

DS2211 HIL I/O Board

Features

Release 2021-A – May 2021

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	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.

© 2003 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

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

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

Contents

About This Document	7
Introduction to the Features of the DS2211	9
DS2211 Architecture.....	10
Feature Overview.....	12
DS2211 Interfaces.....	14
Sensor and Actuator Interface	17
Standard I/O.....	18
ADC Unit.....	18
DAC Unit.....	20
D/R Converter.....	23
Bit I/O Unit.....	25
Timing I/O.....	27
PWM Signal Measurement.....	27
PWM Signal Generation.....	31
Frequency Measurement (F2D).....	34
Square-Wave Signal Generation (D2F).....	37
Serial Interface	41
Basics on the Serial Interface.....	41
Comparing RS232 and RS422.....	43
Specifying the Baud Rate of the Serial Interface.....	44
Software FIFO Buffer.....	45
Single Edge Nibble Transmission (SENT) Support	47
Basics of the SENT Protocol.....	48
Using the SENT Protocol on a DS2211.....	51
Implementing SENT Transmitters in Simulink.....	53
Implementing SENT Receivers in Simulink.....	56
Implementing SENT Transmitters Using RTLib Functions.....	60
Implementing SENT Receivers Using RTLib Functions.....	63

Angular Processing Unit	71
APU Basics.....	72
APU Overview.....	73
Engine Position Phase Accumulator.....	75
Cascading I/O Boards.....	76
Crankshaft Signal Generator.....	77
Reverse Crankshaft Rotation.....	78
Camshaft Signal Generator.....	81
Spark Event Capture Unit.....	83
Injection Event Capture Unit.....	84
Event Capture Windows.....	88
Continuous Value Capturing.....	92
Complex Comparators.....	93
Angular Processing Unit - Variant.....	97
Basics on Simulating Engine Variants.....	97
Building a Simulink Model for Engine Variants.....	99
Multiple Event Capture Mode of the VAR APU Blockset.....	100
Example of Capturing a Multiple Event with the VAR APU Blockset.....	101
APU Reference.....	104
Crankshaft Sensor Signal Generation.....	104
Camshaft Sensor Signal Generation.....	106
Wave Table Generation.....	108
Spark Event Capture.....	109
Injection Pulse Position and Fuel Amount Measurement.....	111
Features of the Slave DSP	115
Basics of the Slave DSP.....	116
Knock Sensor Simulation.....	117
Wheel Speed Sensor Simulation.....	119
DSP DAC Unit.....	121
DSP Bit I/O Unit and Capture Input Access.....	122
Slave DSP Interrupts.....	123
CAN Support	125
Setting Up a CAN Controller.....	126
Initializing the CAN Controller.....	126
CAN Transceiver Types.....	128
DS2211: Selecting the CAN Controller Frequency.....	132

Defining CAN Messages.....	133
Implementing a CAN Interrupt.....	134
Using RX Service Support.....	135
Removing a CAN Controller (Go Bus Off).....	137
Getting CAN Status Information.....	137
Using the RTI CAN MultiMessage Blockset.....	140
Basics on the RTI CAN MultiMessage Blockset.....	140
Basics on Working with CAN FD.....	145
Basics on Working with a J1939-Compliant DBC File.....	150
Transmitting and Receiving CAN Messages.....	156
Manipulating Signals to be Transmitted.....	159
CAN Signal Mapping.....	163
CAN Signal Mapping.....	163
Interrupts of the DS2211	165
Basics of DS2211 Interrupts.....	165
Interrupt Handling.....	167
Use Scenarios	169
Simulating Ionic Current Measurement.....	170
Basics of Ionic Current Measurement.....	170
How to Simulate Ionic Current Measurement.....	172
Implementing the Simulation of Ionic Current Measurement.....	174
Simulating Engine Variants.....	178
Basics of Modeling Engine Variants.....	178
Implementing the Engine Model.....	179
Reconfiguring the Engine Real-Time Application.....	183
DS2211 Versus DS2210	187
Comparing DS2210 and DS2211 Features.....	188
rti221xConv Conversion Tool.....	188
How to Convert a Model with DS2210 Blocks to a Model with DS2211 Blocks.....	191
How to Convert a Model with DS2211 Blocks to a Model with DS2210 Blocks.....	192
How to Migrate Handcoded Applications for DS2211.....	193
Working with a Real-Time System Containing a DS2210 and a DS2211.....	193

Limitations	195
Quantization Effects.....	196
DS2211 Board Revision.....	196
Conflicting I/O Features.....	197
Limited Number of CAN Messages.....	211
Limitations with RTICANMM.....	213
Limitations with J1939-Support.....	217
Index	219









About This Document

Contents

This document provides feature-oriented access to the reference information you need to implement the functions provided by the DS2211 HIL I/O Board.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
	Indicates a hazard that, if not avoided, could result in property damage.
	Indicates important information that you should take into account to avoid malfunctions.
	Indicates tips that can make your work easier.
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files


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

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

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

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

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

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

Introduction to the Features of the DS2211

DS2211 HIL I/O Board

The DS2211 HIL I/O Board is designed to simulate and measure automotive signals. It combines a variety of typical HIL I/O functions on one board and also contains signal conditioning for typical signal levels of 12 V, 24 V and 42 V automotive systems, including two voltage systems.

Where to go from here

Information in this section

DS2211 Architecture.....	10
The DS2211 consists of several functional units in the architecture.	
Feature Overview.....	12
Introduction of the DS2211 features, summarized in alphabetical order.	
DS2211 Interfaces.....	14
The DS2211 has interfaces for connection to a PHS-bus-based system and for cascading boards.	

Information in other sections

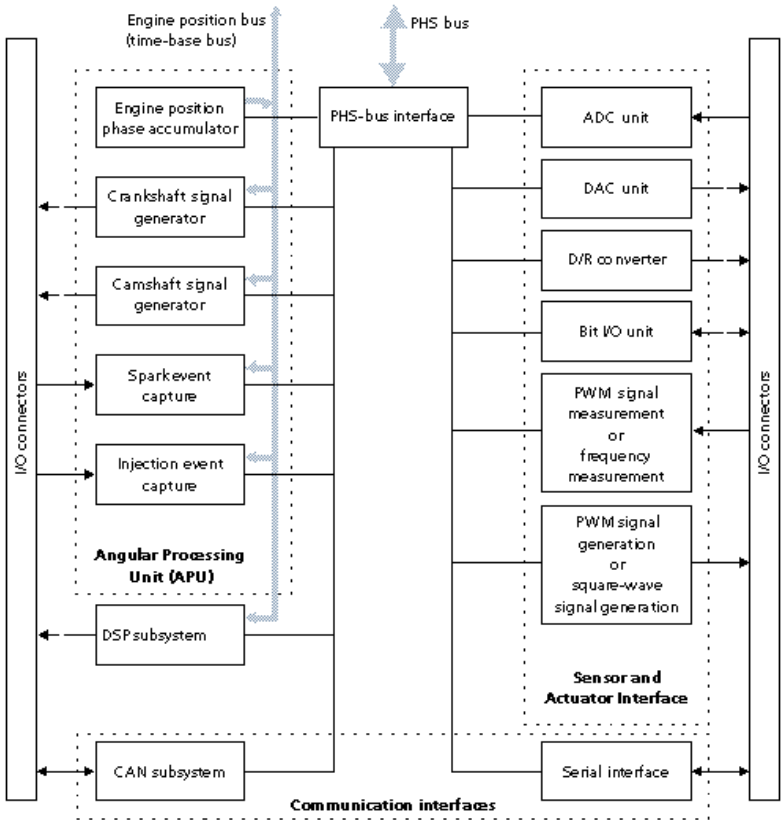
DS2211 Versus DS2210.....	187
The DS2211 is the successor of the DS2210. It is possible to work with real-time systems that have both types of I/O boards. If you have worked with the DS2210, you can migrate your Simulink models and handcoded applications easily for the DS2211.	

DS2211 Architecture

Introduction	The DS2211 includes the following functional units.
Sensor and actuator interface	The DS2211 contains a sensor and actuator interface that provides a typical set of automotive I/O functions, including A/D conversion, digital I/O, wheel speed sensor signal generation, and others.
Angular processing unit	<p>The most important feature of the DS2211 is the angular processing unit (APU), which provides the engine HIL core functions:</p> <ul style="list-style-type: none"> ▪ Crankshaft/camshaft signal generation ▪ Spark event measurement ▪ Injection pulse position and fuel amount measurement ▪ Knock sensor simulation <p>The angular processing unit can be used to simulate different engine variants.</p>
DSP subsystem	The DSP subsystem is based on the TMS320VC33. It includes ready-to-use applications that allow you to generate knock sensor signals or wheel speed sensor signals. As an alternative, you can program the DSP to generate user-specific signals. The serial port of the slave DSP allows you to connect a DS2302 board, for example.
Communication interfaces	<p>The DS2211 provides the following communication interfaces:</p> <ul style="list-style-type: none"> ▪ A standard serial interface (RS232, RS422 based on a Texas Instruments TL 16C550 UART). ▪ A CAN subsystem that is based on the Siemens SAB 80C164 microcontroller. It provides connections to two CAN buses. ▪ SENT (single edge nibble transmission), a protocol used between sensors and ECUs. It is defined in the standard SAE J2716 by the Society of Automotive Engineers (SAE).
Master and slaves	The processor board has access to both the DSP and the CAN subsystems. In terms of interprocessor communication, the processor board is the master, and the DS2211 microcontrollers are slaves.
Voltage systems	The DS2211 supports two independent voltage systems with a voltage in the range 5 ... 60 V. The voltage must come from external power supplies.

Overview of functional units

The illustration shows the functional units of the DS2211.



Legend The illustration uses the following abbreviations:

- ADC: Analog/digital converter
- CAN: Controller area network
- DAC: Digital/analog converter
- D/R: Digital/resistance (converter)
- DSP: Digital signal processor
- PWM: Pulse width modulation

Related topics

References

DS2211 Interfaces.....	14
Feature Overview.....	12

Feature Overview

Introduction	The DS2211 provides the following features, summarized in alphabetical order.
A/D conversion	The ADC unit provides 16 unipolar A/D channels with 14-bit resolution and 20 μ s conversion time for all channels. Refer to ADC Unit on page 18.
Bit I/O	The bit I/O unit provides 16 discrete input lines and 16 discrete outputs. Refer to Bit I/O Unit on page 25.
CAN support	The CAN support serves two CAN controllers that meet the CAN 2.0A (11-bit identifier) and CAN 2.0B (29-bit identifier) specifications. Refer to CAN Support on page 125.
D/A conversion	The DAC unit provides 20 unipolar D/A channels (for user output) with 12-bit resolution and 20 μ s full-scale settling time to 1 least significant bit (LSB). Refer to DAC Unit on page 20.
D/R conversion	The D/R converter provides 10 independent resistance outputs with 16-bit resolution covering a resistance range of 15.8 Ω ... ∞ . Refer to D/R Converter on page 23.
Engine HIL simulation	<p>The angular processing unit provides the following features:</p> <ul style="list-style-type: none"> ▪ Simulation of engine core functions is based on a 16-bit engine position (angle) for an engine cycle of 0 ... 720° and a resolution of 0.011°. The engine position is updated every 250 ns. Refer to Engine Position Phase Accumulator on page 75. ▪ Crankshaft sensor simulation provides one crankshaft wave form output with 8 selectable wave forms. Refer to Crankshaft Signal Generator on page 77 and Crankshaft Sensor Signal Generation on page 104. ▪ Camshaft sensor simulation provides 2 camshaft wave form outputs with analog or digital signal and 2 camshaft wave form outputs with digital signal. Each output has 8 selectable wave forms. Refer to Camshaft Signal Generator on page 81 and Camshaft Sensor Signal Generation on page 106. ▪ 6 different interrupts can be generated depending on the engine position. Refer to APU Overview on page 73 and Interrupts of the DS2211 on page 165.

Event capture

The angular processing unit includes 2 event capture units with the following functions:

- For spark event capture, 8 digital ignition inputs (6 ignition and 2 auxiliary channels) for up to 8-cylinder engines are available. The two auxiliary channels can be individually configured either for various position measurements or for ignition capture. You can define up to two event capture windows for each digital ignition input. Refer to [Spark Event Capture Unit](#) on page 83 and [Spark Event Capture](#) on page 109.
- For injection pulse position and fuel amount measurement, 8 digital injection inputs are available with up to two event capture windows for each channel. Refer to [Injection Event Capture Unit](#) on page 84 and [Injection Pulse Position and Fuel Amount Measurement](#) on page 111.

Tip

If you do not use the ignition capture channels for spark event capture, you can use them additionally for injection capture.

- You can define threshold voltages and hysteresis for the inputs of the spark event capture, injection pulse position and fuel amount measurement. Refer to [Complex Comparators](#) on page 93.

Frequency measurement

For frequency measurement, 24 channels are available with a resolution of 16 bit. Refer to [Frequency Measurement \(F2D\)](#) on page 34.

Interrupt control

The DS2211 PHS bus interrupt controller provides 8 hardware interrupts for the serial interface, the CAN subsystem, and the angular processing unit. Refer to [Interrupts of the DS2211](#) on page 165.

Ionic current signal measurement

Using the interrupt from the complex comparators, the DS2211 can be used for measuring ionic current signals. For an example, see [Simulating Ionic Current Measurement](#) on page 170.

Knock processor

A ready-to-use application of the slave DSP provides 4 knock sensor signal outputs. Each output simulates a knock sensor signal for engines with up to 8 cylinders. Refer to [Knock Sensor Simulation](#) on page 117. Knock sensor simulation and wheel speed sensor simulation cannot be used at the same time.

Pulse generation

9 independent outputs with run-time-adjustable frequencies and duty cycles are available for PWM signal generation. Refer to [PWM Signal Generation](#) on page 31.

Pulse measurement	24 independent input channels are available for PWM signal measurement. Refer to PWM Signal Measurement on page 27.
SENT support	Five SENT transmitters and four SENT receivers can be implemented on a DS2211. Refer to Single Edge Nibble Transmission (SENT) Support on page 47.
Serial interface	The serial interface is based on the standard UART TL16C550C from Texas Instruments, which can be used in the RS232 or RS422 mode. Refer to Serial Interface on page 41.
Square-wave signal generation	Nine channels are available with a resolution of 16 bit for square-wave signal generation. Refer to Square-Wave Signal Generation (D2F) on page 37.
User-specific slave DSP applications	You can program your own slave DSP applications to generate specific signals. Refer to Basics of the Slave DSP on page 116.
Wheel speed sensor simulation	A ready-to-use application of the slave DSP provides four independent wheel speed sensor outputs. Each output simulates one wheel speed sensor signal. Refer to Wheel Speed Sensor Simulation on page 119. Knock sensor simulation and wheel speed sensor simulation cannot be used at the same time.
Limitations	There are some limitations when you work with the DS2211. See Limitations on page 195.
Related topics	References <div> DS2211 Architecture..... 10 DS2211 Interfaces..... 14 </div>

DS2211 Interfaces

Introduction	The DS2211 has interfaces for connection to a PHS-bus-based system and for cascading boards.
Integration into a PHS-bus-based system	As an I/O board, the DS2211 is always part of a PHS-bus-based system. While the DS2211 measures and simulates the signals required, the processor board

performs the calculation of the real-time model. That is, applications using DS2211 I/O features are implemented on the processor board. Together with a processor board, the DS2211 constitutes a basic HIL simulator.

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

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

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

The I/O board support depends on the dSPACE software release which you use. For a list of supported I/O boards, refer to [DS802 Data Sheet \(PHS Bus System Hardware Reference !\[\]\(83f22ed94ec5517769dd76d702c6bfd8_img.jpg\)](#)).

Cascading DS2211 boards

Several DS2211 boards can be cascaded to expand from 8-cylinder simulation to 16-cylinder simulation, or even more via the engine position bus available at the time-base connector. For details, refer to [Cascading I/O Boards](#) on page 76.

Related topics

References

DS2211 Architecture	10
DS2211 Components (PHS Bus System Hardware Reference )	
Feature Overview	12

Sensor and Actuator Interface

Introduction

The sensor and actuator interface (SAI) provides standard I/O components and timing I/O components. Wheel speed sensor simulation is performed by a ready-to-use application implemented on the slave DSP.

Where to go from here

Information in this section

[Standard I/O..... 18](#)

The DS2211 has standard I/O units such as ADC, DAC, D/R converter and bit I/O units.

[Timing I/O.....27](#)

The DS2211 has an timing I/O unit to measure or generate PWM or square-wave signals.

Information in other sections

[Features of the Slave DSP..... 115](#)

The slave DSP is used for wheel speed sensor simulation and knock sensor simulation. It provides a DAC unit, has access to the Bit I/O unit and can be triggered by interrupts.

Standard I/O

Introduction

Standard I/O includes the following components:

- ADC unit
- DAC unit
- D/R converter
- Bit I/O unit

Where to go from here

Information in this section

ADC Unit.....	18
Introduction of the analog/digital converter (ADC) Unit	
DAC Unit.....	20
Introduction of the digital/analog converter (DAC) Unit	
D/R Converter.....	23
Introduction of the digital/resistance (D/R) converter	
Bit I/O Unit.....	25
Introduction of the bit (I/O) unit	

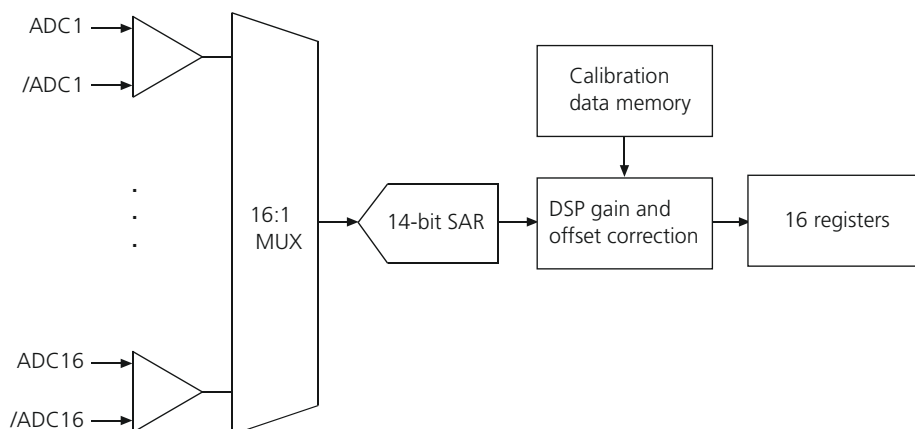
ADC Unit

Characteristics

The ADC unit consists of a successive approximation register (SAR) A/D converter with a 16:1 input multiplexer that provides 16 inputs (ADC1 ... ADC16), with 14-bit resolution each, 20 μ s conversion time for all channels, and one integrated sample/hold for all inputs. All inputs are differential and can measure unipolar signals with 0 ... 60 V input span, lowpass input filters (1st order/–3 dB at 80 kHz), 1 M Ω input impedance to system ground, and continuous \pm 75 V DC overvoltage protection.

The input channels can be read individually or blockwise. The control logic allows conversion of the first 4, 8, 12 or 16 channels (starting from channel 1).

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



Note

ADC inputs are differential inputs. The ADC unit measures the voltage difference of $(ADCx - \overline{ADCx})$. The \overline{ADCx} lines function as individual ground sense lines for each input. They must be connected to the ground of your system near the sensor or to GND at the DS2211 connector, for all ADC channels used.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference\)](#).

I/O mapping

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

A/D Channel	Signal	Connector Pin	Sub-D Pin	Description	Voltage Range
1	ADC1	P1 65	P1B 28	14-bit ADC	$(ADC1 - \overline{ADC1}) = 0 \dots 60 \text{ V}$
	$\overline{ADC1}$	P1 67	P1B 12		
2	ADC2	P1 66	P1A 28	14-bit ADC	$(ADC2 - \overline{ADC2}) = 0 \dots 60 \text{ V}$
	$\overline{ADC2}$	P1 68	P1A 12		
3	ADC3	P1 69	P1B 45	14-bit ADC	$(ADC3 - \overline{ADC3}) = 0 \dots 60 \text{ V}$
	$\overline{ADC3}$	P1 71	P1B 29		
4	ADC4	P1 70	P1A 45	14-bit ADC	$(ADC4 - \overline{ADC4}) = 0 \dots 60 \text{ V}$
	$\overline{ADC4}$	P1 72	P1A 29		
5	ADC5	P1 73	P1B 13	14-bit ADC	$(ADC5 - \overline{ADC5}) = 0 \dots 60 \text{ V}$
	$\overline{ADC5}$	P1 75	P1B 46		
6	ADC6	P1 74	P1A 13	14-bit ADC	$(ADC6 - \overline{ADC6}) = 0 \dots 60 \text{ V}$
	$\overline{ADC6}$	P1 76	P1A 46		

A/D Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range
7	ADC7	P1	77	P1B	30	14-bit ADC	$(\text{ADC7} - \overline{\text{ADC7}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC7}}$	P1	79	P1B	14		
8	ADC8	P1	78	P1A	30	14-bit ADC	$(\text{ADC8} - \overline{\text{ADC8}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC8}}$	P1	80	P1A	14		
9	ADC9	P1	83	P1B	31	14-bit ADC	$(\text{ADC9} - \overline{\text{ADC9}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC9}}$	P1	85	P1B	15		
10	ADC10	P1	84	P1A	31	14-bit ADC	$(\text{ADC10} - \overline{\text{ADC10}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC10}}$	P1	86	P1A	15		
11	ADC11	P1	87	P1B	48	14-bit ADC	$(\text{ADC11} - \overline{\text{ADC11}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC11}}$	P1	89	P1B	32		
12	ADC12	P1	88	P1A	48	14-bit ADC	$(\text{ADC12} - \overline{\text{ADC12}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC12}}$	P1	90	P1A	32		
13	ADC13	P1	91	P1B	16	14-bit ADC	$(\text{ADC13} - \overline{\text{ADC13}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC13}}$	P1	93	P1B	49		
14	ADC14	P1	92	P1A	16	14-bit ADC	$(\text{ADC14} - \overline{\text{ADC14}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC14}}$	P1	94	P1A	49		
15	ADC15	P1	95	P1B	33	14-bit ADC	$(\text{ADC15} - \overline{\text{ADC15}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC15}}$	P1	97	P1B	17		
16	ADC16	P1	96	P1A	33	14-bit ADC	$(\text{ADC16} - \overline{\text{ADC16}}) = 0 \dots 60 \text{ V}$
	$\overline{\text{ADC16}}$	P1	98	P1A	17		

Related topics

References

[ADC Unit \(DS2211 RTLib Reference !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)](#))
[Analog Inputs \(PHS Bus System Hardware Reference !\[\]\(034433b90593e82e5460e34e3ed48e9b_img.jpg\)](#))
[DS2211MUX_ADC_Bx \(DS2211 RTI Reference !\[\]\(5f24500834b50a8307ffe63e419281a9_img.jpg\)](#))
[Introduction to the Features of the DS2211..... 9](#)

DAC Unit

Characteristics

The DAC unit contains D/A converters for user output that provide 20 unipolar analog outputs (DAC1 ... DAC20) with 12-bit resolution (fully monotonic) and 20 μs full-scale settling time to 1 least significant bit (LSB).

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

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

Note

DAC outputs are differential outputs in the range $0 \dots V_{REF}$. Each output has an individual ground sense line (\overline{DACx}), which must be connected to the ground of your system near the actuator or to GND at the DS2211 connector, for all DAC channels used. The DAC controls the voltage difference of $(DACx - \overline{DACx})$. If the external reference is used, the same applies to the $\overline{DAC_REF}$ pin.

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

Output swing referenced to ground, when $DACx$ is connected to a potential other than GND.

The outputs have lowpass output filters (1st order/–3 dB at 100 kHz). The maximum sink/source current of the DAC outputs is $\pm 5\text{ mA}$. They are protected against short circuit to GND or any voltage within $\pm 60\text{ V}$.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(0b5e7e25e8775f7e7e80906ada4f0021_img.jpg\)](#)).

I/O mapping

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

D/A Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range/Output Current
–	DAC_REF	P1	33	P1B	39	DAC external reference voltage input	$V_{REF} = (DAC_REF - \overline{DAC_REF}) = 5 \dots 10\text{ V}$
	$\overline{DAC_REF}$	P1	35	P1B	23	DAC external reference voltage sense line	
1	DAC1	P1	37	P1B	7	12-bit DAC/20 μs	$(DAC1 - \overline{DAC1}) = 0 \dots V_{REF}; \pm 5\text{ mA}$
	$\overline{DAC1}$	P1	39	P1B	40		
2	DAC2	P1	38	P1A	7	12-bit DAC/20 μs	$(DAC2 - \overline{DAC2}) = 0 \dots V_{REF}; \pm 5\text{ mA}$
	$\overline{DAC2}$	P1	40	P1A	40		
3	DAC3	P1	41	P1B	24	12-bit DAC/20 μs	$(DAC3 - \overline{DAC3}) = 0 \dots V_{REF}; \pm 5\text{ mA}$
	$\overline{DAC3}$	P1	43	P1B	8		
4	DAC4	P1	42	P1A	24	12-bit DAC/20 μs	$(DAC4 - \overline{DAC4}) = 0 \dots V_{REF}; \pm 5\text{ mA}$
	$\overline{DAC4}$	P1	44	P1A	8		
5	DAC5	P1	45	P1B	41	12-bit DAC/20 μs	$(DAC5 - \overline{DAC5}) = 0 \dots V_{REF}; \pm 5\text{ mA}$
	$\overline{DAC5}$	P1	47	P1B	25		

D/A Channel	Signal	Connector Pin	Sub-D Pin	Description	Voltage Range/Output Current
6	DAC6	P1 46	P1A 41	12-bit DAC/20 μ s	$(\text{DAC6} - \overline{\text{DAC6}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC6}}$	P1 48	P1A 25		
7	DAC7	P1 51	P1B 42	12-bit DAC/20 μ s	$(\text{DAC7} - \overline{\text{DAC7}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC7}}$	P1 53	P1B 26		
8	DAC8	P1 52	P1A 42	12-bit DAC/20 μ s	$(\text{DAC8} - \overline{\text{DAC8}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC8}}$	P1 54	P1A 26		
9	DAC9	P1 55	P1B 10	12-bit DAC/20 μ s	$(\text{DAC9} - \overline{\text{DAC9}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC9}}$	P1 57	P1B 43		
10	DAC10	P1 56	P1A 10	12-bit DAC/20 μ s	$(\text{DAC10} - \overline{\text{DAC10}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC10}}$	P1 58	P1A 43		
11	DAC11	P1 59	P1B 27	12-bit DAC/20 μ s	$(\text{DAC11} - \overline{\text{DAC11}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC11}}$	P1 61	P1B 11		
12	DAC12	P1 60	P1A 27	12-bit DAC/20 μ s	$(\text{DAC12} - \overline{\text{DAC12}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC12}}$	P1 62	P1A 11		
13	DAC13	P3 7	– –	12-bit DAC/20 μ s	$(\text{DAC13} - \overline{\text{DAC13}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC13}}$	P3 40	– –		
14	DAC14	P3 24	– –	12-bit DAC/20 μ s	$(\text{DAC14} - \overline{\text{DAC14}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC14}}$	P3 8	– –		
15	DAC15	P3 41	– –	12-bit DAC/20 μ s	$(\text{DAC15} - \overline{\text{DAC15}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC15}}$	P3 25	– –		
16	DAC16	P3 9	– –	12-bit DAC/20 μ s	$(\text{DAC16} - \overline{\text{DAC16}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC16}}$	P3 42	– –		
17	DAC17	P3 26	– –	12-bit DAC/20 μ s	$(\text{DAC17} - \overline{\text{DAC17}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC17}}$	P3 10	– –		
18	DAC18	P3 43	– –	12-bit DAC/20 μ s	$(\text{DAC18} - \overline{\text{DAC18}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC18}}$	P3 27	– –		
19	DAC19	P3 11	– –	12-bit DAC/20 μ s	$(\text{DAC19} - \overline{\text{DAC19}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC19}}$	P3 44	– –		
20	DAC20	P3 28	– –	12-bit DAC/20 μ s	$(\text{DAC20} - \overline{\text{DAC20}}) = 0 \dots V_{\text{REF}}; \pm 5 \text{ mA}$
	$\overline{\text{DAC20}}$	P3 12	– –		

Related topics

References

[Analog Outputs \(DAC\) \(PHS Bus System Hardware Reference !\[\]\(d3fb9f94af8b26d1c844efa9a98805b0_img.jpg\)](#))
[DAC Unit \(DS2211 RTLib Reference !\[\]\(78eb1652b591ce460bbb1a853a52e223_img.jpg\)](#))
[DS2211DAC_Bx_Cy \(DS2211 RTI Reference !\[\]\(73569cd2fc0daa66f7f79a4aa32515bb_img.jpg\)](#))
[Introduction to the Features of the DS2211.....9](#)
[Power Supply Outputs \(PHS Bus System Hardware Reference !\[\]\(b12ef55cf189ee2bc15774f5ac32d31a_img.jpg\)](#))

D/R Converter

Introduction

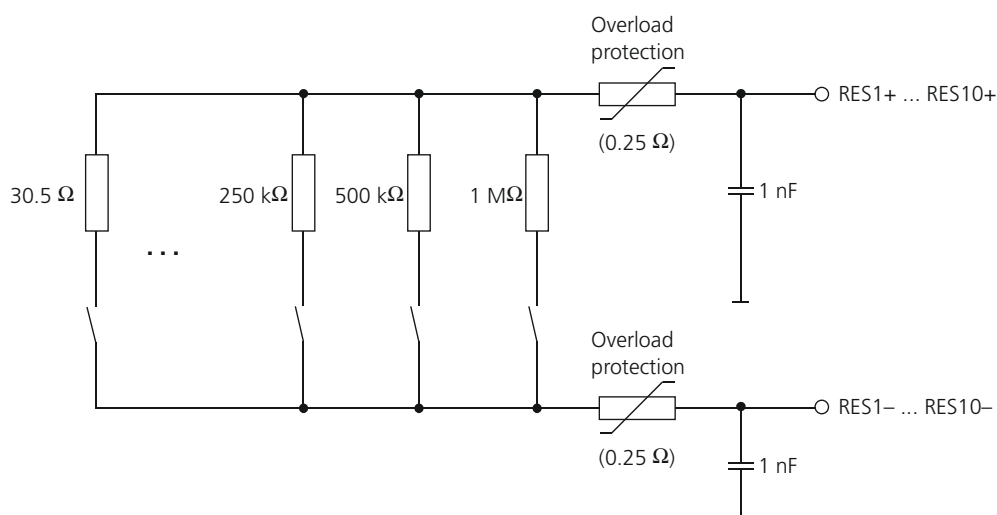
The D/R converter allows the simulation of sensors that have a resistance output, for example, thermistors or RTDs for temperature measurements. You can also use the D/R converter to simulate loose connections. To generate the desired resistance, your application writes a value to the D/R converter, which simulates the resistance electronically between the two output pins (RESx+, RESx-) of the specified resistor channel.

Characteristics

The D/R converter provides 10 independent resistance outputs with 16-bit resolution, covering a resistance range of $15.8\ \Omega \dots \text{M}\Omega$ or infinity. You can set the resistance value to $1\ \text{M}\Omega / x + 0.5\ \Omega$ (with x in the range $0 \dots 65535$).

The maximum output power per channel is $P_{\text{max}} = 250\ \text{mW}$. The maximum output current per channel is $I_{\text{max}} = \pm 80\ \text{mA}$.

The illustration shows a simplified block diagram of one resistor channel.



Tip

You can connect several resistor channels

- In parallel to decrease R_{min} , or
- In series to increase R_{max} and the resolution in higher resistance ranges.

Note

Resistors are not grounded. Terminals must stay in the range $\pm 10\ \text{V}$ to system ground for operation. For simulating a resistor with one terminal grounded, it is recommended that you use RESx- as the grounded terminal. Taking RESx- outside $\pm 5\ \text{V}$ from system GND results in extra resistance error (typically $2\ \Omega$) due to the on-resistance of the solid state switches.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(34b4f260a8587d2e97eeaee361cc357b_img.jpg\)](#)).

I/O mapping

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

Channel Number	Connector Pin		Sub-D Pin		Signal	Description	Power/Output Current
1	P2	3	P2B	34	RES1+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P2	5	P2B	18	RES1–		
2	P2	4	P2A	34	RES2+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P2	6	P2A	18	RES2–		
3	P2	7	P2B	2	RES3+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P2	9	P2B	35	RES3–		
4	P2	8	P2A	2	RES4+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P2	10	P2A	35	RES4–		
5	P2	11	P2B	19	RES5+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P2	13	P2B	3	RES5–		
6	P2	12	P2A	19	RES6+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P2	14	P2A	3	RES6–		
7	P3	18	–	–	RES7+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P3	35	–	–	RES7–		
8	P3	2	–	–	RES8+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P3	19	–	–	RES8–		
9	P3	3	–	–	RES9+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P3	20	–	–	RES9–		
10	P3	36	–	–	RES10+	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	P3	4	–	–	RES10–		

Related topics**References**

[D/R Converter \(DS2211 RTLib Reference !\[\]\(e8fb589d58dad1692debababa5e928b6_img.jpg\)](#))
[DS2211RES_Bx_Cy \(DS2211 RTI Reference !\[\]\(e0595260a7e7840628d1fda6c7638537_img.jpg\)](#))
[Introduction to the Features of the DS2211.....9](#)

Bit I/O Unit

Characteristics

The bit I/O unit contains one 16-bit port for input that provides 16 discrete digital input lines, and one 16-bit port for output that provides 16 discrete digital outputs.

The slave DSP also has access to all digital inputs and outputs. The digital inputs are read by the bit I/O unit and the slave DSP at the same time. The states of the digital outputs are set by the bit I/O unit and the slave DSP using an OR operation.

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

Digital outputs The outputs have push-pull drivers running from an external source (2 independent VBAT rails, each operable in the range 5 ... 60 V). The maximum output current per channel is ± 50 mA. Short circuit protection to GND and VBAT or any voltage between GND and +60 V is implemented by self-resetting fuses, which have been designed for occasional faults only. They are therefore not suitable for failure simulation. The outputs are in their high impedance states after reset and power-up. You can enable or disable all outputs individually, using RTI/RTLib functions. A disabled output is in high impedance state. You can set the supply rail for each output individually (low side, high side VBAT1, or high side VBAT2) using RTI/RTLib functions.

Note

- Before the digital outputs of the bit I/O unit are operated, an external power supply (VBAT) must be connected to at least one of the 2 VBATx supply rails.
- To ensure push-pull driver functionality, the low side switch (LOW) must be set.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(cf531ed27e91483460120fcc057b3901_img.jpg\)](#)).

I/O mapping

The following table shows the mapping of the channel numbers to the corresponding I/O pins of I/O connectors P1 and P2, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. Refer to [Conflicting I/O Features](#) on page 197.

Channel Number	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range
1	DIG_IN1	P1	3	P1B	34	Digital input	12/42 V compatible
2	DIG_IN2	P1	4	P1A	34	Digital input	12/42 V compatible
3	DIG_IN3	P1	5	P1B	18	Digital input	12/42 V compatible
4	DIG_IN4	P1	6	P1A	18	Digital input	12/42 V compatible
5	DIG_IN5	P1	7	P1B	2	Digital input	12/42 V compatible
6	DIG_IN6	P1	8	P1A	2	Digital input	12/42 V compatible
7	DIG_IN7	P1	9	P1B	35	Digital input	12/42 V compatible
8	DIG_IN8	P1	10	P1A	35	Digital input	12/42 V compatible
9	DIG_IN9	P1	11	P1B	19	Digital input	12/42 V compatible
10	DIG_IN10	P1	12	P1A	19	Digital input	12/42 V compatible
11	DIG_IN11	P1	13	P1B	3	Digital input	12/42 V compatible
12	DIG_IN12	P1	14	P1A	3	Digital input	12/42 V compatible
13	DIG_IN13	P1	15	P1B	36	Digital input	12/42 V compatible
14	DIG_IN14	P1	16	P1A	36	Digital input	12/42 V compatible
15	DIG_IN15	P1	17	P1B	20	Digital input	12/42 V compatible
16	DIG_IN16	P1	18	P1A	20	Digital input	12/42 V compatible
1	DIG_OUT1	P2	32	P2A	6	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
2	DIG_OUT2	P2	34	P2A	39	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
3	DIG_OUT3	P2	36	P2A	23	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
4	DIG_OUT4	P2	38	P2A	7	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
5	DIG_OUT5	P2	40	P2A	40	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
6	DIG_OUT6	P2	42	P2A	24	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
7	DIG_OUT7	P2	44	P2A	8	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
8	DIG_OUT8	P2	46	P2A	41	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
9	DIG_OUT9	P2	50	P2A	9	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
10	DIG_OUT10	P2	52	P2A	42	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
11	DIG_OUT11	P2	54	P2A	26	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
12	DIG_OUT12	P2	56	P2A	10	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
13	DIG_OUT13	P2	58	P2A	43	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
14	DIG_OUT14	P2	60	P2A	27	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
15	DIG_OUT15	P2	62	P2A	11	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
16	DIG_OUT16	P2	64	P2A	44	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA

Related topics

References

[Bit I/O Unit \(DS2211 RTI Reference !\[\]\(0f848bbd71cef6b345273b16f905912a_img.jpg\)](#))

Timing I/O

Introduction

- Timing I/O includes the following components:
- PWM signal measurement
 - PWM signal generation
 - Frequency measurement
 - Square-wave signal generation

Where to go from here

Information in this section

PWM Signal Measurement.....	27
PWM signal measurement is used to capture digital signals in hardware-in-the-loop applications.	
PWM Signal Generation.....	31
9 independent PWM outputs are available for the generation of square-wave signals with a run-time adjustable frequency and run-time adjustable duty cycle.	
Frequency Measurement (F2D).....	34
24 independent channels are available to measure the frequency of square-wave signals.	
Square-Wave Signal Generation (D2F).....	37
9 independent channels are available to generate square-wave signals with variable frequencies.	

PWM Signal Measurement

Introduction

In hardware-in-the-loop applications, PWM signal measurement is used to capture digital signals. To evaluate frequencies and duty cycles, digital pulses have to be recorded at high speed and transferred to the processor board that performs the analysis.

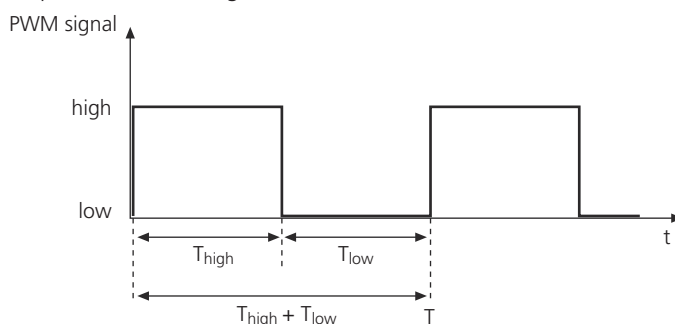
Note

- The channels can be used for either frequency measurement or PWM signal measurement.
- Due to quantization effects, you will encounter considerable deviations between the input PWM period T_P and the measured PWM period, especially for higher PWM frequencies. To avoid a poor frequency resolution, you should therefore select the frequency range with the best possible resolution (resolution values as small as possible). Refer to [Quantization Effects](#) on page 196.

Characteristics

For analysis of the duty cycle, frequency, and low and high periods of PWM type signals, 24 independent PWM channels are available. 16 of these are shared with bit I/O channels.

At 16-bit resolution, you can measure the duty cycle within 0 ... 100% for PWM frequencies in the range 0.01 Hz ... 100 kHz.



The measurements of the T_{low} and T_{high} periods are used to calculate

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

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

Measurement range

Depending on the prescaler setting, the periods can be measured within the following limits and resolutions:

Range	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
1	200 ns	10 μ s	3.28 ms	50 ns
2	400 ns	10 μ s	6.55 ms	100 ns
3	800 ns	10 μ s	13.1 ms	200 ns
4	1.6 μ s	10 μ s	26.2 ms	400 ns
5	3.2 μ s	10 μ s	52.4 ms	800 ns
6	6.4 μ s	10 μ s	105 ms	1.6 μ s

Range	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
7	12.8 μ s	12.8 μ s	210 ms	3.2 μ s
8	25.6 μ s	25.6 μ s	419 ms	6.4 μ s
9	51.2 μ s	51.2 μ s	839 ms	12.8 μ s
10	102 μ s	102 μ s	1.68 s	25.6 μ s
11	205 μ s	205 μ s	3.36 s	51.2 μ s
12	410 μ s	410 μ s	6.71 s	102 μ s
13	819 μ s	819 μ s	13.4 s	205 μ s
14	1.64 ms	1.64 ms	26.8 s	410 μ s
15	3.28 ms	3.28 ms	53.6 s	819 μ s
16	6.55 ms	6.55 ms	107.3 s	1.64 ms

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

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

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

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

Note

Signal periods and resolution

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

Update mode

The update mode describes the time interval when the measured values are updated. The DS2211 supports two different update modes:

Asynchronous 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 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.

Input voltage

The inputs are 12/42 V compatible (fully operational up to 60 V). After software initialization, the input threshold is set to 2.5 V. You can set the input threshold for all digital inputs in the range 1 ... 23.8 V using RTI/RTLib functions.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(d3fb9f94af8b26d1c844efa9a98805b0_img.jpg\)\)](#).

I/O mapping

The following table shows the mapping of the PWM channel numbers to the corresponding I/O pins of I/O connector P1, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 197.

PWM Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage
1	P1	21	P1B	37	PWM_IN1	PWM signal measurement	12/42 V compatible
2	P1	22	P1A	37	PWM_IN2	PWM signal measurement	12/42 V compatible
3	P1	23	P1B	21	PWM_IN3	PWM signal measurement	12/42 V compatible
4	P1	24	P1A	21	PWM_IN4	PWM signal measurement	12/42 V compatible
5	P1	25	P1B	5	PWM_IN5	PWM signal measurement	12/42 V compatible
6	P1	26	P1A	5	PWM_IN6	PWM signal measurement	12/42 V compatible
7	P1	27	P1B	38	PWM_IN7	PWM signal measurement	12/42 V compatible
8	P1	28	P1A	38	PWM_IN8	PWM signal measurement	12/42 V compatible
9	P1	3	P1B	34	PWM_IN9 (DIG_IN1)	PWM signal measurement (shared with digital input)	12/42 V compatible
10	P1	4	P1A	34	PWM_IN10 (DIG_IN2)	PWM signal measurement (shared with digital input)	12/42 V compatible
11	P1	5	P1B	18	PWM_IN11 (DIG_IN3)	PWM signal measurement (shared with digital input)	12/42 V compatible
12	P1	6	P1A	18	PWM_IN12 (DIG_IN4)	PWM signal measurement (shared with digital input)	12/42 V compatible
13	P1	7	P1B	2	PWM_IN13 (DIG_IN5)	PWM signal measurement (shared with digital input)	12/42 V compatible
14	P1	8	P1A	2	PWM_IN14 (DIG_IN6)	PWM signal measurement (shared with digital input)	12/42 V compatible
15	P1	9	P1B	35	PWM_IN15 (DIG_IN7)	PWM signal measurement (shared with digital input)	12/42 V compatible
16	P1	10	P1A	35	PWM_IN16 (DIG_IN8)	PWM signal measurement (shared with digital input)	12/42 V compatible
17	P1	11	P1B	19	PWM_IN17 (DIG_IN9)	PWM signal measurement (shared with digital input)	12/42 V compatible
18	P1	12	P1A	19	PWM_IN18 (DIG_IN10)	PWM signal measurement (shared with digital input)	12/42 V compatible
19	P1	13	P1B	3	PWM_IN19 (DIG_IN11)	PWM signal measurement (shared with digital input)	12/42 V compatible

PWM Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage
20	P1	14	P1A	3	PWM_IN20 (DIG_IN12)	PWM signal measurement (shared with digital input)	12/42 V compatible
21	P1	15	P1B	36	PWM_IN21 (DIG_IN13)	PWM signal measurement (shared with digital input)	12/42 V compatible
22	P1	16	P1A	36	PWM_IN22 (DIG_IN14)	PWM signal measurement (shared with digital input)	12/42 V compatible
23	P1	17	P1B	20	PWM_IN23 (DIG_IN15)	PWM signal measurement (shared with digital input)	12/42 V compatible
24	P1	18	P1A	20	PWM_IN24 (DIG_IN16)	PWM signal measurement (shared with digital input)	12/42 V compatible

Related topics

Basics

[PWM Signal Generation..... 31](#)

References

[PWM Signal Measurement \(DS2211 RTI Reference !\[\]\(8d0f0e0fe25b320c33272c52aec1fbca_img.jpg\)\)](#)

PWM Signal Generation

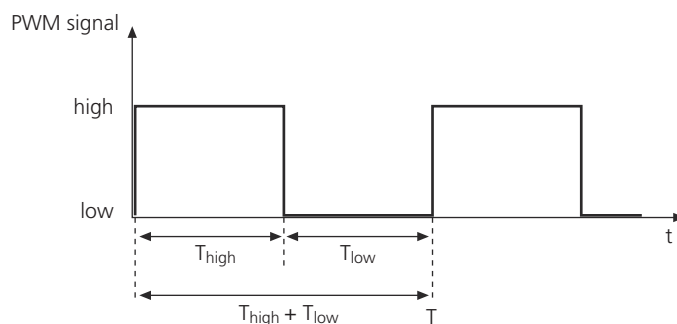
Characteristics

9 independent PWM outputs are available for the generation of square-wave signals with a run-time adjustable frequency and run-time adjustable duty cycle.

Note

- Before operating the digital outputs of PWM signal generation, you must connect an external power supply (V_{Bat}) to at least one of the two VBAT supply rails.
- The channels can be used for either square-wave signal generation or PWM signal generation.
- Due to quantization effects, you may encounter considerable deviations between the desired PWM period T_P and the generated PWM period, especially for higher PWM frequencies. To avoid a poor frequency resolution, you should therefore select the frequency range with the best possible resolution (resolution values as small as possible). Refer to [Quantization Effects](#) on page 196.

At 16-bit resolution, you can set the duty cycle in the range 0 ... 100% for PWM frequencies in the range 0.01 Hz ... 100 kHz. The duty cycle values 0 and 1 yield a constant low or constant high output signal. The following illustration shows how the duty cycle = $(T_{\text{high}} / (T_{\text{low}} + T_{\text{high}}))$ is defined.



Limits and resolution

Depending on the prescaler setting, the signals can be generated within the following limits and resolutions:

Range	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
1	100 ns	10 μ s	3.28 ms	50 ns
2	200 ns	10 μ s	6.55 ms	100 ns
3	400 ns	10 μ s	13.1 ms	200 ns
4	800 ns	10 μ s	26.2 ms	400 ns
5	1.6 μ s	10 μ s	52.4 ms	800 ns
6	3.2 μ s	10 μ s	105 ms	1.6 μ s
7	6.4 μ s	10 μ s	210 ms	3.2 μ s
8	12.8 μ s	12.8 μ s	419 ms	6.4 μ s
9	25.6 μ s	25.6 μ s	839 ms	12.8 μ s
10	51.2 μ s	51.2 μ s	1.68 s	25.6 μ s
11	102 μ s	102 μ s	3.36 s	51.2 μ s
12	205 μ s	205 μ s	6.71 s	102 μ s
13	410 μ s	410 μ s	13.4 s	205 μ s
14	819 μ s	819 μ s	26.8 s	410 μ s
15	1.64 ms	1.64 ms	53.6 s	819 μ s
16	3.28 ms	3.28 ms	107.3 s	1.64 ms

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

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

Output voltage

The outputs have push-pull drivers running from an external source (2 independent VBAT rails, each of which can operate in the range 5 ... 60 V). The maximum output current per channel is ± 50 mA. Short circuit protection to GND and VBAT or any voltage between GND and +60 V is implemented by self-resetting fuses, which have been designed for occasional faults only. They are therefore not suitable for failure simulation. The outputs are in their high impedance states after reset and power-up.

You can enable or disable all outputs individually, using RTI/RTLib functions. A disabled output is in high impedance state. You can set the supply rail for each output individually (low side, high side VBAT1, or high side VBAT2) using RTI/RTLib functions.

Note

- Before the digital outputs of the bit I/O unit are operated, an external power supply (VBAT) must be connected to at least one of the 2 VBATx supply rails.
- To ensure push-pull driver functionality, the low side switch (LOW) must be set.

Update mode

The update mode describes the time interval when the new values for the duty cycle is set. The DS2211 supports two different update modes:

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

Note

For PWM signal generation with *asynchronous* update, it is possible that a high or low pulse is cut off. This occurs when the new T_{high} or T_{low} value is shorter than the current one and exceeds the time which has elapsed in the current T_{high} or T_{low} period, respectively. The result is a non-constant PWM period during update (i.e. actual $T_{\text{high}} + T_{\text{low}}$). If this is not desirable, use the synchronous mode instead.

Synchronous The new values are updated at the end of the next T_{low} period of the PWM signal. The update is synchronous to the period.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(95b425611cbd2b8716a140cf67c81822_img.jpg\)](#)).

I/O mapping

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

I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 197.

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

Related topics

References

[PWM Signal Generation \(DS2211 RTI Reference !\[\]\(339a16584d5da0f0a3ca4e9ec17bf6a1_img.jpg\)](#))

Frequency Measurement (F2D)

Characteristics

24 independent channels are available to measure the frequency of square-wave signals.

Note

The channels can be used for either frequency measurement or PWM signal measurement.

At 21-bit resolution, you can measure frequencies in the range 0.3 mHz ... 100 kHz.

Measurement range

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

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	100 kHz	50 ns
2	4.77 Hz	100 kHz	100 ns
3	2.39 Hz	100 kHz	200 ns

Range	Minimum Frequency	Maximum Frequency	Resolution
4	1.19 Hz	100 kHz	400 ns
5	0.60 Hz	100 kHz	800 ns
6	0.30 Hz	100 kHz	1.6 μ s
7	0.15 Hz	78.12 kHz	3.2 μ s
8	75 mHz	39.06 kHz	6.4 μ s
9	38 mHz	19.53 kHz	12.8 μ s
10	19 mHz	9.76 kHz	25.6 μ s
11	10 mHz	4.88 kHz	51.2 μ s
12	5.0 mHz	2.44 kHz	102 μ s
13	2.5 mHz	1.22 kHz	205 μ s
14	1.2 mHz	610.35 Hz	410 μ s
15	0.6 mHz	305.17 Hz	819 μ s
16	0.3 mHz	152.59 Hz	1.64 ms

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

Range	Restriction
Frequency < f_{\min}	The measured frequency is measured as a 0 Hz signal.
Frequency > f_{\max}	Faulty measurement because of quantization problems, refer to Quantization Effects on page 196.

Note

Signal periods and resolution

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

Input voltage

The inputs are 12/42 V compatible (fully operational up to 60 V). The default input threshold value is 2.5 V. You can set the input threshold for all digital inputs in the range 1 ... 23.8 V using RTI/RTLib functions.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(870f5d5e9c0d57485634be3ecf52f3ca_img.jpg\)](#)).

I/O mapping

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

features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 197.

PWM Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage
1	P1	21	P1B	37	PWM_IN1	Frequency measurement	12/42 V compatible
2	P1	22	P1A	37	PWM_IN2	Frequency measurement	12/42 V compatible
3	P1	23	P1B	21	PWM_IN3	Frequency measurement	12/42 V compatible
4	P1	24	P1A	21	PWM_IN4	Frequency measurement	12/42 V compatible
5	P1	25	P1B	5	PWM_IN5	Frequency measurement	12/42 V compatible
6	P1	26	P1A	5	PWM_IN6	Frequency measurement	12/42 V compatible
7	P1	27	P1B	38	PWM_IN7	Frequency measurement	12/42 V compatible
8	P1	28	P1A	38	PWM_IN8	Frequency measurement	12/42 V compatible
9	P1	3	P1B	34	PWM_IN9 (DIG_IN1)	Frequency measurement (shared with digital input)	12/42 V compatible
10	P1	4	P1A	34	PWM_IN10 (DIG_IN2)	Frequency measurement (shared with digital input)	12/42 V compatible
11	P1	5	P1B	18	PWM_IN11 (DIG_IN3)	Frequency measurement (shared with digital input)	12/42 V compatible
12	P1	6	P1A	18	PWM_IN12 (DIG_IN4)	Frequency measurement (shared with digital input)	12/42 V compatible
13	P1	7	P1B	2	PWM_IN13 (DIG_IN5)	Frequency measurement (shared with digital input)	12/42 V compatible
14	P1	8	P1A	2	PWM_IN14 (DIG_IN6)	Frequency measurement (shared with digital input)	12/42 V compatible
15	P1	9	P1B	35	PWM_IN15 (DIG_IN7)	Frequency measurement (shared with digital input)	12/42 V compatible
16	P1	10	P1A	35	PWM_IN16 (DIG_IN8)	Frequency measurement (shared with digital input)	12/42 V compatible
17	P1	11	P1B	19	PWM_IN17 (DIG_IN9)	Frequency measurement (shared with digital input)	12/42 V compatible
18	P1	12	P1A	19	PWM_IN18 (DIG_IN10)	Frequency measurement (shared with digital input)	12/42 V compatible
19	P1	13	P1B	3	PWM_IN19 (DIG_IN11)	Frequency measurement (shared with digital input)	12/42 V compatible
20	P1	14	P1A	3	PWM_IN20 (DIG_IN12)	Frequency measurement (shared with digital input)	12/42 V compatible
21	P1	15	P1B	36	PWM_IN21 (DIG_IN13)	Frequency measurement (shared with digital input)	12/42 V compatible
22	P1	16	P1A	36	PWM_IN22 (DIG_IN14)	Frequency measurement (shared with digital input)	12/42 V compatible
23	P1	17	P1B	20	PWM_IN23 (DIG_IN15)	Frequency measurement (shared with digital input)	12/42 V compatible
24	P1	18	P1A	20	PWM_IN24 (DIG_IN16)	Frequency measurement (shared with digital input)	12/42 V compatible

Related topics

References

[Frequency Measurement \(DS2211 RTI Reference !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)\)](#)

Square-Wave Signal Generation (D2F)

Characteristics

9 independent channels are available to generate square-wave signals with variable frequencies.

Note

- Before the digital outputs of PWM signal generation are operated, an external power supply (V_{Bat}) must be connected to at least one of the two VBAT supply rails.
- The channels can be used for either square-wave signal generation or PWM signal generation.

Using a 20-bit resolution, you can generate frequencies in the range 0.3 mHz ... 100 kHz.

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

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	100 kHz	50 ns
2	4.77 Hz	100 kHz	100 ns
3	2.39 Hz	100 kHz	200 ns
4	1.19 Hz	100 kHz	400 ns
5	0.60 Hz	100 kHz	800 ns
6	0.30 Hz	100 kHz	1.6 μ s
7	0.15 Hz	100 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	102 μ s
13	2.5 mHz	2.44 kHz	205 μ s
14	1.2 mHz	1.22 kHz	410 μ s

Range	Minimum Frequency	Maximum Frequency	Resolution
15	0.6 mHz	610.35 Hz	819 μ s
16	0.3 mHz	305.18 Hz	1.64 ms

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

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

Output voltage

The outputs have push-pull drivers running from an external source (2 independent VBAT rails, each of which can operate in the range 5 ... 60 V). The maximum output current per channel is ± 50 mA. Short circuit protection to GND and VBAT or any voltage between GND and +60 V is implemented by self-resetting fuses, which have been designed for occasional faults only. They are therefore not suitable for failure simulation. The outputs are in their high impedance states after reset and power-up. You can enable or disable all outputs individually, using RTI/RTLib functions. A disabled output is in high impedance state. Using RTI/RTLib functions, you can set the supply rail for each output individually (low side, high side VBAT1, or high side VBAT2).

Note

- Before the digital outputs of the bit I/O unit are operated, an external power supply (VBAT) must be connected to at least one of the two VBATx supply rails.
- To ensure push-pull driver functionality, the low side switch (LOW) must be set.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(73002692dd5e7a64e60946be3158e719_img.jpg\)](#)).

I/O mapping

The following table shows the mapping of the logical PWM channel numbers to the corresponding I/O pins of I/O connectors P2 and P3, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. Refer to [Conflicting I/O Features](#) on page 197.

PWM Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage
1	P2	31	P2B	6	PWM_OUT1	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
2	P2	33	P2B	39	PWM_OUT2	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA

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

Related topics

References

[Square-Wave Signal Generation \(DS2211 RTI Reference !\[\]\(9dfdaff1d86ba3c1f8353b4d1b61b8c5_img.jpg\)\)](#)

Serial Interface

Where to go from here

Information in this section

Basics on the Serial Interface.....	41
Provides information on board's <i>universal asynchronous receiver and transmitter</i> (UART) for performing serial asynchronous communication with external devices.	
Comparing RS232 and RS422.....	43
The DS2211 allows you to use the RS232 or RS422 transceiver mode.	
Specifying the Baud Rate of the Serial Interface.....	44
Provides information on the baud rate that you can specify for the board's serial interface.	
Software FIFO Buffer.....	45
The serial interface features a memory section (software FIFO buffer) providing the UART with additional space for data storage. The buffer stores data that will be written to (transmit buffer) or was read by (receive buffer) the UART.	

Basics on the Serial Interface

UART

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

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

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

communication device, or to a network of devices. For details, see [Comparing RS232 and RS422](#) on page 43.

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

For details, see [Specifying the Baud Rate of the Serial Interface](#) on page 44.

- Selectable number of data bits, parity bit and stop bits
- Software FIFO buffer of selectable size. For details, see [Software FIFO Buffer](#) on page 45.

Serial data transfer

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

UART interrupt

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

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

For information on the interrupt handling, see [Basics of DS2211 Interrupts](#) on page 165.

RTI/RTLib support

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

- RTI: [Serial Interface \(DS2211 RTI Reference !\[\]\(adb0331d22f78481623cc605df40612a_img.jpg\)](#))
- RTLib: [Serial Interface Communication \(DS2211 RTLib Reference !\[\]\(7e3a264c08e10137510d1aa76522412b_img.jpg\)](#))

I/O mapping

The following table shows pins used by the serial interface.

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

Note

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

Comparing RS232 and RS422

Introduction

The DS2211 allows you to use the RS232 or RS422 transceiver mode.

RS232 transceiver mode

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

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

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

RS422 transceiver mode

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

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

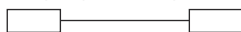
Cable length and baud rate Since the RS422 transceiver modes use differential signals, the effects of induced noise signals that appear as common mode voltages on a network are reduced. Compared to the RS232 transceiver mode, higher baud rates between transmitters and receivers are therefore possible.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

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

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

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

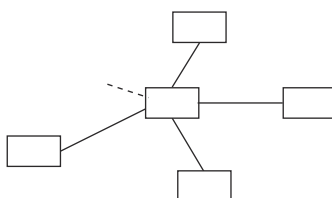
- Simple point-to-point connections



- Daisy-chain connections



- Backbone connections



Related topics

Basics

Serial Interface.....	41
Software FIFO Buffer.....	45

Specifying the Baud Rate of the Serial Interface

Oscillator frequency

The serial interface of the DS2211 is driven by an oscillator with a frequency $f_{\text{OSC}} = 16 \text{ MHz}$.

Baud rate range

Depending on the selected transceiver mode, you can specify the baud rate for serial communication with the DS2211 in the following range:

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

Available baud rates

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

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

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

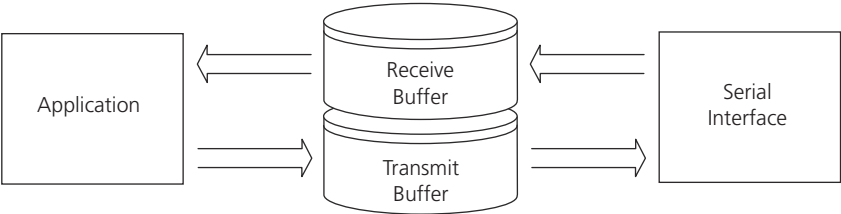
When you specify a baud rate in RTI or RTLib, the closest available baud rate is actually used for serial communication. For example, if you specify 70,000 baud as the baud rate, the baud rate used is 71,429 baud.

Software FIFO Buffer

Introduction

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

The following illustration shows the buffer principle:



Transmit buffer

Data to be transmitted usually is sent immediately.

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

Receive buffer

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

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

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

Related topics

Basics	
Comparing RS232 and RS422.....	43

Single Edge Nibble Transmission (SENT) Support

Introduction

You can implement the SENT protocol (SAE J2716 JAN2010 and earlier) on a DS2211 board using RTI blocks or RTLib functions. This feature is supported only for DS2211 boards starting from a certain board revision.

Where to go from here

Information in this section

Basics of the SENT Protocol.....	48
Provides basic information on the SENT (single edge nibble transmission) protocol.	
Using the SENT Protocol on a DS2211.....	51
Provides information on the board revision which can be used for implementing SENT protocol and other general information.	
Implementing SENT Transmitters in Simulink.....	53
Provides information on how you can implement SENT transmitters in a Simulink model.	
Implementing SENT Receivers in Simulink.....	56
Provides information on how you can implement a SENT receiver in a Simulink model.	
Implementing SENT Transmitters Using RTLib Functions.....	60
Provides information on the parameters for a SENT transmitter and explains how you can implement it in a handcoded model.	
Implementing SENT Receivers Using RTLib Functions.....	63
Provides information on the parameters for a SENT receiver and explains how you can implement it in a handcoded model.	

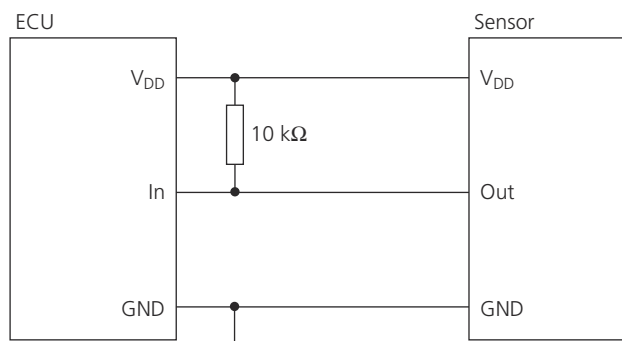
Basics of the SENT Protocol

Introduction

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 measured 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

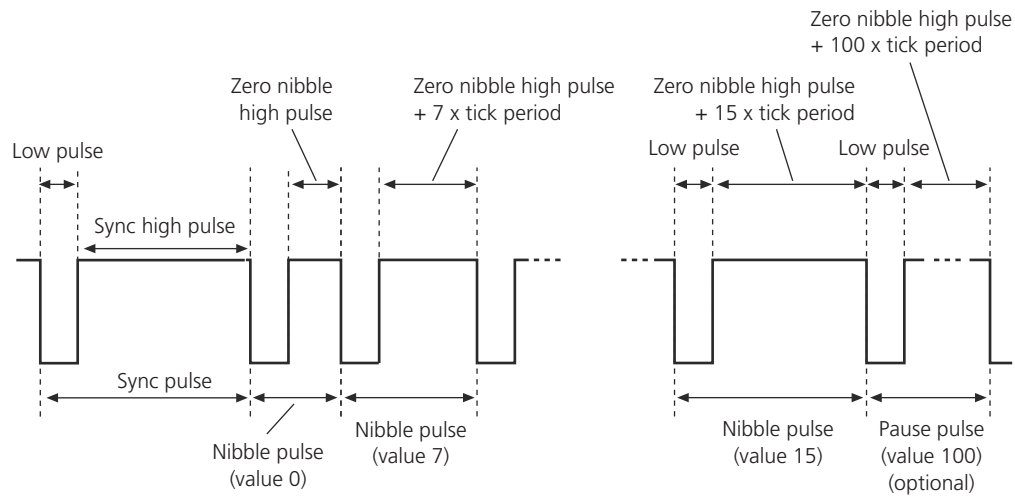
Every SENT message consists of a sync (synchronization) pulse followed by one or more nibble pulses. A nibble represents 4 bit. For regular SENT specification, the first nibble is a status nibble and the last nibble is a CRC nibble. All other nibbles contain data information.

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 default 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 SENT pulse is a digital signal pulse consisting of a low pulse and a high pulse. There are three different SENT pulses, a sync pulse, a nibble pulse, and a pause pulse. The meaning of a SENT pulse is recognized by its pulse duration and position in a message. The pulse length is measured between two consecutive falling edges.

Low pulse A low pulse marks the beginning of every SENT pulse. Its length is defined by the number of tick periods. The default is 5 tick periods.

High pulse A high pulse is 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 A sync high pulse is the high part of a SENT synchronization pulse. Its length is defined by the number of tick periods. The default is 51 tick periods.

Zero nibble high pulse A zero nibble pulse is the high part of a SENT nibble pulse and a pause pulse with a value of 0. Its length is defined by the number of tick periods. The default is 7 tick periods.

Sync pulse A sync pulse is the beginning of every SENT message and 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 nibble pulse is a SENT pulse consisting of a low pulse and a specific high pulse. The minimum length is the length of the zero nibble high pulse. The length of the high pulse defines the data value transmitted by the nibble. The 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 A pause pulse is a SENT pulse which can be optionally specified. It is used to fill up a SENT message to get a constant message length. If the length of the nibble pulses differs, the pause pulse length can be adjusted to fill up the message to a constant length.

A pause pulse is a SENT pulse consisting of a low pulse and a specific high pulse. The minimum length is the length of the zero nibble high pulse. The length of the high pulse defines the data value transmitted by the pause. The length is defined by the number of tick periods. The pause pulse length can be determined with the following formula:

$$\text{Pause pulse} = \text{Low pulse} + \text{Zero nibble high pulse} + \text{value} \cdot \text{tick period}$$

Maximum time between two read operations

If data is lost, the reason might be that the time between two read operations is too long. The maximum time between two read operations without data loss can be calculated as follows:

$$\begin{aligned} T_{\text{Read, max}} &= \text{Expected number of messages} \cdot T_{\text{Message, min}} \\ T_{\text{Message, min}} &= [(\text{LowTics} + \text{SyncHighTics}) + \text{NibbleCount} \cdot (\text{LowTics} + \\ &\quad (\text{without pause pulse}) \text{ZeroNibbleHighTics})] \cdot [(1 - \text{CD}) \cdot \text{TicPeriod}] \\ T_{\text{Message, min}} &= [(\text{LowTics} + \text{SyncHighTics}) + (\text{NibbleCount} + 1) \cdot \\ &\quad (\text{with pause pulse}) (\text{LowTics} + \text{ZeroNibbleHighTics})] \cdot [(1 - \text{CD}) \cdot \text{TicPeriod}] \end{aligned}$$

Where

$T_{\text{Read, max}}$	is the maximum time between two consecutive read operations without loss of data.
$T_{\text{Message, min}}$	is the minimum possible message duration
LowTics	is the number of ticks for a low pulse
SyncHighTics	is the number of ticks for a synchronization high pulse
NibbleCount	is the number of nibbles included in every SENT message
ZeroNibbleHighTics	is the number of ticks for the high part of a SENT nibble pulse with a value of 0
CD	is the clock drift (tick period tolerance)
TicPeriod	is the 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 μ s	51	7	5	6	0.2	2.0 ms

TicPeriod	Tics			NibbleCount	CD	T _{Read, max.}
	SyncHigh	ZeroNibbleHigh	Low			
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_{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_{Read, max} can be calculated as follows:

$$T_{\text{Read, max}} = \text{ExpectedMsgLen} \cdot \text{TicPeriod} \cdot (1 - \text{CD})$$

Using the SENT Protocol on a DS2211

Introduction

You can implement the SENT (single edge nibble transmission) protocol on a DS2211 using RTI blocks or RTLib functions.

Supported SENT version

The DS2211 supports SENT in version SAE J2716 JAN2010 and earlier.

Board revisions supporting SENT

You can implement the SENT protocol only on DS2211 boards with the following board versions:

Board Revision ¹⁾	FPGA Revision
02	≥ 004
03	≥ 004
04	002...009 or ≥ 012
05 or higher	001 or higher

¹⁾ Refer to [DS2211 Board Revision](#) on page 196

According to the SAE J2716 JAN2010 specification, the following features are required:

- SENT transmitter:
 - Pause pulse
 - Tick period > 51.1875 µs (up to 204.7875 µs)
- SENT receiver:
 - Timeout detection

To use these features, the DS2211 board must have on of the following board/FPGA versions or later.

Board Revision ¹⁾	FPGA Revision
02	≥ 005
03	≥ 005
04	003...009 or ≥ 015
05 or higher	001 or higher

¹⁾ Refer to [DS2211 Board Revision](#) on page 196

Transmitting SENT messages from a DS2211

There are five independent SENT transmitters on a DS2211 board. Every transmitter has a FIFO that stores messages to be sent.

The CRC nibble, the status nibble and the pause pulse length of a fixed message length are not evaluated by the transmitter. These values must be written by the model like every other data nibble.

The SENT transmitter channels [1...5] use the digital outputs DIG_OUT1 ... DIG_OUT5 of a DS2211 board.

Electrical connection A SENT transmitter is normally connected to the SENT receiver via three lines:

- **V_{DD} line:** The V_{DD} line provides the power supply for the sensor, normally 5 V. It can be connected to VBat1 or VBat2, which are connected to the DS2211 or an additional 5 V-power supply. If you use an additional 5 V power supply, a resistor of 10 kΩ must be connected between the V_{DD} and the data line.
- **GND line:** The GND line must be connected to the ground of the power supply used and the GND of the DS2211 board.
- **Data line:** The data line must be connected to the channel which is used for transmission. Note that the channel is shared with other I/O features (digital output) which can therefore not be used.

To use the electrical connection, some configuration must be done via software. Refer to [Implementing SENT Transmitters in Simulink](#) on page 53 or [Implementing SENT Transmitters Using RTLib Functions](#) on page 60.

I/O mapping The following table shows the mapping of the channel numbers to the corresponding I/O pins of I/O connector P2, as used in RTLib. Some I/O features of the DS2211 conflict with each other. Refer to [Conflicting I/O Features](#) on page 197.

Channel Number	Signal	Connector Pin		Sub-D Pin		Description
1	DIG_OUT1	P2	32	P2A	6	Digital output
2	DIG_OUT2	P2	34	P2A	39	Digital output
3	DIG_OUT3	P2	36	P2A	23	Digital output
4	DIG_OUT4	P2	38	P2A	7	Digital output
5	DIG_OUT5	P2	40	P2A	40	Digital output

Receiving SENT messages on a DS2211

There are four independent SENT receivers on a DS2211 board. The receiver channels [1 ... 4] use digital inputs DIG_IN1 ... DIG_IN4.

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 normally connected to the SENT transmitter via three lines:

- V_{DD} line: The V_{DD} line provides the power supply for the sensor, normally 5 V. The threshold level of the DS2211's digital input channels must be 2.5 V, which is the default value of the channels.
- GND line: The GND line must be connected to the ground of the power supply used and the GND of the DS2211 board.
- Data line: The data line must be connected to the channel which is used for receiving. Note that the channel is shared with other I/O features (digital input, square-wave signal measurement, PWM signal measurement) which can therefore not be used.

To use the electrical connection, some configuration must be done via software. Refer to [Implementing SENT Receivers in Simulink](#) on page 56 or [Implementing SENT Receivers Using RTLib Functions](#) on page 63.

I/O mapping The following table shows the mapping of the channel numbers to the corresponding I/O pins of I/O connector P1, as used in RTLib. Some I/O features of the DS2211 conflict with each other. Refer to [Conflicting I/O Features](#) on page 197.

Channel Number	Signal	Connector Pin		Sub-D Pin		Description
1	DIG_IN1	P1	3	P1B	34	Digital input
2	DIG_IN2	P1	4	P1A	34	Digital input
3	DIG_IN3	P1	5	P1B	18	Digital input
4	DIG_IN4	P1	6	P1A	18	Digital input

Related topics

Basics

[Basics of the SENT Protocol](#)..... 48

Implementing SENT Transmitters in Simulink

Introduction

RTI provides two blocks for implementing a SENT transmitter in a Simulink model. The DS2211SENT_TX_BLx block writes the messages to the transmit FIFO so that they can be sent from the DS2211. The DS2211SENT_TXFIFO_BL block can be used to read the fill level of the transmit FIFO to avoid writing too many messages to the transmit FIFO.

Configuring the channels

To configure channels for sending SENT messages defined by the SENT specification, you must perform some preparatory steps.

- The channels must be connected to the SENT receivers, see [Using the SENT Protocol on a DS2211](#) on page 51.
- Drag the DS2211DIO_SETUP_Bx block into your Simulink model and specify the parameters of the digital output channels used on the DIG_OUT page.
 - Select the low-side switches of the channels.
 - If you use an additional 5 V-power supply, clear both high-side switches of the channels. Otherwise select the high-side switch of the power supply used (VBat1 or VBat2).

Tip

The DS2211DIO_SETUP_Bx block is not necessary if the default settings of the digital output channels are sufficient. The default settings are:

- Low-side switch is enabled.
- High-side switch of VBAT1 is enabled.
- High-side switch of VBAT2 is disabled.

- For each channel which is used as a SENT transmitter, drag a DS2211SENT_TX_BLx block into your Simulink model and specify the board and channel on the Unit page.

Specifying the SENT parameters

The parameters for SENT are specified on the TX Parameters and Advanced pages of the DS2211SENT_TX_BLx block. Some parameters of the block are run-time-tunable. The values of run-time-tunable parameters can be modified during run time via the experiment software. The changed values take effect immediately. This makes it possible to simulate a SENT transmitter that does not comply with the specification.

Tick period The length of the tick period is specified on the Advanced page. It can be specified by a block parameter, or an inport can be enabled so that you can specify it by a Simulink signal. The parameter is run-time-tunable.

Low pulse A low pulse is defined by a number of tick periods. You can specify the number on the TX Parameters page. The parameter is run-time-tunable.

Zero nibble high pulse A zero nibble high pulse is defined by a number of tick periods. You can specify the number on the TX Parameters page. The parameter is run-time-tunable.

Sync high pulse A sync high pulse is defined by a number of tick periods. You can specify the number on the TX Parameters page. The parameter is run-time-tunable.

Autorepeat mode If the autorepeat mode is set, the message written last is repeated when the transmit FIFO runs empty. This message is repeated until a new message is written to the transmit FIFO. The autorepeat mode is set on the TX Parameters page.

Pause pulse mode If the pause pulse is enabled, the transmitter appends a pause pulse at the end of every message. The pause pulse length is defined for each message at the Pause pulse input of the DS2211SENT_TX_BLx block. The pause pulse mode is set on the TX Parameters page.

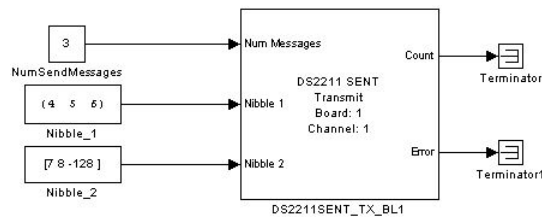
It depends on the board/FPGA revision whether the pause pulse mode is supported, refer to [Using the SENT Protocol on a DS2211](#) on page 51.

Sending messages and nibbles

Messages The maximal number of messages must be specified on the TX Parameters page. The number of messages to be written to the transmit FIFO is specified by a Simulink signal at the Num Messages input. If no message must be written to the transmit FIFO, you can write a 0 to the input.

Nibbles The number of nibbles must be specified on the TX Parameters page. The number must include the status nibble and CRC nibble as they are sent like "normal" nibbles. The DS2211SENT_TX_BLx block gets inputs for each nibble. The signal for the Nibble inputs must be a vector with the length of the specified maximal number of messages. The block reads the values of the Nibble inputs one after the other and forms the messages to be sent. If a nibble must not be transmitted, you must specify a value of '-128' so that the SENT transmitter ignores the nibble.

Example The following illustration shows an example of a SENT transmitter.



In the example the messages have two nibbles. Three messages are written to the transmit 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. If the NumSendMessages Constant block has the value 2, only the first two messages are written to the transmit FIFO.

Getting the fill level of transmit FIFO

To prevent the transmit FIFO overrunning, you can check its fill level before writing new messages to it. The DS2211SENT_TXFIFO block gives the number of messages which remain in the transmit FIFO.

The maximum entries in the transmit FIFO depend on the buffer size, the number of nibbles used, and the pause pulse mode. If the pause pulse mode is disabled, it is

$$N_{MessagesMax} = \text{RoundDown}(63 / \text{RoundUp}(N_{Nibble} / 7))$$

If the pause pulse mode is enabled, it is

$$N_{MessagesMax} = \text{RoundDown}(63 / (\text{RoundUp}(N_{Nibble} / 7) + 1))$$

Where

$N_{MessagesMax}$ is the number of messages which can maximally be buffered
 N_{Nibble} is the number of nibbles in each message

If the fill level plus the number of new messages written to the transmit FIFO exceeds the maximum entries, an overrun occurs. To avoid an overrun, you can reduce the number of new messages at the Num Messages input of the DS2211SENT_TX_BLx block.

Getting the send status

The DS2211SENT_TX_BLx block has two outputs which give you status information. Both outputs are optional and can be enabled on the **Advanced** page.

Count The Count output provides the number of messages which are successfully written to the transmit FIFO.

Error The Error output indicates whether the write operation was successful (value = 0) or not (value = 1).

Related topics

References

[DS2211DIO_SETUP_Bx \(DS2211 RTI Reference !\[\]\(d8ab143e904bfa3467271eec5af75a9b_img.jpg\)](#))
[DS2211SENT_TX_BLx \(DS2211 RTI Reference !\[\]\(567bfd12bbf6d2452d8ec3264a002612_img.jpg\)](#))

Implementing SENT Receivers in Simulink

Introduction

RTI provides one block for implementing a SENT receiver in a Simulink model. The DS2211SENT_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 a DS2211](#) on page 51.
- Drag the DS2211DIO_SETUP_Bx block into your Simulink model and specify the parameters of the digital input channels on the DIG_IN page.
 - Specify 2.5 V as threshold level for the channels.

Tip

The DS2211DIO_SETUP_Bx block is not necessary if the default settings of the digital input channels are sufficient. The default threshold level is 2.5 V.

- For each channel which is used as a SENT receiver, drag a DS2211SENT_RX_BLx block into your Simulink model and specify the board and channel on the Unit page.

Specifying the SENT parameters

The parameters for SENT are specified on the RX Parameters page of the DS2211SENT_RX_BLx block. Some parameters are stop-run-tuneable. The values of stop-run-tuneable parameters can be modified during run time via the experiment software. The changed values take effect when the simulation state of the model changes from STOP to RUN or PAUSE.

Low pulse A low pulse is defined by a number of tick periods. You can specify the number on the RX Parameters page. The parameter is stop-run-tunable.

Zero nibble high pulse A zero nibble high pulse is defined by a number of tick periods. You can specify the number on the RX Parameters page. The parameter is stop-run-tunable.

Sync high pulse A sync high pulse is defined by a number of tick periods. You can specify the number on the RX Parameters page. The parameter is stop-run-tunable.

Tick period The expected length of the tick period and the tick period tolerance are specified on the RX Parameters page. As the DS2211 board measures the actual tick period when a message is received, the block can set an error flag if the measured length of the tick period is outside the specified tolerance. The parameters of the expected tick period and the tick period tolerance are stop-run-tunable.

Receiving messages and nibbles

Messages The expected number of messages which are received must be specified on the RX Parameters page. The messages are written to the receive FIFO. If more messages are received than specified, the superfluous messages are discarded and an error flag is set. The number of messages which were currently received is given by the Count output.

Nibbles The number of nibbles must be specified on the RX Parameters page. The number must include the status nibble and CRC nibble as they are treated as "normal" nibbles. The DS2211SENT_RX_BLx block gets outputs for each nibble. The Nibble n output 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

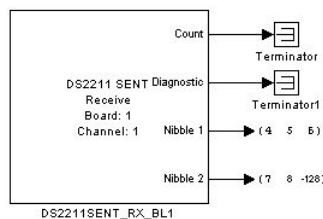
number of messages which are actually received (and given by the Count output).

Read mode The read mode is set on the RX Parameters page. The block can read all new messages which were received since the last read operation or the newest complete message.

Pause pulse mode You can enable or disable the pause pulse mode on the RX Parameters page. If the pause pulse mode is enabled, the receiver accepts a pause pulse at the end of every message, between last nibble and synchronization pulse of the next message. Otherwise, pause pulses cannot be handled by the receiver.

If you enable the pause pulse mode, you can specify an expected message length. If this value is unequal to 0, a constant message length is expected. If the length of the received message (all low and high ticks of sync pulse, nibble pulses and pause pulse) differs from the specified expected message length, a flag is set at the Diagnostic output. If the Expected message length is 0, a pause pulse is expected but no constant message length.

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.

Getting diagnostic information

The DS2211SENT_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 measured 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. The meanings of the flags are as follows:

- Bit 0 (value 1): Too many nibbles in message
- Bit 1 (value 2): Too few nibbles in message
- Bit 2 (value 4): Nibble value is out of range [0..15]
- Bit 3 (value 8): Synchronization pulse too long
- Bit 4 (value 16): Synchronization pulse too short

- Bit 5 (value 32): Current synchronization pulse differs from the previous synchronization pulse by more than a factor of 1/64
- Bit 6 (value 64): The received message length in ticks differs from the expected one
- Bit 7 (value 128): The received ratio of synchronization pulse to message length differs by more than 1/64 from the nominal ratio

For details on the diagnostic flags, refer to [Block Description \(DS2211SENT_RX_BLx\)](#) (DS2211 RTI Reference [\[9\]](#)).

Error The Error output provides information on the current read operation. The values have the following meanings:

- 0: No error occurred during this read operation.
- Bit 0 (value 1): The number of received messages exceeded the expected number of messages set by the Count parameter. At least one message was discarded to prevent an overrun of the allocated message buffer.
- Bit 1 (value 2): A receiver timeout occurred. This means that at least one pulse length exceeded the maximum measurable pulse length. The receiver discards the message with the timeout and searches for the next synchronization pulse. The timeout information is active for one read operation only.

Avoiding data loss

It is recommended to call the DS2211SENT_RX_BLx block regularly to read out the receive FIFO because it can store only 128 pulses with high and low parts. When it runs full, all new received pulses get lost until the next call of the block. Thus nibbles or messages are lost.

Maximal number of messages The number of messages that can be maximally buffered between two read operation can be calculated as follows if the pause pulse mode is disabled:

$$N_{MessagesMax} = \text{RoundDown}[128 / (N_{Nibble} + 1)]$$

If the pause pulse mode is enabled, it is

$$N_{MessagesMax} = \text{RoundDown}[128 / (N_{Nibble} + 2)]$$

Where

$N_{MessagesMax}$ is the number of messages which can maximally be buffered

N_{Nibble} is the number of nibbles in each message

Maximal cycle time If data gets lost, the time between two read operations may be too long. The maximum time between two necessary read operations can be calculated as follows if the pause pulse mode is disabled:

$$t_{MaxCycle} = \text{RoundDown}[128 / (N_{Nibble} + 1)] \cdot [N_{Sync} + N_{Zero} \cdot N_{Nibble} + N_{Low} \cdot (N_{Nibble} + 1)] \cdot T_{Tick} \cdot (1 - \Delta T_{TickPeriod})$$

If the pause pulse mode is enabled, it is

$$t_{MaxCycle} = \text{RoundDown}[128 / (N_{Nibble} + 2)] \cdot [N_{Sync} + N_{Zero} \cdot (N_{Nibble} + 1) + N_{Low} \cdot (N_{Nibble} + 2)] \cdot T_{Tick} \cdot (1 - \Delta T_{TickPeriod})$$

Where

$t_{MaxCycle}$ is the maximum time between two necessary read operations (maximal cycle time)

N_{Nibble} is the number of nibbles in each message

N_{Sync} is the number of ticks of a synchronization pulse

N_{Zero} is the number of ticks of a zero nibble high pulse

N_{Low} is the number of ticks of a low pulse


T_{Tick} is the tick period

$\Delta T_{TickPeriod}$ is the tolerance of tick period (clock drift)

Related topics

References

[DS2211DIO_SETUP_Bx](#) (DS2211 RTI Reference )

[DS2211SENT_RX_BLx](#) (DS2211 RTI Reference )

Implementing SENT Transmitters Using RTLib Functions

Channel configuration

To configure channels for sending SENT messages defined by the regular SENT specification, you must perform some preparatory steps:

- The channels must be connected to the SENT receivers, see [Using the SENT Protocol on a DS2211](#) on page 51.
- Enable the low-side switches of the channels using the `ds2211_digout_ls_set` function.
- If you use an additional 5-V power supply, disable both high-side switches of the channels using the `ds2211_digout_hs_vbat1_clear` and `ds2211_digout_hs_vbat2_clear` functions.
Otherwise enable the high-side switch of the power supply used (VBat1 or VBat2) using the `ds2211_digout_hs_vbat1_set` or `ds2211_digout_hs_vbat2_set` functions.
- Enable the digital outputs for a SENT transmitter using the `ds2211_digout_mode_set` function.

Configuration of a SENT transmitter

Every SENT transmitter must be configured with the following parameters before transmitting messages:

Parameter	Type	Description
nibble_count	Initialization	Defines the number of nibbles per SENT message. This number is constant and is set during initialization of the SENT transmitter in the range 1 ... 210 or 217 (depending on the pause pulse mode).

Parameter	Type	Description
pause_mode	Initialization	Enables or disables the pause pulse generation at the end of every message.
low_tics	Run time	Defines the length of a low pulse in tick periods within the range 1 ... 15. The standard value is 5 tick periods.
sync_high_tics	Run time	Defines the length of the high pulse of a synchronization pulse in tick periods in the range 1 ... 255. The standard value is 51 tick periods.
zero_nib_high_tics	Run time	Defines the high pulse length of a nibble and a pause pulse with value 0 in tick periods in the range 1 ... 15. The standard value is 7 tick periods.
autorepeat	Run time	Defines whether the last SENT message is repeated if no new message is available. If it is deactivated, nothing is transmitted when the FIFO runs empty and the output stays logic high (this is forbidden by the SENT specification).
tic_period	Run time	Defines the length of a tick period of a SENT pulse in seconds. This is the base clock every SENT pulse is generated with. Range: 500 ns ... 51.1875 μ s / 204.7875 μ s (depending on the board/FPGA reversion), resolution 12.5 ns. The standard value is 3 μ s.

Continuous transmission of messages

The number of available FIFO entries depends on the number of nibbles per message and the pause pulse mode which are defined during initialization. The FIFO size can be calculated as follows if the pause pulse mode is disabled:

$$\text{Max_FIFO_Entries} = \text{RoundDown} (63 / \text{RoundUp}(\text{nibble_count} / 7))$$

If the pause pulse mode is enabled:

$$\text{Max_FIFO_Entries} = \text{RoundDown}(63/(\text{RoundUp}(\text{nibble_count}/7) + 1))$$

The following table gives some examples.

Nibbles per Message	FIFO Depth (No Pause)	FIFO Depth (With Pause)
1 - 7	63	31
8 - 14	31	21
15 - 21	21	15
22 - 28	15	12
...

The SENT specification requires continuous transmission of SENT messages without any interrupt of transmission. So you have to ensure that the run-time transmit function is called regularly, or activate the autorepeat feature.

Information about the current FIFO fill level is provided by the `ds2211_sent_tx_fifo_state` function. When the FIFO runs empty without the autorepeat feature enabled, the SENT transmitter output stays high until the next message is written.

Format of messages

Messages are written as a vector of

```
Int8 tx_data[100];
```

The transmit function reads nibbles from 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 and `num_msg` is the number of messages):

<code>tx_data[0]</code>	Message 1, nibble 1
<code>tx_data[1]</code>	Message 1, nibble 2
...	...
<code>tx_data[nibble_count - 1]</code>	Message 1, nibble <i>nibble_count</i>
<code>tx_data[nibble_count]</code>	Message 2, nibble 1
<code>tx_data[2*nibble_count - 1]</code>	Message 2, nibble <i>nibble_count</i>
...	...
<code>tx_data[(msg-1) * nibble_count + (nib-1)]</code>	Message <i>msg</i> , nibble <i>nib</i>

Format of pause pulse values

If pause pulse mode is enabled, the transmit function reads pause pulse values from the pause data buffer one after the other. One pause pulse value per message must be included in the buffer. The expected format of the data array is as follows:

```
Int16 tx_pause[10];
```

<code>tx_pause[0]</code>	Pause of message 1
<code>tx_pause[1]</code>	Pause of message 2
...	...
<code>tx_pause[m]</code>	Pause of message m-1

Missing nibbles in a message

To simulate missing nibbles in a SENT message, nibbles can be marked as missing by setting their value to `DS2211_SENT_MISSING_NIBBLE` (-128). The SENT transmitter will ignore these nibbles. It is not possible to ignore all the nibbles in a message. If all nibbles of a message are marked as missing, the entire message including the sync pulse, all nibble pulses and the pause pulse is ignored by the transmitter.

Missing pause pulses in a message

To simulate missing pause pulses in a SENT message, pause values can be marked as missing by setting their value to `DS2211_SENT_MISSING_PAUSE` (-32768). The SENT transmitter will ignore these pause pulses.

Calculation of the pause pulse length

The pause pulse is handled like any other nibble in a message. Its low duration is defined by the `low_tics` parameter, its high duration for a pause value of 0 is defined by `zero_nib_high_tics`. In addition, negative pause durations are supported to reduce the high duration of the pause pulse to generate high pulses shorter than `zero_nib_high_tics`. The pulse length calculates as follows:

$$\text{HIGH_LENGTH} = (\text{ZERO_NIBBLE_HIGH_TICS} + \text{PAUSE_VALUE}) \cdot \text{TIC_PERIOD}$$

$$\text{LOW_LENGTH} = \text{LOW_TICS} \cdot \text{TIC_PERIOD}$$

If a `HIGH_LENGTH` of 0 or lower is calculated, the entire pause pulse including the low pulse is missing. It behaves like using a pause value of `DS2211_SENT_MISSING_PAUSE` (-32768).

Range of pulse length

A DS2211 can only generate low pulses and high pulses greater than 2 µs. You must consider this restriction for your SENT transmitter, otherwise the pulse cannot be generated properly.

RTLib functions

For information on the RTLib functions used for programming a SENT transmitter, refer to [Configuring a SENT Transmitter \(DS2211 RTLib Reference !\[\]\(3cb60d42b10e53f9522bb0b392c1c4cd_img.jpg\)](#)).

Related topics**Basics**

[Using the SENT Protocol on a DS2211..... 51](#)

References

[ds2211_digout_hs_vbat1_clear \(DS2211 RTLib Reference !\[\]\(274fd520e03b61c1b9ffc861754cacdc_img.jpg\)](#))
[ds2211_digout_hs_vbat1_set \(DS2211 RTLib Reference !\[\]\(cd41623988aee99b5b915fa19a633d9e_img.jpg\)](#))
[ds2211_digout_hs_vbat2_clear \(DS2211 RTLib Reference !\[\]\(cd7d9647d36803fafef189ffeac2b3c1_img.jpg\)](#))
[ds2211_digout_hs_vbat2_set \(DS2211 RTLib Reference !\[\]\(9643629c51032ad6267a10569acb0497_img.jpg\)](#))
[ds2211_digout_ls_set \(DS2211 RTLib Reference !\[\]\(cabdd29f768f022926ec4fc4a2a655ee_img.jpg\)](#))
[ds2211_digout_mode_set \(DS2211 RTLib Reference !\[\]\(9147d99247e2046e1a8594ac57bead29_img.jpg\)](#))
[ds2211_sent_tx_fifo_state \(DS2211 RTLib Reference !\[\]\(1ddb1cb5d2399dd91e9a94348f7937c2_img.jpg\)](#))

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 a DS2211](#) on page 51.

- Set the input threshold to 2.5 V using the `ds2211_digin_threshold_set` function.

Configuration of a SENT receiver

Every SENT receiver must be configured with the following parameters before receiving messages:

Parameter	Type	Description
nibble_count	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 .. 217.
timing_range	Initialization	Specifies a timing range for pulse length measurement. You can use this to adapt the range of measurable pulse length to the transmitter clock and pulse length.
pause_mode	Initialization	Sets the pause pulse mode of the receiver. You can enable and disable the pause pulse mode. If the mode is enabled, the receiver accepts a pause pulse at the end of every message.
fixed_mes_len_tics	Initialization	Defines an expected message length in ticks for additional diagnostic evaluation. It is only recommended if a pause pulse is used to generate a fixed message length. A value of 0 disables the additional diagnostic evaluation.
low_tics	Initialization	Defines the length of a low pulse in ticks in the range 1 .. 15. The standard value is 5 tick periods.
sync_high_tics	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.
zero_nibble_high_tics	Initialization	Defines the high pulse length of a nibble and a pause pulse with value 0 in ticks in the range 1 .. 15. The standard value is 7 tick periods.
tic_period	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 500 ns .. 200 µs. The standard value is 3 µs.
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

Parameter	Type	Description
		this specified range are recognized as invalid synchronization pulses or nibble pulses with an invalid value (< 0; >15). When an invalid synchronization pulse is received, the current message is cut and the receiver searches 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 (±20%).

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 128 pulses each with a high part and a low part.

The maximum time between two necessary read operations can be calculated as follows if the pause pulse mode is disabled:

$$\begin{aligned} \text{Max_Cycle_Time} &= \text{RoundDown}[128 / (\text{Nibble_Count} + 1)] \cdot \\ &\quad [\text{Sync_High_Tics} + \\ &\quad \text{Zero_Nibble_High_Tics} \cdot \text{Nibble_Count} + \\ &\quad \text{Low_Tics} \cdot (\text{Nibble_Count} + 1)] \cdot \\ &\quad \text{tic_period} \cdot (1 - \text{clockdrift}) \end{aligned}$$

If the pause pulse mode is enabled, it is

$$\begin{aligned} \text{Max_Cycle_Time} &= \text{RoundDown}[128 / (\text{Nibble_Count} + 2)] \cdot \\ &\quad [\text{Sync_High_Tics} + \\ &\quad \text{Zero_Nibble_High_Tics} \cdot (\text{Nibble_Count} + 1) + \\ &\quad \text{Low_Tics} \cdot (\text{Nibble_Count} + 2)] \cdot \\ &\quad \text{tic_period} \cdot (1 - \text{clockdrift}) \end{aligned}$$

Max_Cycle_Time is the maximum time between two read operations without loss of data.

Example (pause pulse mode disabled):

Tick Period	Sync High Ticks	Zero Nibble High Ticks	Low Ticks	Nibble Count	Clock Drift	Max Cycle Time
3 µs	51	7	5	6	20%	5.53 ms
3 µs	51	7	5	10	20%	4.65 ms
3 µs	51	7	5	6	30%	4.84 ms
1 µs	51	7	5	6	20%	1.84 ms

If one message consists of more than 128 pulses, more than one receive operation is required for receiving it. Because only complete messages are

delivered to the model, not every read operation returns a message. Nevertheless it is recommended to execute the read function regularly to read out the internal data buffer, which can store a maximum of 128 pulses with high part and low part.

Diagnostic of messages

The SENT receiver generates diagnostic information for every received SENT message. The following diagnostic information is evaluated and reported via flags in a diagnostic word. The diagnostic information is saved to a vector of UInt32. The number of diagnostic words always matches the number of returned messages.

Bit	Information
0	Too many nibbles in message. If too many nibbles are received in a message, the superfluous nibbles are ignored and this diagnostic flag is set. This ensures that every message is saved to the data buffer with the defined number of nibbles (<i>nibble_count</i>).
1	Too few nibbles in message. If a message with too few nibbles is received, the missing nibbles are marked with the value of <code>DS2211_SENT_MISSING_NIBBLE</code> (-128) and the diagnostic flag reports missing nibbles. This ensures that every message is saved to the data buffer with the defined number of nibbles (<i>nibble_count</i>).
2	Nibble value is out of range 0 .. 15. If a nibble with a value <0 or >15 is received, this nibble is nevertheless saved to the data buffer and the diagnostic flag for a nibble outside the valid range is set.
3	Synchronization pulse too long (outside specified allowed clock drift). If a synchronization pulse exceeds the valid range defined by clockdrift, the diagnostic flag is set. Nevertheless the nibble values are evaluated.
4	Synchronization pulse too short (outside specified allowed clock drift). If a synchronization pulse exceeds the valid range defined by clockdrift, the diagnostic flag is set. Nevertheless the nibble values are evaluated.
5	Current synchronization pulse differs from the last synchronization pulse by more than a factor of 1/64. If two consecutive synchronization pulses differ by more than 1/64 of the current synchronization pulse, this diagnostic flag is set.
6	Fixed message length difference. The received message length in ticks differs from the expected one.
7	Sync to message ratio difference. The received ratio of sync pulse to message length differs by more than 1/64 from the nominal ratio.

Format of messages

Messages are delivered to the model as a vector of Int8 `rx_data[100]`.

```
Int8 rx_data[100];
```

The receive function 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[nibble_count - 1]</code>	Message 1, nibble <i>nibble_count</i>
<code>rx_data[nibble_count]</code>	Message 2, nibble 1
...	...
<code>rx_data[2 · nibble_count - 1]</code>	Message 2, nibble <i>nibble_count</i>
...	...
<code>rx_data[(msg - 1) · nibble_count + (nib - 1)]</code>	Message <i>msg</i> , nibble <i>nib</i>

The format for a 2-dimensional vector looks like this:

```
Int8 rx_data[num_msg][nibble_count];
```

The vector looks like this (*nibble_count* 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][nibble_count - 1]</code>	Message 1, nibble <i>nibble_count</i>
<code>rx_data[1][0]</code>	Message 2, nibble 1
...	...
<code>rx_data[1][nibble_count - 1]</code>	Message 2, nibble <i>nibble_count</i>
...	...
<code>rx_data[msg - 1][nib - 1]</code>	Message <i>msg</i> , nibble <i>nib</i>

Format of pause pulse values

If pause pulse mode is enabled, the receive function writes pause pulse values to the data buffer one after the other. One pause pulse value per message is stored to the buffer. The expected format of the data array is as follows:

```
Int16 rx_pause[10];
```

<code>rx_pause[0]</code>	Pause of message 1
<code>rx_pause[1]</code>	Pause of message 2
...	...
<code>rx_pause[m]</code>	Pause of message m-1

Calculation of the pause pulse values

The pause pulse is handled like any other nibble in a message. Its low duration is defined by the `low_ticks` parameter. Its high duration for a pause value of 0 is defined by `zero_nib_high_ticks`. The pause pulse value is calculated as follows:

$$\text{Pause_Value} = \text{Int16}(\text{Pulse_Length} / \text{Current_Tic_Period}) - (\text{Low_Tics} + \text{Zero_Nib_High_Tics})$$

Where:

Pulse_Length Is the measured pulse length between two consecutive falling edges for this pulse

Current_Tic_Period Is the current transmitter tick period extracted from the synchronization pulse

Modes of receiving messages

The SENT receivers support two different modes of reading messages:

- **Read All Mode:** The `ds2211_sent_rx_receive_all_pause` function reads all new messages received since the last read operation. When no complete new message is available, nothing is returned and the `len` parameter is 0. To avoid writing more messages to the user data buffer than memory was allocated, the `count` 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 `count`, 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 setting the bit 0 of the return value (`DS2211_SENT_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 65).
- **Most Recent Mode:** The `ds2211_sent_rx_receive_most_recent_pause` function reads the newest complete message only. If no message was received at all, a message with all nibbles marked as missing nibbles `DS2211_SENT_MISSING_NIBBLE` (-128) is returned and the `msg_count` parameter is 0.

Range of pulse length

A DS2211 can only measure low pulses and high pulses in a specific range. The lower and the upper limit depend on the selected timing range. Shorter pulses can lead to loss of data or erroneous measured pulses. Pulses longer than the upper limit lead to a receiver timeout.

Timeout detection

When a pulse length exceeds the measurable pulse length which depends on the selected timing range, a timeout is reported by the read function. Then the entire message is discarded and the receiver will wait for the next sync pulse.

Selection of a timing range

To adapt the receiver to the timings of the transmitter, a timing range must be selected. Choosing higher timing ranges leads to a reduced resolution, but allow measuring longer pulse length. The following timing ranges are selectable:

Predefined Symbol	Timing Range	Resolution	Recommended Minimum Tick Period	Minimum Measurable Pulse Length	Maximum Measurable Pulse Length
DS2211_SENT_TIMING_RANGE1	1	50 ns	300 ns	2 μ s	819 μ s
DS2211_SENT_TIMING_RANGE2	2	100 ns	600 ns	2 μ s	1.64 ms
DS2211_SENT_TIMING_RANGE3	3	200 ns	1.2 μ s	2 μ s	3.28 ms
DS2211_SENT_TIMING_RANGE4	4	400 ns	2.4 μ s	2.4 μ s	6.55 ms
DS2211_SENT_TIMING_RANGE5	5	800 ns	4.8 μ s	4.8 μ s	13.1 ms
DS2211_SENT_TIMING_RANGE6	6	1.6 μ s	9.6 μ s	9.6 μ s	26.2 ms
DS2211_SENT_TIMING_RANGE7	7	3.2 μ s	19.2 μ s	19.2 μ s	52.4 ms
DS2211_SENT_TIMING_RANGE8	8	6.4 μ s	38.4 μ s	38.4 μ s	105 ms
DS2211_SENT_TIMING_RANGE9	9	12.8 μ s	76.8 μ s	76.8 μ s	210 ms

RTLib functions

For information on the RTLib functions used for programming a SENT receiver, refer to [Configuring a SENT Receiver \(DS2211 RTLib Reference !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)](#)).

Related topics

Basics

[Using the SENT Protocol on a DS2211.....51](#)

References

[ds2211_digin_threshold_set \(DS2211 RTLib Reference !\[\]\(0b5e7e25e8775f7e7e80906ada4f0021_img.jpg\)](#))
[ds2211_sent_rx_receive_all_pause \(DS2211 RTLib Reference !\[\]\(740312fd467f47b04cab841ab3868d83_img.jpg\)](#))
[ds2211_sent_rx_receive_most_recent_pause \(DS2211 RTLib Reference !\[\]\(dbb8da2687e90ededffd3484b6b666cf_img.jpg\)](#))

Angular Processing Unit

Introduction	See the following sections for information on the angular processing unit (APU), which is designed to simulate core engine processing functions (crankshaft signal generation, for example).
--------------	--

Where to go from here

Information in this section

APU Basics.....	72
You can find basic information on how simulation of engine core functions works. The most important terms when working with RTI and RTLib are explained.	
Angular Processing Unit - Variant.....	97
If you use the VAR APU blockset, you can change the engine variant without the need to rebuild and download the real-time application.	
APU Reference.....	104
You can find the reference data required to implement the APU functions. This includes restrictions and limitations as well as the I/O mappings.	

APU Basics

Introduction

The following is aimed at users who need basic information on how simulation of engine core functions works. The most important terms when working with RTI and RTLib are explained.

Where to go from here

Information in this section

APU Overview.....	73
Presenting the functional units of the APU and their interconnections.	
Engine Position Phase Accumulator.....	75
Explaining the unit that supplies the engine position.	
Cascading I/O Boards.....	76
To get the same time base on several I/O boards, for example, to simulate an engine with more than eight cylinders.	
Crankshaft Signal Generator.....	77
Explaining how the crankshaft signal is generated and what wave tables are for.	
Reverse Crankshaft Rotation.....	78
Explaining how a reverse crankshaft signal is generated.	
Camshaft Signal Generator.....	81
Explaining how the camshaft signals are generated and what wave tables are for.	
Spark Event Capture Unit.....	83
Explaining how ignition pulses are captured.	
Injection Event Capture Unit.....	84
Explains how injection pulses are captured and how the fuel amount is evaluated.	
Event Capture Windows.....	88
Giving information on the definition of event capture windows.	
Continuous Value Capturing.....	92
Explaining how you can determine ignition position, injection pulse position and duration values continuously.	
Complex Comparators.....	93
Explains the complex comparator functionality which is needed for capturing complex signals, for example, the current through an injector.	

Information in other sections

[Features of the Slave DSP..... 115](#)

The slave DSP is used for wheel speed sensor simulation and knock sensor simulation. It provides a DAC unit, has access to the Bit I/O unit and can be triggered by interrupts.

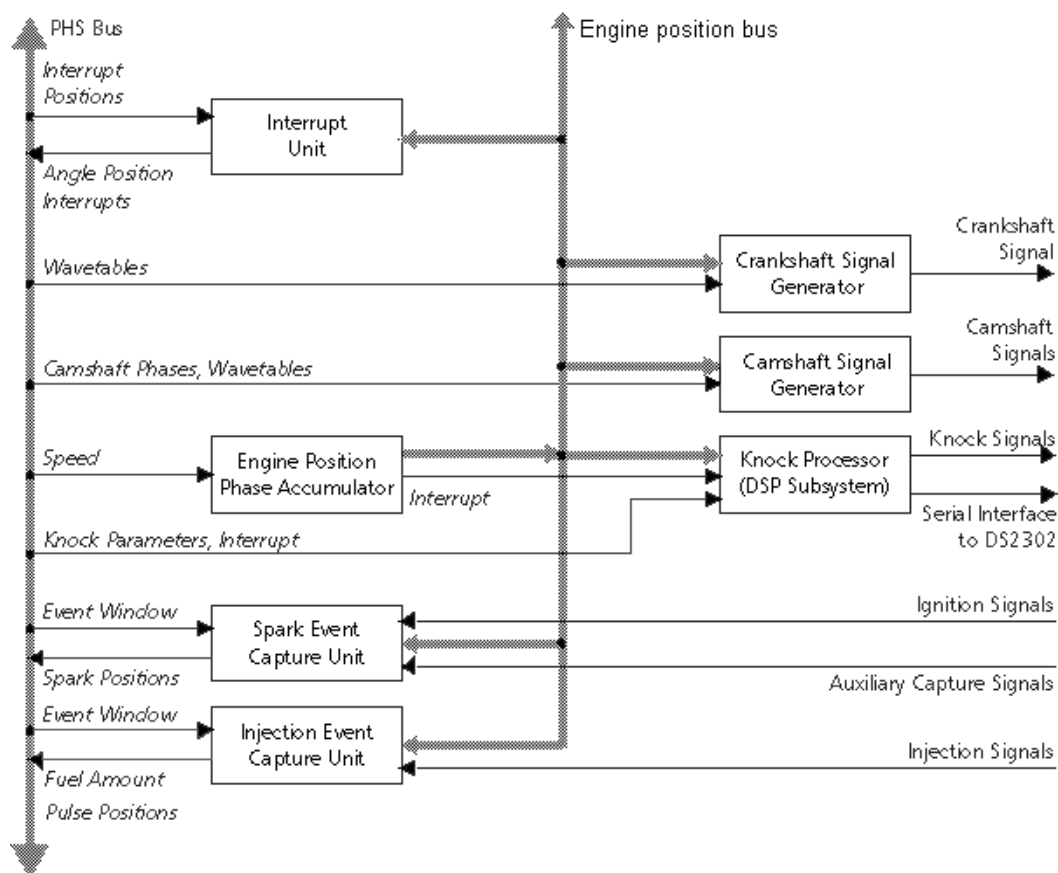
[Angular Processing Unit - Variant..... 97](#)

If you use the VAR APU blockset, you can change the engine variant without the need to rebuild and download the real-time application.

APU Overview

Introduction

The illustration shows the functional units of the angular processing unit, their interconnections, their most important input parameters and I/O signals.



Engine position bus

All APU components are interconnected by the engine position bus. The engine position phase accumulator supplies the engine position according to the Speed input parameter. Based on the engine position,

- Crankshaft, camshaft, and knock signals can be generated, and
- Capturing of spark events and injection signals can be triggered.

Using the engine position bus, you can connect the APU components of the following I/O boards (the engine position bus is equal to the time-base bus of the DS4002 and DS5001 boards):

- DS2210
- DS2211
- DS4002 (starting from board revision DS4002-04)
- DS5001 (starting from board revision DS5001-06)
- DS5203


The boards are connected via the time-base connector. Refer to [Board Overview \(PHS Bus System Hardware Reference !\[\]\(529949c2c3dadbaa4e538e8c643454bc_img.jpg\)](#)).

PHS bus

In single-processor and multiprocessor PHS-bus-based systems, the Peripheral High-Speed bus (PHS bus) connects the I/O boards – a DS2211, for example – to the processor board that executes the real-time application.

Angle position interrupt

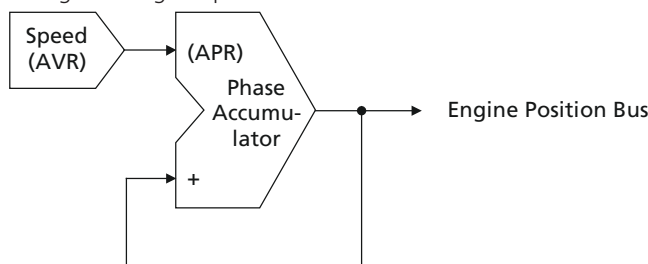
Depending on the engine position (angle), the angular processing unit can generate interrupts for up to 6 angle positions. You can use these lines to request interrupts when the cylinder has passed the top dead center (TDC), for example. Any interrupt position can be chosen, and several interrupt positions are possible. For further information on interrupts, refer to [Interrupts of the DS2211](#) on page 165.

Related topics	Basics
	APU Basics..... 72
	References
	APU Reference..... 104 DS2211 HIL I/O Board (PHS Bus System Hardware Reference )

Engine Position Phase Accumulator

Introduction

The engine position phase accumulator converts the run-time adjustable 19-bit Speed input parameter to engine position information and supplies 16-bit position information to the engine position bus. The engine cycle is 0 ... 720° (4π), corresponding to one cycle of a four-stroke engine. The resulting resolution is 0.011° or 0.0002 rad. The engine position is calculated every 250 ns, that is, changes of engine speed take effect after 250 ns at the latest.



The internal resolution for position accumulation is 32 bit. The angle velocity register (AVR) provides a 19bit signed engine speed value. This value is extended to 32 bit and added to the internal 32-bit angle position register (APR) every 250 ns. The most significant 16 bits are supplied to the engine position bus that provides the information to the other components of the APU and to the time-base connector.

The relation between engine speed in revolutions per minute (RPM) and the AVR value is defined as follows:

$$\text{RPM} = 60 \cdot 2 / (250 \text{ ns} \cdot 2^{32}) \cdot \text{AVR}$$

This results in a maximum velocity of ±29296 rpm (±3068 rad/s) with a speed resolution of 0.112 rpm (0.01175 rad/s).

The engine speed is converted into engine position values within the range 0 ... 719.989° (periodically). The values mirror the current position of the APU, which is related to the crankshaft position. These absolute position values can be converted into position values related to the TDC or another specified reference position. Refer to [DS2211APU_ANG_REL_Bx](#) ([DS2211 RTI Reference](#)).

Related topics

References

[APU Reference](#)..... 104

Cascading I/O Boards

Introduction

You can cascade a DS2211 board with other I/O boards (DS2210, DS2211, DS2302, DS4002, DS5001, DS5203). When I/O boards are cascaded their angular processing units (APUs) or timing I/O units are given the same time base for their RTI blocks or RTLib functions. This is necessary, for example, if you want to simulate an engine with more than eight cylinders, because one DS2211 supports only eight cylinders.

Functionality

All I/O boards to be synchronized must be connected to a network via the time-base connector. One of the I/O boards must be configured as the time-base master. This board supplies the time base or engine position for all other connected I/O boards. The I/O boards which read the time base must be configured as time-base slaves.

Usable I/O boards

You can connect the time-base bus (engine position bus) of the following I/O boards (the engine position bus of the DS2210 and DS2211 boards is the same as the time-base bus of the DS2302, DS4002, DS5001, and DS5203 boards):

- DS2210
- DS2211
- DS2302 (as of board revision DS2302-04)
- DS4002 (as of board revision DS4002-04)
- DS5001 (as of board revision DS5001-06)
- DS5203 (as of board revision DS5203-05)


Note

The DS2302 board cannot be used as the bus master.

Limitation

DS2211 boards can also be cascaded with DS2210 boards. Because the DS2210 has a slower APU cycle time, a DS2210 must be the time-base master to avoid overrun.

Hardware settings

To set up the network, you have to connect the I/O boards physically. For this purpose, these boards are equipped with a time-base connector. Use a standard 26-pin ribbon cable to set up the network. You can set up a network of two or more I/O boards. For the location of the time-base connector on an I/O board, refer to the corresponding board overview in the [PHS Bus System Hardware Reference](#) .

Software settings

Using RTI, you configure the time-base master/slave settings via the DS2211APU_CRANK_Bx or DS2211VARAPU_CRANK_Bx blocks. Using RLib, you configure the time-base master/slave settings via the `ds2211_mode_set` function.

You have to set one of the DS2211 boards to master mode, the others to slave mode. On the time-base slave, the engine position phase accumulator is disabled, and the engine position is supplied by the time-base master. All other functions, including initialization, are not affected by the master/slave setting and must be performed for each DS2211 board individually.

Related topics

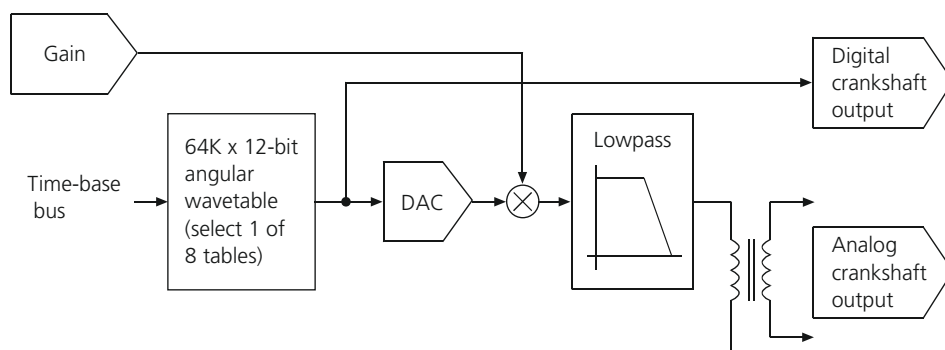
References

Board Overview (PHS Bus System Hardware Reference [📖](#))
[ds2211_mode_set](#) (DS2211 RLib Reference [📖](#))
[DS2211APU_CRANK_Bx](#) (DS2211 RTI Reference [📖](#))
[DS2211VARAPU_CRANK_Bx](#) (DS2211 RTI Reference [📖](#))

Crankshaft Signal Generator

Introduction

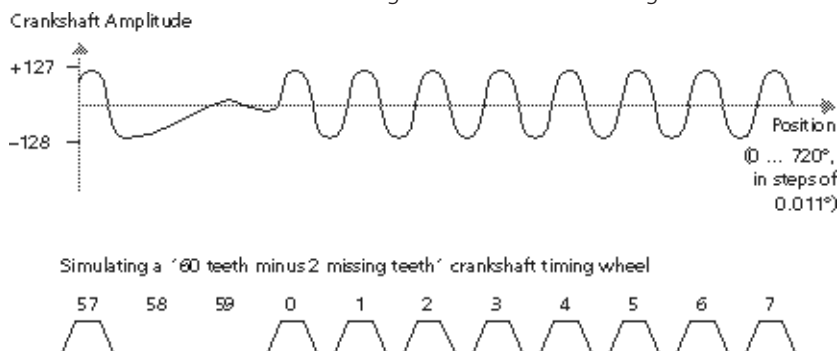
The crankshaft signal generator converts the engine position input to a crankshaft signal with analog and digital outputs. Conversion is done by using a wave table look-up mechanism. The analog output is fed through output transformers. The level of the analog output can be adapted to your application during run time using a Gain parameter. The digital output generates pulse patterns that represent the sign of the analog wave form.



The angular wave table defines a signal level for each 0.011° engine position. The resolution is 12-bit signed ($-2048 \dots +2047$). Due to the output transformers, the wave table data must be without a mean value. The data is used to define the wave form. For more details on wave tables, refer to [Wave Table Generation](#) on page 108.

If you need to simulate forward *and* reverse cranking, refer to [Reverse Crankshaft Rotation](#) on page 78.

The illustration shows the analog crankshaft output of a typical wave form that simulates a "60 teeth minus 2 missing teeth" crankshaft timing wheel.



For reference information, including the I/O mapping, refer to [Crankshaft Sensor Signal Generation](#) on page 104.

Related topics

References

[APU Reference](#)..... 104

Reverse Crankshaft Rotation

Introduction

If you need to simulate a crankshaft sensor which detects the rotation direction, for example, when simulating the start-stop mechanism of a motor vehicle, you can do this by simulating a reverse crank sensor signal.

Hardware requirements

One of the following DS2211 Board/FPGA revisions are required for reverse crankshaft rotation:

Board Revision	FPGA Revision
02	004 or higher
03	004 or higher
04	002 ... 009 or 12 or higher
05 or higher	001 or higher

Reverse crank sensor signal

A reverse crank sensor usually defines the rotation direction of a crankshaft by generating different pulse durations for the forward and reverse rotating directions. The pulses are triggered by the crankshaft timing wheel. To simulate a

reverse crank sensor signal, you have to specify these pulse durations and load a wave table that encodes the trigger points for the sensor signal.

Note

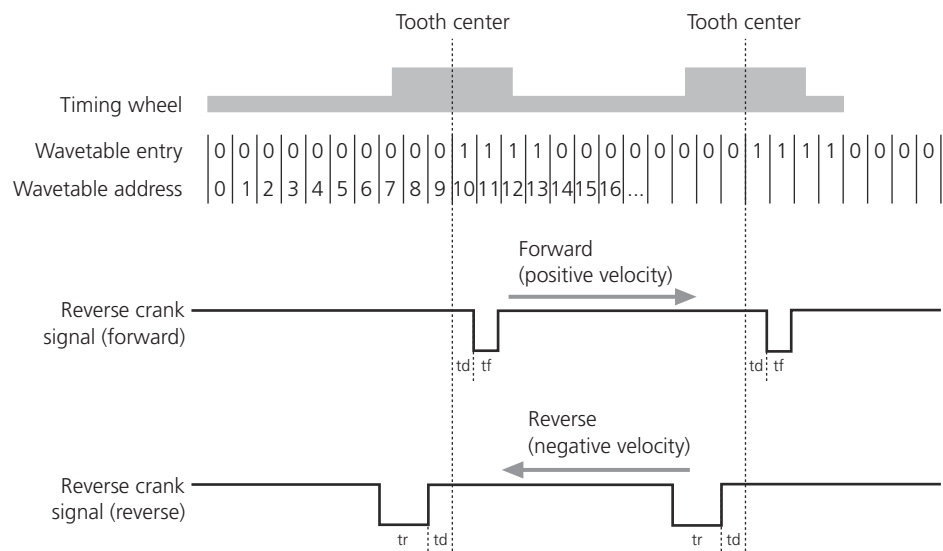
The reverse crank signal is generated as a digital signal. However, the analog crankshaft output is not automatically disabled and might generate an irrelevant analog signal.

Reverse crankshaft wave table

The wave table for reverse crank sensor simulation specifies the trigger points of the timing wheel via 0–1 transitions (forward direction) or 1–0 transitions (reverse direction). The lengths of the generated pulses are not determined by the wave table entries but specified via the Wave Tables Page (DS2211APU_CRANK_Bx) block (RTL) or the `ds2211_reverse_crank_setup` RTLlib function.

Note

To ensure that a trigger point in the wave table is detected even at a high engine speed, a minimum of 4 (if a DS2211 is the time-base master) or 16 (if a DS2210 is the time-base master) equal consecutive wave-table entries are necessary.



The illustration shows a part of a wave table for reverse crank sensor simulation. Each trigger point is represented by a 0–1 transition (forward rotation) or 1–0 transition (backward rotation). In this example, the tooth centers of the timing wheel are the trigger points, and the different pulse durations for a forward rotation (tf) and a reverse rotation (tr) are shown as low active pulses.

The wave table does not show the rotation direction, but only the trigger points. The rotation direction is specified via the sign of the Speed input variable. The

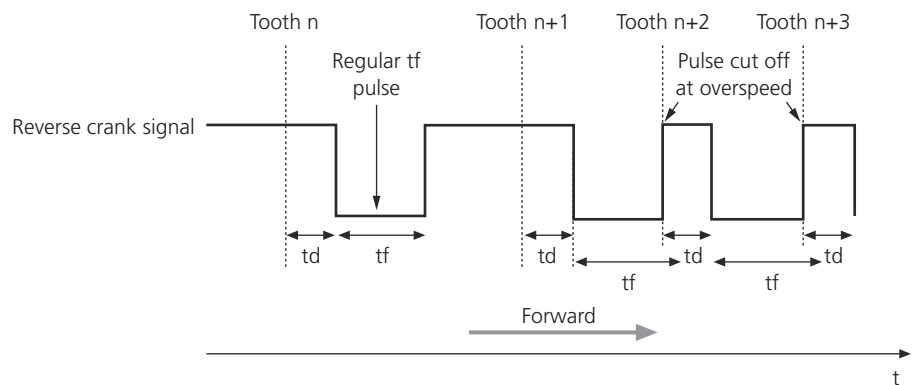
resulting engine positions then determine forward or backward stepping through the wave table addresses.

Minimum length of inactive pulses

Each generated active pulse is preceded and followed by inactive pulses of a minimum length, specified via the DS2211APU_CRANK_Bx RTI block or the ds2211_reverse_crank_setup RTLib function.

- Each trigger is followed by a short inactive pulse of the length t_d (time delay). This lets you simulate a short delay between trigger and pulse generation.
- If the generated active pulse indicates the same rotation direction as its successor, it is followed by an inactive pulse whose minimum length is t_d (time delay).
- If the rotation direction changes, the active pulse is followed by an inactive pulse whose minimum length is t_v (forced pulse duration).

Fast forward rotation

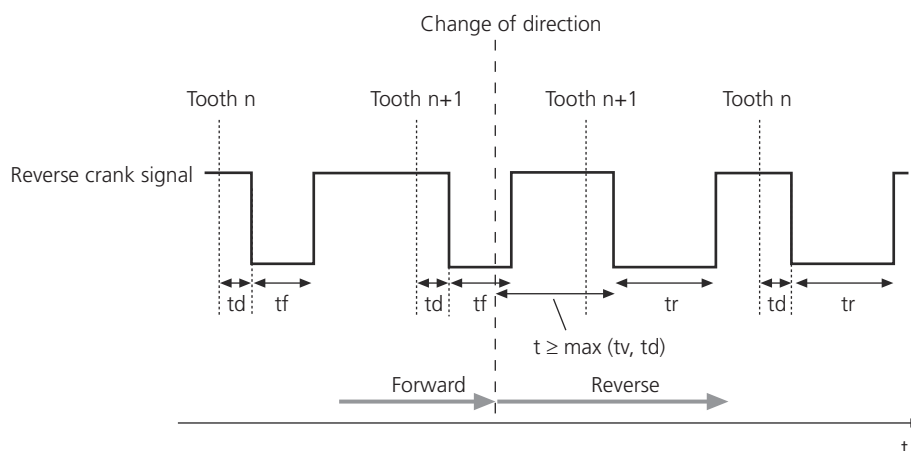


The illustration shows a low active reverse crank sensor signal. The rotation direction is always the same, and the time delay (t_d) is kept after each trigger point.

After tooth $n+1$ you can see what happens if the rotation speed becomes too fast to generate $t_d + t_f$ completely: t_f is cut by the next trigger point. That means that the time length of t_f is no longer suitable to transmit the rotation direction.

This is acceptable because at a high rotation speed, correct speed/position information is more important than the rotation direction, and a change of the rotation direction is very unlikely.

Change of direction



The illustration shows a low active reverse crank sensor signal. The rotation direction changes shortly after the trigger point of tooth n+1 has passed.

At the *change of direction* mark, you can see that the active pulse t_f is not followed by the time delay t_d , but by the forced pulse duration t_v . After t_v an active pulse t_r is generated. The pulse after t_v is always a complete one. This ensures the correct transmission of the rotation direction.

Related topics

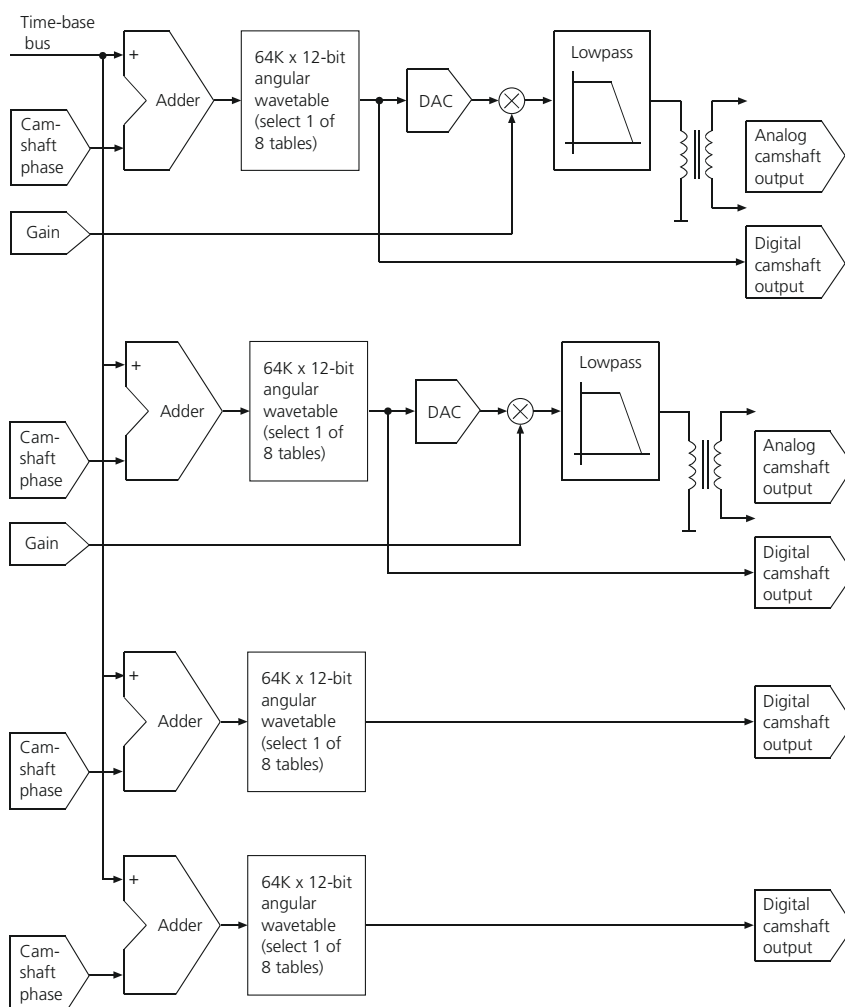
References

[ds2211_reverse_crank_setup \(DS2211 RTLib Reference !\[\]\(003082e50e3009141f59bd5df831749f_img.jpg\)\)](#)
[Wave Tables Page \(DS2211APU_CRANK_Bx\) \(DS2211 RTI Reference !\[\]\(f439ede8735757e3190eab35e168f1de_img.jpg\)\)](#)

Camshaft Signal Generator

Introduction

The camshaft signal generator converts the engine position and camshaft phase input to up to two camshaft signals, each with analog and digital outputs. Conversion is done by using a wave table look-up mechanism. The analog outputs are fed through output transformers. The level of the analog output can be adapted to your application during run time using a Gain parameter. The digital outputs generate pulse patterns that represent the sign of the corresponding analog wave forms.



The angular wave table defines a signal level for each engine position. The signal level resolution is 12-bit signed (–2048 ... +2047). Due to the output transformers, the wave table data must be without a mean value. The data is used to define the wave form. For more details on wave tables, refer to [Wave Table Generation](#) on page 108.

The phases between crankshaft signals and camshaft signals are defined by a 16-bit offset for each camshaft signal (camshaft phases). The phase offsets can be changed during run time.

For reference information, including the I/O mapping, refer to [Camshaft Sensor Signal Generation](#) on page 106.

Related topics

References

APU Reference..... 104

Spark Event Capture Unit

Introduction

The spark event capture unit is designed for ignition position measurement with one or two definable event capture windows for up to 8 cylinders. Ignition pulses can be captured from the input lines IGN1 ... IGN6, AUXCAP1 and AUXCAP2. You can choose between active high and active low pulses. The voltage level when a pulse is captured can be defined with a complex comparator (refer to [Complex Comparators](#) on page 93).

In addition, the spark event capture unit provides two independent auxiliary capture channels (AUXCAP1, AUXCAP2), which can be individually configured and also used for various measurements based on the engine position.

Note

- If you do not use the channels for ignition capture, you can use them for injection capture.
- The channels of the spark event capture unit can be read simultaneously in the event capture mode and the continuous mode by using RTLib functions. Refer to [Continuous Value Capturing](#) on page 92.

Event capture windows can be defined according to the 16-bit engine position resolution (0.011°) provided by the engine position bus (refer also to [Event Capture Windows](#) on page 88).

Interrupt

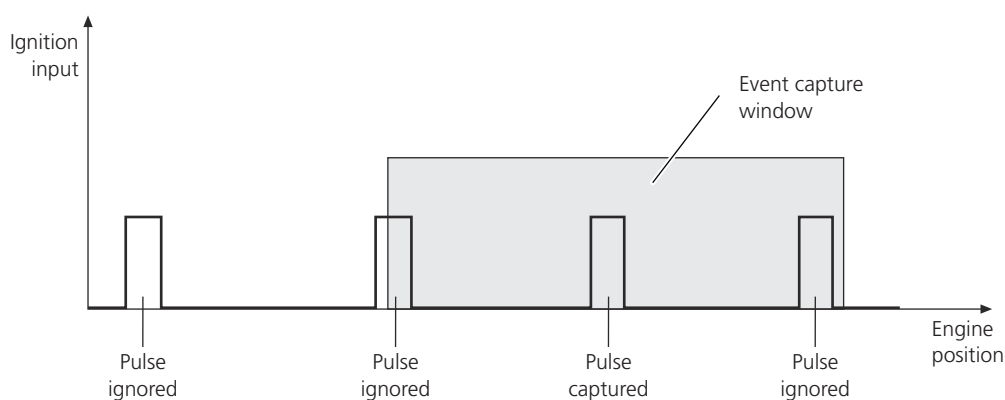
The spark event capture unit can trigger an interrupt on the slave DSP when an edge is detected. Refer to [Basics of DS2211 Interrupts](#) on page 165.

Capture modes

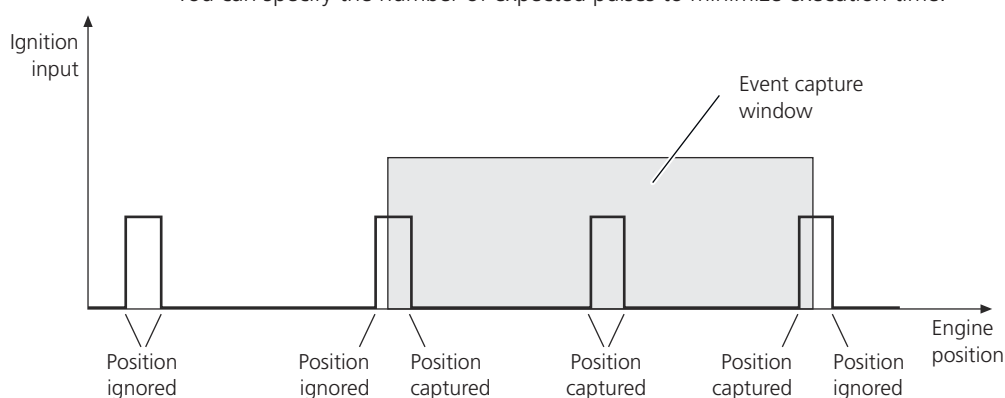
Three capture modes are available:

Continuous capture mode Continuous reading of the spark event data is available within each sample hit. Up to 64 events per channel can be read continuously out of the capture FIFO.

Single event capture mode The position of the leading edge of the first input pulse is evaluated within each event capture window. Further pulses within the same event capture window are ignored.



Multiple event capture mode The position values of all leading and trailing edges of up to 64 input pulses are evaluated within each event capture window. You can specify the number of expected pulses to minimize execution time.



For reference information, including the I/O mapping, refer to [Spark Event Capture](#) on page 109.

Related topics

References

[APU Reference](#)..... 104

Injection Event Capture Unit

Introduction

The injection event capture unit is designed for injection position and fuel amount measurements with one or two definable event capture windows for up to 16 channels. You can choose between active high and active low pulses. The voltage level when a pulse is captured can be defined by a complex comparator (refer to [Complex Comparators](#) on page 93). The resolution for fuel amount

measurement is 250 ns (the pulse duration is proportional to fuel amount). Up to 8 channels per group can be captured:

- Group 1 uses input lines INJ1 ... INJ6 on connector P2 and INJ7 (PWM_IN7) and INJ8 (PWM_IN8) on connector P1.
- Group 2 uses input lines IGN1 ... IGN6 on connector P2 and AUXCAP1 and AUXCAP2 on connector P2.

Note

- If you use the input lines of group 2 for injection capture, ignition capture is not possible.
- The channels of the injection event capture unit can be read simultaneously in the event capture mode and the continuous mode by using RTLib functions. Refer to [Continuous Value Capturing](#) on page 92.

Event capture windows can be defined according to the 16-bit engine position resolution (0.011°) provided by the engine position bus (refer also to [Event Capture Windows](#) on page 88).

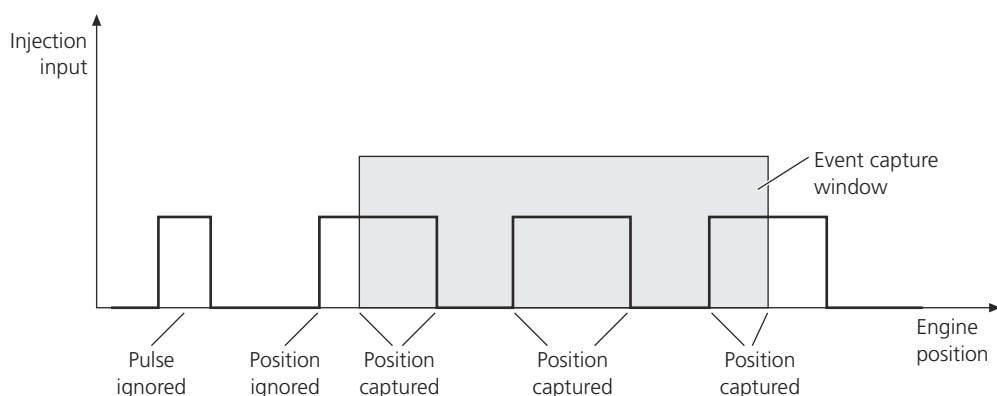
Interrupt

The injection event capture unit can trigger an interrupt on the slave DSP when an edge is detected. Refer to [Basics of DS2211 Interrupts](#) on page 165.

Capture modes

The following capture modes are available:

Position mode (start position, end position) The position values of leading and trailing edges of up to 64 input pulses are evaluated in each event capture window. The values are measured relative to the TDC. You can specify the number of expected pulses to minimize the execution time.



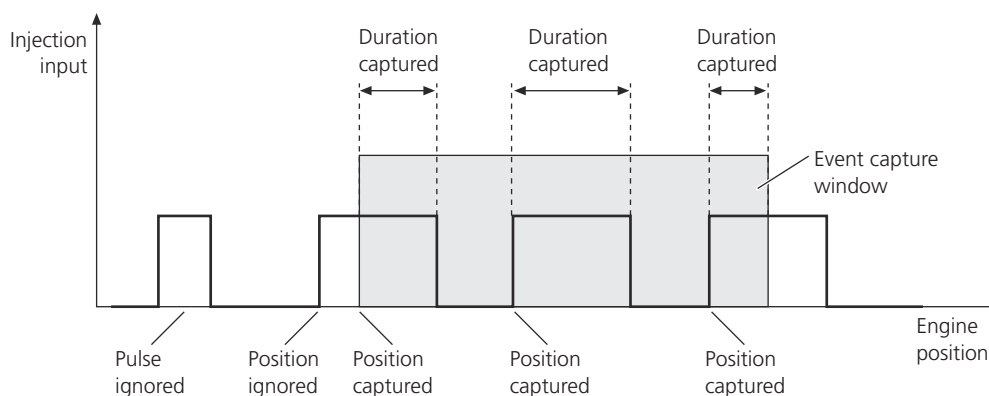
If either the leading or the trailing edge of a pulse is outside the event capture window, the position value of the corresponding window border is counted as an edge.

If no leading and trailing edges of a pulse are captured (continuous injection), the borders of the event capture window are counted as edges.

If the leading edge of a pulse is in one event capture window and the trailing edge in the event window of the next engine cycle, two pulses will be captured.

If the event capture window covers the whole engine cycle (0 ... 720°), an input pulse that overlaps the border of the window is detected as two separate pulses.

Duration mode (start position, duration) The position values of leading edges and the durations of up to 64 pulses are evaluated in each event capture window. The values are measured relative to the TDC. Further pulses in the same event capture window are ignored. You can specify the number of expected pulses to minimize execution time.

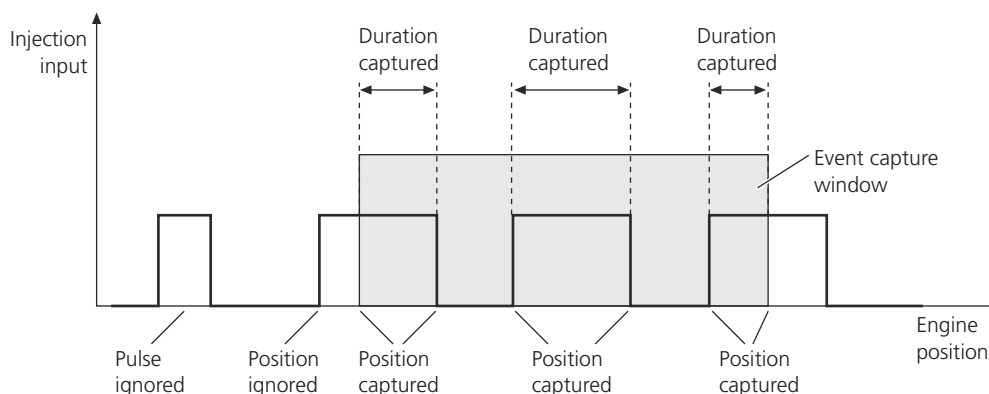


If either the leading or the trailing edge of a pulse is outside the event capture window, it is assumed that the pulse starts/ends at the respective border of the window.

If no leading and trailing edges of a pulse are captured (continuous injection), the duration is assumed to be the duration of the whole event capture window.

If the event capture window covers the whole engine cycle (0 ... 720°), an input pulse that overlaps the border of the window is detected as two separate pulses.

Position and duration mode (start position, end position, duration) This mode can be used starting from board revision 3 and FPGA revision 3. The position values of leading and trailing edges and the durations of up to 64 input pulses are evaluated in each event capture window. The values are measured relative to the TDC. You can specify the number of expected pulses to minimize execution time.



If either the leading or the trailing edge of a pulse is outside the event capture window, the position value of the respective window border is counted as an edge.

If no leading and trailing edges of a pulse are captured (continuous injection), the borders of the event capture window are counted as edges.

If the leading edge of a pulse is in one event capture window and the trailing edge in the event window of the next engine cycle, two pulses are captured.

If the event capture window covers the whole engine cycle (0 ... 720°), an input pulse that overlaps the border of the window is detected as two separate pulses.

Absolute mode (start position, end position, start time stamp, end time stamp)

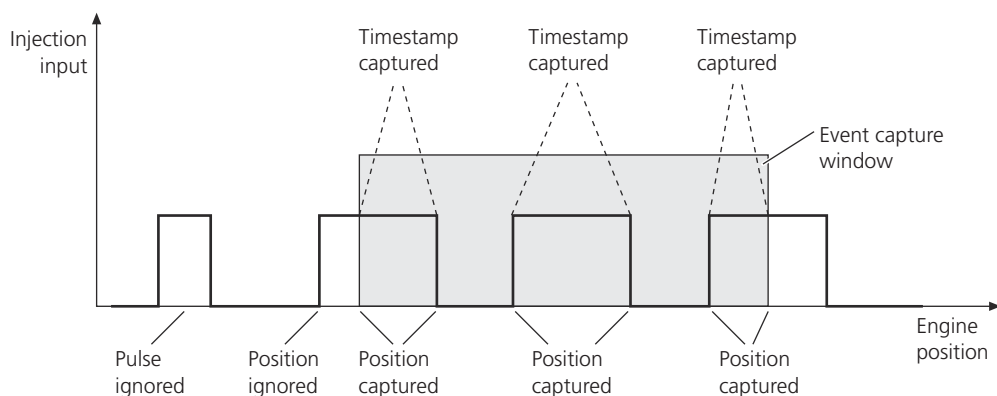
This mode can be used starting from board revision 3 and FPGA revision 3. The position values and time stamps of leading and trailing edges of up to 64 input pulses are evaluated within each event capture window.

The values are measured as absolute values, relative to a user-defined starting point. Initially, the starting point is set to the start of the APU. Position values and time values are increased until you reset the starting point using the DS2211APU_ABS_CNT_RESET_Bx block or the DS2211_ABS_COUNTER_RESET function.

The maximum length of a measurement without resetting the starting point is shown in the following table.

Absolute values	Resolution	Update Rate	Maximum Duration of Measurement Without Reset
Time values	40 bit	250 ns	Approx. 76 hours
Position values (angles)	24 bit	Depending on revolutions per minute (rpm)	<ul style="list-style-type: none"> At 6000 rpm: Approx. 93 hours. At 12000 rpm: Approx. 46 hours.

You can specify the number of expected pulses to minimize the execution time.



If either the leading or the trailing edge of a pulse is outside the event capture window, the position value of the respective window border is counted as an edge.

If no leading and trailing edges of a pulse are captured (continuous injection), the borders of the event capture window are counted as edges.

If the leading edge of a pulse is in one event capture window and the trailing edge in the event window of the next engine cycle, two pulses are captured.

If the event capture window covers the whole engine cycle (0 ... 720°), an input pulse that overlaps the border of the window is detected as two separate pulses.

For reference information including the I/O mapping, refer to [Injection Pulse Position and Fuel Amount Measurement](#) on page 111.

Related topics

Basics

[DS2211 Board Revision](#)..... 196

References

[APU Reference](#)..... 104

[DS2211_ABS_COUNTER_RESET](#) (DS2211 RTLib Reference )

[DS2211APU_ABS_CNT_RESET_Bx](#) (DS2211 RTI Reference )

Event Capture Windows

Introduction

You can define event capture windows for spark event and injection event capturing. These windows allow you to capture different signals or situations on the cylinders of an engine. You can define one or two event capture windows within one motor cycle for each signal. Only pulses that occur within the range of the event capture window will be recognized. You can specify the number of expected pulses to minimize the execution time.

Capture modes

For spark event capture, you can use the following capture modes:

- Single event capture mode
- Multiple event capture mode

Refer to [Spark Event Capture Unit](#) on page 83.

For injection event capture, you can use the following capture modes:

- Position mode (start position, end position)
- Duration mode (start position, duration)
- Position and duration mode (start position, end position, duration)
- Absolute mode (start position, end position, start time stamp, end time stamp)

Refer to [Injection Event Capture Unit](#) on page 84.

Continuous value capture

For spark event and injection event capture, you can continuously measure up to 64 events per channel via the capture FIFO buffer. Continuous value capturing can cover the whole engine cycle of 0 ... 720° using one event capture window. Refer to [Continuous Value Capturing](#) on page 92.

Setting event capture windows

To define event capture windows in Simulink models, you can use the following RTI blocks.

RTI Blocks of APU Blockset	RTI Blocks of VARAPU Blockset
<ul style="list-style-type: none"> ▪ DS2211APU_INJ_Bx_Gy ▪ DS2211APU_IGN_Bx ▪ DS2211APU_AUXCAP_Bx_Cy ▪ DS2211APU_IGNCONT_Bx ▪ DS2211APU_INJCONT_Bx_Gy ▪ DS2211APU_AUXCAPCONT_Bx_Cy 	<ul style="list-style-type: none"> ▪ DS2211VARAPU_IGN_Bx ▪ DS2211VARAPU_INJ_Bx_Gy ▪ DS2211VARAPU_AUXCAP_Bx_Cy

To define event capture windows in handcoded application, use the `ds2211_multi_eventwin_set` RTLib function.

Restrictions

The following restrictions apply to event capture windows:

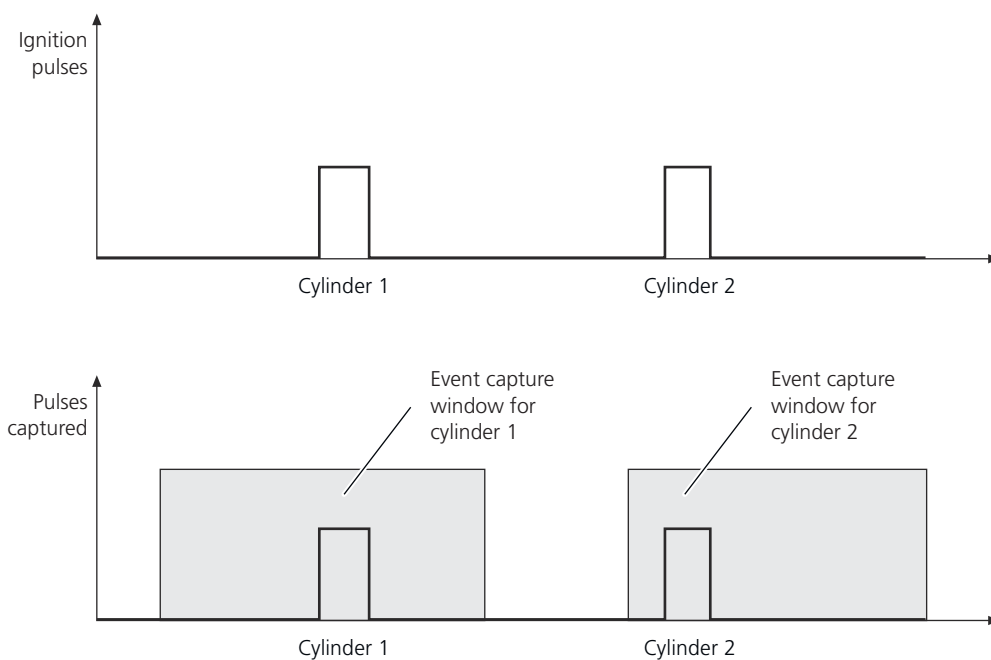
- An event window must not cover the whole engine cycle of 0 ... 720°. The maximum width of an event window is 719.824° (= 12.5633 rad).
Continuous value capturing is an exception. It allows you to cover the whole engine cycle of 0 ... 720°.
- The minimum width of an event window is 0.176° (= 0.003 rad).
- If you specify the same value for the start and the end positions, no event capture window will be set and an error message is issued if RTI is being used.
- The minimum distance between two event windows is 16 engine position steps ($16 \cdot 720^\circ / 65536 = 0.176^\circ$).

The following restrictions apply if you define two event capture windows:

- The sum of both event capture windows must not exceed the maximum width of 720°.
- The range of the event capture windows must not overlap each other.

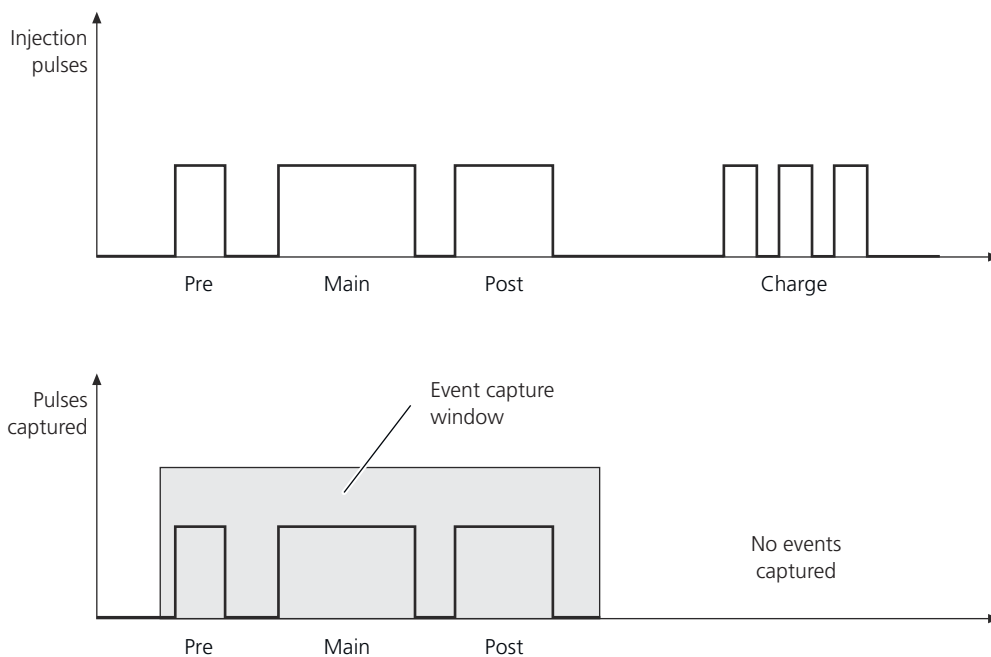
Example of Double Spark Ignition

There are two ignition pulses per coil but only one signal for the two cylinders. The cylinder is identified by its engine angle. You can use event capture windows to specify possible ignition pulse positions.

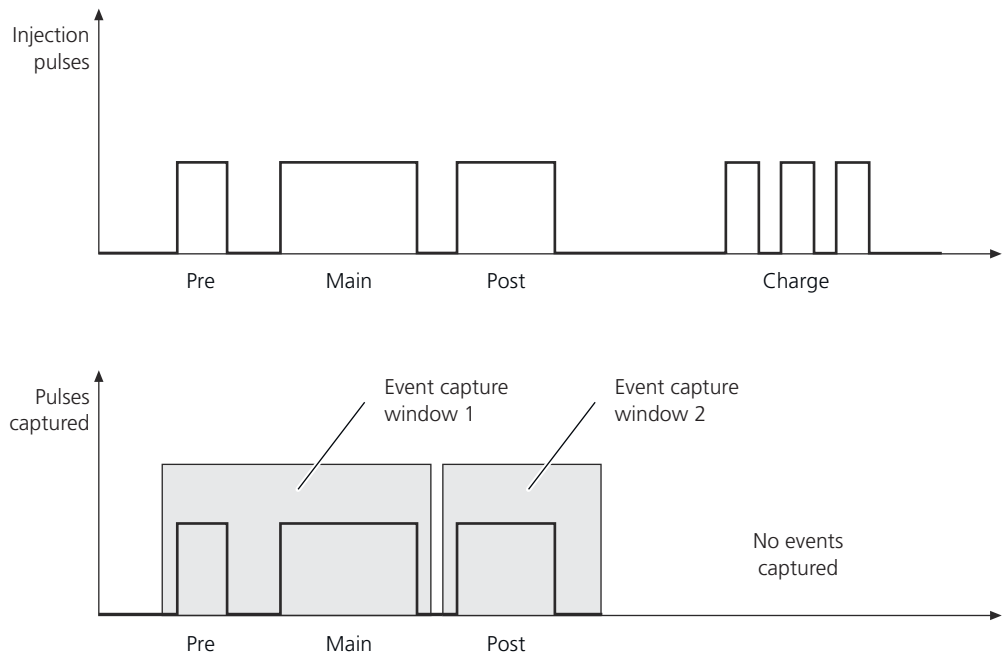


Example of Common Rail Injection

The start positions and durations of several injection pulses can be captured. Additional charge pulses can be ignored using an event capture window of an appropriate length.

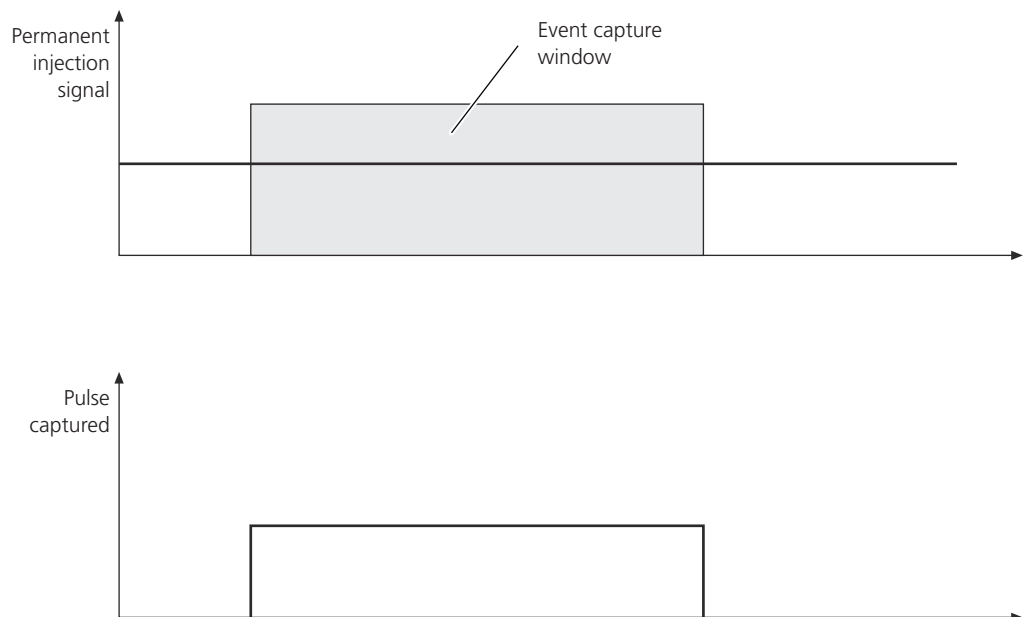


You can use two event capture windows to distinguish between torque and exhaust relevant injection.



Example of Permanent Injection

The width of the event capture window is captured for a permanent injection signal.



Related topics**Basics**

Continuous Value Capturing.....	92
Injection Event Capture Unit.....	84
Spark Event Capture Unit.....	83

Continuous Value Capturing

Introduction

Several engine models need continuous detection of ignition and/or injection position, and duration values. For example, in common rail engines, you need the current rail pressure continuously. The values can be read from the capture FIFO. You can define capture windows for the whole engine cycle of 0 ... 720°. Continuous capturing uses the same input lines as "normal" capturing.

Spark event capture

In single event mode, up to 64 position values per channel of the leading edge of the first input pulse within the capture window are read. In multiple event capture mode, up to 64 position values per channel of trailing and leading edges can be read.

Injection pulse position and duration

In position mode, the position values of all trailing and leading edges of up to 32 pulses (= 64 events) per channel can be read within one sample hit (the sample hit is defined by the sample time of your model). In duration mode, the position values of all leading edges and the duration of up to 32 pulses per channel within one sample hit can be read (refer to [Injection Event Capture Unit](#) on page 84).

Auxiliary event capture

Two channels are available to measure various positions. In single event mode, up to 64 position values of the leading edge of the first input pulse are evaluated. In multiple event mode, up to 64 position values of leading and trailing edges can be captured. Auxiliary event capture is similar to spark event capture (refer to [Spark Event Capture Unit](#) on page 83).

Note

The channels of the spark event capture unit or the injection event capture unit can be read simultaneously in the event window mode and the continuous mode by using RTLib functions. Refer to [Angular Processing Unit \(APU\) \(DS2211 RTLib Reference !\[\]\(a8f9309f944226d1420f5fed22e2b6e6_img.jpg\)](#)).

Related topics**Basics**

[Event Capture Windows.....](#) 88

References

[Injection Pulse Position and Fuel Amount Measurement.....](#) 111
[Spark Event Capture.....](#) 109

Complex Comparators

Introduction

The complex comparators can be used to capture complex signals.

Supported units

The complex comparator can be used by the spark event capture and the injection event capture unit. The following pins are connected to a complex comparator:

- Ignition capture: IGNx with x = 1 ... 6
- Auxiliary capture: AUXCAPx with x = 1 or 2
- Injection capture: INJx with x = 1 ... 6
- Injection capture: INJx with x = 7 or 8 (shared with PWM and square-wave signal measurement)

Characteristics

Each pin is connected to two complex comparators, named A and B. These are set by two parameters, threshold voltage and hysteresis, which can be set using RTI blocks or RTLib functions. In RTI the parameters are tuneable, that means, they can be set per block parameter or block input port (accessible via the experiment software).

Threshold voltage The threshold voltage of comparator A can be set individually for different groups or pins in the range 1 V ... 23.8 V (group 1 IGN1 .. IGN6, group 2 INJ1 .. INJ6, and the pins: AUXCAP1, AUXCAP2, INJ7, INJ8). The threshold voltage of comparator B can be set in groups of 8 pins in the range 1 V ... 22.65 V (group 1 is IGN1 ... IGN6, AUXCAP1, AUXCAP2 and group 2 is INJ1 ... INJ8).

Hysteresis The hysteresis of comparator A is always 0.2 V. It cannot be changed. The hysteresis of comparator B can be set to up to 2.4 V via RTI blocks or RTLib functions.

Dependency of the comparators To get reasonable results, the voltage ranges that are measured by the comparator B and A must not overlap. This means that the threshold voltage (TH_B) minus half of the hysteresis (Hyst_B) of

comparator B must be larger than the threshold voltage (TH_A) plus half of the hysteresis (HYST_A) of comparator A:

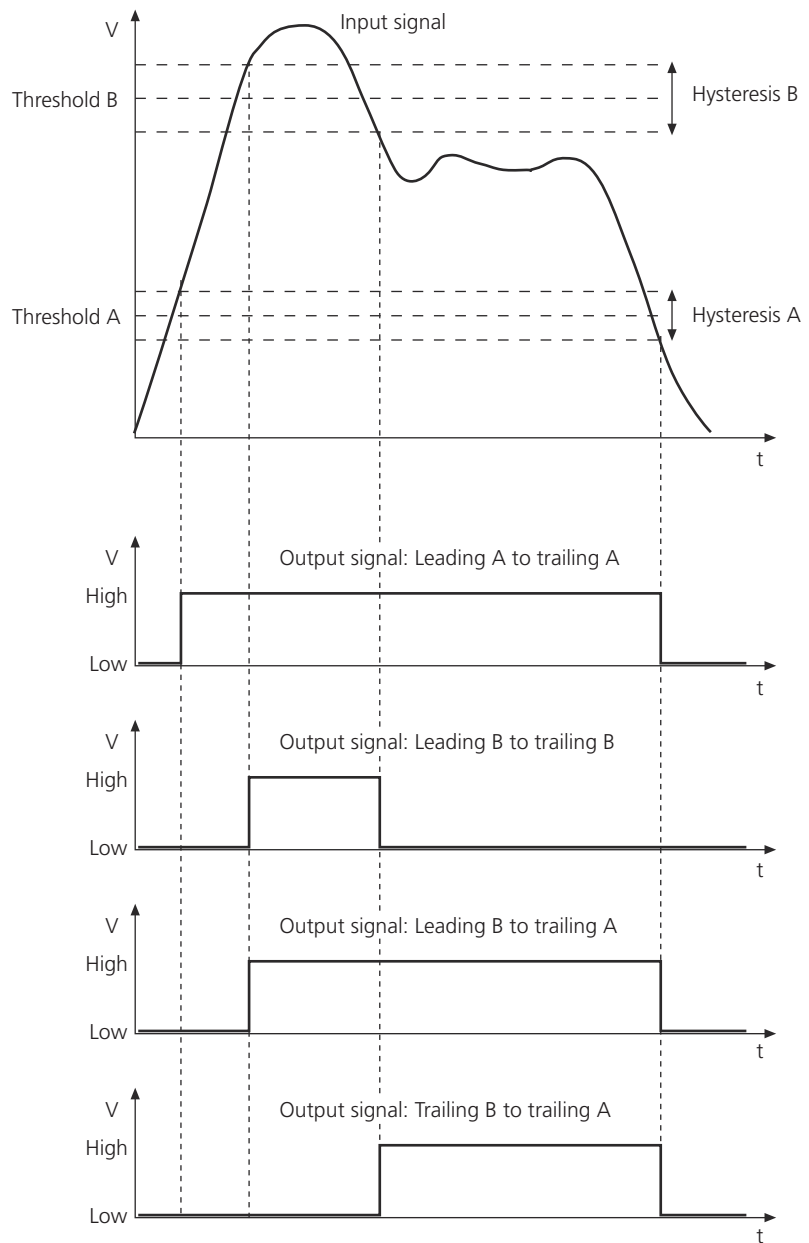
$$TH_B - 0.5 \cdot Hyst_B > TH_A + 0.5 \cdot Hyst_A$$

Signal capture modes

The complex comparator has four different signal capture modes, which are defined by the threshold voltages (TH) and hysteresis (Hyst). The signal capture modes define which times are considered for capturing.

Mode	Description
Leading A to trailing A	Time between rising edge A and falling edge A: $TH_A + Hyst_A / 2 \dots TH_A - Hyst_A / 2$
Leading B to trailing B	Time between rising edge B and falling edge B: $TH_B + Hyst_B / 2 \dots TH_B - Hyst_B / 2$
Leading B to trailing A	Time between rising edge B and falling edge A: $TH_B + Hyst_B / 2 \dots TH_A - Hyst_A / 2$
Trailing B to trailing A	Time between falling edge B and falling edge A: $TH_B - Hyst_B / 2 \dots TH_A - Hyst_A / 2$

The following illustration shows a voltage curve corresponding to a typical current through an injection valve and the output signal in the different modes.

**Related RTI block**





DS2211DIO_SETUP_Bx

Related RTLib functions

ds2211_digin_threshold_set, ds2211_apu_ignition_cc_setup,
ds2211_apu_injection_cc_setup

Related topics

References

[ds2211_apu_ignition_cc_setup](#) (DS2211 RTLib Reference )
[ds2211_apu_injection_cc_setup](#) (DS2211 RTLib Reference )
[ds2211_digin_threshold_set](#) (DS2211 RTLib Reference )
[DS2211DIO_SETUP_Bx](#) (DS2211 RTI Reference )

Angular Processing Unit - Variant

Simulating engine variants	If you use the VAR APU blockset, you can change the engine variant without the need to rebuild and download the real-time application. All relevant parameters can be changed via the experiment software.
-----------------------------------	--

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

Basics on Simulating Engine Variants.....	97
A dSPACE simulator or a processor board with a DS2211 can be used to simulate different engine variants.	
Building a Simulink Model for Engine Variants.....	99
You should use the blocks of the VAR APU blockset to build a Simulink model for engine variants.	
Multiple Event Capture Mode of the VAR APU Blockset.....	100
In contrast to the blocks of the APU blockset, the capture blocks of the VAR APU blockset behave differently.	
Example of Capturing a Multiple Event with the VAR APU Blockset.....	101
The example demonstrates how the output values of a capture block change when a signal is captured in the multiple capture mode.	

Basics on Simulating Engine Variants

Introduction	A dSPACE simulator or a processor board with a DS2211 can be used to simulate different engine variants.
---------------------	--

APU blockset	If you work with the APU (Angular Processing Unit) blockset, you can build the model for one engine variant only. You have to rebuild and download the real-time application if you want to simulate another engine variant. This requires a lot of time, especially when working with large engine models.
---------------------	---

VAR APU blockset	The Angular Processing Unit - Variant (VAR APU) blockset is designed for simulating engine variants. All relevant parameters can be modified during run time via the experiment software. The changed values take effect only when the simulation state of the model changes from STOP to RUN or PAUSE.
-------------------------	---

Capture modes

The capture method of the VAR APU blockset differs from the mode implemented with the APU blockset. The parameters of the capture blocks are not only updated at the end of an event capture window but are also updated even within the opened event capture window.

Modifiable parameters

The modifiable parameters to allow a flexible handling of engine variants are:

- Number of cylinders
- Depending on the TDC (top dead center) calculation:
 - cylinder sequence and first TDC position or
 - firing sequence
- Start and end position of event windows
- Cylinder sequence of the knock signals

Limitations

If you use the VAR APU blocks, consider the following limitations:

- Do not mix the blocks of the VAR APU and APU blockset.
- Do not use the VAR APU blockset if you want to connect the DS2211 with a DS2210 via the engine position bus (cascading). The DS2210 does not support the VAR APU blockset.

Related RTI blocks

DS2211VARAPU_CRANK_Bx, DS2211VARAPU_ANG_REL_Bx,
DS2211VARAPU_IGN_Bx, DS2211VARAPU_INJ_Bx_Gy,
DS2211VARAPU_AUXCAP_Bx_Cy, and DS2211VARSL_KNSG_Bx_Cy

Related RTLib function

ds2211_multiwin_ign_cap_read_var,
ds2211_multiwin_ign_cap_read_var_ext,
ds2211_multiwin_ign_cap_read_var_abs,
ds2211_multiwin_inj_cap_read_var,
ds2211_multiwin_inj_cap_read_var_ext, and
ds2211_multiwin_inj_cap_read_var_abs

Related topics**Basics**

[Simulating Engine Variants.....](#) 178

Building a Simulink Model for Engine Variants

Introduction

If you want to build a Simulink model for engine variants, you should use the blocks of the VAR APU blockset.

Note

Do not mix blocks of the VAR APU and APU blockset in one model. Only blocks which are shared can be used with both blocksets (see table below).

Differences of VAR APU and APU blocks

The following table shows the differences of the blocks of the VAR APU blockset to the APU blockset:

RTI Block ¹⁾	Differences
DS2211VARAPU_CRANK_Bx	This block specifies the engine parameters: <ul style="list-style-type: none"> ▪ Number of cylinders ▪ Cylinder sequence and first TDC (top dead center) ▪ Firing sequence These parameters can be modified via the experiment software.
DS2211VARAPU_CAM_Bx_Cy	No difference (shared with the APU blockset)
DS2211APU_ANG_Bx	No difference (shared with the APU blockset)
DS2211APU_ANG_REL_Bx	The block's output depends on the engine parameters specified by DS2211VARAPU_CRANK_Bx. No difference in the functionality.
DS2211VARAPU_IGN_Bx	The block's parameters depend on the engine parameters specified by DS2211VARAPU_CRANK_Bx. The start and end positions of the first and second event windows are relative to the position of the TDC. In addition you can modify the trigger mode. Event capturing uses an extended method. This method supplies the captured events on based on event windows and count the events continuously. Refer to Multiple Event Capture Mode of the VAR APU Blockset on page 100.
DS2211VARAPU_INJ_Bx_Gy	
DS2211VARAPU_AUXCAP_Bx_Cy	
DS2211APU_ABS_CNT_RESET_Bx	No difference (shared with the APU blockset)
DS2211APU_INT_Bx_Iy	No difference (shared with the APU blockset)
DS2211VARSL_KNSG_Bx_Cy	The block's parameters depend on the engine parameters specified by DS2211VARAPU_CRANK_Bx. All engine-related parameters are tunable and can be changed at run time.

¹⁾ For details on the RTI blocks, refer to [Angular Processing Unit - Variant \(DS2211 RTI Reference\)](#).

Implementing the model	Implement the engine model using the VAR APU blockset. The real-time model must be implemented for the highest number of cylinders. As the width of the in- and outputs are fixed and cannot be changed during run time, you can simulate only engine variants with equal or less number of cylinders. In addition, you can only simulate with two event capture windows when it was specified in the real-time model.
Build and download	Build and download the real-time application as usual. This must be done only once for the real-time application.
Varying the engine parameters	The engine parameters are available in the SDF file, so they can be changed in the real-time application via the experiment software. The changes on the parameter values get valid during the transition of the simulation state from STOP to PAUSE or RUN. If new parameter values are invalid, appropriate error messages are displayed and the last values of these parameters remain valid. Nevertheless, the simulation starts. It cannot be stopped due to technical reason. The simulation may run with a mix of old and new parameter values.

Related topics**Basics**

[Simulating Engine Variants..... 178](#)

Multiple Event Capture Mode of the VAR APU Blockset

Introduction	In contrast to the blocks of the APU blockset, the capture blocks of the VAR APU blockset behave differently.
Fixed dimension	The output data has a fixed dimension, which is mainly determined by the number of expected pulses and the number of event windows.
Update behavior	<p>The capture blocks update the output values at every sample step. The output values are updated even within the current event capture window. This update behavior makes it possible to measure the torque relevant injection pulse faster. This is required by new engine ECUs for an improved control of the engine's idle speed.</p> <p>The output data of the capture blocks can be a mix of old and new data. To distinguish the data, the blocks have two additional output parameters: update state and update counter.</p>

Update state The `update state` parameter displays the current status of the update process. `Update state` is 0 when the block data is currently being updated. The output port has data from the current and previous event capture window. The `update state` is set to 0 when the first event of a event capture window is captured. Consequently, a value of 0 means that an event capture window is opened. `Update state` is 1 when the number of expected pulses or the end position of the event capture window is reached. `Update state` remains 1 until the first event of the following event capture window is captured. A value of 1 means that no data is currently updated, the block output data is complete for the last event capture window.

Update counter The `update counter` parameter counts the number of pulses within the currently active event capture window. To distinguish between leading and trailing edges, it alternates its sign. The value is negative for a leading edge and positive for a trailing edge.

Related topics

Examples

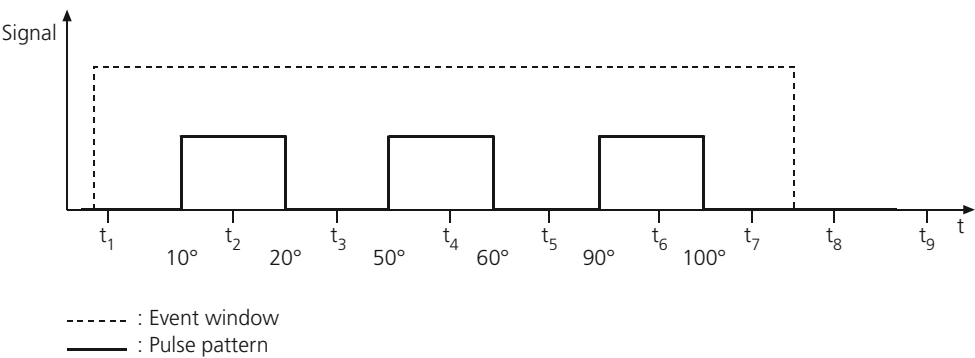
[Example of Capturing a Multiple Event with the VAR APU Blockset.....](#) 101

Example of Capturing a Multiple Event with the VAR APU Blockset

Introduction

This example demonstrates how the output values of a capture block change when a signal is captured in the multiple capture mode.

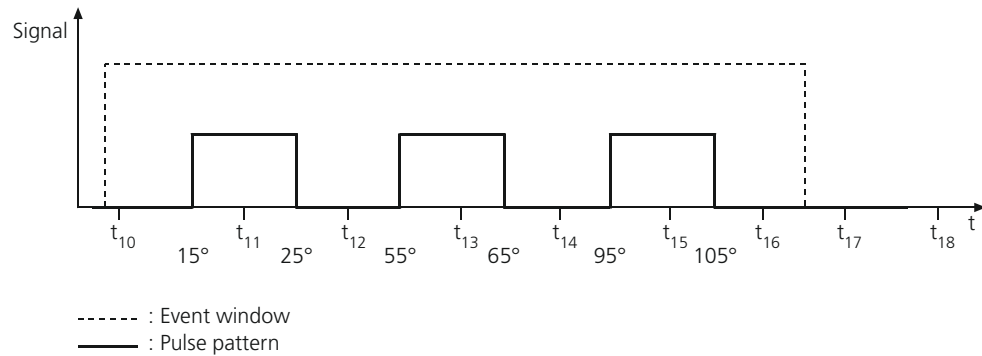
The following illustration shows the captured signal. Three pulses are expected within one defined capture event window. The initial value of the start and end position is set to 999.



The following table shows the values of the parameters when the pulses are captured for the first time. Updated or changed data is italicized. t_n are the sample steps.

Output	Values								
Block execution	t ₁	t ₂	t ₃	t ₄	t ₅	t ₆	t ₇	t ₈	t ₉
Start1	999	<i>10</i>	10	10	10	10	10	10	10
Start2	999	999	999	<i>50</i>	50	50	50	50	50
Start3	999	999	999	999	999	<i>90</i>	90	90	90
End1	999	999	<i>20</i>	20	20	20	20	20	20
End2	999	999	999	999	<i>60</i>	60	60	60	60
End3	999	999	999	999	999	999	<i>100</i>	100	100
Count	0	0	0	0	0	0	0	3	3
State	0	0	0	0	0	0	0	0	0
Update counter	0	<i>-1</i>	<i>1</i>	<i>-2</i>	<i>2</i>	<i>-3</i>	<i>3</i>	<i>3</i>	<i>3</i>
Update state	0	0	0	0	0	0	<i>1</i>	<i>1</i>	<i>1</i>

The following illustration shows the signal in a subsequent engine cycle. The pulses start 5 degrees later.



The following table shows the values of the parameters when the pulses are captured. Updated or changed data is italicized. t_n are the sample steps.

Output	Values								
Block execution	t ₁₀	t ₁₁	t ₁₂	t ₁₃	t ₁₄	t ₁₅	t ₁₆	t ₁₇	t ₁₈
Start1	10	<i>15</i>	15	15	15	15	15	15	15
Start2	50	50	50	<i>55</i>	55	55	55	55	55
Start3	90	90	90	90	90	<i>95</i>	95	95	95
End1	20	20	<i>25</i>	25	25	25	25	25	25
End2	60	60	60	60	<i>65</i>	65	65	65	65
End3	100	100	100	100	100	100	<i>105</i>	105	105
Count	3	3	3	3	3	3	3	3	3
State	0	0	0	0	0	0	0	0	0
Update counter	3	<i>-1</i>	<i>1</i>	<i>-2</i>	<i>2</i>	<i>-3</i>	<i>3</i>	<i>3</i>	<i>3</i>
Update state	1	<i>0</i>	0	0	0	0	<i>1</i>	<i>1</i>	<i>1</i>

Related topics

Basics

Multiple Event Capture Mode of the VAR APU Blockset..... 100

APU Reference

Introduction

The following sections provide the reference information you need to implement your real-time model with engine simulation functions provided by the angular processing unit.

Where to go from here

Information in this section

[Crankshaft Sensor Signal Generation..... 104](#)

The crankshaft signal generator has one analog and one digital crankshaft output.

[Camshaft Sensor Signal Generation..... 106](#)

Provides general information on the camshaft signal generator and its I/O mapping.

[Wave Table Generation..... 108](#)

Wave tables are used to define different wave forms for crankshaft and camshaft sensor signal generation.

[Spark Event Capture..... 109](#)

The spark event capture unit provides 6 digital ignition inputs for ignition position measurement and 2 digital auxiliary capture inputs for various position measurements.

[Injection Pulse Position and Fuel Amount Measurement..... 111](#)

The injection event capture unit provides 16 digital injection inputs split into 2 groups for injection pulse position and fuel amount measurement.

Information in other sections

[APU Basics..... 72](#)

You can find basic information on how simulation of engine core functions works. The most important terms when working with RTI and RTLib are explained.

[Knock Sensor Simulation..... 117](#)

Knock sensor simulation is performed by a ready-to-use application implemented on the slave DSP.

Crankshaft Sensor Signal Generation

Characteristics

The crankshaft signal generator has one analog and one digital crankshaft output. Both the analog and the digital outputs provide the same crankshaft information. The digital output generates pulse patterns that represent the sign

of the analog wave form. For basic information on how crankshaft sensor signal generation works, refer to [Crankshaft Signal Generator](#) on page 77.

Using wave tables, you can define specific crankshaft wave forms to simulate different crankshaft types. The DS2211 can load 8 different wave tables for crankshaft sensor signal generation. The wave form to be generated can be switched during run time by using RTI or RTLib functions. For detailed information on wave tables, refer to [Wave Table Generation](#) on page 108.

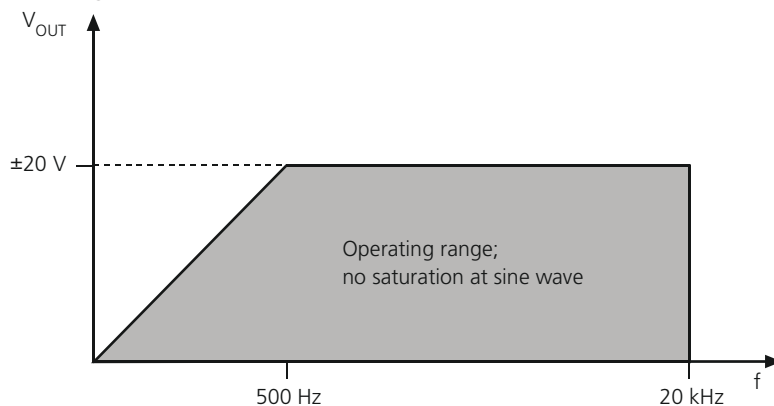
Digital output The digital output has a push-pull driver running from an external source (VBAT within the range 5 ... 60 V). It is protected against overvoltage up to 60 V. The maximum output current is ± 50 mA. Short circuit protection to GND and VBAT is implemented. The digital outputs are in their high impedance state after reset or after power-up. When RTI is used, the output is enabled if the block is part of the model. You can enable or disable the output using RTLib functions.

Analog output The analog output signal is fed through an output transformer to provide signals that are decoupled from ground. The transformer has a maximum physical signal level of 40 V_{pp}. Within this range, you can change the amplitude of the whole wave form at run time. Transformers do not transmit a DC voltage. If the signal has a DC voltage component, you can bypass the transformer by modifying the jumper settings. For details, refer to [Jumper Settings J1 ... J7 \(PHS Bus System Hardware Reference \[1\]\)](#). The analog output can be enabled or disabled.

Note

The nominal operating range of the transformers is 500 Hz ... 20 kHz with full output voltage. They can operate from 40 Hz up to 500 Hz if the amplitude of the signal is reduced.

The amplitude of the output signal must be reduced to avoid saturation of the transformer and with it a distortion of the output signal. The amplitude must be reduced proportionally to the frequency, see the operating range in the following illustration.



Neither the RTI blocks nor the RTLib function reduce the amplitude automatically, you must therefore reduce the amplitude in the real-time model.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the corresponding measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(3dfb8d66e81160ad61421a3452093d1b_img.jpg\)](#)).

I/O mapping

The following table shows the mapping of the crankshaft output (digital and analog) to the related I/O pins of the I/O connector P2.

Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage Range/Output Current
Digital	P2	45	P2B	41	CRANK_DIG	Digital crankshaft output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
Analog	P2	68	P2A	12	CRANK+	Crankshaft wave form output	With transformer (AC output mode): (CRANK+ – CRANK–) = ± 20 V Bypassed transformer (DC output mode): (CRANK+ – GND) = ± 10 V; ± 5 mA
	P2	70	P2A	45	CRANK–		

Related topics**References**

[Crankshaft Sensor Signal Generation \(DS2211 RTLib Reference !\[\]\(3211b5d1d968fc1665909b34f9f16010_img.jpg\)](#))
[Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(d47ad152ec3d86a04ad64c8049e1f17f_img.jpg\)](#))
[DS2211APU_CRANK_Bx \(DS2211 RTI Reference !\[\]\(6b7fbb0b7bdb78cadf73d50851a4dfb1_img.jpg\)](#))
[DS2211DIO_SETUP_Bx \(DS2211 RTI Reference !\[\]\(cd0f39e2b8d76d7e84d5eed1ed02b2df_img.jpg\)](#))
[Power Supply Outputs \(PHS Bus System Hardware Reference !\[\]\(11e014e490252374320bfedb9a33c896_img.jpg\)](#))
[Transformer Outputs \(APU and Slave DSP\) \(PHS Bus System Hardware Reference !\[\]\(4971f4a68f0e91a16206aa5c8215b1e8_img.jpg\)](#))

Camshaft Sensor Signal Generation

Characteristics

The camshaft signal generator provides four camshaft sensor signals. Two signals are digital and analog, two signals are digital only. The digital signals are pulse patterns that represent the sign of the corresponding analog wave forms. For basic information on how camshaft sensor signal generation works, refer to [Camshaft Signal Generator](#) on page 81.

Using wave tables, you can define specific camshaft wave forms to simulate different camshaft types. The DS2211 is capable of loading 8 different wave tables for each camshaft output. The wave form to be generated can be switched during run time. For detailed information on wave tables, refer to [Wave Table Generation](#) on page 108.

Digital outputs The digital outputs have push-pull drivers running from an external source (VBAT within the range 5 ... 60 V). The maximum output current per channel is ± 50 mA. Short circuit protection to GND and VBAT is

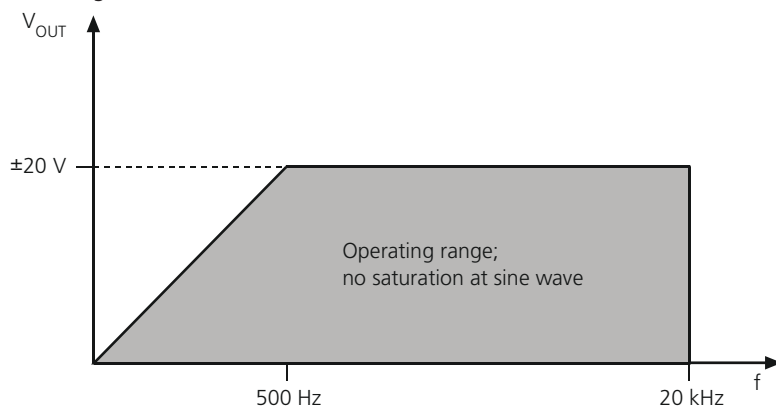
implemented. The outputs are in their high impedance state after reset or after power-up. You can enable or disable the outputs using RTI/RTLib functions.

Analog outputs The analog output signals are fed through output transformers to provide signals that are decoupled from ground. The transformers have a maximum physical amplitude of $40 V_{pp}$. Within this range, you can change the amplitude of the whole wave form at run time. Transformers do not transmit a DC voltage. If the signals have a DC voltage component, you can bypass the transformers via jumper settings. For details, refer to [Jumper Settings J1 ... J7 \(PHS Bus System Hardware Reference !\[\]\(c507f772dba2b921f86777f01218e570_img.jpg\)\)](#). The analog outputs can be enabled or disabled.

Note

The nominal operating range of the transformers is 500 Hz ... 20 kHz with full output voltage. They can operate from 40 Hz up to 500 Hz if the amplitude of the signal is reduced.

The amplitude of the output signal must be reduced to avoid saturation of the transformer and with it a distortion of the output signal. The amplitude must be reduced proportionally to the frequency, see the operating range in the following illustration.



Neither the RTI blocks nor the RTLib function reduce the amplitude automatically, you must therefore reduce the amplitude in the real-time model.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(5361750c22c4e047a52f4eac1ec2d4cc_img.jpg\)\)](#).

I/O mapping

The following table shows the mapping of the camshaft outputs (analog and digital) to the related I/O pins of the I/O connector P2. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 197.

Channel		Connector Pin		Sub-D Pin		Signal	Description	Voltage Range/Output Current
1 (A)	Digital	P2	47	P2B	25	CAM1_DIG	Digital camshaft output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
	Analog	P2	72	P2A	29	CAM1+	Camshaft wave form output	With transformer (AC output mode): (CAM1+ – CAM1–) = ±20 V Bypassed transformer (DC output mode): (CAM1+ – GND) = ±10 V; ±5 mA
		P2	74	P2A	13	CAM1–		
2 (B)	Digital	P2	49	P2B	9	CAM2_DIG	Digital camshaft output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
	Analog	P2	76	P2A	46	CAM2+	Camshaft wave form output	With transformer (AC output mode): (CAM2+ – CAM2–) = ±20 V Bypassed transformer (DC output mode): (CAM2+ – GND) = ±10 V; ±5 mA
		P2	78	P2A	30	CAM2–		
3 (C)	Digital	P2	62	P2A	11	CAM3_DIG	Digital camshaft output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
4 (D)	Digital	P2	64	P2A	44	CAM4_DIG	Digital camshaft output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA

Related topics

References

[Camshaft Sensor Signal Generation \(DS2211 RTLib Reference !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)](#))
[Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(034433b90593e82e5460e34e3ed48e9b_img.jpg\)](#))
[DS2211APU_CAM_Bx_Cy \(DS2211 RTI Reference !\[\]\(5f24500834b50a8307ffe63e419281a9_img.jpg\)](#))
[DS2211DIO_SETUP_Bx \(DS2211 RTI Reference !\[\]\(8502790a2fc8970abb55aa7f7a7a6bae_img.jpg\)](#))
[Power Supply Outputs \(PHS Bus System Hardware Reference !\[\]\(7f7375e819602f97f5594a28879b3e09_img.jpg\)](#))
[Transformer Outputs \(APU and Slave DSP\) \(PHS Bus System Hardware Reference !\[\]\(f947a10a261625f35f2b4defe4317e0a_img.jpg\)](#))

Wave Table Generation

Wave table

Wave tables are used to define different wave forms for crankshaft and camshaft sensor signal generation.

Wave table basics

According to the 16-bit resolution of the engine position, each wave table defines the signal wave form for $2^{16} = 65536$ consecutive engine positions within the range $0 \dots 720^\circ$ (4π). The signal level must be specified in the range $-1.0 \dots +1.0$. During wave table generation (see below), the signal level is converted with a resolution of 12-bit signed ($-2048 \dots +2047$). The corresponding signal value is proportional to the wave table data. The amplitude can be set via RTI and RTLib (max. 40 Vpp). Due to the output transformers, the mean value of the wave table data must be zero. The assignment of signal levels to engine positions results in a wave form.

Wave tables are downloaded to the processor board together with the real-time application. The real-time processor transfers the wave tables to the DS2211, where they are stored in the memories of the crankshaft and camshaft signal generators.

At run time, the wave table look-up mechanism outputs the signal value, which is defined for the current engine position (every 250 ns).

Generating wave tables

Wave tables must be supplied as MAT files. Other formats are not supported. In MATLAB, you can create wave forms by using standard functions or you can import data measured at a real engine.

You can find example wave tables in
`<RCP_HIL_InstallationPath>\Demos\DS100x\IOBoards\DS2211\APU\WaveTables`.

Generating wave tables using MATLAB

The M file `Mat2C2211.m` is installed in
`<RCP_HIL_InstallationPath>\matlab\rtlib100x\tools`. This collects the specified wave table MAT files in a common MAT file, which it converts to a C source file. Up to 40 wave tables can be loaded for each board (8 for each signal generation). Use the `DS2211_crank_table_load` or `DS2211_cam_table_load` functions to load wave tables to the real-time hardware.

Generating wave tables using RTI

The Simulink RTI blocks for crankshaft sensor signal generation and camshaft sensor signal generation allow you to select up to 8 wave table MAT files for each output channel. RTI generates a common MAT file and converts it to a C source file that is compiled and downloaded to the real-time hardware together with your real-time application.

Related topics

References

Camshaft Sensor Signal Generation.....	106
Crankshaft Sensor Signal Generation.....	104

Spark Event Capture

Characteristics

The spark event capture unit provides

- 6 digital ignition inputs (IGN1 ... IGN6) for ignition position measurement
- 2 digital auxiliary capture inputs (AUXCAP1, AUXCAP2) for various position measurements

For detailed information on capture modes and event capture windows, refer to [Spark Event Capture Unit](#) on page 83 and [Event Capture Windows](#) on page 88.

You can choose between active high and active low pulses.

To minimize execution time, you have to specify the number of expected pulses for each event capture window. Up to 64 pulses are allowed.

You can read up to 64 events from the capture FIFO continuously within one sample hit. You can use the whole event capture window within the range 0 ... 720°.

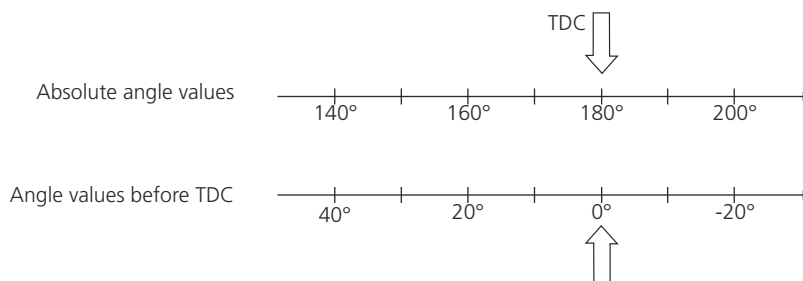
The inputs are 12/42 V compatible. They are protected against overvoltage up to 60 V. The voltage level when a pulse is captured can be defined with a complex comparator (refer to [Complex Comparators](#) on page 93). After software initialization, the input threshold is set to 2.5 V.

Note

There are different reference points for measuring engine positions:

- RTLib functions and the DS2211APU_AUXCAP_Bx_Cy, DS2211APU_AUXCAPCONT_Bx_Cy and DS2211VARAPU_AUXCAP_Bx_Cy RTI blocks measure relative to the absolute engine position in the range 0 ... 720°.
- The DS2211APU_IGN_Bx, DS2211APU_IGNCONT_Bx and DS2211VARAPU_IGN_Bx RTI blocks measure relative to top dead center (TDC).
- Using the absolute capture mode RTLib functions and RTI blocks measure relative to a user-defined starting point.

TDC as reference All angle values that are used in RTI are relative to “before TDC”. You get these values by subtracting the current absolute angle value from the absolute TDC angle value.



Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference\)](#).

I/O mapping

The following table shows the mapping of the cylinder/channel numbers to the related I/O pins of the DS2211 I/O connectors P1 and P2, as used in RTI and

RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 197.

RTI Cylinder/ Channel	RTLib Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage Range
1	1	P2	19	P2B	4	IGN1	Ignition capture	12/42 V compatible
2	2	P2	20	P2A	4	IGN2	Ignition capture	12/42 V compatible
3	3	P2	23	P2B	21	IGN3	Ignition capture	12/42 V compatible
4	4	P2	24	P2A	21	IGN4	Ignition capture	12/42 V compatible
5	5	P2	27	P2B	38	IGN5	Ignition capture	12/42 V compatible
6	6	P2	28	P2A	38	IGN6	Ignition capture	12/42 V compatible
1	7	P1	34	P1A	39	AUXCAP1	Auxiliary capture	12/42 V compatible
2	8	P1	36	P1A	23	AUXCAP2	Auxiliary capture	12/42 V compatible

Related topics

Basics

Event Capture Windows.....	88
Spark Event Capture Unit.....	83

References

[Digital Inputs \(PHS Bus System Hardware Reference !\[\]\(642aa997563f9a325b310230bb5078b7_img.jpg\)\)](#)
[DS2211APU_CAM_Bx_Cy \(DS2211 RTI Reference !\[\]\(9bef82f5a53106f2ad06a2de7acf5bcf_img.jpg\)\)](#)
[DS2211APU_IGN_Bx \(DS2211 RTI Reference !\[\]\(7ed4b959e7161d2c60a33aeb43710ff2_img.jpg\)\)](#)
[DS2211DIO_SETUP_Bx \(DS2211 RTI Reference !\[\]\(9a1c9bf02665d1d8af419e98d46187a2_img.jpg\)\)](#)
[Spark Event Capture \(DS2211 RTLib Reference !\[\]\(0eb1c3fb8762ba6f12cea583077849e5_img.jpg\)\)](#)

Injection Pulse Position and Fuel Amount Measurement

Characteristics

The injection event capture unit provides 16 digital injection inputs split into 2 groups for injection pulse position and fuel amount measurement. For detailed information on event capture windows and capture modes, refer to [Event Capture Windows](#) on page 88 and [Injection Event Capture Unit](#) on page 84.

You can choose between active high and active low pulses.

To minimize execution time, you have to specify the number of expected pulses for each event capture window. Up to 64 pulses are allowed.

You can read up to 64 events from the capture FIFO continuously within one sample hit. You can use the whole event capture window within the range 0 ... 720°.

The digital inputs are 12/42 V compatible (operational up to 60 V). The voltage level when a pulse is captured can be defined with a complex comparator (refer

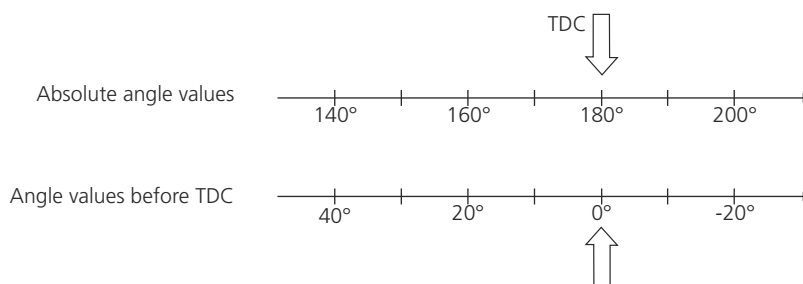
to [Complex Comparators](#) on page 93). After software initialization, the input threshold is set to 2.5 V.

Note

There are different reference points for measuring engine positions:

- RTLib functions and the DS2211APU_AUXCAP_Bx_Cy, DS2211APU_AUXCAPCONT_Bx_Cy and DS2211VARAPU_AUXCAP_Bx_Cy RTI blocks measure relative to the absolute engine position in the range 0 ... 720°.
- The DS2211APU_IGN_Bx, DS2211APU_IGNCONT_Bx and DS2211VARAPU_IGN_Bx RTI blocks measure relative to top dead center (TDC).
- Using the absolute capture mode RTLib functions and RTI blocks measure relative to a user-defined starting point.

TDC as reference All angle values that are used in RTI are relative to “before TDC”. You get these values by subtracting the current absolute angle value from the absolute TDC angle value.



I/O mapping

The following table shows the mapping of the cylinder/channel numbers to the related I/O pins of the DS2211 I/O connector P1, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 197.

The following table shows the I/O pins for group 1:

RTI Port/ Channel	RTLib Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage Range
1	1	P2	17	P2B	20	INJ1	Injection capture	12/42 V compatible
2	2	P2	18	P2A	20	INJ2	Injection capture	12/42 V compatible
3	3	P2	21	P2B	37	INJ3	Injection capture	12/42 V compatible
4	4	P2	22	P2A	37	INJ4	Injection capture	12/42 V compatible
5	5	P2	25	P2B	5	INJ5	Injection capture	12/42 V compatible
6	6	P2	26	P2A	5	INJ6	Injection capture	12/42 V compatible
7	7	P1	27	P1B	38	INJ7 (PWM_IN7)	Injection capture (shared with PWM input)	12/42 V compatible
8	8	P1	28	P1A	38	INJ8 (PWM_IN8)	Injection capture (shared with PWM input)	12/42 V compatible

The following table shows the I/O pins for group 2:

RTI Port/ Channel	RTLib Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage Range
1	1	P2	19	P2B	4	IGN1	Ignition capture	12/42 V compatible
2	2	P2	20	P2A	4	IGN2	Ignition capture	12/42 V compatible
3	3	P2	23	P2B	21	IGN3	Ignition capture	12/42 V compatible
4	4	P2	24	P2A	21	IGN4	Ignition capture	12/42 V compatible
5	5	P2	27	P2B	38	IGN5	Ignition capture	12/42 V compatible
6	6	P2	28	P2A	38	IGN6	Ignition capture	12/42 V compatible
7	7	P1	34	P1B	39	AUXCAP1	Auxiliary capture	12/42 V compatible
8	8	P1	36	P1A	23	AUXCAP2	Auxiliary capture	12/42 V compatible

Related topics

Basics

Event Capture Windows	88
Spark Event Capture Unit	83

References

[Digital Inputs \(PHS Bus System Hardware Reference !\[\]\(003082e50e3009141f59bd5df831749f_img.jpg\)\)](#)
[DS2211APU_INJ_Bx_Gy \(DS2211 RTI Reference !\[\]\(f439ede8735757e3190eab35e168f1de_img.jpg\)\)](#)
[DS2211DIO_SETUP_Bx \(DS2211 RTI Reference !\[\]\(f5c165e0bd35116675db6686a30b1fea_img.jpg\)\)](#)
[Injection Pulse Position and Fuel Amount Measurement \(DS2211 RTLib Reference !\[\]\(8eeb5cc52b4d0f9a4ccc73b2d771855c_img.jpg\)\)](#)

Features of the Slave DSP

Introduction	The slave DSP is used for wheel speed sensor simulation and knock sensor simulation. It provides a DAC unit, has access to the Bit I/O unit and can be triggered by interrupts.
Where to go from here	<div>Information in this section<div><div>Basics of the Slave DSP..... 116</div><div>The DS2211 is equipped with a TMS320VC33 DSP from Texas Instruments as the slave DSP.</div><div>Knock Sensor Simulation..... 117</div><div>Knock sensor simulation is performed by a ready-to-use application implemented on the slave DSP.</div><div>Wheel Speed Sensor Simulation..... 119</div><div>Wheel speed sensor simulation is performed by a ready-to-use application implemented on the slave DSP.</div><div>DSP DAC Unit..... 121</div><div>The slave DSP contains eight digital analog converters. Four digital analog converters are used for knock sensor simulation or wheel speed sensor simulation. The other four channels can be used for user output.</div><div>DSP Bit I/O Unit and Capture Input Access..... 122</div><div>The slave DSP has access to the 16 digital inputs and the 16 digital outputs of the bit I/O unit.</div><div>Slave DSP Interrupts..... 123</div><div>The slave DSP receives APU update interrupts, DPMEM interrupts, and edge detection interrupts.</div></div></div>

Basics of the Slave DSP

Introduction	The DS2211 is equipped with a TMS320VC33 DSP from Texas Instruments, running at 150 MHz as the slave DSP.
Characteristics	The slave DSP is used to generate knock sensor signals on 4 analog outputs or wheel speed sensor signals on 4 analog outputs. Both functions are performed by ready-to-use applications that are controlled by slave DSP access functions running on the master processor board. Additionally, the slave DSP contains 4 analog outputs and has access to 16 digital inputs and 16 digital outputs from the bit I/O unit.
Output circuits	The resolution of the 8 analog outputs is 12 bit. For information on the I/O circuits, refer to Digital Outputs (PHS Bus System Hardware Reference [1]) .
User applications	As an alternative, you can program your own applications to generate user-defined signals. The available functions and macros for slave DSP programming are explained in Slave DSP Functions and Macros (DS2211 RTLib Reference [1]) .
Download slave applications	The slave DSP applications are downloaded to the master processor board together with the master application, and then transferred to the DS2211 slave DSP's memory.
Engine position bus	The slave DSP is connected to the engine position bus of the angular processing unit and receives an interrupt when the engine position has been updated (every 1 μ s). The master processor board can interrupt the slave DSP by writing to a predefined location of the slave DSP's dual-port memory (DPMEM).
Serial interface	In addition to the analog outputs, the slave DSP provides a serial interface. This serial interface can be used only for connecting slave DSPs of the same family from other I/O boards, for example, a DS2302 Direct Digital Synthesis Board. For the location of the connector, refer to Serial Interface (PHS Bus System Hardware Reference [1]) .
TMS320VC33	For more information on the TMS320VC33 slave DSP, refer to the Texas Instruments web site at http://www.ti.com and search for "TMS320C33".

Related topics

References

[Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)\)](#)
[Serial Interface \(PHS Bus System Hardware Reference !\[\]\(d4257ae6a3e163e6d467b3eb87960fa1_img.jpg\)\)](#)
[Slave DSP Functions and Macros \(DS2211 RTLib Reference !\[\]\(37da042f270bb1ebdb248503fcdcdd43_img.jpg\)\)](#)

Knock Sensor Simulation

Introduction

Knock sensor simulation is performed by a ready-to-use application implemented on the slave DSP. The slave DSP provides 4 analog outputs to simulate 4 independent knock sensors.

Basics

Knock sensor simulation is triggered by the engine position, which is supplied by the engine position bus (see [APU Overview](#) on page 73). Like a knock sensor in a real engine that measures several cylinders, multiple knock signals can be generated for up to 8 cylinders in one engine cycle (0 ... 720°). Knock sensor signals are generated as cosine wave signals plus Gaussian noise. All parameters to determine the characteristics of the generated signal can be changed at run time by using RTI or RTLib functions.

Knock signal calculation

The parameters are passed to the slave DSP for knock sensor simulation and the knock signal $u(t)$ is generated according to the formula

$$u(t) = a \cdot e^{-d 2\pi f t} \cdot \cos(2\pi f t) + Noise$$

Where

a	is the amplitude
$e^{-d 2\pi f t}$	is the envelope
d	is the damping
f	is the frequency
t	is the time

Knock signal parameters

For the knock signals to be generated, you have to specify the following parameters for each cylinder:

Start angle Start position of the cosine signal in degrees. For RTI blocks, the angle relates to top dead center (TDC), for RTLib functions, it relates to the absolute engine position.

Knock length The signal generation ends after this length (stated in degrees).

Amplitude Amplitude of the cosine signal in V_{pp}

Frequency Frequency of the cosine signal in Hz

Damping Damping factor of the cosine signal

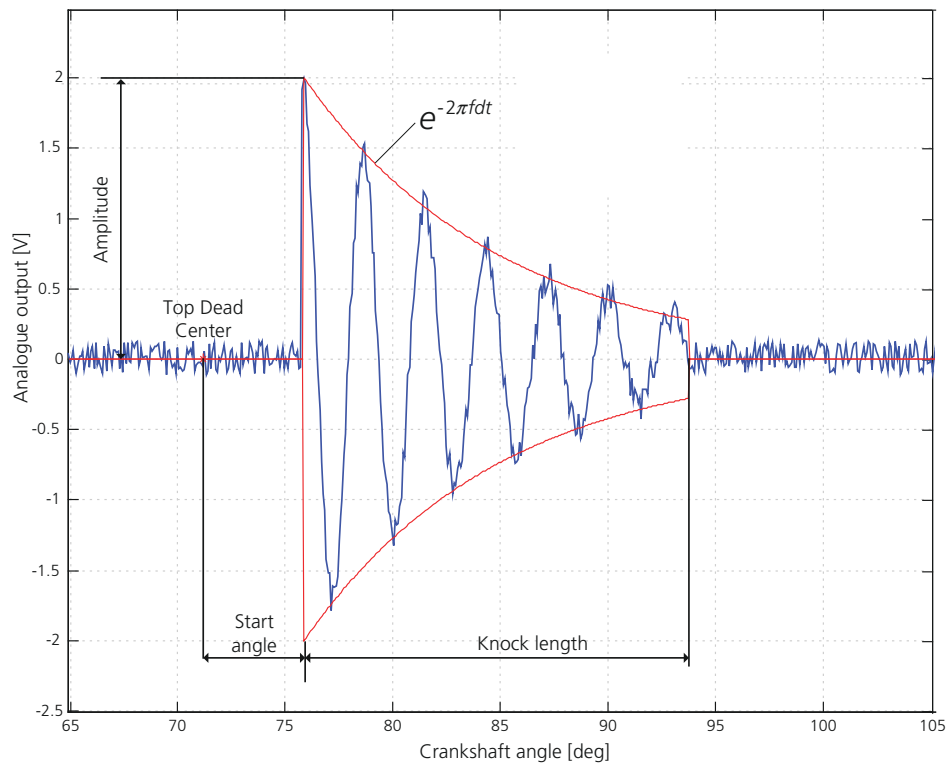
Knock number This parameter defines the maximum number of knock signals which are generated for each cylinder while the application is running. When the limit is reached, no further pulses are generated. To specify an infinity number of knock signals, set Knock number = 0 and Knock rate = 1.

Knock rate This parameter defines the number of engine cycles after which knock signal generation will start again for the given cylinder.

The additional noise signal can be selected independently for knock sensors 1, 2, 3, and 4:

Noise Amplitude of the noise in V_{pp}

The following illustration shows the most important parameters of a knock signal with the optional Gaussian noise.



The output signals are fed through output transformers with a maximum physical amplitude of $40 V_{pp}$. The outputs can be enabled or disabled.

Transformers do not transmit a DC voltage. If the signal has a DC voltage component, you can bypass the transformer via jumper settings. For details, refer to [Jumper Settings J1 ... J7 \(PHS Bus System Hardware Reference\)](#).

Note

You cannot use wheel speed sensor simulation and knock sensor simulation at the same time.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the corresponding measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(9dfdaff1d86ba3c1f8353b4d1b61b8c5_img.jpg\)\)](#).

I/O mapping

The following table shows the mapping of the channel numbers to the related I/O pins of the I/O connector P2, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 197.

Channel Number	Signal	Connector Pin	Sub-D Pin	Description	Voltage Range/Output Current
1	VC33-Waveform 0+ VC33-Waveform 0–	P2 63 P2 65	P2B 44 P2B 28	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 0+ – VC33-Waveform 0–) = ± 20 V
2	VC33-Waveform 1+ VC33-Waveform 1–	P2 67 P2 69	P2B 12 P2B 45	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 1+ – VC33-Waveform 1–) = ± 20 V
3	VC33-Waveform 2+ VC33-Waveform 2–	P2 71 P2 73	P2B 29 P2B 13	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 2+ – VC33-Waveform 2–) = ± 20 V
4	VC33-Waveform 3+ VC33-Waveform 3–	P2 75 P2 77	P2B 46 P2B 30	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 3+ – VC33-Waveform 3–) = ± 20 V

Related topics**References**

[DS2211SL_KNSG_Bx_Cy \(DS2211 RTI Reference !\[\]\(d0262bbe9d2356661a2e89321dfcc781_img.jpg\)\)](#)

[DS2211VARSL_KNSG_Bx_Cy \(DS2211 RTI Reference !\[\]\(51514032c8ca341817228f39f1307b05_img.jpg\)\)](#)

[Knock Sensor Simulation \(DS2211 RTLib Reference !\[\]\(c444627dab9fee9a1550c053ffaaaae2_img.jpg\)\)](#)

[Transformer Outputs \(APU and Slave DSP\) \(PHS Bus System Hardware Reference !\[\]\(0d7ca0919e6c47bbd874bfa0189fe22e_img.jpg\)\)](#)

Wheel Speed Sensor Simulation

Characteristics

Wheel speed sensor simulation is performed by a ready-to-use application implemented on the slave DSP. You can use the 4 analog slave DSP outputs to generate up to 4 independent wheel speed sensor signals at the same time, one for each of the four wheels.

Wheel speed sensor signals are generated as sine wave signals plus an optional Gaussian noise. You can specify wheel speed, periods per revolution, sine amplitude, and noise amplitude.

Wheel speed signal calculation

The parameters are passed to the slave DSP for wheel speed simulation and the wheel speed signal $u(t)$ is generated according to the formula

$$u(t) = a \cdot \frac{f_{Wheel}}{500 \text{ Hz}} \cdot \sin(2\pi f_{Wheel} \cdot t) + \text{Noise} \quad f_{Wheel} < 500 \text{ Hz}$$

$$u(t) = a \cdot \sin(2\pi f_{Wheel} \cdot t) + \text{Noise} \quad f_{Wheel} > 500 \text{ Hz}$$

Where

a is the amplitude

f_{Wheel} is the wheel speed signal frequency which is calculated as follows:

$$f_{Wheel} = \text{speed} \cdot \text{teeth} \cdot 1[\text{min}]/60[\text{s}]$$


speed: Wheel speed

teeth: Number of wheel teeth

t is the time

Outputs

The output signals are fed through output transformers with a maximum physical amplitude of 40 V_{pp}.

Transformers do not transmit a DC voltage. If the signal has a DC voltage component, you can bypass the transformer via jumper settings, refer to [Jumper Settings J1 ... J7](#) (PHS Bus System Hardware Reference ).

The outputs can be enabled or disabled.

Note

- In the frequency range below 500 Hz, the application reduces the maximum output voltage so that is proportional to the frequency.
- The nominal operating range of the transformers is 500 Hz ... >20 kHz.
- You cannot use wheel speed sensor simulation and knock sensor simulation at the same time.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the corresponding measurement setup, refer to [Function Execution Times](#) (DS2211 RTLib Reference .

I/O mapping

The following table shows the mapping of the channel numbers to the related I/O pins of the I/O connector P2, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 197.

Channel Number	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range/Output Current
1	VC33-Waveform 0+ VC33-Waveform 0–	P2	63	P2B	44	Knock sensor / wheel speed sensor signal	(VC33-Waveform 0+ – VC33-Waveform 0–) = ± 20 V
2	VC33-Waveform 1+ VC33-Waveform 1–	P2	67	P2B	12	Knock sensor / wheel speed sensor signal	(VC33-Waveform 1+ – VC33-Waveform 1–) = ± 20 V
3	VC33-Waveform 2+ VC33-Waveform 2–	P2	71	P2B	29	Knock sensor / wheel speed sensor signal	(VC33-Waveform 2+ – VC33-Waveform 2–) = ± 20 V
4	VC33-Waveform 3+ VC33-Waveform 3–	P2	75	P2B	46	Knock sensor / wheel speed sensor signal	(VC33-Waveform 3+ – VC33-Waveform 3–) = ± 20 V

Related topics

References

[DS2211SL_WSSG_Bx_Cy \(DS2211 RTI Reference !\[\]\(c3d993ca47bfe2a953c700506ce31fa0_img.jpg\)\)](#)
[Wheel Speed Sensor Simulation \(DS2211 RTLib Reference !\[\]\(c468cde8f04e2e2a6ba3c2a373e05c45_img.jpg\)\)](#)

DSP DAC Unit

Characteristics

The slave DSP contains 8 digital analog converters with 12-bit resolution. 4 digital analog converters are used for knock sensor simulation or wheel speed sensor simulation. The other 4 channels (DSP_DAC5 ... DSP_DAC8) can be used for user output. The channels can be accessed only via RTLib functions.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the corresponding measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(faf942dc3e59ce8eb64b4ac481eca7e0_img.jpg\)\)](#).

I/O mapping

The following table shows the mapping of the D/A channel numbers to the related I/O pins of the I/O connector P3, as used in RTLib.

D/A Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range/Output Current
5	DSP_DAC4	P3	30	–	–	12-bit DAC/10 μ s	(DSP_DAC5 – GND) = ± 10 V; ± 5 mA
6	DSP_DAC5	P3	14	–	–	12-bit DAC/10 μ s	(DSP_DAC6 – GND) = ± 10 V; ± 5 mA
7	DSP_DAC6	P3	47	–	–	12-bit DAC/10 μ s	(DSP_DAC7 – GND) = ± 10 V; ± 5 mA
8	DSP_DAC7	P3	31	–	–	12-bit DAC/10 μ s	(DSP_DAC8 – GND) = ± 10 V; ± 5 mA

Related topics**Basics**

Knock Sensor Simulation.....	117
Wheel Speed Sensor Simulation.....	119

DSP Bit I/O Unit and Capture Input Access

Characteristics

The slave DSP has access to the 16 digital inputs and the 16 digital outputs of the bit I/O unit.

Digital inputs The bit I/O unit and the slave DSP read the digital inputs at the same time.

Digital outputs The bit I/O unit and the slave DSP set the state of the digital outputs using an OR operation.

For a description of the characteristics, refer to [Bit I/O Unit](#) on page 25.

Capture inputs The slave DSP has access to the capture inputs (IGN1 ... IGN6, AUXCAP1, AUXCAP2, INJ1 ... INJ8). You can read the state of the complex comparator outputs. An edge on one or more comparator output lines may request an interrupt on the slave DSP. The settings of the capture windows and the capture modes takes effect in the same way as on the DS2211 master I/O.

The DSP bit I/O unit an capture input access can only be used with RTLib functions. Refer to [Digital I/O via DS2211 I/O Unit \(DS2211 RTLib Reference !\[\]\(6a9b39b98eb945faa14c645ec99e4eaa_img.jpg\)](#)) and [Capture Input Access Functions \(DS2211 RTLib Reference !\[\]\(182077db5bac9ff62bf376fe37ffa951_img.jpg\)](#)).



Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the corresponding measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(e3275251d0893157c3584e20c81dc3ba_img.jpg\)](#)).

Related topics**Basics**

Bit I/O Unit.....	25
-------------------	----

References

Capture Input Access Functions (DS2211 RTLib Reference )
Digital I/O via DS2211 I/O Unit (DS2211 RTLib Reference )

Slave DSP Interrupts

Introduction

The slave DSP receives 4 different interrupts.

APU update interrupt Two interrupts are issued by the angular processing unit when the engine position has been updated (one interrupt every 250 ns, another interrupt every 1 μ s). You can use this interrupt to synchronize the slave DSP to the angular processing unit.

DPMEM interrupt The master processor can request slave DSP interrupts by writing to a predefined location of the DPMEM. The interrupt is acknowledged by the slave by reading this value.

Edge detection interrupt The edge detection interrupt is triggered when an edge has been detected at the outputs of the 16 DS2211 complex comparators.

Interrupt sources

The following interrupt sources are available:

Interrupt Line	Interrupt Source
IRQ0	APU update interrupt (every 1 μ s)
IRQ1	DPMEM interrupt
IRQ2	APU update interrupt (every 250 ns)
IRQ3	Edge detection interrupt

CAN Support

Introduction

The following topics provide all the information required for working with dSPACE CAN boards.

Where to go from here

Information in this section

- [Setting Up a CAN Controller.....](#) 126
Explains how to set up a CAN controller to use a dSPACE board with CAN bus interface.
- [Using the RTI CAN MultiMessage Blockset.....](#) 140
Provides information on using the RTI CAN MultiMessage Blockset.
- [CAN Signal Mapping.....](#) 163
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.....](#) 211
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..... 126](#)

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

[CAN Transceiver Types..... 128](#)

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

[DS2211: Selecting the CAN Controller Frequency..... 132](#)

The board's CAN controller supports the several maximum clock frequency.

[Defining CAN Messages..... 133](#)

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

[Implementing a CAN Interrupt..... 134](#)

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..... 135](#)

RTI CAN Blockset provides two concepts for receiving CAN messages.

[Removing a CAN Controller \(Go Bus Off\)..... 137](#)

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

[Getting CAN Status Information..... 137](#)

You can use the Error Management Logic (EML) of a CAN controller to get error and status information on the CAN bus and the controller.

Initializing the CAN Controller

Introduction

The CAN controller performs serial communication according to the CAN protocol. You can take control of or communicate with other members of a CAN bus via the controller. This means you must configure the CAN controller — called the CAN channel — according to the application.

Standard configuration

You must specify the baud rate for the CAN application and the sample mode:

Sample Mode	Description
1-sample mode	(supported by all dSPACE CAN boards) The controller samples a bit once to determine if it is dominant or recessive.
3-sample mode	(supported by the DS4302 only) The controller samples a bit three times and uses the majority to determine if it is dominant or recessive.

The required bit timing parameters are automatically calculated by the dSPACE CAN software.

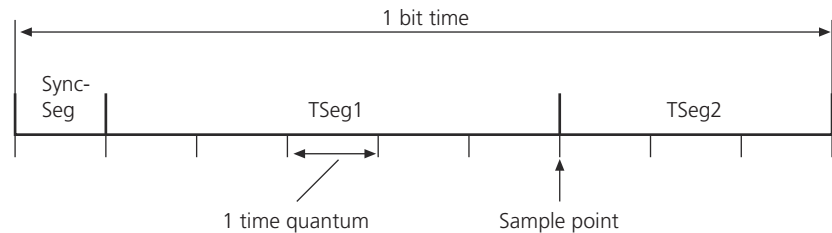
Advanced configuration (bit timing parameters)

The bits of a CAN message are transmitted in consecutive bit times. According to the CAN specification, a bit time consists of two programmable time segments and a synchronization segment:

TSeg1 Timing segment 1. The time before the sample point.

TSeg2 Timing segment 2. The time after the sample point.

SyncSeg Used to synchronize the various bus members (nodes).



The following parameters are also part of the advanced configuration:

SP Sample point. Defines the point in time at which the bus voltage level (CAN-H, CAN-L) is read and interpreted as a bit value.

SJW Synchronization jump width. Defines how far the CAN controller can shift the location of the sample point to synchronize itself to the other bus members.

BRP Baud rate prescaler value. The BRP defines the length of one time quantum.

SMPL Sample mode. Either 1-sample or 3-sample mode. Applicable to the DS4302 only.

Except for the SyncSeg parameter, you must define all these parameters via the values of the bit timing registers (BTR0, BTR1), located on the CAN controller.

Note

Setting up bit timing parameters requires advanced knowledge of the CAN controller hardware and the CAN bus hardware.

RTI support

You initialize a CAN controller with the RTICAN CONTROLLER SETUP block.

Refer to [RTICAN CONTROLLER SETUP \(RTI CAN Blockset Reference !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)\)](#).

Related topics**References**

[RTICAN CONTROLLER SETUP \(RTI CAN Blockset Reference !\[\]\(e8fb589d58dad1692debababa5e928b6_img.jpg\)\)](#)

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<xxxx>_ISO11898_6_SST) before you build the model. You cannot enable automatic wake-up during run time.
- If the transceiver is in *power on / listen only* mode, the CAN controller does not send an acknowledge message to the transmitter. The transmitter therefore continues to send the message until it receives the acknowledge signal. This might result in a task overrun if an RX interrupt is configured for the CAN controller.
- If the transceiver is in *power on / listen only* mode, it is not able to send CAN messages. Automatic wake-up is not possible if the transceiver is in *power on / listen only* mode. Because no message is sent on the CAN bus by the transceiver in *power on / listen only* mode, CAN arbitration fails. The CAN controller changes to the BUS OFF state. It is not possible to set the BUS state automatically to BUS ON via an interrupt, because the reason for the BUS OFF state still remains. You must set the CAN controller to BUS ON after you have switched the transceiver state to normal, standby, or sleep mode.

RS485 transceiver

The RS485 transceiver supports baud rates of up to 500 kD. It is often used in the automotive industry. A CAN bus using this transceiver can connect up to 25 CAN nodes.

Termination To terminate the CAN bus lines, a 120-Ω resistor must be used at both ends of the CAN bus.


C252 fault-tolerant transceiver

The C252 fault-tolerant transceiver supports baud rates of up to 125 kD. Its main feature is on-chip error management, which allows the CAN bus to continue operating even if errors such as short circuits between the bus lines occur.

When this transceiver is used, the CAN bus can interconnect nodes that are widely distributed. You can switch the C252 transceiver between sleep and normal (awake) mode.

Termination There are two ways to terminate the CAN bus lines: Use a 10 k 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 !\[\]\(e474458956c9a37fbf9586ddb60a7fa1_img.jpg\)\)](#).

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 !\[\]\(21199eb166cc97331a0c54c649195dcc_img.jpg\)](#)).

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 !\[\]\(cdf2842d82858164c68c92720a337fb9_img.jpg\)](#)) 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.

DS2211: Selecting the CAN Controller Frequency

Introduction

Depending on the version of your DS2211, the board's CAN controller supports the following maximum clock frequency:

Board Version	Frequency
Up to 4.10	24 MHz
4.10 and higher	36 MHz

To get the board version of your DS2211, open the board's Properties dialog in the dSPACE experiment software.

Highest possible frequency automatically selected

The real-time application built with RTI CAN Blockset is compatible with both board 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 !\[\]\(758ebdf4629c903da74c2e079717ae32_img.jpg\)](#)).

Related topics**References**

[Advanced Configuration Dialog \(RTICAN CONTROLLER SETUP\) \(RTI CAN Blockset Reference !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)\)](#)

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 !\[\]\(799877f5c2f906134441300079881630_img.jpg\)\)](#).

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 )


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 )

Related topics**References**

[RTICAN Interrupt](#) (RTI CAN Blockset Reference )

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 !\[\]\(666e09182d4cd268646ea700ea60dcdf_img.jpg\)](#))

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 !\[\]\(e40bb48ad1470e3a14017c64c5673877_img.jpg\)](#)).
- To restart data transfer, use the RTICAN Bus Off Recovery block. Refer to [RTICAN Bus Off Recovery \(RTI CAN Blockset Reference !\[\]\(de28875f44a359ca6d30bbb1d9f6cdbd_img.jpg\)](#)).

Related topics**References**

[RTICAN Bus Off Recovery \(RTI CAN Blockset Reference !\[\]\(4b7a79268f6ba26c1471d4232fffa85a_img.jpg\)](#))
[RTICAN Go Bus Off \(RTI CAN Blockset Reference !\[\]\(87d978583253c9bde1db2d6dfafe8de0_img.jpg\)](#))

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 \(DS2211 RLib Reference !\[\]\(4729e517bc6a7cd81c8025b9646574fb_img.jpg\)\)](#)
[RTICAN Status \(RTI CAN Blockset Reference !\[\]\(90a2fb2f2c617b26262139ae4159c0a0_img.jpg\)\)](#)

Using the RTI CAN MultiMessage Blockset

Introduction

The RTI CAN MultiMessage Blockset is a Simulink blockset for efficient and dynamic handling of complex CAN setups in hardware-in-the-loop (HIL) applications.

Where to go from here

Information in this section

[Basics on the RTI CAN MultiMessage Blockset](#)..... 140

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

[Basics on Working with CAN FD](#)..... 145

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](#)..... 150

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](#)..... 156

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

[Manipulating Signals to be Transmitted](#)..... 159

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 145.

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 !\[\]\(73002692dd5e7a64e60946be3158e719_img.jpg\)](#)).

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 1341 659 1371" data-label="Section-Header">Note</div> <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 <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

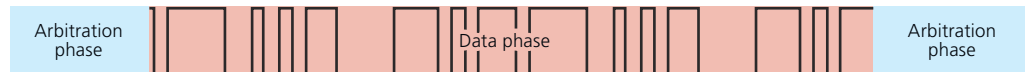
CAN FD stands for *CAN with Flexible Data Rate*. The CAN FD protocol is based on the CAN protocol as specified in ISO 11898-1. Compared with the classic CAN protocol, CAN FD comes with an increased bandwidth for the serial communication. The improvement is based on two factors:

- 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.

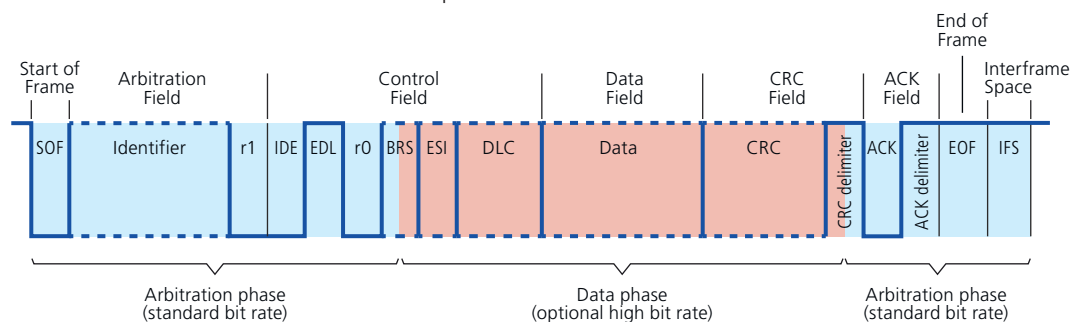
The RTI CAN MultiMessage Blockset supports both CAN FD protocols.

CAN FD message characteristics

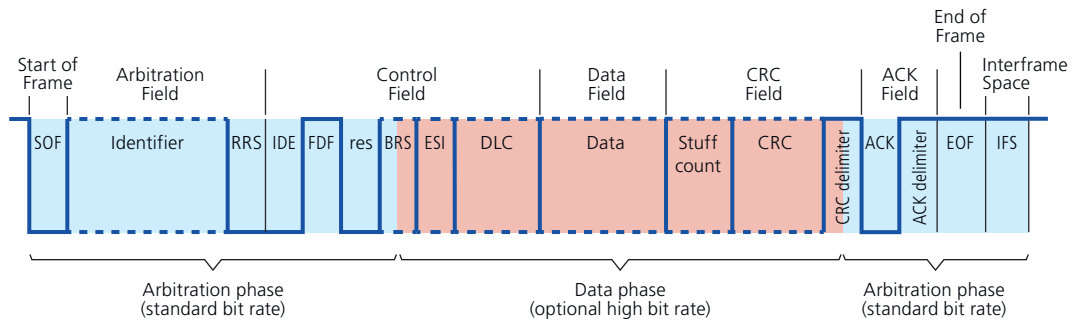
In principle, CAN FD messages and CAN messages consist of the same elements and have the same message structure.

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 !\[\]\(223f1a84e0bc2cacb9c165f716817dcc_img.jpg\)\)](#)
- MicroAutoBox II: [DS4342 Connections in Different Topologies \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(c437123967ec19fa50ef7951237304ba_img.jpg\)\)](#)

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 !\[\]\(bd3b31712ad9bab5a241210fa6925cdd_img.jpg\)\)](#)

Basics on Working with a J1939-Compliant DBC File

Introduction

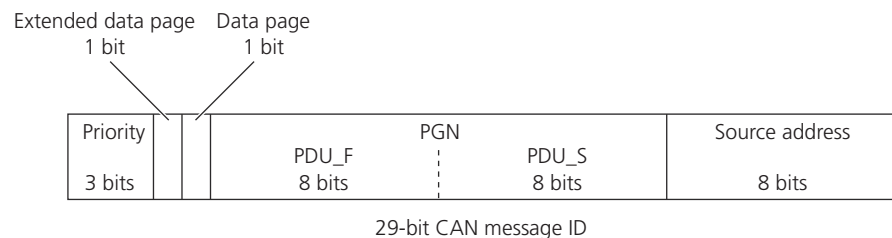
J1939 is a vehicle CAN bus standard defined by the Society of Automotive Engineers (SAE). It is used for communication in heavy-duty vehicles, for example, for communication between a tractor and its trailer.


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 $\text{PDU_F} \geq 240$ (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.

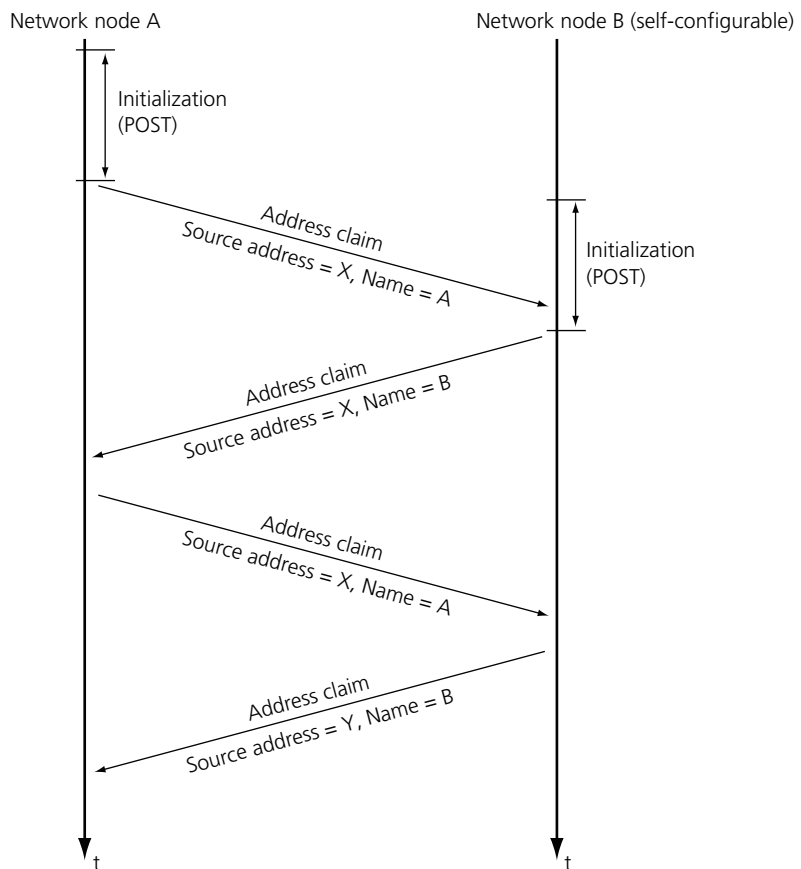
Name The J1939 standard defines a 64-bit *name* to identify each network node. The name indicates the main function of the network node with the associated address and provides information on the manufacturer.

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.

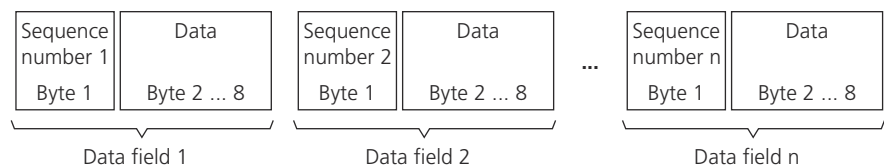
Note

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 !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)](#))

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 !\[\]\(aa53ad6fea213b8b2226d3077e30533a_img.jpg\)](#)). 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 !\[\]\(feabb98897b440bc8695a03336a6e2df_img.jpg\)\)](#)

Manipulating Signals to be Transmitted

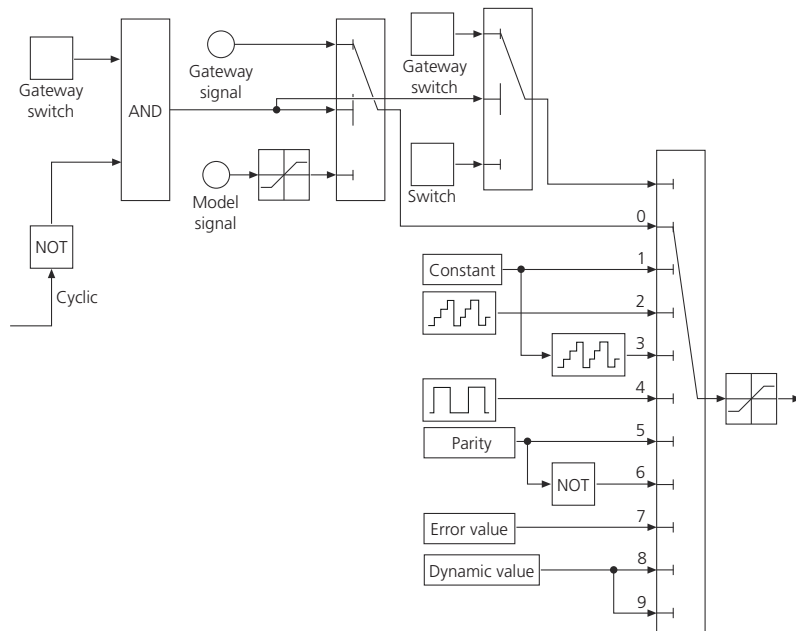
Introduction

All the signals of all the RX and TX messages (see [Defining RX messages and TX messages](#) on page 157) 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.



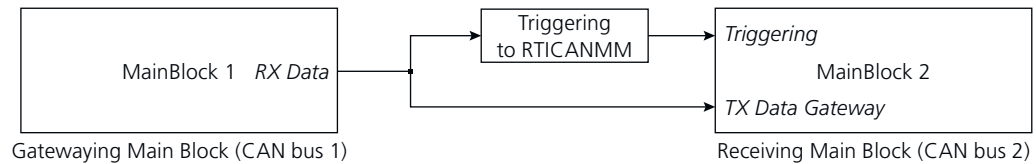
You can switch between these options in ControlDesk.

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 !\[\]\(c444627dab9fee9a1550c053ffaaaae2_img.jpg\)\)](#).

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 gateway messages and their signals to MainBlock2. Specifying gateway signals for MainBlock2 adds a TX Data Gateway input to it. The specified gateway signals are transmitted via CAN bus 2 with the signal values received from MainBlock1 when triggered by the messages received from MainBlock1. You therefore have to specify triggered message transmission for the messages of the gateway signals on the [Message Cyclic Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference). In addition, you can enable signal value switching for gateway signals during run time, for example, to transmit the signal constant value instead of the gateway value.

Note

Implementing gateway signals at least doubles the number of block inputs and therefore reduces performance. If you want to exchange messages between CAN controllers and do not want to perform signal manipulation, you should use the RTICANMM Gateway block instead.

You can specify gateway signals on the pages located in the [Gateway Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference).

Toggle signal A 1-bit signal that can be used, for example, to indicate whether CAN messages are transmitted. If a CAN message is transmitted, the toggle signal value alternates between 0 and 1. Otherwise, the toggle signal value remains constant.

You can specify toggle signals on the [Toggle Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference).

Parity signal A signal that a parity bit is appended to. You can specify one or more signals of a TX message as parity signals. A parity bit is calculated for the specified signals according to whether even or odd parity is selected. The bit is appended to the signal and the TX message is transmitted with the parity signal.

You can specify parity signals on the [Parity Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).

Counter signal A signal of a TX message that is used to check for correct message transmission or to trigger the transmission of signals in a message.

- **Behavior of counter signals**

The value of a counter signal changes with every message transmission. You can specify the counter start, step, and stop values, etc. Each time the counter reaches the stop value, it turns around to the counter start value.

- **Use of counter signals**

You can specify counter signals to check for correct transmission of a message or trigger the transmission of signals in a message.

- *Checking correct message transmission:* The receiver of a message expects a certain counter signal value. For example, if the transmission of a message was stopped for two transmissions, the expected counter signal value and the real counter signal value can differ, which indicates an error. Counter signals used in this way are often called alive counters.

- *Triggering the transmission of signals:* You can trigger the transmission of signals if you specify a mode signal as a counter signal. The signal value of the mode signal triggers the transmission of mode-dependent signals. Because the counter signal value changes with every message transmission, this triggers the transmission of the mode-dependent signals. By using counter signals in this way, you can work with signals which use the same bytes of a message. Counter signals used in this way are often called mode counters.

- **Using counter signals in ControlDesk**

In ControlDesk, you can transmit the signal's constant, counter, or increment counter value. The increment counter value is the counter value incremented by the signal's constant value.

You can specify counter signals on the [Counter Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).

Error value A static signal value that indicates an error. You can specify an error value for each signal to be transmitted. Alternatively, error values can be defined in a database file. In ControlDesk, you can switch to transmit the signal's error value, constant value, etc. However, you cannot change the error value during run time. In ControlDesk, you can use a Variable Array (MultiState LED value cell type) to indicate if the error value is transmitted.

You can specify error values on the [Signal Errors Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).

Dynamic value A signal value that is transmitted for a defined number of times.

- *Behavior of dynamic values:* You can specify to use a dynamic value for each signal to be transmitted. In ControlDesk, you can specify to transmit the signal's constant value or dynamic value. If you switch to the dynamic value, it is transmitted for a defined number of times (countdown value). Then signal manipulation automatically switches back to the signal manipulation option used before the dynamic value.

- *Example of using dynamic values:* Suppose you specify a dynamic value of 8 and a countdown value of 3. If you switch to the dynamic value of the signal, the signal value 8 is sent the next 3 times the TX message is transmitted. Then the signal manipulation option is reset.

You can specify dynamic values on the pages located in the [Dynamic Signal Page \(RTICANMM MainBlock\)](#) ([RTI CAN MultiMessage Blockset Reference](#) ).

CAN Signal Mapping

Introduction

Lists the CAN signals of the dSPACE real-time hardware and the mapping of these signals to RTI blocks and RTLlib functions.

CAN Signal Mapping


Introduction

The CAN subsystem of the DS2211 provides 2 independent CAN bus interfaces that meet the ISO/DIS 11898 specifications.

I/O signals

The following table lists the CAN signals of the DS2211 and the mapping of these signals to RTI blocks and RTLlib functions.

The table also provides the mapping of the I/O signals to the I/O pins on the DS2211 and on the Sub-D connectors (P1A, P1B, P2A, P2B).

Signal	Channel/Bit Numbers of Related RTI Blocks/RTLlib Functions				I/O Pin on ...	
	Related RTI Block(s)	Ch/Bit (RTI)	Related RTLlib Functions	Ch/Bit (RTLlib)	DS2211	Sub-D Conn.
CAN Support						
<ul style="list-style-type: none">▪ CANxL: CAN dominant low▪ CANxH: CAN dominant high▪ Electrical characteristics: see CAN Bus Interface (PHS Bus System Hardware Reference )						
CAN1L	<ul style="list-style-type: none">▪ RTICAN CONTROLLER SETUP▪ RTICANMM ControllerSetup	CAN1	Slave CAN Access Functions	CAN1	P2 83	P2B 31
CAN1H					P2 81	P2B 47
GND					P2 79, 85	P2B 14, 15
CAN2L		CAN2		CAN2	P2 84	P2A 31
CAN2H					P2 82	P2A 47
GND					P2 80, 86	P2A 14, 15

Interrupts of the DS2211

Introduction The DS2211 provides interrupts, which you can use in your Simulink model and your handcoded model.

Where to go from here

Information in this section

[Basics of DS2211 Interrupts.....](#) 165

Providing information on the available interrupts.

[Interrupt Handling.....](#) 167

Explaining how interrupts are handled with RTI or RTLib.

Information in other sections

[Slave DSP Interrupts.....](#) 123

The slave DSP receives APU update interrupts, DPMEM interrupts, and edge detection interrupts.

Basics of DS2211 Interrupts

Basics

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

The inputs of the DS2211's slave interrupt controller are connected to the serial interface, the angular processing unit (angle position interrupts, capture interrupts), the CAN controller, and the slave DSP (VC33) interrupts. All interrupts can be masked individually. A global interrupt enable/disable flag is also available.

Serial interface interrupt The serial interface uses one hardware interrupt. Subinterrupt handlers permit handling of all interrupts that the UART can generate.

Angle position interrupts Depending on the engine position (angle), the angular processing unit can generate angle position interrupts for up to 6 cylinders in any angle position. You can generate interrupts when the cylinder has passed the top dead center (TDC), for example. Several interrupt positions are possible for each cylinder. Use RTI (see [DS2211APU_INT_Bx_ly \(DS2211 RTI Reference\)](#)) or RTLib (see [ds2211_int_position_set \(DS2211 RTLib Reference\)](#)) functions to define the interrupt positions.

Capture interrupts (only RTLib) The angular processing unit can trigger capture interrupts according to capture events (leading and trailing edges of ignition and injection inputs). Use the RTLib function [ds2211_cap_interrupt_setup](#) to define the capture events (see [ds2211_cap_interrupt_setup \(DS2211 RTLib Reference\)](#)).

Slave CAN MC interrupt The CAN MC can request an interrupt to the PHS-bus master (the processor board) by writing to a predefined location in its dual-port memory (DPMEM). You can define subinterrupts for different message events or CAN bus events. For details, refer to [RTICAN Interrupt \(RTI CAN Blockset Reference\)](#).

Slave DSP (VC33) interrupts (only RTLib) You can trigger DS2211 interrupts IRQ0...IRQ5 from the slave DSP via RTLib macros. For details, refer to [Triggering Interrupts on the DS2211 \(DS2211 RTLib Reference\)](#).

Interrupt sources

The following table shows which interrupt sources are available and how the interrupt lines are shared. All these interrupts are combined by logical OR, so you must plan their use carefully.

Interrupt Line	Interrupt Source				
	Capture Interrupt	Angle Interrupt	VC33 Interrupt	CAN Interrupt	UART Interrupt
IRQ0	✓	✓	✓	—	—
IRQ1	—	✓	✓	—	—
IRQ2	—	✓	✓	—	—
IRQ3	—	✓	✓	—	—
IRQ4	—	✓	✓	—	—
IRQ5	—	✓	✓	—	—
IRQ6	—	—	—	✓	—
IRQ7	—	—	—	—	✓

Interrupt Handling

Introduction

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

Interrupt-driven subsystems in RTI

You can use interrupts or subinterrupts to trigger interrupt-driven subsystems in your Simulink model. When the task in one system has finished, the interrupt handler automatically creates an "End of interrupt" (EOI) message to indicate the state to other units.

Handcoded models

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

Use Scenarios

Introduction The following use scenarios demonstrates how you can simulate complex tasks with the DS2211.

Where to go from here	Information in this section
	Simulating Ionic Current Measurement..... 170 You can use the angular processing unit to simulate an ionic current measurement.
	Simulating Engine Variants..... 178 You can use the angular processing - variant unit to simulate engine variants.

Simulating Ionic Current Measurement

Introduction

This application example demonstrates the simulation of the output signals of ionic current measurement sensors for up to six cylinders.

Where to go from here

Information in this section

[Basics of Ionic Current Measurement..... 170](#)

Ionic current measurement is a method of monitoring the combustion in gasoline engines via the spark plugs.

[How to Simulate Ionic Current Measurement..... 172](#)

The application example demonstrates how an ionic current measurement can be simulated.

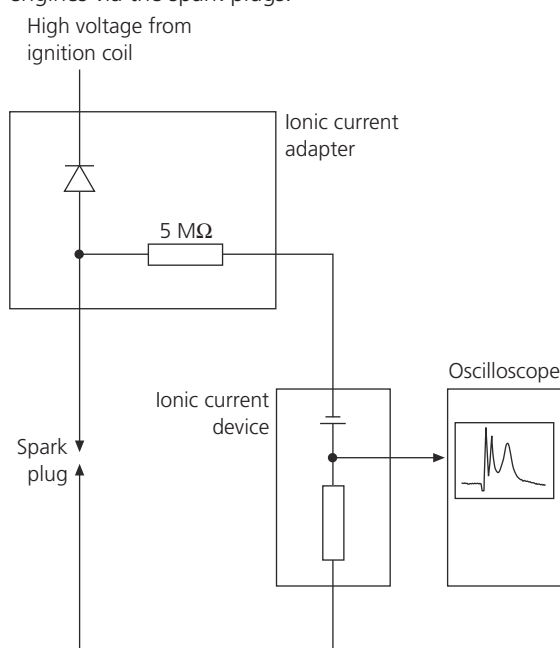
[Implementing the Simulation of Ionic Current Measurement..... 174](#)

This application example shows how the slave DSP of the DS2211 can be used to generate signals.

Basics of Ionic Current Measurement

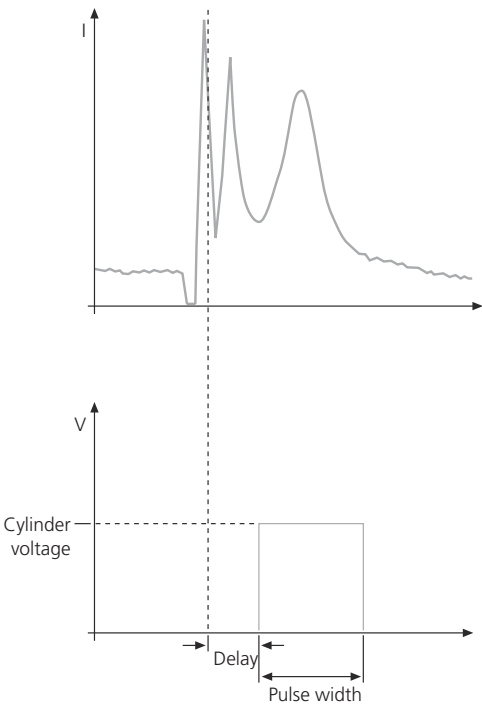
Introduction

Ionic current measurement is a method of monitoring the combustion in gasoline engines via the spark plugs.



The spark plugs are supplied by the high voltage from the ignition coil, which has a low DC voltage superimposed. The DC voltage causes a DC current to flow through the electrodes and the spark gap. In the spark gap, ions carry the current, the quantity of which depends on the state of the combustion and the pressure in the cylinder.

The following illustration shows the typical curve of the ionic current. If the measured curve differs from the ideal curve, combustion knocks or missing combustions are detected. The ionic current device reports this by supplying a DC voltage whose level describes the state of the combustion.



The state of the combustion is decoded using different voltage levels. For example:

Voltage Level	Meaning
0.00 ... 0.20 V	Error in the wiring (short to ground)
0.20 ... 1.06 V	Error in the ionic current ECU
1.14 ... 1.99 V	Combustion knock
2.07 ... 2.93 V	Combustion okay
3.01 ... 3.86 V	Combustion missing
3.94 ... 4.80 V	Reserved
4.80 ... 5 (12) V	Error in the wiring (short to battery voltage)

This application example simulates the report of the combustion state. It generates a pulse after an ignition. You can define all the parameters of the pulse: delay time after ignition, pulse width, and voltage level.

Related topics

HowTos

[How to Simulate Ionic Current Measurement.....](#) 172

How to Simulate Ionic Current Measurement

Objective

All the files needed for using the application example are installed with the Real-Time Interface. The application is ready to use for a PHS-bus-based system containing a DS2211. It is assumed that you know how to work with ControlDesk and how to use the DS2211 APU blocks, refer to [Angular Processing Unit \(DS2211 RTI Reference !\[\]\(c694a3ff3b077d76910920a6a1593ab4_img.jpg\)](#)).

Generating ignition signals

In this application demo, the DS2211APU_CAM_B1_C2 block generates a pulse pattern, which simulates the ignition signals for the spark event capture unit (DS2211APU_IGN_B1 block). The pin of the CAM2_DIG signal must be connected to pins IGN1 ... IGN6. The Simulink model detects only the relevant pulses for each cylinder, because the capture unit evaluates the signals within the defined event capture window.

Ionic current measurement

A controller measures the ionic current through a spark plug and generates a pulse whose voltage level represents the state of the combustion. In this application you can simulate such a pulse, starting with a delay after the rising or falling edge of the ignition pulse. The voltage level, pulse width, and delay time of the pulse can be defined for each cylinder. The pulse defines the result of the ionic current measurement.

Preconditions

You must have a PHS-bus-based system containing a DS2211 HIL I/O Board.

Method

To simulate ionic current measurement

- 1 Before starting the application, you must connect some pins of the DS2211 I/O connector. See the following table:

Output		Input		Description
Signal	Connector Pin	Signal	Connector Pin	
CAM1_DIG	P2, pin 47	ADC1	P1, pin 65	Measuring camshaft position channel 1
GND	P1, pin 99 or 100	ADC1	P1, pin 67	Differential input is connected to ground
CAM2_DIG	P2, pin 49	ADC2	P1, pin 66	Camshaft position channel 2 provides
		IGN1 ... IGN6	P2, pin 19, 20, 23, 34, 27, 28	the simulated ignition pulses

Output		Input		Description
Signal	Connector Pin	Signal	Connector Pin	
GND	P1, pin 99 or 100	$\overline{\text{ADC2}}$	P1, pin 71	Differential input is connected to ground
CRANK_DIG	P2, pin 45	ADC3	P1, pin 69	Measuring the crankshaft position
GND	P1, pin 99 or 100	$\overline{\text{ADC3}}$	P1, pin 68	Differential input is connected to ground
+12 V	P2, pin 91 or 93	VBAT	P1, pin 29 or 31	Battery voltage

The voltage level which represents the result of the ionic current measurement can be measured at four connector pins. You can define the channels to use in the Ionic Signal Measurement block:

Output Signal	Connector Pin
DSP_DAC4	P3, pin 30
DSP_DAC5	P3, pin 14
DSP_DAC6	P3, pin 47
DSP_DAC7	P3, pin 31

Tip

You can connect an oscilloscope to the pins to measure the voltage level.

- Open the Simulink model of the demo. It is in the RTI library of your DS1005 or DS1006 processor board: DS2211 – Demo – IONIZATION.
- If necessary, specify the block parameters for the standard RTI blocks. For information on the parameters, refer to the [Angular Processing Unit \(DS2211 RTI Reference\)](#).
- Specify the block parameters of the Ionic Signal Measurement block. This block has the following parameters:

Parameter	Description
Board number	Lets you select the DS2211 board number within the range 1 ... 16.
Ion. current channel for cylinder x	Lets you select the output channel, where you can measure the voltage level representing the result of the ionic current measurement within the range 1 ... 4 (corresponds to DSP_DAC4 ... DSP_DAC7).
Unit mode	Lets you select seconds or degrees as the unit. The selected unit is valid for the delay width and pulse width.
Edge mode	Lets you select the edge of the ignition signals which is evaluated (leading or trailing edge)
Idle voltage	Lets you specify the idle voltage within the range 0 ... 1.0 (corresponds to 0 ... 10 V)

- Double-click Open Experiment to start ControlDesk and load the experiment belonging to this application example.
- Start the simulation. Use ControlDesk to vary the parameters and watch the results.

Result The output signal of the Slave DSP (DSP_DACx) simulates the ionic current measurement.

Related topics

Basics

[Basics of Ionic Current Measurement.....](#) 170

Implementing the Simulation of Ionic Current Measurement

Introduction

The following instructions describe how to implement simulation of the ionic current measurement and how to use the slave DSP for generating signals within a Simulink model.

It is assumed that you are familiar with C programming, because programming the slave DSP is an essential part of the implementation.

Overview

The simulation of the ionic current measurement is implemented on the slave DSP of the DS2211. The slave DSP generates the pulse which represents the result of the ionic current measurement. The start time is triggered by the edge detection interrupt. This interrupt is triggered by the complex comparators when a rising or falling edge of an ignition pulse has been detected.

An S-function is used to download the slave application to the slave DSP and to provide the parameter values. The S-function is used in a Simulink model which simulates the camshaft and crankshaft signals for an ECU. The Simulink model measures the position of the ignitions.

Source files

The source files for this application are installed in
 <RCP_HIL_InstallationPath>\Demos\DS100x\RTI\DS2211\Ionization

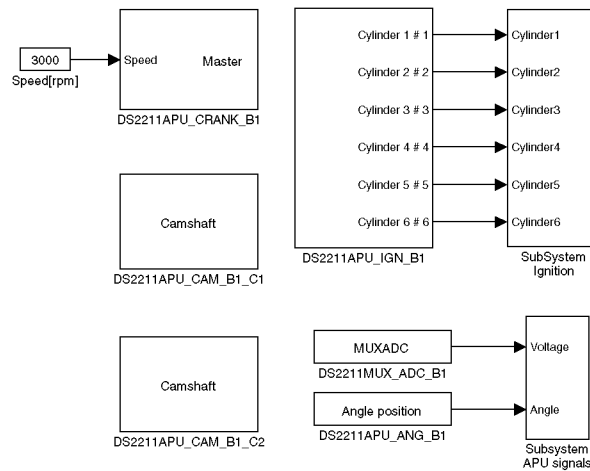
The following table introduces the most important files:

File	Description
demom_ds2211_ionization.mdl	RTI model, refer to Description of the Simulink Model on page 175
demom2211_ionization_sfunc.c	S-function for the RTI model, refer to Description of the S-Function on page 176
ionization.c	Slave application, refer to Description of the Slave Application on page 176

Description of the Simulink Model

The Simulink model generates the ignition pulses and calls the S-function which simulates the ionic current measurement.

The following illustration shows the part of the Simulink model that simulates ignition pulses.



The blocks that are used are described briefly below. For detailed information on the blocks, refer to the [Angular Processing Unit \(DS2211 RTI Reference\)](#).

DS2211APU_CRANK_B1 This block simulates a crankshaft signal generation. It defines the engine position which is the time base for all other APU blocks. The signal is based on a wave table. Click [Show Wavetables](#) to view the wave table.

DS2211APU_CAM_B1_C1 This block simulates a camshaft signal. The signal is based on a wave table.

DS2211APU_CAM_B1_C2 This block simulates the ignition pulses of all six cylinders. The signal is based on a wave table. It is connected to all six ignition channels via the I/O connector, so each ignition channel has the ignition pulse of all cylinders.

DS2211APU_IGN_B1 This block captures the ignition pulses. An event capture window is defined for each cylinder to capture only the corresponding ignition pulse.

Subsystem Ignition This subsystem grabs all the parameters of the ignition pulses so that they can be accessed by ControlDesk.

DS2211MUX_ADC_B1 This block reads the signals for crankshaft, camshaft, and ignition pulses. The signals are generated within this Simulink model (see above) and read in via the I/O connector (see [How to Simulate Ionic Current Measurement](#) on page 172).

DS2211APU_ANG_B1 This block provides the engine position.

Subsystem APU signals This subsystem grabs all the parameters of the APU signals so that they can be accessed by ControlDesk.

Ionic Signal Measurement This block calls the S-function for simulating the ionic current measurement, see below.

Description of the S-Function

As a DS2211 has a slave DSP, you have to handle two applications: the application running on the processor board (main application) and the application running on the slave DSP (slave application). A slave application cannot be modeled in Simulink. It must be programmed in C and handled by an S-function.

demom2211_ionization_sfunc is the S-function of this application example. The way how it handles the slave application is described below. For detailed information on the functions, see the remarks in the source code or refer to the [Slave DSP Functions and Macros \(DS2211 RTLib Reference !\[\]\(3d8c13c92b853674f749aac6fa869926_img.jpg\)](#))).

Include and declaration The slave application is included and declared in the body of the S-function. The included application data, was generated by the CL2211 utility (see [Loading Slave Applications \(DS2211 RTLib Reference !\[\]\(6605b201d6f14d9b3bcb8ab5f274d107_img.jpg\)](#))). For example,

```
#include "Slv2211_ionization.slc"
extern unsigned long ionization[]
```

Loading The slave application is loaded to the slave DSP during the start of model execution. This is done in the **mdlStart** function of the S-function. For example,

```
ds2211_slave_dsp_appl_load(DS2211_1_BASE, (Int32 *) &ionization);
```

Exchanging data The data is exchanged via the dual-port memory. To ensure data consistency, the semaphores are requested before the read or write operation and released afterwards.

Starting signal generation The signal generation of the slave DSP application must be started explicitly. This is done in the **mdlOutputs** function, for example,

```
ds2211_slave_dsp_signal_enable( (Int32) boardBase, enb1);
```

Description of the Slave Application

The slave application generates a pulse when a rising or falling edge of an ignition pulse was detected.

Process The ignition signals are connected to the inputs of the complex comparator. An ignition pulse causes a rising or falling edge at the complex comparator, which triggers an interrupt at the slave DSP (IRQ3 = edge detection interrupt). This interrupt starts the INT3 interrupt service routine (isr_int3). This function identifies the cylinder which triggered the interrupt and starts the signal generation for it. The signal (pulse) is generated in the timer interrupt service routine (isr_t0). It is output via the slave D/A converters (DSP_DAC4 ... DSP_DAC7), because these D/A converters can output constant voltage (they are not connected to a transformer).

Related topics

Basics

Basics of Ionic Current Measurement.....	170
--	-----

HowTos

How to Simulate Ionic Current Measurement.....	172
--	-----

Simulating Engine Variants

Introduction

The following use scenario shows how you can simulate engine variants without building and downloading the real-time application again. All relevant parameters can be changed in the real-time application via the experiment software.

Where to go from here

Information in this section

[Basics of Modeling Engine Variants](#)..... 178

The use scenario describes how two different engine variants can be simulated based on the same engine model.

[Implementing the Engine Model](#)..... 179

For the flexible handling, the engine model must be implemented using the VAR APU blockset. These blocks make the variant-specific parameters accessible for ControlDesk.

[Reconfiguring the Engine Real-Time Application](#)..... 183

If the model was implemented, you can reconfigure all relevant parameters for another engine variant.

Basics of Modeling Engine Variants

Introduction

Engine variants can be implemented with one real-time model if you use the VAR APU blockset. If you use the APU blockset, you have to build a real-time model for each engine variant.

User knowledge

Working with this blockset is aimed to advanced users. To be able to understand the procedure, you should have knowledge in building an engine model with RTI and working with ControlDesk.

Basics

If you want to simulate engine variants, you can implement the real-time model using the VAR APU blockset. The real-time application is built and downloaded to the processor board as usual. All parameters which are relevant for an engine variant, can be changed via ControlDesk. So it is unnecessary to build and download the real-time application again. For details, refer to [Basics on Simulating Engine Variants](#) on page 97.

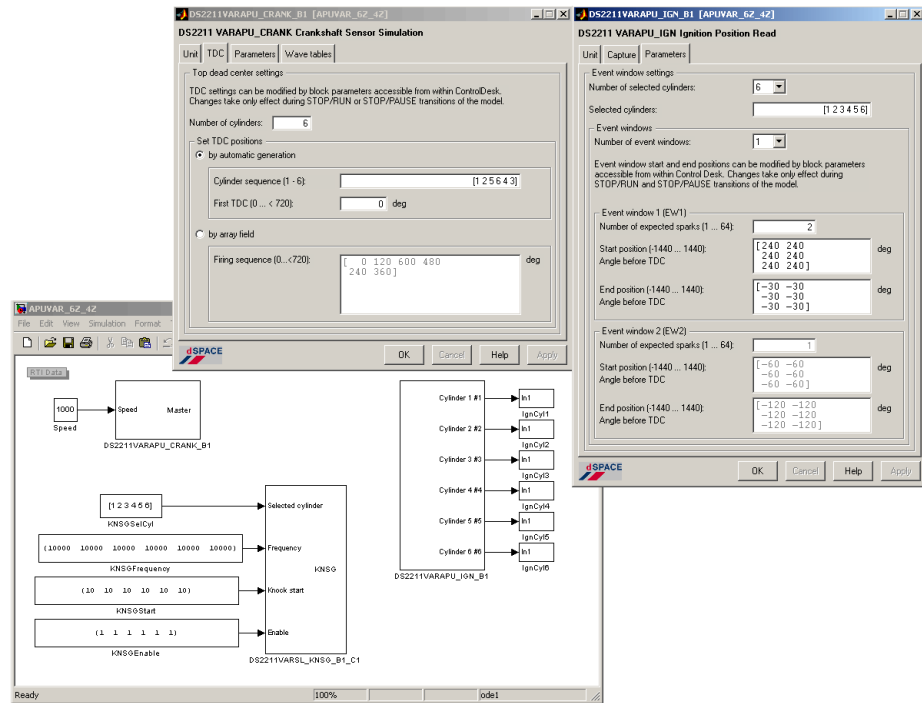
Use scenario	The following use scenario is a simple example, which demonstrates the flexible handling of engine variants. First a model for a 6-cylinder engine is built, downloaded, and simulated. After that, the relevant parameters are changed to simulate a 4-cylinder engine.
Sequence of engine variants	If your engine variant has a different number of cylinders, you must implement the engine variant with the highest number of cylinders first. The number or width of the block's in- and outports cannot be changed during run time. When an engine model with less cylinders is simulated, the remaining cylinders are not considered. In this use scenario, the 6-cylinder engine is simulated first and afterwards the 4-cylinder engine.
Two event capture windows	The number of event capture windows which you can use for your engine variants, depends on the number set in the engine model. If you want to use two event capture windows in at least one engine variant, you must select two event capture windows in the engine model. You can set the start position equal to the end position of the second event capture window in the engine variants that require only one event capture window.

Implementing the Engine Model

Introduction	The following describes the procedure to implement the engine model. This scenario use the ignition blocks, you can use the injection blocks in the same way.
---------------------	---

Creating the engine model

As usually, you create the engine model using the blocks of the VAR APU blockset.



DS2211VARAPU_CRANK_B1 In the DS2211VARAPU_CRANK_B1 block, the number of cylinder is set to 6. This is the highest number of cylinders for the real-time model. After building the real-time application, you can only reduce the number of cylinders later. As the first top dead center (TDC) is 0° and the cylinder sequence is [1 2 5 6 4 3], the firing angles are:

Cylinder sequence	1	2	5	6	4	3
Firing angle	0°	120°	240°	360°	480°	600°

DS2211VARAPU_IGN_B1 In the DS2211VARAPU_IGN_B1 block, all 6 cylinders are selected. The following values of the Parameter page are valid for each cylinder:

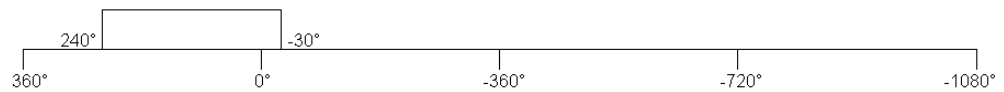
- Number of event windows: 1
- Number of expected sparks: 2
- Start position of the event window: 240°
- End position of the event window: -30°

Start and end position are related to the TDC of each cylinder. In relation to the TDC of the first cylinder, the event capture windows have the following positions

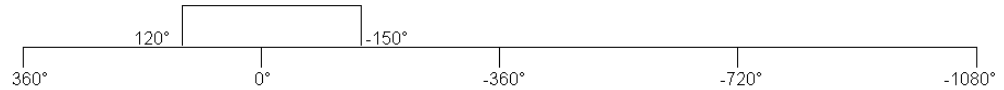
- 1. cylinder: 240° to -30°
- 2. cylinder: 120° to -150°
- 3. cylinder: -360° to -630°
- 4. cylinder: -240° to -510°
- 5. cylinder: 0° to -270°
- 6. cylinder: -120° to -390°

The following illustration shows the event capture windows.

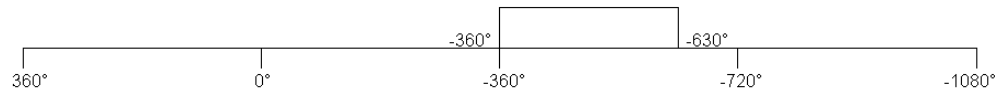
Event capture window of cylinder 1:



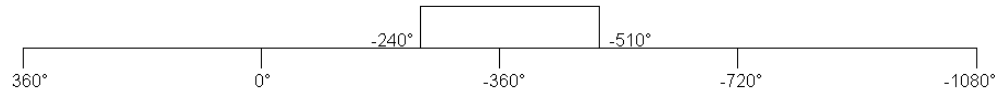
Event capture window of cylinder 2:



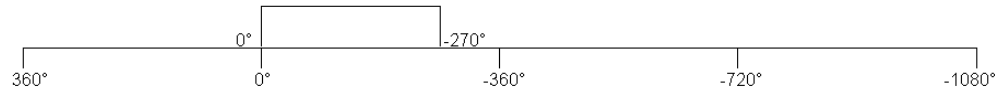
Event capture window of cylinder 3:



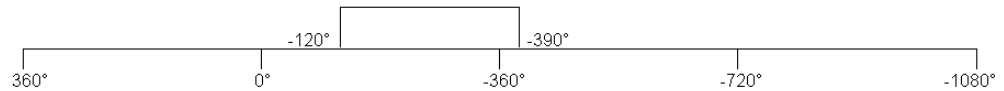
Event capture window of cylinder 4:



Event capture window of cylinder 5:



Event capture window of cylinder 6:



DS2211VARSL_KNSG_B1_C1 The DS2211VARSL_KNSG_B1_C1 block generates the knock signals. To be able to reduce the number of cylinders, the **Selected cylinders** and **Enable** parameters are specified via input port and connect them with Constant blocks. This makes it easy to change the number of cylinders later on with ControlDesk.

Downloading the engine model

When the engine model is created, you can build the real-time application and download it to the processor board. The procedure is the same as usual, whereas the generated SDF file contains additional parameters. These parameters are required to simulate other engine variants.

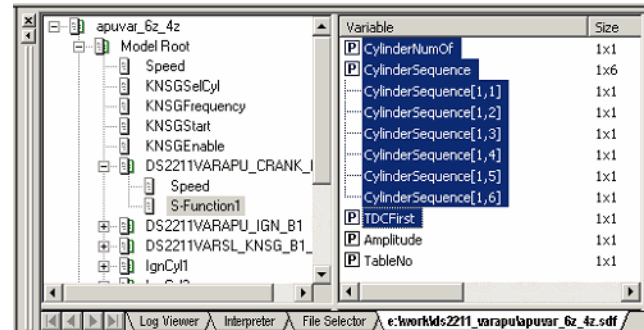
Creating an experiment

Open ControlDesk and create a project and experiment as usual.

Parameters for engine variants To switch to another engine variant, some parameters must be changed. The following list shows these parameters and their nodes of the SDF file:

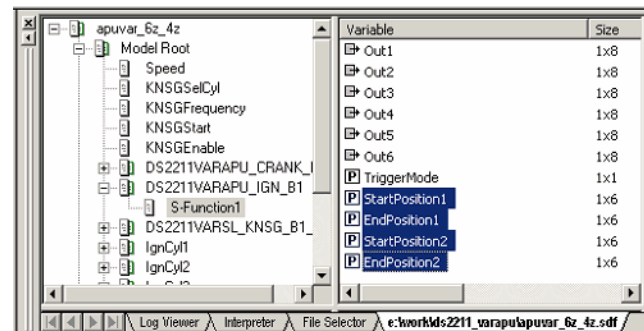
- \Model Root\DS2211VARAPU_CRANK_B1\S-Function1
 - CylinderNumOf
 - CylinderSequence[1,n]
 - TDCFirst

The following illustration shows the SDF file.



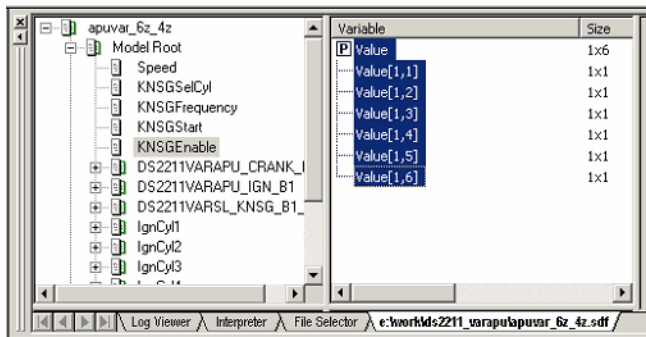
- \Model Root\DS2211VARAPU_IGN_B1\S-Function1
 - StartPosition1
 - EndPosition1
 - StartPosition2
 - EndPosition2

The following illustration shows the SDF file.



- \Model Root\KNSGEnable
 - Value[1,n]

The following illustration shows the SDF file.



To be able to change these parameters, add appropriate instruments to a layout and connect them to the parameters.

Simulation control The changed parameters become valid when the simulation state change from STOP to PAUSE or RUN. You should add an appropriate instrument to a layout to change the value of the `simState` parameter.

Simulating the 6-Cylinder Engine Model

You can simulate the engine model as usual.

When you want to change the engine variant, stop the simulation (`simState = 0`) first. Then you can reconfigure the real-time application. Refer to [Reconfiguring the Engine Real-Time Application](#) on page 183.

Reconfiguring the Engine Real-Time Application

Introduction

The following describes the procedure to reconfigure a real-time application. The real-time application was implemented for a 6-cylinder engine and should simulate a 4-cylinder engine.

Stopping the Simulation of the 6-Cylinder Engine Model

If the simulation runs, stop it first (`simState = 0`). The parameter values are changed only during the transition of the simulation state from STOP to PAUSE or RUN.

Configuring the 4-Cylinder Engine Model

If you have created a layout with a control for the engine parameters, it is easy to configure an engine variant.

Engine parameters Set the parameters to the following values:

- `CylinderNumOf`: 4
- `CylinderSequence[1,1]`: 1
- `CylinderSequence[1,2]`: 3
- `CylinderSequence[1,3]`: 4

- CylinderSequence[1,4]: 2
- CylinderSequence[1,5]: X (is not evaluated)
- CylinderSequence[1,6]: X (is not evaluated)

As the First TDC is 0° and the cylinder sequence is [1 3 4 2], the firing angles are:

Cylinder sequence	1	3	4	2
Firing angle	0°	180°	360°	540°

The firing sequence is:

- 1. cylinder: 0°
- 2. cylinder: 540°
- 3. cylinder: 180°
- 4. cylinder: 360°

Adapting the Capture Event Windows

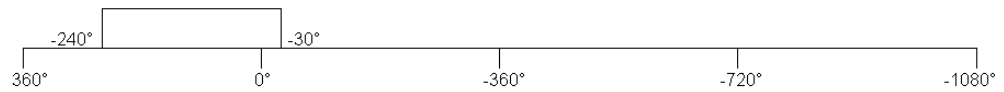
The event capture windows are automatically adapted to the new engine parameters. However, you can change the position of the event capture windows for the new 4-cylinder engine variant. Note that start and end positions of the event capture windows are set by a 1x6 vector. This is the dimension, which was set in the real-time model. The dimension cannot be changed in the real-time application. Ignore the redundant cylinder.

Start and end positions Without changing the values, the start position is 240° and the end position is -30° for each cylinder. The positions are related to the top dead center (TDC) of each cylinder. In relation to the TDC of the first cylinder, the event capture windows have the following positions:

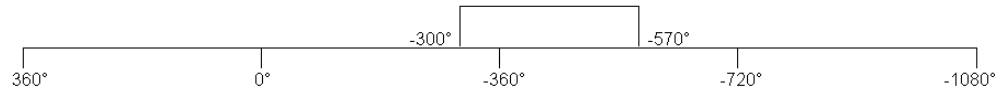
- 1. cylinder: 240° to -30°
- 2. cylinder: -300° to -570°
- 3. cylinder: -60° to -210°
- 4. cylinder: -120° to -390°
- 5. cylinder: 0° to -270° (not changed)
- 6. cylinder: -120° to -390° (not changed)

The following illustration shows the positions of the event capture windows.

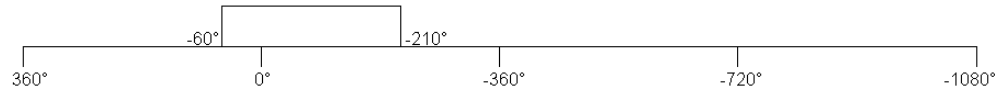
Event capture window of cylinder 1:



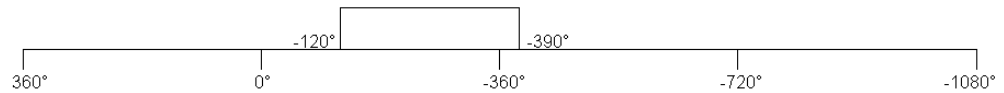
Event capture window of cylinder 2:



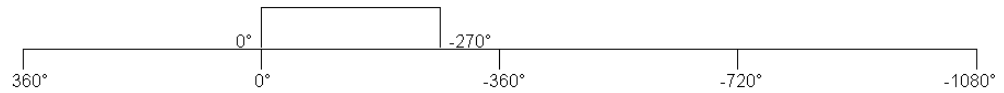
Event capture window of cylinder 3:



Event capture window of cylinder 4:



Event capture window of cylinder 5 (unchanged):



Event capture window of cylinder 6 (unchanged):



Adapting the Knock Signal

The knock signal generation is automatically adapted to the new engine parameters. However, you can change the parameters of the knock signal generation.

In this case, you must suppress the knock signal generation for the 5. and 6. cylinders. Set elements of the `KNSGEnable` vector to the following values:

- `KNSGEnable[1, 1]: 1`
- `KNSGEnable[1, 2]: 1`
- `KNSGEnable[1, 3]: 1`
- `KNSGEnable[1, 4]: 1`
- `KNSGEnable[1, 5]: 0`
- `KNSGEnable[1, 6]: 0`

Simulating the 4-Cylinder Engine

When all parameters are set, you can start the simulation of the 4-cylinder engine. The simulation is started when the value of `simState` parameter is changed (RUN: `simState = 2`, PAUSE: `simState = 1`).

Error handling If you set wrong or invalid values for the parameters, error messages are displayed. The simulation is started with the last valid parameters. You cannot prevent the simulation from starting.

Related topics**Basics**

[Implementing the Engine Model..... 179](#)

DS2211 Versus DS2210

Introduction	The DS2211 is compatible with the DS2210. You can migrate Simulink models and handcoded applications for DS2210 boards easily for the DS2211. It is also possible to work with real-time systems that have both types of I/O boards.
--------------	--

Where to go from here	Information in this section
	Comparing DS2210 and DS2211 Features..... 188 The DS2211 is the successor of the DS2210. Several features have been improved.
	rti221xConv Conversion Tool..... 188 rti221xConv is an M file which converts Simulink models containing DS2210 or DS2211 RTI blocks.
	How to Convert a Model with DS2210 Blocks to a Model with DS2211 Blocks..... 191 You can convert your Simulink models which have DS2210 RTI blocks that they can be used with a DS2211 board.
	How to Convert a Model with DS2211 Blocks to a Model with DS2210 Blocks..... 192 You can convert your Simulink models which have DS2211 RTI blocks that they can be used with a DS2210 board.
	How to Migrate Handcoded Applications for DS2211..... 193 You can convert handcoded applications for a DS2211 board to applications for a DS2211 board.
	Working with a Real-Time System Containing a DS2210 and a DS2211..... 193 A real-time system can contain DS2210 and DS2211 boards which can work independently of each other.

Comparing DS2210 and DS2211 Features

Introduction

The DS2211 is the successor of the DS2210. Several features have been improved, see the following table.

Feature	DS2210	DS2211
Voltage compatibility	12 V systems	12 V, 24 V or 2 voltage 12 V/42 V systems
Digital out channels	16	16
Digital in channels	16	8 + 16 shared channels
PWM out channels	6	9
PWM in channels	8	16 shared channels
Update mode of PWM channels	Asynchronous	Asynchronous or synchronous
Supply rails for out channels	1 (8 ... 18 V)	2 (5 ... 60 V)
Resistive out channels	6	10
Analog out channels	12 (12 bit, 0 ... 10 V)	20 (12 bit, 0 ... 10 V)
Analog in channels	16 (12 bit, 0 ... 20 V)	16 (14 bit, 0 ... 60 V)
CAN	2 CAN	2 CAN
Serial interface	RS232/RS422	RS232/RS422
APU resolution	0.09° (13 bit)	0.011° (16 bit)
Camshaft out channels	2 digital and analog	2 digital and analog 2 digital
Complex comparator	–	Complex comparator for ignition and injection signals
Event capture windows	1 per motor cycle	2 per motor cycle
Maximal configurable cylinders in APU blockset	18	96
Slave DSP	TMS320C31, 80 MHz	TMS320VC33, 150 MHz (4 additional DACs, access to bit I/O unit)

rti221xConv Conversion Tool

Introduction

rti221xConv is an M file which converts Simulink models containing DS2210 or DS2211 RTI blocks. The M file is located in the `<RCP_HIL_InstallationPath>\Matlab\RTI\M` folder. It searches and

replaces all DS2210 blocks by the corresponding DS2211 blocks and visa versa, including the blocks of the RTICAN blockset.

Syntax

It has the following syntax:

```
rti221xConv('ModelName','ConversionDirection')
```

Parameter	Description
ModelName	ModelName specifies the name of the model. If the model exists in the MATLAB's search path, the file name is sufficient. Otherwise, you must additionally specify the whole path and the extension ".mdl".
ConversionDirection	ConversionDirection specifies the conversion direction: 'DS2210' means, all DS2211 blocks are replaced by DS2210 blocks. 'DS2211' means, all DS2210 blocks are replaced by DS2211 blocks.

You can use the M file in the MATLAB Command Window or by including it in a script or another function.

Results

After successful conversion, three new files are created within the folder of the model:

<ModelName>.mdl	The converted model
<ModelName>_bak.mdl	A backup copy of the origin model
<ModelName>_log.txt	A log file containing information on the conversion process

If <ModelName>_bak.mdl or <ModelName>_log.txt already exist, the conversion process is cancelled.

Converting from DS2211 to DS2210

A DS2211 has more features than a DS2210, for example, some units of the DS2211 have more channels. For details, refer to [Comparing DS2210 and DS2211 Features](#) on page 188. If you convert a model with DS2211 blocks to a model with DS2210 blocks, these features are not supported. To be able to reverse the conversion, two new blocks are created: Switch Data block and DS2211 Dummy block.

The Switch Data block acts as a data container. It stores DS2211-specific data which is not supported by the corresponding DS2210 block. The block must be kept on the root level on the model, otherwise it is not evaluated.

The DS2211 Dummy block acts as a placeholder for the related DS2211 block. It has no functionality. It contains identification data for the corresponding DS2211 which is need for reverse conversation.

Do not delete these blocks. Otherwise it is not possible to reverse the conversion.

Affected DS2211 blocks The following table shows, which interfaces of which DS2211 blocks are affected.

Block	Condition
DS2211APU_ABS_CNT_RESET_Bx	The DS2210 does not support this block.
DS2211APU_AUXCAP_Bx_Cy	If the number of event windows is 2
DS2211APU_AUXCAPCONT_Bx_Cy	If the number of event windows is 2
DS2211APU_CAM_Bx_Cy	If channel 3 or 4 are used
DS2211APU_INJ_Bx_Gy	If the capture mode is set to "Position and Duration" or "Absolute"
DS2211D2F_Bx_Cy	If channel 7 ... 9 are used
DS2211DIO_SETUP_Bx	If an input port is used
DS2211F2D_Bx_Cy	If channel 9 ... 24 are used
DS2211PWM_Bx_Cy	If channel 7 ... 9 are used
DS2211PWM2D_Bx_Cy	If channel 9 ... 24 are used
DS2211DAC_Bx_Cy	If channel 13 ... 20 are used
DS2211RES_Bx_Cy	If channel 7 ... 10 are used
DS2211RES_Bx_Gy	If channel 7 ... 10 are used
DS2211VARAPU_ANG_REL_Bx	The DS2210 does not support this block.
DS2211VARAPU_AUXCAP_Bx_Cy	The DS2210 does not support this block.
DS2211VARAPU_CRANK_Bx	The DS2210 does not support this block.
DS2211VARAPU_IGN_Bx	The DS2210 does not support this block.
DS2211VARAPU_INJ_Bx_Gy	The DS2210 does not support this block.
DS2211VARSL_KNSG_Bx_Cy	The DS2210 does not support this block.

Wave tables DS2210's wave tables have lower resolutions for the engine position and signal level than DS2211's wave tables. Converting DS2211's wave tables is not supported. You have to convert them by yourself.

Converting from DS2210 to DS2211

Wave tables DS2211's wave tables have higher resolutions for the engine position and signal level than DS2210's wave tables. The wave tables are converted by the RTI blocks via interpolation of the entries.

Related topics

HowTos

[How to Convert a Model with DS2210 Blocks to a Model with DS2211 Blocks..... 191](#)


How to Convert a Model with DS2210 Blocks to a Model with DS2211 Blocks

Objective	You can convert your Simulink models which have DS2210 RTI blocks that they can be used with a DS2211 board.
rti221xConv	For detailed information on the rti221xConv M file and the conversion process, refer to rti221xConv Conversion Tool on page 188.
Wave tables	DS2211's wave tables have higher resolutions for the engine position and signal level than DS2210's wave tables. The wave tables are converted by the RTI blocks via interpolation of the entries.
Restrictions	The RTI CAN MultiMessage Blockset cannot be converted.
Limitations	Using DS2210 blocks deriving from user defined libraries are not supported. You must convert these libraries before.
Method	<p>To convert a model with DS2210 blocks to a model with DS2211 blocks</p> <ol style="list-style-type: none"> 1 Ensure that no <code><ModelName>_bak.mdl</code> and <code><ModelName>_log.txt</code> exist in the folder where <code><ModelName>.mdl</code> is stored (<code><ModelName></code> is the name of the model). The tool aborts the conversion process if such a file exist. 2 At the MATLAB Command Window type <code>rti221xConv('<ModelName>','DS2211')</code>
Result	The converted model is saved under the file name <code><ModelName>.mdl</code> in the current working folder. The old model file is saved under <code><ModelName>_bak.mdl</code> . A log file is created under <code><ModelName>_log.txt</code> .

Related topics

References

How to Convert a Model with DS2211 Blocks to a Model with DS2210 Blocks

Objective	You can convert your Simulink models which have DS2211 RTI blocks that they can be used with a DS2210 board.
rti221xConv	For detailed information on the <code>rti221xConv</code> M file and the conversion process, refer to rti221xConv Conversion Tool on page 188.
Wave tables	DS2210's wave tables have lower resolutions for the engine position and signal level than DS2211's wave tables. Converting DS2211's wave tables is not supported. You have to convert them by yourself.
Restrictions	<ul style="list-style-type: none"> ▪ The RTICAN MultiMessage blockset cannot be converted. ▪ As the blocks of DS2211 has enhanced features, some blocks cannot be converted completely. Refer to Converting from DS2211 to DS2210 on page 189. ▪ The APU blockset for the DS2210 supports only a maximum number of 18 cylinders. If more than 18 cylinders are configured in the model with DS2211 blocks, it cannot be converted. For details on the number of cylinders, refer to DS2211APU_CRANK_Bx (DS2211 RTI Reference .
Limitations	Using DS2211 blocks deriving from user defined libraries are not supported. You must convert these libraries before.
Method	<p>To convert a model with DS2211 blocks to a model with DS2210 blocks</p> <ol style="list-style-type: none"> 1 Ensure that no <code><ModelName>_bak.mdl</code> and <code><ModelName>_log.txt</code> exist in the folder where <code><ModelName>.mdl</code> is stored (<code><ModelName></code> is the name of the model). The tool aborts the conversion process if such a file exist. 2 At the MATLAB Command Window type <pre>rti221xConv('<ModelName>','DS2210')</pre>
Result	The converted model is saved under the file name <code><ModelName>.mdl</code> in the current working folder. This model contains two new blocks: Switch Data block and DS2211 Dummy block. These blocks contains the information needed for a reverse conversion. The old model file is saved under <code><ModelName>_bak.mdl</code> . A log file is created under <code><ModelName>_log.txt</code> .

How to Migrate Handcoded Applications for DS2211

Objective	You can convert handcoded applications for a DS2211 board to applications for a DS2211 board.
C code compatibility	All function of the RTLib of the DS2210 have a corresponding function in the RTLib of a DS2211. You must only adapt the function names.
Slave DSP compatibility	<p>Although, the TMS320C31 code is object compatible to the code of the TMS320VC33, a DS2210 application must be recompiled, because the DS2211 needs a different startup-code.</p> <p>As the TMS320VC33 features 32 KW additional internal on-chip memory, the 64 KW local memory available on the DS2210 was removed on the DS2211. Therefore you must use the linker command files installed with the DS2211.</p>
Wave tables	DS2211's wave tables have higher resolutions for the engine position and signal level than DS2210's wave tables. If you use the Mat2c2211.m M file, the wave table are automatically converted (interpolated). Otherwise you must convert the wave tables by yourself.
Method	<p>To migrate a handcoded slave DSP applications for DS2211</p> <ol style="list-style-type: none"> 1 Replace the include files ds2210.h by ds2211.h in the C code. 2 Replace the include files util2210.h by util2211.h in the C code. 3 Use the DS2211 RTLib functions (replace "ds2210_" by "ds2211_" in the function names). 4 If you are using custom local linker command files, you must replace it by the ones from the RTLib 2211.
Result	The C code is prepared for the DS2211.

Working with a Real-Time System Containing a DS2210 and a DS2211

Introduction	A real-time system can contain DS2210 and DS2211 boards. As long as they are connected only via the PHS bus, they can work independently of each other like any other I/O boards.
---------------------	---

Engine position bus

If DS2210 boards are connected to DS2211 boards via a time-base connector, a DS2210 must be the master. The APU cycle time, which drives the engine position bus of a DS2211, is smaller than the APU cycle time of a DS2210. A engine position bus controlled by a DS2211 would therefore lead to an overrun of a DS2210.

If the DS2211 model contains blocks of the VAR APU blockset, the engine position bus of the DS2211 board must not be connected to a DS2210 board. A DS2210 board does not support these blockset.

Limitations

Introduction	There are some limitations you have to take into account when working with the DS2211.
--------------	--

Where to go from here

Information in this section

Quantization Effects.....	196
Quantization effects occur in signal generation or measurement.	
DS2211 Board Revision.....	196
Some features are only available for DS2211 boards with higher board revision.	
Conflicting I/O Features.....	197
Explaining the conflicts concerning different I/O features.	
Limited Number of CAN Messages.....	211
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.....	213
There are a number of general limitations with RTICANMM.	
Limitations with J1939-Support.....	217
There are a number of limitations regarding the J1939 support of RTICANMM.	

Information in other sections

Introduction to the Features of the DS2211.....	9
---	-------------------

Quantization Effects

Introduction

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

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

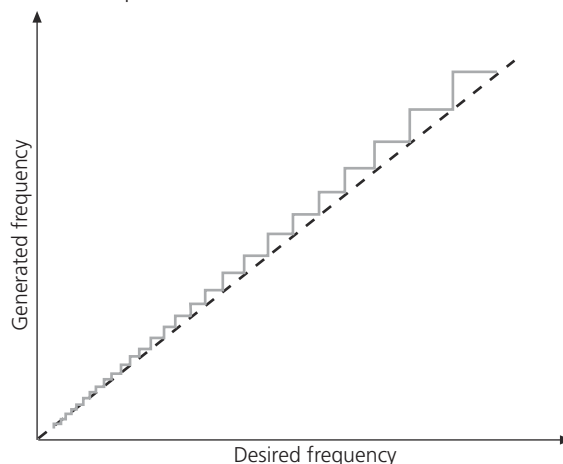
$$f = \frac{1}{n \cdot R}$$

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

Example

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

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



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

DS2211 Board Revision

Introduction

Several features are supported only for DS2211 boards with specific revisions and higher, for example, if you want to use the position and duration mode or

absolute mode for injection event capturing. The following board is extended in functionality:

- DS2211 boards with board revision 3 and FPGA revision 3 or higher (FPGA = field programmable gate array).

Revision numbers

The revision numbers are displayed with the following syntax:

<board>.<FPGA>

For example, 3.2 means you have a DS2211 board with the revision 3 and a FPGA revision number of 2 installed.

The revision number is printed on the board. You can also read out the number with ControlDesk, refer to [Board Details Properties \(ControlDesk Platform Management !\[\]\(830769b31eeeaca920791081939ff8ba_img.jpg\)](#)).

Conflicting I/O Features

Types of I/O conflicts

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

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

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

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

Conflicts for the DS2211





The following tables list the I/O features of the DS2211 that conflict with other I/O features, and the related RTI blocks/RTLib functions.

- Conflicts for the sensor and actuator interface
 - [Conflicts for Digital Inputs](#) on page 198
 - [Conflicts for Digital Outputs](#) on page 199
 - [Conflicts for PWM Signal Generation](#) on page 199
 - [Conflicts for Square-Wave Signal Generation \(D2F\)](#) on page 199
 - [Conflicts for PWM Signal Measurement \(PWM2D\)](#) on page 200
 - [Conflicts for Square-Wave Signal Measurement \(F2D\)](#) on page 201
 - [Conflicts for Wheel Speed Sensor Simulation](#) on page 202
 - [Conflicts for Camshaft Sensor Signal Generation](#) on page 202

- Conflicts for the angular processing unit (APU)
 - [Conflicts for Spark Event Capture \(Last Event Window Read\)](#) on page 203
 - [Conflicts for Spark Event Capture, Auxiliary Inputs \(Last Event Window Read\)](#) on page 204
 - [Conflicts for Spark Event Capture \(Continuous Read\)](#) on page 205
 - [Conflicts for Spark Event Capture, Auxiliary Inputs \(Continuous Read\)](#) on page 206
 - [Conflicts for Injection Pulse Position and Fuel Amount Measurement \(Last Event Window Read\)](#) on page 207
 - [Conflicts for Injection Pulse Position and Fuel Amount Measurement \(Continuous Read\)](#) on page 208
 - [Conflicts for Knock Sensor Simulation](#) on page 202
- Conflicts for communication channels
 - [Conflicts for Single Edge Nibble Transmission \(SENT\)](#) on page 210
 - [Conflicts for the Serial Interface](#) on page 211




Conflicts for Digital Inputs

The following I/O features of the DS2211 conflict with digital inputs provided by the sensor and actuator interface:

Digital Inputs *)		Signal	Conflicting I/O Feature **)	
Ch (RTI)	Ch (RTLib)		Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels				
Ch 1 ... ch 16	Ch 1 ... ch 16	DIG_IN1 ... DIG_IN16	<div><div>Square-wave signal measurement (F2D)</div><div>PWM signal measurement (PWM2D)</div></div>	<div>Ch 9 ... ch 24</div> <div>Ch 9 ... ch 24</div>
Although DIG_IN channels 1 ... 16 are shared with PWM input channels 9 ... 24, they can be used at the same time.				
Ch 1 ... ch 4	Ch 1 ... ch 4	DIG_IN1 ... DIG_IN4	Single edge nibble transmission (SENT receiver)	<div>Ch 1 ... ch 4</div> <div>Ch 1 ... ch 4</div>
<div>*) Related RTI blocks and RTLib functions:</div> <div><div>DS2211BIT_IN_Bx_Cy</div><div>DS2211BIT_IN16_Bx</div><div>Bit I/O Unit (DS2211 RTLib Reference </div></div>			<div>***) Related RTI blocks and RTLib functions:</div> <div><div>Square-wave signal measurement</div><div><div>DS2211F2D_Bx_Cy</div><div>Frequency Measurement (DS2211 RTLib Reference </div></div><div>PWM signal measurement</div><div><div>DS2211PWM2D_Bx_Cy</div><div>PWM Signal Measurement (DS2211 RTLib Reference </div></div><div>Single edge nibble transmission (SENT receiver):</div><div><div>DS2211SENT_RX_BLx</div><div>Single Edge Nibble Transmission (SENT) (DS2211 RTLib Reference </div></div></div>	



Conflicts for Digital Outputs

The following I/O features of the DS2211 conflict with digital outputs provided by the sensor and actuator interface:

Digital Outputs *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 15 and ch 16	Ch 15 and ch 16	DIG_OUT15 and DIG_OUT16	Camshaft sensor signal generation	Ch 3 and ch 4	Ch 3 and ch 4
Ch 1 ... ch 5	Ch 1 ... ch 5	DIG_OUT1 ... DIG_OUT5	Single edge nibble transmission	Ch 1 ... ch 5	Ch 1 ... ch 5
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211BIT_OUT_Bx_Cy DS2211BIT_OUT16_BxBit I/O Unit (DS2211 RTLib Reference 			**) Related RTI blocks and RTLib functions: Camshaft sensor signal generation <ul style="list-style-type: none">DS2211APU_CAM_Bx_CyCamshaft Sensor Signal Generation (DS2211 RTLib Reference  Single edge nibble transmission <ul style="list-style-type: none">DS2211SENT_TX_BLxSingle Edge Nibble Transmission (SENT) (DS2211 RTLib Reference 		

Conflicts for PWM Signal Generation



The following I/O features of the DS2211 conflict with PWM signal generation provided by the sensor and actuator interface:

PWM Signal Generation *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 9	Ch 1 ... ch 9	PWM_OUT1 ... PWM_OUT9	Square-wave signal generation (D2F)	Ch 1 ... ch 9	Ch 1 ... ch 9
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211PWM_Bx_CyPWM Signal Generation (DS2211 RTLib Reference 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211D2F_Bx_CySquare-Wave Signal Generation (DS2211 RTLib Reference 		

Conflicts for Square-Wave Signal Generation (D2F)






The following I/O features of the DS2211 conflict with square-wave signal generation provided by the sensor and actuator interface:

Square-Wave Signal Generation *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 9	Ch 1 ... ch 9	PWM_OUT1 ... PWM_OUT9	PWM signal generation	Ch 1 ... ch 9	Ch 1 ... ch 9

Square-Wave Signal Generation *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	Ch (RTLib)
Ch (RTI)	Ch (RTLib)				
*) Related RTI blocks and RTLib functions: ▪ DS2211D2F_Bx_Cy ▪ Square-Wave Signal Generation (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: ▪ DS2211PWM_Bx_Cy ▪ PWM Signal Generation (DS2211 RTLib Reference )		






Conflicts for PWM Signal Measurement (PWM2D)

The following I/O features of the DS2211 conflict with PWM signal measurement provided by the sensor and actuator interface:

PWM Signal Measurement *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	Ch (RTLib)
Ch (RTI)	Ch (RTLib)				
Conflicts Concerning Single Channels					
Ch 1 ... ch 6	Ch 1 ... ch 6	PWM_IN1 ... PWM_IN6	Square-wave signal measurement (F2D)	Ch 1 ... ch 6	Ch 1 ... ch 6
Ch 7 and ch 8	Ch 7 and ch 8	PWM_IN7 and PWM_IN8	<ul style="list-style-type: none">Square-wave signal measurement (F2D)Injection pulse position and fuel amount measurement	<ul style="list-style-type: none">Ch 7 and ch 8Group 1, ch 7 and group 1, ch 8	<ul style="list-style-type: none">Ch 7 and ch 8Group 1, ch 7 and group 1 ,ch 8
Ch 9 ... ch 24	Ch 9 ... ch 24	PWM_IN9 ... PWM_IN24	<ul style="list-style-type: none">Square-wave signal measurement (F2D)Digital input	<ul style="list-style-type: none">Ch 9 ... ch 24Ch 1 ... ch 16	<ul style="list-style-type: none">Ch 9 ... ch 24Ch 1 ... ch 16
Ch 9 ... ch 12	Ch 9 ... ch 12	PWM_IN9 ... PWM_IN12	Single edge nibble transmission (SENT receiver)	Ch 1 ... ch 4	Ch 1 ... ch 4
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211PWM2D_Bx_CyPWM Signal Measurement (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">Square-wave signal measurement (F2D):<ul style="list-style-type: none">DS2211F2D_Bx_CyFrequency Measurement (DS2211 RTLib Reference )Injection pulse position and fuel amount measurement:<ul style="list-style-type: none">DS2211APU_INJ_Bx_GyDS2211APU_INJCONT_Bx_GyDS2211VARAPU_INJ_Bx_GyInjection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference )Digital input:<ul style="list-style-type: none">DS2211BIT_IN16_BxDS2211BIT_IN_Bx_CyBit I/O Unit (DS2211 RTLib Reference )Single edge nibble transmission (SENT receiver):<ul style="list-style-type: none">DS2211SENT_RX_BLxSingle Edge Nibble Transmission (SENT) (DS2211 RTLib Reference )		

Conflicts for Square-Wave Signal Measurement (F2D)

The following I/O features of the DS2211 conflict with square-wave signal measurement provided by the sensor and actuator interface:

Square-Wave Signal Measurement *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 6	Ch 1 ... ch 6	PWM_IN1 ... PWM_IN6	PWM signal measurement (PWM2D)	Ch 1 ... ch 6	Ch 1 ... ch 6
Ch 7 and ch 8	Ch 7 and ch 8	PWM_IN7 and PWM_IN8	<ul style="list-style-type: none"> PWM signal measurement (PWM2D) Injection pulse position and fuel amount measurement 	<ul style="list-style-type: none"> Ch 7 and ch 8 Group 1, ch 7 and group 1, ch 8 	<ul style="list-style-type: none"> Ch 7 and ch 8 Group 1, ch 7 and group 1, ch 8
Ch 9 ... ch 24	Ch 9 ... ch 24	PWM_IN9 ... PWM_IN24	<ul style="list-style-type: none"> PWM signal measurement (PWM2D) Digital input 	<ul style="list-style-type: none"> Ch 9 ... ch 24 Ch 1 ... ch 16 	<ul style="list-style-type: none"> Ch 9 ... Ch 24 Ch 1 ... ch 16
Ch 9 ... ch 12	Ch 9 ... ch 12	PWM_IN9 ... PWM_IN12	Single edge nibble transmission (SENT receiver)	Ch 1 ... ch 4	Ch 1 ... ch 4
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> DS2211F2D_Bx_Cy Frequency Measurement (DS2211 RTLib Reference ) 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> PWM2D: <ul style="list-style-type: none"> DS2211PWM2D_Bx_Cy PWM Signal Measurement (DS2211 RTLib Reference ) Injection Pulse Position and Fuel Amount Measurement: <ul style="list-style-type: none"> DS2211APU_INJ_Bx_Gy DS2211APU_INJCONT_Bx_Gy DS2211VARAPU_INJ_Bx_Gy Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference ) Digital input: <ul style="list-style-type: none"> DS2211BIT_IN16_Bx DS2211BIT_IN_Bx_Cy Bit I/O Unit (DS2211 RTLib Reference ) Single edge nibble transmission (SENT receiver): <ul style="list-style-type: none"> DS2211SENT_RX_Bx Single Edge Nibble Transmission (SENT) (DS2211 RTLib Reference ) 		



Conflicts for Wheel Speed Sensor Simulation

The following I/O features of the DS2211 conflict with wheel speed sensor simulation provided by the sensor and actuator interface:

Wheel Speed Sensor Simulation *)		Signal	Conflicting I/O Feature **)	
Ch (RTI)	Ch (RTLib)		Ch (RTI)	Ch (RTLib)
Conflicts Concerning Wheel Speed Sensor Simulation as a Whole				
<ul style="list-style-type: none">If you perform wheel speed sensor simulation, you cannot perform knock sensor simulation at the same time.				
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SL_WSSG_Bx_CyWheel Speed Sensor Simulation (DS2211 RTLib Reference 📖)			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SL_KNSG_Bx_CyDS2211VARSL_KNSG_Bx_CyKnock Sensor Simulation (DS2211 RTLib Reference 📖)	

Conflicts for Camshaft Sensor Signal Generation

The following I/O features of the DS2211 conflict with wheel speed sensor simulation provided by the sensor and actuator interface:

Camshaft Sensor Signal Generation *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 3 and ch 4	Ch 3 and ch 4	CAM3_DIG and CAM4_DIG	Digital outputs	Ch 15 and ch 16	Ch 15 and ch 16
*) Related RTI blocks and RTLib functions: ▪ DS2211APU_CAM_Bx_Cy ▪ <i>Camshaft Sensor Signal Generation (DS2211 RTLib Reference </i>)			**) Related RTI blocks and RTLib functions: ▪ DS2211BIT_OUT16_Bx_DS2211BIT_OUT_Bx_Cy ▪ <i>Bit I/O Unit (DS2211 RTLib Reference </i>)		



Conflicts for Knock Sensor Simulation

The following I/O features of the DS2211 conflict with knock sensor simulation provided by the angular processing unit:

Knock Sensor Simulation *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch(RTLib)
Conflicts Concerning Knock Sensor Simulation as a Whole					
<ul style="list-style-type: none">If you perform knock sensor simulation, you cannot perform wheel speed sensor simulation at the same time.					
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SL_KNSG_Bx_Cy DS2211VARSL_KNSG_Bx_CyKnock Sensor Simulation (DS2211 RTLib Reference 📖)			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SL_WSSG_Bx_CyWheel Speed Sensor Simulation (DS2211 RTLib Reference 📖)		

Conflicts for Spark Event Capture (Last Event Window Read)

The following I/O features of the DS2211 conflict with spark event capture provided by the angular processing unit:

Spark Event Capture (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 6	Ch 1 ... ch 6	IGN1... IGN6	<ul style="list-style-type: none">▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Ch 1 ... ch 6▪ Group 2, ch 1 ... ch 6	<ul style="list-style-type: none">▪ –▪ Ch 1 ... ch 6
Ch 7 and ch 8	Ch 7 and ch 8	AUXCAP1 and AUXCAP2	<ul style="list-style-type: none">▪ Spark event capture, auxiliary input (continuous read and last event window read)▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Aux. ch 1 and Aux. ch 2▪ Ch 7 and ch 8▪ Group 2, ch 7 and group 2, ch 8	<ul style="list-style-type: none">▪ –▪ –▪ Ch 7 and ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211APU_IGN_Bx DS2211VARAPU_IGN_Bx▪ Spark Event Capture (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Spark event capture (continuous read):<ul style="list-style-type: none">▪ DS2211APU_IGNCONT_Bx▪ Injection pulse position and fuel amount measurement (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_INJ_Bx_Gy DS2211APU_INJCONT_Bx_Gy(group 2) DS2211VARAPU_INJ_Bx_Gy▪ Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference )▪ Spark event capture, auxiliary input (continuous read and last event window read):<ul style="list-style-type: none">▪ DS2211APU_AUXCAPCONT_Bx_Cy DS2211APU_AUXCAP_Bx_Cy DS2211VARAPU_AUXCAP_Bx_Cy		



Conflicts for Spark Event Capture, Auxiliary Inputs (Last Event Window Read)

The following I/O features of the DS2211 conflict with spark event capture provided by the angular processing unit when using the auxiliary inputs:

Spark Event Capture (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Aux. ch 1	Aux. ch 1	AUXCAP1	<ul style="list-style-type: none">▪ Spark event capture, auxiliary input (continuous read)▪ Spark event capture (continuous read)▪ Spark event capture (last event window)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Aux. ch 1/ Aux. ch 2▪ All channels▪ Ch 7▪ Group 2, ch 7	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 7
Aux. ch 2	Aux. ch 2	AUXCAP2	<ul style="list-style-type: none">▪ Spark event capture, auxiliary input (continuous read)▪ Spark event capture (continuous read)▪ Spark event capture (last event window)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Aux. ch 1/ Aux. ch 2▪ All channels▪ Ch 8▪ Group 2, ch 8	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211APU_AUXCAP_Bx_Cy DS2211VARAPU_AUXCAP_Bx_Cy▪ Spark Event Capture (DS2211 RTLib Reference)			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_IGN_Bx DS2211APU_IGNCONT_Bx DS2211VARAPU_IGN_Bx▪ Injection pulse position and fuel amount measurement (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_INJ_Bx_Gy DS2211APU_INJCONT_Bx_Gy(group 2) DS2211VARAPU_INJ_Bx_Gy▪ Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference)▪ Spark event capture, auxiliary input (continuous read):<ul style="list-style-type: none">▪ DS2211APU_AUXCAPCONT_Bx_Cy		



Conflicts for Spark Event Capture (Continuous Read) *

The following I/O features of the DS2211 conflict with spark event capture provided by the angular processing unit:

Spark Event Capture (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 6	Ch 1 ... ch 6	IGN1 ... IGN6	<ul style="list-style-type: none">▪ Spark event capture (last event window read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)▪ Spark event capture, auxiliary input (last event window read)	<ul style="list-style-type: none">▪ Ch 1 ... ch 6▪ Group 2, ch 1 ... ch 6▪ Aux. ch 1/ Aux. ch 2	<ul style="list-style-type: none">▪ –▪ Ch 1 ... ch 8▪ –
Ch 7	Ch 7	AUXCAP1	<ul style="list-style-type: none">▪ Spark event capture (last event window read)▪ Spark event capture, auxiliary input (last event window read)▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Ch 1 ... ch 8▪ Aux. ch 1/ Aux. ch 2▪ Ch 7▪ Ch 7	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 7
Ch 8	Ch 8	AUXCAP2	<ul style="list-style-type: none">▪ Spark event capture (last event window read)▪ Spark event capture, auxiliary input (last event window read)▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Ch 1 ... ch 8▪ Aux. ch 1/ Aux. ch 2▪ Ch 8▪ Ch 8	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211APU_IGNCONT_Bx▪ <i>Spark Event Capture (DS2211 RTLib Reference </i>)			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Spark event capture (last event window read):<ul style="list-style-type: none">▪ DS2211APU_IGN_BxDS2211VARAPU_IGN_Bx▪ Spark event capture, auxiliary input (last event window read):<ul style="list-style-type: none">▪ DS2211APU_AUXCAP_Bx_CyDS2211VARAPU_AUXCAP_Bx_Cy▪ Injection pulse position and fuel amount measurement (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_INJ_Bx_GyDS2211APU_INJCONT_Bx_Gy(group 2)DS2211VARAPU_INJ_Bx_Gy▪ <i>Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference </i>)		




Conflicts for Spark Event Capture, Auxiliary Inputs (Continuous Read)

The following I/O features of the DS2211 conflict with spark event capture provided by the angular processing unit when using the auxiliary inputs:

Spark Event Capture (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Aux. ch 1	Aux. ch 1	AUXCAP1	<ul style="list-style-type: none">▪ Spark event capture (last event window read)▪ Spark event capture, auxiliary input (last event window read)▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ All channels▪ Aux. ch 1/ Aux. ch 2▪ Ch 7▪ Ch 7	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 7
Aux. ch 2	Aux. ch 2	AUXCAP2	<ul style="list-style-type: none">▪ Spark event capture (last event window read)▪ Spark event capture, auxiliary input (last event window read)▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ All channels▪ Aux. ch 1/ Aux. ch 2▪ Ch 8▪ Ch 8	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211APU_AUXCAPCONT_Bx_Cy▪ <i>Spark Event Capture (DS2211 RTLib Reference</i> 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Spark event capture (last event window read):<ul style="list-style-type: none">▪ DS2211APU_IGN_Bx▪ DS2211VARAPU_IGN_Bx▪ Spark event capture, auxiliary input (last event window read):<ul style="list-style-type: none">▪ DS2211APU_AUXCAP_Bx_Cy▪ DS2211VARAPU_AUXCAP_Bx_Cy▪ Spark event capture (continuous read):<ul style="list-style-type: none">▪ DS2211APU_IGNCONT_Bx▪ Injection pulse position and fuel amount measurement (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_INJ_Bx_Gy▪ DS2211APU_INJCONT_Bx_Gy(group 2)▪ DS2211VARAPU_INJ_Bx_Gy▪ <i>Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference</i> 		

Conflicts for Injection Pulse Position and Fuel Amount Measurement (Last Event Window Read)

The following I/O features of the DS2211 conflict with injection pulse position (group 1) provided by the angular processing unit:

Injection Pulse Position and Fuel Amount Measurement (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)		Group (RTI)	Ch (RTLib)	
Conflicts Concerning Group 1					
Group 1, ch 1 ... ch 6	Ch 1 ... ch 6	INJ1 ... INJ6	Injection pulse position and fuel amount measurement (continuous read)	Group 1 (all channels)	–
Group 1, ch 7	Ch 7	INJ7	<ul style="list-style-type: none">Injection pulse position and fuel amount measurement (continuous read)Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D)	<ul style="list-style-type: none">Ch 7Ch 7	<ul style="list-style-type: none">Ch 7Ch 7
Group 1, ch 8	Ch 8	INJ8	<ul style="list-style-type: none">Injection pulse position and fuel amount measurement (continuous read)Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D)	<ul style="list-style-type: none">Ch 8Ch 8	<ul style="list-style-type: none">Ch 8Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211APU_INJ_Bx_Gy (group 1) DS2211VARAPU_INJ_Bx_Gy (group 1)Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">Injection pulse position and fuel amount measurement (continuous read)<ul style="list-style-type: none">DS2211APU_INJCONT_Bx_Gy (group 1)Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D):<ul style="list-style-type: none">DS2211PWM2D_Bx_Cy DS2211F2D_Bx_CyPWM Signal Measurement (DS2211 RTLib Reference ) Frequency Measurement (DS2211 RTLib Reference )		

The following I/O features of the DS2211 conflict with fuel amount measurement (group 2) provided by the angular processing unit:

Injection Pulse Position and Fuel Amount Measurement (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)		Group (RTI)	Ch (RTLib)	
Conflicts Concerning Group 2					
Group 2, ch 1 ... ch 6	Ch 1 ... ch 6	IGN1 ... IGN6	<ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)▪ Injection pulse position and fuel amount measurement (continuous read)	<ul style="list-style-type: none">▪ All channels▪ Group 2 (all channels)	<ul style="list-style-type: none">▪ All channels▪ –

Injection Pulse Position and Fuel Amount Measurement (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)			Group (RTI)	Ch (RTLib)
Group 2, ch 7	Ch 7	AUXCAP1	<ul style="list-style-type: none"> Spark event capture (last event window and continuous read) Injection pulse position and fuel amount measurement (continuous read) Spark event capture, auxiliary input (last event window and continuous read) 	<ul style="list-style-type: none"> All channels Group 2 (all channels) Aux. ch 1 	<ul style="list-style-type: none"> All channels – Aux. ch 1
Group 2, ch 8	Ch 8	AUXCAP2	<ul style="list-style-type: none"> Spark event capture (last event window and continuous read) Injection pulse position and fuel amount measurement (continuous read) Spark event capture, auxiliary input (last event window and continuous read) 	<ul style="list-style-type: none"> All channels Group 2 (all channels) Aux. ch 2 	<ul style="list-style-type: none"> All channels – Aux. ch 2
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> DS2211APU_INJ_Bx_Gy (group 2) DS2211VARAPU_INJ_Bx_Gy (group 2) Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference) 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> Spark event capture (last event window and continuous read): <ul style="list-style-type: none"> DS2211APU_IGN_Bx DS2211APU_IGNCONT_Bx DS2211VARAPU_IGN_Bx Spark Event Capture (DS2211 RTLib Reference) Injection pulse position and fuel amount measurement (continuous read): <ul style="list-style-type: none"> DS2211APU_INJCONT_Bx_Gy (group 2) Spark event capture, auxiliary input (last event window and continuous read): <ul style="list-style-type: none"> DS2211APU_AUXCAP_Bx_Cy DS2211APU_AUXCAPCONT_Bx_Cy DS2211VARAPU_AUXCAP_Bx_Cy Spark Event Capture (DS2211 RTLib Reference) 		

Conflicts for Injection Pulse Position and Fuel Amount Measurement (Continuous Read)




The following I/O features of the DS2211 conflict with injection pulse position (group 1) provided by the angular processing unit:

Injection Pulse Position and Fuel Amount Measurement (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)			Group (RTI)	Ch (RTLib)
Conflicts Concerning Group 1					
Group 1, ch 1 ... ch 6	Ch 1 ... ch 6	INJ1 ... INJ6	Injection pulse position and fuel amount measurement (last event window read)	Group 1 (all channels)	—

Injection Pulse Position and Fuel Amount Measurement (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)			Group (RTI)	Ch (RTLib)
Group 1, ch 7	Ch 7	INJ7	<ul style="list-style-type: none"> Injection pulse position and fuel amount measurement (last event window read) Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D) 	<ul style="list-style-type: none"> Group 1 (all channels) Ch 7 	<ul style="list-style-type: none"> – Ch 7
Group 1, ch 8	Ch 8	INJ8	<ul style="list-style-type: none"> Injection pulse position and fuel amount measurement (last event window read) Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D) 	<ul style="list-style-type: none"> Group 1 (all channels) Ch 8 	<ul style="list-style-type: none"> – Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> DS2211APU_INJCONT_Bx_Gy (group 1) <i>Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference 📖)</i> 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> Injection pulse position and fuel amount measurement (last event window read): <ul style="list-style-type: none"> DS2211APU_INJ_Bx_Gy (group 1) DS2211VARAPU_INJ_Bx_Gy (group 1) PWM2D/F2D: <ul style="list-style-type: none"> DS2211PWM2D_Bx_Cy DS2211F2D_Bx_Cy <i>PWM Signal Measurement (DS2211 RTLib Reference 📖)/ Frequency Measurement (DS2211 RTLib Reference 📖)</i> 		



The following I/O features of the DS2211 conflict with fuel amount measurement (group 2) provided by the angular processing unit:

Injection Pulse Position and Fuel Amount Measurement (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)			Group (RTI)	Ch (RTLib)
Conflicts Concerning Group 2					
Group 2, ch 1 ... ch 6	Ch 1 ... ch 6	IGN1 ... IGN6	<ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)▪ Injection pulse position and fuel amount measurement (last event window read)	<ul style="list-style-type: none">▪ All channels▪ Group 2 (all channels)	<ul style="list-style-type: none">▪ All channels▪ –
Group 2, ch 7	Ch 7	AUXCAP1	<ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)▪ Injection pulse position and fuel amount measurement (last event window read)▪ Spark event capture, auxiliary input (last event window and continuous read)	<ul style="list-style-type: none">▪ All channels▪ Group 2 (all channels)▪ Aux. ch 1	<ul style="list-style-type: none">▪ All channels▪ –▪ Aux. ch 1

Injection Pulse Position and Fuel Amount Measurement (Continuous Read) *)		Signal	Conflicting I/O Feature **)	
Group (RTI)	Ch (RTLib)		Group (RTI)	Ch (RTLib)
Group 2, ch 8	Ch 8	AUXCAP2	<ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)▪ Injection pulse position and fuel amount measurement (last event window read)▪ Spark event capture, auxiliary input (last event window and continuous read)	<ul style="list-style-type: none">▪ All channels▪ Group 2 (all channels)▪ Aux. ch 2 <ul style="list-style-type: none">▪ All channels▪ –▪ Aux. ch 2
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211APU_INJCONT_Bx_Gy (group 2)▪ Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)<ul style="list-style-type: none">▪ DS2211APU_IGN_Bx▪ DS2211APU_IGNCONT_Bx▪ DS2211VARAPU_IGN_Bx▪ Spark Event Capture (DS2211 RTLib Reference )▪ Injection pulse position and fuel amount measurement (last event window read)<ul style="list-style-type: none">▪ DS2211APU_INJ_Bx_Gy (group 2)/▪ DS2211VARAPU_INJ_Bx_Gy (group 2)▪ Spark event capture, auxiliary input (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_AUXCAP_Bx_Cy▪ DS2211APU_AUXCAPCONT_Bx_Cy▪ DS2211VARAPU_AUXCAP_Bx_Cy▪ Spark Event Capture (DS2211 RTLib Reference )	

Conflicts for Single Edge Nibble Transmission (SENT)

The following I/O features of the DS2211 conflict with a SENT transmitter:

SENT Transmitter *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)		Ch (RTI)	Ch (RTLib)	
Conflicts Concerning Single Channels					
Ch 1 ... ch 5	Ch 1 ... ch 5	DIG_OUT1 ... DIG_OUT5	Digital output	Ch 1 ... ch 5	Ch 1 ... ch 5
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211SENT_TX_BLx▪ Single Edge Nibble Transmission (SENT) (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211BIT_OUT_Bx_Cy▪ DS2211BIT_OUT16_Bx▪ Bit I/O Unit (DS2211 RTLib Reference )		

The following I/O features of the DS2211 conflict with a SENT receiver:

SENT Receiver *)		Signal	Conflicting I/O Feature **)	
Ch (RTI)	Ch (RTLib)		Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels				
Ch 1 ... ch 4	Ch 1 ... ch 4	DIG_IN1 ... DIG_IN4	<ul style="list-style-type: none">Digital inputsSquare-wave signal measurement (F2D)PWM signal measurement (PWM2D)	<ul style="list-style-type: none">Ch 1 ... ch 4Ch 9 ... ch 12Ch 9 ... ch 12 <ul style="list-style-type: none">Ch 1 ... ch 4Ch 9 ... ch 12Ch 9 ... ch 12
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SENT_RX_BLxSingle Edge Nibble Transmission (SENT) (DS2211 RTLib Reference 📖)		**) Related RTI blocks and RTLib functions: Digital inputs <ul style="list-style-type: none">DS2211BIT_IN_Bx_Cy DS2211BIT_IN16_BxBit I/O Unit (DS2211 RTLib Reference 📖) Square-wave signal measurement (F2D) <ul style="list-style-type: none">DS2211F2D_Bx_CyFrequency Measurement (DS2211 RTLib Reference 📖) PWM signal measurement (PWM2D) <ul style="list-style-type: none">DS2211PWM2D_Bx_CyPWM Signal Measurement (DS2211 RTLib Reference 📖)		

Conflicts for the Serial Interface

The DS2211 supports only one serial interface. It can be configured to either RS232 or RS422 mode.

Related topics

References

[Introduction to the Features of the DS2211](#) 9

Limited Number of CAN Messages

Limitation

When you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions, the number of CAN messages in an application is limited.

This applies to the following message types:

- Transmit (TX) messages
- Receive (RX) messages

- Request (RQ) messages

An RQ message and the corresponding RX message are interpreted as a single (RQ) message. You cannot enable RX service support for the corresponding RX message.

- Remote (RM) messages

The sum of these messages is n_{sum} :

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

Maximum number of CAN messages

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

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

n_{max} in one application depends on:

- Whether you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions.
- Whether you use RX service support.

The maximum number of CAN messages n_{max} is listed in the table below:

Platform	n_{max} with RTLib	n_{max} with RTI CAN Blockset							
		RX Service Support Disabled				RX Service Support Enabled			
		1 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 135.

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 J1939-Support

Limitations

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

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

A

ADC unit 18
 I/O mapping 19
 analog/digital converter 18
 angle position interrupt 74
 angular processing unit
 overview 73
 angular processing unit - variant 97

B

basics
 SENT 48
 bit I/O unit 25
 I/O mapping 25

C

camshaft phase 81
 camshaft sensor signal generation 106
 I/O mapping 107
 camshaft signal generator 81
 CAN
 channel 126
 fault-tolerant transceiver 130
 interrupts 134
 physical layer 128
 service 137
 setup 126
 status information 137
 CAN controller
 frequency 132
 CAN FD 145
 CAN subsystem 10
 CAN support 125
 cascading I/O boards 76
 clock drift (SENT) 49
 Common Program Data folder 8
 comparing features of DS2210 and DS2211 188
 comparing transceiver modes 43
 complex comparator 93
 connecting time-base 76
 continuous capture 88
 continuous capture mode 83
 crankshaft sensor signal generation 104
 I/O mapping 106
 crankshaft signal generator 77

D

D/A channel 121
 D/R converter 23
 I/O mapping 24
 DAC unit 20
 I/O mapping 21
 data file support 133
 defining CAN messages 133
 digital/analog converter 20
 digital/resistance converter 23
 Documents folder 8

DS2210 and DS2211
 comparing features 188
 working with 193
 DS2211
 master board 77
 slave board 77
 DS2211 board revision 196
 DS2211 interrupts
 basics 165
 DS2211SENT_TX_BLx
 implementing 53
 DS2211SENT_TXFIFO_BLx
 implementing 53
 DS802
 partitioning PHS bus 15
 DSP DAC unit 121
 I/O mapping 121
 DSP subsystem 10
 duty cycle
 measuring 27

E

engine position bus 74
 engine position interrupt 74
 engine position phase accumulator 75
 event capture window 83, 88

F

feature summary 12
 frequency
 CAN controller 132
 frequency measurement 34
 fuel amount measurement 84
 I/O mapping 112

H

high pulse (SENT) 49
 hysteresis 93

I

I/O mapping
 ADC unit 19
 bit I/O unit 25
 camshaft sensor signal generation 107
 crankshaft sensor signal generation 106
 D/R converter 24
 DAC unit 21, 121
 frequency measurement 35
 fuel amount measurement 112
 knock sensor simulation 119
 PWM signal generation 33
 PWM signal measurement 30
 spark event measurement 110
 square-wave signal generation 38
 wheel speed sensor simulation 120
 ignition pulses 83
 implementing
 SENT 51
 injection event capture unit 84

injection pulse position measurement 111
 continuous capture 92
 I/O mapping 112
 interrupts 165
 basics 165
 ionic current measurement 170
 ISO11898 129
 ISO11898-6 130

J

J1939
 broadcast/peer-to-peer messages 150
 limitations 217
 working with J1939-compliant DBC files 150
 J2716 (SAE) 48

K

knock sensor simulation 117
 I/O mapping 119

L

limitations
 J1939 217
 RTI CAN MultiMessage Blockset 213
 Local Program Data folder 8
 low pulse (SENT) 49

M

master DS2211 77
 measuring duty cycle 27
 messages
 defining CAN messages 133
 delay time 133
 multiple 133
 multiple data files 133
 multiple event capture mode 84
 multiple message support 133

N

nibble pulse (SENT) 50

P

partitioning PHS bus with DS802 15
 pause pulse (SENT) 50
 phase accumulator 75
 PHS++ bus 14
 piggyback support 131
 PWM analysis 27
 PWM signal generation 31
 I/O mapping 33
 PWM signal measurement 27
 I/O mapping 30

Q

quantization effects 196

R

- receive buffer 45
- receiving
 - SENT message 63
 - SENT message via RTI 56
- resistor channel
 - I/O mapping 24
- resistor output circuits 23
- reverse cranking 78
- RS485 130
- RTI CAN MultiMessage Blockset
 - limitations 213
 - supported platforms 140
- rti221xConv 188
- RTICANMM
 - J1939 217
- RX service support 135
- RX SW FIFO 45

S

- SAE J2716 48
- SAI 17
- sending
 - SENT message via RTI 53
 - SENT message via RTLib 60
- sensor and actuator interface 17
- SENT 47
 - basics 48
 - implementing on DS2211 51
 - message 48
 - terms and definitions 48
- SENT message
 - receiving 63
 - receiving via RTI 56
 - sending via RTI 53
 - sending via RTLib 60
- SENT pulse 49
- serial interface 41
 - baud rates 44
 - cable length/ baud rate (RS232) 43
 - cable length/ baud rate (RS422) 43
 - oscillator frequency 44
 - RS232 transceiver mode 43
 - RS422 networks 43
 - RS422 topologies 44
 - RS422 transceiver mode 43
- signal
 - SENT message 48
- signal capture mode 94
- simulating
 - engine variants 97
- simulating ionic current measurement 170
- single event capture mode 83
- slave DS2211 77
- slave DSP
 - basics 116
- slave DSP TMS320C31 116
- spark event
 - continuous capture 92
 - spark event capture unit 83

- spark event measurement 109
 - I/O mapping 110
- square-wave signal generation 37
 - I/O mapping 38
- start position/fuel amount capture mode 86
- sync high pulse (SENT) 49
- sync pulse (SENT) 49

T

- TDC 110
- threshold voltage 93
- tick period (SENT) 49
- tick period tolerance (SENT) 49
- time-base bus 74
- time-base connector 76
- time-base master 76
- time-base slave 76
- TJA1041
 - limitations 131
- TJA1054
 - transceiver 131
- top dead center (TDC) 110
- transceiver
 - TJA1054 131
- transmit buffer 45
- TX SW FIFO 45

U

- UART 41
- use scenario 169
- using VAR APU blockset 99

V

- VAR APU blockset 97
 - differences to APU blockset 99
 - example of capturing multiple event 101
 - related RTI blocks 98
 - related RTLib functions 98
 - using 99

W

- wave table
 - basics 108
 - generation 108
- wheel speed sensor simulation 119
 - I/O mapping 120
- working with CAN FD 145
- working with DS2210 and DS2211 193

Z

- zero nibble pulse (SENT) 49