycles.

Reporting With the reporting option, the challenge-response monitoring only reports the exceeded failure limit. The code to be executed can be individually implemented for each instantiated monitoring unit. With RTI, the exceeding of the failure limit is reported at the **Failure Limit Exceeded** output port, which you can connect to other blocks or subsystems. With RTLib, the activation of the failure is reported to the **pIsLimitExceeded** parameter of the **ds1401_crm_unit_fail_cntrs_get** function.

When reporting is the set failure action, the flag of the exceeded limit is not reset even if the current failure counter is reset by a valid response value later on. Once activated, you can reset the flag only by reloading the real-time application.

Note

If you have specified that only reports are triggered if the failure count limit is exceeded, you have to note the following points:

- You should use reporting as the failure action for testing or debugging purposes only.
- If you use this failure action for other purposes, it is your responsibility to implement operations that put your real-time application into a safe state.

Self-test

If the challenge-response monitoring is started, a self-test is periodically executed in the background service of your real-time application. It is checked whether the monitoring hardware is correctly working. The result of the self-test is *passed* or *failed*. The result is stored in the supervision status information. If the status is once set to *failed*, you can reset it only by reloading the real-time application.

With RTI, you can use the `CRM_SUPERVISION_STATUS` block to read the supervision status information.

With RTLib, you can use the `ds1401_crm_supervision_stat_get` function to read the supervision status information.

Note

The turnaround time of your application must be less than 400 ms. Otherwise the self-test returns the *failed* status.

Stopping the real-time application

The challenge-response monitoring does not react to the `simState` variable. If you stop or pause your real-time application, the challenge-response monitoring will execute the specified failure handling. Your hardware might be reset.

It is recommended not to pause or stop the real-time application when using the challenge-response monitoring.

RTI/RTLib support

You can access the challenge-response monitoring feature via RTI and RTLib. Refer to:

- [Challenge-Response Monitoring \(RTI Watchdog Blockset Reference !\[\]\(896151ec231b70900e969d67696ca48d_img.jpg\)](#))

The blockset also provides a demo model.

Note

The RTI blocks for the challenge-response monitoring are available in the Challenge-Response Monitoring sublibrary of the RTI Watchdog Blockset. To use the RTI Watchdog Blockset, a separate license is required.

- [Challenge-Response Monitoring \(MicroAutoBox II RTLib Reference !\[\]\(a43b62a38b6e2844e794f4301a08d3ba_img.jpg\)](#))

Basics on Memory Integrity and Extras

Introduction

With the memory integrity and extras feature, you can monitor the supply voltage of the real-time hardware and specific memory sections used by the real-time application. With the custom monitoring you can freely configure a condition that will trigger one of the failure actions.

Characteristics

The characteristics of the memory integrity and extras feature are:

- Periodically monitoring the following facilities:
 - Supply voltage (VBAT monitoring)
 - ROM memory integrity (ROM monitoring)
 - Heap memory integrity (heap monitoring)
 - Stack memory integrity (stack monitoring)

Each monitoring facility can be enabled separately.

- Using RTI, you can additionally specify to check the checksum of the ROM contents not only periodically during runtime (ROM monitoring), but also once directly after the download of the real-time application (ROM check). This allows you to detect failures during download, e.g., a failure in processing the PPC file.

Using RTLib, the execution of the ROM check is only controlled by the related flag that is written to the PPC file. You cannot deactivate it via RTLib, because the ROM check is processed before the start of the real-time application. For more information, refer to [Prerequisites](#) on page 58.

- Setting to specify the minimum threshold level of the supply voltage that triggers a reaction in the range 4.0 ... 40.0 V. The accuracy is 0.0244 V.
- Triggering the failure action via a user-defined condition.
- As a failure action, you can specify to generate a machine check (MCP) interrupt, a hardware reset (HRESET) pulse, a combination of both, or only to report the failure.

If you specified the combination of interrupt and HRESET pulse generation, you can specify the time delay for generating the HRESET pulse in the range 0.001 ... 16.383 s.

For more information, refer to [Failure actions](#) on page 59.

Prerequisites

The ROM check, the monitoring of the ROM, and the heap memory sections requires additional data in the PPC file of your real-time application. The preparation is done via the **EPK1401.exe** utility that, e.g., calculates the checksum value and writes the additional data to the PPC file.

- When you use RTI, the preparation via the **EPK1401.exe** utility is automatically included in the build process if you have enabled the relevant monitoring facilities in the **MEMORY_VBAT_MONITORING** block.
- When you use RTLib, and you want to use the ROM check, or monitor the ROM or heap memory, you must manually execute the **EPK1401.exe** utility to write the required data to the PPC file. For each rebuild of your PPC file, you must repeat this step. For information on the **EPK1401.exe** utility, refer to [EPK1401 \(MicroAutoBox II RTLib Reference !\[\]\(609f3372828e3526d7ce4ba9a1b5248e_img.jpg\)](#)).

The reference data for heap monitoring is dynamically added to the allocated heap blocks. Heap monitoring therefore increases the required heap memory by 64 bytes per heap block.

Failure actions

For failure handling four failure actions are available.

Generating an HRESET pulse An HRESET pulse resets the PowerPC processor and the intermodule bus. This is the most simple method to react to a failure. The real-time application immediately stops without executing any termination code.

Note

A hardware reset does not reboot MicroAutoBox II completely. Only the PowerPC processor and the intermodule bus are reset. This means that the real-time application is stopped and the connection to the I/O boards is refreshed. The components for the host communication are not affected. If the real-time application is stored in the flash memory, the real-time application is automatically restarted.

Generating an interrupt If you specify to generate an interrupt, you can trigger a routine that executes individual termination code, for example, to store data or set the outputs to their termination states. With RTI, you do this by triggering a subsystem. With RTLlib, you register a hook function that is to be executed when an interrupt is generated. You can specify one subsystem or hook function for the memory and VBAT monitoring and one for the implemented custom failure actions.

After the execution of the triggered routine, the real-time application exits and the intermodule bus is reset, which also resets the outputs.

⚠ CAUTION

Risk of personal injury

It is not guaranteed that the hook function or subsystem is executed. There might be a fatal system crash that triggers the generation of an interrupt but prevents the hook function or subsystem from being executed.

Generating a combination of interrupt and HRESET pulse If you specify to generate a combination of interrupt and hardware reset, at first the real-time application executes the specified hook function or subsystem and exits. Then, after the specified delay, an HRESET pulse is generated to reset the PowerPC and the intermodule bus. This guarantees a reaction on a failure even if the triggered subsystem or service routine for terminating the real-time application is corrupted.

If the real-time application is stored in the flash memory, the real-time application is automatically restarted.

The hardware component that generates the HRESET pulse uses a free-running timer with a fixed period of 1 ms. The delay of the pulse generation starts with the period that follows the point of time when the interrupt was triggered. The effective delay is therefore up to 1 ms greater than the specified delay.

Reporting With the reporting option, the monitoring facilities only report a detected failure. The code to be executed can be individually implemented for

each provided failure flag. With RTI, the failure detection is reported at the related Failure Detected output ports, which you can connect to other blocks or subsystems. With RTLib, a detected failure is reported to the `pIsFailureDetected` parameter of the related `ds1401_fse_xxx_mon_stat_get` function.

In contrast to the generation of an interrupt or an HRESET pulse, which is immediately executed when triggered, the report flag is set in the next sample step.

When reporting is the set failure action, the flag of the detected failure is not reset even if the monitored facility fulfills the conditions again. Once activated, you can reset the flag only by reloading the real-time application.

Note

If you have specified that only reports are triggered if a failure is detected, you have to note the following points:

- Use reporting as the failure action only for testing or debugging purposes.
- If you use this failure action for other purposes, it is your responsibility to implement operations that put your real-time application into a safe state.

Run-time behavior

Because the memory integrity and extras facilities are executed in the background loop of your real-time application, they do not react to the `simState` variable. Stopping or pausing the real-time application has no effect on the monitoring.

Note

The enabled monitoring facilities are executed in the current sample step only, if there is sufficient idle time in the background service.

RTI/RTLib support

You can access the memory integrity and extras feature via RTI and RTLib. Refer to:

- [Memory Integrity And Extras \(RTI Watchdog Blockset Reference !\[\]\(eb2da236c8e866008a78d7aa69bcc6c9_img.jpg\)](#))

The blockset also provides a demo model.

Note

The RTI blocks for the memory integrity and extras feature are available in the Memory Integrity and Extras sublibrary of the RTI Watchdog Blockset. To use the RTI Watchdog Blockset, a separate license is required.

- [Memory Integrity and Extras \(MicroAutoBox II RTLib Reference !\[\]\(0a8200bef1826f1b69430bdc847acc6c_img.jpg\)](#))

Nonvolatile Data Handling

Introduction

The MicroAutoBox II Base board provides memory units that can be used for nonvolatile data handling.

Hardware requirements

Supported MicroAutoBox II hardware:

Nonvolatile Data Handling	DS1401 Base Board	MicroAutoBox II with I/O Boards			
		DS1507	DS1511	DS1513	DS1514
RTC RAM Access	✓	–	–	–	–
Nonvolatile RAM Access	✓	–	–	–	–
Flash Memory Access	✓	–	–	–	–

Where to go from here

Information in this section

Basics on RTC RAM Access.....	61
Basics on Nonvolatile RAM Access.....	62
Basics on Flash Memory Access.....	63

Basics on RTC RAM Access

Introduction

MicroAutoBox II's real-time clock (RTC) provides a user RAM that you can use to store intermediate results, for example, the mileage. The real-time clock is battery-buffered. Stored data is not deleted if you switch off the system. You can store up to 16 byte.

Characteristics

You can set the start address of the RTC-RAM within the range of 0 ... 15. The following constants are predefined:

Predefined constant	Meaning
SYS1401_DATA_TYPE_INT8	8-bit signed integer
SYS1401_DATA_TYPE_UINT8	8-bit unsigned integer

Predefined constant	Meaning
SYS1401_DATA_TYPE_INT16	16-bit signed integer
SYS1401_DATA_TYPE_UINT16	16-bit unsigned integer
SYS1401_DATA_TYPE_INT32	32-bit signed integer
SYS1401_DATA_TYPE_UINT32	32-bit unsigned integer
SYS1401_DATA_TYPE_FLOAT32	32-bit float
SYS1401_DATA_TYPE_FLOAT64	64-bit float

Related topics

References

[DS1401_RAM_READ \(MicroAutoBox II RTI Reference !\[\]\(d3fb9f94af8b26d1c844efa9a98805b0_img.jpg\)\)](#)
[DS1401_RAM_WRITE \(MicroAutoBox II RTI Reference !\[\]\(78eb1652b591ce460bbb1a853a52e223_img.jpg\)\)](#)
[ds1401_rtc_ram_read \(MicroAutoBox II RTLib Reference !\[\]\(73569cd2fc0daa66f7f79a4aa32515bb_img.jpg\)\)](#)
[ds1401_rtc_ram_write \(MicroAutoBox II RTLib Reference !\[\]\(b12ef55cf189ee2bc15774f5ac32d31a_img.jpg\)\)](#)

Basics on Nonvolatile RAM Access

Introduction

MicroAutoBox II provides a non-volatile RAM that you can use to store intermediate results, for example, the mileage.

Characteristics

Stored data in the non-volatile RAM is not deleted if you switch off the system. You can store up to 4096 bytes in the address range 0 ... 4095.

To check the size of the non-volatile RAM, you can use the predefined constant **DS1401_NVRAM_SIZE**.

Related topics

References

[DS1401_NV_RAM_READ_BLx \(MicroAutoBox II RTI Reference !\[\]\(104fbf564e2e5a8fbd84f31656d114c7_img.jpg\)\)](#)
[DS1401_NV_RAM_WRITE_BLx \(MicroAutoBox II RTI Reference !\[\]\(59fb7c3d0d149ddaef5c4152c50f6f25_img.jpg\)\)](#)
[ds1401_nvram_init \(MicroAutoBox II RTLib Reference !\[\]\(172f8f80a482dff27553f72a01cb986d_img.jpg\)\)](#)
[ds1401_nvram_read \(MicroAutoBox II RTLib Reference !\[\]\(f6f37f9ebe3de993a2c92d16506e2d5e_img.jpg\)\)](#)
[ds1401_nvram_write \(MicroAutoBox II RTLib Reference !\[\]\(9a35b590bfea8a7bac885a7f277bf9a4_img.jpg\)\)](#)

Basics on Flash Memory Access

Basics

You can use the nonvolatile data feature to store data you may want to use again when restarting the application, for example, if you want to resume a simulation with the last mileage or the settings for an air conditioner.

Characteristics

The flash memory (16 MB) of MicroAutoBox II consists of 64 blocks, each 256 KB in size. This nonvolatile flash memory can be used to store one application for stand-alone booting and flight recorder data. Additionally, it can be used for the Nonvolatile Data Handling. Each data set requires a complete 256-KB flash memory block, which has to be allocated and registered in the flash module. You can configure how much memory space you want to use for the application, the nonvolatile data, and for flight recording. For further information, refer to *Flash partitioning* in [Basics on Flight Recorder](#) on page 66.

Tip

You can store calibrated parameter values to the flash memory of the MicroAutoBox II. Using the *Store Calibration Parameter to Flash* solution, you can let a flash application store calibrated parameter values during the regular shutdown process or when you switch the SimState from RUN to STOP in ControlDesk.

When the application restarts from the flash memory of the MicroAutoBox II, the application starts with the parameter values recently calibrated. Without the solution, the application would start with the original parameter values as defined in the Simulink® model.

For detailed information on the solution, install it from the dSPACE Solutions DVD and read the user guide. For information on installing it, refer to the `ReadMe.txt` file.

Termination

The nonvolatile data to be stored is collected in a temporary buffer until the application is terminated. If MicroAutoBox II is switched off and you have implemented the Power Hold feature in your application, buffered data is transferred to the flash memory before MicroAutoBox II is shut down, so that no data is lost. For further information, refer to [Basics of Power Hold Control](#) on page 76.

Buffered data might be lost, if:

- Power supply is switched off without using the Power Hold feature.
- The execution of the real-time application is stopped because of a **Stop RTP** or **Real-Time Application - Reload** command in ControlDesk (the `simState` variable is then not switched to 0).

Note

If you clear the flash memory, load a real-time application that accesses the flash memory and then stop the RTP (without changing `simState` to 0), the nonvolatile data handling feature remains in its initialization state and can not store any data. In this state, the `Status` signal of a `RESTORE_FROM_FLASH_BL` block permanently outputs 1.

To finish the initialization state of the nonvolatile data handling, you have to do the following steps:

1. Clear the flash memory
2. Reload the application
3. Set `simState` to Stop (0)

The same termination behavior is used when the internal voltage monitoring detects a voltage below the minimum voltage level (only if the flight recorder is used and not for the nonvolatile data).

When the application is started again, the nonvolatile data is copied from the flash memory to the temporary buffer. If no data is stored in the flash or the data is corrupt, the initial parameter values of the application are used.

Demo model

For a demo model using the Power Hold On/Off feature, refer to `<RCP_HIL_InstallationPath>\Demos\DS1401\RTI\Demo1401FlashAccess.md1`. You find this demo in the MicroAutoBox II's RTI demo library.

Related topics**References**

[Flash Memory Access \(MicroAutoBox II RTLib Reference !\[\]\(d8ab143e904bfa3467271eec5af75a9b_img.jpg\)\)](#)

Flight Recorder

Introduction

MicroAutoBox II offers two different flight recorders for long-term data acquisition.

Where to go from here

Information in this section

Flight Recorder (Flash Memory).....	65
USB Flight Recorder.....	69

Flight Recorder (Flash Memory)

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓	–	–	–	–

Where to go from here

Information in this section

Basics on Flight Recorder.....	66
The flight recorder is used to store time histories of real-time variables in nonvolatile memory.	
Using the Flight Recorder.....	67
You can use the flight recorder to write data to the flash memory.	
MAT File Format for the Flight Recorder.....	68
Flight Recorder uses the MAT file format.	

Basics on Flight Recorder

Characteristics

The flight recorder is used to store time histories of real-time variables in nonvolatile memory. During the real-time simulation, the values of real-time variables are written to the 16 MB flash memory of MicroAutoBox II Board. Up to 13 MB (with the nonvolatile feature) or 13.25 MB (without the nonvolatile feature) of the flash memory can be used for flight recording. The great capacity of the flight recorder permits long-term data acquisition.

The flight recorder section of the flash memory is organized as FIFO storage. A maximum of 250 different real-time variables can be recorded.

After the simulation has finished, the acquired data can be read out by the host PC. On the host PC, the flight recorder data is stored in binary or reference data (MAT file) format. Because MicroAutoBox II has a power-down feature that terminates all currently executed applications, flight recording is stopped and the captured data is transferred to the flash memory. This avoids data loss during termination.

For detailed information on using the flight recorder, refer to [Using the Flight Recorder](#) on page 67.

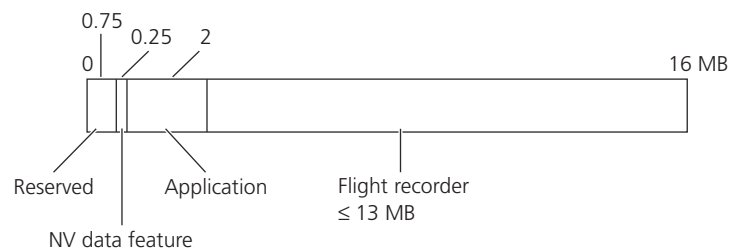
Flash partitioning

The flash memory stores various information:

- Internal data, for example, the boot firmware
- Data section for the nonvolatile data handling
- Application data (one application for stand-alone booting)
- Flight recorder data

According to your settings in the FLASH_SETUP block or the dsflrec_initialize function, this data is used as a memory configuration to partition the flash memory. If an application image exceeds the 2 MB section reserved by default or the specified size of the application data section, the complete flash memory must be cleared to permit repartitioning. The stored data gets lost. For instructions how to clear the flash, refer to [How to Clear an Application from the Flash Memory of dSPACE Real-Time Hardware \(ControlDesk Platform Management\)](#).

The default memory configuration is shown in the following illustration.



Time stamps

In the flight recorder, data captures are stored together with time stamps. Time stamps are measured in seconds with a resolution of 10.24 μ s relative to the time base 1970-01-01. Time stamps are interpreted appropriately by MATLAB or dSPACE experiment software. You can change the time base using M-program code. For an example, refer to [MAT File Format for the Flight Recorder](#) on page 68.

Related topics**References**

[Flash Memory Access \(MicroAutoBox II RTI Reference !\[\]\(a03a7eb2f4046e1d3c76772003e549ea_img.jpg\)\)](#)

[Flight Recorder \(Flash Memory\) \(MicroAutoBox II RTLib Reference !\[\]\(cbe2492b119e39e02a1dab2af4a4b296_img.jpg\)\)](#)

Using the Flight Recorder

Introduction

You can use RTI's library rtiflashlib or RTLib functions to write flight recorder data to the flash memory.

Memory overwrite mode

You can choose via RTI or RTLib functions how data will be handled when the memory block for flight recording is full:

Discard new data (blocked mode) When the memory block for flight recording is full, no further data will be recorded.

Replace old data (overwrite mode) When the memory block for flight recording is full, the oldest entries will be replaced.

Note


To avoid the loss of data you should save the data to the PC and delete the data from the flash memory in regular intervals. The application must be stopped before.

Demo model

For a demo model using the Flash Memory Access feature and the flash memory-based flight recorder, refer to

<RCP_HIL_InstallationPath>\Demos\DS1401\RTI\
Demo1401FlashAccess.slx. You find the Flash Access demo also in the MicroAutoBox II's RTI demo library.

Note

This demo does not run correctly, if you use the preconfigured cabling by dSPACE for the power input connector, because the REMOTE pin is shortened to VBAT. For further information on the power input connector and the required cabling, refer to [Connecting to Power Supply](#) (MicroAutoBox II Hardware Installation and Configuration Guide ).

Loading data to the host PC

After the simulation has finished, the acquired data can be read out by the host PC. Reading out the data does not delete the data from the flash memory.

Tip

Reading the data in MAT format requires more resources on the host PC (processor and memory). In the vehicle, you can upload a binary file to the host PC and convert the data to a MAT file later on. In the laboratory, you can upload the data directly to a MAT file (refer to [MAT File Format for the Flight Recorder](#) on page 68).

To handle flight recorder data with ControlDesk, refer to [Flight Recording](#) (ControlDesk Measurement and Recording ).

Related topics**Basics**

[Basics on Flight Recorder](#)..... 66



MAT File Format for the Flight Recorder

Introduction

Flight Recorder uses the MAT file format.

MAT file format

MAT files generated by the flight recorder contain a separate x-axis (timestamp vector) for each y-axis. To find the correct x-axis, the XIndex element of the y-axis data struct must be evaluated. For an example, see the code below.

For instructions on how to access the data of a MAT file, see [Postprocessing Recorded Data With MATLAB](#) (ControlDesk Measurement and Recording ). For details on the MAT file structure, see [Structure of MAT Files Generated by Exporting Recorded Data](#) (ControlDesk Measurement and Recording ).

Changing the time base

The following program code shows you how to convert flight recorder data based on the 1970-01-01 to data relative to an entered time base in MATLAB. You have to change <MATFILENAME> with your own names.

```
load <MATFILENAME>.mat
varValues = <MATFILENAME>;
base_date = '01-Jan-1970';
plot_date = varValues.FlightRec.StartDateTime;
n = 1;
figure;
plot_start = (datenum(plot_date) - datenum(base_date))*24*3600;
use_XIndex = varValues.Y(n).XIndex;
plot(varValues.X(use_XIndex).Data - plot_start, varValues.Y(n).Data);
xlabel(strcat('sec since [', datestr(plot_start/24/3600 + datenum(base_date)), ']'));
ylabel(varValues.Y(n).Name);
```

Let the index n increase to display all the recorded data.

Related topics

Basics

Basics on Flight Recorder.....	66
Using the Flight Recorder.....	67

USB Flight Recorder

Purpose

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

In contrast to the flash memory-based Flight Recorder (see [Flight Recorder \(Flash Memory\)](#) on page 65), the storage size is only restricted by the USB mass storage device.

You can use both the flash memory-based Flight Recorder and the USB Flight Recorder in your application.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓	–	–	–	–

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

Basics on USB Flight Recorder.....	70
Handling the Data of the USB Flight Recorder.....	74

Basics on USB Flight Recorder

General information

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

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

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

The default maximum size of a single file is 32 MB. With RTLib, you can specify a maximum file size of up to 256 MB. If more than the specified maximum file size is recorded during one simulation, the data is split into several files. Consecutive files are created until the storage capacity of the USB device has been reached. Then the captured data is discarded or older files are overwritten, according to your setting (refer to [Memory overwrite mode](#) on page 70).

After the simulation has finished, the recorded data can be read out by the host PC, refer to [Handling the Data of the USB Flight Recorder](#) on page 74.

With MicroAutoBox II's power-down feature, all currently executed applications are terminated before the hardware is shut down. The flight recording is also stopped in a definite state to avoid data loss.

Memory overwrite mode

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

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

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

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

Requirements on the USB mass storage device

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

Note

A connection via a USB hub is not supported.

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

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

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

Avoiding data loss

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

Note

Risk of data loss

While a USB Flight Recorder session is active:

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

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

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

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

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

- Stop the real-time application.

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

- Eject by predefined signal.

The power input connector of MicroAutoBox II provides the REMOTE pin that you can use for unmounting the USB device. If you connect an external *Eject button*, that is to be implemented by you, to this pin, the unmount procedure is started when you press the button. You can remove the USB device when the USB status LED is off. A host message is also generated.

For further information on the power input connector and the required cabling, refer to [Connecting to Power Supply \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(d263118e0bfd47dc6bc704167d936b83_img.jpg\)](#)).

- Eject by user-defined signal.

You can use the `USB_FLIGHT_REC_EJECT` block in your Simulink model or the `dsflrec_usb_eject` command in your handcoded application to unmount the USB device as a reaction to a signal, for example, a specific model variable.

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

Removing the USB device when the box is switched off To safely unmount the USB device from your MicroAutoBox II before it is switched off, you have to use the power hold control feature. Switching the MicroAutoBox II on and off is then controlled by the REMOTE input pin provided by the power input connector. The signal for switching off is followed by a termination procedure, including unmounting the USB device, before the power is actually switched off. For further information, refer to [Basics of Power Hold Control](#) on page 76 and [Connecting to Power Supply \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)](#)).

Tip

Alternatively, you can use MicroAutoBox II's flash-memory based flight recorder to guarantee the data is saved, refer to [Basics on Flash Memory Access](#) on page 63.

Accessing USB Flight Recorder files via FTP

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

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

Note


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

USB status LED

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

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

Time base

In the flight recorder, data captures are stored together with time stamps. Time stamps are measured in seconds relative to the time base 01/01/1970. Time stamps are interpreted appropriately by MATLAB or dSPACE experiment software. You can change the time base using M-program code. For an example, refer to [MAT File Format for the Flight Recorder](#) on page 68 or [MAT File Format for the USB Flight Recorder](#) (RTI USB Flight Recorder Blockset Reference ).

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

Startup behavior and maximum data rate

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

Using MicroAutoBox II If the real-time application is loaded to RAM, a data rate of maximum 1 MB/s is possible without data loss. If the real-time application is loaded to flash memory, firstly a data rate of maximum 600 kB/s is possible without data loss in the startup phase of MicroAutoBox II which takes about 6 seconds. After the startup, a maximum data rate of 1 MB/s is possible.

Limitations using the USB Flight Recorder

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

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

- It is recommended to use a separate USB mass storage device for flight recording. Other files in the root folder of the device will be deleted by the USB Flight Recorder.
- Do not use a USB hub. The device must be directly connected to your hardware.

- Do not manually create or delete files while the USB device is used by an active flight recorder.
- Do not create subfolders on the USB device.

RTI/RTLib support

Using RTI You can use the RTI blocks from the RTI USB Flight Recorder blockset to write flight recorder data to the USB mass storage device, refer to [Components of the RTI USB Flight Recorder Blockset \(RTI USB Flight Recorder Blockset Reference !\[\]\(99f58673407353e96a019fbca558fd72_img.jpg\)\)](#).

Using RTLib You can use the `dsf1rec_usb` RTLib functions to write flight recorder data to the USB mass storage device, refer to [USB Flight Recorder \(USB Flight Recorder RTLib Reference !\[\]\(0f848bbd71cef6b345273b16f905912a_img.jpg\)\)](#).

Handling the Data of the USB Flight Recorder

Introduction

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

Loading data to the host PC

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

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

For further information, refer to [How to Upload Flight Recorder Data Written to a USB Mass Storage Device \(ControlDesk Measurement and Recording !\[\]\(6a9b39b98eb945faa14c645ec99e4eaa_img.jpg\)\)](#).

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

Note


Do not delete any files on the USB mass storage device while a flight recorder session is still running.

For further information, refer to *Accessing USB Flight Recorder files via FTP* in [Basics on USB Flight Recorder](#) on page 70.

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

Note

See the section *Avoiding data loss* in [Basics on USB Flight Recorder](#) on page 70 for information on how to safely remove the USB device from your hardware.

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

Related topics**Basics**

[Basics on USB Flight Recorder](#)..... 70

Power Hold Control

Introduction

You can control MicroAutoBox II's power hold feature via Simulink model or real-time application. This way the MicroAutoBox II is shut down after all termination processes have finished.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓	–	–	–	–

Basics of Power Hold Control

Introduction

You can control MicroAutoBox II's power hold feature by using RTLib functions or Simulink blocks in your model. This way the MicroAutoBox II is shut down after all termination processes have finished.

Required connections

- The Remote In signal must be connected to the ignition/driving switch (KL15).
- The Vbat input must be permanently connected to the battery power (KL30).

Power hold control via RTLib

To control MicroAutoBox II's power hold feature via RTLib functions, you have to use the `ds1401_power_hold_on` and `ds1401_power_hold_off` functions. If the Remote In signal is set to active low and the simulation state (`simState`) is set to STOP, the program executes its termination. Finally, the MicroAutoBox II is shut down. For further information on `simState`, refer to [Simulation Control \(RUN/STOP Mechanism\)](#) (RTI and RTI-MP Implementation Guide [\[1\]](#)).

Power control via Simulink blocks

When a DS1401_POWER_DOWN block is triggered, it sets the simulation state (`simState`) to STOP and the termination code is executed. The DS1401_POWER_DOWN block is typically triggered by the Remote In signal. If the Remote In signal is set to low, MicroAutoBox II is shutdown.

The Remote In signal can be accessed by:

- DS1401_REMOTE_IN_BLx

For details and preconditions, refer to [DS1401_POWER_DOWN \(MicroAutoBox II RTI Reference !\[\]\(8af806fb1314382d09bc5ec5b767526c_img.jpg\)](#)).

Note

To monitor the REMOTE In signal you can use the DS1401_REMOTE_IN_BLx block. You must use another signal in your model to trigger the DS1401_POWER_DOWN block. The remote signal can be handled via the ZIF I/O connector or the power input connector. For further information, refer to [Connecting to Power Supply \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)](#)).

Tip

You can check, whether the Remote In signal is available and connected with the RTLib functions `ds1401_remote_in_init` and `ds1401_remote_in_read`.

**Switching off
MicroAutoBox II and RapidPro
I/O subsystem**

If you want to ensure that a combination of MicroAutoBox II and RapidPro I/O subsystem powers down after the termination tasks of all the blocks in the master application are finished, you can use the DS1401_POWER_DOWN block in the same way as it is possible for a MicroAutoBox II alone. For details, refer to [DS1401_POWER_DOWN \(MicroAutoBox II RTI Reference !\[\]\(830769b31eeeaca920791081939ff8ba_img.jpg\)](#)).

Note

The *Remote In* hardware input pins (KL15) of the MicroAutoBox II and the RapidPro I/O subsystem must be connected.

Related topics**References**

[DS1401_POWER_DOWN \(MicroAutoBox II RTI Reference !\[\]\(bd3b31712ad9bab5a241210fa6925cdd_img.jpg\)](#))
[DS1401_REMOTE_IN_BLx \(MicroAutoBox II RTI Reference !\[\]\(882be629d4a853dc90d60f084b0d185d_img.jpg\)](#))
[ds1401_remote_in_init \(MicroAutoBox II RTLib Reference !\[\]\(cadb1a36ec331fde129feec52622b01a_img.jpg\)](#))
[ds1401_remote_in_read \(MicroAutoBox II RTLib Reference !\[\]\(993d39f42bf03c4f62d9b7c594e41af9_img.jpg\)](#))

Onboard Sensors

Where to go from here

Information in this section

Sensor for Acceleration Measurement.....	78
To measure the acceleration of the board.	
Sensor for Pressure Measurement.....	81
To measure the air pressure.	

Sensor for Acceleration Measurement

Introduction

The acceleration sensor of MicroAutoBox II allows you to use the position and movements of the system in your real-time application, for example, for vehicle dynamics.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓ ¹⁾	–	–	–	–

¹⁾ Board revision DS1401-23 and later required.

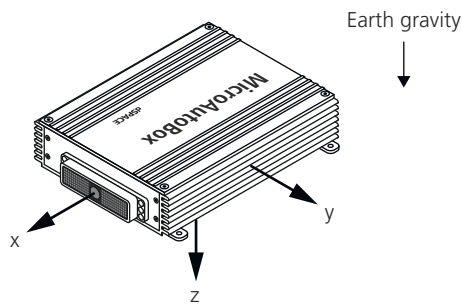
System PLD firmware version 1.4 and later required.

Measuring acceleration

The acceleration sensor continuously measures the gravity on its three axes with an adjustable output data rate of maximal 800 Hz.

If your MicroAutoBox II is exactly horizontal, the acceleration sensor returns 1 for the z-axis and 0 for the x-axis and the y-axis. This means that a force of 1 g impacts on the MicroAutoBox II.

Thus, any static position or movement of the system can be detected and calculated.



The following settings of the acceleration measurement can be configured during initialization:

- **Measurement range**
To specify the range for the expected gravity values.
- **Read mode**
To specify whether to read only the most recent value or the entire buffer of up to 512 values.
- **Output data rate**
To specify the sample rate of the sensor.

Measurement range You have to specify the measurement range to fit the expected measurement values. The lowest measurement range provides the highest resolution. The resolution and the maximum acceleration values depend on the configured data width of the sensor. The default configuration after initialization uses a data width of 10 bits. This leads to the following values:

Measurement Range	Max. Acceleration Negative Direction	Max. Acceleration Positive Direction	Resolution
± 2 g	-2.0000 g	1.9961 g	3.9 mg
± 4 g	-4.0000 g	3.9922 g	7.8 mg
± 8 g	-8.0000 g	7.9844 g	15.6 mg

Read mode You can specify whether to read only the most recent value or the entire buffer of up to 512 values. Because each value provides information on all the three axes, it is also called a *value set*. The measured value sets are stored in a FIFO buffer from which they can be read and transferred to the specified variables in your real-time application. If you want to read the entire FIFO buffer, each specified variable has to provide a size of at least 512 float values.

Read access is controlled by the Count output of the DS1401_ACCEL_READ_BLx block or the `ValueSetsReadCount` parameter of the `ds1401_accel_sensor_xyz_axis_read` function. The parameter returns the number of currently available value sets in the FIFO buffer. If there is no new

value available since the last read operation, it returns zero and the values in your variables are not overwritten.

Output data rate You can specify how often the acceleration sensor measures a new value according to your requirements. For example, to detect movements, the output data rate should be higher than the output data rate for detecting only new positions. The maximum time interval for reading the values before a buffer overflow happens directly depends on the specified output rate.

Rate Number	Output Data Rate		Max. Time Interval
	Period	Frequency	
1	1.25 ms	800 Hz	0.64 s
2	2.5 ms	400 Hz	1.28 s
3	5 ms	200 Hz	2.56 s
4	10 ms	100 Hz	5.12 s
5	20 ms	50 Hz	10.24 s
6	80 ms	12.5 Hz	40.96 s
7	160 ms	6.25 Hz	81.92 s
8	640 ms	1.56 Hz	327.68 s

User configuration of the acceleration sensor

Using RTLib, you can directly access the registers from the acceleration sensor. This allows you to configure the sensor for specific purposes and to read the measured data from the related data register. If you have configured the acceleration sensor to use one of the two available interrupt lines, you can receive the interrupt signal via the **DS1401_INT_ACCEL_SENS** interrupt provided by the MicroAutoBox II base board (see also **ds1401_set_interrupt_vector**).

Note

- Use the register access functions only if you are experienced in using the features of the acceleration sensor.
For a detailed description of the available device registers, refer to the data sheet of the acceleration sensor device MMA8453Q by Freescale Semiconductor.
- Do not reconfigure the acceleration sensor when using the **DS1401_ACCEL_READ_BLx** block in a Simulink model.
- If you reconfigure the acceleration sensor the **ds1401_accel_sensor_xyz_axis_read** function might not return correct values. You also have to use the register access functions to get the measured values.

RTI/RTLib support

You can access the acceleration sensor via RTI and RTLib. For details, see:

- DS1401_ACCEL_READ_BLx** (MicroAutoBox II RTI Reference )
- Acceleration Measurement** (MicroAutoBox II RTLib Reference )

Sensor for Pressure Measurement

Introduction

The pressure sensor of MicroAutoBox II allows you to consider the air pressure to your real-time application, for example, for engine control.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓ ¹⁾	–	–	–	–

¹⁾ Board revision DS1401-23 and later required.

System PLD firmware version 1.3 and later required.

Measuring air pressure

The air pressure is continuously measured with a rate of approximately 200 Hz. The measurement range is 50 ... 115 kPa with a resolution of 1 kPa. The value is returned as a float value.

RTI/RTLib support

You can access the pressure sensor via RTI and RTLib. For details, see:

- [DS1401_PRESSURE_BLx \(MicroAutoBox II RTI Reference !\[\]\(05a3150ca7eafd44fce8deaa48838121_img.jpg\)\)](#)
- [Pressure Measurement \(MicroAutoBox II RTLib Reference !\[\]\(6ce459b4dcae8e7d92253a855b1dd385_img.jpg\)\)](#)

Related topics

References

[DS1401_PRESSURE_BLx \(MicroAutoBox II RTI Reference !\[\]\(3342c215b2a8b663596a81468d5dc314_img.jpg\)\)](#)
[ds1401_pressure_read \(MicroAutoBox II RTLib Reference !\[\]\(5e22d44aef1f9548ca8274cbfb388e9d_img.jpg\)\)](#)

MicroAutoBox II I/O Features

Introduction

The following I/O features are supported by MicroAutoBox II. For information on which combination of Base board and I/O board is supporting a specific feature, refer to the relevant feature description.

Where to go from here

Information in this section

Information on the I/O Module Availability.....	84
A/D Conversion.....	86
Bit I/O.....	111
D/A Conversion.....	126
Ethernet I/O Interface.....	131
ECU Interface.....	133
ECU Software Porting Kit.....	142
PWM Signal Generation with a Variable Period.....	144
Multi-Channel PWM Signal Generation (MC_PWM).....	169
PWM Signal Measurement (PWM2D).....	195
Pulse Pattern Measurement.....	209
Pulse Width Measurement (PW2D).....	220
Incremental Encoder Interface.....	228

Information in other sections

Hardware Concept.....	12
-----------------------	----

Information on the I/O Module Availability

Introduction

The I/O features are provided by special I/O modules. Depending on the MicroAutoBox II variant different numbers of the I/O modules are available.

Overview of the Number of Available I/O Modules

Introduction

Depending on the MicroAutoBox II variant different numbers of the I/O modules are available. Usually, there is only one I/O module representing a certain type of I/O feature, but some I/O modules are available up to four times.

Using RTI

If you use the RTI Blockset of MicroAutoBox II (RTI1401), the **Module number** setting specifies the I/O module you want to use for the selected I/O feature. For example, if there are two CAN Type 1 modules available, you can specify whether to use the first or second CAN Type 1 module.

Using RTLib

If you use the RTLib1401, the **ModuleAddr** parameter specifies the I/O module you want to use for the selected I/O feature. For example, if there are two CAN Type 1 modules available, you can specify whether to use the first or second CAN Type 1 module.

I/O Modules of MicroAutoBox II

Below is a table with the numbers of I/O modules when using MicroAutoBox II.

I/O Modules	I/O Boards				
	DS1507	DS1511	DS1511/DS1514	DS1513	DS1513/DS1514
ADC Type 4	–	1	1	1	1
AIO Type 1	–	–	–	1	1
DAC Type 3	–	1	1	–	–
DIO Type 3	–	1	1	–	–
DIO Type 4	–	–	–	1	1
ECU Type 1 ¹⁾	3	2	2	2	2
CAN Type 1	2	2	2	3	3
SER Type 1 ²⁾	2	2	2	3	3

I/O Modules	I/O Boards				
	DS1507	DS1511	DS1511/DS1514	DS1513	DS1513/DS1514
IP Type 1 ³⁾	2	–	2	–	2
FPGA Type 1	–	–	1	–	1

¹⁾ For further information, refer to [Hardware](#) on page 134.

²⁾ The SER Type 1 module realizes the serial interface support based on the CAN Type 1 module.

³⁾ Supports standard IP modules, third-party FlexRay IP modules, DS4340 FlexRay Interface Modules, or DS4342 CAN FD Interface Modules.

A/D Conversion

Where to go from here

Information in this section

Overview of the A/D Conversion Units.....	86
ADC Unit Type 4.....	87
AIO Unit Type 1 (ADC).....	99
ADC 1552 Type 1 Unit.....	101
ADC 1552 Type 2 Unit.....	108

Overview of the A/D Conversion Units

Introduction

The MicroAutoBox II variants provide various A/D conversion units.

Hardware requirements

Supported MicroAutoBox II hardware:

ADC Unit	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
ADC Type 4	–	✓	✓	–
AIO Type 1 ADC	–	–	✓	–
ADC 1552 Type 1	–	–	–	✓ ¹⁾
ADC 1552 Type 2	–	–	–	✓ ¹⁾

¹⁾ Requires DS1552 Multi-I/O Module

Overview of the characteristics

The main characteristics of the ADC units are listed below.

Characteristics	ADC Type 4	AIO Type 1 ADC	ADC 1552 Type 1	ADC 1552 Type 2
Number of converters	16	16	8	16
Channels per converter	1	1	1	1
Resolution	16 bit	16 bit	16 bit	16 bit
Conversion time	1 µs	4 µs	1 µs	4 µs

Characteristics	ADC Type 4	AIO Type 1 ADC	ADC 1552 Type 1	ADC 1552 Type 2
Simultaneous start via software	Yes	No	Yes	No
Burst conversion	Yes	No	No	No
Interrupt generation	<ul style="list-style-type: none"> ▪ Burst start ▪ Data ready ▪ Conversion trigger overflow ▪ Data lost 	No	<ul style="list-style-type: none"> ▪ Data ready ▪ Conversion trigger overflow ▪ Data lost 	No

Related topics

Basics

ADC 1552 Type 1 Unit.....	101
ADC 1552 Type 2 Unit.....	108
ADC Unit Type 4.....	87
AIO Unit Type 1 (ADC).....	99

ADC Unit Type 4

Configurable A/D converter

The ADC Unit Type 4 provides A/D converters that can be configured for various use cases. The features are described in detail.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
—	—	✓	✓	—

Characteristics

The ADC Unit Type 4 module consists of 16 parallel A/D converters. The A/D converters of a MicroAutoBox II variant with a DS1511 or DS1513 I/O board are equipped with single-ended inputs and particularly meet the requirements for digitizing analog input signals at high sampling rates, for example, for measuring internal cylinder pressures.

Each of the 16 A/D conversion channels provides:

- A single-ended input with a sample&hold unit
- 16-bit resolution and a maximum conversion time of 500 ns
- The applicable input voltage range depends on the I/O board used:
 - DS1511: 0 ... +5 V
 - DS1511B1: -10 ... +10 V
 - DS1513: -10 ... +10 V

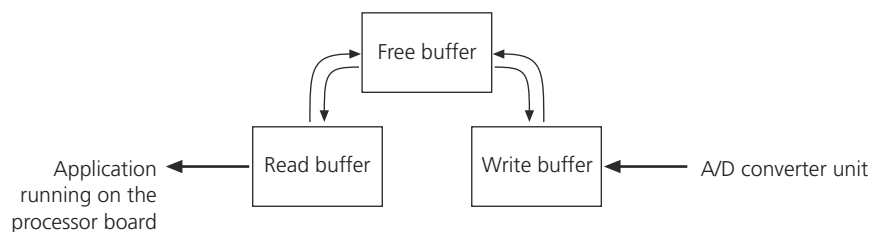
If the specified input voltage range does not suit to the connected hardware, the real-time application stops with an error message.

The board versions are printed on a type plate on the bottom of your MicroAutoBox II.

- Burst mode for digitizing a data set of up to 8192 analog values per burst:
 - Triggered sample mode with selectable trigger source for starting the bursts and A/D conversions (see [Burst triggered sample mode](#) on page 90).
 - Continuous sample mode with automatically started successive bursts and A/D conversions (see [Burst continuous sample mode](#) on page 91).
- Single A/D conversion mode to use the channel as a standard A/D converter without utilizing its burst capability (see [Single conversion mode](#) on page 92).
- Time-stamping mode to use a pair of channels for storing time stamps with a resolution of 10 ns (see [Time-stamping mode](#) on page 94).
- Selectable sources for triggering A/D conversions, for example, external trigger input, channel timer or software trigger (see [Trigger signals](#) on page 89).
- Swinging buffers for decoupling the conversion process from the read process (see [Swinging Buffer](#) on page 88).
- Four independent hardware interrupts associated to the A/D conversion state. For information on ADC Type 4 interrupt handling, see [Interrupts provided by the ADC Type 4 module](#) on page 95.

Swinging Buffer

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



Swinging buffer principle The swinging buffer, comprising a write, a free, and a read buffer, passes all conversion results from the A/D converter unit to the application running on MicroAutoBox II. The buffers can change places, as indicated by the arrows between them in the illustration above. This is implemented by pointer management, so that no buffer needs to be copied from one position to another. The events which trigger the buffer exchange are

described below. The number of temporarily stored conversion results, the burst size, can be uniformly specified in the range 1 ... 8192.

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

Read buffer The application running on MicroAutoBox II reads the conversion results from the read buffer. Before the conversion results are transferred, the read function used in the application requests a read buffer. Depending on the read method used by the application,

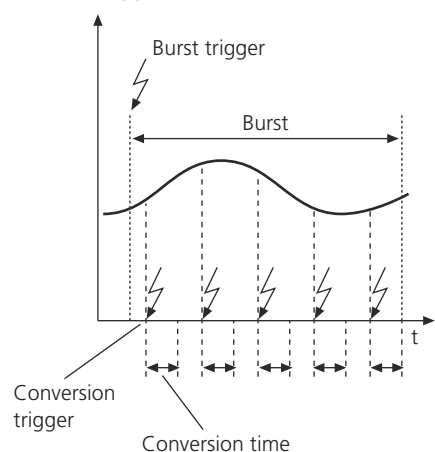
- The current conversion results in the read buffer are transferred immediately.
- or
- The A/D conversion function waits until the buffer control unit exchanges the read buffer with the free buffer containing new conversion results.

For more information on the read methods, refer to [Methods of reading conversion results](#) on page 93.

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

Trigger signals

To configure an A/D converter channel of the ADC Type 4 module, up to two kinds of triggers must be selected.



Burst trigger A burst trigger initiates a sequence of A/D conversions, called a conversion burst. You select the burst trigger by assigning a burst trigger source to a converter channel, for example, an external trigger input.

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

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

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

If a conversion is in progress and has not completed before another conversion trigger occurs, this trigger is ignored, and a conversion trigger overflow interrupt is generated (if the generation of the conversion trigger overflow interrupt was initialized beforehand). For more information, refer to [Interrupts provided by the ADC Type 4 module](#) on page 95.

Trigger sources An ADC Type 4 conversion channel can react to the following trigger sources:

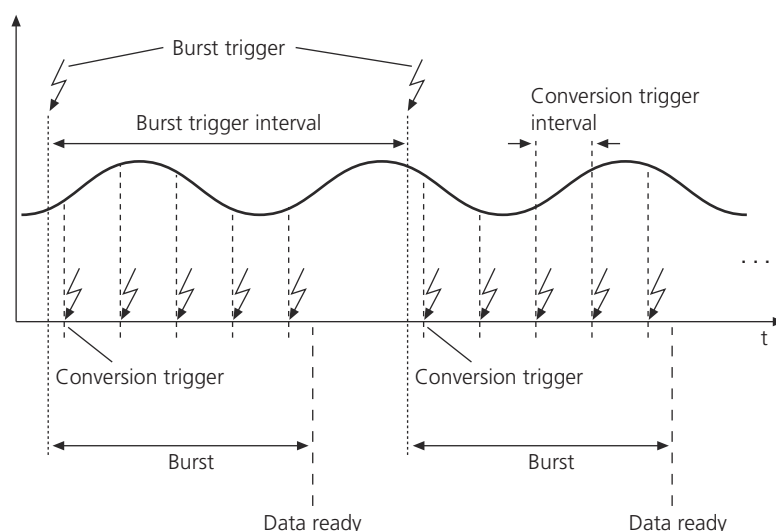
- 4 external trigger inputs
- Individual channel timer for each channel
- Software trigger (using RTLib) and sample base rate (using RTI).

External trigger The external trigger inputs can be used with a maximum frequency of 1 MHz according to the conversion time of 1 μ s. For detailed information on the electrical characteristics, refer to:

- [Data Sheet MicroAutoBox II 1401/1511 \(MicroAutoBox II Hardware Reference !\[\]\(c6a8736a601a632e2c96605cf66055ed_img.jpg\)](#))
- [Data Sheet MicroAutoBox II 1401/1511/1514 \(MicroAutoBox II Hardware Reference !\[\]\(64ef2b19d70b31fbbfce0e0e2aa3d7b4_img.jpg\)](#))
- [Data Sheet MicroAutoBox II 1401/1513 \(MicroAutoBox II Hardware Reference !\[\]\(9ba1c633ca37327550476fd7d0d00348_img.jpg\)](#))
- [Data Sheet MicroAutoBox II 1401/1513/1514 \(MicroAutoBox II Hardware Reference !\[\]\(9123a11efb62a56709757215846100c3_img.jpg\)](#))

Burst triggered sample mode

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



The figure shows the behavior when you use an external trigger input for the conversion trigger. The minimum delay between the burst trigger and the first conversion trigger in the burst must be 75 ns. Otherwise the first conversion will not be started.

With a timer as conversion trigger, the burst trigger and the first conversion trigger in the burst have no time delay.

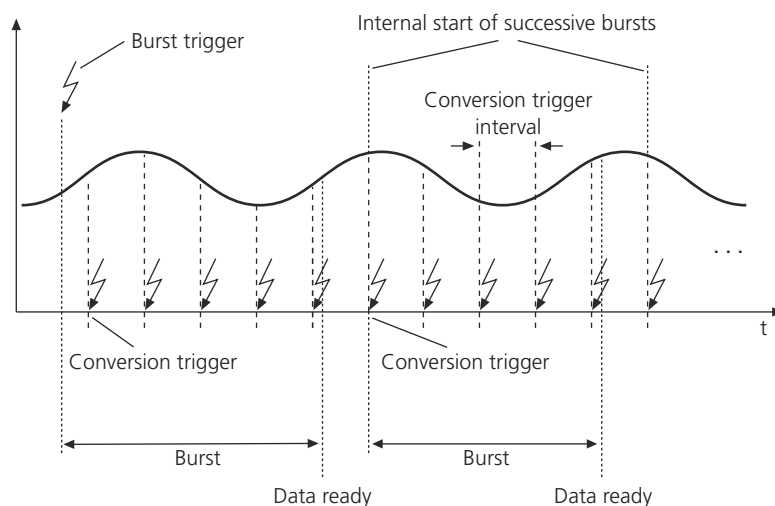
The conversion results can be read after a burst has completely finished or was stopped before the last A/D conversion was executed. There are three methods of reading the conversion results. For details, refer to [Methods of reading conversion results](#) on page 93.

Burst continuous sample mode

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

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

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



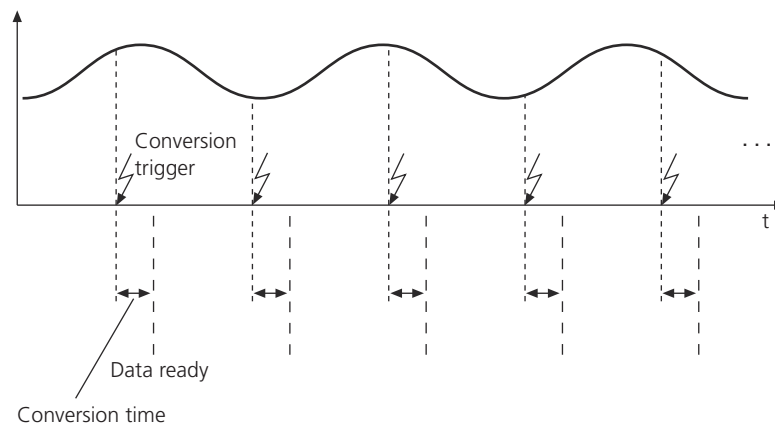
The figure shows the behavior when you use an external trigger input for the conversion trigger. The minimum delay between the burst trigger and the first conversion trigger must be 75 ns. Otherwise the first conversion will not be started.

With a timer as conversion trigger, the burst trigger and the first conversion trigger have no time delay.

The conversion results can be read after a burst has completely finished or was stopped before the last A/D conversion was executed. There are three methods of reading the conversion results. For details, refer to [Methods of reading conversion results](#) on page 93.

Single conversion mode

In single conversion mode, the burst capabilities of an ADC Type 4 conversion channel are not used. The conversion channels work as standard A/D converters for converting a single value after receiving a conversion trigger. Each trigger event produces one conversion result which can be read immediately after its conversion time.



Methods of reading conversion results

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

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

- **RTI Blockset**

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

- **RTLib offers the following polling functions:**

- `adc_tp4_burst_new_read`
- `adc_tp4_single_new_read`

Note

As this read-out method polls for a new buffer with A/D conversion results, it is important that the required number of A/D conversions is triggered by the selected conversion trigger source. Otherwise the application remains in the internal polling loop and hangs up.

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

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

The data ready interrupt indicates that a single conversion or a burst of A/D conversions has finished and the new conversion results can be read using a non-polling read method. The interrupt must be made available by adding an `ADC_TYPE4_HWINT_BLx` block to your Simulink model. The `adc_tp4_data_ready_int_enable` function enables the interrupt in your handcoded C application and with the `dssint_xxx` functions they can be handled.

The RTI block which reads the conversion results and other functions you want to react to the interrupt must be embedded in a subsystem driven by the data ready interrupt. In handcoded C applications, the reading functions and other functions are generally placed in the interrupt service routine. For details on the MicroAutoBox II interrupts, refer to [Interrupt Handling](#) on page 37.

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

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

- RTI Blockset

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

- RTLib offers the following functions which read the current buffer and provide the buffer state:
 - `adc_tp4_burst_current_read`
 - `adc_tp4_burst_immediate_read`
 - `adc_tp4_single_current_read`

To avoid reading the current conversion results repeatedly until a new read buffer is available, the RTLib alternatively offers the `adc_tp4_data_ready` function, which only queries if a new buffer is available but does not read the conversion results.

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

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

- Sample base rate (ADC) means that the `ADC_TYPE4_BLx` block triggers the conversion start and burst start.
- Sample base rate (ADCSTART) means that the `ADC_TYPE4_START_BLx` block triggers the conversion start and burst start.

Time-stamping mode

In time-stamping mode, the free-running 32-bit counter of the time-stamping unit is used to store time stamps of specific A/D conversions. To store the 32-bit value, you need two additional channels. The first channel stores the lower 16 bits of the time stamp, and the second channel stores the higher 16 bits of the time stamp. These two channels are not available for A/D conversion.

The time-stamping channels must be identically configured to the A/D channel for which you want to get the time stamps. A time stamp is captured when the related A/D channel captures data.

The time-stamp counter increments every 10 ns. Your implementation has to handle a counter overflow.

Note

The time-stamping mode is supported by RTLib only. Refer to [adc_tp4_timestamping_mode_set](#) (MicroAutoBox II RTLib Reference ).

Interrupts provided by the ADC Type 4 module

The hardware interrupts from the ADC Type 4 module are used to trigger interrupt-driven tasks which are executed on MicroAutoBox II.

To be used, an interrupt must be enabled. The functions you want to trigger by interrupt must be embedded in an interrupt-driven subsystem. For instructions on doing this, refer to [Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(642aa997563f9a325b310230bb5078b7_img.jpg\)](#)).

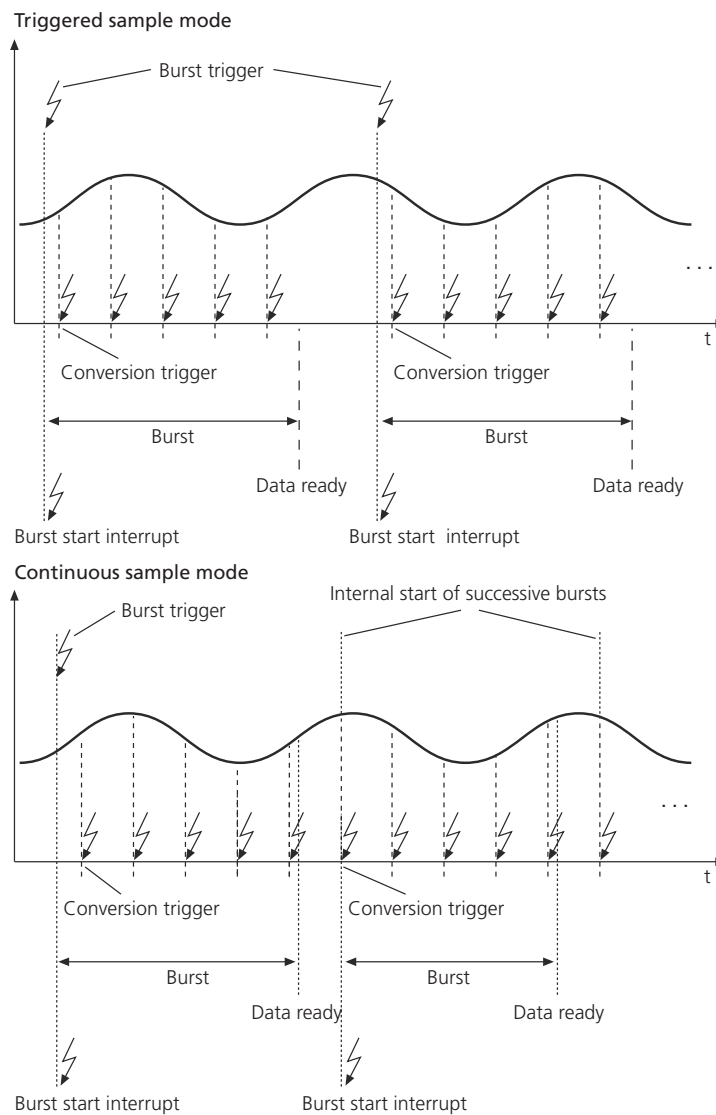
The interrupt which is to trigger the subsystem must be made available using the ADC_TYPE4_HWINT_BLx block. The appropriate channel number and the respective interrupt specification in the block's dialog have to be selected. The model must contain an ADC_TYPE4_BLx block using the specified channel.

Note

You need a separate ADC_TYPE4_HWINT_BLx block for each interrupt that you want to use on a conversion channel.

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

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



The two figures above show the behavior when you use an external trigger input for the conversion trigger. The minimum delay between the burst trigger and the first conversion trigger must be 75 ns. Otherwise the first conversion will not be started.

With a timer as conversion trigger, the burst trigger and the first conversion trigger have no time delay.

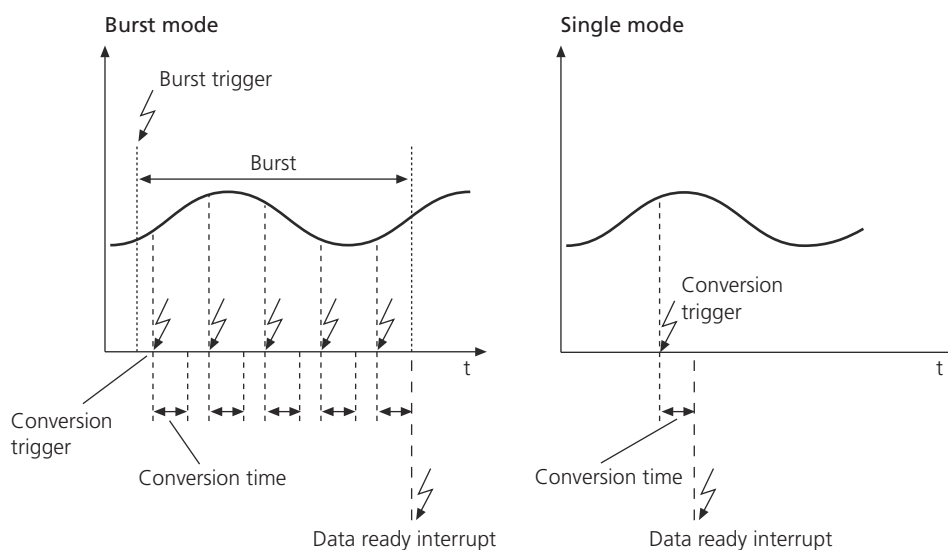
The interrupt can be made available for single conversion mode, but it is of no relevance for possible use cases.

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

For burst conversion mode, the interrupt shows that a conversion burst has finished and the new conversion results are available. If a burst is terminated

before it has reached the number of specified conversions, the interrupt is generated when the current conversion has finished and the incomplete conversion results are available.

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

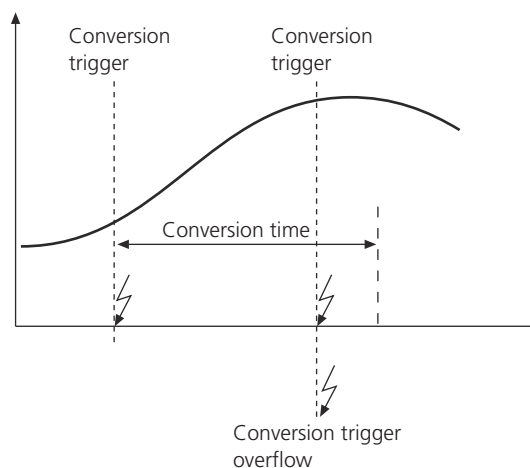


Data lost interrupt The data lost interrupt is provided by the A/D converters, one per channel.

The interrupt shows that a filled free buffer in the swinging buffer was overwritten by new conversion results before the old conversion results could be read. This condition occurs if conversion bursts are started more frequently than the buffers are read. For more information on the swinging buffer, refer to [Swinging Buffer](#) on page 88.

Conversion trigger overflow interrupt The conversion trigger overflow interrupt is provided by the A/D converters, one per channel.

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



Note


A conversion trigger overflow can occur if:

- The signal of the specified external trigger source is noisy.
- The conversion trigger signal period is lower than the conversion time.



I/O mapping

The following table shows the mapping of converter and channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector:

A/D Converter	Channel	Signal	I/O Connector Pin
1	1	Analog ch 1	Z3
2	2	Analog ch 2	Y3
3	3	Analog ch 3	X3
4	4	Analog ch 4	W3
5	5	Analog ch 5	Z4
6	6	Analog ch 6	Y4
7	7	Analog ch 7	X4
8	8	Analog ch 8	W4
9	9	Analog ch 9	Z5
10	10	Analog ch 10	Y5
11	11	Analog ch 11	X5
12	12	Analog ch 12	W5
13	13	Analog ch 13	Z6
14	14	Analog ch 14	Y6
15	15	Analog ch 15	X6
16	16	Analog ch 16	W6
Additional relevant signals ¹⁾			
External trigger input 1		Ana trigger 1	a3
External trigger input 2		Ana trigger 2	a4
External trigger input 3		Ana trigger 3	a5
External trigger input 4		Ana trigger 4	a6

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference\)](#) 
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference\)](#) 

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(6302aad5aed157b291fddf37b4870784_img.jpg\)\)](#)
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(a9ca2c237943a6d0a9f22252f295b6f3_img.jpg\)\)](#)

Related topics

Basics

[Overview of the A/D Conversion Units..... 86](#)
[Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(cbe80b694ebd74fcfe136a095b608235_img.jpg\)\)](#)

References

[ADC Unit Type 4 \(MicroAutoBox II RTLib Reference !\[\]\(cbe2492b119e39e02a1dab2af4a4b296_img.jpg\)\)](#)
[ADC_TYPE4_BLx \(MicroAutoBox II RTI Reference !\[\]\(2f36c159ea3670f7a62f64a4f1cf5c05_img.jpg\)\)](#)
[ADC_TYPE4_HWINT_BLx \(MicroAutoBox II RTI Reference !\[\]\(97ea327f5be815eae3219211de8871e0_img.jpg\)\)](#)
[ADC_TYPE4_START_BLx \(MicroAutoBox II RTI Reference !\[\]\(b9e364404d24453c513f2e1f7e489b5b_img.jpg\)\)](#)

AIO Unit Type 1 (ADC)

Standard A/D converter

The AIO Unit Type 1 provides A/D converters that can be configured for various use cases. The features are described in detail.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Characteristics

The AIO Unit Type 1 consists of 16 parallel A/D converters. They are equipped with single-ended inputs and particularly meet the standard requirements for digitizing analog input signals.

Each of the 16 A/D conversion channels provides:

- Single-ended inputs (negative inputs are internally connected to GND)
- 16-bit resolution and a maximum conversion time of 4 μ s
- Maximum sample rate of 200 KSPS
- An input voltage range of -10 ... +10 V


Conversion behavior

After starting the A/D conversion, the incoming signals are continuously converted in the interval of the sample rate and can be read from a register.



I/O mapping

The following table shows the mapping of converter and channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector (DS1513 ZIF connector):

A/D Converter	Channel	Signal ¹⁾	I/O Connector Pin
1	1	AnalogIn ch 1	V3
2	2	AnalogIn ch 2	U3
3	3	AnalogIn ch 3	T3
4	4	AnalogIn ch 4	S3
5	5	AnalogIn ch 5	V4
6	6	AnalogIn ch 6	U4
7	7	AnalogIn ch 7	T4
8	8	AnalogIn ch 8	S4
9	9	AnalogIn ch 9	V5
10	10	AnalogIn ch 10	U5
11	11	AnalogIn ch 11	T5
12	12	AnalogIn ch 12	S5
13	13	AnalogIn ch 13	V6
14	14	AnalogIn ch 14	U6
15	15	AnalogIn ch 15	T6
16	16	AnalogIn ch 16	S6

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference\)](#) 
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference\)](#) 

Related topics

Basics

[Overview of the A/D Conversion Units..... 86](#)

References

[ADC 1552 Type 2 Unit for DS1552 Multi-I/O Module \(MicroAutoBox II RTLib Reference !\[\]\(74d4806277d7e73349d8e8c0897931e9_img.jpg\)\)](#)
[AIO_TYPE1_ADC_BLx \(MicroAutoBox II RTI Reference !\[\]\(5f42d2cd7ad901bc24e5d35a38c777fd_img.jpg\)\)](#)

ADC 1552 Type 1 Unit

Configurable A/D converter

The ADC Type 1 unit of the DS1552 Multi-I/O Module (abbreviated as *ADC 1552 Type 1*) provides A/D converters that can be configured for various use cases. The features are described in detail.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	–	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module.

Characteristics

The ADC 1552 Type 1 unit consists of 8 parallel A/D converters. They are equipped with single-ended inputs and particularly meet the requirements for digitizing analog input signals at high sample rates, for example, for measuring internal cylinder pressures.

Each of the 8 A/D conversion channels provides:

- Single-ended inputs
(Separated GND pin as reference for each ADC channel, internally connected to GND plane)
- 16-bit resolution and a maximum conversion time of 1 μ s
- Maximum sample rate of 1 MSPS

- The applicable input voltage range depends on the I/O module used:

- DS1552: 0 ... +5 V
- DS1552B1: -10 ... +10 V

If the specified input voltage range does not suit to the connected hardware, the real-time application stops with an error message.

If the DS1552 module has been installed by dSPACE, the board versions are printed on a type plate on the bottom of your MicroAutoBox II.

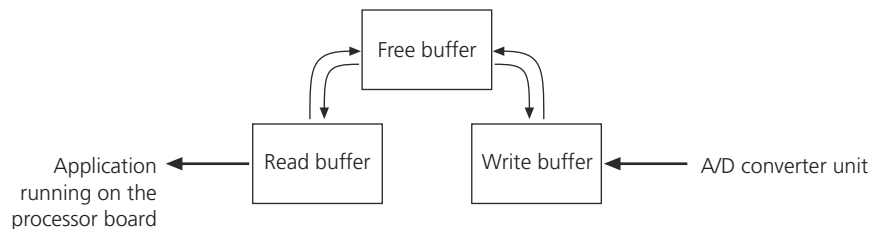
- Single A/D conversion mode to use the channel as a standard A/D converter (see [Single conversion mode](#) on page 103).
- Selectable sources for triggering A/D conversions, for example, external trigger input, channel timer or software trigger (see [Trigger signals](#) on page 103).
- Swinging buffers for decoupling the conversion process from the read process (see [Swinging Buffer](#) on page 102).
- Three independent hardware interrupts associated to the A/D conversion state. For information on ADC 1552 Type 1 interrupt handling, see [Interrupts provided by the ADC 1552 Type 1 unit](#) on page 105.

Note

Basically, the ADC 1552 Type 1 unit works like the ADC Type 4 module, but it does not support the burst sample mode.

Swinging Buffer

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



Swinging buffer principle The swinging buffer, comprising a write, a free, and a read buffer, passes all conversion results from the A/D converter unit to the application running on MicroAutoBox. The buffers can change places, as indicated by the arrows between them in the illustration above. This is implemented by pointer management, so that no buffer needs to be copied from one position to another. The events which trigger the buffer exchange are

described below. The number of temporarily stored conversion results in one buffer is 1.

Write buffer The A/D converter unit writes the conversion result to the write buffer. When this event occurs, the write buffer changes places with the free buffer, which is then filled with the new conversion result.

Read buffer The application running on MicroAutoBox reads the conversion result from the read buffer. Before the conversion result is transferred, the read function used in the application requests a read buffer.

Depending on the read method used by the application,

- The current conversion result in the read buffer is transferred immediately.
- or
- The A/D conversion function waits until the buffer control unit exchanges the read buffer with the free buffer containing a new conversion result.

For more information on the read methods, refer to [Methods of reading conversion results](#) on page 104.

If the application does not read the conversion result fast enough, it can happen that the free buffer, containing a new result, is overwritten by the write buffer before it could become the read buffer. In this case a data lost interrupt is generated on the channel (if the generation of the data lost interrupt was initialized beforehand).

Trigger signals

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

Trigger control If a conversion is in progress and has not completed before another conversion trigger occurs, this trigger is ignored, and a conversion trigger overflow interrupt is generated (if the generation of the conversion trigger overflow interrupt was initialized beforehand). For more information, refer to [Interrupts provided by the ADC 1552 Type 1 unit](#) on page 105.

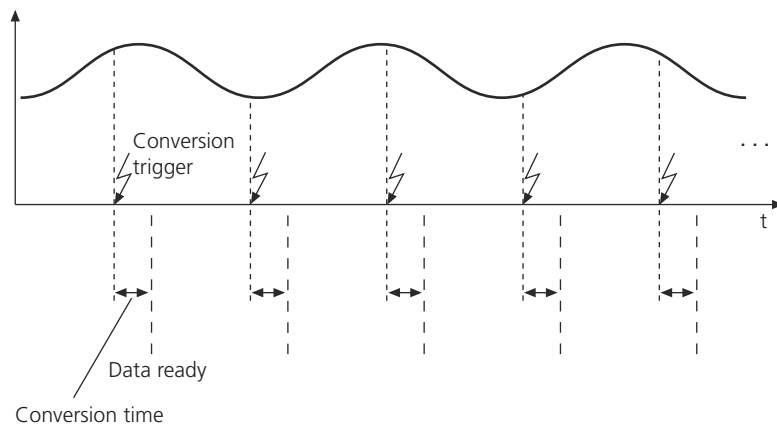
Trigger sources An ADC 1552 Type 1 conversion channel can react to the following trigger sources:

- 4 external trigger inputs
- Individual channel timer for each channel
- Software trigger (using RTLib) and sample base rate (using RTI).

External trigger The external trigger inputs can be used with a maximum frequency of 1 MHz according to the conversion time of 1 μ s. For detailed information on the electrical characteristics, refer to [Data Sheet MicroAutoBox II 1401/1511/1514 \(MicroAutoBox II Hardware Reference !\[\]\(0d7ca0919e6c47bbd874bfa0189fe22e_img.jpg\)](#)).

Single conversion mode

In single conversion mode the conversion channels work as standard A/D converters for converting a single value after receiving a conversion trigger. Each trigger event produces one conversion result which can be read immediately after its conversion time.



Methods of reading conversion results

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

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

- RTI Blockset

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

- RTLib offers the following polling function:

- `adc_1552_tp1_single_new_read`

Note

As this read-out method polls for a new buffer with A/D conversion results, it is important that the required number of A/D conversions is triggered by the selected conversion trigger source. Otherwise the application remains in the internal polling loop and hangs up.

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

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

The data ready interrupt indicates that a new conversion result can be read using a non-polling read method. The interrupt must be made available by adding an ADC1552_TP1_HWINT_BLx block to your Simulink model. The `adc_1552_tp1_data_ready_int_enable` function enables the interrupt in your handcoded C application and with the `dssint_xxx` functions they can be handled.

The RTI block which reads the conversion results and other functions you want to react to the interrupt must be embedded in a subsystem driven by the data ready interrupt. In handcoded C applications the reading functions and other functions

are generally placed in the interrupt service routine. For details on the MicroAutoBox interrupts, refer to [Interrupt Handling](#) on page 37.

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

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

- **RTI Blockset**

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

- **RTLib offers the following function which reads the current buffer and provide the buffer state:**

- `adc_1552_tp1_single_current_read`

To avoid reading the current conversion results repeatedly until a new read buffer is available, the RTLib alternatively offers the `adc_1552_tp1_data_ready` function, which only queries if a new buffer is available but does not read the conversion results.

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

Conversion Mode	Conversion Trigger	Read Method
Single conversion	Sample base rate (ADC)	Polling
	Sample base rate (ADCSTART)	Non-polling
	External trigger	Non-polling
<ul style="list-style-type: none"> ▪ Sample base rate (ADC) means that the ADC1552_TP1_BLx block triggers the conversion start. ▪ Sample base rate (ADCSTART) means that the ADC1552_TP1_START_BLx block triggers the conversion start. 		

Interrupts provided by the ADC 1552 Type 1 unit

The hardware interrupts from the ADC 1552 Type 1 unit are used to trigger interrupt-driven tasks which are executed on MicroAutoBox.

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

The interrupt which is to trigger the subsystem must be made available using the ADC1552_TP1_HWINT_BLx block. The appropriate channel number and the

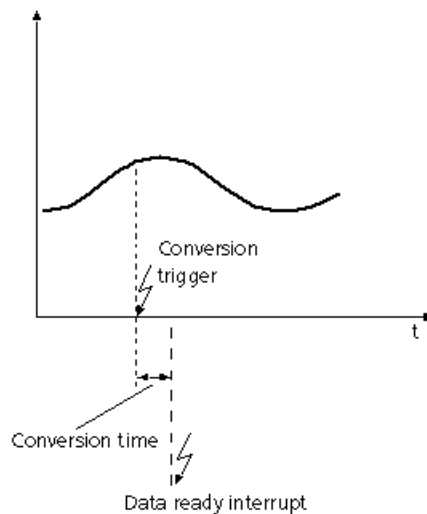
respective interrupt specification in the block's dialog have to be selected. The model must contain an ADC1552_TP1_BLx block using the specified channel.

Note

You need a separate ADC1552_TP1_HWINT_BLx block for each interrupt that you want to use on a conversion channel.

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

The interrupt shows that a single conversion has finished and the new conversion result is available.

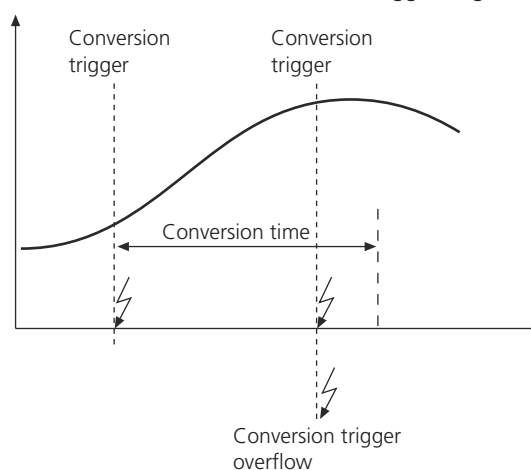


Data lost interrupt The data lost interrupt is provided by the A/D converters, one per channel.

The interrupt shows that a filled free buffer in the swinging buffer was overwritten by new conversion results before the old conversion results could be read. This condition occurs if conversions are started more frequently than the buffers are read. For more information on the swinging buffer, refer to [Swinging Buffer](#) on page 102.

Conversion trigger overflow interrupt The conversion trigger overflow interrupt is provided by the A/D converters, one per channel.

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



Note

A conversion trigger overflow can occur if:

- The signal of the specified external trigger source is noisy.
- The conversion trigger signal period is lower than the conversion time.

I/O mapping

The following table shows the mapping of converter and channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox I/O connector (DS1514 ZIF connector):

A/D Converter	Channel	Signal	I/O Connector Pin
1	1	AnalogIn+ ch 1	X3
		AnalogIn- ch 1 ¹⁾	X4
2	2	AnalogIn+ ch 2	W3
		AnalogIn- ch 2 ¹⁾	W4
3	3	AnalogIn+ ch 3	V3
		AnalogIn- ch 3 ¹⁾	V4
4	4	AnalogIn+ ch 4	U3
		AnalogIn- ch 4 ¹⁾	U4
5	5	AnalogIn+ ch 5	H3
		AnalogIn- ch 5 ¹⁾	H4
6	6	AnalogIn+ ch 6	G3
		AnalogIn- ch 6 ¹⁾	G4
7	7	AnalogIn+ ch 7	F3
		AnalogIn- ch 7 ¹⁾	F4

A/D Converter	Channel	Signal	I/O Connector Pin
8	8	AnalogIn+ ch 8	E3
		AnalogIn- ch 8 ¹⁾	E4
Additional relevant signals			
External trigger input 1		DigIn ch 1	V5
External trigger input 2		DigIn ch 2	U5
External trigger input 3		DigIn ch 3	U6
External trigger input 4		DigIn ch 4	T2

¹⁾ Negative input line of the ADC channel is connected to GND. To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

Note

DigIn ch 1 ... DigIn ch 4 of the DIO 1552 Type 1 unit are shared with the external trigger inputs of the ADC 1552 Type 1 unit.

For a complete overview on the pinout, refer to:





- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference\)](#) 

Related topics

Basics

[Overview of the A/D Conversion Units](#).....86
[Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide\)](#) 

References

[ADC 1552 Type 1 Unit of the DS1552 Multi-I/O Module \(MicroAutoBox II RTLib Reference\)](#) 
[ADC1552_TP1_BLx \(MicroAutoBox II RTI Reference\)](#) 
[ADC1552_TP1_HWINT_BLx \(MicroAutoBox II RTI Reference\)](#) 
[ADC1552_TP1_START_BLx \(MicroAutoBox II RTI Reference\)](#) 

ADC 1552 Type 2 Unit

Standard A/D converter

The ADC Type 2 unit of the DS1552 Multi-I/O Module (abbreviated as *ADC 1552 Type 2*) provides A/D converters that can be configured for various use cases. The features are described in detail.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	–	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module.**Characteristics**

The ADC 1552 Type 2 unit consists of 1 A/D converter with 16 parallel A/D converter channels. The channels are simultaneously sampled. They are equipped with single-ended inputs and particularly meet the standard requirements for digitizing analog input signals.

Each of the 16 A/D conversion channels provides:

- Single-ended inputs (negative inputs are internally connected to GND)
- 16-bit resolution and a maximum conversion time of 4 µs
- Maximum sample rate of 200 KSPS
- An input voltage range of -10 ... +10 V

Conversion behavior


After starting the A/D conversion, the incoming signals are continuously converted in the interval of the sample rate and can be read from a register.

I/O mapping

The following table shows the mapping of converter and channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector (DS1514 ZIF connector):

A/D Converter	Channel	Signal ¹⁾	I/O Connector Pin
1	1	AnalogIn ch 1	b2
2	2	AnalogIn ch 2	a2
3	3	AnalogIn ch 3	Z2
4	4	AnalogIn ch 4	Y2
5	5	AnalogIn ch 5	X2
6	6	AnalogIn ch 6	W2
7	7	AnalogIn ch 7	V2
8	8	AnalogIn ch 8	U2
9	9	AnalogIn ch 9	M2
10	10	AnalogIn ch 10	L2
11	11	AnalogIn ch 11	K2

A/D Converter	Channel	Signal ¹⁾	I/O Connector Pin
12	12	AnalogIn ch 12	J2
13	13	AnalogIn ch 13	H2
14	14	AnalogIn ch 14	G2
15	15	AnalogIn ch 15	F2
16	16	AnalogIn ch 16	E2

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

For a complete overview on the pinout, refer to:


- MicroAutoBox II 1401/1511/1514: [Connector Pinouts](#) (MicroAutoBox II [Hardware Reference](#) )

Related topics

Basics

[Overview of the A/D Conversion Units](#)..... 86

References

[ADC 1552 Type 2 Unit for DS1552 Multi-I/O Module](#) (MicroAutoBox II RTLib [Reference](#) )

[ADC1552_TP2_BLx](#) (MicroAutoBox II RTI [Reference](#) )

Bit I/O

Where to go from here

Information in this section

Overview of the Bit I/O Units.....	111
Bit I/O Unit (DIO Type 3).....	112
Bit I/O Unit (DIO Type 4).....	117
Bit I/O Unit (DIO 1552 Type 1).....	122

Overview of the Bit I/O Units

Introduction

The MicroAutoBox II variants provide various Bit I/O units.

Hardware requirements

Supported MicroAutoBox II hardware:

Bit I/O Unit	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
DIO Type 3	–	✓	–	–
DIO Type 4	–	–	✓	–
DIO 1552 Type 1	–	–	–	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module

Overview of the characteristics

The main characteristics of the Bit I/O units are listed below.

Characteristics	Bit I/O (DIO Type 3)	Bit I/O (DIO Type 4)	Bit I/O (DIO 1552 Type 1)
Digital input channels	40 3 ports with 16/16/8 bits	24 2 ports with 16/8 bits	16
Digital output channels	40 3 ports with 16/16/8 bits	24 2 ports with 16/8 bits	16
Bidirectional channels	None	None	8 ¹⁾

¹⁾ Only available when using the RTI FPGA Programming Blockset

Related topics

Basics

Bit I/O Unit (DIO 1552 Type 1).....	122
Bit I/O Unit (DIO Type 3).....	112
Bit I/O Unit (DIO Type 4).....	117

Bit I/O Unit (DIO Type 3)

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	✓	–	–

Characteristics

The Bit I/O unit of the DIO Type 3 module consists of 80 I/O channels:

- 40 digital input channels
- 40 digital output channels

The digital input channels and the digital output channels are grouped on three ports with different numbers of channels/bits.

Port	Number of Channels/Bits
1	16
2	16
3	8

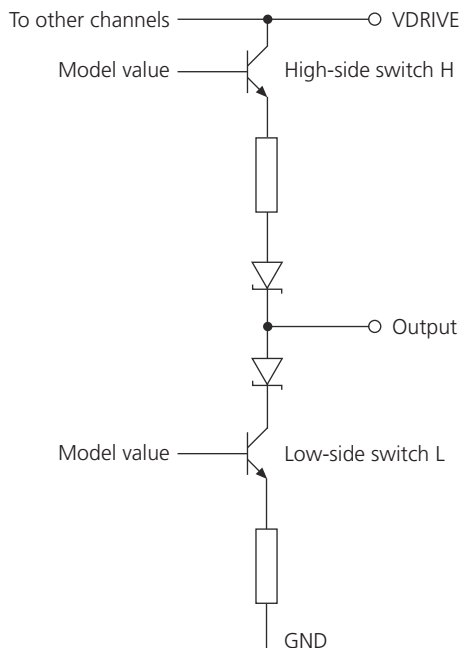
The assignment of I/O channels can only be done within one port. This guarantees data consistency.

After power on, while no application is running, all outputs are set to high impedance state (tristate). For further information, refer to [Signal Descriptions \(MicroAutoBox II Hardware Reference !\[\]\(e3f255517d37bb309a3a931ec4849e6a_img.jpg\)](#)).

Low-side and high-side switches The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.

- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).



Model Value	High-Side Switch (VDRIVE)	Low-Side Switch (GND)	Output (DigP1 ch 1 ... DigP3 ch 8)	Description
0	Disabled	Disabled	High-Z	Individual output disabled
1	Disabled	Disabled	High-Z	
0	Disabled	Enabled	GND	Low-side switch
1	Disabled	Enabled	High-Z	
0	Enabled	Disabled	High-Z	High-side switch
1	Enabled	Disabled	VDRIVE	
0	Enabled	Enabled	GND	Push-pull output
1	Enabled	Enabled	VDRIVE	

For more details, refer to [Data Sheet MicroAutoBox II 1401/1511 \(MicroAutoBox II Hardware Reference !\[\]\(c3d993ca47bfe2a953c700506ce31fa0_img.jpg\)](#)).

Other functions using digital I/O channels

The following functions use digital I/O channels:

- Inputs
 - [PWM Measurement \(PWM2D\) on the DIO Type 3 Unit](#) on page 197
 - [Frequency Measurement \(F2D\) on the DIO Type 3 Unit](#) on page 209
 - [Incremental Encoder Interface on the DIO Type 3 Unit](#) on page 231
 - [SPI: MISO - Serial Peripheral Interface on the DIO Type 3 Unit](#) on page 351

- Outputs
 - [PWM Generation \(PWM\) on the DIO Type 3 Unit](#) on page 147
 - [Square-Wave Signal Generation \(FREQ\) on the DIO Type 3 Unit](#) on page 159
 - [Basics on Multi-Channel PWM Signal Generation on the DIO Type 3](#) on page 169
 - SPI: CLK, MOSI, CS1 ... CS4 - [Serial Peripheral Interface on the DIO Type 3 Unit](#) on page 351

I/O mapping

The following table shows the mapping of the port numbers and channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector.

Digital inputs

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	V2
	2	DigP 1 ch 2	U2
	3	DigP 1 ch 3	T2
	4	DigP 1 ch 4	S2
	5	DigP 1 ch 5	R2
	6	DigP 1 ch 6	P2
	7	DigP 1 ch 7	N2
	8	DigP 1 ch 8	M2
	9	DigP 1 ch 9	V3
	10	DigP 1 ch 10	U3
	11	DigP 1 ch 11	T3
	12	DigP 1 ch 12	S3
	13	DigP 1 ch 13	R3
	14	DigP 1 ch 14	P3
	15	DigP 1 ch 15	N3
	16	DigP 1 ch 16	M3

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	V4
	2	DigP 2 ch 2	U4
	3	DigP 2 ch 3	T4
	4	DigP 2 ch 4	S4
	5	DigP 2 ch 5	R4
	6	DigP 2 ch 6	P4
	7	DigP 2 ch 7	N4
	8	DigP 2 ch 8	M4
	9	DigP 2 ch 9	V5
	10	DigP 2 ch 10	U5
	11	DigP 2 ch 11	T5
	12	DigP 2 ch 12	S5
	13	DigP 2 ch 13	R5
	14	DigP 2 ch 14	P5
	15	DigP 2 ch 15	N5
	16	DigP 2 ch 16	M5
3	1	DigP 3 ch 1	V6
	2	DigP 3 ch 2	U6
	3	DigP 3 ch 3	T6
	4	DigP 3 ch 4	S6
	5	DigP 3 ch 5	R6
	6	DigP 3 ch 6	P6
	7	DigP 3 ch 7	N6
	8	DigP 3 ch 8	M6

¹⁾ DigP <Port number> ch <Channel number>

Digital outputs

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	L2
	2	DigP 1 ch 2	K2
	3	DigP 1 ch 3	J2
	4	DigP 1 ch 4	H2
	5	DigP 1 ch 5	G2
	6	DigP 1 ch 6	F2
	7	DigP 1 ch 7	E2
	8	DigP 1 ch 8	D2
	9	DigP 1 ch 9	L3
	10	DigP 1 ch 10	K3
	11	DigP1 ch 11	J3
	12	DigP 1 ch 12	H3
	13	DigP 1 ch 13	G3
	14	DigP 1 ch 14	F3
	15	DigP 1 ch 15	E3
	16	DigP 1 ch 16	D3
2	1	DigP 2 ch 1	L4
	2	DigP 2 ch 2	K4
	3	DigP 2 ch 3	J4
	4	DigP 2 ch 4	H4
	5	DigP 2 ch 5	G4
	6	DigP 2 ch 6	F4
	7	DigP 2 ch 7	E4
	8	DigP 2 ch 8	D4
	9	DigP 2 ch 9	L5
	10	DigP 2 ch 10	K5
	11	DigP 2 ch 11	J5
	12	DigP 2 ch 12	H5
	13	DigP 2 ch 13	G5
	14	DigP 2 ch 14	F5
	15	DigP 2 ch 15	E5
	16	DigP 2 ch 16	D5

Port	Channel	Signal	I/O Connector Pin
3	1	DigP 3 ch 1	L6
	2	DigP 3 ch 2	K6
	3	DigP 3 ch 3	J6
	4	DigP 3 ch 4	H6
	5	DigP 3 ch 5	G6
	6	DigP 3 ch 6	F6
	7	DigP 3 ch 7	E6
	8	DigP 3 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(ce77bba2916ff045bdb9f4584b191293_img.jpg\)\)](#)
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(b31d4eff00ee94d2cc889725763ab186_img.jpg\)\)](#)

Related topics

Basics

[Hardware Concept..... 12](#)

References

[Bit I/O \(MicroAutoBox II RTLib Reference !\[\]\(8bba887393ca45b761e5cb49e755e762_img.jpg\)\)](#)
[Bit I/O \(MicroAutoBox II RTI Reference !\[\]\(b898b980f2d860cdb0237afbc3664529_img.jpg\)\)](#)

Bit I/O Unit (DIO Type 4)

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Characteristics

The Bit I/O unit of the DIO Type 4 module consists of 48 I/O channels:

- 24 digital input channels
- 24 digital output channels

The digital input channels and the digital output channels are grouped on two ports with different numbers of channels/bits.

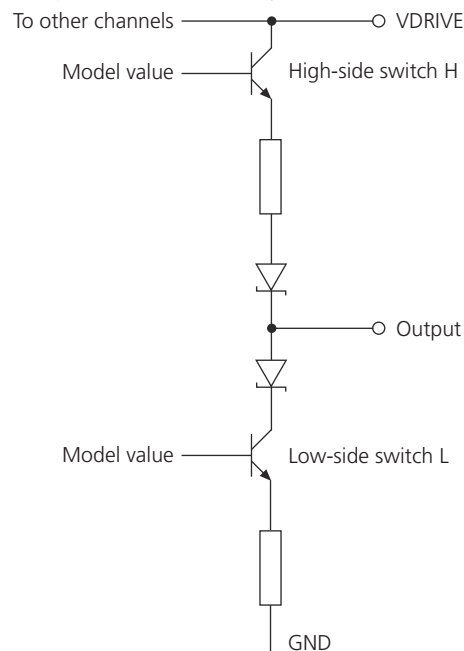
Port	Number of Channels/Bits
1	16
2	8

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

After power on, while no application is running, all outputs are set to high impedance state (tristate). For further information, refer to [Signal Descriptions \(MicroAutoBox II Hardware Reference !\[\]\(10f8862fc183b400327470ea85afe9ae_img.jpg\)](#)).

Low-side and high-side switches The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).



Model Value	High-Side Switch (VDRIVE)	Low-Side Switch (GND)	Output (DigP1 ch 1 ... DigP2 ch 8)	Description
0	Disabled	Disabled	High-Z	Individual output disabled
1	Disabled	Disabled	High-Z	
0	Disabled	Enabled	GND	Low-side switch
1	Disabled	Enabled	High-Z	
0	Enabled	Disabled	High-Z	High-side switch
1	Enabled	Disabled	VDRIVE	
0	Enabled	Enabled	GND	Push-pull output
1	Enabled	Enabled	VDRIVE	

For more details, refer to [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(d84e7ea36f695d92cb39ec32c307ac93_img.jpg\)](#)).

Other functions using digital I/O channels

The following functions use digital I/O channels:

- Inputs
 - [PWM Measurement \(PWM2D\) on the DIO Type 4 Unit](#) on page 202
 - [Frequency Measurement \(F2D\) on the DIO Type 4 Unit](#) on page 213
 - [Incremental Encoder Interface on the DIO Type 4 Unit](#) on page 236
 - [SPI: MISO - Serial Peripheral Interface on the DIO Type 4 Unit](#) on page 360
- Outputs
 - [PWM Generation \(PWM\) on the DIO Type 4 Unit](#) on page 151
 - [Square-Wave Signal Generation \(FREQ\) on the DIO Type 4 Unit](#) on page 162
 - [Basics on Multi-Channel PWM Signal Generation on the DIO Type 4](#) on page 182
 - [SPI: CLK, MOSI, CS1 ... CS4 - Serial Peripheral Interface on the DIO Type 4 Unit](#) on page 360

I/O mapping

The following table shows the mapping of the port numbers and channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector.

Digital inputs

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	M2
	2	DigP 1 ch 2	L2
	3	DigP 1 ch 3	K2
	4	DigP 1 ch 4	J2
	5	DigP 1 ch 5	H2
	6	DigP 1 ch 6	M3
	7	DigP 1 ch 7	L3
	8	DigP 1 ch 8	K3
	9	DigP 1 ch 9	J3
	10	DigP 1 ch 10	H3
	11	DigP 1 ch 11	M4
	12	DigP 1 ch 12	L4
	13	DigP 1 ch 13	K4
	14	DigP 1 ch 14	J4
	15	DigP 1 ch 15	H4
	16	DigP 1 ch 16	M5
2	1	DigP 2 ch 1	L5
	2	DigP 2 ch 2	K5
	3	DigP 2 ch 3	J5
	4	DigP 2 ch 4	H5
	5	DigP 2 ch 5	M6
	6	DigP 2 ch 6	L6
	7	DigP 2 ch 7	K6
	8	DigP 2 ch 8	J6

¹⁾ DigP <Port number> ch <Channel number>

Digital outputs

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	G2
	2	DigP 1 ch 2	F2
	3	DigP 1 ch 3	E2
	4	DigP 1 ch 4	D2
	5	DigP 1 ch 5	C2
	6	DigP 1 ch 6	G3
	7	DigP 1 ch 7	F3
	8	DigP 1 ch 8	E3
	9	DigP 1 ch 9	D3
	10	DigP 1 ch 10	C3
	11	DigP 1 ch 11	G4
	12	DigP 1 ch 12	F4
	13	DigP 1 ch 13	E4
	14	DigP 1 ch 14	D4
	15	DigP 1 ch 15	C4
	16	DigP 1 ch 16	G5
2	1	DigP 2 ch 1	F5
	2	DigP 2 ch 2	E5
	3	DigP 2 ch 3	D5
	4	DigP 2 ch 4	C5
	5	DigP 2 ch 5	G6
	6	DigP 2 ch 6	F6
	7	DigP 2 ch 7	E6
	8	DigP 2 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(30a147af384f9f71632c2ff17bc706c8_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(9b33568d5c136f08ca688ce48be37574_img.jpg\)](#))

Related topics

Basics

[Hardware Concept](#) 12

References

[Bit I/O \(MicroAutoBox II RTLib Reference\)](#)

[Bit I/O \(MicroAutoBox II RTI Reference\)](#)

Bit I/O Unit (DIO 1552 Type 1)

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
—	—	—	—	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module.

Characteristics

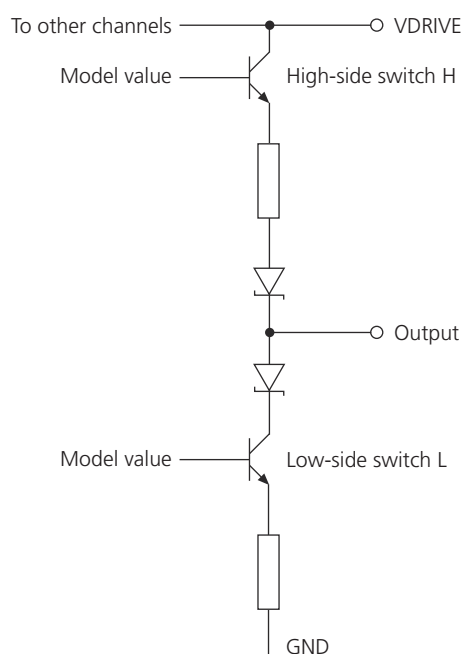
The bit I/O of the DIO Type 1 unit of the DS1552 Multi-I/O Module (abbreviated as *DIO 1552 Type 1*) consists of 32 I/O channels:

- 16 digital input channels
- 16 digital output channels

After power on, while no application is running, all outputs remain disabled in high impedance state (tristate) automatically. For further information, refer to [Signal Descriptions \(MicroAutoBox II Hardware Reference\)](#).

Low-side and high-side switches The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).



Model Value	High-Side Switch (VDRIVE)	Low-Side Switch (GND)	Output (DigOut ch 1 ... DigOut ch 16)	Description
0	Disabled	Disabled	High-Z	Individual output disabled
1	Disabled	Disabled	High-Z	
0	Disabled	Enabled	GND	Low-side switch
1	Disabled	Enabled	High-Z	
0	Enabled	Disabled	High-Z	High-side switch
1	Enabled	Disabled	VDRIVE	
0	Enabled	Enabled	GND	Push-pull output
1	Enabled	Enabled	VDRIVE	

For more details, refer to [Data Sheet MicroAutoBox II 1401/1511/1514](#) (MicroAutoBox II Hardware Reference )

Other functions using digital I/O channels

The following functions use digital I/O channels:

- Inputs
 - [PWM Measurement \(PWM2D\) on the DIO 1552 Type 1 Unit](#) on page 205
 - [Frequency Measurement \(F2D\) on the DIO 1552 Type 1 Unit](#) on page 216
- Outputs
 - [PWM Generation \(PWM\) on the DIO 1552 Type 1 Unit](#) on page 156
 - [Square-Wave Signal Generation \(FREQ\) on the DIO 1552 Type 1 Unit](#) on page 165

I/O mapping

The following table shows the mapping of the channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector (DS1514 ZIF connector).

Digital inputs

Channel	Signal	I/O Connector Pin
1	DigIn ch 1	V5
2	DigIn ch 2	U5
3	DigIn ch 3	U6
4	DigIn ch 4	T2
5	DigIn ch 5	T3
6	DigIn ch 6	T4
7	DigIn ch 7	T5
8	DigIn ch 8	T6
9	DigIn ch 9	S2
10	DigIn ch 10	S3
11	DigIn ch 11	S5
12	DigIn ch 12	R2
13	DigIn ch 13	R5
14	DigIn ch 14	R6
15	DigIn ch 15	P5
16	DigIn ch 16	P6

Note

DigIn ch 1 ... DigIn ch 4 of the DIO 1552 Type 1 unit are shared with the external trigger inputs of the ADC 1552 Type 1 unit.

Digital outputs

Channel	Signal	I/O Connector Pin
1	DigOut ch 1	F5
2	DigOut ch 2	E5
3	DigOut ch 3	E6
4	DigOut ch 4	D2
5	DigOut ch 5	D3
6	DigOut ch 6	D4
7	DigOut ch 7	D5
8	DigOut ch 8	D6
9	DigOut ch 9	C2
10	DigOut ch 10	C3
11	DigOut ch 11	C5

Channel	Signal	I/O Connector Pin
12	DigOut ch 12	B2
13	DigOut ch 13	B5
14	DigOut ch 14	B6
15	DigOut ch 15	A5
16	DigOut ch 16	A6

For a complete overview on the pinout, refer to:


- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(86b7331e04fe40a56bcff2e9c065738b_img.jpg\)\)](#)

Related topics

Basics

[Hardware Concept..... 12](#)

References

Bit I/O on the DIO 1552 Type 1 Unit of the DS1552 Multi-I/O Module
 (MicroAutoBox II RTLib Reference )
[DIO1552_TP1_BIT_IN_BLx \(MicroAutoBox II RTI Reference !\[\]\(740312fd467f47b04cab841ab3868d83_img.jpg\)\)](#)
[DIO1552_TP1_BIT_IN_CH_BLx \(MicroAutoBox II RTI Reference !\[\]\(dbb8da2687e90ededffd3484b6b666cf_img.jpg\)\)](#)
[DIO1552_TP1_BIT_OUT_BLx \(MicroAutoBox II RTI Reference !\[\]\(a571c88051e93c7a1a7313cd64ac7c59_img.jpg\)\)](#)
[DIO1552_TP1_BIT_OUT_CH_BLx \(MicroAutoBox II RTI Reference !\[\]\(0cc57fb16357458f6f0d10999171155d_img.jpg\)\)](#)

D/A Conversion

Where to go from here

Information in this section

Overview of the D/A Conversion Units.....	126
AIO Unit Type 1 (DAC).....	127
DAC Unit Type 3.....	128
DAC 1552 Type 1 Unit.....	129

Overview of the D/A Conversion Units

Introduction

The MicroAutoBox II variants provide various D/A conversion units.

Hardware requirements

Supported MicroAutoBox II hardware:

DAC Unit	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
AIO Type 1 DAC	–	–	✓	–
DAC Type 3	–	✓	–	–
DAC 1552 Type 1	–	–	–	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module

Overview of the characteristics

The main characteristics of the DAC units are listed below.

Characteristics	AIO Type 1 DAC	DAC Type 3	DAC 1552 Type 1
Number of converter channels	8	4	4
Output voltage range	-10.0 ... +10.0 V	0 ... +4.5 V	0 ... +5.0 V
Resolution	16 bit	12 bit	16 bit

Related topics

Basics

AIO Unit Type 1 (DAC).....	127
DAC 1552 Type 1 Unit.....	129
DAC Unit Type 3.....	128

AIO Unit Type 1 (DAC)

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–


Characteristics

The AIO Unit Type 1 provides 8 outputs with 16-bit resolution and 1 μ s maximum settling time. The outputs have -10.0 ... +10.0 V voltage, low pass output filters (1st order/3 dB at 500 kHz), 60 V overvoltage protection, overcurrent and short circuit protection. The maximum sink/source current is 8 mA.

I/O mapping

The following table shows the mapping of the converter and channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector (DS1513 ZIF connector):

D/A Channel	Signal ¹⁾	I/O Connector Pin
1	AnalogOut ch 1	Z2
2	AnalogOut ch 2	Y2
3	AnalogOut ch 3	X2
4	AnalogOut ch 4	W2
5	AnalogOut ch 5	V2
6	AnalogOut ch 6	U2
7	AnalogOut ch 7	T2
8	AnalogOut ch 8	S2

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(9063468a59e93f469b71000ac5796bc3_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(1db6320223680ab4bd04b0d269ab6c8a_img.jpg\)](#))

Related topics

References

[AIO_TYPE1_DAC_BLx \(MicroAutoBox II RTI Reference !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)](#))

[DAC 1552 Type 1 Unit of the DS1552 Multi-I/O Module \(MicroAutoBox II RTLib Reference !\[\]\(e658400d40ca763c7cf4c8c420885c6a_img.jpg\)](#))

DAC Unit Type 3

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	✓	–	–

Characteristics


The DAC unit provides 4 outputs with 12-bit resolution and 150 μ s maximum settling time (to 1 LSB). The outputs have 0 ... +4.5 V voltage, low pass output filters (1st order/3 dB at 10 kHz), 40 V overvoltage protection, overcurrent and short circuit protection. The maximum sink/source current is 5 mA.

I/O mapping



The following table shows the mapping of the logical converter and channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector (DS1511 ZIF connector):

D/A Channel	Signal ¹⁾	I/O Connector Pin
1	Analog ch 1	Z2
2	Analog ch 2	Y2

D/A Channel	Signal ¹⁾	I/O Connector Pin
3	Analog ch 3	X2
4	Analog ch 4	W2



¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference](#) )
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference](#) )

Related topics

References

[DAC Unit Type 3 \(MicroAutoBox II RTLib Reference](#) )
[DAC_TYPE3_Mx_Cy \(MicroAutoBox II RTI Reference](#) )

DAC 1552 Type 1 Unit

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	–	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module.


Characteristics

The DAC Type 1 unit of the DS1552 Multi-I/O Module (abbreviated as *DAC 1552 Type 1*) provides 4 outputs with 16-bit resolution and 1 µs maximum settling time. The outputs have 0 ... +5.0 V voltage, low pass output filters (1st order/3 dB at 500 kHz), 60 V overvoltage protection, overcurrent and short circuit protection. The maximum sink/source current is 8 mA.

I/O mapping

The following table shows the mapping of the converter and channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector (DS1514 ZIF connector):

D/A Channel	Signal ¹⁾	I/O Connector Pin
1	AnalogOut ch 1	c2
2	AnalogOut ch 2	c3
3	AnalogOut ch 3	c4
4	AnalogOut ch 4	c5

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference](#) )

Related topics

References

[DAC 1552 Type 1 Unit of the DS1552 Multi-I/O Module \(MicroAutoBox II RTLib Reference](#) )
[DAC1552_TP1_BLx \(MicroAutoBox II RTI Reference](#) )

Ethernet I/O Interface

Introduction

To perform ECU bypassing with optimized data transfer latencies in comparison to a standard Ethernet interface.

Hardware requirements


Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓	–	–	–	–

Basics on the Ethernet I/O Interface

Field of application

You can use the Ethernet I/O Interface to perform the following I/O activities:

- Bypassing an ECU, that means, that a subset of ECU functions is executed by the MicroAutoBox II by using the RTI Bypass Blockset and the XCP on UDP/IP Protocol.
- Performing communication between the real-time application and any external device that also provides an Ethernet interface using UDP/IP protocol, such as another dSPACE board or a calibration device. The RTI Ethernet (UDP) Blockset lets you implement the communication in a Simulink® model. For further information, refer to [RTI Ethernet \(UDP\) Blockset Reference](#) .

Data transfer

The Ethernet I/O Interface allows 1 GBit connections and supports the UDP/IP protocol. This interface is optimized to provide lowest data transfer latencies and highest data transfer rates.

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

- You can use up to 4 sockets.
- Each socket is used for bidirectional communication.
- Each socket can be configured with a maximum datagram size of 1472 bytes.
- Auto negotiation is supported.
- Listening to any IP address and port is supported.

- IP fragmentation is not supported. Each UDP message is limited to the maximum Ethernet datagram size.
- DHCP is not supported.
- Routing is not supported. All participants in communication must be available in the same subnet.

XCP on UDP/IP In combination with the RTI Bypass Blockset, the Ethernet I/O Interface lets you access external systems via the XCP on UDP/IP protocol. This protocol allows you to access ECUs for bypassing, controlling, sensor emulation, etc. Additionally, it allows you to access modules providing an XCP on UDP/IP service interface, for example, to read values from modules which incorporate specific measurement technology.

Specifying the IP addresses

As the IP protocol is used, you must specify the IP addresses of the Ethernet I/O Interface (source) and the target device (e.g., ECU). With the RTI Bypass Blockset, the RTIBYPASS_SETUP_BLx block provides the relevant IP address parameters, refer to [Options Page \(RTIBYPASS_SETUP_BLx for XCP on UDP/IP\) \(RTI Bypass Blockset Reference !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)](#)). Using the RTI Ethernet (UDP) Blockset, the ETHERNET_UDP_SETUP_BLx block provides the relevant IP address parameters, refer to [ETHERNET_UDP_SETUP_BLx \(RTI Ethernet \(UDP\) Blockset Reference !\[\]\(034433b90593e82e5460e34e3ed48e9b_img.jpg\)](#).

Related topics

Basics

[Features of the RTI Bypass Blockset \(RTI Bypass Blockset Reference !\[\]\(ec9132f1d27c8919987d92907322654d_img.jpg\)](#))

HowTos

[How to Connect a MicroAutoBox II to an ECU with XCP on Ethernet \(UDP/IP\) \(ECU Interfaces Hardware Installation and Configuration !\[\]\(aa53ad6fea213b8b2226d3077e30533a_img.jpg\)](#))

References

[RTI Ethernet \(UDP\) Blockset Reference](#)

ECU Interface

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓	✓	–	–	–

Where to go from here

Information in this section

General Description.....	133
Hardware.....	134
Working Modes.....	136
DPMEM Addresses Seen from the ECU.....	137
Data Type Formats.....	139
ECU Interrupts.....	140

General Description

Introduction

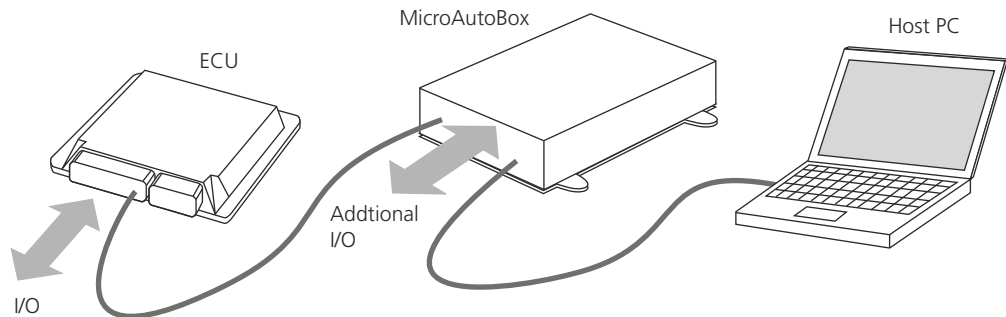
The ECU interfaces of MicroAutoBox II can be used for several purposes, for example:

- Connecting RapidPro Control Unit to MicroAutoBox II, refer to [Connecting a Control Unit and a RCP System via LVDS \(RapidPro System Hardware Installation Guide !\[\]\(c44db1e92ba1244b2894d325c806ff8a_img.jpg\)](#)).
- Communicating via Ethernet using an LVDS Ethernet link cable
For further information, refer to [Basics on the Ethernet I/O Interface](#) on page 131.
- Bypass-based prototyping, see the following descriptions.

Bypass-based prototyping

In bypass-based prototyping, existing ECUs are optimized or partially revised to obtain a new control strategy. The following illustration shows an automotive ECU connected to MicroAutoBox II via an ECU interface. The ECU executes all

the control functions that will remain unchanged, while the new algorithms are calculated in MicroAutoBox II. The results will then be transmitted back to the original ECU.

**Note**

In typical applications, a custom-specific plug-on device (target adapter) is necessary. The target adapter provides the physical connection between MicroAutoBox II and the ECU and performs signal conditioning, for example.

Related topics**Basics**

ECU Interface.....	133
Hardware Concept.....	12

References

- [ECU Interface Unit \(MicroAutoBox II RTLib Reference !\[\]\(c33cb967c8fc4f5e27188a389b621c8e_img.jpg\)\)](#)
- [ECU Interface Unit \(MicroAutoBox II RTI Reference !\[\]\(38e1383487ca0f0e9e2c9378b9dbcae7_img.jpg\)\)](#)

Hardware

Introduction

The ECU interface unit provides a serial interface and a 16k · 16-bit dual-port memory for accessing an ECU. The communication between the ECU and MicroAutoBox II can be controlled by subinterrupts.

For example, if a DPMEM POD is connected to the ECU interface, you can access ECU variables using the RTI Bypass Blockset. The interface-specific parameters are to be provided by an A2L file.

I/O mapping

ECU interface channels The following table shows the available ECU interfaces (ECUx) or ECU interface channels (IFx) according to MicroAutoBox II variant. The I/O connector type of each ECU interface is also shown (ZIF or LEMO I/O connector).

The ECU interface channel number is used to specify the module. For example, if you want to use IF2, you have to set the Module number to 2 in your RTI block, or the ModuleAddr parameter to ECU_TP1_2_MODULE_ADDR in your RTLib function.

MicroAutoBox II Variant	ECU 1 (ZIF)	ECU 2 (LEMO)	ECU A (LEMO)	ECU B (LEMO)	ECU C (LEMO)
MicroAutoBox II 1401/1507	-	-	IF2	IF3	IF1
MicroAutoBox II 1401/1511	-	-	IF1	IF2	-
MicroAutoBox II 1401/1511/1514	-	-	IF1	IF2	-
MicroAutoBox II 1401/1513	-	-	IF1	IF2	-
MicroAutoBox II 1401/1513/1514	-	-	IF1	IF2	-

I/O mapping of ECU 1 The following table shows the ECU interface channel 1 (IF1) I/O pins of the ZIF I/O connector:

Signal	I/O Connector Pin
ECU / IF1 RX +	Y5
ECU / IF1 RX –	Z5
ECU / IF1 TX +	b5
ECU / IF1 TX –	a5

Note

This memory-based ECU bypassing interface is proprietary to dSPACE. Connect the ZIF-RX pair to the ECU-TX pair and the ZIF-TX pair to the ECU-RX pair. Always use a CAT5 twisted pair cable to make this connection.

I/O mapping of ECU 2, ECU A, ECU B and ECU C The pinout of the LEMO I/O connectors is not documented. Use a suitable dSPACE connection cable.

For hardware details, refer to [MicroAutoBox II Hardware Reference](#) .



Related topics

Basics

DPMEM Addresses Seen from the ECU..... 137

ECU Interrupts.....	140
MicroAutoBox II Base Board.....	28

References

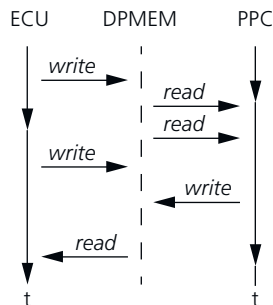
ECU Interface Unit (MicroAutoBox II RTLib Reference )
 ECU Interface Unit (MicroAutoBox II RTI Reference )

Working Modes

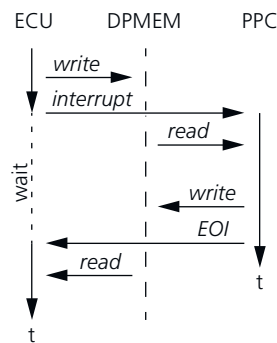
Introduction

The ECU interface supports three working modes. In your ECU program, you determine the mode to be used.

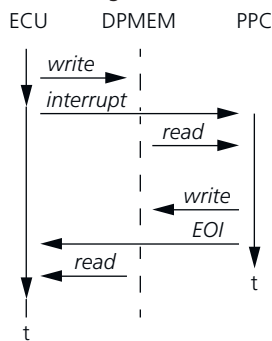
Asynchronous mode In this mode no synchronization between ECU and MicroAutoBox II (PPC) is performed. The ECU does not generate interrupts and does not wait for status messages from the PPC. You will always read the current value from the DPMEM, independently whether an expected value was written beforehand. This can result in data inconsistencies or even data losses. This mode is therefore used for parameter monitoring and for the tuning of parameters not written or written with a low period by the ECU application.



Synchronous mode The synchronization between ECU and PPC is performed by an interrupt mechanism. For Simulink models, the interrupt (from ECU to PPC) triggers an interrupt-driven subsystem. In this subsystem, the read and write access to the DPMEM is performed: The ECU writes the value to the DPMEM, generates an interrupt and waits for the *End of interrupt* message (EOI) from the PPC. The PPC reads this value, performs the necessary calculations, writes the calculated new value back to the DPMEM and generates the *End of interrupt* message. Following, the ECU will read the calculated new value. This mode is used for modifying ECU programs.



Semi-synchronous mode This – simulated bypassing – mode corresponds to the synchronous mode except that the ECU does not wait for an *End of interrupt* message from the PPC. Data inconsistencies may occur if the ECU reads data from the DPMEM before the calculations of the PPC in the interrupt routine or the interrupt-driven subsystem are finished. To ensure correct data the ECU should access the DPMEM not before the *End of interrupt* message from the PPC has been generated or after the PPC has finished its calculations.



Related topics

Basics

[ECU Interface..... 133](#)

References

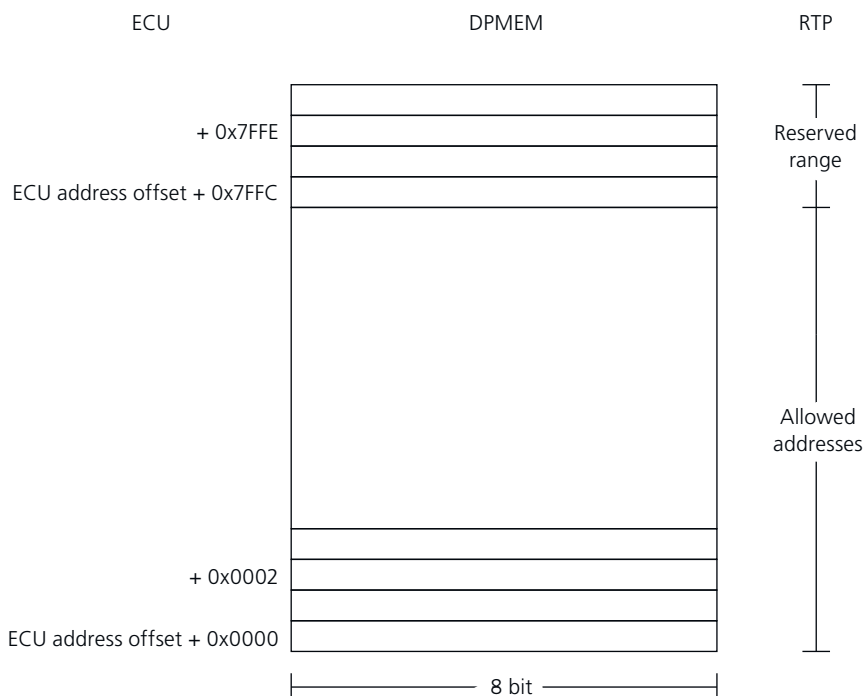
[ECU Interface Unit \(MicroAutoBox II RTLib Reference !\[\]\(17413706fd4997a1a4bdf85c6864eee1_img.jpg\)\)](#)
[ECU Interface Unit \(MicroAutoBox II RTI Reference !\[\]\(f419710cbe076aa30a9c6c031b5cbe84_img.jpg\)\)](#)

DPMEM Addresses Seen from the ECU

Introduction

To use the same DPMEM addresses in the Simulink dialogs dSPACE's RTI converts the DPMEM addresses from the view of the ECU to the 16-bit view of the Master PPC. The conversion algorithm depends on the ECU offset address and other parameters specified in the ECU configuration file. The ECU address offset

specifies the start address of the DPMEM address range seen from the ECU side. The following illustration shows the DPMEM from the view of the ECU:



Note

These addresses and address ranges are valid for ECUs supporting the same DPMEM size as MicroAutoBox II. For ECUs supporting a smaller address range, the addresses have to be adapted.

Address ranges

The following DPMEM addresses and address ranges are specified on MicroAutoBox II. To access these addresses from the ECU side you have to add the address offset.

Addresses and address ranges (ECU side)	Address name	Meaning
+ 0x7FFE	ConnectionAddress	Connection between ECU and PPC
+ 0x7FFC	InterruptAddress	Interrupt address (interrupt ECU to PPC) to generate the physical interrupt
+ 0x0000 ... 0x7FDE		Usable range (specified in the ECU configuration file)

In addition to these addresses, a range of $30 + x$ (with x = number of subinterrupts to be used) successive DPMEM addresses is required for subinterrupt handling. It is recommended to allocate this area at the start or the

end of the range "Allowed addresses" to preserve a consecutive range for the data transfer.

Related topics

Basics

[ECU Interface..... 133](#)

References

[ECU Interface Unit \(MicroAutoBox II RTLib Reference !\[\]\(cbe2492b119e39e02a1dab2af4a4b296_img.jpg\)\)](#)
[ECU Interface Unit \(MicroAutoBox II RTI Reference !\[\]\(2f36c159ea3670f7a62f64a4f1cf5c05_img.jpg\)\)](#)

Data Type Formats

Introduction

Existing ECUs use different data formats. dSPACE's RTI allows to select the following data types:

Data Type
16 bit signed integer
16 bit unsigned integer
32 bit signed integer
32 bit unsigned integer
Single float 32 bit (IEEE Std. 754)

dSPACE software ...

- reads bit patterns from the DPMEM and provides the data in the given format for processing in Simulink or
- provides data of a Simulink variable in the given format to be written to the DPMEM.

Note

For the data formats, two or more subsequent addresses are used. For 16-bit data, only even addresses have to be used, for 32-bit data the addresses have to be multiples of 4. This refers to the 8-bit view of the DPMEM from the ECU. Consider this, when entering the read or write address for absolute addressing. Relative addressing takes this into account automatically.

RTI provides a check on overlapping DPMEM addresses. If you use handcoded models, you have to ensure correct addresses yourself.

Related topics**Basics**

[ECU Interface..... 133](#)

References

[ECU Interface Unit \(MicroAutoBox II RTLib Reference !\[\]\(23d9fc146e83b5c3013cfa32c784f8d5_img.jpg\)\)](#)
[ECU Interface Unit \(MicroAutoBox II RTI Reference !\[\]\(f5c463b8c1554ac5049d611bd8e33a51_img.jpg\)\)](#)
[ECU_TYPE1_READ_Mx_BLy \(MicroAutoBox II RTI Reference !\[\]\(54f1390f33a36173a1b97c4b6eb40204_img.jpg\)\)](#)
[ECU_TYPE1_WRITE_Mx_BLy \(MicroAutoBox II RTI Reference !\[\]\(1301e78e125668a3a0cedabdef0db7f3_img.jpg\)\)](#)

ECU Interrupts

Introduction

The ECU can generate an interrupt to MicroAutoBox II's Master PPC.

Subinterrupts on the ECU side

On the ECU side, the ECU-to-PPC interrupt can be split into 16 subinterrupts via ECU Software Porting Kit.

Note

If you do not use dSPACE's ECU Software Porting Kit, you have to program an equivalent subinterrupt handling yourself to ensure proper communication between ECU and MicroAutoBox II (PPC).

Subinterrupts on the PPC side

Interrupt-driven subsystems in RTI You can use interrupts or subinterrupts to trigger interrupt-driven subsystems of your Simulink model. For example, this allows you to perform the read and write operations from or to the DPMEM. If you use dSPACE's RTI interrupt blocks the interrupt handling will be performed automatically on the PPC side. When the task in this system has finished, the interrupt handling creates automatically an *End of interrupt* (EOI) message to indicate the state for other units, an ECU, for example.



Handcoded models If you use handcoded models, you can use the functions provided by dSPACE's Word-Based Subinterrupt Handling to program the subinterrupt handling on the PPC side yourself.

Related topics**Basics**

[Basics on Interrupt Handling..... 37](#)
[Basics on Subinterrupt Handling..... 44](#)
[ECU Interface..... 133](#)

ECU Software Porting Kit.....	142
Word-Based Subinterrupt Handling.....	40

References

ECU Interface Unit (MicroAutoBox II RTLib Reference )
ECU Interface Unit (MicroAutoBox II RTI Reference )

ECU Software Porting Kit

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
✓	✓	–	–	–

Working with the ECU Software Porting Kit

Basics

The ECU Software Porting Kit allows you to use up to 16 subinterrupts (0 ... 15) from the ECU to MicroAutoBox II. You can use the subinterrupts without detailed knowledge of the subinterrupt functionality.

The settings needed for your specific ECU must be specified in the `dsECUframe.c` file. This file allows you to define, among others, the type of the ECU interface board and the DPMEM addresses used for (sub-)interrupt handling. All other required information are included in this file.

Tip

You have to modify existing ECU code only if you want to use subinterrupts in your application.

For further information, refer to [ECU Software Porting Kit \(MicroAutoBox II RTLib Reference !\[\]\(35dc653d59570f8f891c312eeece91a2_img.jpg\)](#)).

Using the ECU Software Porting Kit

If you want to use the ECU Software Porting Kit to set subinterrupts you have to perform the following steps:

- Create the working directory for the ECU application and copy `dsECUframe.c` to this directory. For details, refer to [dsECUnew.bat \(MicroAutoBox II RTLib Reference !\[\]\(09885fa7dbc7efea01a3982f2e00fbcd_img.jpg\)](#)).
- For existing ECU projects copy this file to the working directory.
- Specify the settings for your specific ECU in the file `dsECUframe.c`. For details refer to [Defines to be Specified in the dsECUframe.c File \(MicroAutoBox II RTLib Reference !\[\]\(efbba78d414c0d979a2d6cb4f74c9442_img.jpg\)](#)).
- Include `dsECUframe.c` into your application.

- You can now perform subinterrupt handling in your application code:
 - DSECU_SINT_INIT calls `dsecu_define_sender` to initialize the interrupt sender.
 - DSSINT_SINT_SEND calls `dsecu_sint_send` to trigger a subinterrupt and set the end-of-subinterrupt flag.
 - DSECU_EOSI_POLL polls the end-of-subinterrupt flag.

Example

The following example demonstrates how to use the ECU Software Porting Kit:

```
#include <stdlib.h>
#include <dseCUframe.c>
void main(void)
{
    unsigned int sint_number;
    ();
    while(1)
    {
        static int first_alive_flag = 1;    /* used for revive */
        if (!first_alive_flag)
        {
            (&dssint_sender); /* first alive only one time to revive ECU */
            first_alive_flag = 1;
        }
        sint_number = rand();               /* random subinterrupt */
        write_bypass_data(sint_number);
        (sint_number);
        if ((sint_number) == DSECU_BYPASS_DATA_VALID)
        {
            read_bypass_data(sint_number);
        }
    }
}
```

Related topics

References

[dsecu_define_sender \(MicroAutoBox II RTLib Reference !\[\]\(642aa997563f9a325b310230bb5078b7_img.jpg\)\)](#)
[DSECU_EOSI_POLL \(MicroAutoBox II RTLib Reference !\[\]\(9bef82f5a53106f2ad06a2de7acf5bcf_img.jpg\)\)](#)
[DSECU_SINT_INIT \(MicroAutoBox II RTLib Reference !\[\]\(7ed4b959e7161d2c60a33aeb43710ff2_img.jpg\)\)](#)
[dsecu_sint_send \(MicroAutoBox II RTLib Reference !\[\]\(9a1c9bf02665d1d8af419e98d46187a2_img.jpg\)\)](#)
[dsecu_startup \(MicroAutoBox II RTLib Reference !\[\]\(0eb1c3fb8762ba6f12cea583077849e5_img.jpg\)\)](#)

PWM Signal Generation with a Variable Period

Introduction

MicroAutoBox II provides the timing I/O feature to generate pulse-width modulated signals with a variable period.

Where to go from here

Information in this section

Overview of the Signal Generation Functionalities.....	144
PWM Generation (PWM) on the DIO Type 3 Unit.....	147
PWM Generation (PWM) on the DIO Type 4 Unit.....	151
PWM Generation (PWM) on the DIO 1552 Type 1 Unit.....	156
Square-Wave Signal Generation (FREQ) on the DIO Type 3 Unit.....	159
Square-Wave Signal Generation (FREQ) on the DIO Type 4 Unit.....	162
Square-Wave Signal Generation (FREQ) on the DIO 1552 Type 1 Unit.....	165

Overview of the Signal Generation Functionalities

Introduction

MicroAutoBox II provides various methods for signal generation depending on the available hardware.

Hardware requirements

The signal generation functions can be divided into the following three main groups:

- PWM signal generation with a variable period (PWM)
- Square-wave signal generation (FREQ)
- Multi-channel PWM signal generation (MC_PWM)

Supported MicroAutoBox II hardware:

Signal Generation	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
PWM	–	✓	✓	✓ ¹⁾
FREQ	–	✓	✓	✓ ¹⁾
MC_PWM	–	✓	✓	–

¹⁾ Requires a DS1552 Multi-I/O Module.

Overview of the characteristics

Characteristics of the functions for signal generation with variable period and duty cycle The following table shows the main characteristics. For detailed information, refer to the function descriptions.

Characteristics	PWM (DIO Type 3)	PWM (DIO Type 4)	PWM (DIO 1552 Type 1)
Number of output channels	40 (3 ports with 16/16/8 channels)	24 (2 ports with 16/8 channels)	16
Signal polarity	Active high Active low		
Run-time adjustable parameter	Period Duty cycle		
Update mode	Synchronous Asynchronous		
Period range	6.7 μ s ... 107.3 s Divided into 16 subranges		
Resolution	50 ns ... 1.64 ms Depends on the specified range.		

Characteristics of the functions for signal generation with a variable period The following table shows the main characteristics. For detailed information, refer to the function descriptions.

Characteristics	FREQ (DIO Type 3)	FREQ (DIO Type 4)	FREQ (DIO 1552 Type 1)
Number of output channels	40 (3 ports with 16/16/8 channels)	24 (2 ports with 16/8 channels)	16
Signal polarity	Active high Active low		
Run-time adjustable parameter	Period		
Update mode	Synchronous		
Period range	6.7 μ s ... 333.3 s Divided into 16 subranges		

Characteristics	FREQ (DIO Type 3)	FREQ (DIO Type 4)	FREQ (DIO 1552 Type 1)
Resolution	50 ns ... 1.64 ms Depends on the specified range.		

Characteristics of the functions for multi-channel PWM signal

generation The following table shows the main characteristics. For detailed information, refer to the function descriptions.

Characteristics	MC_PWM (DIO Type 3)	MC_PWM (DIO Type 4)
Number of output channels	40 (3 ports with 16/16/8 channels)	24 (2 ports with 16/8 channels)
Alignment mode	<ul style="list-style-type: none"> ▪ Edge-aligned ▪ Center-aligned 	
Inverted signals	Supported	
Run-time adjustable parameters	<ul style="list-style-type: none"> ▪ Period ▪ Duty cycle 	
Dead time	Supported ¹⁾	
Update mode	Edge-aligned mode: <ul style="list-style-type: none"> ▪ Synchronized to rising edges Center-aligned mode: <ul style="list-style-type: none"> ▪ Synchronized to the center of the high pulses (middle of a period) ▪ Synchronized to the center of the low pulses (start of a period) ▪ Combination of both 	
Period range	6.7 μ s ... 107.3 s Divided into 16 subranges	
Resolution	50 ns ... 1.64 ms Depends on the specified range.	
Interrupt generation	Interrupt generation after 1 ... 255 periods.	
Trigger generation	Trigger generation after 1 ... 255 periods.	
Adjustable event (interrupt/trigger) position	Supported	

¹⁾ Depends on selected alignment mode.

Related topics**Basics**

Basics on Multi-Channel PWM Signal Generation on the DIO Type 3.....	169
Basics on Multi-Channel PWM Signal Generation on the DIO Type 4.....	182
PWM Generation (PWM) on the DIO 1552 Type 1 Unit.....	156
PWM Generation (PWM) on the DIO Type 3 Unit.....	147
PWM Generation (PWM) on the DIO Type 4 Unit.....	151
Square-Wave Signal Generation (FREQ) on the DIO 1552 Type 1 Unit.....	165
Square-Wave Signal Generation (FREQ) on the DIO Type 3 Unit.....	159
Square-Wave Signal Generation (FREQ) on the DIO Type 4 Unit.....	162

PWM Generation (PWM) on the DIO Type 3 Unit

Purpose

To generate a PWM signal with the period and duty cycle adjustable during run time.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	✓	–	–

Characteristics

40 digital outputs are available for generating PWM signals. These outputs are shared with the other DIO Type 3 output functions. The output channels are grouped to three ports with a different number of channels.

Port	Number of Channels/Bits
1	16
2	16
3	8

The following ranges are defined for PWM signal generation:

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
1	100 ns ¹⁾	3.27 ms	50 ns
2	200 ns ¹⁾	6.55 ms	100 ns
3	400 ns ¹⁾	13.1 ms	200 ns
4	800 ns ¹⁾	26.2 ms	400 ns
5	1.6 µs ¹⁾	52.4 ms	800 ns
6	3.2 µs ¹⁾	104 ms	1.6 µs
7	6.4 µs ¹⁾	209 ms	3.2 µs
8	12.8 µs	419 ms	6.4 µs
9	25.6 µs	838 ms	12.8 µs
10	51.2 µs	1.67 s	25.6 µs
11	103 µs	3.35 s	51.2 µs

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
12	205 μ s	6.71 s	103 μ s
13	410 μ s	13.4 s	205 μ s
14	820 μ s	26.8 s	410 μ s
15	1.64 ms	53.6 s	820 μ s
16	3.28 ms	107.3 s	1.64 ms

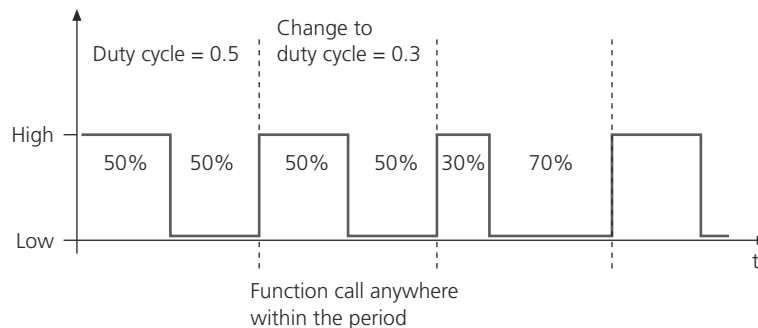
¹⁾ This is a theoretical value. In practice, the minimum period is limited to 6.7 μ s.

Update mode

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

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

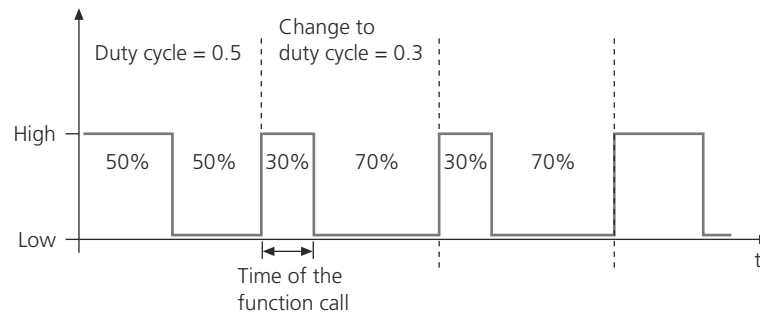
The following figure shows an example how the duty cycle will be updated in synchronous update mode.



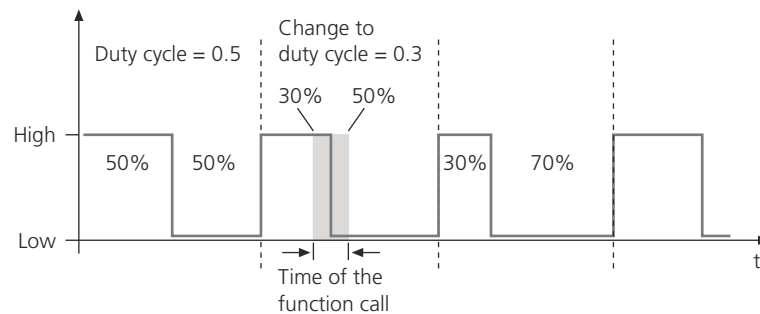
Asynchronous update mode New values for the period and/or the duty cycle are updated immediately. The update is asynchronous to the period. This can result in period and/or duty cycle values that differ from the old *and* the new values for one period.

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

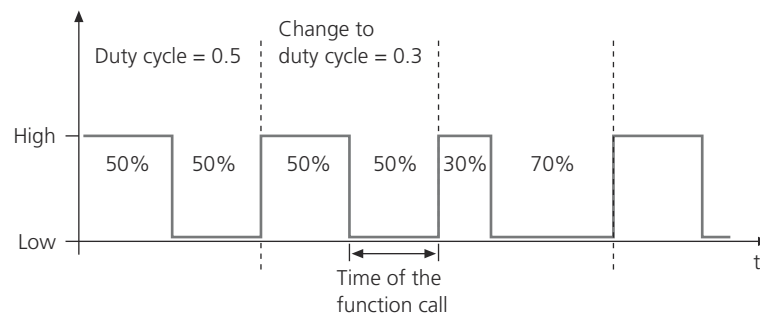
If you want to reduce the current duty cycle, the update depends also on the position within the high level. The following figures show how the duty cycle will be updated in asynchronous update mode.



If the function call for updating is executed on high level before the new duty cycle has been reached, the update takes place within the same period.



If the function call for updating is executed on high level between the new duty cycle and the old duty cycle, the update results in a duty cycle within this interval in the same period and a fully updated duty cycle in the next period.



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

Configuring the switches

The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).

I/O mapping

The following table shows the mapping of the port number and channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	L2
	2	DigP 1 ch 2	K2
	3	DigP 1 ch 3	J2
	4	DigP 1 ch 4	H2
	5	DigP 1 ch 5	G2
	6	DigP 1 ch 6	F2
	7	DigP 1 ch 7	E2
	8	DigP 1 ch 8	D2
	9	DigP 1 ch 9	L3
	10	DigP 1 ch 10	K3
	11	DigP1 ch 11	J3
	12	DigP 1 ch 12	H3
	13	DigP 1 ch 13	G3
	14	DigP 1 ch 14	F3
	15	DigP 1 ch 15	E3
	16	DigP 1 ch 16	D3
2	1	DigP 2 ch 1	L4
	2	DigP 2 ch 2	K4
	3	DigP 2 ch 3	J4
	4	DigP 2 ch 4	H4
	5	DigP 2 ch 5	G4
	6	DigP 2 ch 6	F4
	7	DigP 2 ch 7	E4
	8	DigP 2 ch 8	D4
	9	DigP 2 ch 9	L5
	10	DigP 2 ch 10	K5
	11	DigP 2 ch 11	J5
	12	DigP 2 ch 12	H5
	13	DigP 2 ch 13	G5
	14	DigP 2 ch 14	F5
	15	DigP 2 ch 15	E5
	16	DigP 2 ch 16	D5

Port	Channel	Signal	I/O Connector Pin
3	1	DigP 3 ch 1	L6
	2	DigP 3 ch 2	K6
	3	DigP 3 ch 3	J6
	4	DigP 3 ch 4	H6
	5	DigP 3 ch 5	G6
	6	DigP 3 ch 6	F6
	7	DigP 3 ch 7	E6
	8	DigP 3 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital output channel(s) by other DIO Type 3 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(5774573cf757c446bb08af21f46b2969_img.jpg\)](#))
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(a502cb21d600ba28a5cdf414d68eef89_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Generation Functionalities..... 144](#)

References

[DIO_TYPE3_PWM_BLx \(MicroAutoBox II RTI Reference !\[\]\(51514032c8ca341817228f39f1307b05_img.jpg\)](#))
[PWM Generation \(PWM\) on the DIO Type 3 \(MicroAutoBox II RTLib Reference !\[\]\(aba7c07a80262aa874bfebb3cd21d047_img.jpg\)](#))

PWM Generation (PWM) on the DIO Type 4 Unit

Purpose

To generate a PWM signal with the period and duty cycle adjustable during run time.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Characteristics

24 digital outputs are available for generating PWM signals. These outputs are shared with the other DIO Type 4 output functions. The output channels are grouped to two ports with a different number of channels.

Port	Number of Channels/Bits
1	16
2	8

The following ranges are defined for PWM signal generation:

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
1	100 ns ¹⁾	3.27 ms	50 ns
2	200 ns ¹⁾	6.55 ms	100 ns
3	400 ns ¹⁾	13.1 ms	200 ns
4	800 ns ¹⁾	26.2 ms	400 ns
5	1.6 µs ¹⁾	52.4 ms	800 ns
6	3.2 µs ¹⁾	104 ms	1.6 µs
7	6.4 µs ¹⁾	209 ms	3.2 µs
8	12.8 µs	419 ms	6.4 µs
9	25.6 µs	838 ms	12.8 µs
10	51.2 µs	1.67 s	25.6 µs
11	103 µs	3.35 s	51.2 µs
12	205 µs	6.71 s	103 µs
13	410 µs	13.4 s	205 µs
14	820 µs	26.8 s	410 µs
15	1.64 ms	53.6 s	820 µs
16	3.28 ms	107.3 s	1.64 ms

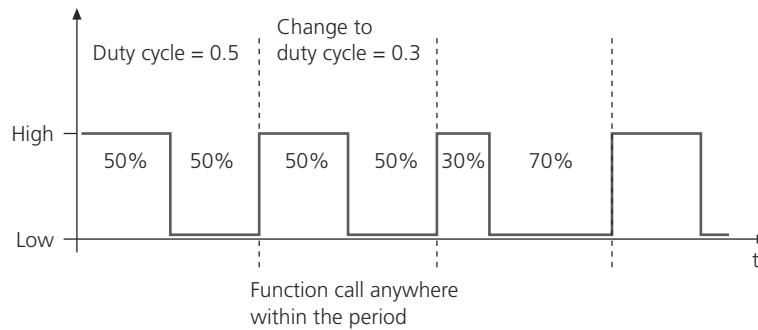
¹⁾ This is a theoretical value. In practice, the minimum period is limited to 6.7 µs.

Update mode

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

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

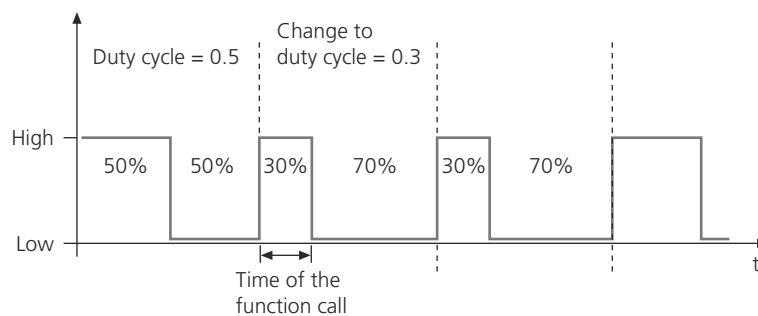
The following figure shows an example how the duty cycle will be updated in synchronous update mode.



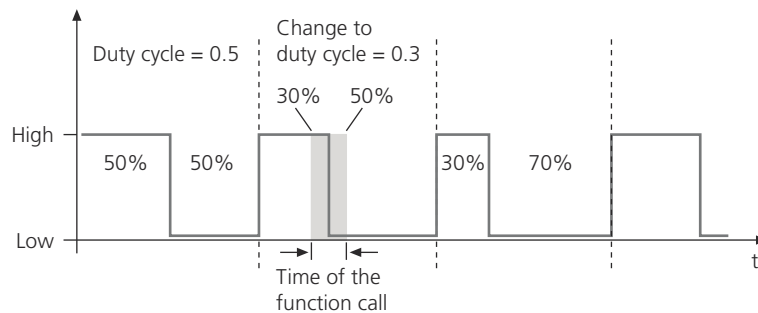
Asynchronous update mode New values for the period and/or the duty cycle are updated immediately. The update is asynchronous to the period. This can result in period and/or duty cycle values that differ from the old *and* the new values for one period.

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

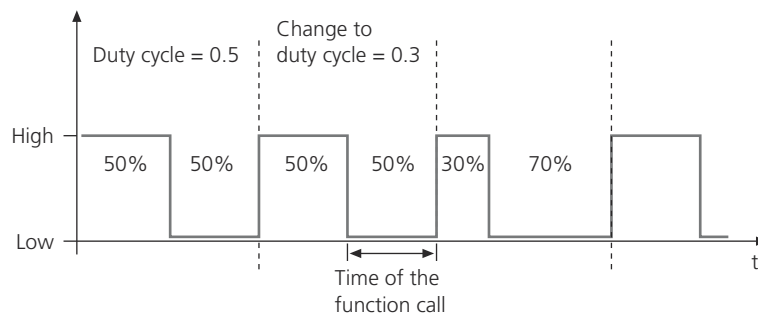
If you want to reduce the current duty cycle, the update depends also on the position within the high level. The following figures show how the duty cycle will be updated in asynchronous update mode.



If the function call for updating is executed on high level before the new duty cycle has been reached, the update takes place within the same period.



If the function call for updating is executed on high level between the new duty cycle and the old duty cycle, the update results in a duty cycle within this interval in the same period and a fully updated duty cycle in the next period.



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

Configuring the switches

The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).

I/O mapping

The following table shows the mapping of the port number and channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	G2
	2	DigP 1 ch 2	F2
	3	DigP 1 ch 3	E2
	4	DigP 1 ch 4	D2
	5	DigP 1 ch 5	C2
	6	DigP 1 ch 6	G3
	7	DigP 1 ch 7	F3
	8	DigP 1 ch 8	E3
	9	DigP 1 ch 9	D3
	10	DigP 1 ch 10	C3
	11	DigP 1 ch 11	G4
	12	DigP 1 ch 12	F4
	13	DigP 1 ch 13	E4
	14	DigP 1 ch 14	D4
	15	DigP 1 ch 15	C4
	16	DigP 1 ch 16	G5
2	1	DigP 2 ch 1	F5
	2	DigP 2 ch 2	E5
	3	DigP 2 ch 3	D5
	4	DigP 2 ch 4	C5
	5	DigP 2 ch 5	G6
	6	DigP 2 ch 6	F6
	7	DigP 2 ch 7	E6
	8	DigP 2 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital output channel(s) by other DIO Type 4 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(79de0df6c6ddd2d4eb74f1cc5f48ec50_img.jpg\)\)](#)
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(d4c9768318b38eff1042b07478e20b4c_img.jpg\)\)](#)

Related topics**Basics**

[Overview of the Signal Generation Functionalities..... 144](#)

References

[DIO_TYPE4_PWM_BLx \(MicroAutoBox II RTI Reference !\[\]\(23d9fc146e83b5c3013cfa32c784f8d5_img.jpg\)\)](#)

[PWM Generation \(PWM\) on the DIO Type 4 \(MicroAutoBox II RTLib Reference !\[\]\(c694a3ff3b077d76910920a6a1593ab4_img.jpg\)\)](#)

PWM Generation (PWM) on the DIO 1552 Type 1 Unit

Purpose

To generate a PWM signal with the period and duty cycle adjustable during run time using the DS1552 Multi-I/O Module.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
—	—	—	—	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module.

Characteristics

16 digital outputs are available for generating PWM signals. These outputs are shared with the other functions on the DIO 1552 Type 1 unit.

The following ranges are defined for PWM signal generation:

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
1	100 ns ¹⁾	3.27 ms	50 ns
2	200 ns ¹⁾	6.55 ms	100 ns
3	400 ns ¹⁾	13.1 ms	200 ns
4	800 ns ¹⁾	26.2 ms	400 ns
5	1.6 µs ¹⁾	52.4 ms	800 ns
6	3.2 µs ¹⁾	104 ms	1.6 µs

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
7	6.4 μs ¹⁾	209 ms	3.2 μs
8	12.8 μs	419 ms	6.4 μs
9	25.6 μs	838 ms	12.8 μs
10	51.2 μs	1.67 s	25.6 μs
11	103 μs	3.35 s	51.2 μs
12	205 μs	6.71 s	103 μs
13	410 μs	13.4 s	205 μs
14	820 μs	26.8 s	410 μs
15	1.64 ms	53.6 s	820 μs
16	3.28 ms	107.3 s	1.64 ms

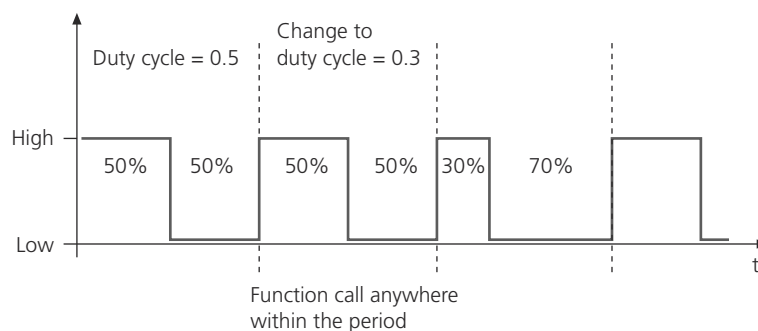
¹⁾ This is a theoretical value. In practice, the minimum period is limited to 6.7 μs .

Update mode

A PWM signal is specified by its period and duty cycle. To change the signal during run time, you can specify new values for the period and the duty cycle. The new values are updated according to the specified update mode.

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

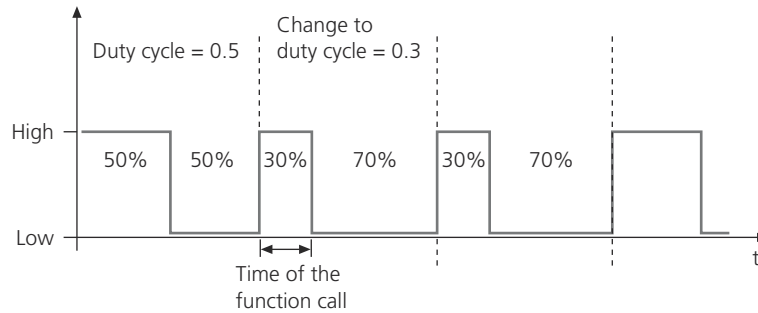
The following figure shows an example how the duty cycle will be updated in synchronous update mode.



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

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

If you want to reduce the current duty cycle, the update depends also on the position within the high level. The following figure shows an example how the duty cycle will be updated in asynchronous update mode.



Configuring the switches

The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).

I/O mapping

The following table shows the mapping of the channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector (DS1514 ZIF connector).

Channel	Signal	I/O Connector Pin
1	DigOut ch 1	F5
2	DigOut ch 2	E5
3	DigOut ch 3	E6
4	DigOut ch 4	D2
5	DigOut ch 5	D3
6	DigOut ch 6	D4
7	DigOut ch 7	D5
8	DigOut ch 8	D6
9	DigOut ch 9	C2
10	DigOut ch 10	C3
11	DigOut ch 11	C5
12	DigOut ch 12	B2
13	DigOut ch 13	B5
14	DigOut ch 14	B6

Channel	Signal	I/O Connector Pin
15	DigOut ch 15	A5
16	DigOut ch 16	A6

Note

Concurrent access to the same digital output channel by other DIO 1552 Type 1 blocks or functions is not allowed.



For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(e662c6fdc679f154c0e75d901761d894_img.jpg\)\)](#)

Related topics**Basics**

[Overview of the Signal Generation Functionalities..... 144](#)

References

DIO1552_TP1_PWM_BLx (MicroAutoBox II RTI Reference )
 PWM Generation (PWM) on the DIO 1552 Type 1 Unit of the DS1552 Multi-I/O Module (MicroAutoBox II RTLib Reference )

Square-Wave Signal Generation (FREQ) on the DIO Type 3 Unit

Purpose

To generate square-wave signals with variable frequencies and a fixed duty cycle of 50%.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	✓	–	–

Characteristics

40 digital outputs are available for generating square-wave signals. These outputs are shared with the other DIO Type 3 output functions. The output channels are grouped to three ports with a different number of channels.

Port	Number of Channels/Bits
1	16
2	16
3	8

The following ranges are defined for square-wave signal generation:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	150 kHz	50 ns
2	4.77 Hz	150 kHz	100 ns
3	2.39 Hz	150 kHz	200 ns
4	1.20 Hz	150 kHz	400 ns
5	0.60 Hz	150 kHz	800 ns
6	0.30 Hz	150 kHz	1.6 μ s
7	0.15 Hz	150 kHz	3.2 μ s
8	75 mHz	78.12 kHz	6.4 μ s
9	38 mHz	39.06 kHz	12.8 μ s
10	19 mHz	19.53 kHz	25.6 μ s
11	10 mHz	9.76 kHz	51.2 μ s
12	5.0 mHz	4.88 kHz	103 μ s
13	2.5 mHz	2.44 kHz	205 μ s
14	1.2 mHz	1.22 kHz	410 μ s
15	0.6 mHz	610.35 Hz	820 μ s
16	0.3 mHz	305.17 Hz	1.64 ms

Update mode

A frequency value changed during run time, is updated at the next rising edge of the output signal (synchronous update mode).

Configuring the switches

The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).

I/O mapping

The following table shows the mapping of the port number and channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	L2
	2	DigP 1 ch 2	K2
	3	DigP 1 ch 3	J2
	4	DigP 1 ch 4	H2
	5	DigP 1 ch 5	G2
	6	DigP 1 ch 6	F2
	7	DigP 1 ch 7	E2
	8	DigP 1 ch 8	D2
	9	DigP 1 ch 9	L3
	10	DigP 1 ch 10	K3
	11	DigP1 ch 11	J3
	12	DigP 1 ch 12	H3
	13	DigP 1 ch 13	G3
	14	DigP 1 ch 14	F3
	15	DigP 1 ch 15	E3
	16	DigP 1 ch 16	D3
2	1	DigP 2 ch 1	L4
	2	DigP 2 ch 2	K4
	3	DigP 2 ch 3	J4
	4	DigP 2 ch 4	H4
	5	DigP 2 ch 5	G4
	6	DigP 2 ch 6	F4
	7	DigP 2 ch 7	E4
	8	DigP 2 ch 8	D4
	9	DigP 2 ch 9	L5
	10	DigP 2 ch 10	K5
	11	DigP 2 ch 11	J5
	12	DigP 2 ch 12	H5
	13	DigP 2 ch 13	G5
	14	DigP 2 ch 14	F5
	15	DigP 2 ch 15	E5
	16	DigP 2 ch 16	D5

Port	Channel	Signal	I/O Connector Pin
3	1	DigP 3 ch 1	L6
	2	DigP 3 ch 2	K6
	3	DigP 3 ch 3	J6
	4	DigP 3 ch 4	H6
	5	DigP 3 ch 5	G6
	6	DigP 3 ch 6	F6
	7	DigP 3 ch 7	E6
	8	DigP 3 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital output channel(s) by other DIO Type 3 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(4e333a6106fc298d0ae6dff272a736ef_img.jpg\)](#))
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(97089f8e07e24e31baa67366e358a709_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Generation Functionalities..... 144](#)

References

[DIO_TYPE3_FREQ_BLx \(MicroAutoBox II RTI Reference !\[\]\(6a9b39b98eb945faa14c645ec99e4eaa_img.jpg\)](#))
[PWM Generation \(PWM\) on the DIO Type 3 \(MicroAutoBox II RTLib Reference !\[\]\(182077db5bac9ff62bf376fe37ffa951_img.jpg\)](#))

Square-Wave Signal Generation (FREQ) on the DIO Type 4 Unit

Purpose

To generate square-wave signals with variable frequencies and a fixed duty cycle of 50%.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Characteristics

24 digital outputs are available for generating square-wave signals. These outputs are shared with the other DIO Type 4 output functions. The output channels are grouped to two ports with a different number of channels.

Port	Number of Channels/Bits
1	16
2	8

The following ranges are defined for square-wave signal generation:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	150 kHz	50 ns
2	4.77 Hz	150 kHz	100 ns
3	2.39 Hz	150 kHz	200 ns
4	1.20 Hz	150 kHz	400 ns
5	0.60 Hz	150 kHz	800 ns
6	0.30 Hz	150 kHz	1.6 μ s
7	0.15 Hz	150 kHz	3.2 μ s
8	75 mHz	78.12 kHz	6.4 μ s
9	38 mHz	39.06 kHz	12.8 μ s
10	19 mHz	19.53 kHz	25.6 μ s
11	10 mHz	9.76 kHz	51.2 μ s
12	5.0 mHz	4.88 kHz	103 μ s
13	2.5 mHz	2.44 kHz	205 μ s
14	1.2 mHz	1.22 kHz	410 μ s
15	0.6 mHz	610.35 Hz	820 μ s
16	0.3 mHz	305.17 Hz	1.64 ms

Update mode

A frequency value changed during run time, is updated at the next rising edge of the output signal (synchronous update mode).

Configuring the switches

The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).

I/O mapping

The following table shows the mapping of the port number and channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	G2
	2	DigP 1 ch 2	F2
	3	DigP 1 ch 3	E2
	4	DigP 1 ch 4	D2
	5	DigP 1 ch 5	C2
	6	DigP 1 ch 6	G3
	7	DigP 1 ch 7	F3
	8	DigP 1 ch 8	E3
	9	DigP 1 ch 9	D3
	10	DigP 1 ch 10	C3
	11	DigP 1 ch 11	G4
	12	DigP 1 ch 12	F4
	13	DigP 1 ch 13	E4
	14	DigP 1 ch 14	D4
	15	DigP 1 ch 15	C4
	16	DigP 1 ch 16	G5

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	F5
	2	DigP 2 ch 2	E5
	3	DigP 2 ch 3	D5
	4	DigP 2 ch 4	C5
	5	DigP 2 ch 5	G6
	6	DigP 2 ch 6	F6
	7	DigP 2 ch 7	E6
	8	DigP 2 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital output channel(s) by other DIO Type 4 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(9a53fe79a03d38d8322f7a2c5a875b36_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(01f19d40f03100aa8a158c4891453b0d_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Generation Functionalities..... 144](#)

References

[DIO_TYPE4_FREQ_BLx \(MicroAutoBox II RTI Reference !\[\]\(47734e4656765d20df4fdbd5b7aff048_img.jpg\)](#))
[PWM Generation \(PWM\) on the DIO Type 4 \(MicroAutoBox II RTLib Reference !\[\]\(effba44ea72cb8c77bdc1dac75561f86_img.jpg\)](#))

Square-Wave Signal Generation (FREQ) on the DIO 1552 Type 1 Unit

Purpose

To generate square-wave signals with variable frequencies and a fixed duty cycle of 50% using the DS1552 Multi-I/O Module.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	–	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module.**Characteristics**

16 digital outputs are available for generating square-wave signals. These outputs are shared with the other functions of the DIO 1552 Type 1 unit.

The following ranges are defined for square-wave signal generation:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	150 kHz	50 ns
2	4.77 Hz	150 kHz	100 ns
3	2.39 Hz	150 kHz	200 ns
4	1.20 Hz	150 kHz	400 ns
5	0.60 Hz	150 kHz	800 ns
6	0.30 Hz	150 kHz	1.6 μ s
7	0.15 Hz	150 kHz	3.2 μ s
8	75 mHz	78.12 kHz	6.4 μ s
9	38 mHz	39.06 kHz	12.8 μ s
10	19 mHz	19.53 kHz	25.6 μ s
11	10 mHz	9.76 kHz	51.2 μ s
12	5.0 mHz	4.88 kHz	103 μ s
13	2.5 mHz	2.44 kHz	205 μ s
14	1.2 mHz	1.22 kHz	410 μ s
15	0.6 mHz	610.35 Hz	820 μ s
16	0.3 mHz	305.17 Hz	1.64 ms

Update mode

A frequency value changed during run time, is updated at the next rising edge of the output signal (synchronous update mode).

Configuring the switches

The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).

I/O mapping

The following table shows the mapping of the channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector (DS1514 ZIF connector).

Channel	Signal	I/O Connector Pin
1	DigOut ch 1	F5
2	DigOut ch 2	E5
3	DigOut ch 3	E6
4	DigOut ch 4	D2
5	DigOut ch 5	D3
6	DigOut ch 6	D4
7	DigOut ch 7	D5
8	DigOut ch 8	D6
9	DigOut ch 9	C2
10	DigOut ch 10	C3
11	DigOut ch 11	C5
12	DigOut ch 12	B2
13	DigOut ch 13	B5
14	DigOut ch 14	B6
15	DigOut ch 15	A5
16	DigOut ch 16	A6

Note

Concurrent access to the same digital output channel by other DIO 1552 Type 1 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(dce81645e0100714e86d66fe4d06ecba_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Generation Functionalities.....](#) 144

References

[DIO1552_TP1_FREQ_BLx \(MicroAutoBox II RTI Reference !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)\)](#)
[Square-Wave Signal Generation \(FREQ\) on the DIO 1552 Type 1 Unit of the DS1552](#)
[Multi-I/O Module \(MicroAutoBox II RTLib Reference !\[\]\(e658400d40ca763c7cf4c8c420885c6a_img.jpg\)\)](#)

Multi-Channel PWM Signal Generation (MC_PWM)

Introduction

MicroAutoBox II provides the timing I/O feature to generate pulse-width modulated signals on multi channels (MC_PWM).

Where to go from here

Information in this section

Basics on Multi-Channel PWM Signal Generation on the DIO Type 3.....	169
Basics on Multi-Channel PWM Signal Generation on the DIO Type 4.....	182

Basics on Multi-Channel PWM Signal Generation on the DIO Type 3

Purpose

To generate different kinds of multi-channel PWM signals.

Note

To use the multi-channel PWM signal generation on the DIO Type 3, firmware version 1.3 or higher is required.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards				
	DS1507	DS1511	DS1513	DS1514	
—	—	✓	—	—	

Basics on multi-channel PWM signal generation on the DIO Type 3

The multi-channel PWM signal generation on the DIO Type 3 can be flexibly configured for any number of channels within the limits of available output channels. You can use it to generate from 1-phase PWM signals up to 16-phase PWM signals according to your needs.

Characteristics

The DIO Type 3 unit provides 40 digital output channels. These channels are grouped to three ports with 16, 16 and 8 channels. Besides the number of channels, each port can be identically configured. The assignment of I/O channels can only be done within one port. This guarantees data consistency.

In the initialization phase of the multi-channel PWM signal generation, the following settings can be specified:

- Number of channels used for the non-inverted PWM signal generation including the start channel within one port
- Settings of the low-side and high-side switches
- Period range for the signals to be generated and the related resolutions (refer to [Period ranges and resolutions](#) on page 178)
- Alignment of the channels (refer to [Alignment mode](#) on page 170)
- Update mode (refer to [Update mode](#) on page 173)
- Generation of inverted signals (refer to [Generation of inverted signals](#) on page 172)
- Generation of interrupts (refer to [Generation and configuration of interrupts and triggers](#) on page 176)
- Generation of external trigger signals (refer to [Generation and configuration of interrupts and triggers](#) on page 176)
- Rate and time delay for interrupts and triggers (refer to [Generation and configuration of interrupts and triggers](#) on page 176)
- Dead time (refer to [Specifying dead time](#) on page 177)

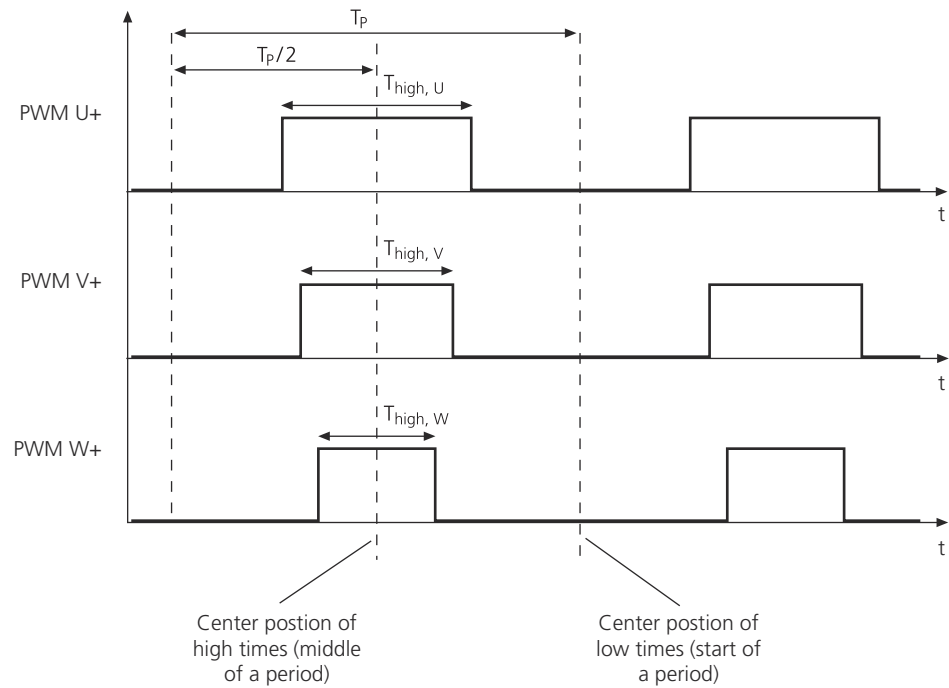
During run time, you can change the following parameters:

- Period
- Duty cycles

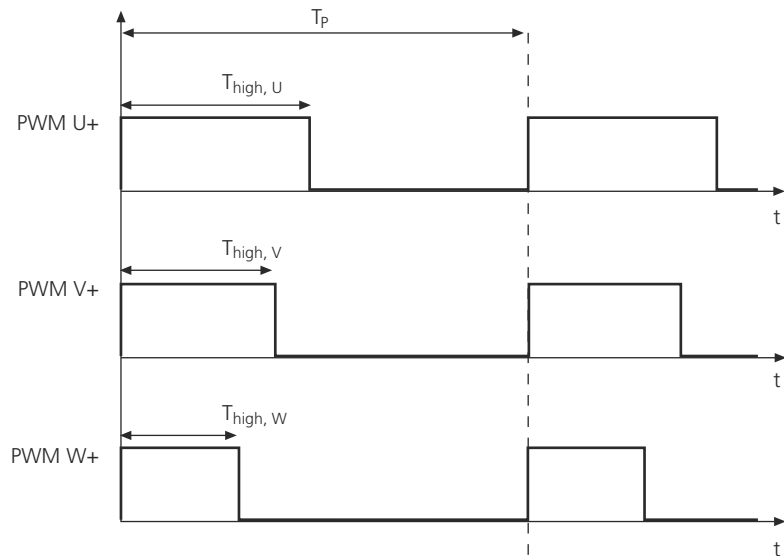
Alignment mode

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

Signal characteristics of a center-aligned PWM signal The following example shows a center-aligned 3-phase PWM signal. The signals are aligned to the middle of their periods, which is also the middle of their high pulses.



Signal characteristics of an edge-aligned PWM signal The following example shows an edge-aligned 3-phase PWM signal. The signals are aligned to their rising edges.



Settings that depend on the alignment mode Depending on the specified alignment mode, not all the configurations are available for some settings. The following table gives you an overview.

Alignment	Update Mode	Dead Time
Edge-aligned	<ul style="list-style-type: none"> At the start of a period 	Not available

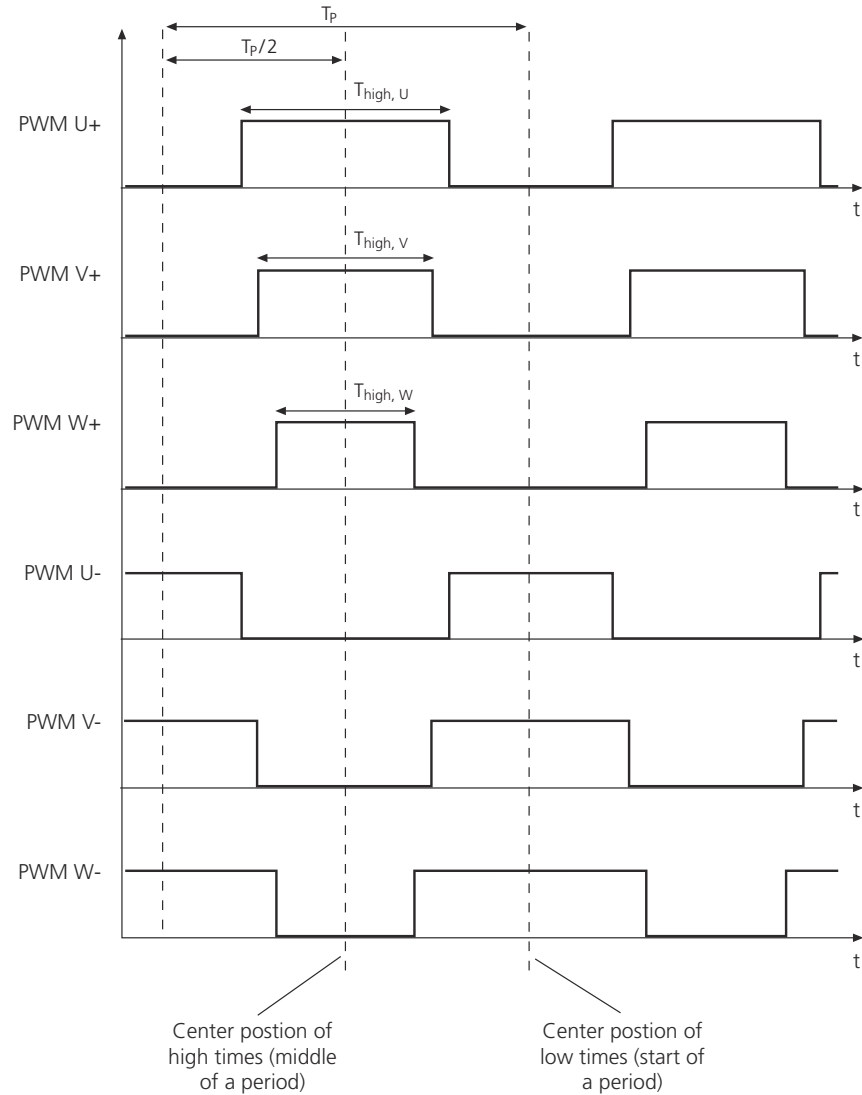
Alignment	Update Mode	Dead Time
Center-aligned	<ul style="list-style-type: none">▪ At the start of a period and/or▪ at the middle of a period	Available

Generation of inverted signals

You can specify the generation of inverted signals for center-aligned PWM signal generation. The number of required channels is then automatically doubled. The number of required channels must not exceed the number of available channels.

Signal characteristics of a center-aligned PWM signal with inverted signals

The following example shows a center-aligned 3-phase PWM signal generation with inverted signals.

**Update mode**

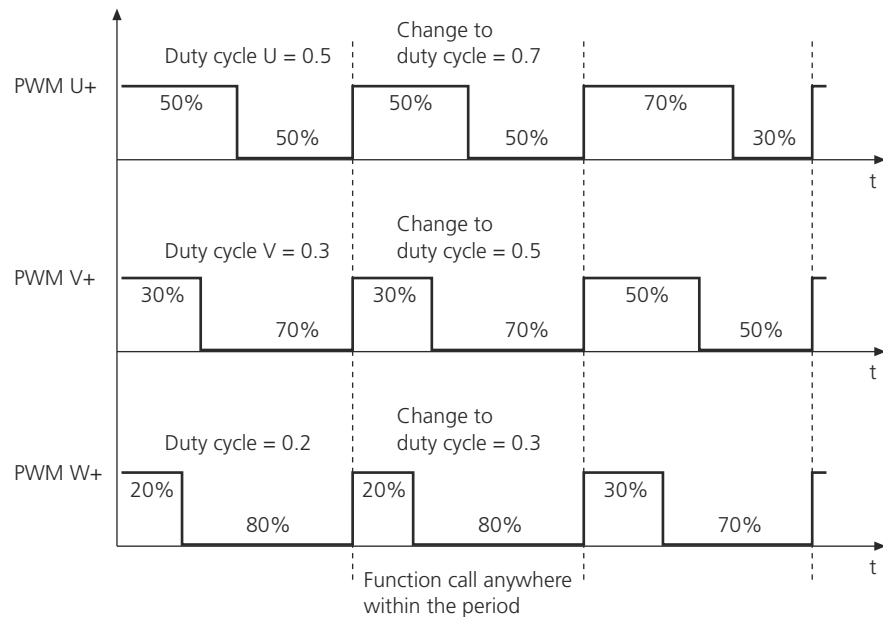
During run time, you can specify new values for the period and the duty cycles. With the update mode, you can specify the timing behavior of the update.

The duty cycles have to be specified for the non-inverted channels. If you have enabled the generation of inverted signals, their duty cycles are automatically adjusted.

Synchronous update for edge-aligned multi-channel PWM signal

generation New values for the period and/or the duty cycles are updated at the next rising edge of the PWM output signal.

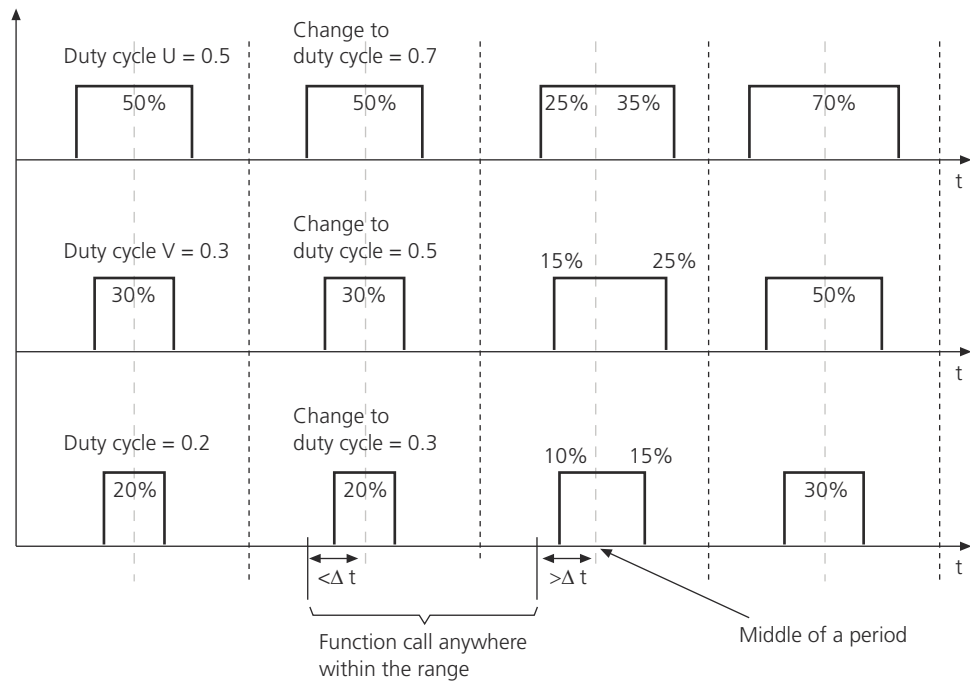
The following figure shows an example how the duty cycles of a 3-phase PWM will be updated in synchronous update mode.

**Update at the middle of the high pulses for center-aligned multi-channel PWM signal generation**

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

The following figure shows an example how the duty cycles of a 3-phase PWM will be updated in the *CENTER* update mode.

The update function must be called before the next middle of the high pulses (Δt) for considering the new values at the second half of the current high pulses.

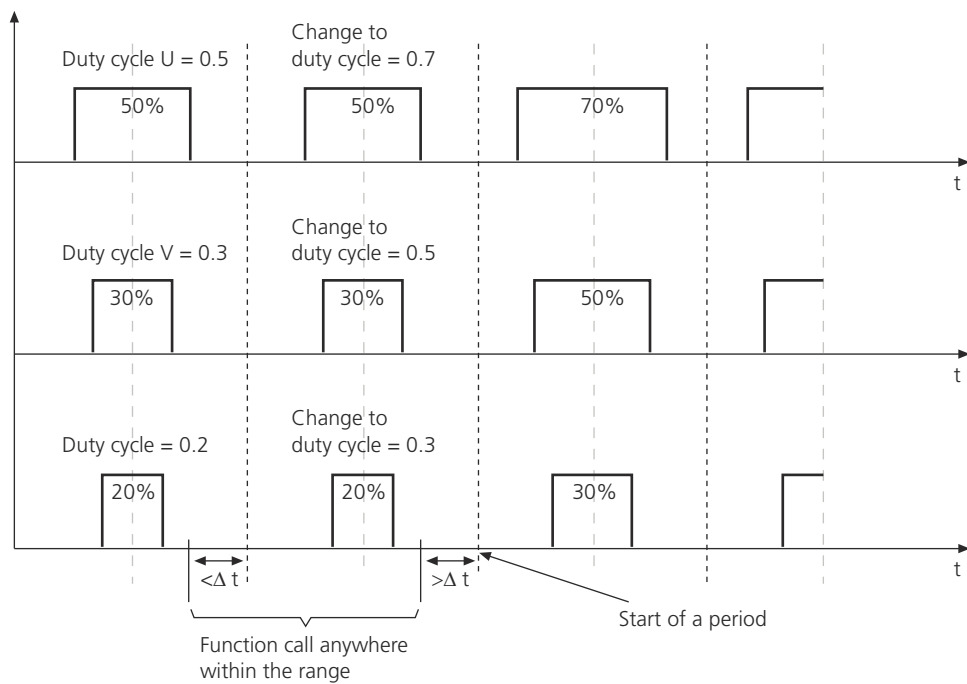


Update at the start of a period for center-aligned multi-channel PWM signal generation

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

The following figure shows an example how the duty cycles of a 3-phase PWM will be updated in the *START* update mode.

The update function must be called before the middle of the low pulses (Δt ; start of the period) for considering the new values at the following period.



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

Generation and configuration of interrupts and triggers

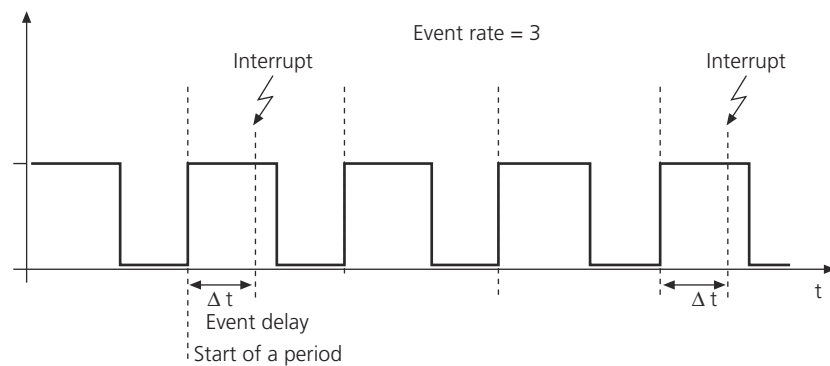
You can specify to generate interrupts and external trigger pulses during multi-channel PWM signal generation.

An interrupt can be used to trigger an interrupt service routine, an external trigger pulse can be used for external triggering. For example, you can use an external trigger pulse to start an A/D conversion. When you enable the trigger pulse generation an additional output channel is automatically allocated.

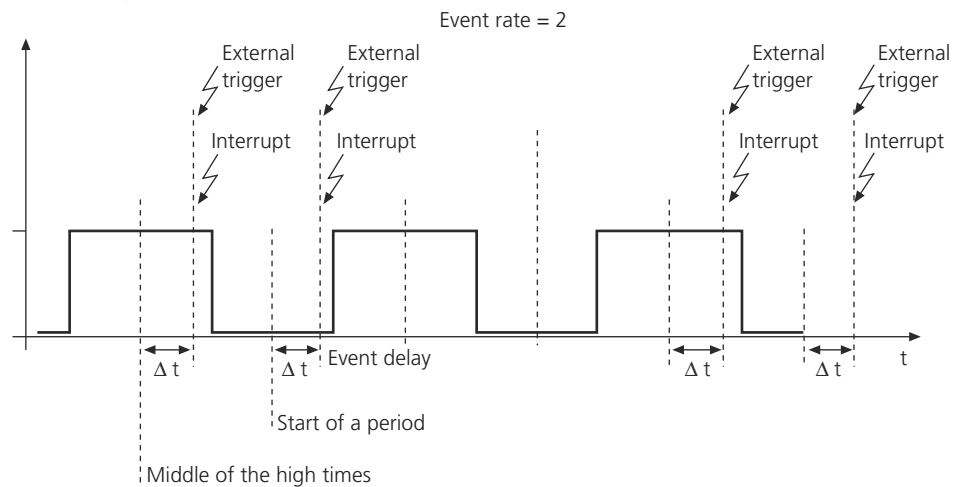
Interrupts and external trigger pulses can be generated at the start and/or in the middle of a period.

The *generation* of interrupts and external trigger pulses can be specified separately. Either you enable the interrupt generation or the trigger pulse generation, or both. However, the *configuration* of interrupts and external trigger pulses is commonly. Interrupts and external trigger pulses are handled as *events* of the multi-channel PWM signal generation. For these events, you can specify an event rate in the range 1 ... 255 and a delay.

The following example shows an edge-aligned multi-channel PWM signal generation with an event rate of 3 and an event delay Δt . Each 3rd rising edge (start of a period) is used for generating an event (interrupt and/or external trigger pulse).

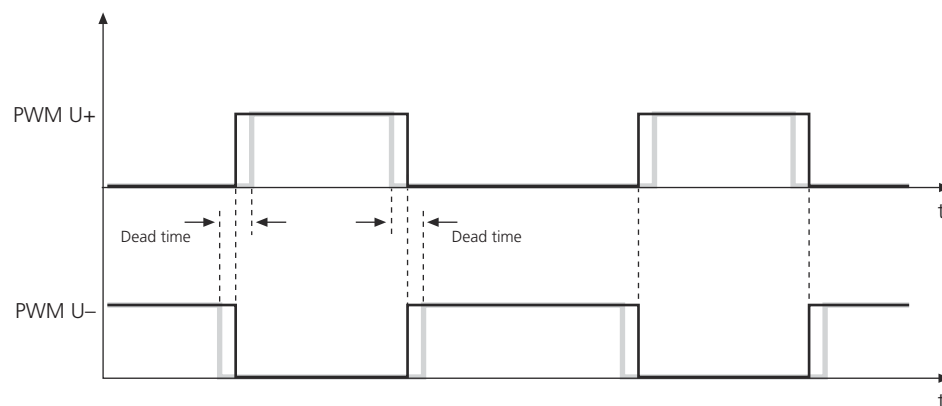


The following example shows a center-aligned multi-channel PWM signal generation with an event rate of 2 and an event delay Δt . An interrupt and an external trigger pulse are generated at each second middle of the high times (middle of the period) and at each second middle of the low times (start of a period).



Specifying dead time

With the dead time parameter, you can define the timing relationship between the PWM output signals and their corresponding inverted output signals for center-aligned PWM signal generation. The specified dead time is valid for all the channels of the current multi-channel PWM group. Dead times are used, for example, to avoid ripple currents or prevent shoot-through currents on the phase drivers. The high times of the inverted and non-inverted output signals are reduced symmetrically. The following illustration shows center-aligned multi-channel PWM with a specified dead time.



The dead time can be specified in the range 0 ... 12.8 μ s. If the high time of the inverted or non-inverted signal is less than the specified dead time, the signal remains in low state.

Configuring the switches

The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).

Period ranges and resolutions

The period range is divided into 16 ranges. For each range, there is a specific resolution.

Due to quantization effects, you might encounter considerable deviations between the desired PWM period and the generated PWM period, especially for higher PWM frequencies. To avoid poor frequency resolution, you should therefore select the period range with the best possible resolution (resolution values as small as possible).

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
1	200 ns ¹⁾	3.27 ms	50 ns
2	400 ns ¹⁾	6.55 ms	100 ns
3	800 ns ¹⁾	13.1 ms	200 ns
4	1.6 μ s ¹⁾	26.2 ms	400 ns
5	3.2 μ s ¹⁾	52.4 ms	800 ns
6	6.4 μ s ¹⁾	104 ms	1.6 μ s
7	12.8 μ s	209 ms	3.2 μ s
8	25.6 μ s	419 ms	6.4 μ s

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
9	51.2 μ s	838 ms	12.8 μ s
10	103 μ s	1.67 s	25.6 μ s
11	205 μ s	3.35 s	51.2 μ s
12	410 μ s	6.71 s	103 μ s
13	820 μ s	13.4 s	205 μ s
14	1.64 ms	26.8 s	410 μ s
15	3.28 ms	53.6 s	820 μ s
16	6.56 ms	107.3 s	1.64 ms

¹⁾ This is a theoretical value. In practice, the minimum period is limited to 6.7 μ s.

RTI/RTLib support

The multi-channel PWM signal generation is supported by RTI and RTLib. For detailed information, refer to:

- [DIO_TYPE3_MC_PWM_BLx \(MicroAutoBox II RTI Reference !\[\]\(511a36c244659513b679df9c639945de_img.jpg\)\)](#)
- [Multi-Channel PWM Generation \(MC_PWM\) on the DIO Type 3 \(MicroAutoBox II RTLib Reference !\[\]\(2c0783baf87a2728b2fe49eb1c34c456_img.jpg\)\)](#)

I/O mapping

The following table shows the order of the generated signals.

Without Inverted Signals	With Inverted Signals
Non-inverted signal 1	Non-inverted signal 1
...	...
Non-inverted signal n	Non-inverted signal n
External trigger signal (optional)	Inverted signal 1
	...
	Inverted signal n
	External trigger signal (optional)

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

The following table shows the mapping of the port number and channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	L2
	2	DigP 1 ch 2	K2
	3	DigP 1 ch 3	J2
	4	DigP 1 ch 4	H2
	5	DigP 1 ch 5	G2
	6	DigP 1 ch 6	F2
	7	DigP 1 ch 7	E2
	8	DigP 1 ch 8	D2
	9	DigP 1 ch 9	L3
	10	DigP 1 ch 10	K3
	11	DigP1 ch 11	J3
	12	DigP 1 ch 12	H3
	13	DigP 1 ch 13	G3
	14	DigP 1 ch 14	F3
	15	DigP 1 ch 15	E3
	16	DigP 1 ch 16	D3
2	1	DigP 2 ch 1	L4
	2	DigP 2 ch 2	K4
	3	DigP 2 ch 3	J4
	4	DigP 2 ch 4	H4
	5	DigP 2 ch 5	G4
	6	DigP 2 ch 6	F4
	7	DigP 2 ch 7	E4
	8	DigP 2 ch 8	D4
	9	DigP 2 ch 9	L5
	10	DigP 2 ch 10	K5
	11	DigP 2 ch 11	J5
	12	DigP 2 ch 12	H5
	13	DigP 2 ch 13	G5
	14	DigP 2 ch 14	F5
	15	DigP 2 ch 15	E5
	16	DigP 2 ch 16	D5

Port	Channel	Signal	I/O Connector Pin
3	1	DigP 3 ch 1	L6
	2	DigP 3 ch 2	K6
	3	DigP 3 ch 3	J6
	4	DigP 3 ch 4	H6
	5	DigP 3 ch 5	G6
	6	DigP 3 ch 6	F6
	7	DigP 3 ch 7	E6
	8	DigP 3 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital output channel(s) by other DIO Type 3 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(9a53fe79a03d38d8322f7a2c5a875b36_img.jpg\)](#))
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(01f19d40f03100aa8a158c4891453b0d_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Generation Functionalities..... 144](#)

References

[DIO_TYPE3_MC_PWM_BLx \(MicroAutoBox II RTI Reference !\[\]\(47734e4656765d20df4fdbd5b7aff048_img.jpg\)](#))
[Multi-Channel PWM Generation \(MC_PWM\) on the DIO Type 3 \(MicroAutoBox II RTLib Reference !\[\]\(effba44ea72cb8c77bdc1dac75561f86_img.jpg\)](#))

Basics on Multi-Channel PWM Signal Generation on the DIO Type 4

Purpose To generate different kinds of multi-channel PWM signals.

Hardware requirements Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Basics on multi-channel PWM signal generation on the DIO Type 4

The multi-channel PWM signal generation on the DIO Type 4 can be flexibly configured for any number of channels within the limits of available output channels. You can use it to generate from 1-phase PWM signals up to 16-phase PWM signals according to your needs.

Characteristics

The DIO Type 4 unit provides 24 digital output channels. These channels are grouped to two ports with 16 and 8 channels. Besides the number of channels, each port can be identically configured. The assignment of I/O channels can only be done within one port. This guarantees data consistency.

In the initialization phase of the multi-channel PWM signal generation, the following settings can be specified:

- Number of channels used for the non-inverted PWM signal generation including the start channel within one port
- Settings of the low-side and high-side switches
- Period range for the signals to be generated and the related resolutions (refer to [Period ranges and resolutions](#) on page 191)
- Alignment of the channels (refer to [Alignment mode](#) on page 183)
- Update mode (refer to [Update mode](#) on page 186)
- Generation of inverted signals (refer to [Generation of inverted signals](#) on page 185)
- Generation of interrupts (refer to [Generation and configuration of interrupts and triggers](#) on page 189)
- Generation of external trigger signals (refer to [Generation and configuration of interrupts and triggers](#) on page 189)
- Rate and time delay for interrupts and triggers (refer to [Generation and configuration of interrupts and triggers](#) on page 189)
- Dead time (refer to [Specifying dead time](#) on page 190)

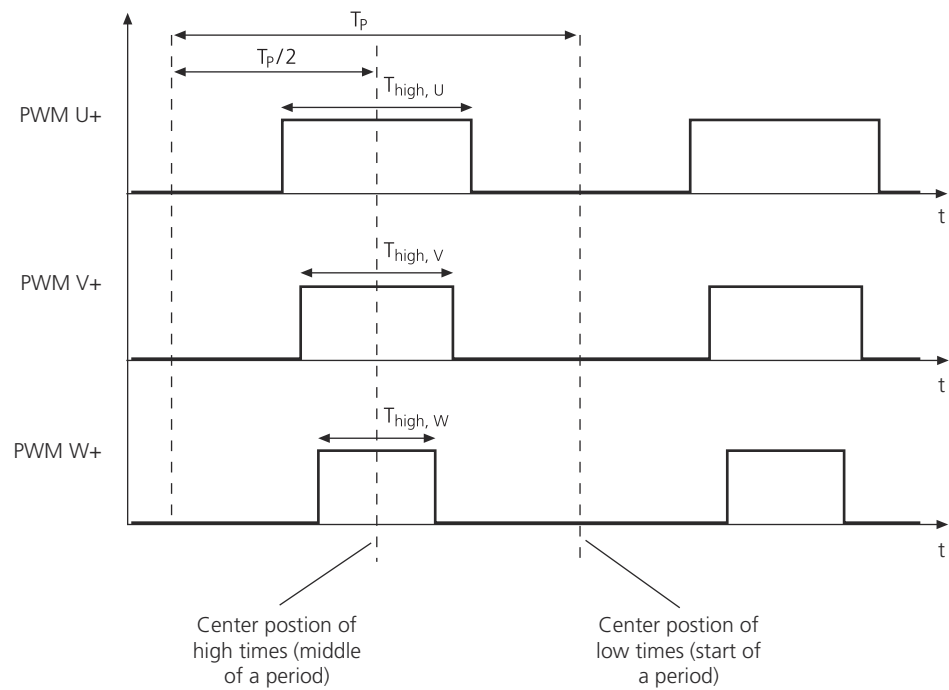
During run time, you can change the following parameters:

- Period
- Duty cycles

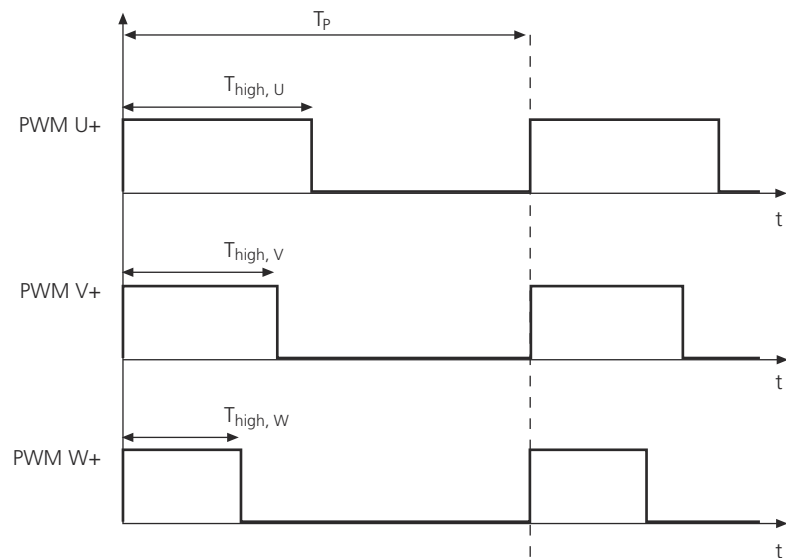
Alignment mode

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

Signal characteristics of a center-aligned PWM signal The following example shows a center-aligned 3-phase PWM signal. The signals are aligned to the middle of their periods, which is also the middle of their high pulses.



Signal characteristics of an edge-aligned PWM signal The following example shows an edge-aligned 3-phase PWM signal. The signals are aligned to their rising edges.



Settings that depend on the alignment mode Depending on the specified alignment mode, not all the configurations are available for some settings. The following table gives you an overview.

Alignment	Update Mode	Dead Time
Edge-aligned	<ul style="list-style-type: none"> At the start of a period 	Not available

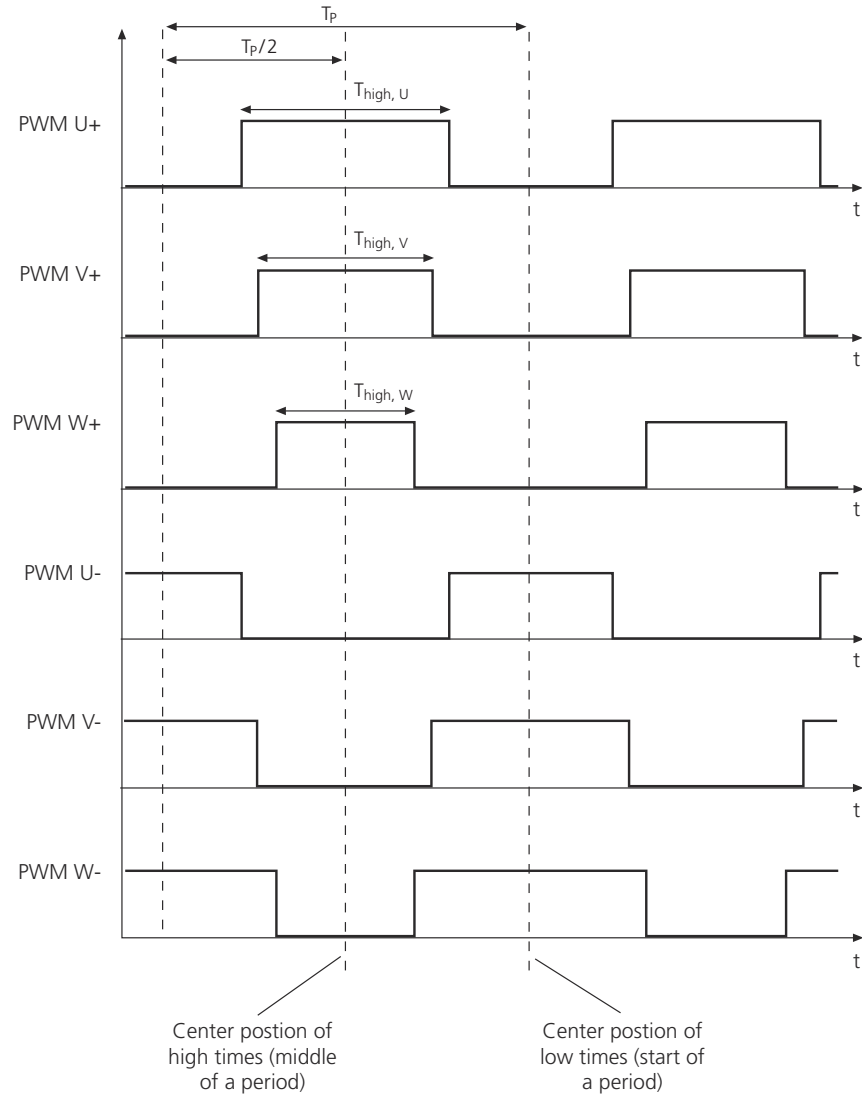
Alignment	Update Mode	Dead Time
Center-aligned	<ul style="list-style-type: none">▪ At the start of a period and/or▪ at the middle of a period	Available

Generation of inverted signals

You can specify the generation of inverted signals for center-aligned PWM signal generation. The number of required channels is then automatically doubled. The number of required channels must not exceed the number of available channels.

Signal characteristics of a center-aligned PWM signal with inverted signals

The following example shows a center-aligned 3-phase PWM signal generation with inverted signals.



Update mode

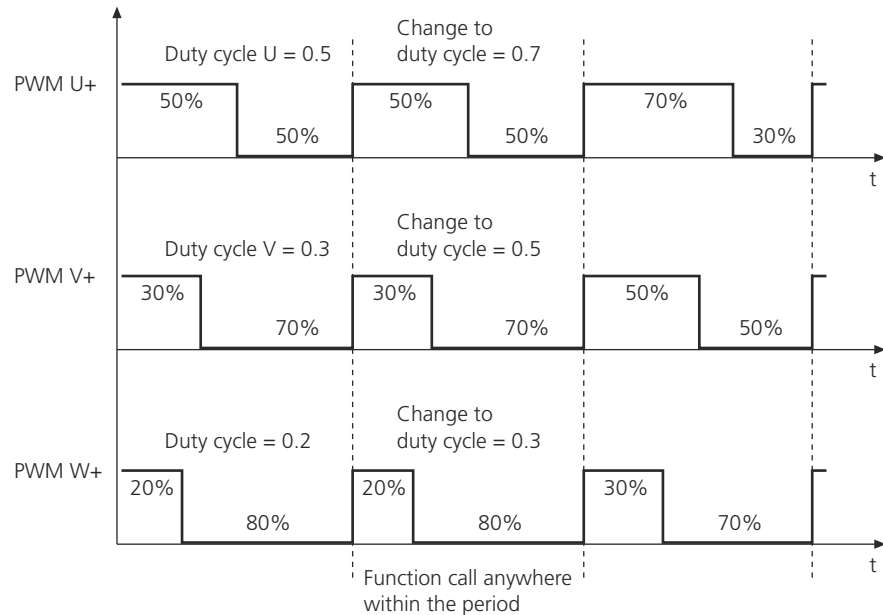
During run time, you can specify new values for the period and the duty cycles. With the update mode, you can specify the timing behavior of the update.

The duty cycles have to be specified for the non-inverted channels. If you have enabled the generation of inverted signals, their duty cycles are automatically adjusted.

Synchronous update for edge-aligned multi-channel PWM signal generation

generation New values for the period and/or the duty cycles are updated at the next rising edge of the PWM output signal.

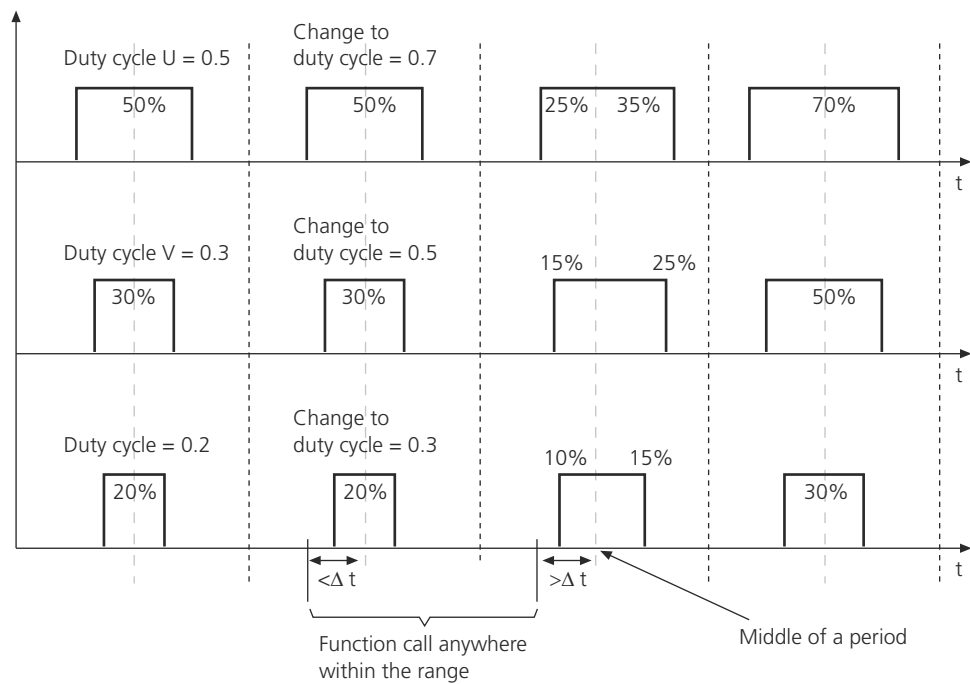
The following figure shows an example how the duty cycles of a 3-phase PWM will be updated in synchronous update mode.

**Update at the middle of the high pulses for center-aligned multi-channel PWM signal generation**

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

The following figure shows an example how the duty cycles of a 3-phase PWM will be updated in the *CENTER* update mode.

The update function must be called before the next middle of the high pulses (Δt) for considering the new values at the second half of the current high pulses.

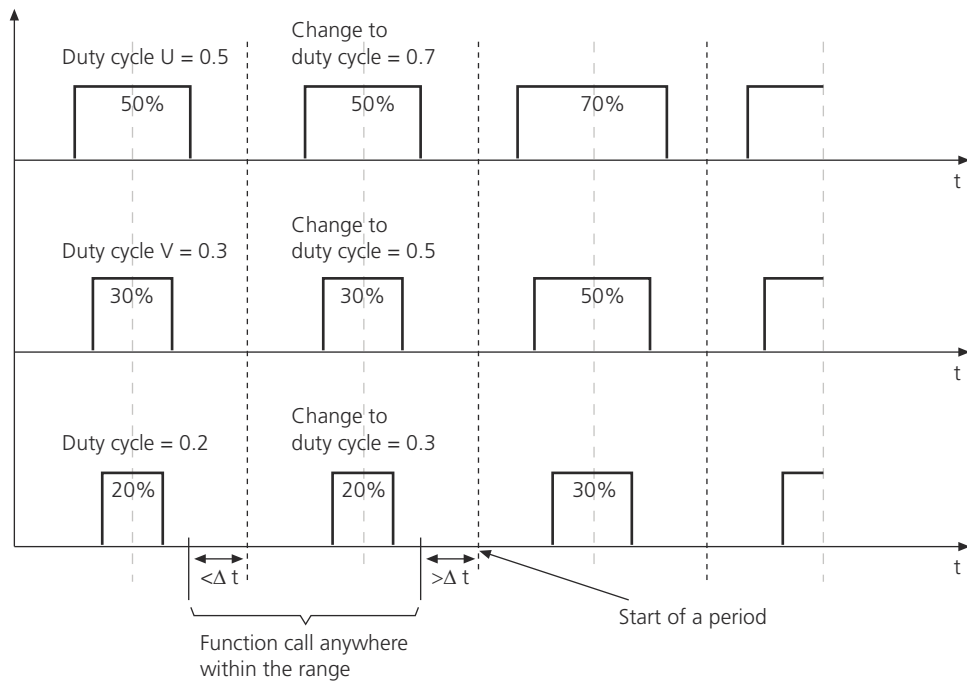


Update at the start of a period for center-aligned multi-channel PWM signal generation

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

The following figure shows an example how the duty cycles of a 3-phase PWM will be updated in the *START* update mode.

The update function must be called before the middle of the low pulses (Δt ; start of the period) for considering the new values at the following period.



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

Generation and configuration of interrupts and triggers

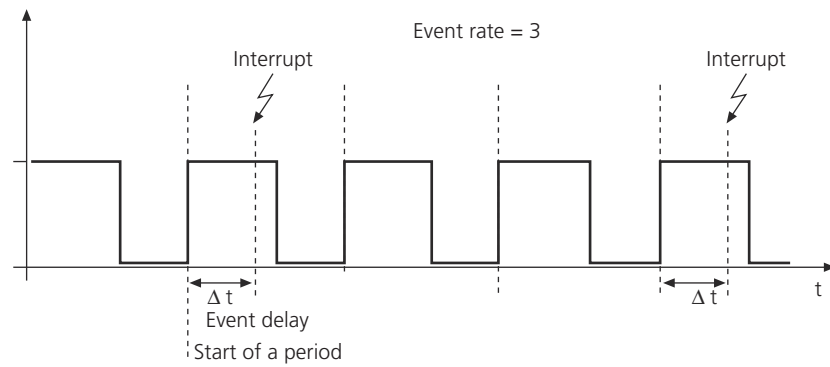
You can specify to generate interrupts and external trigger pulses during multi-channel PWM signal generation.

An interrupt can be used to trigger an interrupt service routine, an external trigger pulse can be used for external triggering. For example, you can use an external trigger pulse to start an A/D conversion. When you enable the trigger pulse generation an additional output channel is automatically allocated.

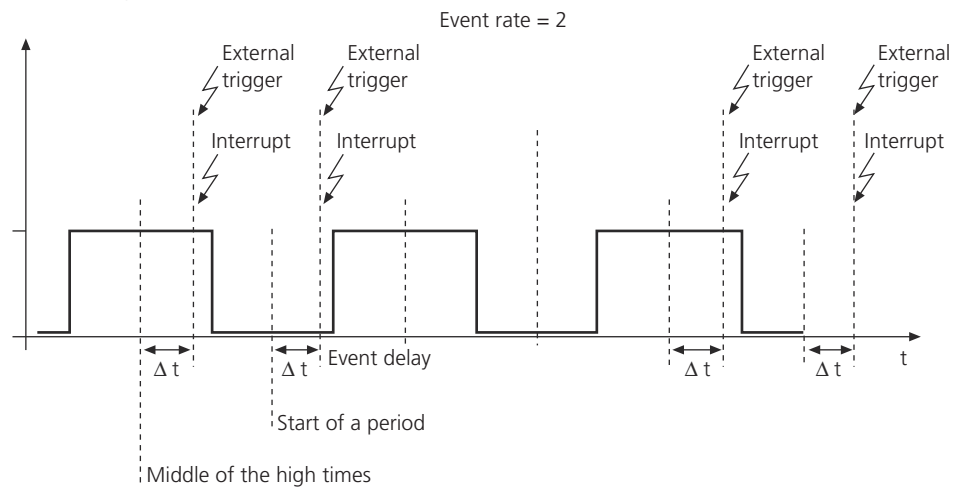
Interrupts and external trigger pulses can be generated at the start and/or in the middle of a period.

The *generation* of interrupts and external trigger pulses can be specified separately. Either you enable the interrupt generation or the trigger pulse generation, or both. However, the *configuration* of interrupts and external trigger pulses is commonly. Interrupts and external trigger pulses are handled as *events* of the multi-channel PWM signal generation. For these events, you can specify an event rate in the range 1 ... 255 and a delay.

The following example shows an edge-aligned multi-channel PWM signal generation with an event rate of 3 and an event delay Δt . Each 3rd rising edge (start of a period) is used for generating an event (interrupt and/or external trigger pulse).

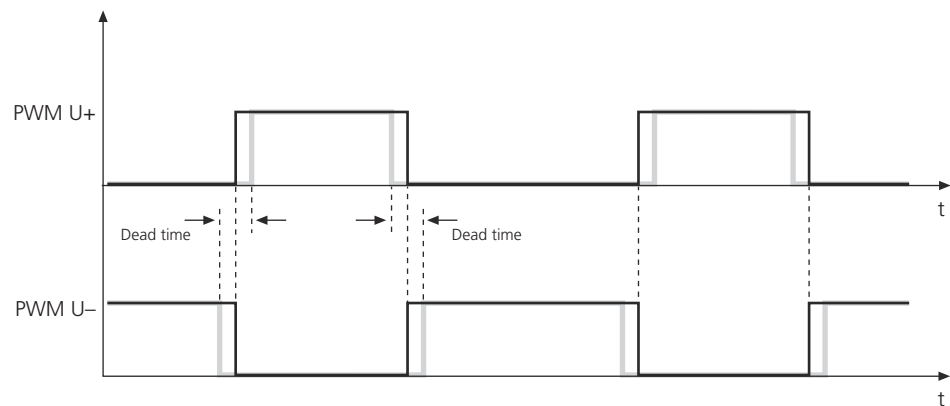


The following example shows a center-aligned multi-channel PWM signal generation with an event rate of 2 and an event delay Δt . An interrupt and an external trigger pulse are generated at each second middle of the high times (middle of the period) and at each second middle of the low times (start of a period).



Specifying dead time

With the dead time parameter, you can define the timing relationship between the PWM output signals and their corresponding inverted output signals for center-aligned PWM signal generation. The specified dead time is valid for all the channels of the current multi-channel PWM group. Dead times are used, for example, to avoid ripple currents or prevent shoot-through currents on the phase drivers. The high times of the inverted and non-inverted output signals are reduced symmetrically. The following illustration shows center-aligned multi-channel PWM with a specified dead time.



The dead time can be specified in the range 0 ... 12.8 μs . If the high time of the inverted or non-inverted signal is less than the specified dead time, the signal remains in low state.

Configuring the switches

The output state of a digital output channel depends on its individual settings for the low-side switch L (GND) and the high-side switch H (VDRIVE).

- If the low-side switch L (GND) is enabled, the output is actively driven to GND.
- If the high-side switch H (VDRIVE) is enabled, the output is actively driven to VDRIVE.
- If you set low-side switch L (GND) and high-side switch H (VDRIVE), the digital output channel is actively driven to both VDRIVE and GND (push-pull mode).

Period ranges and resolutions

The period range is divided into 16 ranges. For each range, there is a specific resolution.

Due to quantization effects, you might encounter considerable deviations between the desired PWM period and the generated PWM period, especially for higher PWM frequencies. To avoid poor frequency resolution, you should therefore select the period range with the best possible resolution (resolution values as small as possible).

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
1	200 ns ¹⁾	3.27 ms	50 ns
2	400 ns ¹⁾	6.55 ms	100 ns
3	800 ns ¹⁾	13.1 ms	200 ns
4	1.6 μs ¹⁾	26.2 ms	400 ns
5	3.2 μs ¹⁾	52.4 ms	800 ns
6	6.4 μs ¹⁾	104 ms	1.6 μs
7	12.8 μs	209 ms	3.2 μs
8	25.6 μs	419 ms	6.4 μs

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
9	51.2 μ s	838 ms	12.8 μ s
10	103 μ s	1.67 s	25.6 μ s
11	205 μ s	3.35 s	51.2 μ s
12	410 μ s	6.71 s	103 μ s
13	820 μ s	13.4 s	205 μ s
14	1.64 ms	26.8 s	410 μ s
15	3.28 ms	53.6 s	820 μ s
16	6.56 ms	107.3 s	1.64 ms

¹⁾ This is a theoretical value. In practice, the minimum period is limited to 6.7 μ s.

RTI/RTLib support

The multi-channel PWM signal generation is supported by RTI and RTLib. For detailed information, refer to:

- [DIO_TYPE4_MC_PWM_BLx \(MicroAutoBox II RTI Reference !\[\]\(448bd415caa8b52d2aeb4d58499267b2_img.jpg\)](#))
- [Multi-Channel PWM Generation \(MC_PWM\) on the DIO Type 4 \(MicroAutoBox II RTLib Reference !\[\]\(23be4c52910c50d5908bb101588c4f4e_img.jpg\)](#))

I/O mapping

The following table shows the order of the generated signals.

Without Inverted Signals	With Inverted Signals
Non-inverted signal 1	Non-inverted signal 1
...	...
Non-inverted signal n	Non-inverted signal n
External trigger signal (optional)	Inverted signal 1
	...
	Inverted signal n
	External trigger signal (optional)

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

The following table shows the mapping of the port number and channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	G2
	2	DigP 1 ch 2	F2
	3	DigP 1 ch 3	E2
	4	DigP 1 ch 4	D2
	5	DigP 1 ch 5	C2
	6	DigP 1 ch 6	G3
	7	DigP 1 ch 7	F3
	8	DigP 1 ch 8	E3
	9	DigP 1 ch 9	D3
	10	DigP 1 ch 10	C3
	11	DigP 1 ch 11	G4
	12	DigP 1 ch 12	F4
	13	DigP 1 ch 13	E4
	14	DigP 1 ch 14	D4
	15	DigP 1 ch 15	C4
	16	DigP 1 ch 16	G5
2	1	DigP 2 ch 1	F5
	2	DigP 2 ch 2	E5
	3	DigP 2 ch 3	D5
	4	DigP 2 ch 4	C5
	5	DigP 2 ch 5	G6
	6	DigP 2 ch 6	F6
	7	DigP 2 ch 7	E6
	8	DigP 2 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital output channel(s) by other DIO Type 4 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(b1b781be830eb908d845c527ab08d5f8_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(2176a4ba510fa27404d783166e891577_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Generation Functionalities.....](#) 144

References

[DIO_TYPE4_MC_PWM_BLx \(MicroAutoBox II RTI Reference !\[\]\(339a16584d5da0f0a3ca4e9ec17bf6a1_img.jpg\)\)](#)
[Multi-Channel PWM Generation \(MC_PWM\) on the DIO Type 4 \(MicroAutoBox II RTLib Reference !\[\]\(e06a1d39938b2f5d7a2c3618fea4f77f_img.jpg\)\)](#)

PWM Signal Measurement (PWM2D)

Introduction

MicroAutoBox II provides the timing I/O feature to measure pulse-width modulated signals (PWM2D).

Where to go from here

Information in this section

Overview of the Signal Measurement Functionalities.....	195
PWM Measurement (PWM2D) on the DIO Type 3 Unit.....	197
PWM Measurement (PWM2D) on the DIO Type 4 Unit.....	202
PWM Measurement (PWM2D) on the DIO 1552 Type 1 Unit.....	205

Overview of the Signal Measurement Functionalities

Introduction

MicroAutoBox II provides various methods for signal measurement depending on the available hardware.

Hardware requirements

Supported MicroAutoBox II hardware:

Signal Measurement	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
PWM2D	—	✓	✓	✓ ¹⁾
F2D	—	✓	✓	✓ ¹⁾
PW2D	—	✓	✓	—

¹⁾ Requires a DS1552 Multi-I/O Module.

Overview of the characteristics

The signal measurement functions can be divided into the following three main groups:

- PWM signal measurement (PWM2D)
- Frequency measurement (F2D)
- Pulse width measurement (PW2D)

Characteristics of the functions for PWM signal measurement The following table shows the main characteristics. For detailed information, refer to the function descriptions.

Characteristics	PWM2D (DIO Type 3)	PWM2D (DIO Type 4)	PWM2D (DIO 1552 Type 1)
Number of input channels	40 (3 ports with 16/16/8 channels)	24 (2 ports with 16/8 channels)	16
Measured values	Pulse width Frequency		
Signal detection	Depends on the update mode		
Averaging	No		
Update mode	Synchronous Asynchronous		
Period range	6.7 μ s ... 107.3 s Divided into 16 subranges		
Resolution	50 ns ... 1.62 ms Depends on the specified range.		

Characteristics of the functions for pulse pattern and frequency measurement The following table shows the main characteristics. For detailed information, refer to the function descriptions.

Characteristics	F2D (DIO Type 3)	F2D (DIO Type 4)	F2D (DIO 1552 Type 1)
Number of input channels	40 (3 ports with 16/16/8 channels)	24 (2 ports with 16/8 channels)	16
Measured values	Frequency		
Signal detection	Rising edge		
Averaging	No		
Update mode	Synchronous		
Frequency range	0.3 mHz ... 150 kHz Divided into 16 subranges		
Resolution	50 ns ... 1.64 ms Depends on the specified period range		

Characteristics of the functions for pulse width measurement The following table shows the main characteristics. For detailed information, refer to the function descriptions.

Characteristics	PW2D (DIO Type 3)	PW2D (DIO Type 4)
Number of input channels	40 (3 ports with 16/16/8 channels)	24 (2 ports with 16/8 channels)
Measured values	Pulse width	
Measurement mode	High time pulses or low time pulses	

Characteristics	PW2D (DIO Type 3)	PW2D (DIO Type 4)
Averaging	No	
Update mode	After each specified pulse, according to the specified measurement mode	
Pulse width range	3.33 μ s ... 53.6 s Divided into 16 subranges	
Resolution	50 ns ... 1.64 ms Depends on the specified range.	

Related topics**Basics**

Frequency Measurement (F2D) on the DIO 1552 Type 1 Unit.....	216
Frequency Measurement (F2D) on the DIO Type 3 Unit.....	209
Frequency Measurement (F2D) on the DIO Type 4 Unit.....	213
PWM Measurement (PWM2D) on the DIO 1552 Type 1 Unit.....	205
PWM Measurement (PWM2D) on the DIO Type 3 Unit.....	197
PWM Measurement (PWM2D) on the DIO Type 4 Unit.....	202

PWM Measurement (PWM2D) on the DIO Type 3 Unit

Purpose

To measure the frequency and duty cycle of PWM signals.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
—	—	✓	—	—

Characteristics

40 digital inputs are available for analyzing the frequency and duty cycle of PWM signals. Three subsequent edges are detected and used to measure the time of the related period. With the measured times, the frequency and duty cycle are calculated and returned.

The inputs are shared with the other DIO Type 3 input functions. The input channels are grouped to three ports with a different number of channels.

Port	Number of Channels/Bits
1	16
2	16
3	8

The following ranges are defined for PWM2D measurement:

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
1	200 ns ¹⁾	3.27 ms	50 ns
2	400 ns ¹⁾	6.55 ms	100 ns
3	800 ns ¹⁾	13.1 ms	200 ns
4	1.6 µs ¹⁾	26.2 ms	400 ns
5	3.2 µs ¹⁾	52.4 ms	800 ns
6	6.4 µs ¹⁾	104 ms	1.6 µs
7	12.8 µs	209 ms	3.2 µs
8	25.6 µs	419 ms	6.4 µs
9	51.2 µs	838 ms	12.8 µs
10	103 µs	1.67 s	25.6 µs
11	205 µs	3.35 s	51.2 µs
12	410 µs	6.71 s	103 µs
13	820 µs	13.4 s	205 µs
14	1.64 ms	26.8 s	410 µs
15	3.28 ms	53.6 s	820 µs
16	6.56 ms	107.3 s	1.64 ms

¹⁾ This is a theoretical value. In practice, the minimum period is limited to 6.7 µs.

Note

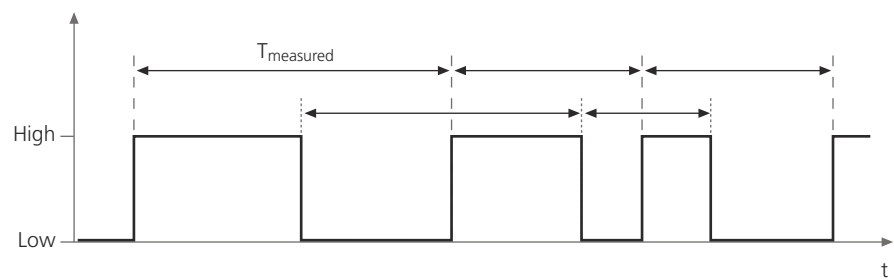
- A measured period that is above the maximum range limit results in a frequency of 0 Hz. If the counter for the low periods has an overflow, the duty cycle is set to 0. If the counter for the high periods has an overflow, the duty cycle is set to 1. If both counters have an overflow, the duty cycle is also set to 0.
- A measured period that is below the minimal range limit returns unpredictable results due to alias effects.

Averaging There is no averaging, the measured values are based on one period.

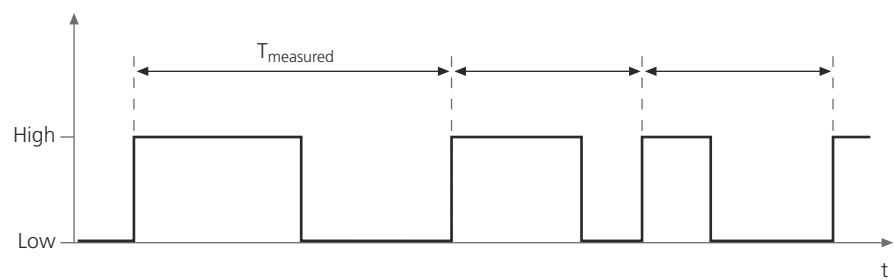
Update mode

The measured values can be updated in different modes.

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



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



I/O mapping

The following table shows the mapping of the port number and PWM2D channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	V2
	2	DigP 1 ch 2	U2
	3	DigP 1 ch 3	T2
	4	DigP 1 ch 4	S2
	5	DigP 1 ch 5	R2
	6	DigP 1 ch 6	P2
	7	DigP 1 ch 7	N2
	8	DigP 1 ch 8	M2
	9	DigP 1 ch 9	V3
	10	DigP 1 ch 10	U3
	11	DigP 1 ch 11	T3
	12	DigP 1 ch 12	S3
	13	DigP 1 ch 13	R3
	14	DigP 1 ch 14	P3
	15	DigP 1 ch 15	N3
	16	DigP 1 ch 16	M3
2	1	DigP 2 ch 1	V4
	2	DigP 2 ch 2	U4
	3	DigP 2 ch 3	T4
	4	DigP 2 ch 4	S4
	5	DigP 2 ch 5	R4
	6	DigP 2 ch 6	P4
	7	DigP 2 ch 7	N4
	8	DigP 2 ch 8	M4
	9	DigP 2 ch 9	V5
	10	DigP 2 ch 10	U5
	11	DigP 2 ch 11	T5
	12	DigP 2 ch 12	S5
	13	DigP 2 ch 13	R5
	14	DigP 2 ch 14	P5
	15	DigP 2 ch 15	N5
	16	DigP 2 ch 16	M5

Port	Channel	Signal	I/O Connector Pin
3	1	DigP 3 ch 1	V6
	2	DigP 3 ch 2	U6
	3	DigP 3 ch 3	T6
	4	DigP 3 ch 4	S6
	5	DigP 3 ch 5	R6
	6	DigP 3 ch 6	P6
	7	DigP 3 ch 7	N6
	8	DigP 3 ch 8	M6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital input channel by other DIO Type 3 blocks or functions is not supported.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(b1b781be830eb908d845c527ab08d5f8_img.jpg\)](#))
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(2176a4ba510fa27404d783166e891577_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Measurement Functionalities..... 195](#)

References

[DIO_TYPE3_PWM2D_BLx \(MicroAutoBox II RTI Reference !\[\]\(d3102649f02e825ddb76dc3de0190154_img.jpg\)](#))

[PWM Signal Measurement \(PWM2D\) on the DIO Type 3 \(MicroAutoBox II RTLib Reference !\[\]\(55ca3a38dbb940110628e54e3ea7505d_img.jpg\)](#))

PWM Measurement (PWM2D) on the DIO Type 4 Unit

Purpose To measure the frequency and duty cycle of PWM signals.

Hardware requirements Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Characteristics 24 digital inputs are available for analyzing the frequency and duty cycle of PWM signals. Three subsequent edges are detected and used to measure the time of the related period. With the measured times, the frequency and duty cycle are calculated and returned.

The inputs are shared with the other DIO Type 4 input functions. The input channels are grouped to two ports with a different number of channels.

Port	Number of Channels/Bits
1	16
2	8

The following ranges are defined for PWM2D measurement:

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
1	200 ns ¹⁾	3.27 ms	50 ns
2	400 ns ¹⁾	6.55 ms	100 ns
3	800 ns ¹⁾	13.1 ms	200 ns
4	1.6 µs ¹⁾	26.2 ms	400 ns
5	3.2 µs ¹⁾	52.4 ms	800 ns
6	6.4 µs ¹⁾	104 ms	1.6 µs
7	12.8 µs	209 ms	3.2 µs
8	25.6 µs	419 ms	6.4 µs
9	51.2 µs	838 ms	12.8 µs
10	103 µs	1.67 s	25.6 µs

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
11	205 μ s	3.35 s	51.2 μ s
12	410 μ s	6.71 s	103 μ s
13	820 μ s	13.4 s	205 μ s
14	1.64 ms	26.8 s	410 μ s
15	3.28 ms	53.6 s	820 μ s
16	6.56 ms	107.3 s	1.64 ms

¹⁾ This is a theoretical value. In practice, the minimum period is limited to 6.7 μ s.

Note

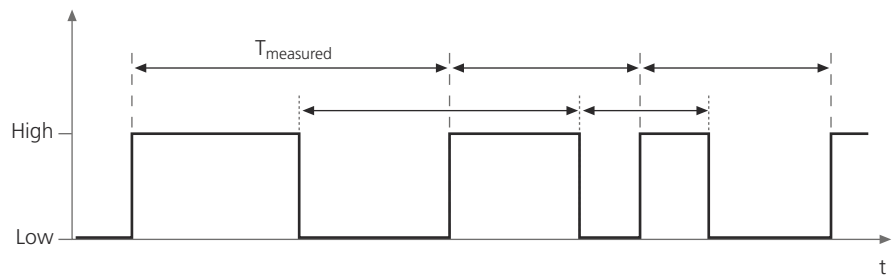
- A measured period that is above the maximum range limit results in a frequency of 0 Hz. If the counter for the low periods has an overflow, the duty cycle is set to 0. If the counter for the high periods has an overflow, the duty cycle is set to 1. If both counters have an overflow, the duty cycle is also set to 0.
- A measured period that is below the minimal range limit returns unpredictable results due to alias effects.

Averaging There is no averaging, the measured values are based on one period.

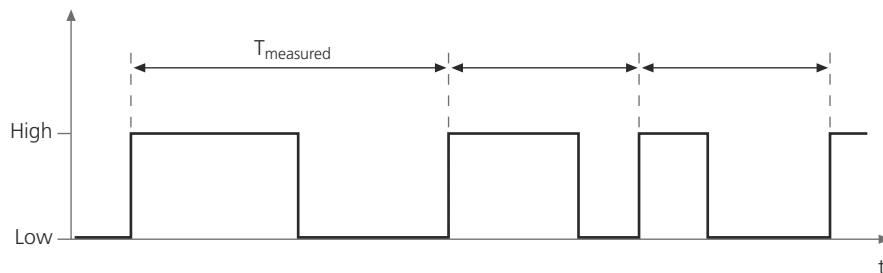
Update mode

The measured values can be updated in different modes.

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



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



I/O mapping

The following table shows the mapping of the port number and PWM2D channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	M2
	2	DigP 1 ch 2	L2
	3	DigP 1 ch 3	K2
	4	DigP 1 ch 4	J2
	5	DigP 1 ch 5	H2
	6	DigP 1 ch 6	M3
	7	DigP 1 ch 7	L3
	8	DigP 1 ch 8	K3
	9	DigP 1 ch 9	J3
	10	DigP 1 ch 10	H3
	11	DigP 1 ch 11	M4
	12	DigP 1 ch 12	L4
	13	DigP 1 ch 13	K4
	14	DigP 1 ch 14	J4
	15	DigP 1 ch 15	H4
	16	DigP 1 ch 16	M5

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	L5
	2	DigP 2 ch 2	K5
	3	DigP 2 ch 3	J5
	4	DigP 2 ch 4	H5
	5	DigP 2 ch 5	M6
	6	DigP 2 ch 6	L6
	7	DigP 2 ch 7	K6
	8	DigP 2 ch 8	J6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital input channel by other DIO Type 4 blocks or functions is not supported.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(9a53fe79a03d38d8322f7a2c5a875b36_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(01f19d40f03100aa8a158c4891453b0d_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Measurement Functionalities..... 195](#)

References

[DIO_TYPE4_PWM2D_BLx \(MicroAutoBox II RTI Reference !\[\]\(47734e4656765d20df4fdbd5b7aff048_img.jpg\)](#))
[PWM Signal Measurement \(PWM2D\) on the DIO Type 4 \(MicroAutoBox II RTLib Reference !\[\]\(effba44ea72cb8c77bdc1dac75561f86_img.jpg\)](#))

PWM Measurement (PWM2D) on the DIO 1552 Type 1 Unit

Purpose

To measure the pulse width, frequency and duty cycle of PWM signals using a DS1552 Multi-I/O Module.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	–	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module.**Characteristics**

16 digital inputs are available for analyzing the frequency and duty cycle of PWM signals. Three subsequent edges are detected and used to measure the time between the edges. With the measured times, the frequency and duty cycle are calculated and returned.

The inputs are shared with the other functions of the DIO 1552 Type 1 unit.

The following ranges are defined for PWM2D measurement:

Range	Meaning		
	Minimum Period	Maximum Period	Resolution
1	200 ns ¹⁾	3.27 ms	50 ns
2	400 ns ¹⁾	6.55 ms	100 ns
3	800 ns ¹⁾	13.1 ms	200 ns
4	1.6 µs ¹⁾	26.2 ms	400 ns
5	3.2 µs ¹⁾	52.4 ms	800 ns
6	6.4 µs ¹⁾	104 ms	1.6 µs
7	12.8 µs	209 ms	3.2 µs
8	25.6 µs	419 ms	6.4 µs
9	51.2 µs	838 ms	12.8 µs
10	103 µs	1.67 s	25.6 µs
11	205 µs	3.35 s	51.2 µs
12	410 µs	6.71 s	103 µs
13	820 µs	13.4 s	205 µs
14	1.64 ms	26.8 s	410 µs
15	3.28 ms	53.6 s	820 µs
16	6.56 ms	107.3 s	1.64 ms

¹⁾ This is a theoretical value. In practice, the minimum period is limited to 6.7 µs.

Note

- A measured period that is above the maximum range limit results in a frequency of 0 Hz. If the counter for the low periods has an overflow, the duty cycle is set to 0. If the counter for the high periods has an overflow, the duty cycle is set to 1. If both counters have an overflow, the duty cycle is also set to 0.
- A measured period that is below the minimal range limit returns unpredictable results due to alias effects.

Averaging There is no averaging, the measured values are based on one period.

Update mode

The measured values can be updated in different modes.

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

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

I/O mapping

The following table shows the mapping of the channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector (DS1514 ZIF connector).

Channel	Signal	I/O Connector Pin
1	DigIn ch 1	V5
2	DigIn ch 2	U5
3	DigIn ch 3	U6
4	DigIn ch 4	T2
5	DigIn ch 5	T3
6	DigIn ch 6	T4
7	DigIn ch 7	T5
8	DigIn ch 8	T6
9	DigIn ch 9	S2
10	DigIn ch 10	S3
11	DigIn ch 11	S5
12	DigIn ch 12	R2
13	DigIn ch 13	R5
14	DigIn ch 14	R6
15	DigIn ch 15	P5
16	DigIn ch 16	P6

Note

- Concurrent access to the same digital input channel by other DIO 1552 Type 1 blocks or functions is not allowed.
- DigIn ch 1 ... DigIn ch 4 are shared with the external trigger inputs of the ADC 1552 Type 1 unit.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(48a7667d09d5a06397e047ee4537bb6f_img.jpg\)](#))

Related topics**Basics**

[Overview of the Signal Measurement Functionalities.....](#) 195

References

[DIO1552_TP1_PWM2D_BLx \(MicroAutoBox II RTI Reference !\[\]\(f95dab70c751fda7d824b8b03650f7aa_img.jpg\)](#))
[PWM Signal Measurement \(PWM2D\) on the DIO 1552 Type 1 Unit of the DS1552 Multi-I/O Module \(MicroAutoBox II RTLib Reference !\[\]\(4f2c4dafe2b36117690cbd57dfbd3413_img.jpg\)](#))

Pulse Pattern Measurement

Introduction MicroAutoBox II provides the timing I/O feature to measure pulse patterns.

Where to go from here	Information in this section
	Frequency Measurement (F2D) on the DIO Type 3 Unit.....209
	Frequency Measurement (F2D) on the DIO Type 4 Unit.....213
	Frequency Measurement (F2D) on the DIO 1552 Type 1 Unit.....216

Frequency Measurement (F2D) on the DIO Type 3 Unit

Purpose To measure the frequency of a square-wave signal. The measurement covers the values of one period.

Hardware requirements Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	✓	–	–

Characteristics All the 40 digital inputs on the Digital I/O Unit Type 3 are available for frequency measurement. These inputs are shared with the other DIO Type 3 input functions. The input channels are grouped to three ports with a different number of channels.

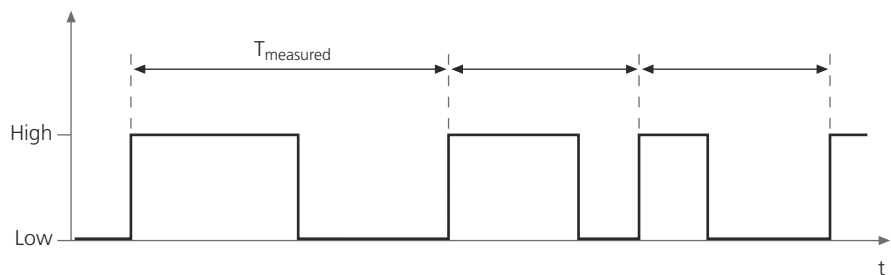
Port	Number of Channels/Bits
1	16
2	16
3	8

You can measure the frequency of signals in the frequency range 0.3 mHz ... 150 kHz. The following ranges are defined for F2D measurements:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	150 kHz	50 ns
2	4.77 Hz	150 kHz	100 ns
3	2.39 Hz	150 kHz	200 ns
4	1.20 Hz	150 kHz	400 ns
5	0.60 Hz	150 kHz	800 ns
6	0.30 Hz	150 kHz	1.6 μ s
7	0.15 Hz	150 kHz	3.2 μ s
8	75 mHz	78.12 kHz	6.4 μ s
9	38 mHz	39.06 kHz	12.8 μ s
10	19 mHz	19.53 kHz	25.6 μ s
11	10 mHz	9.76 kHz	51.2 μ s
12	5.0 mHz	4.88 kHz	103 μ s
13	2.5 mHz	2.44 kHz	205 μ s
14	1.2 mHz	1.22 kHz	410 μ s
15	0.6 mHz	610.35 Hz	820 μ s
16	0.3 mHz	305.17 Hz	1.64 ms

Averaging There is no averaging, the measured values are based on one period.

Update mode The measured values are updated at the end of each T_{low} period of the input signal only. The update is synchronous to the period.



Handling invalid frequencies

The measurement result is forced to zero if:

- The input signal frequency is less than the lower frequency range limit.
- The input signal is constantly LOW.
- The input signal is constantly HIGH.

Note

The measurement returns unpredictable results, if the input signal frequency is greater than the upper frequency range limit.

I/O mapping

The following table shows the mapping of the logical channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	V2
	2	DigP 1 ch 2	U2
	3	DigP 1 ch 3	T2
	4	DigP 1 ch 4	S2
	5	DigP 1 ch 5	R2
	6	DigP 1 ch 6	P2
	7	DigP 1 ch 7	N2
	8	DigP 1 ch 8	M2
	9	DigP 1 ch 9	V3
	10	DigP 1 ch 10	U3
	11	DigP 1 ch 11	T3
	12	DigP 1 ch 12	S3
	13	DigP 1 ch 13	R3
	14	DigP 1 ch 14	P3
	15	DigP 1 ch 15	N3
	16	DigP 1 ch 16	M3

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	V4
	2	DigP 2 ch 2	U4
	3	DigP 2 ch 3	T4
	4	DigP 2 ch 4	S4
	5	DigP 2 ch 5	R4
	6	DigP 2 ch 6	P4
	7	DigP 2 ch 7	N4
	8	DigP 2 ch 8	M4
	9	DigP 2 ch 9	V5
	10	DigP 2 ch 10	U5
	11	DigP 2 ch 11	T5
	12	DigP 2 ch 12	S5
	13	DigP 2 ch 13	R5
	14	DigP 2 ch 14	P5
	15	DigP 2 ch 15	N5
	16	DigP 2 ch 16	M5
3	1	DigP 3 ch 1	V6
	2	DigP 3 ch 2	U6
	3	DigP 3 ch 3	T6
	4	DigP 3 ch 4	S6
	5	DigP 3 ch 5	R6
	6	DigP 3 ch 6	P6
	7	DigP 3 ch 7	N6
	8	DigP 3 ch 8	M6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital input channel by other DIO Type 3 blocks or functions is not supported.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(7a8011739ec4e250e2f89a547d75fb0a_img.jpg\)\)](#)
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(07dce76283bf618e2364d95ae0021e26_img.jpg\)\)](#)

Related topics

Basics

[Overview of the Signal Measurement Functionalities.....](#) 195

References

[DIO_TYPE3_F2D_BLx \(MicroAutoBox II RTI Reference !\[\]\(74d4806277d7e73349d8e8c0897931e9_img.jpg\)\)](#)
[Frequency Measurement \(F2D\) on the DIO Type 3 \(MicroAutoBox II RTLib Reference !\[\]\(5f42d2cd7ad901bc24e5d35a38c777fd_img.jpg\)\)](#)

Frequency Measurement (F2D) on the DIO Type 4 Unit

Purpose

To measure the frequency of a square-wave signal. The measurement covers the values of one period.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Characteristics

All the 24 digital inputs on the Digital I/O Unit Type 4 are available for frequency measurement. These inputs are shared with the other DIO Type 4 input functions. The input channels are grouped to two ports with a different number of channels.

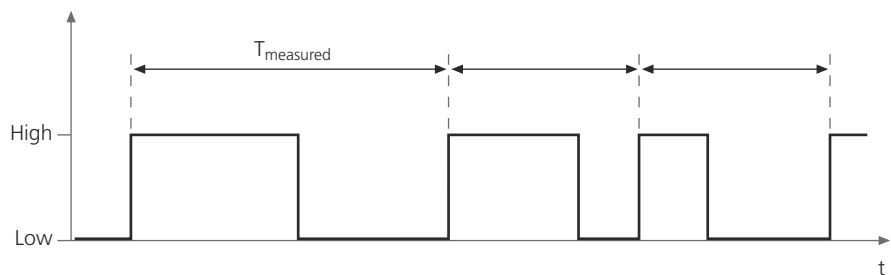
Port	Number of Channels/Bits
1	16
2	8

You can measure the frequency of signals in the frequency range 0.3 mHz ... 150 kHz. The following ranges are defined for F2D measurements:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	150 kHz	50 ns
2	4.77 Hz	150 kHz	100 ns
3	2.39 Hz	150 kHz	200 ns
4	1.20 Hz	150 kHz	400 ns
5	0.60 Hz	150 kHz	800 ns
6	0.30 Hz	150 kHz	1.6 μ s
7	0.15 Hz	150 kHz	3.2 μ s
8	75 mHz	78.12 kHz	6.4 μ s
9	38 mHz	39.06 kHz	12.8 μ s
10	19 mHz	19.53 kHz	25.6 μ s
11	10 mHz	9.76 kHz	51.2 μ s
12	5.0 mHz	4.88 kHz	103 μ s
13	2.5 mHz	2.44 kHz	205 μ s
14	1.2 mHz	1.22 kHz	410 μ s
15	0.6 mHz	610.35 Hz	820 μ s
16	0.3 mHz	305.17 Hz	1.64 ms

Averaging There is no averaging, the measured values are based on one period.

Update mode The measured values are updated at the end of each T_{low} period of the input signal only. The update is synchronous to the period.



Handling invalid frequencies

The measurement result is forced to zero if:

- The input signal frequency is less than the lower frequency range limit.
- The input signal is constantly LOW.
- The input signal is constantly HIGH.

Note

The measurement returns unpredictable results, if the input signal frequency is greater than the upper frequency range limit.

I/O mapping

The following table shows the mapping of the logical channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	M2
	2	DigP 1 ch 2	L2
	3	DigP 1 ch 3	K2
	4	DigP 1 ch 4	J2
	5	DigP 1 ch 5	H2
	6	DigP 1 ch 6	M3
	7	DigP 1 ch 7	L3
	8	DigP 1 ch 8	K3
	9	DigP 1 ch 9	J3
	10	DigP 1 ch 10	H3
	11	DigP 1 ch 11	M4
	12	DigP 1 ch 12	L4
	13	DigP 1 ch 13	K4
	14	DigP 1 ch 14	J4
	15	DigP 1 ch 15	H4
	16	DigP 1 ch 16	M5
2	1	DigP 2 ch 1	L5
	2	DigP 2 ch 2	K5
	3	DigP 2 ch 3	J5
	4	DigP 2 ch 4	H5
	5	DigP 2 ch 5	M6
	6	DigP 2 ch 6	L6
	7	DigP 2 ch 7	K6
	8	DigP 2 ch 8	J6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital input channel by other DIO Type 4 blocks or functions is not supported.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(cd3e54d951a9fb854f48e4697cf550f9_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(cc729e263f29c0a76fbdc4cfe67fceb0_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Measurement Functionalities..... 195](#)

References

[DIO_TYPE4_F2D_BLx \(MicroAutoBox II RTI Reference !\[\]\(fa6f3af6bfa46c5d4a2d362681095beb_img.jpg\)](#))

[Frequency Measurement \(F2D\) on the DIO Type 4 \(MicroAutoBox II RTLib Reference !\[\]\(a9bc825d1a15412853cf9ebcbd72219d_img.jpg\)](#))

Frequency Measurement (F2D) on the DIO 1552 Type 1 Unit

Purpose

To measure the frequency of a square-wave signal using a DS1552 Multi-I/O Module. The measurement covers the values of one period.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	–	✓ ¹⁾

¹⁾ Requires a DS1552 Multi-I/O Module.

Characteristics

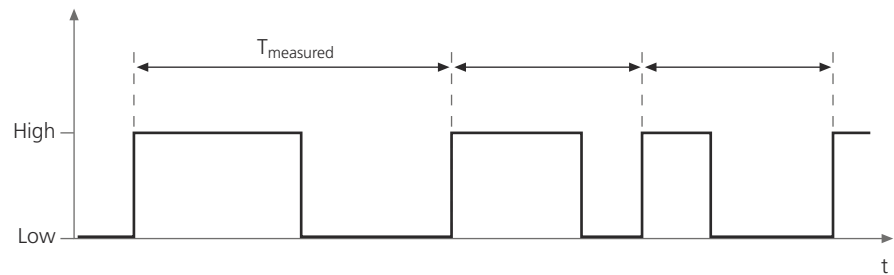
16 digital inputs on the DIO 1552 Type 1 unit of the DS1552 Multi-I/O Module are available for frequency measurement. These inputs are shared with the other DIO 1552 Type 1 input functions.

You can measure the frequency of signals in the frequency range 0.3 mHz ... 150 kHz. The following ranges are defined for F2D measurements:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	150 kHz	50 ns
2	4.77 Hz	150 kHz	100 ns
3	2.39 Hz	150 kHz	200 ns
4	1.20 Hz	150 kHz	400 ns
5	0.60 Hz	150 kHz	800 ns
6	0.30 Hz	150 kHz	1.6 μ s
7	0.15 Hz	150 kHz	3.2 μ s
8	75 mHz	78.12 kHz	6.4 μ s
9	38 mHz	39.06 kHz	12.8 μ s
10	19 mHz	19.53 kHz	25.6 μ s
11	10 mHz	9.76 kHz	51.2 μ s
12	5.0 mHz	4.88 kHz	103 μ s
13	2.5 mHz	2.44 kHz	205 μ s
14	1.2 mHz	1.22 kHz	410 μ s
15	0.6 mHz	610.35 Hz	820 μ s
16	0.3 mHz	305.17 Hz	1.64 ms

Averaging There is no averaging, the measured values are based on one period.

Update mode The measured values are updated at the end of each T_{low} period of the input signal only. The update is synchronous to the period.



Handling invalid frequencies

The measurement result is forced to zero if:

- The input signal frequency is less than the lower frequency range limit.
- The input signal is constantly LOW.
- The input signal is constantly HIGH.

Note

The measurement returns unpredictable results, if the input signal frequency is greater than the upper frequency range limit.

I/O mapping

The following table shows the mapping of the channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector (DS1514 ZIF connector).

Channel	Signal	I/O Connector Pin
1	DigIn ch 1	V5
2	DigIn ch 2	U5
3	DigIn ch 3	U6
4	DigIn ch 4	T2
5	DigIn ch 5	T3
6	DigIn ch 6	T4
7	DigIn ch 7	T5
8	DigIn ch 8	T6
9	DigIn ch 9	S2
10	DigIn ch 10	S3
11	DigIn ch 11	S5
12	DigIn ch 12	R2
13	DigIn ch 13	R5
14	DigIn ch 14	R6
15	DigIn ch 15	P5
16	DigIn ch 16	P6

Note

- Concurrent access to the same digital input channel by other DIO 1552 Type 1 blocks or functions is not allowed.
- DigIn ch 1 ... DigIn ch 4 are shared with the external trigger inputs of the ADC 1552 Type 1 unit.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(e27c4336460e9e6729a19580c0456728_img.jpg\)](#))

Related topics

Basics

Overview of the Signal Measurement Functionalities.....	195
---	---------------------

References

[DIO1552_TP1_F2D_BLx \(MicroAutoBox II RTI Reference !\[\]\(47572387bd17ceb152153a4149a71b7c_img.jpg\)\)](#)
[Frequency Measurement \(F2D\) on the DIO 1552 Type 1 Unit of the DS1552 Multi-I/O Module \(MicroAutoBox II RTLib Reference !\[\]\(20b2ab454a59a31305c644f3027d9474_img.jpg\)\)](#)

Pulse Width Measurement (PW2D)

Introduction

MicroAutoBox II provides the timing I/O feature to measure the pulse width of square-wave signals (PW2D).

Where to go from here

Information in this section

[Pulse Width Measurement \(PW2D\) on the DIO Type 3 Unit.....220](#)

[Pulse Width Measurement \(PW2D\) on the DIO Type 4 Unit.....224](#)

Pulse Width Measurement (PW2D) on the DIO Type 3 Unit

Purpose

To measure the pulse width of a square-wave signal.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	✓	–	–

Characteristics

40 digital inputs are available for analyzing the pulse width of square-wave signals. Each pulse is detected and used to measure its width. You can configure whether to use the high pulse or the low pulse of a signal for measurement.

The inputs are shared with the other DIO Type 3 input functions. The input channels are grouped to three ports with a different number of channels.

The following ranges are defined for PW2D measurement:

Range	Meaning		
	Minimum Pulse Width	Maximum Pulse Width	Resolution
1	3.33 μs ¹⁾	1.63 ms	50 ns
2	3.33 μs ¹⁾	3.27 ms	100 ns
3	3.33 μs ¹⁾	6.55 ms	200 ns
4	3.33 μs ¹⁾	13.1 ms	400 ns
5	3.33 μs ¹⁾	26.2 ms	800 ns
6	3.33 μs ¹⁾	52.4 ms	1.6 μs
7	6.4 μs	104 ms	3.2 μs
8	12.8 μs	209 ms	6.4 μs
9	25.6 μs	419 ms	12.8 μs
10	51.2 μs	838 ms	25.6 μs
11	103 μs	1.67 s	51.2 μs
12	205 μs	3.35 s	103 μs
13	410 μs	6.71 s	205 μs
14	820 μs	13.4 s	410 μs
15	1.64 ms	26.8 s	820 μs
16	3.28 ms	53.6 s	1.64 ms

¹⁾ While the theoretical minimum value is in the range 100 ns ... 3.2 μs , the minimum value to be measured in practice is limited to 3.33 μs .

Note

The pulse width of the input signal must remain in the specified pulse width range, otherwise the measured value is not correct.

- If the pulse width is less than the lower limit, the measured pulse width is unpredictable.
- If the pulse width is higher than the upper limit, the measured pulse width is detected as the maximum float value (FLT_MAX).

Averaging There is no averaging, the measured values are based on one pulse.

Update mode

The measured values are updated at the end of each pulse. If you measure high pulses, the values are updated at the falling edges of the signal. If you measure low pulses, the values are updated at the rising edges of the signal.

Measurement mode

The pulse width can be measured either for the high times of the connected signal or the low times. You configure the measurement mode by specifying the edge polarity to be used for measurement.

Measurement Mode	Meaning
Rising edge	The measurement starts with a rising edge of the connected signal so that the high time of the pulse is measured.
Falling edge	The measurement starts with a falling edge of the connected signal so that the low time of the pulse is measured.

I/O mapping

The following table shows the mapping of the port number and PW2D channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	V2
	2	DigP 1 ch 2	U2
	3	DigP 1 ch 3	T2
	4	DigP 1 ch 4	S2
	5	DigP 1 ch 5	R2
	6	DigP 1 ch 6	P2
	7	DigP 1 ch 7	N2
	8	DigP 1 ch 8	M2
	9	DigP 1 ch 9	V3
	10	DigP 1 ch 10	U3
	11	DigP 1 ch 11	T3
	12	DigP 1 ch 12	S3
	13	DigP 1 ch 13	R3
	14	DigP 1 ch 14	P3
	15	DigP 1 ch 15	N3
	16	DigP 1 ch 16	M3

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	V4
	2	DigP 2 ch 2	U4
	3	DigP 2 ch 3	T4
	4	DigP 2 ch 4	S4
	5	DigP 2 ch 5	R4
	6	DigP 2 ch 6	P4
	7	DigP 2 ch 7	N4
	8	DigP 2 ch 8	M4
	9	DigP 2 ch 9	V5
	10	DigP 2 ch 10	U5
	11	DigP 2 ch 11	T5
	12	DigP 2 ch 12	S5
	13	DigP 2 ch 13	R5
	14	DigP 2 ch 14	P5
	15	DigP 2 ch 15	N5
	16	DigP 2 ch 16	M5
3	1	DigP 3 ch 1	V6
	2	DigP 3 ch 2	U6
	3	DigP 3 ch 3	T6
	4	DigP 3 ch 4	S6
	5	DigP 3 ch 5	R6
	6	DigP 3 ch 6	P6
	7	DigP 3 ch 7	N6
	8	DigP 3 ch 8	M6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital input channel by other DIO Type blocks or functions is not supported.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(5774573cf757c446bb08af21f46b2969_img.jpg\)\)](#)
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(a502cb21d600ba28a5cdf414d68eef89_img.jpg\)\)](#)

Related topics**Basics**

[Overview of the Signal Measurement Functionalities.....](#) 195

References

[DIO_TYPE3_PW2D_BLx \(MicroAutoBox II RTI Reference !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)\)](#)
[Pulse Width Measurement \(PW2D\) on the DIO Type 3 \(MicroAutoBox II RTLib Reference !\[\]\(e658400d40ca763c7cf4c8c420885c6a_img.jpg\)\)](#)

Pulse Width Measurement (PW2D) on the DIO Type 4 Unit

Purpose

To measure the pulse width of a square-wave signal.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Characteristics

24 digital inputs are available for analyzing the pulse width of square-wave signals. Each pulse is detected and used to measure its width. You can configure whether to use the high pulse or the low pulse of a signal for measurement.

The inputs are shared with the other DIO Type 4 input functions. The input channels are grouped to two ports with a different number of channels.

The following ranges are defined for PW2D measurement:

Range	Meaning		
	Minimum Pulse Width	Maximum Pulse Width	Resolution
1	3.33 μs ¹⁾	1.63 ms	50 ns
2	3.33 μs ¹⁾	3.27 ms	100 ns
3	3.33 μs ¹⁾	6.55 ms	200 ns
4	3.33 μs ¹⁾	13.1 ms	400 ns
5	3.33 μs ¹⁾	26.2 ms	800 ns

Range	Meaning		
	Minimum Pulse Width	Maximum Pulse Width	Resolution
6	3.33 μs ¹⁾	52.4 ms	1.6 μs
7	6.4 μs	104 ms	3.2 μs
8	12.8 μs	209 ms	6.4 μs
9	25.6 μs	419 ms	12.8 μs
10	51.2 μs	838 ms	25.6 μs
11	103 μs	1.67 s	51.2 μs
12	205 μs	3.35 s	103 μs
13	410 μs	6.71 s	205 μs
14	820 μs	13.4 s	410 μs
15	1.64 ms	26.8 s	820 μs
16	3.28 ms	53.6 s	1.64 ms

¹⁾ While the theoretical minimum value is in the range 100 ns ... 3.2 μs , the minimum value to be measured in practice is limited to 3.33 μs .

Note

The pulse width of the input signal must remain in the specified pulse width range, otherwise the measured value is not correct.

- If the pulse width is less than the lower limit, the measured pulse width is unpredictable.
- If the pulse width is higher than the upper limit, the measured pulse width is detected as the maximum float value (FLT_MAX).

Averaging There is no averaging, the measured values are based on one pulse.

Update mode

The measured values are updated at the end of each pulse. If you measure high pulses, the values are updated at the falling edges of the signal. If you measure low pulses, the values are updated at the rising edges of the signal.

Measurement mode

The pulse width can be measured either for the high times of the connected signal or the low times. You configure the measurement mode by specifying the edge polarity to be used for measurement.

Measurement Mode	Meaning
Rising edge	The measurement starts with a rising edge of the connected signal so that the high time of the pulse is measured.

Measurement Mode	Meaning
Falling edge	The measurement starts with a falling edge of the connected signal so that the low time of the pulse is measured.

I/O mapping

The following table shows the mapping of the port number and PW2D channel number, as used in RTI and RTLib, to the related I/O pin of the MicroAutoBox II I/O connector.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	M2
	2	DigP 1 ch 2	L2
	3	DigP 1 ch 3	K2
	4	DigP 1 ch 4	J2
	5	DigP 1 ch 5	H2
	6	DigP 1 ch 6	M3
	7	DigP 1 ch 7	L3
	8	DigP 1 ch 8	K3
	9	DigP 1 ch 9	J3
	10	DigP 1 ch 10	H3
	11	DigP 1 ch 11	M4
	12	DigP 1 ch 12	L4
	13	DigP 1 ch 13	K4
	14	DigP 1 ch 14	J4
	15	DigP 1 ch 15	H4
	16	DigP 1 ch 16	M5
2	1	DigP 2 ch 1	L5
	2	DigP 2 ch 2	K5
	3	DigP 2 ch 3	J5
	4	DigP 2 ch 4	H5
	5	DigP 2 ch 5	M6
	6	DigP 2 ch 6	L6
	7	DigP 2 ch 7	K6
	8	DigP 2 ch 8	J6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital input channel by other DIO Type blocks or functions is not supported.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(746d018fdf6ab02bf5fb7681133e8b29_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(5daa6eee1904cb6b9d765700250de764_img.jpg\)](#))

Related topics

Basics

[Overview of the Signal Measurement Functionalities.....](#) 195

References

[DIO_TYPE4_PWM2D_BLx \(MicroAutoBox II RTI Reference !\[\]\(e474458956c9a37fbf9586ddb60a7fa1_img.jpg\)](#))
[Pulse Width Measurement \(PW2D\) on the DIO Type 4 \(MicroAutoBox II RTLib Reference !\[\]\(4d1d3f2547aeece54bb6babd23f4121b_img.jpg\)](#))

Incremental Encoder Interface

Introduction

MicroAutoBox II provides the timing I/O feature to decode the signals of an incremental encoder.

Where to go from here

Information in this section

Overview of the Incremental Encoder Interfaces.....	228
Basics on the Incremental Encoder Interface.....	229
Incremental Encoder Interface on the DIO Type 3 Unit.....	231
Incremental Encoder Interface on the DIO Type 4 Unit.....	236

Overview of the Incremental Encoder Interfaces

Introduction

MicroAutoBox II provides various incremental encoder interfaces.

Hardware requirements

Supported MicroAutoBox II hardware:

Incremental Encoder	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
DIO Type 3	–	✓	–	–
DIO Type 4	–	–	✓	–

Overview of the characteristics

The main characteristics of the incremental encoder interfaces are listed below.

Characteristics	Incremental Encoder	
	(DIO Type 3)	(DIO Type 4)
Encoder interfaces	4	
Index pulse support	Yes	
Position count range	-2,097,152.0 ... +2,097,151.75 (-2^{21} ... $+2^{21}-0.25$)	
Adjustable parameters	Start position Index position	

Characteristics	Incremental Encoder	
	(DIO Type 3)	(DIO Type 4)
Interrupt generation	Min. count value	
	Max. count value	
	Noise filter	
	Gated mode	
	Yes ¹⁾	

¹⁾ For information on the DIO Type 3 and DIO Type 4 interrupts, refer to [Basics on Interrupt Handling](#) on page 37.

Related topics

Basics

Basics on Interrupt Handling	37
Incremental Encoder Interface on the DIO Type 3 Unit	231
Incremental Encoder Interface on the DIO Type 4 Unit	236

Basics on the Incremental Encoder Interface

Introduction

The incremental encoder interface can be used to decode signals of an incremental encoder.

Note

Only single-ended encoders are supported. Differential and analog encoders cannot be used.

Except for drives control center aligned, encoder index signals are supported. The index is used to update the internal position counter with the predefined index value.

Measuring signals

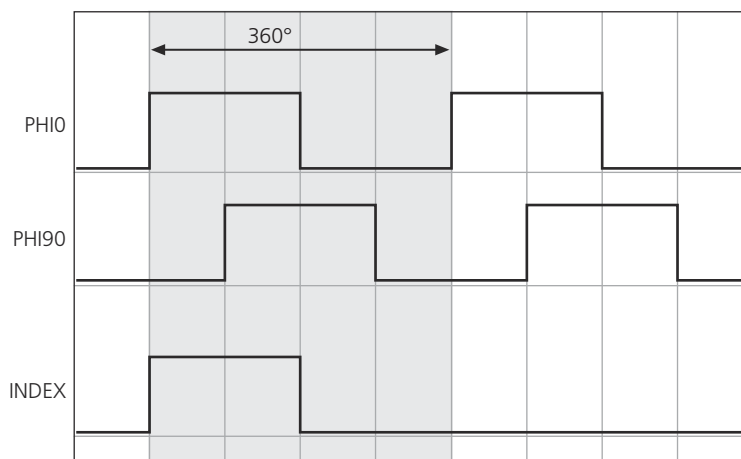
Incremental encoders provide the two encoder signals PHIO and PHI90 and the optional index signal IDX. The encoder signals PHIO and PHI90 have a phase shift of 90°. In addition, most encoders also provide the inverted signals /PHIO, /PHI90 and /IDX.

Tip

Some encoder manufacturers use the terms A, B and Z instead of PHIO, PHI90 and IDX.

The following illustration shows the shape of the PHIO and PHI90 digital signals together with the optional index signal. It also shows the 4-fold subdivision of an

encoder line. The gray-shaded area represents one encoder line (360° means one period). For the number of encoder lines per rotation, refer to the encoder user documentation.



A position counter stores the current value. The count direction depends on the encoder's rotation direction.

Note

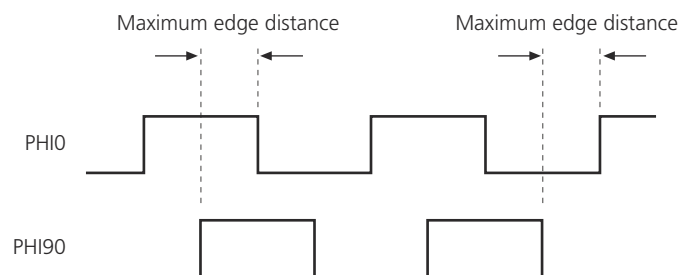
The incremental encoder interface for DCCA does not support encoder index signals.

Measuring speed

The rotation speed is calculated as the ratio between the encoder positions since the last call and the related time intervals. The unit of the returned value is lines/s in steps of 0.25. The sign of the value represents the direction of the encoder rotation. A positive value represents a clockwise rotation.

Maximum edge distance

With RTI, you can configure the capture resolution by specifying the maximum distance between two subsequent edges, see the figure below.



Incremental Encoder Interface on the DIO Type 3 Unit

Introduction

The incremental encoder interface can be used to decode the signals of an incremental encoder, and to evaluate data on an axle's speed and position. An optional index signal can be used to reset the counter of the incremental encoder.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	✓	–	–

Characteristics

The incremental encoder interface of the Digital I/O Unit Type 3 supports up to 4 encoders, each with the signals PHI0, PHI90 and the optional index signal IDX.

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

Each encoder interface can react to an index pulse. You can set the index mode to:

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

Note

The index signal must have a length of at least 0.25 lines. If it is shorter, it will not be detected.

You can use the **GateMode** parameter to specify whether the Index signal is gated to the PHI0 and PHI90 signals. That means, the Index signal is only interpreted, if PHI0, PHI90 and IDX are high at the same time.

For the encoder configuration, you can specify various positions:

Position Parameter	Description
Start position	Specifies the position that is written to the position counter when the specified encoder is initialized.

Position Parameter	Description
Index position	Specifies the position that is written to the position counter when an index signal pulse is detected. If the index signal is not used, this parameter is ignored.
Minimum position count value	Specifies the minimum value for the position count range of the specified encoder.
Maximum position count value	Specifies the maximum value for the position count range of the specified encoder.

To adapt the length of the position counter to the resolution of the connected encoder, you can specify the minimum and maximum position count values. If the position counter's value reaches the maximum position count value, the minimum position count value is written to the position counter with the next increment. If the position counter's value reaches the minimum position count value, the maximum position count value is written to the position counter with the next decrement.

Note

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

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

Noise filter sample rate

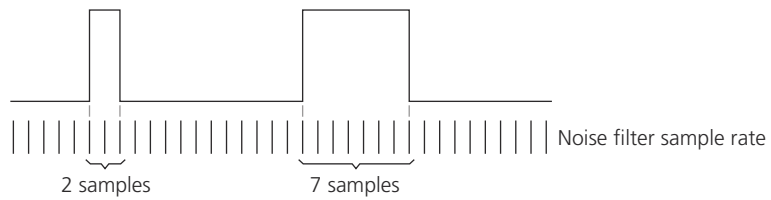
With the noise filter sample rate, you can specify the minimum pulse width of a signal to be used for processing. A signal is ignored if its pulse width is less than the time required for four samples.

Pulses that have a pulse width lower than four times of the period of the specified noise filter sample rate are ignored.

$$\text{PulseWidth_Min} = 4 \cdot 1 / \text{NoiseFilterSampleRate}$$

Noise Filter Sample Rate	Minimum Pulse Width
312.5 kHz	12.8 µs (= 4 · 3.2 µs)
625.0 kHz	6.4 µs
1.25 MHz	3.2 µs
2.5 MHz	1.6 µs
5 MHz	800 ns
10 MHz	400 ns
20 MHz	200 ns

In the figure below, the first signal is ignored because its pulse width is less than four samples. If you specify a higher sample rate, the number of samples increases. If there are four or more samples during the first pulse, it will be used for processing.



The noise filter sample rate can be separately specified for each encoder.

Channel usage

You must specify the first channel for the incremental encoder. Two further channels are allocated automatically for the second encoder signal and the index pulse, even if the index pulse is not used.

The channel usage for the incremental encoder interface is:

Channel Used	Meaning
1st channel	1st encoder input channel (signal PHI0)
2nd channel	2nd encoder input channel (signal PHI90)
3rd channel	Used for the index signal if index mode is set to "Reset counter once" or "Reset counter continuously" (signal IDX).

Tip

Some encoder manufacturers use the terms A, B and Z instead of PHI0, PHI90 and IDX.

I/O mapping

The following table shows the mapping of the logical encoder channel numbers, as used in RTLib, to the related I/O pins of the MicroAutoBox II I/O connector:

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	V2
	2	DigP 1 ch 2	U2
	3	DigP 1 ch 3	T2
	4	DigP 1 ch 4	S2
	5	DigP 1 ch 5	R2
	6	DigP 1 ch 6	P2
	7	DigP 1 ch 7	N2
	8	DigP 1 ch 8	M2
	9	DigP 1 ch 9	V3
	10	DigP 1 ch 10	U3
	11	DigP 1 ch 11	T3
	12	DigP 1 ch 12	S3
	13	DigP 1 ch 13	R3
	14	DigP 1 ch 14	P3
	15	DigP 1 ch 15	N3
	16	DigP 1 ch 16	M3
2	1	DigP 2 ch 1	V4
	2	DigP 2 ch 2	U4
	3	DigP 2 ch 3	T4
	4	DigP 2 ch 4	S4
	5	DigP 2 ch 5	R4
	6	DigP 2 ch 6	P4
	7	DigP 2 ch 7	N4
	8	DigP 2 ch 8	M4
	9	DigP 2 ch 9	V5
	10	DigP 2 ch 10	U5
	11	DigP 2 ch 11	T5
	12	DigP 2 ch 12	S5
	13	DigP 2 ch 13	R5
	14	DigP 2 ch 14	P5
	15	DigP 2 ch 15	N5
	16	DigP 2 ch 16	M5

Port	Channel	Signal	I/O Connector Pin
3	1	DigP 3 ch 1	V6
	2	DigP 3 ch 2	U6
	3	DigP 3 ch 3	T6
	4	DigP 3 ch 4	S6
	5	DigP 3 ch 5	R6
	6	DigP 3 ch 6	P6
	7	DigP 3 ch 7	N6
	8	DigP 3 ch 8	M6

¹⁾ DigP <Port number> ch <Channel number>

Note

- Access to the same input channels by other DIO Type 3 blocks or functions is not allowed.
- You have to specify only the first channel to be used for an encoder, the two subsequent channels are automatically assigned. You can therefore specify up to channel 14 on ports 1 and 2, and up to channel 6 on port 3 for the first channel.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(79de0df6c6ddd2d4eb74f1cc5f48ec50_img.jpg\)](#))
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(d4c9768318b38eff1042b07478e20b4c_img.jpg\)](#))

Related topics

Basics

Basics on the Incremental Encoder Interface.....	229
Overview of the Incremental Encoder Interfaces.....	228

References

[DIO_TYPE3_ENC_BLx \(MicroAutoBox II RTI Reference !\[\]\(4fe57c3593bf1b21d272ae7ac8dfaf77_img.jpg\)](#))
[DIO_TYPE3_ENC_POS_SET_BLx \(MicroAutoBox II RTI Reference !\[\]\(67b4b7a7e28d2fb85c0437cda45ea068_img.jpg\)](#))
[Incremental Encoder on the DIO Type 3 \(MicroAutoBox II RTLib Reference !\[\]\(5f992809aed8ba67de57ee25efabc58b_img.jpg\)](#))

Incremental Encoder Interface on the DIO Type 4 Unit

Introduction

The incremental encoder interface can be used to decode the signals of an incremental encoder, and to evaluate data on an axle's speed and position. An optional index signal can be used to reset the counter of the incremental encoder.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Characteristics

The incremental encoder interface of the Digital I/O Unit Type 4 supports up to 4 encoders, each with the signals PHI0, PHI90 and the optional index signal IDX.

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

Each encoder interface can react to an index pulse. You can set the index mode to:

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

Note

The index signal must have a length of at least 0.25 lines. If it is shorter, it will not be detected.

You can use the **GateMode** parameter to specify whether the Index signal is gated to the PHI0 and PHI90 signals. That means, the Index signal is only interpreted, if PHI0, PHI90 and IDX are high at the same time.

For the encoder configuration, you can specify various positions:

Position Parameter	Description
Start position	Specifies the position that is written to the position counter when the specified encoder is initialized.

Position Parameter	Description
Index position	Specifies the position that is written to the position counter when an index signal pulse is detected. If the index signal is not used, this parameter is ignored.
Minimum position count value	Specifies the minimum value for the position count range of the specified encoder.
Maximum position count value	Specifies the maximum value for the position count range of the specified encoder.

To adapt the length of the position counter to the resolution of the connected encoder, you can specify the minimum and maximum position count values. If the position counter's value reaches the maximum position count value, the minimum position count value is written to the position counter with the next increment. If the position counter's value reaches the minimum position count value, the maximum position count value is written to the position counter with the next decrement.

Note

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

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

Noise filter sample rate

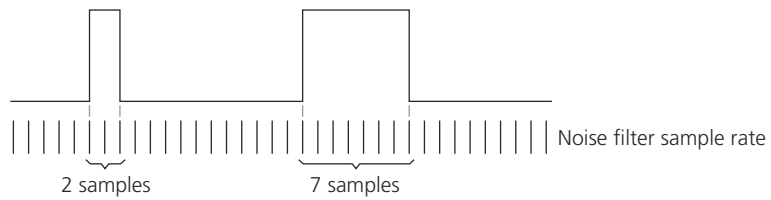
With the noise filter sample rate, you can specify the minimum pulse width of a signal to be used for processing. A signal is ignored if its pulse width is less than the time required for four samples.

Pulses that have a pulse width lower than four times of the period of the specified noise filter sample rate are ignored.

$$\text{PulseWidth_Min} = 4 \cdot 1 / \text{NoiseFilterSampleRate}$$

Noise Filter Sample Rate	Minimum Pulse Width
312.5 kHz	12.8 µs (= 4 · 3.2 µs)
625.0 kHz	6.4 µs
1.25 MHz	3.2 µs
2.5 MHz	1.6 µs
5 MHz	800 ns
10 MHz	400 ns
20 MHz	200 ns

In the figure below, the first signal is ignored because its pulse width is less than four samples. If you specify a higher sample rate, the number of samples increases. If there are four or more samples during the first pulse, it will be used for processing.



The noise filter sample rate can be separately specified for each encoder.

Channel usage

You must specify the first channel for the incremental encoder. Two further channels are allocated automatically for the second encoder signal and the index pulse, even if the index pulse is not used.

The channel usage for the incremental encoder interface is:

Channel Used	Meaning
1st channel	1st encoder input channel (signal PHI0)
2nd channel	2nd encoder input channel (signal PHI90)
3rd channel	Used for the index signal if index mode is set to "Reset counter once" or "Reset counter continuously" (signal IDX).

Tip

Some encoder manufacturers use the terms A, B and Z instead of PHI0, PHI90 and IDX.

I/O mapping

The following table shows the mapping of the logical encoder channel numbers, as used in RTLlib, to the related I/O pins of the MicroAutoBox II I/O connector:

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	M2
	2	DigP 1 ch 2	L2
	3	DigP 1 ch 3	K2
	4	DigP 1 ch 4	J2
	5	DigP 1 ch 5	H2
	6	DigP 1 ch 6	M3
	7	DigP 1 ch 7	L3
	8	DigP 1 ch 8	K3
	9	DigP 1 ch 9	J3
	10	DigP 1 ch 10	H3
	11	DigP 1 ch 11	M4
	12	DigP 1 ch 12	L4
	13	DigP 1 ch 13	K4
	14	DigP 1 ch 14	J4
	15	DigP 1 ch 15	H4
	16	DigP 1 ch 16	M5
2	1	DigP 2 ch 1	L5
	2	DigP 2 ch 2	K5
	3	DigP 2 ch 3	J5
	4	DigP 2 ch 4	H5
	5	DigP 2 ch 5	M6
	6	DigP 2 ch 6	L6
	7	DigP 2 ch 7	K6
	8	DigP 2 ch 8	J6

¹⁾ DigP <Port number> ch <Channel number>

Note

- Access to the same input channels by other DIO Type 4 blocks or functions is not allowed.
- You have to specify only the first channel to be used for an encoder, the two subsequent channels are automatically assigned. You can therefore specify up to channel 14 on port 1, and up to channel 6 on port 2 for the first channel.

For a complete overview on the pinout, refer to:




- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(5774573cf757c446bb08af21f46b2969_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(a502cb21d600ba28a5cdf414d68eef89_img.jpg\)](#))

Related topics

Basics

Basics on the Incremental Encoder Interface.....	229
Overview of the Incremental Encoder Interfaces.....	228

References

DIO_TYPE4_ENC_BLx (MicroAutoBox II RTI Reference )
DIO_TYPE4_ENC_POS_SET_BLx (MicroAutoBox II RTI Reference )
Incremental Encoder on the DIO Type 4 (MicroAutoBox II RTLib Reference )

CAN Support

Introduction

The following topics provide all the information required for working with dSPACE CAN boards.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	✓	✓	✓	–

Where to go from here

Information in this section

[Setting Up a CAN Controller.....](#) 243

Explains how to set up a CAN controller to use a dSPACE board with CAN bus interface.

[Using the RTI CAN MultiMessage Blockset.....](#) 258

Provides information on using the RTI CAN MultiMessage Blockset.

[CAN Signal Mapping.....](#) 281

Lists the CAN signals of the dSPACE real-time hardware and the mapping of these signals to RTI blocks and RTLib functions.

Information in other sections

[Limited Number of CAN Messages.....](#) 369

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..... 244](#)

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

[CAN Transceiver Types..... 245](#)

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

[MicroAutoBox II: Selecting the CAN Controller Frequency..... 249](#)

Depending on the CAN module version of your MicroAutoBox II, MicroAutoBox II supports different maximum CAN controller clock frequencies.

[Defining CAN Messages..... 250](#)

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

[Implementing a CAN Interrupt..... 251](#)

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..... 252](#)

RTI CAN Blockset provides two concepts for receiving CAN messages.

[Removing a CAN Controller \(Go Bus Off\)..... 254](#)

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

[Getting CAN Status Information..... 254](#)

You can use the Error Management Logic (EML) of a CAN controller to get error and status information on the CAN bus and the controller.

[CAN Partial Networking..... 256](#)

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

Initializing the CAN Controller

Introduction

The CAN controller performs serial communication according to the CAN protocol. You can take control of or communicate with other members of a CAN bus via the controller. This means you must configure the CAN controller — called the CAN channel — according to the application.

Standard configuration

You must specify the baud rate for the CAN application and the sample mode:

Sample Mode	Description
1-sample mode	(supported by all dSPACE CAN boards) The controller samples a bit once to determine if it is dominant or recessive.
3-sample mode	(supported by the DS4302 only) The controller samples a bit three times and uses the majority to determine if it is dominant or recessive.

The required bit timing parameters are automatically calculated by the dSPACE CAN software.

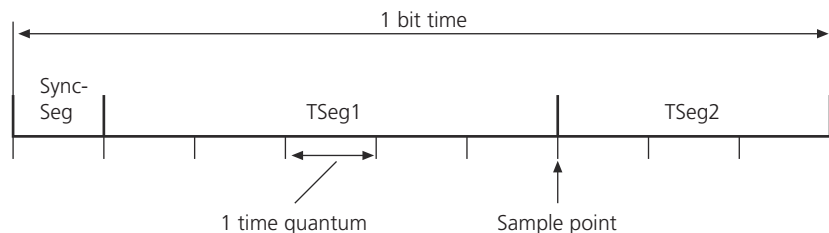
Advanced configuration (bit timing parameters)

The bits of a CAN message are transmitted in consecutive bit times. According to the CAN specification, a bit time consists of two programmable time segments and a synchronization segment:

TSeg1 Timing segment 1. The time before the sample point.

TSeg2 Timing segment 2. The time after the sample point.

SyncSeg Used to synchronize the various bus members (nodes).



The following parameters are also part of the advanced configuration:

SP Sample point. Defines the point in time at which the bus voltage level (CAN-H, CAN-L) is read and interpreted as a bit value.

SJW Synchronization jump width. Defines how far the CAN controller can shift the location of the sample point to synchronize itself to the other bus members.

BRP Baud rate prescaler value. The BRP defines the length of one time quantum.

SMPL Sample mode. Either 1-sample or 3-sample mode. Applicable to the DS4302 only.

Except for the SyncSeg parameter, you must define all these parameters via the values of the bit timing registers (BTR0, BTR1), located on the CAN controller.

Note

Setting up bit timing parameters requires advanced knowledge of the CAN controller hardware and the CAN bus hardware.

RTI support

You initialize a CAN controller with the RTICAN CONTROLLER SETUP block.

Refer to [RTICAN CONTROLLER SETUP \(RTI CAN Blockset Reference !\[\]\(830769b31eeeaca920791081939ff8ba_img.jpg\)](#)).

Related topics

References

[RTICAN CONTROLLER SETUP \(RTI CAN Blockset Reference !\[\]\(6bb0e4f14c4133b37d2887cb37e67ddd_img.jpg\)](#))

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"> DS2202 DS2210 DS2211 	ISO11898
DS4302	<p>The following transceiver types are supported:</p> <ul style="list-style-type: none"> ISO11898 RS485 C252 Piggyback¹⁾ <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p>Note</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"> ISO11898 ISO11898-6^{2), 3)}
MicroLabBox	<p>The following transceiver types are supported:</p> <ul style="list-style-type: none"> ISO11898 ISO11898-6²⁾

¹⁾ If none of the above transceivers matches your application or if a TJA1041 transceiver is used, "piggyback" must be selected as the transceiver type.

²⁾ Selecting the ISO11898-6 transceiver type is required to perform partial networking.

³⁾ Supported only by MicroAutoBox II with DS1513 I/O board.

ISO11898 transceiver

ISO11898 defines a high-speed CAN bus that supports baud rates of up to 1 MBd. This is the most commonly used transceiver, especially for the engine management electronics in automobiles.

CAN-H, CAN-L ISO11898 defines two voltage levels:

Level	Description
CAN-H	High if the bit is dominant (3.5 V), floating (2.5 V) if the bit is recessive.
CAN-L	Low if the bit is dominant (1.0 V), floating (2.5 V) if the bit is recessive.

Termination To terminate the CAN bus lines, ISO11898 requires a 120-Ω resistor at both ends of the bus.

ISO11898-6 transceiver

High-speed transceiver that supports partial networking.

Termination To terminate the CAN bus lines, ISO11898-6 requires a 120-Ω resistor at both ends of the bus.

Note

There are some limitations when you use the optional ISO11898-6 transceiver:

- No wake-up interrupt is implemented.
- Partial networking is supported only for the following baud rates:
 - 125 kbit/s
 - 250 kbit/s
 - 500 kbit/s
 - 1000 kbit/s

Other baud rates can be used for normal CAN operation, but detecting wake-up messages for partial networking is supported only for the baud rates listed above.

- You have to enable Automatic Wake Up on the Parameters Page (RTI<xxx>_ISO11898_6_SST) before you build the model. You cannot enable automatic wake-up during run time.
- If the transceiver is in *power on / listen only* mode, the CAN controller does not send an acknowledge message to the transmitter. The transmitter therefore continues to send the message until it receives the acknowledge signal. This might result in a task overrun if an RX interrupt is configured for the CAN controller.
- If the transceiver is in *power on / listen only* mode, it is not able to send CAN messages. Automatic wake-up is not possible if the transceiver is in *power on / listen only* mode. Because no message is sent on the CAN bus by the transceiver in *power on / listen only* mode, CAN arbitration fails. The CAN controller changes to the BUS OFF state. It is not possible to set the BUS state automatically to BUS ON via an interrupt, because the reason for the BUS OFF state still remains. You must set the CAN controller to BUS ON after you have switched the transceiver state to normal, standby, or sleep mode.

RS485 transceiver

The RS485 transceiver supports baud rates of up to 500 k Bd. 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 k Bd. 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 Ω resistor for many connected bus members, or a 1.6 k Ω 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 !\[\]\(17acf1afa8cdf0b67c53d4865a5ed469_img.jpg\)](#)).

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 !\[\]\(c8d96c8885d3000a912c2582004aed63_img.jpg\)\)](#).

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 !\[\]\(f15d3c54be60b4fd0ce1da9fb3f67256_img.jpg\)\)](#) 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.

MicroAutoBox II: Selecting the CAN Controller Frequency

Introduction

Depending on the CAN module version of your MicroAutoBox II, MicroAutoBox II supports the following maximum CAN controller clock frequency:

Module Version	Frequency
Up to 2.0	24 MHz
2.0 and higher	36 MHz
2.1 and higher	64 MHz

To get the CAN module version of your MicroAutoBox II, use the dSPACE experiment software.

Highest possible frequency automatically selected

The real-time application built with RTI CAN Blockset is compatible with both module versions and frequencies: During board initialization, the highest frequency that is available for the controller is automatically selected, together with the corresponding bit timing values. This applies regardless of the frequency you select in the Advanced Configuration dialog. Refer to [Advanced](#)

[Configuration Dialog \(RTICAN CONTROLLER SETUP\) \(RTI CAN Blockset Reference !\[\]\(529949c2c3dadbaa4e538e8c643454bc_img.jpg\)\).](#)

Related topics

References

[Advanced Configuration Dialog \(RTICAN CONTROLLER SETUP\) \(RTI CAN Blockset Reference !\[\]\(0f848bbd71cef6b345273b16f905912a_img.jpg\)\).](#)

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> ¹⁾ . An RQ message is transmitted with a specific identifier to request data. An RQ message does not contain data.
Remote (RM)	Second part of a <i>remote transmission request</i> ¹⁾ . 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.

¹⁾ 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 !\[\]\(f1c5da15572e3e09d343161be98f508d_img.jpg\)\).](#)

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)	RTICAN Transmit (TX) (RTI CAN Blockset Reference )
Receive (RX)	RTICAN Receive (RX) (RTI CAN Blockset Reference )
Request (RQ)	RTICAN Request (RQ) (RTI CAN Blockset Reference )
Remote (RM)	RTICAN Remote (RM) (RTI CAN Blockset Reference )

Related topics**Basics**

[Configuring CAN Messages via Data Files \(RTI CAN Blockset Reference !\[\]\(870f5d5e9c0d57485634be3ecf52f3ca_img.jpg\)](#))

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 !\[\]\(b64b40baaee5acddc1eab8538ba84754_img.jpg\)](#)).

Related topics**References**[RTICAN Interrupt \(RTI CAN Blockset Reference !\[\]\(dfbd6b3763a6d1d9afaa974f64e2e4b5_img.jpg\)](#))

Using RX Service Support

Concepts for receiving CAN messages

When CAN messages are received, RX blocks access the DPMEM between the master processor and the slave processor.

RTI CAN Blockset provides two concepts for receiving CAN messages:

- Common receive concept
- RX service receive concept

Common receive concept

According to the common receive concept, one data object is created in the DPMEM for each received CAN message. Due to the limited DPMEM size, the number of RX blocks you can use in a model is limited to 100 (200 for the DS4302).


RX service receive concept

When you enable RX service support, one data object is created in the DPMEM for all received CAN messages, and memory on the master processor is used to receive CAN messages. The RX service fills this memory with new CAN data. This concept improves run-time performance.

Tip

In contrast to the common receive concept, the number of RX blocks for which RX service support is enabled is unlimited.

Specifying a message filter When you use RX service, you have to specify a filter to select the messages to receive via RX service. To define the filter, you have to set up a bitmap that represents the message. Each bit position can be assigned 0 (must be matched), 1 (must be matched), or X (don't care). A message is received via RX service only if it matches the bitmap.

You can define the message filter on the [RX Service Page \(RTICAN CONTROLLER SETUP\)](#) (RTI CAN Blockset Reference ).

Specifying the queue size When you use RX service, you have to specify the maximum number of CAN messages that you expect to receive in a sample step. The memory allocated on the master processor used to queue CAN messages is calculated from the specified maximum number of CAN messages.

Note

If more CAN messages than the specified Queue size are received in a sample step, the oldest CAN messages are lost. You should therefore specify the queue size so that no CAN messages are lost.

Example:

A CAN controller is configured to use the baud rate 500 kBd. The slowest RX block assigned to this CAN controller is sampled every 10 ms. At the specified baud rate, a maximum of about 46 CAN messages (STD format) might be received during two consecutive sample steps. To ensure that no CAN message is lost, set the queue size to 46.


Triggering an interrupt when a message is received via RX service You can let an interrupt be triggered when a message is received via RX service.

Note


You cannot let an interrupt be triggered when a message *with a specific ID* is received. An interrupt is triggered each time a message is received via RX service.

You can define the interrupt on the [Unit Page \(RTI CAN Interrupt\)](#) (RTI CAN Blockset Reference .

Precondition for gatewaying messages Enabling the RX service is a precondition for *gatewaying messages* between CAN controllers.

Refer to [Gatewaying Messages Between CAN Buses](#) (RTI CAN Blockset Reference .



Precondition for the TX loop back feature RX service allows you to use the *TX loop back feature*. The feature lets you observe whether message transfer over the bus was successful.

You can enable TX loop back on the [Options Page \(RTICAN Transmit \(TX\)\)](#) (RTI CAN Blockset Reference .

Enabling RX service support

You have to enable RX service support for each CAN controller and for each RX block.

RTI support

- For a CAN controller, you enable the RX service on the RX Service page of the RTICAN CONTROLLER SETUP block. Refer to [RX Service Page \(RTICAN CONTROLLER SETUP\)](#) (RTI CAN Blockset Reference .
- For an RX block, you enable the RX service on the Options page of the RTICAN Receive (RX) block of the RTICAN CONTROLLER. Refer to [Options Page \(RTICAN Receive \(RX\)\)](#) (RTI CAN Blockset Reference .

Related topics**Basics**

[Gatewaying Messages Between CAN Buses \(RTI CAN Blockset Reference !\[\]\(eafc244b53721dd1ec133f0772f70fc7_img.jpg\)](#))

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 !\[\]\(c33cb967c8fc4f5e27188a389b621c8e_img.jpg\)](#)).
- To restart data transfer, use the RTICAN Bus Off Recovery block. Refer to [RTICAN Bus Off Recovery \(RTI CAN Blockset Reference !\[\]\(38e1383487ca0f0e9e2c9378b9dbcae7_img.jpg\)](#)).

Related topics**References**

[RTICAN Bus Off Recovery \(RTI CAN Blockset Reference !\[\]\(ab4e2b3fc7e7887b7a72f548aa6f5e60_img.jpg\)](#))
[RTICAN Go Bus Off \(RTI CAN Blockset Reference !\[\]\(0a20d1259d5ab849a22cc9906b421113_img.jpg\)](#))

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 ≥ 96 .
At least one counter value ≥ 128	Error passive	The CAN controller is still active. The CAN controller can recover from this state itself.
Transmit Error counter value ≥ 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.
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 Message fields (RTI CAN Blockset Reference)).
Lost RX messages	Each time a message cannot be stored in the buffer of the CAN controller, the message is lost and an <i>RX lost error</i> is detected.
Successfully received RX messages	Each time an RX message is received successfully, the appropriate counter is incremented.
Successfully sent TX messages	Each time a TX message is sent successfully, the appropriate counter is incremented.
(DS4302 only) Status of fault tolerant receiver	The error state of the fault tolerant receiver is reported.
(DS4302 only) Fault tolerant transceiver	The value of the output is increased if a CAN bus events occurs.

RTI support

To get status information, use the RTICAN Status block. Refer to [RTICAN Status \(RTI CAN Blockset Reference\)](#).

Related topics

References

[CAN Service Functions \(MicroAutoBox II RTLib Reference !\[\]\(3d8c13c92b853674f749aac6fa869926_img.jpg\)\)](#)
[RTICAN Status \(RTI CAN Blockset Reference !\[\]\(ce455c990c00145a2dda1d9a310cb682_img.jpg\)\)](#)

CAN Partial Networking

Introduction

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

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

- MicroAutoBox II equipped with the DS1513 I/O board
- MicroLabBox

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

- Filtering message IDs: You can define a message filter to select the messages to use as wake-up messages. The filter uses a bitmask which represents the message. A message passes the filter and is used as wake-up message only if it matches the bitmask.
- Filtering message data: You can mask the data bytes of incoming wake-up messages to determine whether they are valid wake-up messages.

Switching the CAN transceiver to sleep mode The CAN transceiver of the dSPACE real-time hardware is switched to sleep mode via the real-time application.

Tip


(MicroAutoBox II only) You can stop and power down the MicroAutoBox II with the DS1401_POWER_DOWN block from the DS1401 MicroAutoBox II Base Board II library. To set MicroAutoBox II to sleep mode, KL15 must be disconnected from the power supply. For further information, refer to [DS1401_POWER_DOWN \(MicroAutoBox II RTI Reference !\[\]\(e9474ce1d70442456f8fe9c393ea149c_img.jpg\)\)](#). This is not possible for MicroLabBox.

Waking up dSPACE real-time hardware You can use partial networking messages to wake up dSPACE real-time hardware after its CAN transceiver is switched to sleep mode.

- (Relevant for MicroAutoBox II only) If a MicroAutoBox II's CAN transceiver is woken up via a partial networking message, the MicroAutoBox II behaves as if

it was powered up manually. Depending on where the real-time application is loaded (flash memory or RAM), the MicroAutoBox II starts the application or waits for further input.

- (Relevant for MicroLabBox only) Unlike MicroAutoBox II, MicroLabBox cannot be powered down and then woken up via partial networking messages. However, the CAN transceiver of MicroLabBox can be set to sleep mode, and then woken up via partial networking messages later on.

RTI support Refer to [Partial Networking Page \(RTICAN CONTROLLER SETUP\)](#) (RTI CAN Blockset Reference ).

Related topics

References

[Partial Networking Page \(RTICAN CONTROLLER SETUP\)](#) (RTI CAN Blockset Reference )

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..... 258](#)

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

[Basics on Working with CAN FD..... 263](#)

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..... 268](#)

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..... 274](#)

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

[Manipulating Signals to be Transmitted..... 277](#)

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 263.

Support of AUTOSAR features

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

Manipulating signals to be transmitted

The RTI CAN MultiMessage Blockset provides several options to manipulate the values of signals before they are transmitted. You can switch between them with the Bus Navigator in ControlDesk via entries in the generated TRC file. In

addition, you can calculate checksum, parity, toggle, counter, and mode counter values.

Updating a model

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

Tip

You do not have to recreate the S-function for the RTI CAN MultiMessage Blockset if you switch from one processor board to another, e.g., from a DS1006 to a DS1007, and vice versa.

When you switch to or from a MicroAutoBox II or MicroLabBox, you only have to recreate the ControllerSetup block. You also have to recreate the ControllerSetup block if you change the controller name.

Modifying model parameters during run time



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

(Relevant only for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board) For information on where to find the signals of the CAN bus in the TRC file, refer to [Available TRC File Variable Entries and Their Locations in the TRC File \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(aa53ad6fea213b8b2226d3077e30533a_img.jpg\)](#)).

User-defined variables

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

Working with variants of CAN communication

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

Gatewaying messages between CAN buses

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

Online modification of gateway exclude list	You can modify the exclude list of RTICANMM Gateways during run time, i.e., specify messages not to be gatewayed.
Dynamic message triggering	You can modify the cycle times and initiate a spontaneous transmission (interactive or model-based) during run time.
Defining free raw messages	<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"> ▪ Trigger options ▪ Ports and displays ▪ Message ID and length adjustable during run time <p>The following features are not supported:</p> <ul style="list-style-type: none"> ▪ Checksum generation ▪ Custom signal manipulation
Capturing messages	<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>
CAN partial networking	<p>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.</p> <div data-bbox="592 1344 659 1373"> Note </div> <div data-bbox="598 1396 1398 1686"> <p>Partial networking is possible for the following dSPACE real-time hardware:</p> <ul style="list-style-type: none"> ▪ 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: <ul style="list-style-type: none"> ▪ PHS-bus-based systems (DS1006 or DS1007 modular systems) with DS4505 Interface Board ▪ MicroAutoBox II variants with DS1507 ▪ MicroAutoBox II variants with DS1514 </div> <p>The RTI CAN MultiMessage Blockset lets you specify the CAN partial networking wake-up messages by filtering message IDs and message data.</p>

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

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

For more information, refer to [Partial Networking Page \(RTICANMM ControllerSetup\)](#) (RTI CAN MultiMessage Blockset Reference .

TRC file entries with initial data

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

Visualization with the Bus Navigator

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

RTI CAN Blockset and RTI CAN MultiMessage Blockset

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

Further information on the RTI CAN MultiMessage Blockset

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

- [RTI CAN MultiMessage Blockset Reference](#) 

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

- [RTI CAN MultiMessage Blockset Tutorial](#) 

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

Basics on Working with CAN FD

Introduction

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

Basics on CAN FD

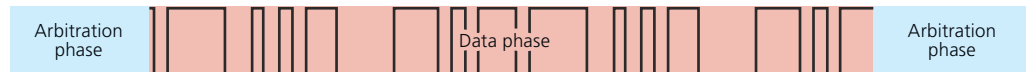
CAN FD stands for *CAN with Flexible Data Rate*. The CAN FD protocol is based on the CAN protocol as specified in ISO 11898-1. Compared with the classic CAN protocol, CAN FD comes with an increased bandwidth for the serial communication. The improvement is based on two factors:

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

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

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

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

Classic CAN message**CAN FD message using a higher bit rate****CAN FD message with longer payload using a higher bit rate****CAN FD protocols**

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

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

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

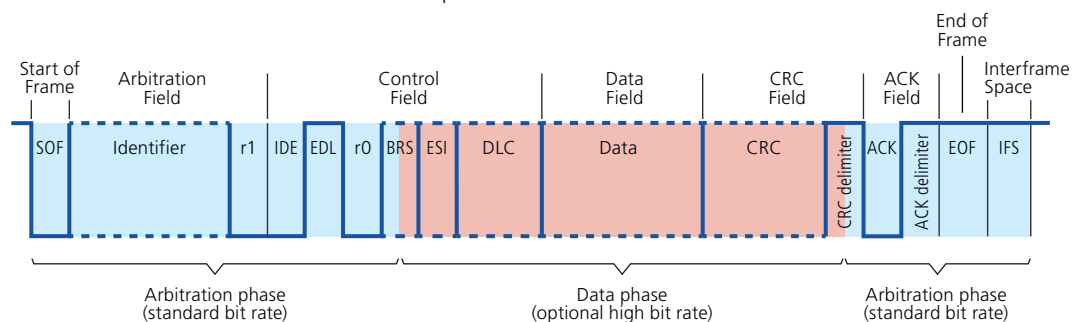
The RTI CAN MultiMessage Blockset supports both CAN FD protocols.

CAN FD message characteristics

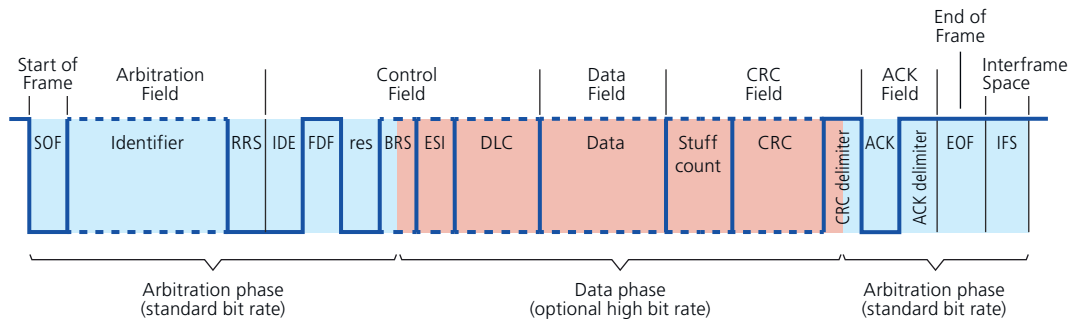
In principle, CAN FD messages and CAN messages consist of the same elements and have the same message structure.

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

- Non-ISO CAN FD protocol:



■ ISO CAN FD protocol:



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

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

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

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

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

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



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

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

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

Activating CAN FD mode in the RTI CAN MultiMessage Blockset

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

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

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

Supported database file types

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

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

CanFrameTxBehavior and CanFrameRxBehavior attributes In AUTOSAR and FIBEX files, the `CanFrameTxBehavior` and/or `CanFrameRxBehavior`

attributes of a message can be defined to specify whether the message is to be treated as a CAN FD message or classic CAN 2.0 message. The RTI CAN MultiMessage Blockset evaluates the attribute as follows:

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

Supported dSPACE platforms

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

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

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

- PHS-bus-based system with DS4505: [DS4342 Connections in Different Topologies \(PHS Bus System Hardware Reference !\[\]\(67ff022fd78f943b679992c2874bbfd1_img.jpg\)](#))
- MicroAutoBox II: [DS4342 Connections in Different Topologies \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(042ea11c58a77088d3dd7150909adec0_img.jpg\)](#))

Working with CAN messages and CAN FD messages

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

Related topics

References

[Setup Page \(RTICANMM ControllerSetup\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(0d5ec72f61334709c3fc9450209b754f_img.jpg\)](#))

Basics on Working with a J1939-Compliant DBC File

Introduction

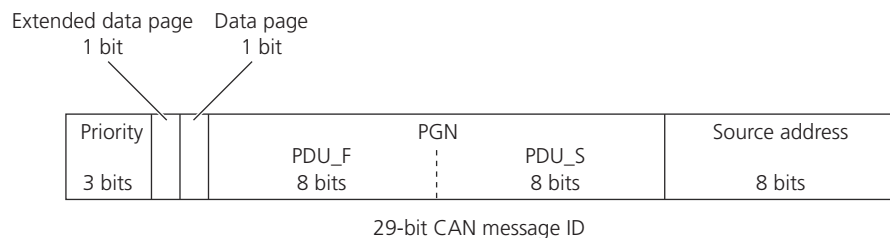
J1939 is a vehicle CAN bus standard defined by the Society of Automotive Engineers (SAE). It is used for communication in heavy-duty vehicles, for example, for communication between a tractor and its trailer.

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

Broadcast and peer-to-peer communication

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

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



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

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

The first 8 bits of the PGN represent the *PDU_F* (Protocol Data Unit format). The *PDU_F* value specifies the communication mode of the message (peer-to-peer or broadcast). The interpretation of the *PDU_S* value (PDU-specific) depends on the *PDU_F* value. For messages with a *PDU_F* < 240 (peer-to-peer communication, also called PDU1 messages), *PDU_S* is not relevant for the PGN, but contains the destination address of the network node that receives the message. For

messages with a $PDU_F \geq 240$ (broadcast messages, also called PDU2 messages), PDU_S specifies the second 8 bits of the PGN and represents the group extension. A group extension is used to increase the number of messages that can be broadcast in the network.

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

Message attributes in J1939-compliant DBC files

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

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

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

Source/destination mapping

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

Tip

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

Container and instance messages

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

Container message A J1939 message defined by its PGN (and Data Page bit). A container can receive all the messages with its PGN in general. If several messages are received in one sampling step, only the last received message is

held in the container. Container messages can be useful, for example, when you configure a gateway with message manipulation.

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

Note

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

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

Network management

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

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

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

The J1939 standard reserves the following addresses:

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

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

Note

The RTI CAN MultiMessage Blockset interprets attributes in the DBC file like this:

- In a DBC file created with CANalyzer 5.1 or earlier, the *name* network node attributes and the *J1939PGSrc* and *J1939PGDest* message attributes are read in. The *J1939PGSrc* attribute is interpreted as the address of the node that sends the message, the *J1939PGDest* attribute is interpreted as the address of the node that receives the message.
- In a DBC file created with CANalyzer 5.2 or later, the *name* and *NMStationAddress* network node attributes are read in. The *NMStationAddress* attribute is interpreted as the network node address.

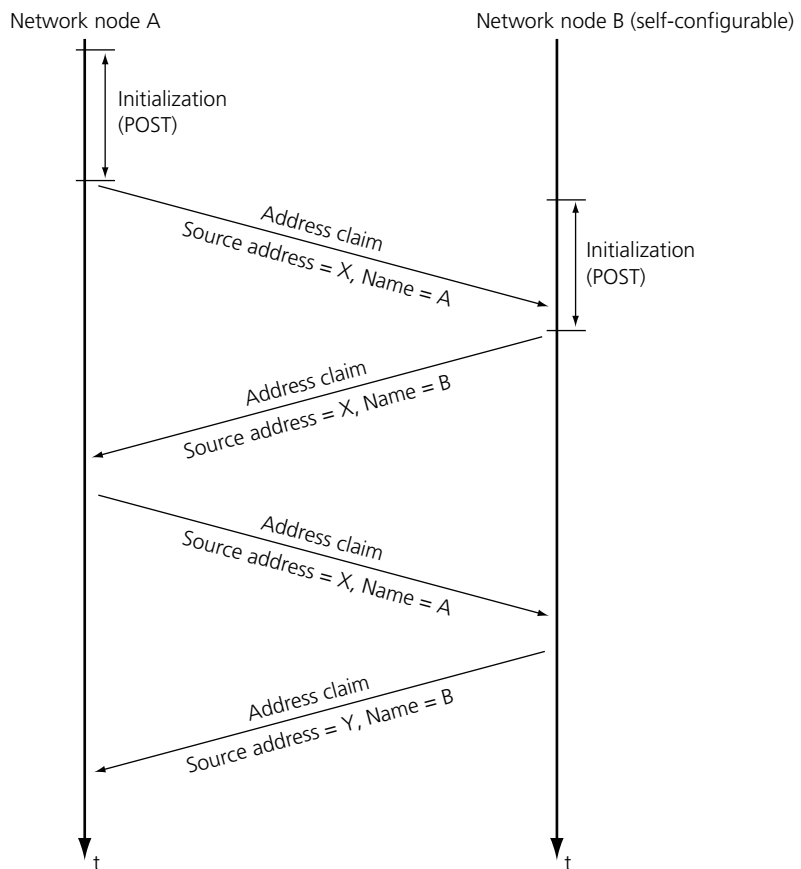
Name The J1939 standard defines a 64-bit *name* to identify each network node. The name indicates the main function of the network node with the associated address and provides information on the manufacturer.

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

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

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

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



The following steps are performed in the address claiming procedure:

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

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

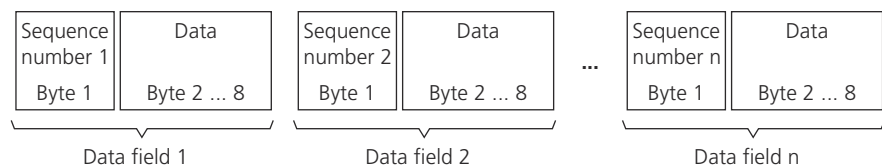
Note

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

Messages > 8 bytes (message packaging)

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

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



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

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

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

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

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

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

Related topics

Basics

[Lesson 13 \(Advanced\): Working with a J1939-Compliant DBC File \(RTI CAN MultiMessage Blockset Tutorial !\[\]\(0f848bbd71cef6b345273b16f905912a_img.jpg\)](#))

Transmitting and Receiving CAN Messages

Introduction

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

Defining CAN communication

To define the CAN communication of a CAN controller, you can specify a DBC, MAT, FIBEX, or AUTOSAR system description file as the database file on the [General Settings Page \(RTICANMM MainBlock\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(6059a5aa8b4ca7bb793408023d6c6e42_img.jpg\)](#)). You can also define CAN communication without using a database file.

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

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

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

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

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

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

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

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

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

Defining RX messages and TX messages

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

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

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


Triggering TX message transmission

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

For details, refer to [Triggering Options Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ).

Triggering reactions to the reception of RX messages


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

For details, refer to [Raw Data Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference ).

Working with raw data

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

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

For details, refer to [Raw Data Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference .

Implementing checksum algorithms

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

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

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

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

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

For details, refer to [Checksum Definition Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference .

Gatewaying messages

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

You can gateway CAN messages in two ways:

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

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

For details, refer to [RTICANMM Gateway](#) (RTI CAN MultiMessage Blockset Reference .

Related topics

References

[General Settings Page \(RTICANMM MainBlock\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)](#))

Manipulating Signals to be Transmitted

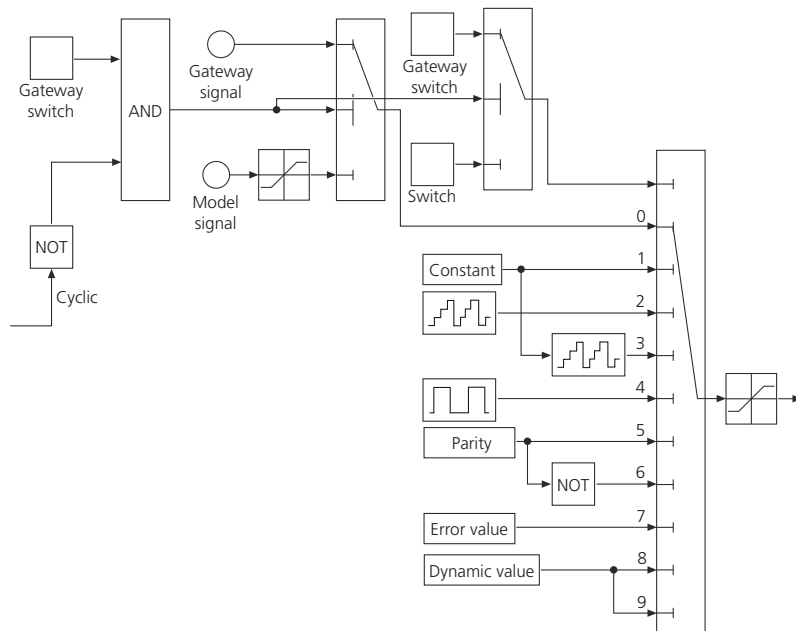
Introduction

All the signals of all the RX and TX messages (see [Defining RX messages and TX messages](#) on page 275) automatically get corresponding entries in the generated TRC file. This allows you to analyze them (signals of RX messages) or change their values (signals of TX messages) with the Bus Navigator in ControlDesk.

Manipulating signals to be transmitted

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

The illustration below visualizes the options.



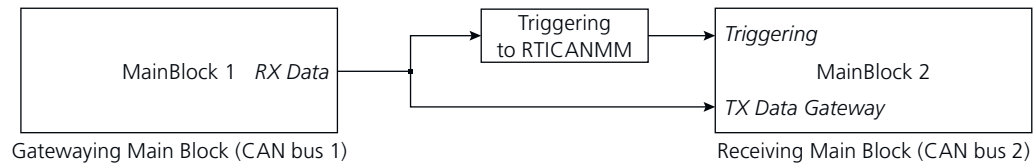
You can switch between these options in ControlDesk.

TX model signal A signal of a TX message whose value can be changed from within the model. By default, the values of TX model signals cannot be changed in ControlDesk. If you also want to manipulate TX model signals from ControlDesk, you have to select them on the [Input Manipulation Page \(RTICANMM MainBlock\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(bd3b31712ad9bab5a241210fa6925cdd_img.jpg\)](#)).

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

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

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



MainBlock1 gatewayes messages and their signals to MainBlock2. Specifying gateway signals for MainBlock2 adds a TX Data Gateway input to it. The specified gateway signals are transmitted via CAN bus 2 with the signal values received from MainBlock1 when triggered by the messages received from MainBlock1. You therefore have to specify triggered message transmission for the messages of the gateway signals on the [Message Cyclic Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference). In addition, you can enable signal value switching for gateway signals during run time, for example, to transmit the signal constant value instead of the gateway value.

Note

Implementing gateway signals at least doubles the number of block inputs and therefore reduces performance. If you want to exchange messages between CAN controllers and do not want to perform signal manipulation, you should use the RTICANMM Gateway block instead.

You can specify gateway signals on the pages located in the [Gateway Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference).

Toggle signal A 1-bit signal that can be used, for example, to indicate whether CAN messages are transmitted. If a CAN message is transmitted, the toggle signal value alternates between 0 and 1. Otherwise, the toggle signal value remains constant.

You can specify toggle signals on the [Toggle Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference).

Parity signal A signal that a parity bit is appended to. You can specify one or more signals of a TX message as parity signals. A parity bit is calculated for the specified signals according to whether even or odd parity is selected. The bit is appended to the signal and the TX message is transmitted with the parity signal.

You can specify parity signals on the [Parity Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).

Counter signal A signal of a TX message that is used to check for correct message transmission or to trigger the transmission of signals in a message.

- **Behavior of counter signals**

The value of a counter signal changes with every message transmission. You can specify the counter start, step, and stop values, etc. Each time the counter reaches the stop value, it turns around to the counter start value.

- **Use of counter signals**

You can specify counter signals to check for correct transmission of a message or trigger the transmission of signals in a message.

- *Checking correct message transmission:* The receiver of a message expects a certain counter signal value. For example, if the transmission of a message was stopped for two transmissions, the expected counter signal value and the real counter signal value can differ, which indicates an error. Counter signals used in this way are often called alive counters.

- *Triggering the transmission of signals:* You can trigger the transmission of signals if you specify a mode signal as a counter signal. The signal value of the mode signal triggers the transmission of mode-dependent signals. Because the counter signal value changes with every message transmission, this triggers the transmission of the mode-dependent signals. By using counter signals in this way, you can work with signals which use the same bytes of a message. Counter signals used in this way are often called mode counters.

- **Using counter signals in ControlDesk**

In ControlDesk, you can transmit the signal's constant, counter, or increment counter value. The increment counter value is the counter value incremented by the signal's constant value.

You can specify counter signals on the [Counter Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).

Error value A static signal value that indicates an error. You can specify an error value for each signal to be transmitted. Alternatively, error values can be defined in a database file. In ControlDesk, you can switch to transmit the signal's error value, constant value, etc. However, you cannot change the error value during run time. In ControlDesk, you can use a Variable Array (MultiState LED value cell type) to indicate if the error value is transmitted.

You can specify error values on the [Signal Errors Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).

Dynamic value A signal value that is transmitted for a defined number of times.

- *Behavior of dynamic values:* You can specify to use a dynamic value for each signal to be transmitted. In ControlDesk, you can specify to transmit the signal's constant value or dynamic value. If you switch to the dynamic value, it is transmitted for a defined number of times (countdown value). Then signal manipulation automatically switches back to the signal manipulation option used before the dynamic value.

- *Example of using dynamic values:* Suppose you specify a dynamic value of 8 and a countdown value of 3. If you switch to the dynamic value of the signal, the signal value 8 is sent the next 3 times the TX message is transmitted. Then the signal manipulation option is reset.

You can specify dynamic values on the pages located in the [Dynamic Signal Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).

CAN Signal Mapping

Introduction

Lists the CAN signals of the dSPACE real-time hardware and the mapping of these signals to RTI blocks and RTLib functions.

CAN Signal Mapping

Introduction

The CAN subsystem of MicroAutoBox II provides CAN interfaces that meet the ISO/DIS 11898 specifications.

I/O mapping

The following tables show the mappings of the logical interface numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector.

▪ MicroAutoBox II 1401/1507

CAN_TP1 Module	Channel/ Controller	Signal	I/O Connector Pin (Sub-D Connector)
1	1	CAN 1 low in/out (CAN-L)	2
		CAN 1 high in/out (CAN-H)	1
		GND	3
	2	CAN 2 low in/out (CAN-L)	5
		CAN 2 high in/out (CAN-H)	4
		GND	6
2	1	CAN 3 low in/out (CAN-L)	41
		CAN 3 high in/out (CAN-H)	40
		GND	42
	2	CAN 4 low in/out (CAN-L)	44
		CAN 4 high in/out (CAN-H)	43
		GND	45

DS4342 CAN FD Module	Channel	Signal	I/O Connector Pin (Sub-D Connector)
1	–	UBAT	35
	1	GND 1	15
		CAN 1 high in/out (CAN-H)	16
		CAN 1 low in/out (CAN-L)	17
		Feed-through line CAN 1 high	38
		Feed-through line CAN 1 low	39
	2	GND 2	18
		CAN 2 high in/out (CAN-H)	19
		CAN 2 low in/out (CAN-L)	20
		Feed-through line CAN 2 high	36
		Feed-through line CAN 2 low	37
2	–	UBAT	74
	1	GND 1	54
		CAN 1 high in/out (CAN-H)	55
		CAN 1 low in/out (CAN-L)	56
		Feed-through line CAN 1 high	77
		Feed-through line CAN 1 low	78
	2	GND 2	57
		CAN 2 high in/out (CAN-H)	58
		CAN 2 low in/out (CAN-L)	59
		Feed-through line CAN 2 high	75
		Feed-through line CAN 2 low	76

▪ MicroAutoBox II 1401/1511, MicroAutoBox II 1401/1511/1514

CAN_TP1 Module	Channel/Controller	Signal	I/O Connector Pin (DS1511 ZIF Connector)
1	1	CAN 1 low in/out (CAN-L)	c3
		CAN 1 high in/out (CAN-H)	c2
		GND	c1
	2	CAN 2 low in/out (CAN-L)	b3
		CAN 2 high in/out (CAN-H)	b2
		GND	b1
2	1	CAN 3 low in/out (CAN-L)	B3
		CAN 3 high in/out (CAN-H)	B2
		GND	B1
	2	CAN 4 low in/out (CAN-L)	A3
		CAN 4 high in/out (CAN-H)	A2
		GND	A1

DS4342 CAN FD Module¹⁾	Channel	Signal	I/O Connector Pin (DS1514 ZIF Connector)
1	–	UBAT	(H5)
	1	GND 1	L3
		CAN 1 high in/out (CAN-H)	M3
		CAN 1 low in/out (CAN-L)	M4
	2	Feed-through line CAN 1 high	(K6)
		Feed-through line CAN 1 low	(K5)
		GND 2	J3
		CAN 2 high in/out (CAN-H)	K3
		CAN 2 low in/out (CAN-L)	K4
		Feed-through line CAN 2 high	(J6)
		Feed-through line CAN 2 low	(J5)
2	–	UBAT	(X5)
	1	GND 1	a3
		CAN 1 high in/out (CAN-H)	b3
		CAN 1 low in/out (CAN-L)	b4
	2	Feed-through line CAN 1 high	(Z6)
		Feed-through line CAN 1 low	(Z5)
		GND 2	Y3
		CAN 2 high in/out (CAN-H)	Z3
		CAN 2 low in/out (CAN-L)	Z4
		Feed-through line CAN 2 high	(Y6)
		Feed-through line CAN 2 low	(Y5)

¹⁾ Not supported by MicroAutoBox II 1401/1511

▪ MicroAutoBox II 1401/1513, MicroAutoBox II 1401/1513/1514

CAN_TP1 Module	Channel/ Controller	Signal	I/O Connector Pin (DS1513 ZIF Connector)
1	1	CAN 1 low in/out (CAN-L)	c3
		CAN 1 high in/out (CAN-H)	c2
		GND	c1
	2	CAN 2 low in/out (CAN-L)	b3
		CAN 2 high in/out (CAN-H)	b2
		GND	b1

CAN_TP1 Module	Channel/ Controller	Signal	I/O Connector Pin (DS1513 ZIF Connector)
2	1	CAN 3 low in/out (CAN-L)	B3
		CAN 3 high in/out (CAN-H)	B2
		GND	B1
	2	CAN 4 low in/out (CAN-L)	A3
		CAN 4 high in/out (CAN-H)	A2
		GND	A1
3	1	CAN 5 low in/out (CAN-L)	P3
		CAN 5 high in/out (CAN-H)	P2
		GND	P4
	2	CAN 6 low in/out (CAN-L)	N3
		CAN 6 high in/out (CAN-H)	N2
		GND	N4

DS4342 CAN FD Module ¹⁾	Channel	Signal	I/O Connector Pin (DS1514 ZIF Connector)
1	–	UBAT	(H5)
	1	GND 1	L3
		CAN 1 high in/out (CAN-H)	M3
		CAN 1 low in/out (CAN-L)	M4
	2	Feed-through line CAN 1 high	(K6)
		Feed-through line CAN 1 low	(K5)
		GND 2	J3
		CAN 2 high in/out (CAN-H)	K3
		CAN 2 low in/out (CAN-L)	K4
		Feed-through line CAN 2 high	(J6)
		Feed-through line CAN 2 low	(J5)
2	–	UBAT	(X5)
	1	GND 1	a3
		CAN 1 high in/out (CAN-H)	b3
		CAN 1 low in/out (CAN-L)	b4
	2	Feed-through line CAN 1 high	(Z6)
		Feed-through line CAN 1 low	(Z5)
		GND 2	Y3
		CAN 2 high in/out (CAN-H)	Z3
		CAN 2 low in/out (CAN-L)	Z4
		Feed-through line CAN 2 high	(Y6)
		Feed-through line CAN 2 low	(Y5)

¹⁾ Not supported by MicroAutoBox II 1401/1513

Related RTI blocks

- RTICAN CONTROLLER SETUP
- RTICANMM ControllerSetup

Related RTLib functions Slave CAN Access Functions

LIN Support

Introduction

The following chapter contains information on the LIN support for MicroAutoBox II.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	✓	✓	✓	–

Characteristics

Depending on the variant you use, MicroAutoBox II is equipped with a different number of CAN_TP1 modules.

MicroAutoBox II Variant	Number of CAN_TP1 Modules
MicroAutoBox II 1401/1507	2
MicroAutoBox II 1401/1511	2
MicroAutoBox II 1401/1511/1514	2
MicroAutoBox II 1401/1513	3
MicroAutoBox II 1401/1513/1514	3

Each CAN_TP1 module provides two channels. The module(s) is/are shared between LIN and serial communication. One channel can be used only for LIN or for serial communication. The channels provide serial communication or LIN as follows:

Channel	Interface	Max. Baudrate
Channel 0	RS232	115.2 kBaud
Channel 1	LIN (K line)	20 kBaud

I/O mapping

The following tables show the mappings of converter and channel numbers, as used in RTI and RTLib, to the I/O pins of the related I/O connector.

▪ MicroAutoBox II 1401/1507

Unit	RTI Channel	RTLib Channel	Signal	I/O Connector Pin (Sub-D Connector)
CAN_TP1 Module1	2	1	Serial 1 TX out	7
			Serial 1 RX in	8
	1	0	Serial 2 LIN (K line)	10
			Serial 2 Not used (L line)	11
CAN_TP1 Module2	2	1	Serial 3 TX out	46
			Serial 3 RX in	47
	1	0	Serial 4 LIN (K line)	49
			Serial 4 Not used (L line)	50

▪ MicroAutoBox II 1401/1511, MicroAutoBox II 1401/1511/1514

Unit	RTI Channel	RTLib Channel	Signal	I/O Connector Pin (DS1511 ZIF Connector)
CAN_TP1 Module1	2	1	Serial 1 TX out	c5
			Serial 1 RX in	c6
	1	0	Serial 2 LIN (K line)	b5
			Serial 2 Not used (L line)	b6
CAN_TP1 Module2	2	1	Serial 3 TX out	B5
			Serial 3 RX in	B6
	1	0	Serial 4 LIN (K line)	A5
			Serial 4 Not used (L line)	A6

▪ MicroAutoBox II 1401/1513, MicroAutoBox II 1401/1513/1514

Unit	RTI Channel	RTLib Channel	Signal	I/O Connector Pin (DS1513 ZIF Connector)
CAN_TP1 Module1	2	1	Serial 1 TX out	c5
			Serial 1 RX in	c6
	1	0	Serial 2 LIN (K line)	b5
			Serial 2 Not used (L line)	b6

Unit	RTI Channel	RTLib Channel	Signal	I/O Connector Pin (DS1513 ZIF Connector)
CAN_TP1 Module2	2	1	Serial 3 TX out	B5
			Serial 3 RX in	B6
	1	0	Serial 4 LIN (K line)	A5
			Serial 4 Not used (L line)	A6
CAN_TP1 Module3	2	1	Serial 5 TX out	P5
			Serial 5 RX in	P6
	1	0	Serial 6 LIN (K line)	N5
			Serial 6 Not used (L line)	N6

RTLib user

For information on the RTLib functions available for LIN, see [LIN Access Functions \(MicroAutoBox II RTLib Reference !\[\]\(666e09182d4cd268646ea700ea60dcdf_img.jpg\)](#)).

Note

When using LIN, RTLib users have to use special firmware modes. For detailed information, refer to [CAN_TP1 Firmware Modes \(MicroAutoBox II RTLib Reference !\[\]\(d66ff64371a51729ac8c1cdaa685ba6f_img.jpg\)](#)).

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

LIN Basics.....	290
Fields of Application.....	294
You can test the LIN network features of an electronic control unit (ECU). Several test scenarios are possible.	
LIN Bus Handling.....	300
Explains how to implement LIN bus communication.	
Using the RTI LIN MultiMessage Blockset.....	311
Provides information on using the RTI LIN MultiMessage Blockset.	

LIN Basics

Where to go from here

Information in this section

[LIN Basics](#).....290

LIN (local interconnect network) is a low-cost serial communication system for distributed electronic systems in cars.

[Master/Slave Concept](#).....291

A LIN bus must have one master (called the LIN master) and may have several slaves (called LIN slaves).

[Example of a LIN Bus](#).....292

Example of a typical network in a car.

Information in other sections

[LIN Support](#).....287

LIN Basics

Basics

LIN (local interconnect network) is a low-cost serial communication system for distributed electronic systems in cars. LIN is an open standard. It was developed to get a standard protocol to reduce the great variety of different low-end multiplex solutions, and cuts the cost of development, production, service, and logistics in car electronics. It is designed to connect electronic systems with low communication requirements, where the bandwidth and versatility of CAN are not required.

Supported LIN specifications and SAE standards

- LIN specifications 1.2 and 1.3
- LIN specifications 2.0, 2.1 and 2.2
- SAE J2602 standard (protocol version: J2602_1_1.0, language version: J2602_3_1.0)

SAE J2602 is a vehicle LIN bus standard based on LIN 2.0 and defined by the Society of Automotive Engineers (SAE).

Required firmware

Before using the RTI LIN MultiMessage Blockset, make sure the boot firmware and CAN firmware versions coming with the same dSPACE installation as the blockset are installed on your MicroAutoBox. For information on how to update the firmware, refer to [How to Update Firmware \(Firmware Manager Manual !\[\]\(166772600a13ad0a433053f90fe45649_img.jpg\)](#)).

Related topics

Examples

[Example of a LIN Bus..... 292](#)

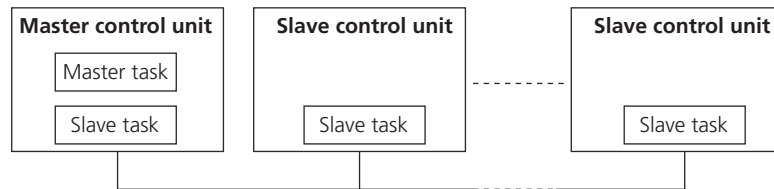
Master/Slave Concept

Introduction

A LIN bus consists of different nodes. It must have one master (called the LIN master) and may have several slaves (called LIN slaves). Each LIN master features one LIN master task and one LIN slave task. A LIN slave only features the LIN slave task.

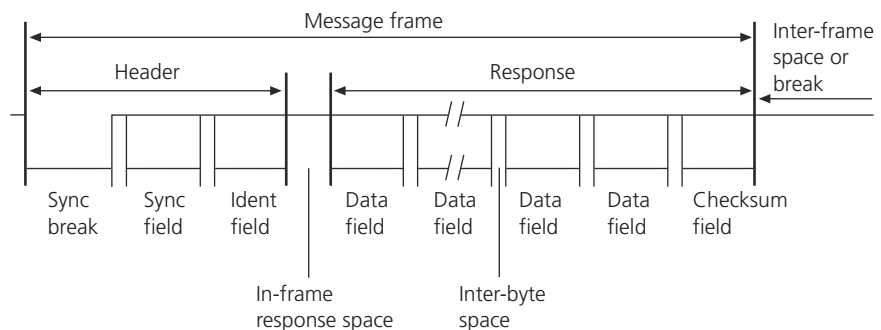
LIN master

The LIN master task is responsible for schedules (lists of contiguous frames including their characteristics) and sends the respective LIN headers. The LIN slave tasks are responsible for transmitting or receiving the LIN response, see the following illustration.



LIN messages

LIN messages are set up in a frame format. A frame is composed of a header, which is sent by the LIN master, and a response, which is sent by the LIN slave task, see the following illustration.



Detailed information

For more information on the LIN bus protocol and the frame format, refer to the *LIN Protocol Specification*.

Related topics

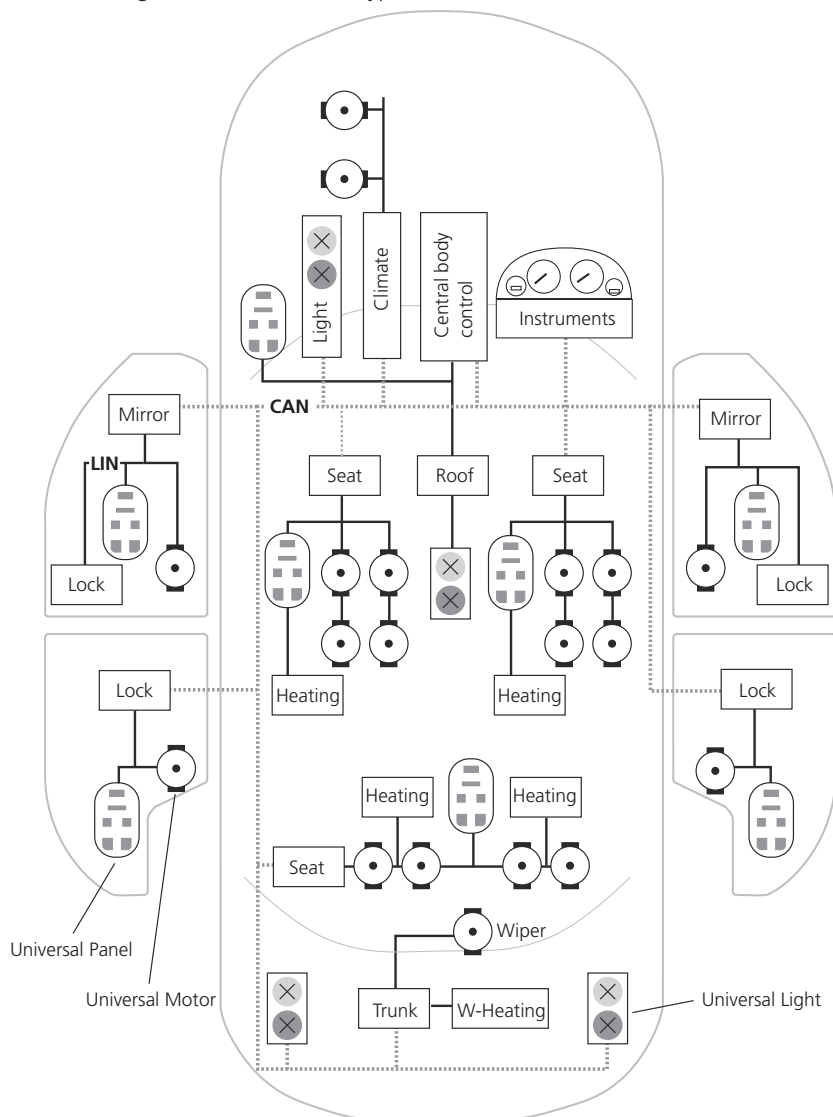
Basics

[LIN Basics.....](#) 290

Example of a LIN Bus

Example

The following illustration shows a typical network in a car.



The CAN bus (dotted line) connects ECUs throughout the whole car. The LIN buses (solid lines) connect ECUs in local areas.

Related topics

Basics

[LIN Basics.....](#) 290

Fields of Application

Introduction

Using MicroAutoBox, you can test the LIN network features of an electronic control unit (ECU). Several test scenarios are possible.

Where to go from here

Information in this section

Remaining Bus Simulation.....	294
Testing the transmission of message frames from the LIN master to simulated and real LIN slaves.	
Testing Communication Timing Constraints.....	295
Testing the time interval when the message frames are transmitted.	
Testing Against Specification Limits.....	295
Testing whether the ECU fulfills the requirements at the specification limits.	
Detecting the Baud Rate of a LIN Bus.....	296
Measuring the baud rate of a LIN bus.	
Testing Diagnostic and Failure Conditions.....	297
Testing the ECU in the case of failures.	
Testing Energy-Saving Modes.....	298
Testing the ECU in energy-saving modes.	

Remaining Bus Simulation

Introduction

An ECU with LIN master functionality can be tested using a dSPACE system with MicroAutoBox. The dSPACE system simulates the LIN nodes, which are connected to the LIN bus. The simulated bus members behave exactly the same way as real bus members. This functionality is called “remaining bus simulation”.

Real and simulated bus members

For specific test purposes it is useful to replace a simulated bus member by the real bus member. Either a bus member is simulated by a dSPACE system or the real controller is connected to the bus. In order to minimize the number of real-time applications required for the simulation, it is possible to switch off a simulated bus member without compiling the real-time application again. Otherwise a dSPACE simulator used to test the complete car body electronics would require hundreds of different real-time applications.

Related topics**Basics**

Testing Against Specification Limits.....	295
Testing Communication Timing Constraints.....	295
Testing Diagnostic and Failure Conditions.....	297
Testing Energy-Saving Modes.....	298

Testing Communication Timing Constraints

Testing communication timing constraints

For control algorithms it is important that an ECU sends its message frame within a given time interval. During simulation the simulator can observe the transmitted message frames and thus check the timing constraints.

Related topics**Basics**

Remaining Bus Simulation.....	294
Testing Against Specification Limits.....	295
Testing Diagnostic and Failure Conditions.....	297
Testing Energy-Saving Modes.....	298

Testing Against Specification Limits

Introduction

An ECU has to work under a wide range of conditions. Suppose you specified your ECU to work correctly in a LIN network at 9.6 kBd $\pm 10\%$. In this case, the dSPACE simulator has to run the LIN bus at 8.64 kBd to 10.56 kBd.

In-frame response time

A second important specification of the LIN network is the in-frame response time. This specifies the time between the end of the LIN header transmitted by the LIN master and the beginning of the LIN response transmitted by the LIN slave. Each LIN slave receiving the corresponding message frame and the LIN master must check whether the complete frame is completed within a given time interval. The RTI LIN MultiMessage Blockset can vary the in-frame response time in order to check whether the ECU behaves correctly under all conditions.

Baud rate detection

The actual available baud rate for a LIN bus can be detected by the RTI LIN MultiMessage Blockset and RTLlib functions. This is useful if a model is designed for various master nodes using different baud rates. If you specify the baud rate

in an LDF file, this baud rate is fixed. A detailed description of the baud rate detection used follows.

Related topics

Basics

Remaining Bus Simulation.....	294
Testing Diagnostic and Failure Conditions.....	297
Testing Energy-Saving Modes.....	298

Detecting the Baud Rate of a LIN Bus

Introduction

Baud rate detection for MicroAutoBox is performed by hardware units on the available channel.

Detected signal

The detected signal must contain a synchronization field consisting of the pattern '0x55' (hex). The hardware measures the 5 falling edges of the bit pattern (1010 1010).

Adjusted baud rate

For the channel on which the baud rate is measured the highest baud rate that is expected to occur must be adjusted.

Baud rate too small

If the baud rates are too small in relation to the adjusted baud rate they cannot be measured. Baud rate measurement is started when a synchronization break signal is detected. The break signal has a length of at least 10 low bit times. It is followed by a synchronization field. Depending on the baud rate, each low bit of a message is evaluated as a break signal, so measurement is started again and again.

Used algorithm

The measurement uses the following algorithm:

$$\text{baudrate_config}/8 \leq \text{baudrate_config} \leq \text{baudrate_config} + 15\%$$

The algorithm describes the actual value range the hardware is able to measure. If the specified baud rate is too small the baud rate cannot be measured correctly. For example, if you configure 9600 baud as the baud rate, but you want to work alternatively with 19,200 baud, this would not be possible. The algorithm first uses the configured value to detect the break signal. A break signal sent with 19,200 baud is too small to be detected with 9600 baud configured hardware. Thus, you must enter the highest available baud rate in the LDF file to start the baud rate measurement with the highest baud rate.

Related topics

Basics

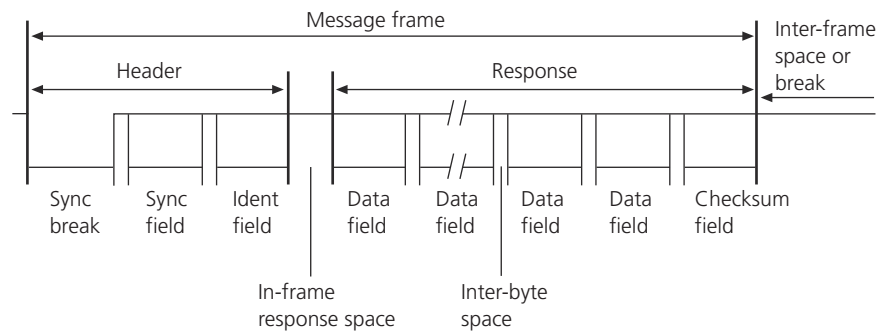
Testing Against Specification Limits..... 295

Testing Diagnostic and Failure Conditions

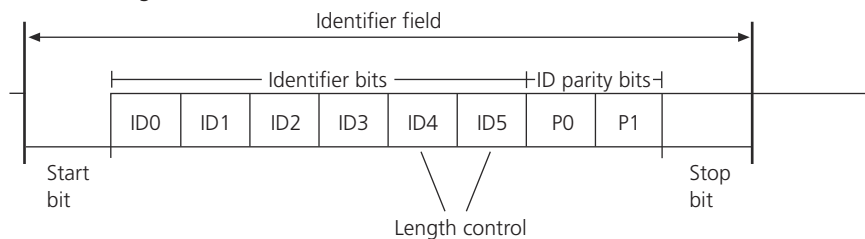
Testing diagnostic and failure conditions

Diagnostic and failure procedures are becoming more and more important within modern ECUs. To check the failure handling of an ECU, dSPACE Simulator can generate errors on the LIN bus.

The following illustration shows the format of a LIN message frame to show the errors which can be simulated.



The second illustration shows the format of the identifier field contained in the LIN message frame.



The following errors can be generated for test purposes.

Inconsistent-synch-byte error The synch byte is used by the ECUs to synchronize themselves to the LIN bus speed. The synch byte is transmitted as pattern 0x55 via the LIN bus. For error simulation a different pattern can be transmitted.

Identifier-parity error The identifier field consists of 6 bits plus 2 identifier parity bits. LIN slaves must not respond to incorrect identifiers or identifiers which are not of interest.

Slave-not-responding error After a LIN header is sent, the LIN slave must complete the frame within a given time interval. If this time interval is too long, the other LIN nodes must ignore the message. Bus collisions may occur if the LIN master starts the next LIN header before the LIN slave has finished its response.

Checksum error The response from the LIN slave is protected by a checksum field. If the checksum value is invalid, the LIN nodes ignore the message.

Related topics

Basics

Remaining Bus Simulation.....	294
Testing Against Specification Limits.....	295
Testing Communication Timing Constraints.....	295
Testing Energy-Saving Modes.....	298

Testing Energy-Saving Modes

Introduction

Body electronic control units are often connected to an unswitched power supply, because they must be operable even if the engine is not running: for example, a door lock unit. Therefore, these ECUs consume power even if the car is parked for days. Modern top-of-the-range cars may have more than 100 ECUs. To avoid discharging the battery, the ECUs must switch to low-power sleep mode when they are not in operation.

Low-power sleep mode

It is an important test scenario to check whether an ECU is correctly changing to low-power sleep mode and can be woken up again by several events. The dSPACE simulator can simulate and analyze these sleep and wake-up events. ECUs connected via buses can often be woken up and put into sleep mode by bus events.

In the case of LIN, the dSPACE simulator can react to the following events:

Event	Reaction
Sleep command received via the LIN bus	Put the LIN transceiver into sleep mode and inform the application that a sleep command has been received.
Wake-up command received via the LIN bus	Wake up the LIN transceiver and inform the application that a wake-up command has been received via the LIN bus.
Application wants to wake up the LIN bus	Wake up the LIN transceiver and send the wake-up command via the LIN bus.
Application wants to set the LIN bus asleep	Send the goto sleep command over the LIN bus and put the LIN transceiver into sleep mode.

Related topics

Basics

Remaining Bus Simulation.....	294
Testing Against Specification Limits.....	295
Testing Communication Timing Constraints.....	295
Testing Diagnostic and Failure Conditions.....	297

LIN Bus Handling

Introduction

The RTLib provides the C functions for implementing a LIN bus communication.

Where to go from here

Information in this section

Setting Up a LIN Bus.....	300
To transmit and receive frames you have to set up the LIN bus.	
Handling Frames.....	304
During simulation you can handle frames that are interchanged between the LIN nodes.	
Controlling a LIN Bus.....	305
The LIN bus can be controlled by the real-time application. You can switch between simulated and real nodes, or simulate energy-saving modes.	
Handling Schedules.....	308
Schedules can be used to send frames according to predefined lists. Interrupts can be triggered for different schedule events.	
Using Interrupts.....	309
Interrupts can be generated by bus events from a node, frame or schedule. The interrupts can be used to trigger interrupt-driven subsystems.	

Setting Up a LIN Bus

Introduction

To transmit and receive frames you have to set up the LIN bus.

Where to go from here

Information in this section

Basics on LIN Bus Handling.....	301
Provides basic information on how to simulate LIN master and LIN slaves.	
Specifying the LIN Bus Parameters.....	302
As the first step you must set up the LIN bus in your model or application.	

Database Files..... 302

Bus communication is specified in databases, which specify the information transmitted by an ECU, how it is coded and how often it is sent via the bus.

Defining LIN Nodes..... 303

After the LIN bus is set up, you can define all simulated LIN nodes.

Basics on LIN Bus Handling

Number of LIN buses

MicroAutoBox II provides two (1401/1507, 1401/1511, and 1401/1511/1514) or three (1401/1513 and 1401/1513/1514) LIN channels. Therefore, you can simulate 2 or 3 LIN buses with LIN master and LIN slaves. LIN data from within a real-time model or application can be transmitted and received. LIN frames and LIN bus events as well as energy-saving scenarios such as wake-up and sleep mode are supported.

Simulating a LIN master

If you want to simulate a LIN master you have to wire a 1 kΩ pull-up resistance in parallel to the slave's 30 kΩ pull-up resistance. Refer to [How to Configure MicroAutoBox II as the LIN Master \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(8bba887393ca45b761e5cb49e755e762_img.jpg\)](#)).

Connection to the LIN bus

The pinout depends on the MicroAutoBox II variant. Refer to:



- MicroAutoBox II 1401/1507: [Sub-D I/O Connector \(MicroAutoBox II Hardware Reference !\[\]\(4695f05050b0d393767d0512587d4e50_img.jpg\)](#))
- MicroAutoBox II 1401/1511: [ZIF I/O Connector \(MicroAutoBox II Hardware Reference !\[\]\(e6380cce6342e403c00cb7c9feb7e762_img.jpg\)](#))
- MicroAutoBox II 1401/1511/1514: [DS1511 ZIF I/O Connector \(MicroAutoBox II Hardware Reference !\[\]\(7528551b00a221bf92619f04e9b6fdc4_img.jpg\)](#))
- MicroAutoBox II 1401/1513: [ZIF I/O Connector \(MicroAutoBox II Hardware Reference !\[\]\(61103e278339dd729c6548dc05b28450_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [DS1513 ZIF I/O Connector \(MicroAutoBox II Hardware Reference !\[\]\(c54f4cb3d90acdac1abe98c1a1e231a3_img.jpg\)](#))

Related topics

HowTos

[How to Configure MicroAutoBox II as the LIN Master \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(41aea2746216b27a6939d696d8e035da_img.jpg\)](#))

Specifying the LIN Bus Parameters

Introduction	As the first step you must set up the LIN bus in your model or application.
Configuration steps	<p>If you want to set up the LIN bus, you have to perform some configuration steps.</p> <ol style="list-style-type: none"> 1. Set the parameters to select the LIN Interface Board, the LIN channel, the transceiver type the LIN bus is connected to and the termination resistance. For information on setting the termination resistance for a LIN master, refer to How to Configure MicroAutoBox II as the LIN Master (MicroAutoBox II Hardware Installation and Configuration Guide ). 2. Use a database file in the LDF, DBC, FIBEX or AUTOSAR XML file format. One database file can be used for several LIN buses. For more information on these file formats, refer to Database Files on page 302. 3. Define a name for the LIN bus. This name must be unique in the model or application. 4. If the baud rate is not specified in the database file, define the baud rate.
RTLib user	Use the <code>dslin_tp1_board_init</code> , <code>dslin_channel_init</code> and <code>dslin_channel_create</code> functions. Define the necessary parameters in them. For detailed information on the functions, refer to LIN Channel Handling (MicroAutoBox II RTLib Reference ).

Database Files

Introduction	Specifying communication via buses in a car is a complex activity, because a bus consists of a lot of different ECUs, which may be developed by several departments. Therefore, bus communication is specified in databases, which specify the information transmitted by an ECU, how it is coded and how often it is sent via the bus.
Supported database file types	<p>LDF file The LDF file format describes a complete LIN network and contains all the information necessary to configure it. The file format was specially developed for LIN networks, so it can describe all the features of a LIN network. For more information on the LDF file format, refer to the <i>LIN Configuration Language Specification</i>.</p> <p>DBC file The DBC file format is designed by Vector Informatik GmbH for the CANalyzer database files. Although it was developed for CAN buses, it can also be used for LIN buses with some limitations:</p> <ul style="list-style-type: none"> ▪ As the CAN protocol has no standard definitions for master nodes and schedules, such definitions are not supported by the DBC files.

- According to the LIN specification only byte layouts in the Intel format are supported. Byte layouts in Motorola format are not supported.

FIBEX file 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 AUTOSAR system description files are XML files that describe a system according to AUTOSAR. A system is a combination of a hardware topology, a software architecture, a network communication, and information on the mappings between these elements. AUTOSAR system description files are instances of the AUTOSAR System Template. The AUTOSAR System Template contains a description of the network communication and hardware topology according to the FIBEX standard defined by ASAM e.V.

AUTOSAR system description files are files of AUTOSAR XML file type that can be used to export or exchange information on system descriptions.

For information on which AUTOSAR Releases are supported, refer to [Features of the RTI LIN MultiMessage Blockset \(RTI LIN MultiMessage Blockset Reference !\[\]\(83f22ed94ec5517769dd76d702c6bfd8_img.jpg\)](#)).

Defining LIN Nodes

Introduction

After the LIN bus is set up, you can define all simulated LIN nodes.

LIN node types

Two types of LIN nodes are available:

- The master node controls the LIN bus and protocol and controls which message has to be sent over the LIN bus at a specified time.
- The slave nodes receive frame headers from the master and send the corresponding frame over the LIN bus.

Setting up a LIN node

The following steps have to be taken to setup a LIN node. You must define at least one LIN node for a LIN bus.

1. Specify the LIN bus to which the LIN node is connected.
2. Select the LIN node from the database file or define a user-defined node (only slave type possible). The name must be unique in the LIN bus.
3. Repeat the above steps for all simulated LIN nodes of the LIN bus.
4. Define the TX and RX error counters available for detecting errors that occur during the sending or receiving of frames.

RTLib user

Use the `dslin_node_create` and `dslin_node_init` functions. Refer to [LIN Node Handling \(MicroAutoBox II RTLib Reference !\[\]\(06a315363e7801bba8c7489a6694af19_img.jpg\)](#)).

Handling Frames

Database file

The signals to be transferred via a LIN bus are composed as frames. The frame composition is described by the database file, which is selected during bus setup.

Where to go from here**Information in this section****[Transmitting and Receiving Frames..... 304](#)**

After you have set up the LIN node, frames can be transmitted or received in the model or application.

[Manipulating Frames..... 305](#)

In some application fields it is necessary to access the frames before they are transmitted or decoded. For this purpose you can manipulate the frame data.

Information in other sections**[Specifying the LIN Bus Parameters..... 302](#)**

As the first step you must set up the LIN bus in your model or application.

Transmitting and Receiving Frames

Introduction

The signals are transmitted in frames. Therefore, the signals must be encoded before they are transmitted via the LIN bus. The rules for encoding are defined in the database file. In the same way, a received frame must be decoded first to get the signals.

RTLib user

The RTLib for the DS4330 provides functions for transmitting and receiving frames. Functions to encode messages to frames or to decode frames to messages are not provided by RTLib. You have to program them yourself. Refer to [LIN Frame Handling \(MicroAutoBox II RTLib Reference !\[\]\(9db214d549b9aeebe72aa11d3a5c4b1a_img.jpg\)](#)).

Manipulating Frames

Introduction

In some application fields it is necessary to access the frames before they are transmitted or decoded. For this purpose you can manipulate the frame data, for example, to simulate a transmission error.

RTLib user

The RTLib for the DS4330 provides functions for transmitting and receiving frames. Functions to encode messages to frames or to decode frames to messages are not provided by RTLib. You have to program them yourself. Additionally, you also have to program the function for manipulating the frames. For information on the transfer functions, refer to [LIN Frame Handling \(MicroAutoBox II RTLib Reference !\[\]\(d66ff64371a51729ac8c1cdaa685ba6f_img.jpg\)](#)).

Controlling a LIN Bus

Introduction

The LIN bus can be controlled by the real-time application. You can switch between simulated and real nodes, or simulate energy-saving modes.

Where to go from here

Information in this section

[Switching LIN Nodes.....](#) 306

The LIN nodes connected to a LIN bus can be real or simulated in the model or application. You can enable and disable the simulation of a LIN node while the application is running.

[Setting a LIN Bus to Sleep Mode.....](#) 306

Only LIN buses in sleep mode (energy saving mode) can be woken up.

[Waking Up a LIN Bus.....](#) 306

An important feature of a LIN bus is the support of energy-saving modes. This means that it can be put to sleep and woken up after receiving a specific signal.

[Changing LIN Bus Parameters.....](#) 307

To test the LIN bus under different operating conditions, some parameters must be changed during simulation.

[Reading Status Information.....](#) 307

During simulation you can read the status information of LIN nodes.

Switching LIN Nodes

Introduction

If you want to replace a simulated LIN node by a real controller you have to disable the LIN node in the model or application. RTLib contains functions to disable or enable simulated LIN nodes during simulation. Therefore, it is not necessary to recompile the model or application even if the LIN bus structure changes.

Note

Do not enable a simulated LIN node if the corresponding real LIN node is also connected to the LIN bus. Otherwise two LIN slaves would answer to the same header and cause errors.

RTLib user

Use `dslin_node_enable` to enable a LIN slave and `dslin_node_disable` to disable a LIN slave. Refer to [LIN Node Handling \(MicroAutoBox II RTLib Reference !\[\]\(de95854c7ee024cfadc48187bbb781b2_img.jpg\)](#)).

Setting a LIN Bus to Sleep Mode

Introduction

For testing the energy-saving mode, the whole LIN bus can be set to sleep mode. The corresponding signal has to be sent by the LIN master node. For waking up the LIN bus, see [Waking Up a LIN Bus](#) on page 306.

RTLib user

Use `dslin_node_command_sleep` to set a LIN bus to sleep mode. Refer to [dslin_node_command_sleep \(MicroAutoBox II RTLib Reference !\[\]\(9c2e8d1b5bd77cb5c9f83b7a9cff79fd_img.jpg\)](#)).

Waking Up a LIN Bus

Introduction

An important feature of a LIN bus is the support of energy-saving modes. This means that it can be put to sleep and woken up after receiving a specific signal. This frame can be generated by an RTLib function.

RTLib user

Use `dslin_node_command_wakeup` to wake up a LIN bus. Refer to the [dslin_node_command_wakeup \(MicroAutoBox II RTLib Reference !\[\]\(eabd9f9ababee93effadc3b380fe65fd_img.jpg\)](#)).

Related topics

Basics

Setting a LIN Bus to Sleep Mode.....	306
Testing Energy-Saving Modes.....	298

Changing LIN Bus Parameters

Introduction






To test if the ECU fulfills the requirements of the specification, you have to change some bus and frame parameters during the simulation (see [Testing Energy-Saving Modes](#) on page 298).

RTLib user

Use `dslin_channel_baudrate_set`, `dslin_channel_breaklength_set`, `dslin_channel_breakdelimiter_set`, `dslin_channel_synchfield_set` to change the parameters and `dslin_channel_apply_settings` to activate the new settings.

Related topics

References


- [dslin_channel_apply_settings](#) (MicroAutoBox II RTLib Reference )
- [dslin_channel_baudrate_set](#) (MicroAutoBox II RTLib Reference )
- [dslin_channel_breakdelimiter_set](#) (MicroAutoBox II RTLib Reference )
- [dslin_channel_breaklength_set](#) (MicroAutoBox II RTLib Reference )
- [dslin_channel_synchfield_set](#) (MicroAutoBox II RTLib Reference )

Reading Status Information

Introduction

Five different message error types are specified in the LIN specification (see *LIN Protocol Specification 1.2*). While the LIN nodes are running, the errors are counted. You can read the counters for each node type (master and slave) with RTLib functions.

RTLib user

Several functions are provided to get status information for both node types. Refer to [LIN Frame Handling](#) (MicroAutoBox II RTLib Reference )

Handling Schedules

Introduction

Schedules can be used to send frames according to predefined lists. Interrupts can be triggered for different schedule events.

Where to go from here

Information in this section

[Setting Up a LIN Schedule..... 308](#)

Currently executed LIN schedules can be interrupted and restarted or resumed. The behavior can be defined.

[Setting the Priority of LIN Schedules..... 309](#)

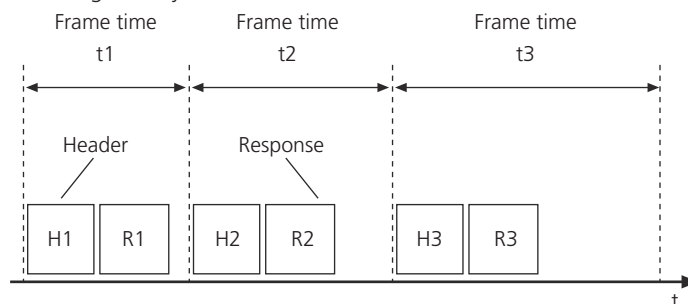
If you have defined several schedules in an LDF file, you can implement a decision logic to rank the schedules according to their priority. With RTLib you can start and stop a schedule with the corresponding functions. No ranking is required.

Setting Up a LIN Schedule

Basics on LIN schedules

LIN schedules have to be predefined in LDF files. The master node sends the listed headers and the corresponding nodes answer by sending the corresponding frame. Running schedules can be interrupted to start another schedule. After completing this schedule, the interrupted schedule can be restarted from the beginning or resumed where it was interrupted.

If you want to send several frames over the LIN bus in a specified time, you must use LIN schedules. RTLib provides functions to set up, start and stop LIN schedules. The following illustration shows a schedule with several frames. The frame time specifies the time to send the header and to get the response including a delay time.



The schedule timing is calculated by the CAN_TP1 module of MicroAutoBox. The times of the CAN_TP1 module and the DS1401 base Board are synchronized to ensure that time stamps and the simulation time are consistent.

RTLib user

Several functions are provided to specify and handle LIN schedules. Refer to [LIN Schedule Handling \(MicroAutoBox II RTLib Reference !\[\]\(2e897e890e69d81eae4503a8342c36b0_img.jpg\)\)](#).

Setting the Priority of LIN Schedules

Introduction

If you use several schedules you can define the priority of LIN schedules. Schedules with higher priority can interrupt schedules with lower priority.

RTLib user

With RTLib you can start and stop a schedule with the corresponding functions. Schedules cannot be ranked.

Using Interrupts

Defining Interrupts

Introduction

To trigger a subsystem, nodes or frames can generate interrupts when different events occur.

Events

The following events are defined for a node.

Node events The following events are defined for nodes:

- Bus wake-up
- Bus sleep request
- No bus activity
- RX error counter threshold exceeded
- TX error counter threshold exceeded
- Identifier-parity error
- Inconsistent-synch-field error

Events for a received frame The following events are defined for a received frame:

- Message received
- Checksum error
- Slave-not-responding error

Events for a transmitted frame The following events are defined for a transmitted frame:

- Header received
- Frame received
- Message transmitted
- Bit error

Schedule events The following events are defined for schedules:

- Schedule started
- Schedule terminated
- Schedule aborted
- Schedule restarted

For a description of the events, refer to the documentation of the interrupt features.

RTLib user

Several functions are provided to implement node, frame and schedule interrupts. Refer to [LIN Interrupt Handling \(MicroAutoBox II RTLib Reference !\[\]\(10f8862fc183b400327470ea85afe9ae_img.jpg\)](#)).

Using the RTI LIN MultiMessage Blockset

Introduction

The RTI LIN MultiMessage Blockset is a Simulink blockset for efficient and dynamic handling of complex LIN setups in hardware-in-the-loop (HIL) applications.

Where to go from here

Information in this section

[Basics on the RTI LIN MultiMessage Blockset.....311](#)

Provides an overview of the features of the RTI LIN MultiMessage Blockset.

[Transmitting and Receiving LIN Frames.....313](#)

Large LIN frame bundles can be managed from a single Simulink block provided by the RTI LIN MultiMessage Blockset.

[How to Define a Checksum Algorithm.....316](#)

You can specify your own checksum algorithm for frames.

[Manipulating Signals to be Transmitted.....317](#)

You can analyze signals of RX frames or change the values of signals of TX frames in the experiment software.

Basics on the RTI LIN MultiMessage Blockset

Introduction

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

Supported dSPACE platforms

The RTI LIN MultiMessage Blockset is supported by the following platforms:

- SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, and/or DS6351 LIN Board
- PHS-bus-based systems (DS1006 or DS1007 modular systems) with a DS4330 LIN Interface Board
- MicroAutoBox II with LIN interface

You have to recreate all the RTI LIN MultiMessage blocks if you switch platform (for example, from SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, and/or DS6351 LIN Board to a DS1007, or from

a DS1007 to a MicroAutoBox II). To recreate all the RTILINMM blocks at once, select **Create S-function for All LIN Blocks** from the options menu of the GeneralSetup block (refer to [Options Menu \(RTILINMM GeneralSetup\)](#) (RTI LIN MultiMessage Blockset Reference )).

Note

The RTI LIN MultiMessage Blockset is not supported by the MicroAutoBox III. If you work with the MicroAutoBox III, you must use the Bus Manager to implement LIN bus simulation.

Managing large LIN frame bundles


With the RTI LIN MultiMessage Blockset, you can configure and control a large number of LIN frames from a single Simulink block. This reduces the size of model files and the time required for code generation and the build process.

Manipulating signals to be transmitted

The RTI LIN MultiMessage Blockset can manipulate the counter values of signals before they are transmitted.

When simulating with signal manipulation, you can specify whether to use the signal from the model or the TRC file. This is useful for simulating error values.

Specifying signal saturation

You can use the signal limits specified in the database file or specify other limits. If the input signal is outside of the specified range, it is set to the minimum or maximum limit. Refer to [Saturation Page \(RTILINMM MainSetup\)](#) (RTI LIN MultiMessage Blockset Reference ).

Updating a model

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

Modifying model parameters during run time

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

(Relevant only for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, and/or DS6351 LIN Board) For information on where to find the signals of the LIN bus in the TRC file, refer to [Details on the Variable Groups of the Behavior Model \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(2b17f17ebbacc911bb0ff784ab641779_img.jpg\)\)](#).

Variables of custom code	You can include variables of custom code in a USR.TRC file in addition to the parameters of the RTI LIN MultiMessage Blockset.
TRC file entries with initial data	TRC/SDF files generated for Simulink models including blocks from the RTI LIN MultiMessage Blockset contain initial data. The RTI LIN MultiMessage Blockset supplies all variables with initial values when they are included in the TRC file. TRC files with initial data allow you to perform offline calibration with ControlDesk.
Visualization with the Bus Navigator	The RTI LIN MultiMessage Blockset supports visualization with the Bus Navigator. Layouts/instruments are generated on demand and provide access to all the LIN signals and all the switches required to configure LIN communication during run time. You do not have to preconfigure layouts by hand.
Monitoring and logging LIN bus communication	The RTI LIN MultiMessage Blockset supports filtered and unfiltered monitoring and logging of LIN bus communication with the Bus Navigator. You can observe the raw and physical data of LIN frames on a LIN bus, and log the raw data of LIN frames. You can monitor and log LIN bus events.

Transmitting and Receiving LIN Frames

Introduction	Large LIN frame bundles (up to 63 frames) can be managed from a single Simulink block provided by the RTI LIN MultiMessage Blockset.
Defining LIN communication	<p>To define the LIN communication of a LIN controller, you must provide a database file.</p> <p>LDF file as the database You can use the LDF file format as the database for LIN communication. The LDF file format describes a complete LIN network and contains all the information necessary to configure it. The file format was specially developed for LIN networks, so it can describe all the features of a LIN network. For more information on the LDF file format, refer to the <i>LIN Configuration Language Specification</i>.</p> <p>DBC file as the database You can also use the DBC file format as the database for LIN communication. The DBC file format was developed by Vector Informatik GmbH, Stuttgart, Germany. For the RTI LIN MultiMessage Blockset, you can use all the DBC files that pass the consistency check of Vector Informatik's CANdb++ Editor.</p>

The DBC file format was developed for CAN buses, but it can also be used for LIN buses with some limitations:

- As the CAN protocol has no standard definitions for master nodes and schedules, such definitions are not supported by the DBC file formats.
- The LIN specification supports only byte layouts in the Intel format. Byte layouts in Motorola format are not supported.

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

MAT file as the database You can also use the MAT file format as the database for LIN communication, or specify other database file formats as the database. You must convert your database files into the MAT file format for this purpose.

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

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


Defining RX frames and TX frames

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


Defining RX frames You can define RX frames on the RX Frames Page (RTILINMM MainSetup).

Defining TX frames You can define TX frames on the TX Frames Page (RTILINMM MainSetup).

Working with event-triggered frames

The RTI LIN MultiMessage Blockset allows you to work with event-triggered frames. In contrast to a standard LIN frame, the frame header of an event-triggered frame is assigned to several frame responses of different LIN nodes. This allows you to transmit the frames of different nodes via the same frame slot. If you work with LIN 2.1 or later, you can specify collision resolver schedules to resolve collisions which might occur if two or more event-triggered frames are sent at the same time via the same frame slot. For details, refer to [Eventtriggered Frames Page \(RTILINMM MainSetup\)](#) (RTI LIN MultiMessage Blockset Reference ).

Working with raw data


The RTI LIN MultiMessage Blockset allows you to work with the raw data of frames. You have to select the frames for this purpose. The RTILINMM MainSetup block then provides the raw data of these frames to the model byte-wise for further manipulation. You can easily access the raw data of RX frames via a Simulink Bus Selector block. For details, refer to [Raw Data Page \(RTILINMM MainSetup\)](#) ([RTI LIN MultiMessage Blockset Reference](#) ).

Implementing checksum algorithms

In addition to the checksums of the LIN frames, you can use a signal to transmit a user-defined checksum. You can implement checksum algorithms for the checksum calculation of TX frames and checksum verification of RX frames.

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

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

Checksum check for RX frames You can assign a checksum algorithm to each RX frame. A checksum is calculated for the frame and compared to the checksum in the received frame. If they differ, this is indicated at the error ports for RX frames if these ports are enabled. For details, refer to [User Checksum Definition Page \(RTILINMM MainSetup\)](#) ([RTI LIN MultiMessage Blockset Reference](#) ).

LIN frame checksum manipulation You can manipulate the checksums of LIN frames. The checksum is increased by a specified fixed offset value for a defined number of times.

Receiving frames with a different frame length than specified in database file

If the length of a received frame is different than that specified in the database file, how the frame is handled depends on its length.

Received frame shorter than expected If a frame is shorter than specified in the database file, it is not decoded. The error port is set to **Slave not responding** (**RX_Error = 16**). The frame status remains 0. The RX_Time is updated. The received data is output at the RX Raw Data port. Raw data bytes which are not received are set to 0. You can read the frame length at the Frame Length port.

Received frame longer than expected If a frame is longer than specified in the database file, only the specified frame length is read. The next byte is interpreted as a checksum. A checksum error is therefore displayed in most cases. The raw data, RX_Time and RX_DeltaTime are updated. The error port is set to **Checksum Error** (**RX_Error = 8**). If the last byte is the correct checksum, the frame is decoded as usual.

How to Define a Checksum Algorithm

Objective


You can specify your own checksum algorithm for frames.

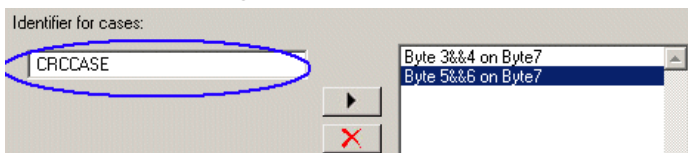
Basics

You can implement checksum algorithms for frames via a *checksum header file*. Each algorithm must have a C-coded switch-case directive in the header file.


Method

To define a checksum algorithm

- 1 Open the User Checksum Definition page.
- 2 Click  to specify an existing checksum header file, or enter a checksum header file name in the Header file edit field.
- 3 Name the checksum algorithms in the Identifier for cases field.



- 4 In the checksum header file, sort the algorithms.
The order you specify for the algorithms corresponds to the `crctype` index of the switch-case directives in the header file.
- 5 Click **Create Header File** to create the header file if you entered its file name in the Header file edit field in step 1.
- 6 Click **Edit Header File** to edit the file in MATLAB's M-File Editor.
- 7 In the checksum header file, edit your checksum algorithms. Note that the header file must contain at least one switch-case directive.
You have access to the following input parameters of the header file:

Input Parameter	Description
<code>crcoption</code>	<ul style="list-style-type: none"> ▪ '0' if applied to a TX frame ▪ '1' if applied to an RX frame See User Checksum Frames Page (RTILINMM MainSetup) (RTI LIN MultiMessage Blockset Reference ).
<code>UInt8* FrameRAW_DATA</code>	Pointer to 8 bytes of raw data
<code>crctype</code>	Index of the switch-case directives for the checksum algorithms. Corresponds to the order you specify in step 3 (see above).
<code>CsBitPos</code>	Start position of the checksum signal
<code>CsLength</code>	Length of the checksum signal

Input Parameter	Description
MsgLength	Frame length (in range 1 ... 8)
MsgId	Frame ID (1 ... 59)

Result Your checksum algorithm is used.

Related topics

References

[RTILINMM MainSetup \(RTI LIN MultiMessage Blockset Reference !\[\]\(74d4806277d7e73349d8e8c0897931e9_img.jpg\)\)](#)
[User Checksum Definition Page \(RTILINMM MainSetup\) \(RTI LIN MultiMessage Blockset Reference !\[\]\(5f42d2cd7ad901bc24e5d35a38c777fd_img.jpg\)\)](#)

Manipulating Signals to be Transmitted

Introduction

All the signals of all the RX and TX frames (see [Transmitting and Receiving LIN Frames](#) on page 313) automatically get corresponding entries in the generated TRC file. This allows you to analyze them (signals of RX frames) or change their values (signals of TX frames) in the experiment software.

Manipulating signals to be transmitted

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

You can switch between these options in the experiment software.

TX signals The RTILINMM MainSetup block allows you to get the value of TX signals either from the Simulink model or from a variable in ControlDesk. The values of these TX signals can be changed from within the model. Their values cannot be changed in ControlDesk if the **Input Manipulation** option is cleared. Refer to [Input Manipulation Page \(RTILINMM MainSetup\) \(RTI LIN MultiMessage Blockset Reference !\[\]\(bd3b31712ad9bab5a241210fa6925cdd_img.jpg\)\)](#).

Counter signals You can specify a counter signal that provides the number of frame transmissions. You can specify the counter start value, the increment, and the counter stop value. Each time the counter reaches the stop value, it turns around to the counter start value. Refer to [Counter Page \(RTILINMM MainSetup\) \(RTI LIN MultiMessage Blockset Reference !\[\]\(0fb13ad0bfa3d86868cdd3883e5665b3_img.jpg\)\)](#).

Related topics

Basics

[Transmitting and Receiving LIN Frames..... 313](#)

FlexRay Support

Purpose To connect MicroAutoBox II to a FlexRay bus.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	✓	–	–	✓

Where to go from here

Information in this section

[Basics on FlexRay Support..... 320](#)

Information in other sections

[Basics of the FlexRay Configuration Tool \(FlexRay Configuration Tool Guide !\[\]\(d0262bbe9d2356661a2e89321dfcc781_img.jpg\)](#))

Provides a short introduction to FlexRay networks and the FlexRay Configuration Tool

[Setting up a FlexRay Network \(FlexRay Configuration Features !\[\]\(c444627dab9fee9a1550c053ffaaaae2_img.jpg\)](#))

Shows how you can work with the RTI FlexRay Configuration Blockset.

Real-Time Simulation with FlexRay Networks (FlexRay Configuration Features)

When the Simulink model is finished, you can build the real-time application, download the code to the real-time system, and start the simulation.

Basics on FlexRay Support

Characteristics

The following MicroAutoBox variants provide up to 2 IP modules to connect to FlexRay networks:

- MicroAutoBox II 1401/1507
- MicroAutoBox II 1401/1511/1514
- MicroAutoBox II 1401/1513/1514

For further information, refer to [Basics on IP Module Support](#) on page 323.

I/O mapping

The following tables show the mappings of converter and channel numbers, as used in RTI and RTLib, to the related I/O pins of the related I/O connector.

- MicroAutoBox II 1401/1507

Unit	Channel	Signal	I/O Connector Pin (Sub-D Connector)
IP Module 1	-	Wake-up 1 ¹⁾	13
		INH 1 ¹⁾	33
		INH 2 ¹⁾	34
		UBAT ¹⁾	35
	FlexRay 1	I1_GND 1	15
		FlexRay 1 high in/out (FlexRay-H)	16
		FlexRay 1 low in/out (FlexRay-L)	17
		Feed-through line positive 1 ¹⁾	38
	FlexRay 2	Feed-through line negative 1 ¹⁾	39
		I1_GND 2	18
		FlexRay 2 high in/out (FlexRay-H)	19
		FlexRay 2 low in/out (FlexRay-L)	20
		Feed-through line positive 2 ¹⁾	36
		Feed-through line negative 2 ¹⁾	37

Unit	Channel	Signal	I/O Connector Pin (Sub-D Connector)
IP Module 2	-	Wake-up 2 ¹⁾	52
		INH 1 ¹⁾	72
		INH 2 ¹⁾	73
		UBAT ¹⁾	74
	FlexRay 1	I2_GND 1	54
		FlexRay 1 high in/out (FlexRay-H)	55
		FlexRay 1 low in/out (FlexRay-L)	56
		Feed-through line positive 1 ¹⁾	77
	FlexRay 2	Feed-through line negative 1 ¹⁾	78
		I2_GND 2	57
		FlexRay 2 high in/out (FlexRay-H)	58
		FlexRay 2 low in/out (FlexRay-L)	59
		Feed-through line positive 2 ¹⁾	75
		Feed-through line negative 2 ¹⁾	76

¹⁾ Available with DS1507-02-003

- MicroAutoBox II 1401/1511/1514, MicroAutoBox II 1401/1513/1514

Unit	Channel	Signal	I/O Connector Pin (DS1514 ZIF Connector)
IP Module 1	-	Wake-up 1	C6
		INH 1 ¹⁾	(G5)
		INH 2 ¹⁾	(H6)
		UBAT ¹⁾	(H5)
	FlexRay 1	I1_GND 1	L3
		FlexRay 1 high in/out (FlexRay-H)	M3
		FlexRay 1 low in/out (FlexRay-L)	M4
		Feed-through line positive 1 ¹⁾	(K6)
	FlexRay 2	Feed-through line negative 1 ¹⁾	(K5)
		I1_GND 2	J3
		FlexRay 2 high in/out (FlexRay-H)	K3
		FlexRay 2 low in/out (FlexRay-L)	K4
		Feed-through line positive 2 ¹⁾	(J6)
		Feed-through line negative 2 ¹⁾	(J5)

Unit	Channel	Signal	I/O Connector Pin (DS1514 ZIF Connector)
IP Module 2	-	Wake-up 2	S6
		INH 1 ¹⁾	(W5)
		INH 2 ¹⁾	(X6)
		UBAT ¹⁾	(X5)
	FlexRay 1	I2_GND 1	a3
		FlexRay 1 high in/out (FlexRay-H)	b3
		FlexRay 1 low in/out (FlexRay-L)	b4
		Feed-through line positive 1 ¹⁾	(Z6)
	FlexRay 2	Feed-through line negative 1 ¹⁾	(Z5)
		I2_GND 2	Y3
		FlexRay 2 high in/out (FlexRay-H)	Z3
		FlexRay 2 low in/out (FlexRay-L)	Z4
		Feed-through line positive 2 ¹⁾	(Y6)
		Feed-through line negative 2 ¹⁾	(Y5)

¹⁾ Do not connect, refer to [Connecting to a FlexRay Bus \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(3dfb8d66e81160ad61421a3452093d1b_img.jpg\)](#)).

Related topics

References

[Data Sheet MicroAutoBox II 1401/1507 \(MicroAutoBox II Hardware Reference !\[\]\(339a16584d5da0f0a3ca4e9ec17bf6a1_img.jpg\)](#))
[Data Sheet MicroAutoBox II 1401/1511/1514 \(MicroAutoBox II Hardware Reference !\[\]\(e06a1d39938b2f5d7a2c3618fea4f77f_img.jpg\)](#))
[Data Sheet MicroAutoBox II 1401/1513/1514 \(MicroAutoBox II Hardware Reference !\[\]\(23ac9e28f2600a1e787d149d7f76716a_img.jpg\)](#))

IP Module Support

Purpose MicroAutoBox II equipped with FlexRay or CAN FD modules can be connected to a FlexRay or CAN bus.

Hardware requirements

Supported MicroAutoBox II hardware:


DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	✓	–	–	✓

Basics on IP Module Support

Introduction

The following MicroAutoBox II variants provide up to 2 IP modules to connect modules that support FlexRay or CAN FD:

- MicroAutoBox II 1401/1507
- MicroAutoBox II 1401/1511/1514
- MicroAutoBox II 1401/1513/1514

Various IP modules are supported, such as the DS4340 FlexRay Interface Modules and the DS4342 CAN FD Interface Modules. Refer to [FlexRay Support](#) on page 319 and [CAN Signal Mapping](#) on page 281. For information on how to connect the IP modules, refer to [MicroAutoBox II Hardware Installation and Configuration Guide](#) .

Note

Because the IP module connector is designed to run FlexRay or CAN FD, not all of the IP modules' signals are available. The IP module hardware is adapted to MicroAutoBox II.

Note

Support of DS4340 FlexRay Interface Module:

- MicroAutoBox II 1401/1507 with board revision DS1507-02-003 and higher is prepared to be used with the DS4340 FlexRay Interface Module. The enlarged signal mapping is described at [FlexRay Support](#) on page 319.
- If you want to use MicroAutoBox II 1401/1507 with lower I/O board revision number as stated above together with the DS4340 FlexRay Interface Module, the hardware must be adapted by dSPACE.

Characteristics

IP modules can be used as I/O devices to connect different carrier boards to the outside world. The MicroAutoBox II variants provide two IP modules.

MicroAutoBox Variant	Number of IP Modules	I/O Board to be Connected
MicroAutoBox II 1401/1507	2	1507
MicroAutoBox II 1401/1511/1514	2	1514
MicroAutoBox II 1401/1513/1514	2	1514

Up to 8 MB memory is available via byte or word addressing. Two clock rates, 8 and 32 MHz, allow a data rate of up to 64 MByte/s. Each module connected to DS1507 or DS1514 supports one interrupt.

RTLib user

You have access to the IP modules via RTLib1401. For details, see [IP Module Access Functions \(MicroAutoBox II RTLib Reference !\[\]\(758ebdf4629c903da74c2e079717ae32_img.jpg\)](#)).

I/O mapping

The complete pin description is available in the data sheet of your MicroAutoBox II variant in [MicroAutoBox II Hardware Reference !\[\]\(626ce8ac21792b9405bfddfea8e0c96a_img.jpg\)](#).

Related topics**Basics**

[FlexRay Support..... 319](#)

FPGA Support

Introduction The FPGA Type 1 unit of the DS1514 I/O board provides an FPGA module that can be used for implementing FPGA applications and integrating them to your MicroAutoBox II real-time application.

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
-	-	-	-	✓

Where to go from here	Information in this section
	General Information on the FPGA Support..... 326 Provides information on hardware and software components relevant for FPGA support.
	Accessing the FPGA Type 1 Unit..... 327 There are some general FPGA access functions to be mentioned which are independent from the implemented FPGA application.

General Information on the FPGA Support

Hardware components

The DS1514 I/O board of MicroAutoBox II provides the FPGA Type 1 unit, that can be equipped with an I/O module (piggyback board).

The FPGA Type 1 unit is directly connected to the MicroAutoBox II processor unit via intermodule bus. For internal data exchange, there are up to 512 kB buffer memory directly addressable on the FPGA Type 1 unit. For external data exchange, there are up to 130 I/O pins mapped to the DS1514 ZIF I/O connector of MicroAutoBox II.

For a graphical overview on the DS1552 Multi-I/O Module, refer to [MicroAutoBox II with DS1552 Multi-I/O Module](#) on page 23.

The FPGA Type 1 unit of the DS1514 I/O board has a Xilinx® Kintex®-7 FPGA XC7K325T FPGA module.

Software components

RTLib1401 includes C functions (`fpga_tp1_xxx`) to implement the communication on the processor side and program the FPGA. For further information, refer to [Accessing the FPGA Type 1 Unit](#) on page 327.

Using DS1552 Multi-I/O Module You can use the RTI DS1552 I/O Extension Blockset as a preconfigured FPGA application for the DS1552 Multi-I/O Module. It provides further I/O channels for DS1511-like I/O features.

Alternatively, you can use the RTI FPGA Programming Blockset with a MicroAutoBox II-specific framework specified to implement custom FPGA applications.

- MicroAutoBox II variant with DS1514 I/O board:
Use the FPGA1401Tp1 (7K325) with Multi-I/O Module (DS1552) framework.

Using DS1554 Engine Control I/O Module You can use the RTI FPGA Programming Blockset with a MicroAutoBox II-specific framework specified to implement custom FPGA applications.

- MicroAutoBox II variant with DS1514 I/O board:
Use the FPGA1401Tp1 (7K325) with Engine Control I/O Module (DS1554) framework.


Note

You cannot use blocks from the RTI FPGA Programming Blockset configured with one of the FPGA1401Tp1 frameworks and the RTI DS1552 I/O Extension Blockset in the same model.

Notes on DS1514 I/O Board's FPGA support

Implementing existing custom FPGA applications The RTI FPGA Programming Blockset let you use existing FPGA model INI files to build the processor interface.

The blockset support is limited to the Processor-Build model mode. You cannot simulate the processor interface.

Supported MicroAutoBox II variants for custom FPGA modeling For details on the dSPACE platforms supported by the RTI FPGA Programming Blockset, refer to [Hardware Supported by the RTI FPGA Programming Blockset](#) (RTI FPGA Programming Blockset Guide ).

Related topics

Basics

[RTI FPGA Programming Blockset Guide](#)

References

[RTI DS1552 I/O Extension Blockset \(MicroAutoBox II RTI Reference !\[\]\(642aa997563f9a325b310230bb5078b7_img.jpg\)\)](#)

Accessing the FPGA Type 1 Unit

Introduction

There are some general FPGA access functions to be mentioned which are independent from the implemented FPGA application.

General information on the access functions

Module initialization Before the initialization of the FPGA Type 1 module, the processor board must be initialized and an FPGA application must be running. Because the module initialization is terminated if there is no FPGA application running, one must be programmed beforehand or loaded to the flash memory of the FPGA module. Module initialization succeeds only if the FPGA application is compatible with the processor application.

Identification To avoid hardware damage, the components used (processor application, FPGA application, FPGA framework, piggyback module) must be compatible with each other. To check their compatibility, their identifiers can be read and compared using the identification functions.

Interrupt handling The FPGA Type 1 module provides 8 interrupt channels which you can handle using the RTLib interrupt functions, refer to [Interrupt Functions \(MicroAutoBox II RTLib Reference !\[\]\(274fd520e03b61c1b9ffc861754cacdc_img.jpg\)\)](#).

Data exchange With the RTLib, you can implement the processor's read and write access to the data storage that is defined in the FPGA framework. Data exchange with the FPGA application requires data type conversion, which is configured by the **scaling** and **mode** parameters.

The FPGA Type 1 module provides the following channel types and channel numbers:

- 32 Buffer In channels
- 32 Buffer Out channels

- 128 Register In channels
- 128 Register Out channels

The maximum number of elements in a buffer is specified in the FPGA framework. The configuration of a buffer is valid for all its elements. Buffers are accessed sequentially from the FPGA.

A register has a data width of 32 or 64 bits. You can specify groups of registers, which can be accessed synchronously by the FPGA. Each register in a group is configured separately.

Programming The FPGA application that you build, must be programmed to the FPGA. You can do this by loading it into the flash memory of the FPGA module or into the RAM of the FPGA. While the RAM of the FPGA must be programmed on each power up, the flash application is automatically loaded to the FPGA when you power up the board. Autoboosting from flash can be disabled.

Note

You can choose the autoboot state on the FPGA Type 1 page of the MicroAutoBox II Configuration Tool.

Note, that you can set the autoboot state only, if you have exclusive access to the connected board.

For further information on the tool, refer to [How to Change the IP Address of MicroAutoBox II \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(e8fb589d58dad1692debababa5e928b6_img.jpg\)](#)).

The FPGA application is represented by a programming data structure that contains the bitstream and further relevant information. For further information, refer to [FPGA Programming Functions \(MicroAutoBox II RTLib Reference !\[\]\(f95dab70c751fda7d824b8b03650f7aa_img.jpg\)](#))

RTLib and RTI FPGA Programming Blockset

These access functions are available via RTLib, refer to [FPGA Module Access Functions \(MicroAutoBox II RTLib Reference !\[\]\(d8ab143e904bfa3467271eec5af75a9b_img.jpg\)](#))

If you have installed the RTI FPGA Programming Blockset, you can also use its Processor Interface sublibrary to implement the FPGA access functions, for example, the data access between the processor model and the FPGA model. With the PROC_SETUP_BL block, you can manage your FPGA applications. Internally, these RTI blocks are using the RTLib functions for the FPGA Type 1 unit.

Serial Interface

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	✓	✓	✓	–

Where to go from here

Information in this section

[Basics on Serial Interface.....](#) 329

[Specifying the Baud Rate of the Serial Interface.....](#) 331

Provides information on the baud rate that you can specify for the board's serial interface.

Basics on Serial Interface

Characteristics

MicroAutoBox II is equipped with a serial interface with two channels. These channels are fixed to interface types that have different maximum baudrates, see the following table:

Channel	Interface	Max. Baudrate
Channel 0 (Serial 1)	RS232	115.2 kBaud
Channel 1 (Serial 2)	K line	50 kBaud

For details, see [Specifying the Baud Rate of the Serial Interface](#) on page 331.

The MicroAutoBox II variants provide one, two or three CAN_TP1 modules. If there are two or three modules, they can be used either for serial communication or for LIN.

MicroAutoBox II Variant	Number of CAN_TP1 Modules
MicroAutoBox II 1401/1507	2
MicroAutoBox II 1401/1511	2
MicroAutoBox II 1401/1511/1514	2
MicroAutoBox II 1401/1513	3
MicroAutoBox II 1401/1513/1514	3

I/O mapping

The following tables show the mappings of converter and channel numbers, as used in RTI and RTLib, to the related I/O pins of the related I/O connector.

▪ MicroAutoBox II 1401/1507

Unit	RTI Channel	RTLib Channel	Signal	I/O Connector Pin (Sub-D Connector)
CAN_TP1 Module1	1	0	Serial 1 TX out	7
			Serial 1 RX in	8
	2	1	Serial 2 K	10
			Serial 2 L	11
CAN_TP1 Module2	1	0	Serial 3 TX out	46
			Serial 3 RX in	47
	2	1	Serial 4 K	49
			Serial 4 L	50

▪ MicroAutoBox II 1401/1511 and MicroAutoBox II 1401/1511/1514

Unit	RTI Channel	RTLib Channel	Signal	I/O Connector Pin (DS1511 ZIF Connector)
CAN_TP1 Module1	1	0	Serial 1 TX out	c5
			Serial 1 RX in	c6
	2	1	Serial 2 K	b5
			Serial 2 L	b6
CAN_TP1 Module2	1	0	Serial 3 TX out	B5
			Serial 3 RX in	B6
	2	1	Serial 4 K	A5
			Serial 4 L	A6

- MicroAutoBox II 1401/1513 and MicroAutoBox II 1401/1513/1514

Unit	RTI Channel	RTLib Channel	Signal	I/O Connector Pin (DS1513 ZIF Connector)
CAN_TP1 Module1	1	0	Serial 1 TX out	c5
	2	1	Serial 1 RX in	c6
CAN_TP1 Module2	1	0	Serial 2 K	b5
	2	1	Serial 2 L	b6
CAN_TP1 Module3	1	0	Serial 3 TX out	B5
	2	1	Serial 3 RX in	B6
CAN_TP1 Module4	1	0	Serial 4 K	A5
	2	1	Serial 4 L	A6
CAN_TP1 Module5	1	0	Serial 5 TX out	P5
	2	1	Serial 5 RX in	P6
CAN_TP1 Module6	1	0	Serial 6 K	N5
	2	1	Serial 6 L	N6

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1507: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(aca6fcc8bd95e8255b9ea1b1d08ef300_img.jpg\)](#))
- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(0083087c61cec498ac803a4aec5bb1bd_img.jpg\)](#))
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(2e94242fda9f31152eb2b29146bfce46_img.jpg\)](#))
- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(680c68b4e62fe5ec9774c1168e904fbf_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(0012cbbec5c5a1cf6c111135ad58ebc0_img.jpg\)](#))

Specifying the Baud Rate of the Serial Interface

Oscillator frequency

The serial interface of MicroAutoBox II is driven by an oscillator with a frequency $f_{osc} = 3.6864 \text{ MHz}$.

Baud rate range

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

Mode	Baud Rate Range
RS232	5 ... 115,200 baud
K-line	5 ... 38,400 baud

Available baud rates

Using RTI and RTLib, you can specify any baud rate in the range listed above. However, the baud rate used by MicroAutoBox II depends on the oscillator frequency f_{osc} since the baud rate is a fraction of f_{osc} . The available baud rates can be calculated according to

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

where n is a positive integer.

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 20,000 baud as the baud rate, the baud rate used is 19,200 baud.

Single Edge Nibble Transmission (SENT) Support

Introduction

You can implement the SENT protocol on MicroAutoBox II using RTI blocks or RTLib functions.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	✓	✓	–

Where to go from here

Information in this section

Basics of the SENT Protocol.....	334
Provides basic information on the SENT (single edge nibble transmission) protocol.	
Basics on SENT Diagnostic Information.....	337
Provides information on the diagnostic information of a received message.	
Using the SENT Protocol on MicroAutoBox II.....	338
Provides general information on implementing the SENT protocol.	
Implementing SENT Receivers in Simulink.....	342
Provides information on how you can implement a SENT receiver in a Simulink model.	

Implementing SENT Receivers Using RTLib Functions..... 345

Provides information on the parameters for a SENT receiver and explains how you can implement it in a handcoded model.

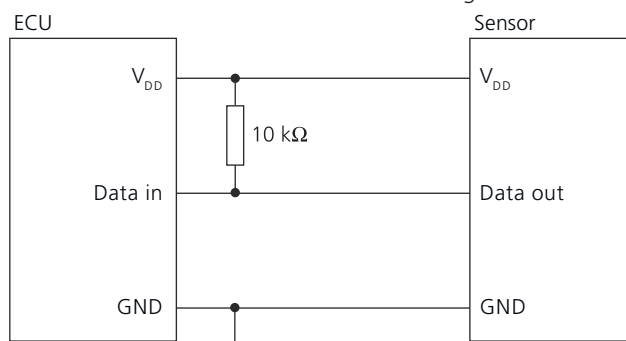
Basics of the SENT Protocol

Basics

SENT (Single Edge Nibble Transmission) is a protocol used between sensors and ECUs. It is defined in the SAE J2716 standard defined by the Society of Automotive Engineers (SAE). It is used to transmit data of high-resolution (10 bits or more) sensors as an alternative to an analog interface. The sensor signal is transmitted as a series of pulses with data encoded as the distance between two consecutive falling edges.

SENT connection

The following illustration shows the connection of a sensor to an ECU for using the SENT protocol. Usually, three lines (V_{DD} , Data, GND) are used to connect ECU and sensor. V_{DD} and GND is the power supply for the sensor (usually 5 V), the Data line is used to transmit the SENT messages.



Signal of a SENT message

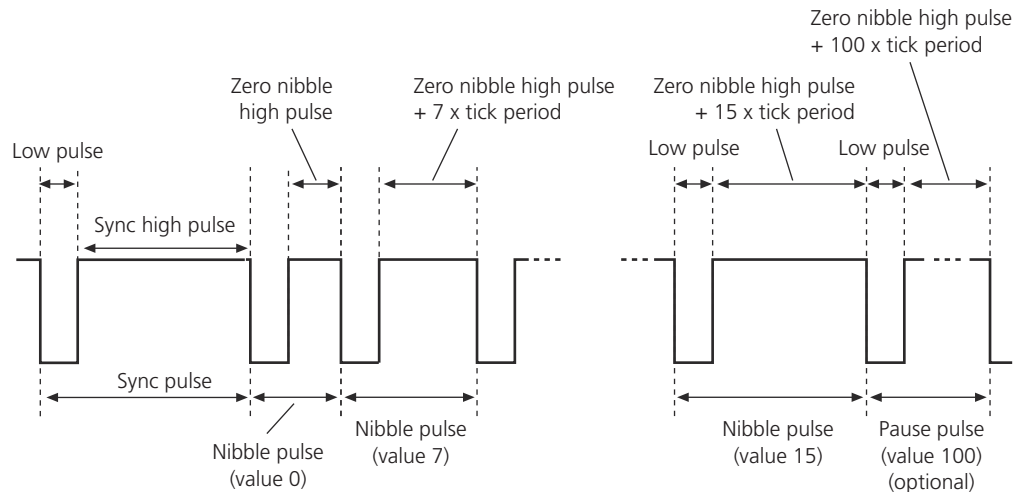
Every SENT message consists of a sync (synchronization) pulse followed by one or more nibble pulses. A nibble represents 4 bit. For regular SENT specification, the first nibble is a status nibble and the last nibble is a CRC nibble. All other nibbles contain data information.

An optional pause pulse (transmitted at the end of the SENT message) can be used, for example, to create SENT messages with a constant length.

The number of nibbles (nibble count), including the status and CRC nibble pulses, is constant for one specific SENT sensor but can vary between different sensors.

Data information is transmitted via the length of nibble pulses. One SENT nibble pulse consists of one low pulse and one high pulse. The value of a nibble is encoded by the length of a nibble pulse.

The following illustration shows a SENT message with relevant timings and nibble pulses transferring the values 0, 7 and 15 and a pause pulse with a value of 100.



Tick period

The lengths of the pulses are based on a clock rate, the tick period. Every SENT pulse is a multiple of this pulse duration. The standard value is 3 μ s.

At the beginning of each message, the SENT transmitter sends a sync pulse of a defined number of tick periods. The SENT receiver measures the length of the sync pulse and calculates the current length of the transmitter tick period.

Tick period tolerance (clock drift) A percentage that defines the maximum possible clock drift of a SENT transmitter. The standard value is up to $\pm 20\%$.

Pulses of a SENT message

A SENT message provides the following pulses.

SENT pulse A digital signal pulse consisting of a low pulse and a high pulse. There are three different types of SENT pulses, a sync pulse, a nibble pulse and a pause pulse. The meaning of a SENT pulse is detected by its pulse duration and its position in the message. The pulse length is measured between two consecutive falling edges.

Low pulse A pulse that marks the beginning of every SENT pulse. Its length is defined by the number of tick periods. The standard value is 5 tick periods.

High pulse The active part of a SENT pulse. It has two different purposes (zero nibble high pulse, sync high pulse). Its length is defined by the number of tick periods.

Sync high pulse The high part of a SENT synchronization pulse. Its length is defined by the number of tick periods. The standard value is 51 tick periods.

Zero nibble high pulse The high part of a SENT nibble pulse with a value of 0. Its length is defined by the number of tick periods. The standard value is 7 tick periods.

Sync pulse The beginning of every SENT message. It consists of a low pulse followed by a sync high pulse. Its length is defined by the number of tick periods. The pulse length must be longer than the maximum possible duration of a nibble pulse.

Nibble pulse A SENT pulse consisting of a low pulse and a specific high pulse with a minimum length of zero nibble high pulse. The nibble pulse length is defined by the number of tick periods. Every nibble pulse transmits the information of four bits. The pulse length is measured between two consecutive falling edges.

Pause pulse An optional fill pulse consisting of one low pulse and one high pulse with the variable length of n tick periods. It is transmitted at the end of a SENT message after the CRC nibble pulse. The pause pulse can be used, for example, to create SENT messages with a constant number of tick periods.

Maximum time between two read operations

If data is lost, the reason might be that the time between two read operations, i.e., executions of the `DIO_TYPE3_SENT_RX_BLx` block, is too long. The maximum time between two read operations without data loss can be calculated as follows:

$T_{\text{Read, max}}$	$= \text{Expected number of messages} \cdot T_{\text{Message, min}}$
$T_{\text{Message, min}}$ (without pause pulse)	$= [(\text{LowTics} + \text{SyncHighTics}) + \text{NibbleCount} \cdot (\text{LowTics} + \text{ZeroNibbleHighTics})] \cdot [(1 - \text{CD}) \cdot \text{TicPeriod}]$
$T_{\text{Message, min}}$ (with pause pulse)	$= [(\text{LowTics} + \text{SyncHighTics}) + (\text{NibbleCount} + 1) \cdot (\text{LowTics} + \text{ZeroNibbleHighTics})] \cdot [(1 - \text{CD}) \cdot \text{TicPeriod}]$
$T_{\text{Read, max}}$: Maximum time between two consecutive read operations without loss of data. $T_{\text{Message, min}}$: Minimum possible message duration LowTics: Number of ticks for a low pulse SyncHighTics: Number of ticks for a synchronization high pulse NibbleCount: Number of nibbles included in every SENT message ZeroNibbleHighTics: Number of ticks for the high part of a SENT nibble pulse with a value of 0. CD: Clock drift (tick period tolerance) TicPeriod: Tick period	

Example:

The following table shows some values of $T_{\text{Read, max}}$ for different SENT message data if Expected number of messages = 20 and pause pulse mode is disabled:

TicPeriod	Tics			NibbleCount	CD	$T_{\text{Read, max.}}$
	SyncHigh	ZeroNibbleHigh	Low			
1 μs	51	7	5	6	0.2	2.0 ms
3 μs	51	7	5	6	0.2	6.1 ms
3 μs	51	7	5	10	0.2	8.4 ms
3 μs	51	7	5	6	0.3	5.3 ms

If the time between the read operation is higher than $T_{\text{Read, max}}$, the message buffer might not be able to store all the received messages and the newest messages get lost.

Tip

If pause mode is enabled, and a fixed message length > 0 has been specified, $T_{\text{Read, max}}$ can be calculated as follows:

$$T_{\text{Read, max}} = \text{ExpectedMsgLen} \cdot \text{TicPeriod} \cdot (1 - \text{CD})$$

Basics on SENT Diagnostic Information

Introduction

The Diagnostic parameter/output provides a diagnostic word for each received message. The diagnostic word consists of flags for different message-specific status and diagnostic information.

Diagnostic flags

The flags of the diagnostic word have the following meanings:

Bit	Value	Description
0	1	Too many nibbles in message. This value is returned when too many nibbles are received in a message. Each message is stored with the number of nibbles as specified by the Number of nibbles (incl. status, CRC) block parameter or NibbleCount function parameter. The surplus nibbles are not stored in the message but used for diagnostics. For example, the <i>Nibble value is out of range</i> flag is also set if one of the surplus nibbles have a nibble value > 15 .
1	2	Too few nibbles in message. This value is returned when a message with too few nibbles is received. The missing nibbles are marked with a value of "-128" (RTLib: DIO_TP_SENT_RX_MISSING_NIBBLE). This ensures that each message is stored with the number of nibbles as specified by the Number of nibbles (incl. status, CRC) block parameter or NibbleCount function parameter.
2	4	Nibble value is out of range [0 ... 15]. This value is returned when a nibble with a value < 0 or > 15 is received. The nibble is saved to the data buffer anyway.
3	8	Synchronization pulse too long. This value is returned when a synchronization pulse is larger than the upper limit of the expected tick period specified by the Tick period and Tick period tolerance block parameters or the TickPeriod and ClockDrift function parameters. The nibble values are evaluated despite this, but the result will be erroneous.
4	16	Synchronization pulse too short. This value is returned when a synchronization pulse is shorter than the lower limit of the expected tick period specified by the Tick period and Tick period tolerance block parameters or the TickPeriod and ClockDrift function parameters. The nibble values are evaluated despite this, but the result will be erroneous.

Bit	Value	Description
5	32	The current synchronization pulse differs from the last synchronization pulse by a factor of more than 1/64. The last received message has to be ignored.
6	64	Message has not the expected length. This value is returned when the pause pulse option is enabled and a fixed message length is specified. The diagnostic information occurs if the length of the received message differs from the specified expected message length.
7	128	Deviation of sync pulse to message length ratio too high. This value is returned when the pause pulse option is enabled and a fixed message length is specified. The diagnostic information occurs if the ratio from sync pulse and fixed message length differs by a factor of more than 1/64.

Using the SENT Protocol on MicroAutoBox II

Introduction

You can implement the SENT (single edge nibble transmission) protocol on MicroAutoBox II using RTI blocks or RTLib functions.

Receiving SENT messages on MicroAutoBox II

There are four independent SENT receivers on MicroAutoBox II. Each of the available digital inputs on the DIO Type 3 or DIO Type 4 units can be specified as a data channel for SENT receiver.

The CRC nibble and the status nibble of a SENT message are not evaluated by the receiver. You must evaluate them in your model.

Electrical connection A SENT receiver is connected to the SENT transmitter via three lines:

- **V_{DD} line:**
The V_{DD} line provides the power supply for the sensor, usually 5 V.
- **GND line:**
The GND line must be connected to the ground of the power supply used and the GND of MicroAutoBox II.
- **Data line:**

The data line must be connected to the channel which is used for receiving.

To use the electrical connection, some configuration must be done via software, see [Implementing SENT Receivers in Simulink](#) on page 342 or [Implementing SENT Receivers Using RTLib Functions](#) on page 345.

Range of pulse length

MicroAutoBox II can measure low pulses and high pulses in the range 300 ns ... 209 ms divided into eight subranges. The connected transmitter has to match the selected range, otherwise data gets lost or pulses cannot be decoded correctly.

Range Number	Pulse Length Range	Resolution
1	300 ns ... 1.63 ms	50 ns
2	600 ns ... 3.27 ms	100 ns
3	1.2 μ s ... 6.55 ms	200 ns
4	2.4 μ s ... 13.1 ms	400 ns
5	4.8 μ s ... 26.2 ms	800 ns
6	9.6 μ s ... 52.4 ms	1.6 μ s
7	19.2 μ s ... 104 ms	3.2 μ s
8	38.4 μ s ... 209 ms	6.4 μ s

Note

The measurement resolution depends on the selected pulse length range. To get the best possible resolution of the measured pulses, you should select the range with the best possible resolution (the pulse length range with the lowest possible range number). For example, if your desired pulse length is 3 μ s, you should use pulse length range 1 (300 ns ... 1.63 ms) rather than pulse length range 2 (600 ns ... 3.27 ms).

I/O mapping

For MicroAutoBox II variants with DS1511 I/O board The following table shows the mapping of the channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector (DS1511 ZIF connector):

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	V2
	2	DigP 1 ch 2	U2
	3	DigP 1 ch 3	T2
	4	DigP 1 ch 4	S2
	5	DigP 1 ch 5	R2
	6	DigP 1 ch 6	P2
	7	DigP 1 ch 7	N2
	8	DigP 1 ch 8	M2
	9	DigP 1 ch 9	V3
	10	DigP 1 ch 10	U3
	11	DigP 1 ch 11	T3
	12	DigP 1 ch 12	S3
	13	DigP 1 ch 13	R3
	14	DigP 1 ch 14	P3
	15	DigP 1 ch 15	N3
	16	DigP 1 ch 16	M3

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	V4
	2	DigP 2 ch 2	U4
	3	DigP 2 ch 3	T4
	4	DigP 2 ch 4	S4
	5	DigP 2 ch 5	R4
	6	DigP 2 ch 6	P4
	7	DigP 2 ch 7	N4
	8	DigP 2 ch 8	M4
	9	DigP 2 ch 9	V5
	10	DigP 2 ch 10	U5
	11	DigP 2 ch 11	T5
	12	DigP 2 ch 12	S5
	13	DigP 2 ch 13	R5
	14	DigP 2 ch 14	P5
	15	DigP 2 ch 15	N5
	16	DigP 2 ch 16	M5
3	1	DigP 3 ch 1	V6
	2	DigP 3 ch 2	U6
	3	DigP 3 ch 3	T6
	4	DigP 3 ch 4	S6
	5	DigP 3 ch 5	R6
	6	DigP 3 ch 6	P6
	7	DigP 3 ch 7	N6
	8	DigP 3 ch 8	M6

¹⁾ DigP <Port number> ch <Channel number>

For MicroAutoBox II variants with DS1513 I/O board The following table shows the mapping of the channel numbers, as used in RTI and RTLib, to the related I/O pins of the MicroAutoBox II I/O connector (DS1513 ZIF connector):

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	M2
	2	DigP 1 ch 2	L2
	3	DigP 1 ch 3	K2
	4	DigP 1 ch 4	J2
	5	DigP 1 ch 5	H2
	6	DigP 1 ch 6	M3
	7	DigP 1 ch 7	L3
	8	DigP 1 ch 8	K3
	9	DigP 1 ch 9	J3
	10	DigP 1 ch 10	H3
	11	DigP 1 ch 11	M4
	12	DigP 1 ch 12	L4
	13	DigP 1 ch 13	K4
	14	DigP 1 ch 14	J4
	15	DigP 1 ch 15	H4
	16	DigP 1 ch 16	M5
2	1	DigP 2 ch 1	L5
	2	DigP 2 ch 2	K5
	3	DigP 2 ch 3	J5
	4	DigP 2 ch 4	H5
	5	DigP 2 ch 5	M6
	6	DigP 2 ch 6	L6
	7	DigP 2 ch 7	K6
	8	DigP 2 ch 8	J6

¹⁾ DigP <Port number> ch <Channel number>

Note

Concurrent access to the same digital input channel by other DIO Type 3 or DIO Type 4 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(9a53fe79a03d38d8322f7a2c5a875b36_img.jpg\)\)](#)
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(01f19d40f03100aa8a158c4891453b0d_img.jpg\)\)](#)
- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(e08cd99387e13601e6c12f535030ab90_img.jpg\)\)](#)
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(e3c5fe615c12e7c56b62fb195faeae4a_img.jpg\)\)](#)

Implementing SENT Receivers in Simulink

Introduction

RTI provides one block for implementing a SENT receiver in a Simulink model. The `DIO_TYPE3_SENT_RX_BLx` or `DIO_TYPE4_SENT_RX_BLx` block reads the messages stored in the receive FIFO so that their nibbles are available in the Simulink model.

Configuring the channels

To configure channels for receiving SENT messages defined by the SENT specification, you must perform some preparatory steps.

- The channels must be connected to the SENT transmitters, see [Using the SENT Protocol on MicroAutoBox II](#) on page 338.
- For each channel which is used as a SENT receiver, drag a `DIO_TYPE3_SENT_RX_BLx` or `DIO_TYPE4_SENT_RX_BLx` block into your Simulink model and specify the module number, port and channel on the Unit page.

Specifying the properties of a SENT signal

The properties of a SENT signal are specified on the RX Parameters page of the `DIO_TYPE3_SENT_RX_BLx` or `DIO_TYPE4_SENT_RX_BLx` block.

Tick period The expected pulse length of the tick period and the tick period tolerance can be specified on the RX Parameters page. This is the base clock every SENT pulse is generated with. As MicroAutoBox II calculates the current tick period of the transmitter whenever a message is received, the block can set an error flag if the tick period is outside the specified tolerance.

Low pulse, zero nibble high pulse, sync high pulse These SENT-specific configurable parameters are defined by a number of tick periods on the RX Parameters page.

Pause pulse If the messages to be received are containing pause pulses, you can configure the SENT receiver block to treat the last pulse of a SENT message as a pause pulse. Because SENT transmitters usually use pause pulses to achieve a constant message length, you can specify the expected length of a message on the RX Parameters page. However, a pause pulse can also be detected, if checking the message length is disabled (message length is set to 0).

Receiving messages and nibbles

Expected number of messages You have to specify the number of SENT messages which can be stored in the receive FIFO. This number also defines the size of the `DIO_TYPE3_SENT_RX_BLx` or `DIO_TYPE4_SENT_RX_BLx` block's receive FIFO. If the receive FIFO runs full, the newest messages are lost. The FIFO will not be overwritten by new input data. In addition an appropriate error flag is set. The number of messages which are currently stored in the receive FIFO is indicated by the Count output.

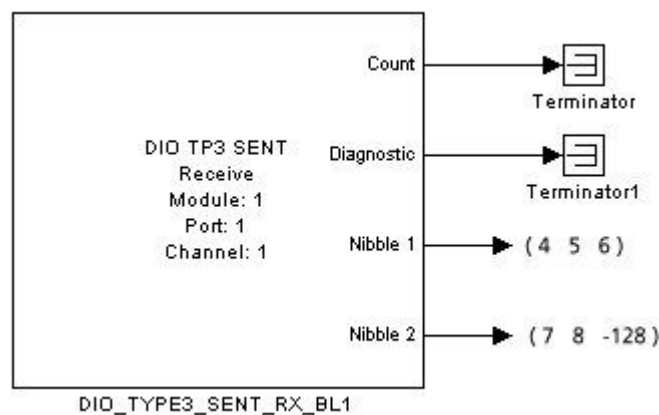
Number of nibbles The number of nibbles includes the status nibble and CRC nibble which are treated as data nibbles. The `DIO_TYPE3_SENT_RX_BLx` or `DIO_TYPE4_SENT_RX_BLx` block gets outputs for each nibble. The Nibble *n*

outport gives a vector with the nibble values of all received messages at the n position. The signal connected to the Nibble outputs must be a vector with a length of the specified expected number of messages. However, the number of nibbles at a Nibble output matches only the number of messages which are actually received (and indicated by the Count output).

Read mode The block supports different ways of reading messages:

- The block reads all new complete received messages and diagnostic information from the receive FIFO since the last read operation.
- The block reads the most recent message and diagnostic information from the receive FIFO. The receive FIFO is then cleared.

Example The following illustration shows an example of a SENT receiver.



In the example the messages have two nibbles. Three messages are read from the receive FIFO. The first message has the nibble values 4 and 7, the second message has 5 and 8. The third message has only one nibble with the value 6. For the missing nibble, a value of '-128' is returned and the error flag in the vector of the Diagnostic output is set.

Reading received SENT messages

During run time, your application can read the received messages stored in the receive FIFO in two different ways:

- Reading periodically triggered by a timer task

During run time the timer task calls the **DIO_TYPE3_SENT_RX_BLx** or **DIO_TYPE4_SENT_RX_BLx** block. This has to be done periodically, for example, every 1.5 ms. To avoid losing received messages, the model cycle should not be longer than the maximum recommended time. For details, refer to [Avoiding data loss](#) on page 344.

The block transfers the new data received since the last read operation to the model. In addition a diagnostic word with diagnostic information is created for each received SENT message. For details, refer to [Basics on SENT Diagnostic Information](#) on page 337.

- Reading asynchronously triggered by an interrupt

To use this method you have to enable interrupt generation by the **Enable Interrupt** parameter of the block. You can specify this option on the **Interrupt** page.

In this case, an interrupt is generated on MicroAutoBox II after a certain number of SENT messages were received, and stored in the receive FIFO. Inside the interrupt service routine, the `DIO_TYPE3_SENT_RX_BLx` or `DIO_TYPE4_SENT_RX_BLx` block can be called to transfer the received message (including diagnostic information) to the model. You can specify the transfer interval via the **Number of messages to trigger interrupt** parameter.

For example, if **Number of messages to trigger interrupt** is set to 6, an interrupt is triggered on MicroAutoBox II after the reception of every 6th message and these last 6 received SENT messages are transferred to the model with the block call.

Avoiding data loss

To avoid losing received messages due to a full receive FIFO, the `DIO_TYPE3_SENT_RX_BLx` or `DIO_TYPE4_SENT_RX_BLx` block must be executed periodically.

If the receive FIFO runs full, the newest messages are lost. The FIFO will not be overwritten by new input data. In addition an appropriate error flag is set. This leads to loss of nibbles or messages. When the `DIO_TYPE3_SENT_RX_BLx` or `DIO_TYPE4_SENT_RX_BLx` block is executed (for example, in a timer task), the block's receive FIFO is read out and new messages can be received again. The receive FIFO can store as many complete SENT messages as specified by the **Expected number of messages (FIFO size)** parameter.

Maximal number of messages The number of messages that can be maximally buffered between two read operations can be calculated as follows:

$$N_{MessagesMax} = RoundDown\left(\frac{4096}{N_{Nibbles} + 1 + PausePulseEnabled}\right)$$

With

$N_{MessagesMax}$	Number of messages which can maximally be buffered
$N_{Nibbles}$	Number of nibbles in each message
<code>PausePulseEnabled</code>	<ul style="list-style-type: none"> ▪ 0, if Pause mode is disabled ▪ 1, if Pause mode is enabled

Maximum time between two read operations If data gets lost, the time between two read operations is too long. The allowed maximum time between two read operations without loss of messages can be calculated as described in [Basics of the SENT Protocol](#) on page 334.

Getting diagnostic information

The `DIO_TYPE3_SENT_RX_BLx` or `DIO_TYPE4_SENT_RX_BLx` block has three outputs which give you status or further information. The outputs are optional and must be enabled on the **Advanced** page.

Tick period The **Tick Period** output provides the actual tick period which is calculated from the last received valid synchronization pulse.

Diagnostic The **Diagnostic** output provides a diagnostic word for each received message. The diagnostic word consists of flags for different message-

specific status and diagnostic information. For details on the diagnostic flags, refer to [Basics on SENT Diagnostic Information](#) on page 337.

Error The Error output provides information on the current read operation. The values have the following meanings:

- 0: No data loss
- 1: Data loss, more messages were received than expected
- 2: Timeout, a SENT pulse was longer than the maximum measurable pulse length
- 3: Combination of data loss and timeout

Related topics

References

[DIO_TYPE3_SENT_RX_BLx \(MicroAutoBox II RTI Reference !\[\]\(e3f8612927870f2e0f9f5989e6dd3064_img.jpg\)\)](#)
[DIO_TYPE4_SENT_RX_BLx \(MicroAutoBox II RTI Reference !\[\]\(a86c7d1c9cb81c81614634a31267440d_img.jpg\)\)](#)

Implementing SENT Receivers Using RTLib Functions

Channel configuration

To configure channels for receiving SENT messages defined by the regular SENT specification, you must perform some preparatory steps:

- The channels must be connected to the SENT transmitters, see [Using the SENT Protocol on MicroAutoBox II](#) on page 338.
- The following functions must be called beforehand when you use a MicroAutoBox II variant with DS1511 I/O board:

- **dio_tp3_init**
- Optional: **dio_tp3_single_source_int_mode_set** or **dio_tp3_multi_source_int_mode_set**

The following functions must be called beforehand when you use a MicroAutoBox II variant with DS1513 I/O board:

- **dio_tp4_init**
- Optional: **dio_tp4_single_source_int_mode_set** or **dio_tp4_multi_source_int_mode_set**

Configuration of a SENT receiver

Every SENT receiver must be configured with the following parameters before receiving messages:

Parameter	Type	Description
NibbleCount	Initialization	Defines the number of nibbles per SENT message. This number is constant and is set during initialization of the SENT receiver in the range 1 ... 64.
LowTickCount	Initialization	Defines the length of a low pulse in ticks in the range 1 ... 15. The standard value is 5 tick periods.

Parameter	Type	Description
SyncHighTickCount	Initialization	Defines the length of the high pulse of a synchronization pulse in ticks in the range 1 ... 255. The standard value is 51 tick periods.
ZeroNibbleHighTickCount	Initialization	Defines the high pulse length of a nibble with value 0 in ticks in the range 1 ... 15. The standard value is 7 tick periods.
TickPeriod	Initialization	Defines the expected length of a tick period of a SENT pulse in seconds. This is the base clock every SENT pulse is generated with in the range 300 ns ... 200 μ s, and a resolution of 50 ns up to 6.4 μ s. The standard tick period value is 3 μ s.
ClockDrift	Initialization	Defines the tick period tolerance (clock drift) that the SENT receiver accepts as valid drift. Synchronization pulses and nibble pulses are recognized as valid pulses within this range. Pulses outside this specified range are recognized as invalid synchronization pulses or as nibble pulses with an unexpected value. When an invalid synchronization pulse is received, the current message is not stored and the receiver waits for the next valid synchronization pulse. The diagnostic port reports an invalid synchronization pulse. The range of clockdrift is 0 ... 0.5. The standard value is 0.2 ($\pm 20\%$).
Range	Initialization	Defines the pulse range of the SENT receiver within the range 400 ns ... 209 ms divided into eight subranges.
InterruptRate	Initialization	Defines the number of messages after an interrupt is generated. The value is in the range 1 ... 128 messages. The value must be less than the number of expected messages.
PauseMode	Initialization	Defines whether pause pulses are to be considered in the received SENT messages.
FixedMsgLength	Initialization	Defines the fixed message length of each message. The value is relevant only, if pause mode is enabled. A pause pulse is detected even if you set the fixed message length to 0. This means, that no fixed message length is expected.

Continuous reception of messages

The SENT specification requires continuous reception of messages. To avoid losing received messages due to a full input buffer, regular execution of the receive functions is required. If a receive buffer runs full, all further pulses are ignored. This leads to loss of nibbles or messages. When a read operation is executed, the receive buffer is read out and new messages can be received again. The internal FIFO can store a maximum of 4096 pulses each with a high part and a low part.

If data gets lost, the time between two read operations is too long. The allowed maximum time between two read operations without loss of messages can be calculated as described in [Basics of the SENT Protocol](#) on page 334.

Format of messages

Messages are delivered to the model as a vector of `Int8 rx_data[100]`.

```
Int8 rx_data[100];
```

The driver writes received nibbles to the data buffer one after the other from the first nibble of the first message to the last nibble of the last message. The vector looks like this (*nibble_count* is the number of nibbles per message):

<code>rx_data[0]</code>	Message 1, nibble 1
<code>rx_data[1]</code>	Message 1, nibble 2
...	...
<code>rx_data[NibbleCount - 1]</code>	Message 1, nibble <i>NibbleCount</i>
<code>rx_data[NibbleCount]</code>	Message 2, nibble 1
...	...
<code>rx_data[2*NibbleCount - 1]</code>	Message 2, nibble <i>NibbleCount</i>
...	...
<code>rx_data[(msg - 1) * NibbleCount + (nib - 1)]</code>	Message <i>msg</i> , nibble <i>nib</i>

The format for a 2-dimensional vector looks like this:

```
Int8 rx_data[NumMsg][NibbleCount];
```

The vector looks like this (*NibbleCount* is the number of nibbles per message):

<code>rx_data[0][0]</code>	Message 1, nibble 1
<code>rx_data[0][1]</code>	Message 1, nibble 2
...	...
<code>rx_data[0][NibbleCount - 1]</code>	Message 1, nibble <i>NibbleCount</i>
<code>rx_data[1][0]</code>	Message 2, nibble 1
...	...
<code>rx_data[1][NibbleCount - 1]</code>	Message 2, nibble <i>NibbleCount</i>
...	...
<code>rx_data[msg - 1][nib - 1]</code>	Message <i>msg</i> , nibble <i>nib</i>

Modes of receiving messages

The SENT receivers support two different modes of reading messages:

- Read All Mode:

If the **ReadMode** parameter of the **dio_tp3_sent_rx_receive** function is set to **DIO_TP3_SENT_RX_RECEIVE_ALL** or the parameter of the **dio_tp4_sent_rx_receive** function is set to **DIO_TP4_SENT_RX_RECEIVE_ALL**, this function reads all new messages received since the last read operation. When no complete new message is available, nothing is returned and the **ReceivedMsgCount** parameter is 0. To avoid writing more messages to the user data buffer than memory was allocated, the **ExpMsgCount** parameter is used. This parameter indicates the maximum number of messages that are written to the **Data** buffer. If the number of received messages exceeds **ExpMsgCount**, writing data to user buffer is aborted and the remaining messages are discarded to avoid an overflow of the internal data buffer. This is reported by a return value of **DIO_TP3_SENT_RX_DATA_LOSS** or **DIO_TP4_SENT_RX_DATA_LOSS**. If loss of data is recognized, the model cycle of reading SENT messages is too long. The model cycle should not be longer than the maximum recommended time (see [Continuous reception of messages](#) on page 346).

- Most Recent Mode:

If the **ReadMode** parameter of the **dio_tp3_sent_rx_receive** function is set to **DIO_TP3_SENT_RX_RECEIVE_MOST_RECENT** or the parameter of the **dio_tp4_sent_rx_receive** function is set to **DIO_TP4_SENT_RX_RECEIVE_MOST_RECENT**, this function reads the newest complete message. If no message was received at all, a message with all nibbles marked as missing nibbles **DIO_TP3_SENT_RX_MISSING_NIBBLE** or **DIO_TP4_SENT_RX_MISSING_NIBBLE** (-128) is returned and the **ReceivedMsgCount** parameter is 0.

Reading of received SENT messages

During run time, your application can read the received messages stored in the data buffer in two different ways:

- Reading triggered by a timer task

During run time the timer task calls a SENT receive function, for example, the **dio_tp3_sent_rx_receive** function. This is done periodically. To avoid losing received messages, the sample time should not be longer than the maximum recommended cycle time. For details, refer to Continuous reception of messages. The function transfers all new messages received since the last read operation to the model. A diagnostic word is also created for each received SENT message.

- Reading triggered by an interrupt

To use this method, you have to enable interrupt generation by the **IntMode** parameter during initialization. In this case, an interrupt is generated on MicroAutoBox II after the specified number of SENT messages is received and stored in the receive FIFO. In the interrupt service routine, the **dio_tp3_sent_rx_receive** or **dio_tp4_sent_rx_receive** function can be called to transfer the received message including diagnostic information to the model. You can implement downsampling by specifying the **InterruptRate** parameter. The interrupt rate may not exceed the specified value of the **ExpMsgCount** parameter.

For example, if **InterruptRate** is set to 6, an interrupt is triggered on MicroAutoBox II after every 6th message and these last 6 received SENT messages are transferred to the model by the **dio_tp3_sent_rx_receive** or **dio_tp4_sent_rx_receive** function call.

RTLib functions

For information on the RTLib functions used for programming a SENT receiver, refer to [Single Edge Nibble Transmission \(SENT\) on the DIO Type 3 \(MicroAutoBox II RTLib Reference !\[\]\(758ebdf4629c903da74c2e079717ae32_img.jpg\)](#)) or [Single Edge Nibble Transmission \(SENT\) on the DIO Type 4 \(MicroAutoBox II RTLib Reference !\[\]\(e7d82ae1e31b23b67694dcc1e3031ff6_img.jpg\)](#)).

Related topics**Basics**

[Using the SENT Protocol on MicroAutoBox II..... 338](#)

References

[dio_tp3_sent_rx_init \(MicroAutoBox II RTLib Reference !\[\]\(74d4806277d7e73349d8e8c0897931e9_img.jpg\)\)](#)

[dio_tp3_sent_rx_receive \(MicroAutoBox II RTLib Reference !\[\]\(0aff635c4179ba9e710b00f4b01d3b20_img.jpg\)\)](#)

Serial Peripheral Interface

Introduction MicroAutoBox II provides a serial peripheral interface (SPI) that can be used to perform high-speed synchronous communication with devices connected to the MicroAutoBox II Base board, such as an A/D converter.

Where to go from here	Information in this section
	Serial Peripheral Interface on the DIO Type 3 Unit..... 351
	Serial Peripheral Interface on the DIO Type 4 Unit..... 360

Serial Peripheral Interface on the DIO Type 3 Unit

Introduction The serial peripheral interface (SPI) of the DIO Type 3 module provides high-speed synchronous communication with devices connected to the MicroAutoBox II Base board, such as an A/D converter.

Hardware requirements Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards				
	DS1507	DS1511	DS1513	DS1514	
-	-	✓	-	-	

Characteristics

The DIO Type 3 unit of the DS1511 I/O board provides up to two serial peripheral interfaces (SPI).

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

Note

- The SPI can be processed only in master mode. It is unable to respond to any externally initiated serial transfers.
- SPI signals are usually TTL signals. Because of MicroAutoBox II's output stages, the signal voltage depends on the VDRIVE voltage, in cars, this is usually VBAT (12 V). In this case, the SPI signals of MicroAutoBox II cannot be directly connected to the external device.

The SPI supports the following features:

- Up to four chip select channels can be configured.
By using a multiplexer, you can access up to 15 independent peripherals.
- Up to 64 chip select cycle configurations per SPI unit can be configured.
During run time you only have to reference the specified number of a cycle configuration for transmitting or receiving SPI data.
A chip select cycle configuration is used to specify the following characteristics of an SPI transmission:
 - Transfer length by specifying the number of words (1 ... 64) and bits per word (1 ... 128). The transfer length must not exceed 2048 bits.
Data received by the SPI is stored temporarily in a FIFO buffer of the DS1511. Buffer overflows are indicated by a status information, and cause old data to be overwritten.
 - Bit direction in a word
 - Period of the clock signal defined by the specified baud rate in the range 306 Baud ... 300 kBaud.
 - Polarity and phase of the clock signal (SPI mode)
 - Timing behavior of the transmission by specifying the time before and after transfer, the minimum time between two chip select cycles and the time between words. For detailed information on the timing parameters, see below.
- Optional, generation of an *end of cycle* interrupt.

Timing behavior

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

- If you specify 0 for the time between data words, the transfer of each word of an SPI cycle follows the same timing parameters.

- If you specify a value in the range 25 ns ... 793.6 μ s for the time between data words, the other timing parameters only affect the timing behavior of the beginning and end of an entire SPI cycle. The chip select signal is not switched to the inactive state to mark the end of a word in the SPI cycle.

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

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

For an illustration of the timing behavior, see below.

The timing parameters can be specified with a maximum value of 793.6 μ s. The value range is internally separated into 12 intervals that are automatically assigned to the specified value. Each interval provides a different step size that is used to saturate a specified value to its next available value.

Interval Number	Lower Limit of Interval	Step Size
1	0 ns	12.5 ns
2	400 ns	25 ns
3	800 ns	50 ns
4	1.6 μ s	100 ns
5	3.2 μ s	200 ns
6	6.4 μ s	400 ns
7	12.8 μ s	800 ns
8	25.6 μ s	1.6 μ s
9	51.2 μ s	3.2 μ s
10	102.4 μ s	6.4 μ s
11	204.8 μ s	12.8 μ s
12	409.6 μ s	25.6 μ s

Note

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

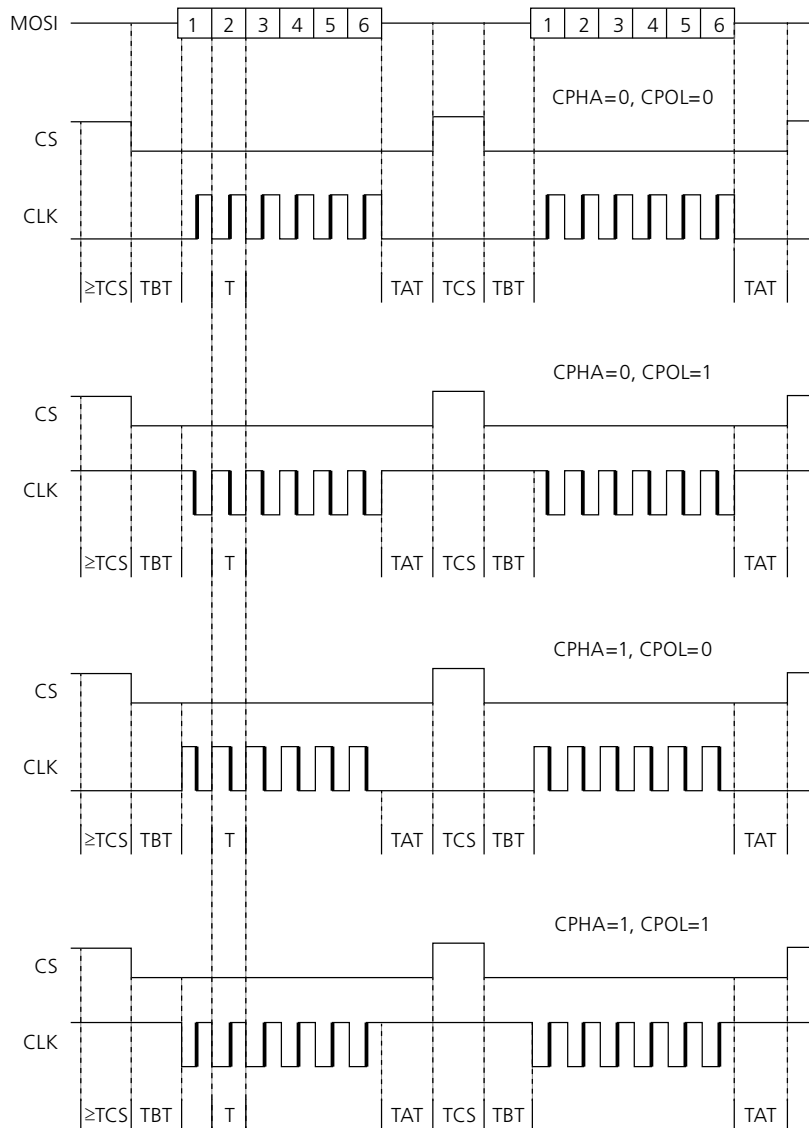
Examples of the timing behavior

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

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

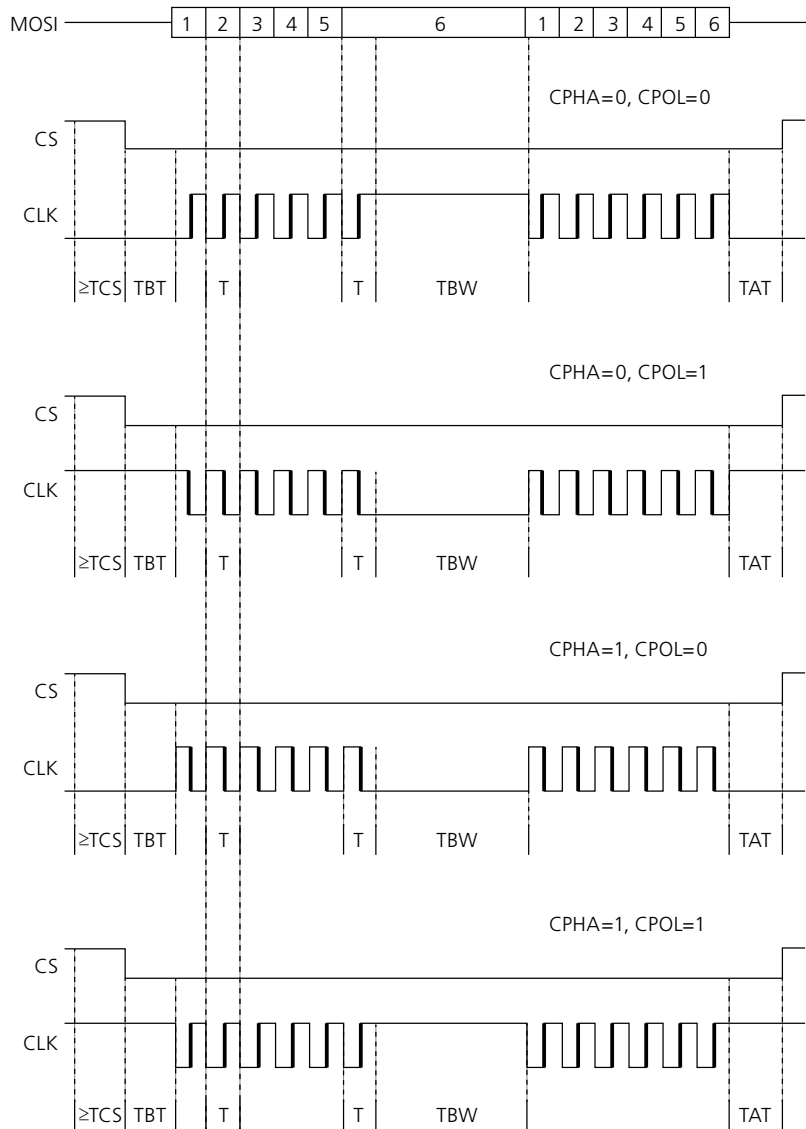
The following abbreviations are used:

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

Example with time between data words set to 0

The DIO Type 3 unit writes data to MOSI at the thin edges and reads data from MISO at the bold edges. While no data is transmitted, the MOSI channel is set to High-Z.

Example with time between data words set to >0



The DIO Type 3 unit writes data to MOSI at the thin edges and reads data from MISO at the bold edges. Between the words the last data bit is put out on the MOSI channel. While no data is transmitted, the MOSI channel is set to High-Z.

RTI/RTLib support

You have access to the serial peripheral interface on the DIO Type 3 module via RTI1401 and RTLib1401. For details, see:

- [Serial Peripheral Interface \(SPI\) \(MicroAutoBox II RTLib Reference\)](#)
- [Serial Peripheral Interface \(MicroAutoBox II RTI Reference\)](#)

I/O mapping

The following table shows the order of the signals. You have to specify the digital input channel (MISO) and the first digital output channel (CLK). The digital input channel can be configured as trigger source for interrupt generation. For the digital output channels, you can configure the electrical interface.

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

You can use one of the following digital input channels available on the DS1511 I/O connector to connect to the MISO signal.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	V2
	2	DigP 1 ch 2	U2
	3	DigP 1 ch 3	T2
	4	DigP 1 ch 4	S2
	5	DigP 1 ch 5	R2
	6	DigP 1 ch 6	P2
	7	DigP 1 ch 7	N2
	8	DigP 1 ch 8	M2
	9	DigP 1 ch 9	V3
	10	DigP 1 ch 10	U3
	11	DigP 1 ch 11	T3
	12	DigP 1 ch 12	S3
	13	DigP 1 ch 13	R3
	14	DigP 1 ch 14	P3
	15	DigP 1 ch 15	N3
	16	DigP 1 ch 16	M3

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	V4
	2	DigP 2 ch 2	U4
	3	DigP 2 ch 3	T4
	4	DigP 2 ch 4	S4
	5	DigP 2 ch 5	R4
	6	DigP 2 ch 6	P4
	7	DigP 2 ch 7	N4
	8	DigP 2 ch 8	M4
	9	DigP 2 ch 9	V5
	10	DigP 2 ch 10	U5
	11	DigP 2 ch 11	T5
	12	DigP 2 ch 12	S5
	13	DigP 2 ch 13	R5
	14	DigP 2 ch 14	P5
	15	DigP 2 ch 15	N5
	16	DigP 2 ch 16	M5
3	1	DigP 3 ch 1	V6
	2	DigP 3 ch 2	U6
	3	DigP 3 ch 3	T6
	4	DigP 3 ch 4	S6
	5	DigP 3 ch 5	R6
	6	DigP 3 ch 6	P6
	7	DigP 3 ch 7	N6
	8	DigP 3 ch 8	M6

¹⁾ DigP <Port number> ch <Channel number>

Note

Multiple access to the same digital input channel by other DIO_TYPE3 blocks or functions is not allowed.

You can use up to six subsequent digital output channels available on the DS1511 I/O connector to connect to the CLK, MOSI, and CS1 ... CS4 signals.

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

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	L2
	2	DigP 1 ch 2	K2
	3	DigP 1 ch 3	J2
	4	DigP 1 ch 4	H2
	5	DigP 1 ch 5	G2
	6	DigP 1 ch 6	F2
	7	DigP 1 ch 7	E2
	8	DigP 1 ch 8	D2
	9	DigP 1 ch 9	L3
	10	DigP 1 ch 10	K3
	11	DigP1 ch 11	J3
	12	DigP 1 ch 12	H3
	13	DigP 1 ch 13	G3
	14	DigP 1 ch 14	F3
	15	DigP 1 ch 15	E3
	16	DigP 1 ch 16	D3
2	1	DigP 2 ch 1	L4
	2	DigP 2 ch 2	K4
	3	DigP 2 ch 3	J4
	4	DigP 2 ch 4	H4
	5	DigP 2 ch 5	G4
	6	DigP 2 ch 6	F4
	7	DigP 2 ch 7	E4
	8	DigP 2 ch 8	D4
	9	DigP 2 ch 9	L5
	10	DigP 2 ch 10	K5
	11	DigP 2 ch 11	J5
	12	DigP 2 ch 12	H5
	13	DigP 2 ch 13	G5
	14	DigP 2 ch 14	F5
	15	DigP 2 ch 15	E5
	16	DigP 2 ch 16	D5

Port	Channel	Signal	I/O Connector Pin
3	1	DigP 3 ch 1	L6
	2	DigP 3 ch 2	K6
	3	DigP 3 ch 3	J6
	4	DigP 3 ch 4	H6
	5	DigP 3 ch 5	G6
	6	DigP 3 ch 6	F6
	7	DigP 3 ch 7	E6
	8	DigP 3 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

Note

Multiple access to the same digital output channels by other DIO_TYPE 3 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1511: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(750841ae7100dc832cb0a4b3af4492f3_img.jpg\)](#))
- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(78e449f8a1164b81ecbd00cd97498e27_img.jpg\)](#))

Related topics

Basics

MicroAutoBox II I/O Features..... 83

Serial Peripheral Interface on the DIO Type 4 Unit

Introduction

The serial peripheral interface (SPI) of the DIO Type 4 module provides high-speed synchronous communication with devices connected to the MicroAutoBox II Base board, such as an A/D converter.

Hardware requirements

Supported MicroAutoBox II hardware:

DS1401 Base Board	MicroAutoBox II with I/O Boards			
	DS1507	DS1511	DS1513	DS1514
–	–	–	✓	–

Characteristics

The DIO Type 4 unit of the DS1513 I/O board provides up to two serial peripheral interfaces (SPI).

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

Note

- The SPI can be processed only in master mode. It is unable to respond to any externally initiated serial transfers.
- SPI signals are usually TTL signals. Because of MicroAutoBox II's output stages, the signal voltage depends on the VDRIVE voltage, in cars, this is usually VBAT (12 V). In this case, the SPI signals of MicroAutoBox II cannot be directly connected to the external device.

The SPI supports the following features:

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

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

- Transfer length by specifying the number of words (1 ... 64) and bits per word (1 ... 128). The transfer length must not exceed 2048 bits.

Data received by the SPI is stored temporarily in a FIFO buffer of the DS1513. Buffer overflows are indicated by a status information, and cause old data to be overwritten.

- Bit direction in a word
- Period of the clock signal defined by the specified baud rate in the range 306 Baud ... 300 kBaud.
- Polarity and phase of the clock signal (SPI mode)
- Timing behavior of the transmission by specifying the time before and after transfer, the minimum time between two chip select cycles and the time between words. For detailed information on the timing parameters, see below.
- Optional, generation of an *end of cycle* interrupt.

Timing behavior

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

- If you specify 0 for the time between data words, the transfer of each word of an SPI cycle follows the same timing parameters.
- If you specify a value in the range 25 ns ... 793.6 µs for the time between data words, the other timing parameters only affect the timing behavior of the beginning and end of an entire SPI cycle. The chip select signal is not switched to the inactive state to mark the end of a word in the SPI cycle.

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

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

For an illustration of the timing behavior, see below.

The timing parameters can be specified with a maximum value of 793.6 µs. The value range is internally separated into 12 intervals that are automatically assigned to the specified value. Each interval provides a different step size that is used to saturate a specified value to its next available value.

Interval Number	Lower Limit of Interval	Step Size
1	0 ns	12.5 ns
2	400 ns	25 ns
3	800 ns	50 ns
4	1.6 μ s	100 ns
5	3.2 μ s	200 ns
6	6.4 μ s	400 ns
7	12.8 μ s	800 ns
8	25.6 μ s	1.6 μ s
9	51.2 μ s	3.2 μ s
10	102.4 μ s	6.4 μ s
11	204.8 μ s	12.8 μ s
12	409.6 μ s	25.6 μ s

Note

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

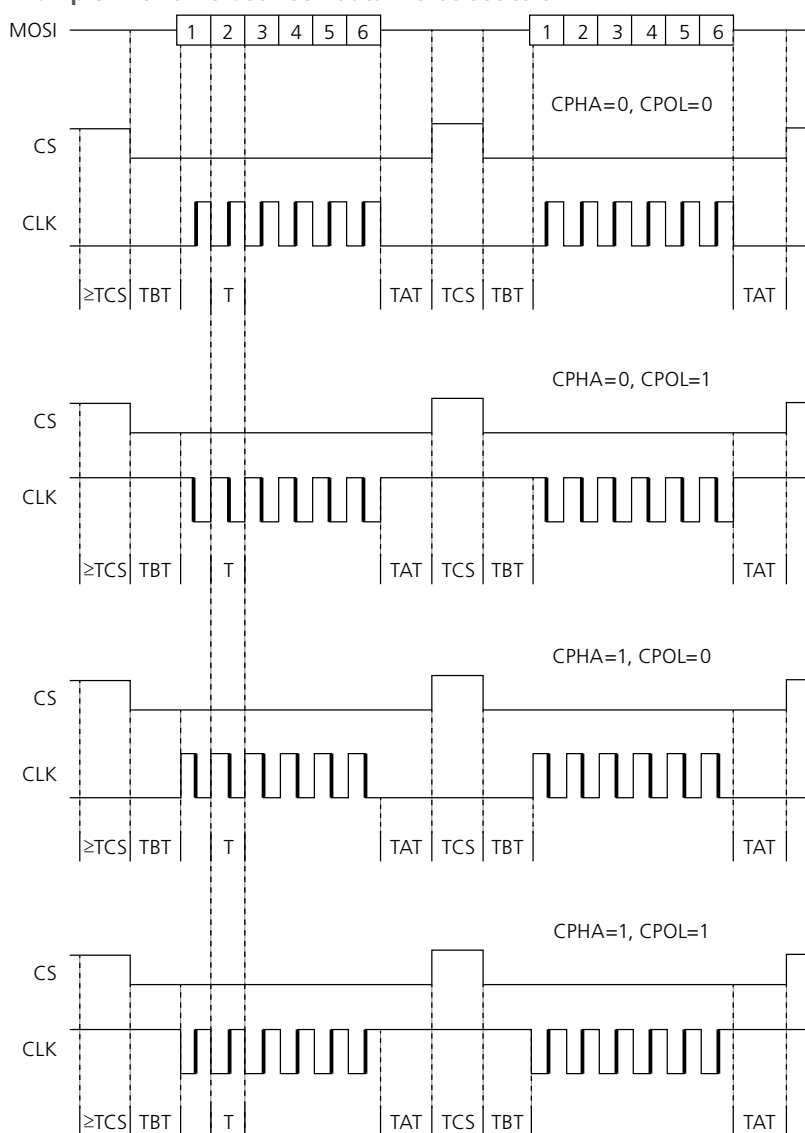
Examples of the timing behavior

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

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

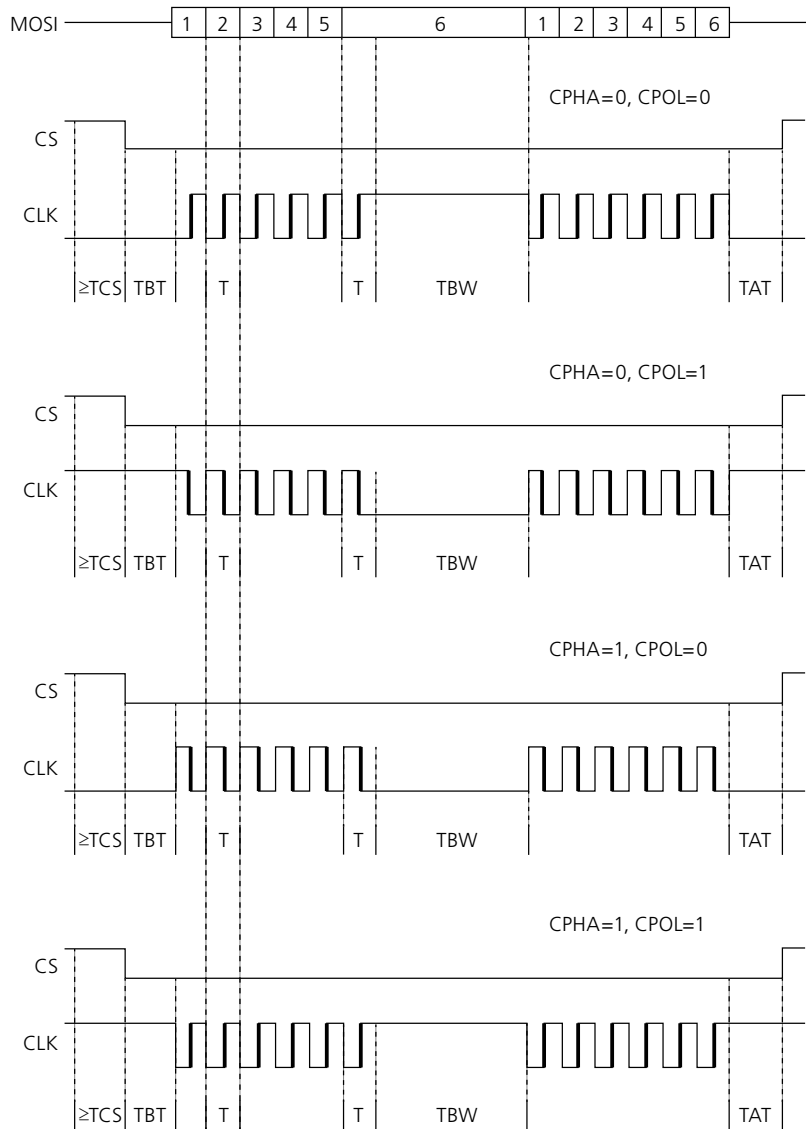
The following abbreviations are used:

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

Example with time between data words set to 0

The DIO Type 4 unit writes data to MOSI at the thin edges and reads data from MISO at the bold edges. While no data is transmitted, the MOSI channel is set to High-Z.

Example with time between data words set to >0



The DIO Type 4 unit writes data to MOSI at the thin edges and reads data from MISO at the bold edges. Between the words the last data bit is put out on the MOSI channel. While no data is transmitted, the MOSI channel is set to High-Z.

RTI/RTLib support

You have access to the serial peripheral interface on the DIO Type 4 module via RTI1401 and RTLib1401. For details, see:

- [Serial Peripheral Interface \(SPI\) \(MicroAutoBox II RTLib Reference !\[\]\(223f1a84e0bc2cacb9c165f716817dcc_img.jpg\)](#))
- [Serial Peripheral Interface \(MicroAutoBox II RTI Reference !\[\]\(c437123967ec19fa50ef7951237304ba_img.jpg\)](#))

I/O mapping

The following table shows the order of the signals. You have to specify the digital input channel (MISO) and the first digital output channel (CLK). The digital input channel can be configured as trigger source for interrupt generation. For the digital output channels, you can configure the electrical interface.

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

You can use one of the following digital input channels available on the DS1513 I/O connector to connect to the MISO signal.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	M2
	2	DigP 1 ch 2	L2
	3	DigP 1 ch 3	K2
	4	DigP 1 ch 4	J2
	5	DigP 1 ch 5	H2
	6	DigP 1 ch 6	M3
	7	DigP 1 ch 7	L3
	8	DigP 1 ch 8	K3
	9	DigP 1 ch 9	J3
	10	DigP 1 ch 10	H3
	11	DigP 1 ch 11	M4
	12	DigP 1 ch 12	L4
	13	DigP 1 ch 13	K4
	14	DigP 1 ch 14	J4
	15	DigP 1 ch 15	H4
	16	DigP 1 ch 16	M5

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	L5
	2	DigP 2 ch 2	K5
	3	DigP 2 ch 3	J5
	4	DigP 2 ch 4	H5
	5	DigP 2 ch 5	M6
	6	DigP 2 ch 6	L6
	7	DigP 2 ch 7	K6
	8	DigP 2 ch 8	J6

¹⁾ DigP <Port number> ch <Channel number>

Note

Multiple access to the same digital input channel by other DIO TYPE 4 blocks or functions is not allowed.

You can use up to six subsequent digital output channels available on the DS1513 I/O connector to connect to the CLK, MOSI, and CS1 ... CS4 signals.

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

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	G2
	2	DigP 1 ch 2	F2
	3	DigP 1 ch 3	E2
	4	DigP 1 ch 4	D2
	5	DigP 1 ch 5	C2
	6	DigP 1 ch 6	G3
	7	DigP 1 ch 7	F3
	8	DigP 1 ch 8	E3
	9	DigP 1 ch 9	D3
	10	DigP 1 ch 10	C3
	11	DigP 1 ch 11	G4
	12	DigP 1 ch 12	F4
	13	DigP 1 ch 13	E4
	14	DigP 1 ch 14	D4
	15	DigP 1 ch 15	C4
	16	DigP 1 ch 16	G5

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	F5
	2	DigP 2 ch 2	E5
	3	DigP 2 ch 3	D5
	4	DigP 2 ch 4	C5
	5	DigP 2 ch 5	G6
	6	DigP 2 ch 6	F6
	7	DigP 2 ch 7	E6
	8	DigP 2 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

Note

Multiple access to the same digital output channels by other DIO TYPE 4 blocks or functions is not allowed.

For a complete overview on the pinout, refer to:

- MicroAutoBox II 1401/1513: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(750841ae7100dc832cb0a4b3af4492f3_img.jpg\)](#))
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(78e449f8a1164b81ecbd00cd97498e27_img.jpg\)](#))

Related topics

Basics

MicroAutoBox II I/O Features..... 83

Limitations

Where to go from here

Information in this section

Limited Number of CAN Messages.....	369
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.	
Limitations with RTICANMM.....	371
There are a number of general limitations with RTICANMM.	
Limitations with CAN FD.....	375
There are limitations regarding the CAN FD support of RTICANMM.	
Limitations with J1939-Support.....	375
There are a number of limitations regarding the J1939 support of RTICANMM.	
Limitations of RTI LIN MultiMessage Blockset.....	376
Limitations apply when you use the RTI LIN MultiMessage Blockset.	

Limited Number of CAN Messages

Limitation

When you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions, the number of CAN messages in an application is limited.

This applies to the following message types:

- Transmit (TX) messages
- Receive (RX) messages

- Request (RQ) messages

An RQ message and the corresponding RX message are interpreted as a single (RQ) message. You cannot enable RX service support for the corresponding RX message.

- Remote (RM) messages

The sum of these messages is n_{sum} :

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

Maximum number of CAN messages

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

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

n_{max} in one application depends on:

- Whether you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions.
- Whether you use RX service support.

The maximum number of CAN messages n_{max} is listed in the table below:

Platform	n_{max} with RTLib	n_{max} with RTI CAN Blockset							
		RX Service Support Disabled				RX Service Support Enabled			
		1 ¹⁾	2 ¹⁾	3 ¹⁾	4 ¹⁾	1 ¹⁾	2 ¹⁾	3 ¹⁾	4 ¹⁾
DS2202 (2 CAN controllers)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
DS2210 (2 CAN controllers)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
DS2211 (2 CAN controllers)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
MicroAutoBox II ³⁾ (2 CAN controllers per CAN_Type1)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
MicroLabBox (2 CAN controllers)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
DS4302 (4 CAN controllers)	200	198	196	194	192	196 ²⁾	192 ²⁾	188 ²⁾	184 ²⁾

¹⁾ Number of CAN controllers used in the application

²⁾ 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.


³⁾ Depending on the variant, the MicroAutoBox II contains up to three CAN_Type1 modules, each with 2 CAN controllers. The values in the list apply to a single CAN_Type1 module.

Ways to implement more CAN messages

There are two ways to implement more CAN messages in an application.

Using RX service support If you use RTI CAN Blockset's RX service support, the number of receive (RX) messages n_{RX} in the equations above applies only to RTICAN Receive (RX) blocks for which RX service support is not enabled.

The number of RTICAN Receive (RX) blocks for which RX service support is enabled is unlimited. Refer to [Using RX Service Support](#) on page 252.

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 ¹⁾ (2 CAN controllers per CAN_Type1)	15
MicroLabBox (2 CAN controllers)	15
DS4302 (4 CAN controllers)	31


¹⁾ 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 CAN FD

Limitations

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

- To use CAN FD, you must provide a suitable DBC or AUTOSAR system description file containing descriptions in CAN FD format.
- For a MicroAutoBox II and DS4342, CAN FD messages cannot cause a transceiver wake-up.
- If message format manipulation and message length manipulation are enabled for a message, and if you switch between classic CAN and CAN FD formats during run time, the range of possible message lengths is not adjusted, neither upwards, nor downwards. The upper limit for the message length remains unchanged.

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.


Limitations of RTI LIN MultiMessage Blockset

Introduction

Limitations apply when you use the RTI LIN MultiMessage Blockset.

RTI LIN MultiMessage Blockset

The following limitations apply to the RTI LIN MultiMessage Blockset:

- Do not use the RTI LIN MultiMessage Blockset in enabled subsystems, triggered subsystems and configurable subsystems. As an alternative, you can disable nodes or frames of the RTI LIN MultiMessage Blockset by setting the corresponding option.
- Do not run the RTI LIN MultiMessage Blockset in a separate task.
- Do not copy blocks of the RTI LIN MultiMessage Blockset. To add further blocks of the RTI LIN MultiMessage Blockset to a model, always take them directly from the `rtilinmm.lib` library.
- If you switch the platform from a SCALEXIO system to a non-SCALEXIO system or vice versa, you have to recreate all RTILINMM blocks. To recreate all RTILINMM blocks at once, select **Create S-function for all LIN Blocks** from the Options menu of the GeneralSetup block (refer to [Options Menu \(RTILINMM GeneralSetup\)](#) ([RTI LIN MultiMessage Blockset Reference](#) )).
If you switch the platform from one non-SCALEXIO system to another (for example, from a DS1006 to a MicroAutoBox II) but the board and channel settings are not suitable for the new platform, you have to open the RTILINMM ControllerSetup blocks and recreate the S-functions for the blocks.

- RTILINMM ControllerSetup blocks that are added to the model but that are not assigned to RTILINMM MainSetup blocks (i.e., 'free floating' RTILINMM ControllerSetup blocks) do not initialize LIN channels of the real-time hardware. Therefore, these LIN channels cannot be accessed during run time via ControlDesk or Real-Time Testing (RTT) (e.g., for monitoring purposes).
- The RTI LIN 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 LIN MultiMessage Blockset, invoke **Create S-Function for all LIN Blocks** from the **Options** menu of the RTILINMM GeneralSetup block (refer to [RTILINMM GeneralSetup \(RTI LIN MultiMessage Blockset Reference\)](#))).

As an alternative, you can create new S-functions for all RTILINMM blocks manually (use the following order):

1. RTILINMM GeneralSetup
2. RTILINMM ControllerSetup
3. RTILINMM MainSetup

- Model path names with multi-byte character encodings are not supported.
- The RTI LIN 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 LIN 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:

- Frames
- Event-triggered frames
- Signals
- UpdateBit signals
- Nodes
- Schedules
- Schedule entries
- 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 LIN MultiMessage Blockset.

DBC file as the database

The following limitations apply if your database file is in the DBC file format:

- As the CAN protocol has no standard definitions for master nodes and schedules, such definitions are not supported by the DBC files.
- According to the LIN specification only byte layouts in the Intel format are supported. Byte layouts in Motorola format are not supported.

LDF file as the database

The RTI LIN MultiMessage Blockset does not support multiple physical encodings in the LDF file. If several physical encoding types are defined for a signal, the RTI LIN MultiMessage Blockset uses the first type it finds for the signal.

FIBEX file as the database

The following limitations apply if you import a FIBEX file as the database:

- The RTI LIN MultiMessage Blockset does not support multiple computation methods for signals. If several CompuMethods are defined for a signal in the FIBEX file, the RTI LIN MultiMessage Blockset uses the first linear computation method it finds for the signal.
- You cannot specify the checksum calculation type individually for each LIN communication frame. The RTI LIN MultiMessage Blockset always uses specific checksum calculation types as follows:
 - As of LIN protocol version 2.0, the 'ENHANCED' checksum calculation type is used.
 - For LIN protocol versions < 2.0, the 'CLASSIC' checksum calculation type is used.
 - The 'CLASSIC' checksum calculation type is always used for the master request and slave response frames.

MAT file as the database

In the RTI LIN 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 RTILINMM, 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 LIN MultiMessage Blockset, the signal names in the Simulink model must not exceed 32 characters.

AUTOSAR system description file as the database

The following limitations apply if you import an AUTOSAR system description file as the database:

Limitation	Limitation Applies to AUTOSAR Release												
	3.1.4	3.2.1	3.2.2	4.0.3	4.1.1	4.1.2	4.2.1	4.2.2	4.3.0	4.3.1	4.4.0	Classic R19-11 ¹⁾	Classic R20-11 ²⁾
Configuring selected slave nodes via RTI LIN MultiMessage Blockset is not possible due to the following limitations, which apply to AUTOSAR System Templates:													
<ul style="list-style-type: none">▪ In AUTOSAR, it is not possible to define collision resolver schedules for event-triggered frames. So if your database file is an AUTOSAR system description file, you have to manually assign a collision schedule to each event-triggered frame on the Collision Resolver Page (RTILINMM MainSetup).	✓	✓	✓	–	–	–	–	–	–	–	–	–	–
<ul style="list-style-type: none">▪ The following initial node configuration parameters for slave nodes are not provided by AUTOSAR system description files:<ul style="list-style-type: none">▪ Supplier ID▪ Function ID▪ Variant ID▪ Initial NAD	✓	✓	✓	–	–	–	–	–	–	–	–	–	–

Limitation	Limitation Applies to AUTOSAR Release												
	3.1.4	3.2.1	3.2.2	4.0.3	4.1.1	4.1.2	4.2.1	4.2.2	4.3.0	4.3.1	4.4.0	Classic R19-11 ¹⁾	Classic R20-11 ²⁾
<p>For the first three parameters, no identifier values are displayed on the Network Node Identification Page (RTILINMM MainSetup). However, you can specify an initial node address (NAD). By default, the configured NAD is used. If you do not specify an initial NAD, the selected node is interpreted as an unconfigured slave node.</p> <ul style="list-style-type: none">AUTOSAR does not support the assignment of lists of configurable frames to network nodes. Thus, you cannot select configurable frames to specify an initial frame ID.	✓	✓	✓	–	–	–	–	–	–	–	–	–	–
When importing an AUTOSAR system description file as the database, the RTI LIN MultiMessage Blockset supports the computation method from source value to physical representation (converted value) only.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<p>The RTI LIN MultiMessage Blockset does not support the following features that can be defined in an AUTOSAR system description file:</p> <ul style="list-style-type: none">End-to-end communication protection (E2E protection)Unit groups	– ³⁾	–	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
When working with an AUTOSAR ECU Extract as the database, the RTI LIN MultiMessage Blockset does not support the import of an AUTOSAR ECU Extract that describes a slave node.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

¹⁾ AUTOSAR Classic Platform Release R19-11

²⁾ AUTOSAR Classic Platform Release R20-11

³⁾ E2E protection is not supported by this AUTOSAR Release.


- The RTI LIN MultiMessage Blockset does not support the new features of AUTOSAR Release 4.4.0 and AUTOSAR Classic Platform Release R19-11 and R20-11.

Limitations with LIN specifications 2.x

- The RTI LIN MultiMessage Blockset does not support sporadic frames.
- The RTI LIN MultiMessage Blockset does not support dynamic frames. (Dynamic frames are provided by the LIN 2.0 specification only.)
- With a LIN node configuration compliant with LIN 2.0 and later, the RTI LIN MultiMessage Blockset supports the slave node configuration only. You can implement master functionality for a node configuration in the Simulink model, for example, by using raw data access for the master request frame. Refer to [Network Node Configuration Page \(RTILINMM MainSetup\) \(RTI LIN MultiMessage Blockset Reference\)](#).
- The RTI LIN MultiMessage Blockset does not support assign NAD frames.
- The RTI LIN MultiMessage Blockset does not support assign frame ID range frames.
- The RTI LIN MultiMessage Blockset does not support conditional change NAD frames.

Limitations with SAE J2602 support

The following limitations apply to J2602 support by the RTI LIN MultiMessage Blockset:

- The RTI LIN MultiMessage Blockset supports the SAE J2602 standard. It expects LIN protocol version J2602_1_1.0 and LIN language version J2602_3_1.0. If you use an LDF or AUTOSAR system description file with a different J2602 protocol version or language version (the version must also start with "J2602"), the RTI LIN MultiMessage Blockset continues working but this may lead to unpredictable behavior. A warning message is generated in the log file.
- The RTI LIN MultiMessage Blockset does not support sporadic frames.
- With a LIN node configuration compliant with J2602, the RTI LIN MultiMessage Blockset supports the slave node configuration according to LIN 2.0 only. You can implement master functionality for a node configuration in the Simulink model, for example, by using raw data access for the master request frame. Refer to [Network Node Configuration Page \(RTILINMM MainSetup\)](#) (RTI LIN MultiMessage Blockset Reference ) .
- When you import a J2602-compliant LDF or AUTOSAR system description file, the following optional master parameters are ignored by the RTI LIN MultiMessage Blockset:
 - max_header_length
 - response_tolerance
- When you import a J2602-compliant LDF or AUTOSAR system description file, the following optional node attributes are ignored by the RTI LIN MultiMessage Blockset:
 - response_tolerance
 - wakeup_time
 - poweron_time
- Compliance with J2602 response tolerances cannot be ensured.
- The maximum number of supported schedule table entries is 255.
- In contrast to the SAE J2602 specification, the unused bits of a frame are transmitted as dominant bits (0), not as recessive bits.
- Every clearly identified signal may be contained only once in a frame.
- The RTI LIN MultiMessage Blockset does not abort header transmission if an error occurs.
- The RTI LIN MultiMessage Blockset cannot access certain error states (ID parity error, framing error, bit error). There is no possibility to react to these errors within the model.
- There is no automatic parameterization of the J2602 status byte.
- The RTI LIN MultiMessage Blockset does not automatically support the Targeted Reset.
- The RTI LIN MultiMessage Blockset does not support the J2602 broadcast reset functionality.
- The RTI LIN MultiMessage Blockset does not support wake-up timings as specified in the J2602 standard. There is no automatic generation of wake up request sequences. If necessary, you can implement wake up request sequences in the Simulink model by using wake up requests which are supported by the RTI LIN MultiMessage Blockset.

- According to the J2602 specification, a sleep mode that lasts for at least four seconds is interpreted as a bus error. This error is indicated by the J2602 status byte. The RTI LIN MultiMessage Blockset does not interpret a sleep mode of this length as an error.
- According to the J2602 specification, a baud rate accuracy of $\pm 0.5\%$ is required. For the MicroAutoBox II, this condition is not met for the standard base baud rate of 10417 bit/s. The closest available baud rate of 10472 bit/s (which corresponds to a variance of $+0.53\%$) is used instead.

Visualization with the Bus Navigator

The current version of the RTI LIN 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 LIN MultiMessage Blockset.

Related topics

References

[Collision Resolver Page \(RTILINMM MainSetup\) \(RTI LIN MultiMessage Blockset Reference !\[\]\(0aff635c4179ba9e710b00f4b01d3b20_img.jpg\)\)](#)
[Network Node Identification Page \(RTILINMM MainSetup\) \(RTI LIN MultiMessage Blockset Reference !\[\]\(29658d981ebdf5edc259074cbf6110e0_img.jpg\)\)](#)

I/O Mappings

Where to go from here

Information in this section

I/O Mapping for MicroAutoBox II 1401/1507.....	383
I/O Mapping for MicroAutoBox II 1401/1511.....	384
I/O Mapping for MicroAutoBox II 1401/1513.....	388
I/O Mapping for MicroAutoBox II with DS1552 Multi-I/O Module.....	392
I/O Mapping for MicroAutoBox II with DS1554 Engine Control I/O Module.....	395

I/O Mapping for MicroAutoBox II 1401/1507

Introduction

MicroAutoBox II 1401/1507 provides access to three ECU interfaces via ECU interface connectors, which are 4-pin LEMO I/O connectors. Matching LVDS Link cables are supplied from dSPACE on request.

ECU Interface Connector	ECU Interface	Signal	I/O Connector Pin
ECU A (On DS1401 Base Board)	IF2	ECU / IF2 TX -	1
		ECU / IF2 TX +	2
		ECU / IF2 RX +	3
		ECU / IF2 RX -	4
ECU B (On DS1401 Base Board)	IF3	ECU / IF3 TX -	1
		ECU / IF3 TX +	2
		ECU / IF3 RX +	3
		ECU / IF3 RX -	4

ECU Interface Connector	ECU Interface	Signal	I/O Connector Pin
ECU C (On DS1507 I/O Board)	IF1	ECU / IF1 TX -	1
		ECU / IF1 TX +	2
		ECU / IF1 RX +	3
		ECU / IF1 RX -	4

Related topics

References

[Data Sheet MicroAutoBox II 1401/1507 \(MicroAutoBox II Hardware Reference !\[\]\(6605b201d6f14d9b3bcb8ab5f274d107_img.jpg\)](#))


I/O Mapping for MicroAutoBox II 1401/1511

Introduction

I/O mapping of the ADC unit


A/D Converter	Channel	Signal	I/O Connector Pin
1	1	Analog ch 1	Z3
2	2	Analog ch 2	Y3
3	3	Analog ch 3	X3
4	4	Analog ch 4	W3
5	5	Analog ch 5	Z4
6	6	Analog ch 6	Y4
7	7	Analog ch 7	X4
8	8	Analog ch 8	W4
9	9	Analog ch 9	Z5
10	10	Analog ch 10	Y5
11	11	Analog ch 11	X5
12	12	Analog ch 12	W5
13	13	Analog ch 13	Z6
14	14	Analog ch 14	Y6
15	15	Analog ch 15	X6
16	16	Analog ch 16	W6
Additional relevant signals ¹⁾			
External trigger input 1		Ana trigger 1	a3
External trigger input 2		Ana trigger 2	a4

A/D Converter	Channel	Signal	I/O Connector Pin
External trigger input 3		Ana trigger 3	a5
External trigger input 4		Ana trigger 4	a6

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

I/O mapping of the DAC unit

D/A Channel	Signal ¹⁾	I/O Connector Pin
1	Analog ch 1	Z2
2	Analog ch 2	Y2
3	Analog ch 3	X2
4	Analog ch 4	W2

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

I/O mapping of the inputs of the DIO Type 3 unit, used for bit I/O and timing I/O.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	V2
	2	DigP 1 ch 2	U2
	3	DigP 1 ch 3	T2
	4	DigP 1 ch 4	S2
	5	DigP 1 ch 5	R2
	6	DigP 1 ch 6	P2
	7	DigP 1 ch 7	N2
	8	DigP 1 ch 8	M2
	9	DigP 1 ch 9	V3
	10	DigP 1 ch 10	U3
	11	DigP 1 ch 11	T3
	12	DigP 1 ch 12	S3
	13	DigP 1 ch 13	R3
	14	DigP 1 ch 14	P3
	15	DigP 1 ch 15	N3
	16	DigP 1 ch 16	M3

Port	Channel	Signal	I/O Connector Pin
2	1	DigP 2 ch 1	V4
	2	DigP 2 ch 2	U4
	3	DigP 2 ch 3	T4
	4	DigP 2 ch 4	S4
	5	DigP 2 ch 5	R4
	6	DigP 2 ch 6	P4
	7	DigP 2 ch 7	N4
	8	DigP 2 ch 8	M4
	9	DigP 2 ch 9	V5
	10	DigP 2 ch 10	U5
	11	DigP 2 ch 11	T5
	12	DigP 2 ch 12	S5
	13	DigP 2 ch 13	R5
	14	DigP 2 ch 14	P5
	15	DigP 2 ch 15	N5
	16	DigP 2 ch 16	M5
3	1	DigP 3 ch 1	V6
	2	DigP 3 ch 2	U6
	3	DigP 3 ch 3	T6
	4	DigP 3 ch 4	S6
	5	DigP 3 ch 5	R6
	6	DigP 3 ch 6	P6
	7	DigP 3 ch 7	N6
	8	DigP 3 ch 8	M6

¹⁾ DigP <Port number> ch <Channel number>

I/O mapping of the outputs of the DIO Type 3 unit, used for bit I/O and timing I/O.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	L2
	2	DigP 1 ch 2	K2
	3	DigP 1 ch 3	J2
	4	DigP 1 ch 4	H2
	5	DigP 1 ch 5	G2
	6	DigP 1 ch 6	F2
	7	DigP 1 ch 7	E2
	8	DigP 1 ch 8	D2
	9	DigP 1 ch 9	L3
	10	DigP 1 ch 10	K3
	11	DigP1 ch 11	J3
	12	DigP 1 ch 12	H3
	13	DigP 1 ch 13	G3
	14	DigP 1 ch 14	F3
	15	DigP 1 ch 15	E3
	16	DigP 1 ch 16	D3
2	1	DigP 2 ch 1	L4
	2	DigP 2 ch 2	K4
	3	DigP 2 ch 3	J4
	4	DigP 2 ch 4	H4
	5	DigP 2 ch 5	G4
	6	DigP 2 ch 6	F4
	7	DigP 2 ch 7	E4
	8	DigP 2 ch 8	D4
	9	DigP 2 ch 9	L5
	10	DigP 2 ch 10	K5
	11	DigP 2 ch 11	J5
	12	DigP 2 ch 12	H5
	13	DigP 2 ch 13	G5
	14	DigP 2 ch 14	F5
	15	DigP 2 ch 15	E5
	16	DigP 2 ch 16	D5

Port	Channel	Signal	I/O Connector Pin
3	1	DigP 3 ch 1	L6
	2	DigP 3 ch 2	K6
	3	DigP 3 ch 3	J6
	4	DigP 3 ch 4	H6
	5	DigP 3 ch 5	G6
	6	DigP 3 ch 6	F6
	7	DigP 3 ch 7	E6
	8	DigP 3 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

For the interface descriptions, refer to [Interfaces \(MicroAutoBox II Hardware Reference !\[\]\(dfbd6b3763a6d1d9afaa974f64e2e4b5_img.jpg\)](#)).

For a complete pinout of the power input connector and the DS1511 ZIF I/O connector, refer to [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)](#)).

MicroAutoBox II 1401/1511 provides no external interrupts.

Related topics

References

[Data Sheet MicroAutoBox II 1401/1511 \(MicroAutoBox II Hardware Reference !\[\]\(05be7c7a8995decd503647c99211f7c2_img.jpg\)](#))


I/O Mapping for MicroAutoBox II 1401/1513

Introduction

I/O mapping of the ADC Type 4 unit


A/D Converter	Channel	Signal	I/O Connector Pin
1	1	Analog ch 1	Z3
2	2	Analog ch 2	Y3
3	3	Analog ch 3	X3
4	4	Analog ch 4	W3
5	5	Analog ch 5	Z4
6	6	Analog ch 6	Y4
7	7	Analog ch 7	X4
8	8	Analog ch 8	W4
9	9	Analog ch 9	Z5
10	10	Analog ch 10	Y5
11	11	Analog ch 11	X5
12	12	Analog ch 12	W5

A/D Converter	Channel	Signal	I/O Connector Pin
13	13	Analog ch 13	Z6
14	14	Analog ch 14	Y6
15	15	Analog ch 15	X6
16	16	Analog ch 16	W6
Additional relevant signals ¹⁾			
External trigger input 1		Ana trigger 1	a3
External trigger input 2		Ana trigger 2	a4
External trigger input 3		Ana trigger 3	a5
External trigger input 4		Ana trigger 4	a6

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

I/O mapping of the AIO Type 1 unit (ADC)

A/D Converter	Channel	Signal ¹⁾	I/O Connector Pin
1	1	AnalogIn ch 1	V3
2	2	AnalogIn ch 2	U3
3	3	AnalogIn ch 3	T3
4	4	AnalogIn ch 4	S3
5	5	AnalogIn ch 5	V4
6	6	AnalogIn ch 6	U4
7	7	AnalogIn ch 7	T4
8	8	AnalogIn ch 8	S4
9	9	AnalogIn ch 9	V5
10	10	AnalogIn ch 10	U5
11	11	AnalogIn ch 11	T5
12	12	AnalogIn ch 12	S5
13	13	AnalogIn ch 13	V6
14	14	AnalogIn ch 14	U6
15	15	AnalogIn ch 15	T6
16	16	AnalogIn ch 16	S6

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

I/O mapping of the AIO Type 1 unit (DAC)

D/A Channel	Signal ¹⁾	I/O Connector Pin
1	AnalogOut ch 1	Z2
2	AnalogOut ch 2	Y2
3	AnalogOut ch 3	X2

D/A Channel	Signal ¹⁾	I/O Connector Pin
4	AnalogOut ch 4	W2
5	AnalogOut ch 5	V2
6	AnalogOut ch 6	U2
7	AnalogOut ch 7	T2
8	AnalogOut ch 8	S2

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#) for connecting the analog channels to GND.

I/O mapping of the inputs of the DIO Type 4 unit, used for bit I/O and timing I/O.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	M2
	2	DigP 1 ch 2	L2
	3	DigP 1 ch 3	K2
	4	DigP 1 ch 4	J2
	5	DigP 1 ch 5	H2
	6	DigP 1 ch 6	M3
	7	DigP 1 ch 7	L3
	8	DigP 1 ch 8	K3
	9	DigP 1 ch 9	J3
	10	DigP 1 ch 10	H3
	11	DigP 1 ch 11	M4
	12	DigP 1 ch 12	L4
	13	DigP 1 ch 13	K4
	14	DigP 1 ch 14	J4
	15	DigP 1 ch 15	H4
	16	DigP 1 ch 16	M5
2	1	DigP 2 ch 1	L5
	2	DigP 2 ch 2	K5
	3	DigP 2 ch 3	J5
	4	DigP 2 ch 4	H5
	5	DigP 2 ch 5	M6
	6	DigP 2 ch 6	L6
	7	DigP 2 ch 7	K6
	8	DigP 2 ch 8	J6

¹⁾ DigP <Port number> ch <Channel number>

I/O mapping of the outputs of the DIO Type 4 unit, used for bit I/O and timing I/O.

Port	Channel	Signal	I/O Connector Pin
1	1	DigP 1 ch 1 ¹⁾	G2
	2	DigP 1 ch 2	F2
	3	DigP 1 ch 3	E2
	4	DigP 1 ch 4	D2
	5	DigP 1 ch 5	C2
	6	DigP 1 ch 6	G3
	7	DigP 1 ch 7	F3
	8	DigP 1 ch 8	E3
	9	DigP 1 ch 9	D3
	10	DigP 1 ch 10	C3
	11	DigP 1 ch 11	G4
	12	DigP 1 ch 12	F4
	13	DigP 1 ch 13	E4
	14	DigP 1 ch 14	D4
	15	DigP 1 ch 15	C4
	16	DigP 1 ch 16	G5
2	1	DigP 2 ch 1	F5
	2	DigP 2 ch 2	E5
	3	DigP 2 ch 3	D5
	4	DigP 2 ch 4	C5
	5	DigP 2 ch 5	G6
	6	DigP 2 ch 6	F6
	7	DigP 2 ch 7	E6
	8	DigP 2 ch 8	D6

¹⁾ DigP <Port number> ch <Channel number>

For the interface descriptions, refer to [Interfaces \(MicroAutoBox II Hardware Reference !\[\]\(9dfdaff1d86ba3c1f8353b4d1b61b8c5_img.jpg\)](#)).

For a complete pinout of the power input connector and the DS1513 ZIF I/O connector, refer to [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(83f22ed94ec5517769dd76d702c6bfd8_img.jpg\)](#)).

MicroAutoBox II 1401/1513 provides no external interrupts.

Related topics

References

[Data Sheet MicroAutoBox II 1401/1511 \(MicroAutoBox II Hardware Reference !\[\]\(3cb60d42b10e53f9522bb0b392c1c4cd_img.jpg\)](#))


I/O Mapping for MicroAutoBox II with DS1552 Multi-I/O Module

Introduction

MicroAutoBox II variants with a DS1514 I/O board can be enlarged with a DS1552 Multi-I/O Module. Its signals are available at the MicroAutoBox II I/O connector (DS1514 ZIF connector).

I/O mapping of the ADC 1552 Type 1 unit

A/D Converter	Channel	Signal	I/O Connector Pin
1	1	AnalogIn+ ch 1	X3
		AnalogIn- ch 1 ¹⁾	X4
2	2	AnalogIn+ ch 2	W3
		AnalogIn- ch 2 ¹⁾	W4
3	3	AnalogIn+ ch 3	V3
		AnalogIn- ch 3 ¹⁾	V4
4	4	AnalogIn+ ch 4	U3
		AnalogIn- ch 4 ¹⁾	U4
5	5	AnalogIn+ ch 5	H3
		AnalogIn- ch 5 ¹⁾	H4
6	6	AnalogIn+ ch 6	G3
		AnalogIn- ch 6 ¹⁾	G4
7	7	AnalogIn+ ch 7	F3
		AnalogIn- ch 7 ¹⁾	F4
8	8	AnalogIn+ ch 8	E3
		AnalogIn- ch 8 ¹⁾	E4
Additional relevant signals			
External trigger input 1		DigIn ch 1	V5
External trigger input 2		DigIn ch 2	U5
External trigger input 3		DigIn ch 3	U6
External trigger input 4		DigIn ch 4	T2

¹⁾ Negative input line of the ADC channel is connected to GND. To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#)  for connecting the analog channels to GND.

Note

DigIn ch 1 ... DigIn ch 4 of the DIO 1552 Type 1 unit are shared with the external trigger inputs of the ADC 1552 Type 1 unit.

I/O mapping of the ADC 1552 Type 2 unit

A/D Converter	Channel	Signal ¹⁾	I/O Connector Pin
1	1	AnalogIn ch 1	b2
2	2	AnalogIn ch 2	a2
3	3	AnalogIn ch 3	Z2
4	4	AnalogIn ch 4	Y2
5	5	AnalogIn ch 5	X2
6	6	AnalogIn ch 6	W2
7	7	AnalogIn ch 7	V2
8	8	AnalogIn ch 8	U2
9	9	AnalogIn ch 9	M2
10	10	AnalogIn ch 10	L2
11	11	AnalogIn ch 11	K2
12	12	AnalogIn ch 12	J2
13	13	AnalogIn ch 13	H2
14	14	AnalogIn ch 14	G2
15	15	AnalogIn ch 15	F2
16	16	AnalogIn ch 16	E2

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#) for connecting the analog channels to GND.

I/O mapping of the DAC 1552 Type 1 unit

D/A Channel	Signal ¹⁾	I/O Connector Pin
1	AnalogOut ch 1	c2
2	AnalogOut ch 2	c3
3	AnalogOut ch 3	c4
4	AnalogOut ch 4	c5

¹⁾ To get optimum analog performance, follow the instructions in [MicroAutoBox II Hardware Installation and Configuration Guide](#) for connecting the analog channels to GND.

I/O mapping of the inputs of the DIO 1552 Type 1 unit, used for bit I/O and timing I/O.

Channel	Signal	I/O Connector Pin
1	DigIn ch 1	V5
2	DigIn ch 2	U5
3	DigIn ch 3	U6
4	DigIn ch 4	T2
5	DigIn ch 5	T3
6	DigIn ch 6	T4

Channel	Signal	I/O Connector Pin
7	DigIn ch 7	T5
8	DigIn ch 8	T6
9	DigIn ch 9	S2
10	DigIn ch 10	S3
11	DigIn ch 11	S5
12	DigIn ch 12	R2
13	DigIn ch 13	R5
14	DigIn ch 14	R6
15	DigIn ch 15	P5
16	DigIn ch 16	P6

Note

DigIn ch 1 ... DigIn ch 4 of the DIO 1552 Type 1 unit are shared with the external trigger inputs of the ADC 1552 Type 1 unit.

I/O mapping of the outputs of the DIO 1552 Type 1 unit, used for bit I/O and timing I/O.

Channel	Signal	I/O Connector Pin
1	DigOut ch 1	F5
2	DigOut ch 2	E5
3	DigOut ch 3	E6
4	DigOut ch 4	D2
5	DigOut ch 5	D3
6	DigOut ch 6	D4
7	DigOut ch 7	D5
8	DigOut ch 8	D6
9	DigOut ch 9	C2
10	DigOut ch 10	C3
11	DigOut ch 11	C5
12	DigOut ch 12	B2
13	DigOut ch 13	B5
14	DigOut ch 14	B6
15	DigOut ch 15	A5
16	DigOut ch 16	A6

For the interface descriptions, refer to [Data Sheet DS1552 Multi-I/O Module \(MicroAutoBox II Hardware Reference !\[\]\(a870788d6ed9b8fd294b7654a8c8526b_img.jpg\)](#)).

For a complete pinout of the power input connector and the MicroAutoBox II I/O connectors, refer to:

- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(6302aad5aed157b291fddf37b4870784_img.jpg\)\)](#)
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(a9ca2c237943a6d0a9f22252f295b6f3_img.jpg\)\)](#)

The MicroAutoBox II variants with DS1514 I/O board provide no external interrupts.

Related topics

References

[Data Sheet DS1552 Multi-I/O Module \(MicroAutoBox II Hardware Reference !\[\]\(a03a7eb2f4046e1d3c76772003e549ea_img.jpg\)\)](#)
[Data Sheet MicroAutoBox II 1401/1511/1514 \(MicroAutoBox II Hardware Reference !\[\]\(844169987a590ed8c7e31d5d18950e8d_img.jpg\)\)](#)
[Data Sheet MicroAutoBox II 1401/1513/1514 \(MicroAutoBox II Hardware Reference !\[\]\(2af34e678d9364b2f32b7174f4964d2c_img.jpg\)\)](#)

I/O Mapping for MicroAutoBox II with DS1554 Engine Control I/O Module

Introduction

MicroAutoBox II variants with a DS1514 I/O board can be enlarged with a DS1554 Engine Control I/O Module. Its signals are available at the MicroAutoBox II I/O connector (DS1514 ZIF connector).

Because the features of the DS1554 Engine Control I/O Module are only supported by the RTI FPGA Programming Blockset, refer to [RTI Block Settings for the FPGA1401Tp1 with Engine Control I/O Module Framework \(RTI FPGA Programming Blockset - FPGA Interface Reference !\[\]\(5361750c22c4e047a52f4eac1ec2d4cc_img.jpg\)\)](#) for information on the I/O mapping.

For the interface descriptions, refer to [Data Sheet DS1554 Engine Control I/O Module \(MicroAutoBox II Hardware Reference !\[\]\(870f5d5e9c0d57485634be3ecf52f3ca_img.jpg\)\)](#).

For a complete pinout of the power input connector and the MicroAutoBox II I/O connectors, refer to:

- MicroAutoBox II 1401/1511/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(69baca079ef3ab6f03d58fd7e9f950f1_img.jpg\)\)](#)
- MicroAutoBox II 1401/1513/1514: [Connector Pinouts \(MicroAutoBox II Hardware Reference !\[\]\(2da321c3dc978a55192cb9c452297973_img.jpg\)\)](#)

The MicroAutoBox II variants with DS1514 I/O board provide no external interrupts.

Related topics

References

[Data Sheet DS1554 Engine Control I/O Module \(MicroAutoBox II Hardware Reference !\[\]\(dfbd6b3763a6d1d9afaa974f64e2e4b5_img.jpg\)\)](#)

[Data Sheet MicroAutoBox II 1401/1511/1514 \(MicroAutoBox II Hardware Reference !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)\)](#)

[Data Sheet MicroAutoBox II 1401/1513/1514 \(MicroAutoBox II Hardware Reference !\[\]\(23d9fc146e83b5c3013cfa32c784f8d5_img.jpg\)\)](#)

Appendix

Where to go from here

Information in this section

Troubleshooting.....	397
Gives you information on known problems and their solutions.	
Glossary.....	398
The glossary briefly explains the most important expressions and naming conventions used in the MicroAutoBox II documentation.	

Troubleshooting

Introduction

Gives you information on known problems and their solutions.

To stay up-to-date with information on possible problems, you should periodically check the known problem reports at <http://www.dspace.com/go/ProblemReports>.

Loading the real-time application not possible or slow

If you want to use the Signal Generator or the Bus Navigator in ControlDesk, you have to set the **Enable real-time testing** option in your Simulink model. For new models this option is set by default. For older models it might be automatically set when you migrate.

If the **Enable real-time testing** option is set, the size of the built real-time application increases. The time for loading the application from the flash memory therefore will be increased. In worst case, the rebuilt real-time application cannot be loaded because the board's memory is not sufficient.

- If you do not need the real-time testing support, clear the **Enable real-time testing** option in your model.

Glossary

A	ADC	Analog/digital converter
C	CAN	Controller area network
D	DAC	Digital/analog converter
	DIO	Digital input/output
	DPMEM	Dual-port memory
E	ECU	Electronic control unit
F	F2D	Frequency measurement Two successive edges of the same level (high or low) are detected and used to measure the time between them.
	FPGA	Field Programmable Gate Array. An FPGA module is used in digital technology to allow modifications in a circuit's functionality without replacing hardware.
L	LIN	Local interconnect network
M	MC	Microcontroller
P	PPC	Power PC processor
	PW2D	Pulse width measurement Two successive edges of different levels (high and low) are detected and used to measure the time between them.
	PWM2D	PWM measurement (frequency and duty cycle) All the three edges within a period (high-low-high or low-high-low) are detected and used to measure the time between them.
	PWM	Pulse width modulation

S

SENT Single edge nibble transmission

SENT is a serial protocol used between sensors and ECUs. It is defined in the SAE J2716 standard defined by the Society of Automotive Engineers (SAE).

SPI Serial peripheral interface

A

- A/D conversion 86
 - ADC 1552 Type 1 101
 - ADC 1552 Type 2 108
 - ADC Unit Type 4 87
 - AIO Type 1 99
- acceleration sensor 78
 - measurement range 79
 - measuring 78
 - output data rate 80
 - read mode 79
 - register access 80
 - RTI/RTLib support 80
 - sample rate 80
- ADC 1552 Type 1
 - characteristics 101
 - conversion trigger 103
 - conversion trigger overflow interrupt 106
 - data lost interrupt 106
 - data ready interrupt 106
 - external trigger 103
 - I/O mapping 107
 - interrupts 105
 - swinging buffer 102
 - timer 103
 - trigger signals 103
 - trigger sources 103
- ADC 1552 Type 2
 - characteristics 108, 109
 - I/O mapping 109
- ADC Type 4
 - burst start interrupt 95
 - burst trigger 89
 - characteristics 87
 - conversion trigger 89
 - conversion trigger overflow interrupt 97
 - data lost interrupt 97
 - data ready interrupt 96
 - external trigger 90
 - I/O mapping 98
 - interrupts 95
 - swinging buffer 88
 - time-stamping mode 94
 - trigger signals 89
 - trigger sources 90
- ADC unit
 - overview 86
- additional I/O 133
- AIO Type 1 ADC
 - characteristics 99
 - I/O mapping 100
- AIO Type 1 DAC 127
 - characteristics 127
 - I/O mapping 127
- application
 - flash memory 33
 - global memory 32
- application start
 - MicroAutoBox II 30
- auto negotiation 131

- AUTOSAR system description file 303
- AUTOSAR System Template 303

B

- base board
 - features 35
- basics
 - SENT 334
- basics of LIN bus 290
- basics on RTI LIN MultiMessage Blockset 311
- baud rate detection 295
- bit I/O
 - DIO 1552 Type 1 122
- Bit I/O
 - DIO Type 3 112
 - DIO Type 4 117
- Bit I/O (DIO 1552 Type 1)
 - I/O mapping 124
- Bit I/O (DIO Type 3)
 - I/O mapping 114
- Bit I/O (DIO Type 4)
 - I/O mapping 119
- Bit I/O unit
 - overview 111
- boot firmware 30
- broadcast 131
- burst trigger
 - ADC Type 4 89
- bypass-based prototyping 133

C

- CAN
 - channel 244
 - fault-tolerant transceiver 247
 - interrupts 251
 - physical layer 245
 - service 254
 - setup 243
 - status information 254
 - subsystem 13
- CAN controller
 - frequency 249
- CAN FD 263
 - limitations 375
- CAN MC clock rate 249
- CAN support 241
- challenge-response monitoring
 - basics 52
 - characteristics 53
 - failure actions 55
 - RTI support 57
 - RTLib support 57
 - self-test 57
 - stopping the real-time application 57
 - timing constraints 54
- characteristics
 - SPI (DIO Type 3) 352
 - SPI (DIO Type 4) 361
- checksum algorithm
 - defining 316

- implementing 315
- checksum error 298
- clock drift 335
- clock rate
 - CAN MC 249
- Common Program Data folder 10
- conversion trigger
 - ADC 1552 Type 1 103
 - ADC Type 4 89

D

- D/A conversion 126
 - AIO Type 1 127
 - DAC 1552 Type 1 129
 - DAC Type 3 128
- DAC 1552 Type 1 129
 - characteristics 129
 - I/O mapping 130
- DAC Type 3 128
 - characteristics 128
 - I/O mapping 128
- DAC unit
 - overview 126
- data file support 250
- data rates 131
- database files 302
- datagram size 131
- DBC file 302
- decrementer 48
- defining
 - LIN frame 304
 - LIN master 303
 - LIN slave 303
- defining CAN messages 250
- defining LIN communication 313
- defining RX and TX frames 314
- DHCP 131
- DIO 1552 Type 1
 - bit I/O 122
 - Bit I/O characteristics 122
 - F2D 216
 - I/O mapping for Bit I/O 124
- DIO 1552 Type 1 characteristics
 - F2D 216
- DIO Type 3 112
 - Bit I/O characteristics 112
 - F2D 209
 - FREQ 159
 - FREQ characteristics 160
 - I/O mapping for Bit I/O 114
 - incremental encoder 231
 - incremental encoder characteristics 231
 - PW2D 220
 - PW2D characteristics 220
 - PWM 147
 - PWM characteristics 147
 - SPI 351
- DIO Type 3 characteristics
 - F2D 209
- DIO Type 4 117
 - Bit I/O characteristics 118

- F2D 213
- FREQ 162
- FREQ characteristics 163
- I/O mapping for Bit I/O 119
- incremental encoder 236
- incremental encoder characteristics 236
- PW2D 224
- PW2D characteristics 224
- PWM 151
- PWM characteristics 152
- PWM2D 202
- PWM2D characteristics 202
- SPI 360
- DIO Type 4 characteristics
 - F2D 213
- DIO_TYPE3_SENT_RX_Blx
 - implementing 342
- DIO_TYPE4_SENT_RX_Blx
 - implementing 342
- Documents folder 10
- DS1401
 - features 35
- DS1401 Base board 13
- DS1552 Multi-I/O Module
 - feature overview 13
- DS1554 Engine Control I/O Module
 - feature overview 14

E

- ECU
 - data types 139
 - DPMEM addresses 137
 - interface unit
 - characteristics 133
 - I/O mapping 135
 - interrupt addresses 138
 - interrupts 39
 - subinterrupts 39
- ECU Software Porting Kit 142
 - example 143
 - working with the 142
- Embedded PC
 - MicroAutoBox 14
- energy-saving modes 298
- Ethernet I/O Interface 131
- example of LIN bus 292
- external trigger
 - ADC 1552 Type 1 103
 - ADC Type 4 90

F

- F2D
 - DIO 1552 Type 1 216
 - DIO Type 3 209
 - DIO Type 4 213
- F2D characteristics
 - DIO 1552 Type 1 216
 - DIO Type 3 209
 - DIO Type 4 213
- FIBEX file 303
- firmware 30
- flash memory
 - application 33
- FlexRay
 - IP Module 13
 - MicroAutoBox 319
 - support 319
- flight recorder 65, 66
 - time stamps 67
 - using 67
- FPGA support 326
 - access functions 327
 - accessing FPGA Type 1 327
 - hardware components 326
 - RTI support 328
 - RTLib support 328
 - software components 326
- frames
 - manipulating 305
 - receiving 304
 - transmitting 304
- FREQ
 - DIO 1552 Type 1 166
 - DIO Type 3 159
 - DIO Type 3 characteristics 160
 - DIO Type 4 162
 - DIO Type 4 characteristics 163
- frequency
 - CAN controller 249

G

- global memory
 - application 32
- glossary
 - MicroAutoBox II terms 398

H

- hardware interrupts 37
- high pulse 335
- high-side switches 112, 118, 122

I

- I/O mapping
 - ADC 1552 Type 1 107
 - ADC 1552 Type 2 109
 - ADC Type 4 98
 - AIO Type 1 ADC 100
 - AIO Type 1 DAC 127
 - DAC 1552 Type 1 130
 - DAC Type 3 128
 - DS1401 IP module 324
 - ECU interface 135
 - F2D (DIO 1552 Type 1) 218
 - F2D (DIOTP3) 211
 - F2D (DIOTP4) 215
 - FREQ (DIO 1552 Type 1) 167
 - FREQ (DIOTP3) 161
 - FREQ (DIOTP4) 164
 - MicroAutoBox FlexRay 320
 - PW2D (DIOTP3) 222
 - PW2D (DIOTP4) 226
 - PWM (DIO 1552 Type 1) 158
 - PWM (DIOTP3) 150
 - PWM (DIOTP4) 154
 - PWM2D (DIO 1552 Type 1) 207
 - PWM2D (DIOTP3) 199
 - PWM2D (DIOTP4) 204
 - serial interface 330
 - SPI (DIO Type 3) 357
 - SPI (DIO Type 4) 366
- I/O modules
 - MicroAutoBox II 13
- identifier-parity error 297
- implementing
 - SENT 338
- implementing checksum algorithm 315
- inconsistent-synch-byte error 297
- incremental encoder 229
 - channel usage (DIO Type 3) 233
 - channel usage (DIO Type 4) 238
 - DIO Type 3 231
 - DIO Type 4 236
 - I/O mapping (DIO Type 3) 233
 - I/O mapping (DIO Type 4) 238
 - noise filter sample rate (DIO Type 3) 232
 - noise filter sample rate (DIO Type 4) 237
- incremental encoder characteristics
 - DIO Type 3 231
 - DIO Type 4 236
- incremental encoder interface
 - overview 228
- in-frame response time 295
- interrupt
 - burst start 95
 - conversion trigger overflow 97, 106
 - data lost 97, 106
 - data ready 96, 106
- interrupt handling 37
- interrupts
 - ADC 1552 Type 1 105
 - ADC Type 4 95
- IP fragmentation 131
- IP module
 - DS1401 324
 - support 323
- IP Module Support 323
- IP modules
 - MicroAutoBox 323
- ISO11898 246
- ISO11898-6 247

J

- J1939
 - broadcast/peer-to-peer messages 268
 - limitations 375
 - working with J1939-compliant DBC files 268
- J2716 (SAE) 334

L

- LDF file 302

- limitations
 - CAN FD 375
 - J1939 375
 - RTI CAN MultiMessage Blockset 371
- limitations of RTI LIN MultiMessage Blockset 376
- LIN
 - I/O mapping 288
 - schedules 308
 - sleep mode 306
 - support 287
- LIN bus
 - connection 301
 - controlling 305
 - handling 300
 - waking up 306
- LIN bus basics 290
- LIN bus parameters
 - changing 307
 - specifying 302
- LIN buses
 - number 301
- LIN communication
 - defining 313
- LIN frame
 - defining 304
- LIN frames
 - receiving 313
 - transmitting 313
- LIN master 291
- LIN nodes
 - switching 306
- LIN slave
 - defining 303
- LIN specifications
 - supported 290
- Local Program Data folder 10
- low pulse 335
- low-side switches 112, 118, 122

M

- manipulating
 - frames 305
- manipulating TX signals 317
- MAT format 74
- measurement mode
 - PW2D (DIOTP3) 221
 - PW2D (DIOTP4) 225
- memory integrity and extras
 - basics 57
 - characteristics 58
 - failure actions 59
 - prerequisites 58
 - RTI support 60
 - RTLib support 60
 - run-time behavior 60
- message frame 291
- messages
 - defining CAN messages 250
 - delay time 250
 - multiple 251

- MicroAutoBox Embedded PC
 - feature overview 14
- MicroAutoBox II
 - application start 30
 - glossary 398
- MicroAutoBox II 1401/1507 19
- MicroAutoBox II 1401/1511 20
- MicroAutoBox II 1401/1511/1514 21
- MicroAutoBox II 1401/1511/1514 with DS1552 Multi-I/O Module 23
- MicroAutoBox II 1401/1511/1514 with DS1554 Engine Control I/O Module 24
- MicroAutoBox II 1401/1513 22
- MicroAutoBox II 1401/1513/1514 23
- MicroAutoBox II 1401/1513/1514 with DS1552 Multi-I/O Module 23
- MicroAutoBox II 1401/1513/1514 with DS1554 Engine Control I/O Module 24
- multiple data files 250
- multiple message support 251

N

- nibble pulse 336

P

- pause pulse 336
- piggyback boards
 - MicroAutoBox II 13
- piggyback support 248
- power hold on 76
- PPC
 - MicroAutoBox II Base board 28
- pressure sensor 81
 - RTI/RTLib support 81
- PW2D
 - DIO Type 3 220
 - DIO Type 3 characteristics 220
 - DIO Type 4 224
 - DIO Type 4 characteristics 224
- PWM
 - DIO 1552 Type 1 156
 - DIO Type 3 147
 - DIO Type 4 151
 - DIO Type 4 characteristics 152
- PWM characteristics
 - DIO Type 3 147
- PWM2D
 - DIO 1552 Type 1 206
 - DIO Type 3 197
 - DIO Type 4 202
 - DIO Type 4 characteristics 202

R

- raw data
 - working with 315
- reading
 - status information 307
- receiving
 - SENT message 345
 - SENT message via RTI 342

- receiving frames 304
- remaining bus simulation 294
- reset 30
- routing 131
- RS485 247
- RTI CAN MultiMessage Blockset
 - limitations 371
 - supported platforms 258
- RTI LIN MultiMessage Blockset
 - basics 311
 - limitations 376
- RTICANMM
 - CAN FD 375
 - J1939 375
- RTILINMM
 - checksum header file 316
- RX frames
 - defining 314
- RX service support 252

S

- SAE J2716 334
- SAE standards
 - supported 290
- safety functions 49
 - basics 49
 - challenge-response monitoring 52
 - functional safety extras 57
 - memory integrity and extras 57
- schedule
 - LIN 308
 - priority 309
 - setting up 308
- SENT 333
 - basics 334
 - implementing on MicroAutoBox II 338
 - signal 334
 - terms and definitions 334
- SENT message
 - receiving 345
 - receiving via RTI 342
- SENT pulse 335
- serial interface 329
 - baud rates 331
 - I/O mapping 330
 - oscillator frequency 331
- serial peripheral interface
 - DIO Type 3 351
 - DIO Type 4 360
- signal
 - SENT message 334
- signal generation
 - overview 144
- signal measurement
 - overview 195
- slave-not-responding error 298
- sleep mode
 - LIN bus 306
- SPI
 - DIO Type 3 351
 - DIO Type 4 360

- SPI (DIO Type 3)
 - characteristics 352
 - examples 354
 - I/O mapping 357
 - timing behavior 352
- SPI (DIO Type 4)
 - characteristics 361
 - examples 363
 - I/O mapping 366
 - timing behavior 362
- status information
 - reading 307
- subinterrupt handling 140
 - word-based 40
- swinging buffer
 - ADC 1552 Type 1 102
 - ADC Type 4 88
- switching
 - LIN nodes 306
- sync high pulse 335
- sync pulse 336

T

- time base
 - USB Flight Recorder 73
- time base counter 48
- time stamps
 - flight recorder 67
- timer 47
- timer control
 - ADC 1552 Type 1 103
- timer services
 - overview 47
- time-stamping mode
 - ADC Type 4 94
- TJA1041
 - limitations 248
- TJA1054
 - transceiver 248
- transceiver
 - TJA1054 248
- transmitting frames 304
- trigger signals
 - ADC 1552 Type 1 103
 - ADC Class 1 89
 - ADC Type 4 89
- trigger sources
 - ADC 1552 Type 1 103
 - ADC Type 4 90
- troubleshooting
 - loading application fails 397
 - loading application slower than before 397
 - MicroAutoBox 397
 - real-time testing support 397
- TX frames
 - defining 314
- TX signals
 - manipulating 317

U

- upcounter 47
- update mode
 - PW2D (DIOTP3) 221
 - PW2D (DIOTP4) 225
- USB Flight Recorder 69
 - avoiding data loss 71
 - basics 70
 - FTP connection 72
 - handling the data 74
 - limitations 73
 - max. data rate 73
 - overwrite mode 70
 - time base 73
 - USB mass storage device 71
 - USB status LED 73
- using
 - flight recorder 67

W

- waking up a LIN bus 306
- watchdog handling
 - basics 50
 - handling interrupts 52
 - hook function 52
 - restart monitoring 51
 - RTI support 52
 - RTLib support 52
 - stop monitoring 51
 - watchdog timer 51
- word-based subinterrupt handling 40
- working with CAN FD 263
- working with raw data 315

Z

- zero nibble pulse 335