DS2210 HIL I/O Board

Features

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Contents

About This Document 7

Introduction to the Features of the DS2210 9

DS2210 Architecture 9
Feature Overview 12
DS2210 Interfaces 14

Sensor and Actuator Interface 17

ADC Unit 18
DAC Unit 20
D/R Converter 22
Bit I/O Unit 24
PWM Signal Measurement 25
PWM Signal Generation 29
Frequency Measurement 32
Square-Wave Signal Generation 34

Serial Interface 37

Basics on the Serial Interface 37
Comparing RS232 and RS422 39
Specifying the Baud Rate of the Serial Interface 40
Software FIFO Buffer 41

Angular Processing Unit 43

APU Basics 44
APU Overview 45
Engine Position Phase Accumulator 46
Cascading I/O Boards 47
Crankshaft Signal Generator 49
Camshaft Signal Generator 50
Spark Event Capture Unit 51
Injection Event Capture Unit 52
Event Capture Windows 55
Continuous Value Capturing 58

APU Reference 59
Crankshaft Sensor Signal Generation 60
Camshaft Sensor Signal Generation 61
Wave Table Basics 62
Generating Wave Tables 63
Spark Event Capture 64
Injection Pulse Position and Fuel Amount Measurement 65

Features Served by the Slave DSP 69

Slave DSP TMS320C31 Basics 69
Knock Sensor Simulation 70
Wheel Speed Sensor Simulation 73

CAN Support 75

Setting Up a CAN Controller 76
Initializing the CAN Controller 76
CAN Transceiver Types 78
Defining CAN Messages 82
Implementing a CAN Interrupt 84
Using RX Service Support 84
Removing a CAN Controller (Go Bus Off) 86
Getting CAN Status Information 87

Using the RTI CAN MultiMessage Blockset 89
Basics on the RTI CAN MultiMessage Blockset 89
Basics on Working with CAN FD 94
Basics on Working with a J1939-Compliant DBC File 99
Transmitting and Receiving CAN Messages 105
Manipulating Signals to be Transmitted 108

CAN Signal Mapping 112
CAN Signal Mapping 112

Interrupts 113

DS2210 Interrupts 113
Interrupt Handling 114

Limitations 117

Quantization Effects 118
DS2210 Board Revision 119

Conflicting I/O Features.. 119
Limited Number of CAN Messages.. 129
Limitations with RTICANMM.. 130
Limitations with J1939-Support.. 134

Index 137

About This Document

Contents

This document provides feature-oriented access to the reference information you need to implement the functions provided by the *DS2210 HIL I/O Board*.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
⚠ CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
?	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
📖	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 DS2210

Introduction

The DS2210 HIL I/O Board is tailored to simulate and measure automotive signals. It combines a variety of typical HIL I/O functions on one board. The board also contains signal conditioning for typical signal levels of 12 V automotive systems.

Where to go from here

Information in this section

DS2210 Architecture........9
The DS2210 consists of several functional units in the architecture

Feature Overview........12
Introduction of the DS2210 features, summarized in alphabetical order.

DS2210 Interfaces........14
The DS2210 has interfaces for connection to a PHS-bus-based system and for cascading boards.

DS2210 Architecture

Introduction

The DS2210 includes the several functional units.

Angular processing unit

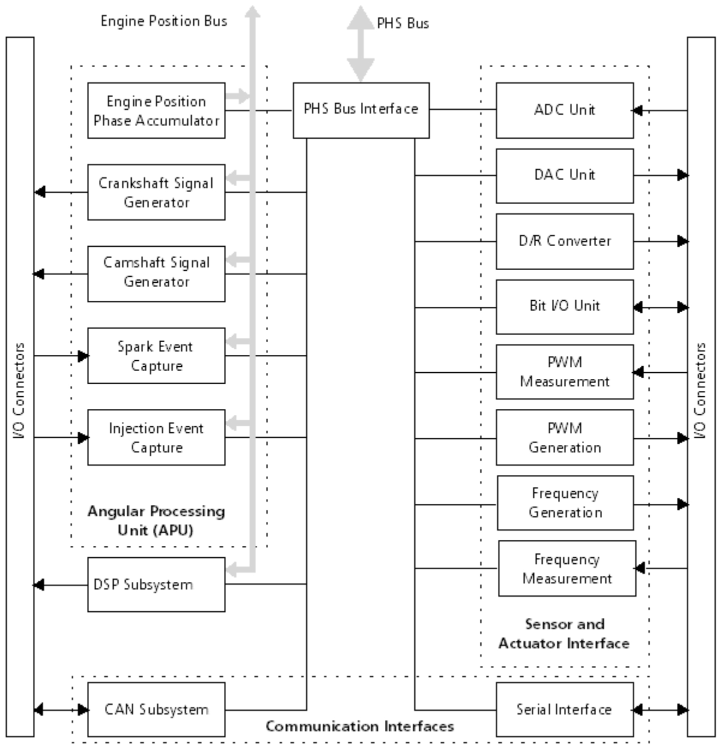
The most important feature of the DS2210 is the angular processing unit (APU) that provides the engine HIL core functions:

- Crankshaft/camshaft signal generation
- Spark event measurement
- Injection pulse position and fuel amount measurement
- Knock sensor simulation

Sensor and actuator interface	The DS2210 contains a sensor and actuator interface that provides a typical set of automotive I/O functions, including A/D conversion, digital I/O, and wheel speed sensor signal generation, for example.
DSP subsystem	The DSP subsystem is based on the TMS320C31. 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	In addition to a standard serial interface (RS232, RS422 based on a Texas Instruments TL 16C550 UART), the DS2210 includes a CAN subsystem that is based on the Siemens SAB 80C167 microcontroller. It provides connections to two CAN buses.
Master and slaves	The processor board has access to both the DSP and the CAN subsystems. In terms of inter-processor communication, the processor board is the master, whereas the DS2210 microcontrollers are slaves.
Limitations	There are some limitations when you work with the DS2210. See Limitations on page 117.

System overview

The illustration shows the architecture and the functional units of the DS2210:



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

Basics

DS2210 Interfaces........ 14
Feature Overview........ 12

Feature Overview

Introduction	The DS2210 provides several features. **Note** Some features are available only for DS2210 boards with extended functionality. To check whether or not your board supports extended functionality, refer to DS2210 Board Revision on page 119.
A/D conversion	The ADC unit provides 16 unipolar A/D channels with 12-bit resolution and 1.1 µs conversion time for each channel. 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 24.
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 75.
D/A conversion	The DAC unit provides 12 unipolar D/A channels (for user output) with 12-bit resolution and 20 µs full-scale settling time to 1 LSB. Refer to DAC Unit on page 20.
D/R conversion	The D/R converter provides 6 independent resistance outputs with 16-bit resolution covering a resistance range of 15 Ω … ∞. Refer to D/R Converter on page 22.
Engine HIL simulation	The angular processing unit provides the following features: ▪ Simulation of engine core functions is based on a 13-bit engine position (angle) for an engine cycle of 0 … 720° and a resolution of 0.088°. The engine position is updated every 1 µs. Refer to Engine Position Phase Accumulator on page 46. ▪ Crankshaft sensor simulation provides one crankshaft waveform output with 8 selectable waveforms. Refer to Crankshaft Signal Generator on page 49 and Crankshaft Sensor Signal Generation on page 60. ▪ Camshaft sensor simulation provides 2 camshaft waveform outputs with 8 selectable waveforms for each output. Refer to Camshaft Signal Generator on page 50 and Camshaft Sensor Signal Generation on page 61.

- 6 different interrupts can be generated depending on the engine position. Refer to APU Overview on page 45 and Interrupts on page 113.

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 measurement or for ignition capture. Refer to Spark Event Capture Unit on page 51 and Spark Event Capture on page 64. As a whole, 8 channels are available for spark event capture.
- For injection pulse position and fuel amount measurement, 8 digital injection inputs are available. Refer to Injection Event Capture Unit on page 52 and Injection Pulse Position and Fuel Amount Measurement on page 65.

Note

If you do not use the ignition capture channels for spark event capture, you can use them additionally for injection capture (only for boards with extended functionality, refer to DS2210 Board Revision on page 119).

Frequency measurement

For frequency measurement, 8 channels are available with a resolution of 21 bit. Refer to Frequency Measurement on page 32.

Frequency generation

For frequency generation, 6 channels are available with a resolution of 20 bit. Refer to Square-Wave Signal Generation on page 34.

Interrupt control

The DS2210 PHS bus interrupt controller provides eight hardware interrupts for the serial interface, the CAN subsystem, and the angular processing unit. Refer to Interrupts on page 113.

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 70. Knock sensor simulation and wheel speed sensor simulation cannot be used at the same time.

Pulse generation

For PWM generation, 6 independent outputs with run-time adjustable frequencies and duty cycles are available. Refer to PWM Signal Generation on page 29.

Pulse measurement

For PWM measurement, 8 independent input channels are available. Refer to PWM Signal Measurement on page 25.

Serial interface

The serial interface is based on the standard UART TL16C550C by Texas Instruments that can be configured as an RS232 or RS422 interface. Refer to Serial Interface on page 37.

User-specific slave DSP applications

You can program your own slave DSP applications to generate specific signals. Refer to Slave DSP TMS320C31 Basics on page 69.

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 73. Knock sensor simulation and wheel speed sensor simulation cannot be used at the same time.

Related topics

Basics

DS2210 Architecture........ 9
DS2210 Interfaces........ 14

DS2210 Interfaces

Introduction

The DS2210 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 DS2210 is always part of a PHS-bus-based system. While the DS2210 measures and simulates the signals required, the processor board takes over the calculation of the real-time model. That is, applications using DS2210 I/O features are implemented on the processor board. Together with a processor board, the DS2210 constitutes a basic HIL simulator.

Communication between processor board and I/O board is performed via the peripheral high-speed bus: That is the PHS++ bus. In the documentation we use the term "PHS bus" for both bus versions.

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 📖).

Cascading DS2210 boards

Several DS2210 boards can be cascaded to expand from 8-cylinder simulation to 16-cylinder simulation, or even more via the engine position bus.

Related topics

Basics

DS2210 Architecture.. 9
Feature Overview.. 12

Sensor and Actuator Interface

Introduction

The sensor and actuator interface (SAI) has several units for signal generation and signal measurement. The following topics provide information on the components.

Where to go from here

Information in this section

ADC Unit 18
The DS2210 has an analog digital converter (DAC) unit providing 16 inputs.

DAC Unit 20
The DS2210 has a digital analog converter (DAC) unit for user output.

D/R Converter 22
The DS2210 has a digital resistance (D/R) converter to simulate sensors that have a resistance output.

Bit I/O Unit 24
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.

PWM Signal Measurement 25
In hardware-in-the-loop applications, PWM signal measurement is used to capture digital signals. For evaluation of frequencies and duty cycles, digital pulses have to be recorded with high speed and transferred to a processor that performs the analysis.

PWM Signal Generation 29
Six independent PWM outputs are available for the generation of nonnegative square-wave signals.

Frequency Measurement 32
Eight independent channels are available to measure the frequency of square-wave signals.

Square-Wave Signal Generation..34
Six independent channels are available to generate square-wave signals with variable frequencies.

Information in other sections

Features Served by the Slave DSP...69
Wheel speed sensor simulation and knock sensor simulation are performed by a ready-to-use application implemented on the slave DSP.

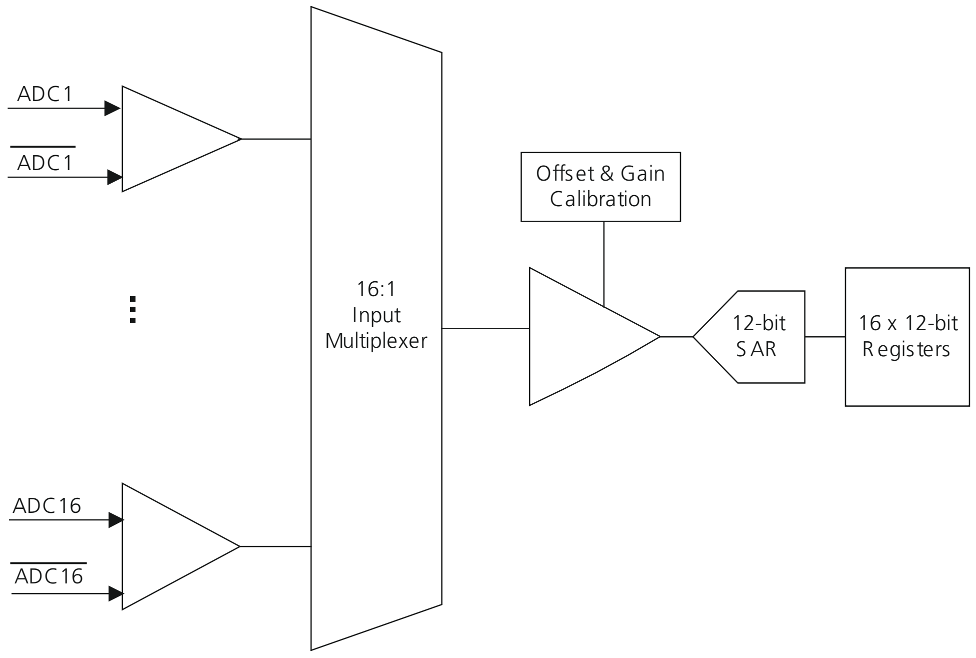
ADC Unit

Characteristics

The ADC unit consists of a 12-bit successive approximation register (SAR) A/D converter with a 16:1 input multiplexer that provides 16 inputs (ADC1 … ADC16), with 12-bit resolution each, 1.1 µs conversion time, and one integrated sample/hold for all inputs. All inputs are differential unipolar inputs with 0 … 20 V input span, lowpass input filters (1st order/–3 dB at 240 kHz), 27 kΩ input impedance to system ground, and continuous ±50 V DC overvoltage protection.

The input channels can be read individually or blockwise. The control logic allows starting conversion for 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. Each input has an individual ground sense line ($\overline{\text{ADCx}}$), which must be connected to the ground of your system near the sensor or to GND at the DS2210 connector, for all ADC channels used. The ADC measures the voltage difference of (ADCx – $\overline{\text{ADCx}}$).

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 (DS2210 RTLib Reference).

I/O mapping

The following table shows the mapping of the A/D channel numbers to the related I/O pins of the 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	12-bit ADC	(ADC1 – $\overline{\text{ADC1}}$) = 0 … 20 V
	$\overline{\text{ADC1}}$	P1	67	P1B	12		
2	ADC2	P1	66	P1A	28	12-bit ADC	(ADC2 – $\overline{\text{ADC2}}$) = 0 … 20 V
	$\overline{\text{ADC2}}$	P1	68	P1A	12		
3	ADC3	P1	69	P1B	45	12-bit ADC	(ADC3 – $\overline{\text{ADC3}}$) = 0 … 20 V
	$\overline{\text{ADC3}}$	P1	71	P1B	29		
4	ADC4	P1	70	P1A	45	12-bit ADC	(ADC4 – $\overline{\text{ADC4}}$) = 0 … 20 V
	$\overline{\text{ADC4}}$	P1	72	P1A	29		
5	ADC5	P1	73	P1B	13	12-bit ADC	(ADC5 – $\overline{\text{ADC5}}$) = 0 … 20 V
	$\overline{\text{ADC5}}$	P1	75	P1B	46		
6	ADC6	P1	74	P1A	13	12-bit ADC	(ADC6 – $\overline{\text{ADC6}}$) = 0 … 20 V
	$\overline{\text{ADC6}}$	P1	76	P1A	46		
7	ADC7	P1	77	P1B	30	12-bit ADC	(ADC7 – $\overline{\text{ADC7}}$) = 0 … 20 V
	$\overline{\text{ADC7}}$	P1	79	P1B	14		
8	ADC8	P1	78	P1A	30	12-bit ADC	(ADC8 – $\overline{\text{ADC8}}$) = 0 … 20 V
	$\overline{\text{ADC8}}$	P1	80	P1A	14		
9	ADC9	P1	83	P1B	31	12-bit ADC	(ADC9 – $\overline{\text{ADC9}}$) = 0 … 20 V
	$\overline{\text{ADC9}}$	P1	85	P1B	15		
10	ADC10	P1	84	P1A	31	12-bit ADC	(ADC10 – $\overline{\text{ADC10}}$) = 0 … 20 V
	$\overline{\text{ADC10}}$	P1	86	P1A	15		
11	ADC11	P1	87	P1B	48	12-bit ADC	(ADC11 – $\overline{\text{ADC11}}$) = 0 … 20 V
	$\overline{\text{ADC11}}$	P1	89	P1B	32		
12	ADC12	P1	88	P1A	48	12-bit ADC	(ADC12 – $\overline{\text{ADC12}}$) = 0 … 20 V
	$\overline{\text{ADC12}}$	P1	90	P1A	32		
13	ADC13	P1	91	P1B	16	12-bit ADC	(ADC13 – $\overline{\text{ADC13}}$) = 0 … 20 V
	$\overline{\text{ADC13}}$	P1	93	P1B	49		

A/D Channel	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range
14	ADC14	P1	92	P1A	16	12-bit ADC	(ADC14 – $\overline{\text{ADC14}}$) = 0 … 20 V
	$\overline{\text{ADC14}}$	P1	94	P1A	49		
15	ADC15	P1	95	P1B	33	12-bit ADC	(ADC15 – $\overline{\text{ADC15}}$) = 0 … 20 V
	$\overline{\text{ADC15}}$	P1	97	P1B	17		
16	ADC16	P1	96	P1A	33	12-bit ADC	(ADC16 – $\overline{\text{ADC16}}$) = 0 … 20 V
	$\overline{\text{ADC16}}$	P1	98	P1A	17		

Related topics

Basics

Introduction to the Features of the DS2210..9

References

ADC Unit (DS2210 RTI Reference 📖)
ADC Unit (DS2210 RTLib Reference 📖)
Analog Inputs (PHS Bus System Hardware Reference 📖)
DS2210MUX_ADC_Bx (DS2210 RTI Reference 📖)

DAC Unit

Characteristics

The DAC unit contains three quad D/A converters for user output that provide 12 unipolar analog outputs (DAC1 … DAC12) with 12-bit resolution (fully monotonic) and 20 µs full-scale settling time to 1 LSB.

The DAC unit works with an internal reference of V_{REF} = 10 V, but you can apply a differential reference input V_{REF} = (DAC_REF – $\overline{\text{DAC\_REF}}$) within the range of 5 … 10 V.

Note

DAC outputs are differential outputs within the range of 0 … V_{REF} . Each output has an individual ground sense line ($\overline{\text{DACx}}$), which must be connected to the ground of your system near the actor or to GND at the DS2210 connector, for all DAC channels used. The DAC controls the voltage difference of (DACx – $\overline{\text{DACx}}$). If the external reference is used, the same applies to the $\overline{\text{DAC\_REF}}$ pin.

The outputs have lowpass output filters (1st order/–3 dB at 130 kHz). The maximum sink/source current of the DAC outputs is ±5 mA.

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 (DS2210 RTLib Reference).

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 P1, 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 – $\overline{\text{DAC\_REF}}$) = 5 … 10 V
	$\overline{\text{DAC\_REF}}$	P1	35	P1B	23	DAC external reference voltage sense line	
1	DAC1	P1	37	P1B	7	12-bit DAC/20 µs	(DAC1 – $\overline{\text{DAC1}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC1}}$	P1	39	P1B	40		
2	DAC2	P1	38	P1A	7	12-bit DAC/20 µs	(DAC2 – $\overline{\text{DAC2}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC2}}$	P1	40	P1A	40		
3	DAC3	P1	41	P1B	24	12-bit DAC/20 µs	(DAC3 – $\overline{\text{DAC3}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC3}}$	P1	43	P1B	8		
4	DAC4	P1	42	P1A	24	12-bit DAC/20 µs	(DAC4 – $\overline{\text{DAC4}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC4}}$	P1	44	P1A	8		
5	DAC5	P1	45	P1B	41	12-bit DAC/20 µs	(DAC5 – $\overline{\text{DAC5}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC5}}$	P1	47	P1B	25		
6	DAC6	P1	46	P1A	41	12-bit DAC/20 µs	(DAC6 – $\overline{\text{DAC6}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC6}}$	P1	48	P1A	25		
7	DAC7	P1	51	P1B	42	12-bit DAC/20 µs	(DAC7 – $\overline{\text{DAC7}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC7}}$	P1	53	P1B	26		
8	DAC8	P1	52	P1A	42	12-bit DAC/20 µs	(DAC8 – $\overline{\text{DAC8}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC8}}$	P1	54	P1A	26		
9	DAC9	P1	55	P1B	10	12-bit DAC/20 µs	(DAC9 – $\overline{\text{DAC9}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC9}}$	P1	57	P1B	43		
10	DAC10	P1	56	P1A	10	12-bit DAC/20 µs	(DAC10 – $\overline{\text{DAC10}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC10}}$	P1	58	P1A	43		
11	DAC11	P1	59	P1B	27	12-bit DAC/20 µs	(DAC11 – $\overline{\text{DAC11}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC11}}$	P1	61	P1B	11		
12	DAC12	P1	60	P1A	27	12-bit DAC/20 µs	(DAC12 – $\overline{\text{DAC12}}$) = 0 … V_{REF}; ±5 mA
	$\overline{\text{DAC12}}$	P1	62	P1A	11		

Related topics

References

DAC Unit (DS2210 RTI Reference 🕮)
DAC Unit (DS2210 RTLib Reference 🕮)

D/R Converter

Introduction

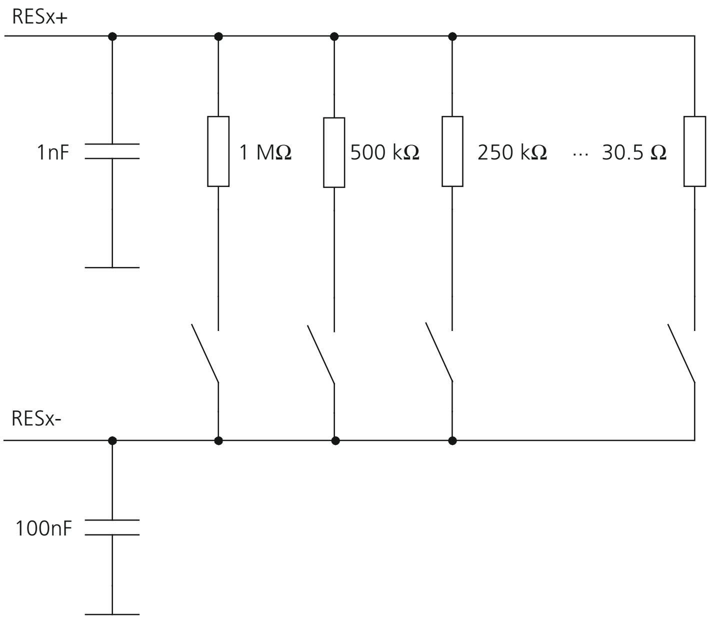
The D/R converter provides the possibility to simulate sensors that have a resistance output, for example, thermistors or RTDs for temperature measurements. You can also use the D/R converter to simulate loose connections. To generate the desired resistance your application writes a value to the D/R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

Characteristics

The D/R converter provides six independent resistance outputs with 16-bit resolution, covering a resistance range of 15 Ω … ∞. You can set the resistance value to 1 MΩ / x (with x within the range of 0 … 65535).

The maximum output power per channel is P_{max} = 250 mW. The maximum output current per channel is I_{max} = ±80 mA.

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



Several resistor channels can be connected in parallel to increase I_{max} and P_{max}, or connected in series to increase R_{max} and the resolution in higher resistance ranges.

Note

Resistors are not grounded. Terminals must stay within the range of ±10 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 ±5 V from system GND results in extra resistance error (typically 2 Ω) 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 corresponding measurement setup, refer to Function Execution Times (DS2210 RTLib Reference 🕮).

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, 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 mW; I_{max} = ±80 mA
	P2	5	P2B	18	RES1–		
2	P2	4	P2A	34	RES2+	Resistance output	P_{max} = 250 mW; I_{max} = ±80 mA
	P2	6	P2A	18	RES2–		
3	P2	7	P2B	2	RES3+	Resistance output	P_{max} = 250 mW; I_{max} = ±80 mA
	P2	9	P2B	35	RES3–		
4	P2	8	P2A	2	RES4+	Resistance output	P_{max} = 250 mW; I_{max} = ±80 mA
	P2	10	P2A	35	RES4–		
5	P2	11	P2B	19	RES5+	Resistance output	P_{max} = 250 mW; I_{max} = ±80 mA
	P2	13	P2B	3	RES5–		
6	P2	12	P2A	19	RES6+	Resistance output	P_{max} = 250 mW; I_{max} = ±80 mA
	P2	14	P2A	3	RES6–		

Related topics

Basics

Introduction to the Features of the DS2210.. 9

References

D/R Converter (DS2210 RTI Reference 🕮)
DS2210RES_Bx_Cy (DS2210 RTI Reference 🕮)

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 inputs are 12 V compatible (fully operational up to 18 V). They are protected against overvoltage higher than 18 V. After software initialization, the input threshold is set to 2.5 V. Using RTI/RTLib functions, you can set the input threshold within the range of 1 … 7 V.

The outputs have push-pull drivers running from an external source (VBAT within the range of 8 … 18 V). They are protected against overvoltage higher than 18 V. The maximum output current per channel is ±50 mA. Short circuit proof to GND and VBAT is implemented. The outputs are in their high impedance states after reset.

Using RTI/RTLib functions, you can enable or disable all outputs.

Note

Before operating the digital outputs of the bit I/O unit, an external power supply (V_{Bat}) must be connected.

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 (DS2210 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 connectors P1 and P2, as used in RTI and RTLib.

Channel Number	Signal	Connector Pin		Sub-D Pin		Description	Voltage Range
1	DIG_IN1	P1	3	P1B	34	Digital input	12 V compatible
2	DIG_IN2	P1	4	P1A	34	Digital input	12 V compatible
3	DIG_IN3	P1	5	P1B	18	Digital input	12 V compatible
4	DIG_IN4	P1	6	P1A	18	Digital input	12 V compatible
5	DIG_IN5	P1	7	P1B	2	Digital input	12 V compatible
6	DIG_IN6	P1	8	P1A	2	Digital input	12 V compatible
7	DIG_IN7	P1	9	P1B	35	Digital input	12 V compatible
8	DIG_IN8	P1	10	P1A	35	Digital input	12 V compatible
9	DIG_IN9	P1	11	P1B	19	Digital input	12 V compatible
10	DIG_IN10	P1	12	P1A	19	Digital input	12 V compatible
11	DIG_IN11	P1	13	P1B	3	Digital input	12 V compatible
12	DIG_IN12	P1	14	P1A	3	Digital input	12 V compatible
13	DIG_IN13	P1	15	P1B	36	Digital input	12 V compatible

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

Related topics

Basics

Introduction to the Features of the DS2210 9

References

Digital Inputs (PHS Bus System Hardware Reference 🕮)
Digital Outputs (PHS Bus System Hardware Reference 🕮)
DS2210DIO_SETUP_Bx (DS2210 RTI Reference 🕮)

PWM Signal Measurement

Introduction

In hardware-in-the-loop applications, PWM signal measurement is used to capture digital signals. For evaluation of frequencies and duty cycles, digital pulses have to be recorded with high speed and transferred to a processor that performs the analysis.

Note

- For boards with extended functionality, the resolution of PWM signal measurement is 16 bit. Else the resolution is 14 bit. To check whether your board supports 16 bit, refer to DS2210 Board Revision on page 119.
- The channels can be used either for frequency measurement or PWM signal measurement (supported only for boards with extended functionality).
- Due to quantization effects, you will encounter considerable deviations between the input PWM period TP 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 highest resolution (resolution values as small as possible), refer to Quantization Effects on page 118.

Characteristics

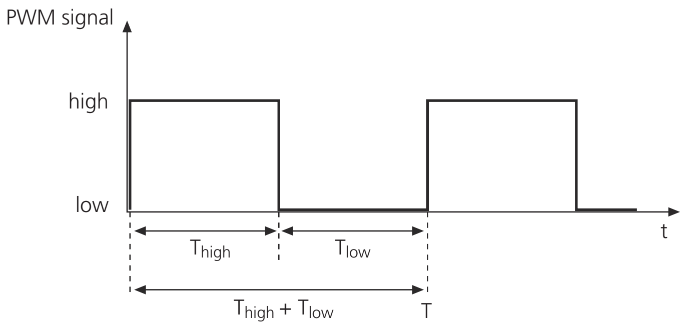
For analysis of the duty cycle, frequency, and low and high periods of PWM type signals, eight independent PWM channels are available.

The inputs are 12 V compatible (fully operational up to 18 V). After software initialization, the input threshold is set to 2.5 V. Using RTI/RTLib functions, you can set the input threshold for all digital inputs within the range of 1 V … 7 V.

Frequency and duty cycle

16-bit resolution boards Based on a 16-bit resolution, you measure the duty cycle within 0 … 100% for PWM frequencies within the range of 0.01 Hz … 20 kHz.

14-bit resolution boards Based on a 14-bit resolution, you measure the duty cycle within 0 … 100% for PWM frequencies within the range of 0.04 Hz … 20 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

16-bit resolution boards Based on a 16-bit resolution and 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	50 µs	3.27 ms	50 ns
2	400 ns	50 µs	6.55 ms	100 ns
3	800 ns	50 µs	13.1 ms	200 ns
4	1.6 µs	50 µs	26.2 ms	400 ns
5	3.2 µs	50 µs	52.4 ms	800 ns
6	6.4 µs	50 µs	104 ms	1.6 µs
7	12.8 µs	50 µs	209 ms	3.2 µs
8	25.6 µs	50 µs	419 ms	6.4 µs
9	51.2 µs	51.2 µs	838 ms	12.8 µs
10	103 µs	103 µs	1.67 s	25.6 µs
11	205 µs	205 µs	3.35 s	51.2 µs
12	410 µs	410 µs	6.71 s	103 µs
13	820 µs	820 µs	13.4 s	205 µs
14	1.64 ms	1.64 ms	26.8 s	410 µs
15	3.28 ms	3.28 ms	53.6 s	820 µs
16	6.55 ms	6.55 ms	107.3 s	1.64 ms

14-bit resolution boards Based on a 14-bit resolution and 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	50 µs	819 µs	50 ns
2	400 ns	50 µs	1.64 ms	100 ns
3	800 ns	50 µs	3.28 ms	200 ns
4	1.6 µs	50 µs	6.55 ms	400 ns
5	3.2 µs	50 µs	13.1 ms	800 ns
6	6.4 µs	50 µs	26.2 ms	1.6 µs
7	12.8 µs	50 µs	52.4 ms	3.2 µs
8	25.6 µs	50 µs	105 ms	6.4 µs
9	51.2 µs	51.2 µs	210 ms	12.8 µs
10	102 µs	102 µs	419 ms	25.6 µs
11	205 µs	205 µs	839 ms	51.2 µs
12	410 µs	410 µs	1.68 s	102 µs
13	819 µs	819 µs	3.36 s	205 µs

Range	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
14	1.64 ms	1.64 µs	6.71 s	410 µs
15	3.28 ms	3.28 ms	13.4 s	819 µs
16	6.55 ms	6.55 ms	26.8 s	1.64 ms

The maximum period applies for 0% < duty cycle < 100%. At 50% of the duty cycle, the maximum period values double.

If these ranges are exceeded, the measurement will be faulty. For values outside the practical period range, you have to consider 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 < 50 µs, T_{high} or T_{low} < 10 µs	Input noise filter may remove pulses that you want to measure.
Max. period < PWM period < 2 · max. period	Frequency measurement with restricted duty cycle.
PWM period > 2 · maximum period	Frequency 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.

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 (DS2210 RTLib Reference 🕮).

I/O mapping

The following table shows the mapping of the PWM channel numbers to the related I/O pins of the I/O connector P1, as used in RTI and RTLib. The I/O features of the DS2210 conflict with each other. Refer to Conflicting I/O Features on page 119.

PWM Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage
1	P1	21	P1B	37	PWM_IN1	PWM signal measurement	12 V compatible
2	P1	22	P1A	37	PWM_IN2	PWM signal measurement	12 V compatible
3	P1	23	P1B	21	PWM_IN3	PWM signal measurement	12 V compatible
4	P1	24	P1A	21	PWM_IN4	PWM signal measurement	12 V compatible
5	P1	25	P1B	5	PWM_IN5	PWM signal measurement	12 V compatible
6	P1	26	P1A	5	PWM_IN6	PWM signal measurement	12 V compatible

PWM Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage
7	P1	27	P1B	38	PWM_IN7	PWM signal measurement	12 V compatible
8	P1	28	P1A	38	PWM_IN8	PWM signal measurement	12 V compatible

Related topics

References

Digital Inputs (PHS Bus System Hardware Reference 🕮)
DS2210PWM2D_Bx_Cy (DS2210 RTI Reference 🕮)
PWM Signal Measurement (DS2210 RTLib Reference 🕮)

PWM Signal Generation

Characteristics

Six independent PWM outputs are available for the generation of nonnegative square-wave signals with a run-time adjustable frequency and run-time adjustable duty cycle. New values for T_{high} and T_{low} are updated immediately. An update can happen anywhere during the PWM period. So 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. This can result in a non-constant PWM period during update (i.e. actual T_{high} + T_{low}).

Note

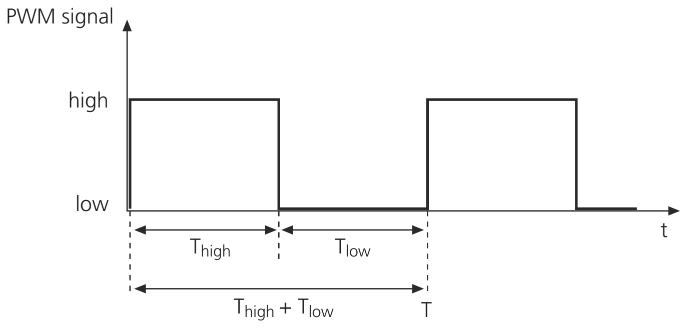
- Before operating the digital outputs of PWM signal generation, an external power supply (V_{Bat}) must be connected.
- For boards with extended functionality, the resolution of PWM signal generation is 16 bit. Else the resolution is 14 bit. To check whether your board supports 16 bit, refer to DS2210 Board Revision on page 119.
- The channels can be either used for square-wave signal generation or PWM signal generation (supported only for boards with extended functionality).
- Due to quantization effects, you will encounter considerable deviations between the input 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 highest resolution (resolution values as small as possible).

Duty cycle

16-bit resolution boards Based on a 16-bit resolution, you can set the duty cycle within 0 ... 100% for PWM frequencies within the range of 0.01 Hz ... 20 kHz.

14-bit resolution boards Based on a 14-bit resolution, you can set the duty cycle within 0 … 100% for PWM frequencies within the range of 0.04 Hz … 20 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_{high} / (T_{low} + T_{high})) is defined.



Limits and resolutions

16-bit resolution boards 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	50 µs	3.27 ms	50 ns
2	200 ns	50 µs	6.55 ms	100 ns
3	400 ns	50 µs	13.1 ms	200 ns
4	800 ns	50 µs	26.2 ms	400 ns
5	1.6 µs	50 µs	52.4 ms	800 ns
6	3.2 µs	50 µs	104 ms	1.6 µs
7	6.4 µs	50 µs	209 ms	3.2 µs
8	12.8 µs	50 µs	419 ms	6.4 µs
9	25.6 µs	50 µs	838 ms	12.8 µs
10	51.2 µs	51.2 µs	1.67 s	25.6 µs
11	103 µs	103 µs	3.35 s	51.2 µs
12	205 µs	205 µs	6.71 s	103 µs
13	410 µs	410 µs	13.4 s	205 µs
14	820 µs	820 µs	26.8 s	410 µs
15	1.64 ms	1.64 ms	53.6 s	820 µs
16	3.28 ms	3.28 ms	107.3 s	1.64 ms

14-bit resolution boards Depending on the prescaler setting, you can generate PWM signals within the following limits and resolutions:

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

Maximum period The maximum period applies for generating signals with a 0 … 100% duty cycle. At 50%, the maximum period values double. If these ranges are exceeded, the PWM signal generation will be faulty. For values outside the practical period range, you have to consider 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 < 50 µs	PWM signal will be distorted due to limited switching speed of the output circuits.
Max. period < PWM period < 2 · max. period	PWM signal with restricted duty cycle.

Outputs

The outputs have push-pull drivers running from an external source (V_{BAT} within the range of 8 … 18 V). They are protected against overvoltage higher than 18 V. The maximum output current per channel is ±50 mA. Short circuit proof to GND and VBAT is implemented. The outputs are in their high impedance states after reset.

Using RTI/RTLib functions, you can enable or disable the outputs.

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 (DS2210 RTLib Reference 📖).

I/O mapping

The following table shows the mapping of the logical PWM channel numbers to the related I/O pins of the I/O connector P2, as used in RTI and RTLib. The I/O features of the DS2210 conflict with each other. Refer to Conflicting I/O Features on page 119.

PWM Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage
1	P2	31	P2B	6	PWM_OUT1	PWM signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
2	P2	33	P2B	39	PWM_OUT2	PWM signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
3	P2	35	P2B	23	PWM_OUT3	PWM signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
4	P2	37	P2B	7	PWM_OUT4	PWM signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
5	P2	39	P2B	40	PWM_OUT5	PWM signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
6	P2	41	P2B	24	PWM_OUT6	PWM signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA

Related topics

References

Digital Outputs (PHS Bus System Hardware Reference 📖)
DS2210PWM_Bx_Cy (DS2210 RTI Reference 📖)
PWM Signal Generation (DS2210 RTLib Reference 📖)

Frequency Measurement

Characteristics

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

Note

- Frequency measurement and generation is supported only for boards with extended functionality. To check whether your board supports extended functionality, refer to DS2210 Board Revision on page 119.
- The channels can be used either for frequency measurement or PWM signal measurement.

Based on a 21-bit resolution, you can measure frequencies within the range of 0.3 mHz … 20 kHz.

The inputs are 12-V compatible (fully operational up to 18 V). The input threshold can be set within the range of 1 … 7 V. The default value is 2.5 V.

Using RTI/RTLib functions, you can set the input threshold for all digital inputs within the range of 1 … 7 V.

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 within the following ranges:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	20 kHz	50 ns
2	4.77 Hz	20 kHz	100 ns
3	2.39 Hz	20 kHz	200 ns
4	1.20 Hz	20 kHz	400 ns
5	0.60 Hz	20 kHz	800 ns
6	0.30 Hz	20 kHz	1.6 µs
7	0.15 Hz	20 kHz	3.2 µs
8	75 mHz	20 kHz	6.4 µs
9	38 mHz	19.53 kHz	12.8 µs
10	19 mHz	9.76 kHz	25.6 µs
11	10 mHz	4.88 kHz	51.2 µs
12	5.0 mHz	2.44 kHz	103 µs
13	2.5 mHz	1.22 kHz	205 µs
14	1.2 mHz	610.35 Hz	410 µs
15	0.6 mHz	305.17 Hz	820 µs
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, you have to consider the following restrictions:

Range	Restriction
Frequency < f_{min}	The measured frequency is measured as 0 Hz signal.
Frequency > f_{max}	Faulty measurement because of quantization problems, refer to Quantization Effects on page 118.

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.

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 (DS2210 RTLib Reference 📖).

I/O mapping

The following table shows the mapping of the PWM channel numbers to the related I/O pins of the I/O connector P1, as used in RTI and RTLib. The I/O features of the DS2210 conflict with each other. For details, see Conflicting I/O Features on page 119.

PWM Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage
1	P1	21	P1B	37	PWM_IN1	Frequency measurement	12 V compatible
2	P1	22	P1A	37	PWM_IN2	Frequency measurement	12 V compatible
3	P1	23	P1B	21	PWM_IN3	Frequency measurement	12 V compatible
4	P1	24	P1A	21	PWM_IN4	Frequency measurement	12 V compatible
5	P1	25	P1B	5	PWM_IN5	Frequency measurement	12 V compatible
6	P1	26	P1A	5	PWM_IN6	Frequency measurement	12 V compatible
7	P1	27	P1B	38	PWM_IN7	Frequency measurement	12 V compatible
8	P1	28	P1A	38	PWM_IN8	Frequency measurement	12 V compatible

Related topics

References

Digital Inputs (PHS Bus System Hardware Reference 🕮)
ds2210_f2d (DS2210 RTLib Reference 🕮)
DS2210F2D_Bx_Cy (DS2210 RTI Reference 🕮)

Square-Wave Signal Generation

Characteristics

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

Note

- Square-wave signal generation is supported only for boards with extended functionality. To check whether your board supports extended functionality, refer to DS2210 Board Revision on page 119.
- Before operating the digital outputs of the unit, an external power supply (V_{Bat}) must be connected.
- The channels can be used either for square-wave signal generation or PWM signal generation.

Resolution and frequency ranges Based on a 20-bit resolution of the board, you can generate square-wave signal within the frequency range of 0.3 mHz … 20 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 square-wave signals within the following frequency ranges:

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	20 kHz	50 ns
2	4.77 Hz	20 kHz	100 ns
3	2.39 Hz	20 kHz	200 ns
4	1.20 Hz	20 kHz	400 ns
5	0.60 Hz	20 kHz	800 ns
6	0.30 Hz	20 kHz	1.6 µs
7	0.15 Hz	20 kHz	3.2 µs
8	75 mHz	20 kHz	6.4 µs
9	38 mHz	20 kHz	12.8 µs
10	19 mHz	19.53 kHz	25.6 µs
11	10 mHz	9.76 kHz	51.2 µs
12	5.0 mHz	4.88 kHz	103 µs
13	2.5 mHz	2.44 kHz	205 µs
14	1.2 mHz	1.22 kHz	410 µs
15	0.6 mHz	610.35 Hz	820 µs
16	0.3 mHz	305.17 Hz	1.64 ms

If these ranges are exceeded, the square-wave signal generation will be faulty.

Outside frequency range For values outside the frequency ranges, you have to consider the following restrictions:

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

Outputs

The outputs have push-pull drivers running from an external source (VBAT within the range of 8 V… 18 V). The maximum output current per channel is ±50 mA. Short circuit proof to GND and VBAT is implemented. The outputs are in their high impedance states after reset.

Using RTI/RTLib functions, you can enable or disable the outputs.

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 (DS2210 RTLib Reference 🕮).

I/O mapping

The following table shows the mapping of the logical PWM channel numbers to the related I/O pins of the I/O connector P2, as used in RTI and RTLib. The I/O features of the DS2210 conflict with each other. Refer to Conflicting I/O Features on page 119.

PWM Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage
1	P2	31	P2B	6	PWM_OUT1	Square-wave signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
2	P2	33	P2B	39	PWM_OUT2	Square-wave signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
3	P2	35	P2B	23	PWM_OUT3	Square-wave signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
4	P2	37	P2B	7	PWM_OUT4	Square-wave signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
5	P2	39	P2B	40	PWM_OUT5	Square-wave signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA
6	P2	41	P2B	24	PWM_OUT6	Square-wave signal generation	0.4 V … (V_{Bat} – 2 V); ±50 mA

Related topics

References

Digital Outputs (PHS Bus System Hardware Reference 🕮)
Square-Wave Signal Generation (DS2210 RTI Reference 🕮)
Square-Wave Signal Generation (DS2210 RTLib Reference 🕮)

Serial Interface

Where to go from here

Information in this section

[Basics on the Serial Interface](#) 37
Provides information on board's *universal asynchronous receiver and transmitter* (UART) for performing serial asynchronous communication with external devices.

Comparing RS232 and RS422 39
The DS2210 allows you to use the RS232 or RS422 transceiver mode.

Specifying the Baud Rate of the Serial Interface 40
Provides information on the baud rate that you can specify for the board's serial interface.

Software FIFO Buffer 41
The serial interface features a memory section (software FIFO buffer) providing the UART with additional space for data storage. The buffer stores data that will be written to (transmit buffer) or was read by (receive buffer) the UART.

Basics on the Serial Interface

UART

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

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

- Selectable transceiver mode (RS232, RS422). Depending on the selected transceiver mode, the I/O board can be connected to only one external serial communication device, or to a network of devices. For details, see Comparing RS232 and RS422 on page 39.

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

  For details, see Specifying the Baud Rate of the Serial Interface on page 40.
- Selectable number of data bits, parity bit and stop bits
- Software FIFO buffer of selectable size. For details, see Software FIFO Buffer on page 41.

Serial data transfer

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

UART interrupt

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

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

For information on the interrupt handling, see DS2210 Interrupts on page 113.

RTI/RTLib support

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

- RTI: Serial Interface (DS2210 RTI Reference)
- RTLib: Serial Interface Communication (DS2210 RTLib Reference)

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 DS2210 allows you to use the RS232 or RS422 transceiver mode.

RS232 transceiver mode

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

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

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

RS422 transceiver mode

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

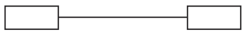
Data signals and control signals In RS422 transceiver mode, the TXD and $\overline{\text{TXD}}$ signals provide the data to be transmitted. The RXD and $\overline{\text{RXD}}$ signals provide the received data.

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

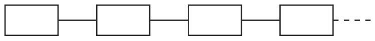
RS422 networks In RS422 networks, data is sent by one transmitter and received by up to 10 receivers. Two twisted pair cables – each providing two transmission lines – are usually used (unidirectional connections) for transmission and reception of data: one twisted pair cable for the transmitted data (TXD and $\overline{\text{TXD}}$), the other for the received data (RXD and $\overline{\text{RXD}}$).

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

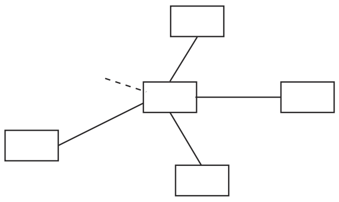
- Simple point-to-point connections



- Daisy-chain connections



- Backbone connections



Related topics

Basics

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

Specifying the Baud Rate of the Serial Interface

Oscillator frequency

The serial interface of the DS2210 is driven by an oscillator with a frequency f_{osc} = 16 MHz.

Baud rate range

Depending on the selected transceiver mode, you can specify the baud rate for serial communication with the DS2210 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 RTLib. However, the baud rate used by the DS2210 is a fraction of the oscillator frequency f_{osc}. The available baud rates can be calculated according to

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

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

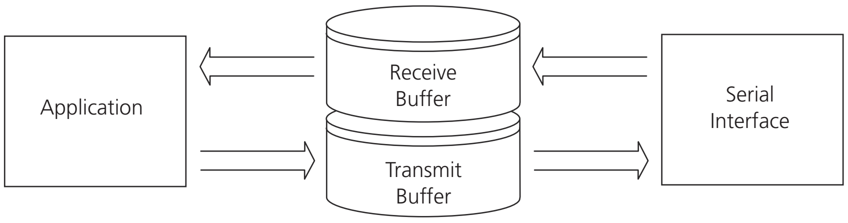
When you specify a baud rate in RTI or RTLib, the closest available baud rate is actually used for serial communication. For example, if you specify 70,000 baud as the baud rate, the baud rate used is 71,429 baud.

Software FIFO Buffer

Introduction

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

The following illustration shows the buffer principle:



Transmit buffer

Data to be transmitted usually is sent immediately.

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

Receive buffer

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

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

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

Related topics

Basics

Comparing RS232 and RS422 39

Angular Processing Unit

Introduction

The following topics provide information on the angular processing unit, which is designed to simulate engine processing core functions (crankshaft signal generation, for example).

Where to go from here

Information in this section

APU Basics.. 44
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.

APU Reference.. 59
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

Where to go from here

Information in this section

APU Overview.......45
Presenting the functional units of the APU and their interconnections.

Engine Position Phase Accumulator.......46
Explaining the unit that supplies the engine position.

Cascading I/O Boards.......47
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.......49
Explaining how the crankshaft signal is generated and what wave tables are good for.

Camshaft Signal Generator.......50
Explaining how the camshaft signals are generated and what wave tables are good for.

Spark Event Capture Unit.......51
Explaining how ignition pulses are captured.

Injection Event Capture Unit.......52
Explaining how injection pulses are captured and how the fuel amount is evaluated.

Event Capture Windows.......55
Giving information on the definition of event capture windows.

Continuous Value Capturing.......58
Explaining how you can determine ignition position, engine position and duration values continuously.

Information in other sections

Features Served by the Slave DSP.......69
Wheel speed sensor simulation and knock sensor simulation are performed by a ready-to-use application implemented on the slave DSP.

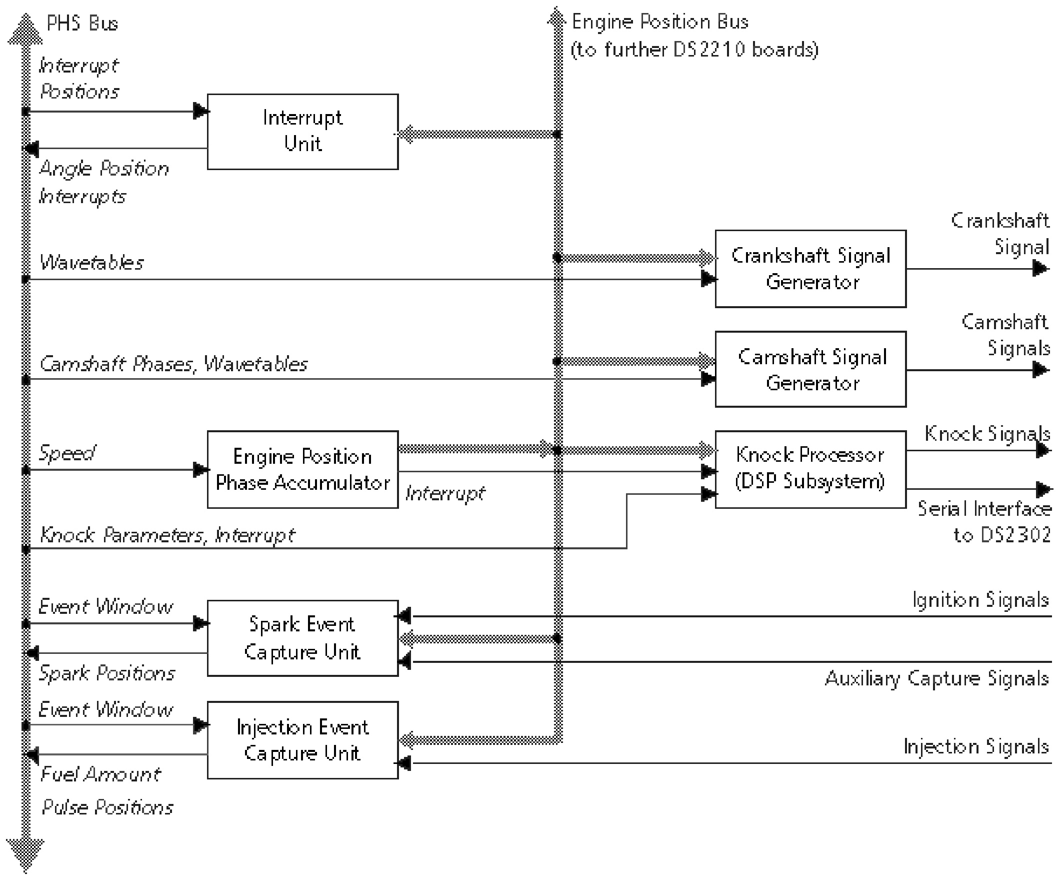
Angular Processing Unit (DS2210 RTI Reference 📖)
The angular processing unit provides RTI blocks fo accessing the APU of a DS2210.

Angular Processing Unit (APU) (DS2210 RTLib Reference 📖)
The angular processing unit provides RTLib functions for accessing the APU of a DS2210.

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.


PHS Bus
Engine Position Bus (to further DS2210 boards)
Interrupt Positions
Interrupt Unit
Angle Position Interrupts
Wavetables
Crankshaft Signal Generator
Crankshaft Signal
Camshaft Phases, Wavetables
Camshaft Signal Generator
Camshaft Signals
Speed
Engine Position Phase Accumulator
Interrupt
Knock Processor (DSP Subsystem)
Knock Signals
Serial Interface to DS2302
Knock Parameters, Interrupt
Event Window
Spark Event Capture Unit
Ignition Signals
Spark Positions
Auxiliary Capture Signals
Event Window
Injection Event Capture Unit
Injection Signals
Fuel Amount
Pulse Positions

Engine position bus

All APU components are interconnected by the engine position bus (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 with board revision DS4002-04)
- DS5001 (starting with board revision DS5001-06)
- DS5203

The boards are connected via the time-base connector. Refer to Board Overview (PHS Bus System Hardware Reference 📖).

PHS bus

In single-processor and multiprocessor PHS-bus-based systems, the Peripheral High Speed Bus (PHS bus) connects the I/O boards – a DS2210, 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 angle position interrupts for up to 8 cylinders. 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 while several interrupt positions are possible for each cylinder. For further information on interrupts, refer to Interrupts on page 113.

Cascading DS2210 boards

The angular processing unit is designed for simulating of engines with up to eight cylinders. To simulate an engine with more than eight cylinders, several DS2210 boards can be cascaded by connecting their engine position buses with the time-base connector.

You have to set one of the DS2210 boards to master mode, the others to slave mode. On DS2210 slave boards, the engine position phase accumulator is disabled and the engine position is supplied by the DS2210 master board. All other functions including initialization are not affected by the master/slave setting and must be performed for each DS2210 board individually.

Related topics

Basics

APU Reference.. 59

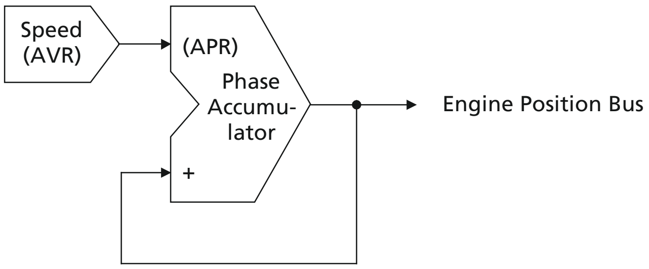
References

DS2210 HIL I/O Board (PHS Bus System Hardware Reference 📖)

Engine Position Phase Accumulator

Introduction

The engine position phase accumulator converts the run-time adjustable 16-bit Speed input parameter to engine position information and supplies a 13-bit position information to the engine position bus. Corresponding to one cycle of a four-stroke engine, the engine cycle is 0 … 720° (4π). The resulting resolution is 0.088° or 0.0015 rad. The engine position is calculated every 1 µs, that is, changes of engine speed take effect after 1 µs at the latest.



The internal resolution for position accumulation is 27 bit. The angle velocity register (AVR) provides a 16bit signed engine speed value. This value is extended to 27 bit and added to the internal 27bit angle position register (APR) every 1 µs. The most significant 13 bit 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 of engine speed given in revolutions per minute (RPM) and the AVR value is defined as follows:

RPM = 60 · 2 / (1 µs · 2^{27}) · AVR

This results in a maximum velocity of ±29297 rpm (±3068 rad/s) with a speed resolution of 0.9 rpm (0.094 rad/s).

The engine speed is converted into engine position values within the range of 0 … 719.91° (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 DS2210APU_ANG_REL_Bx (DS2210 RTI Reference 📖).

Related topics

Basics

APU Reference.. 59

Cascading I/O Boards

Introduction

You can cascade a DS2210 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 8 cylinders, because one DS2210 supports only 8 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 Board Overview (PHS Bus System Hardware Reference 📖).

Software settings

Using RTI, you configure the time-base master/slave settings via the DS2210APU_CRANK_Bx block.

Using RTLib, you configure the time-base master/slave settings via the `ds2210_mode_set` function.

You have to set one of the I/O 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 DS2210 board individually.

Related topics

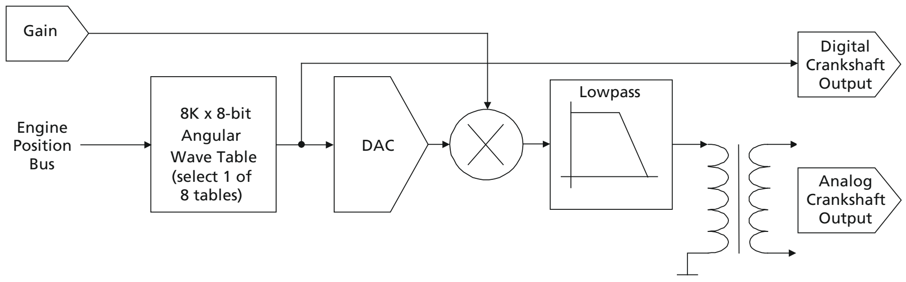
References

ds2210_mode_set (DS2210 RTLib Reference 📖)
DS2210APU_CRANK_Bx (DS2210 RTI Reference 📖)

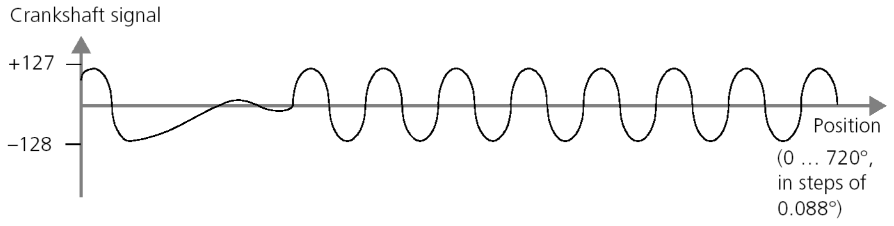
Crankshaft Signal Generator

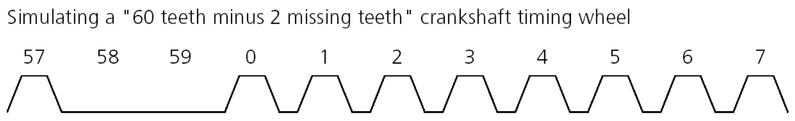
Introduction

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 lookup mechanism. The analog output is fed through output transformers. The level of the analog output can be adapted during run time to your application using a Gain parameter. The digital output generates pulse patterns that represent the sign of the analog waveform.


Gain
Engine Position Bus
8K x 8-bit Angular Wave Table (select 1 of 8 tables)
DAC
Lowpass
Digital Crankshaft Output
Analog Crankshaft Output

For each 0.088° engine position, the angular wave table defines a signal level. The resolution is 8-bit signed (–128 … +127). Due to the output transformers, the wave table data must be without a mean value. Through it, the waveform is defined. The illustration shows the analog crankshaft output of a typical waveform that simulates a "60 teeth minus 2 missing teeth" crankshaft timing wheel.




Simulating a "60 teeth minus 2 missing teeth" crankshaft timing wheel
57
58
59
0
1
2
3
4
5
6
7

The crankshaft to be simulated is defined by the waveform allocated in the angular wave table. Through it, the waveform is defined. For more details on wave tables, refer to Generating Wave Tables on page 63.

For reference information, including the I/O mapping, refer to Crankshaft Sensor Signal Generation on page 60.

Related topics

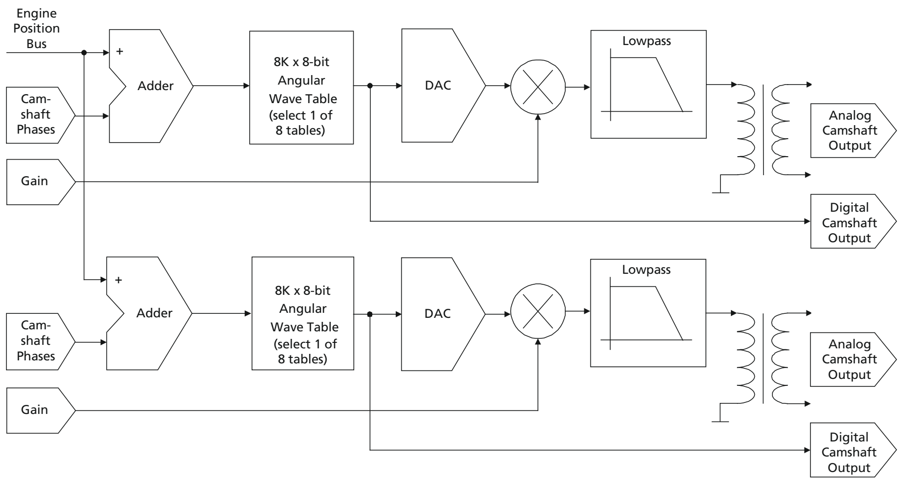
Basics

APU Reference..59

Camshaft Signal Generator

Introduction

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



For each engine position, the angular wave table defines a signal level. The signal level resolution is 8-bit signed (–128 … +127). Due to the output transformers, the wave table data must be without a mean value. Through it, the waveform is defined. For more details on wave tables, refer to Generating Wave Tables on page 63.

The phases between crankshaft signals and camshaft signals are defined by a 13-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 61.

Related topics

Basics

APU Reference 59

Spark Event Capture Unit

Introduction

The spark event capture unit is designed for ignition position measurement with 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 or active low pulses.

Continuous reading of the spark event data is available within each sample hit. Up to 32 events per channel can be read continuously out of the capture FIFO.

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

Note

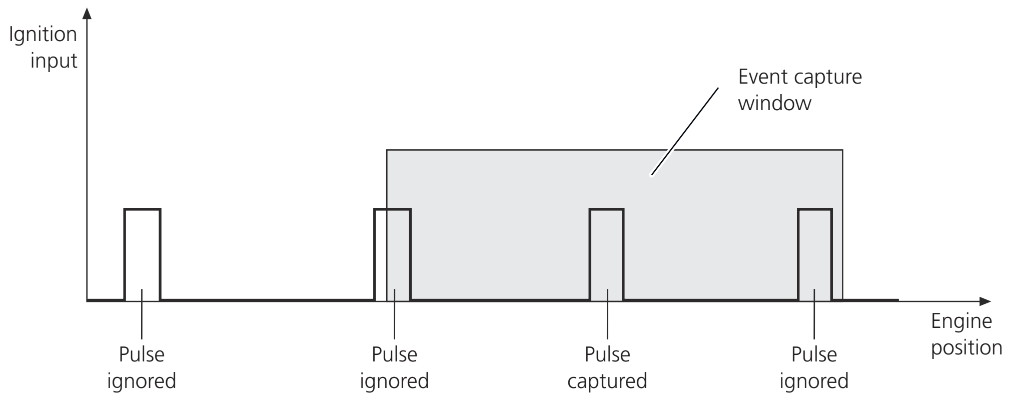
- If you do not use the channels for ignition capture, you can use them for injection capture. This is available only for boards with extended functionality. To check whether your board has extended functionality, refer to DS2210 Board Revision on page 119.
- The channels of the spark event capture unit can be read simultaneously in the event capture mode and the continuos mode by using RTLib functions. Refer to Continuous Value Capturing on page 58.

Event capture windows can be defined according to the 13-bit engine position resolution (0.088°) provided by the engine position bus (refer also to Event Capture Windows on page 55).

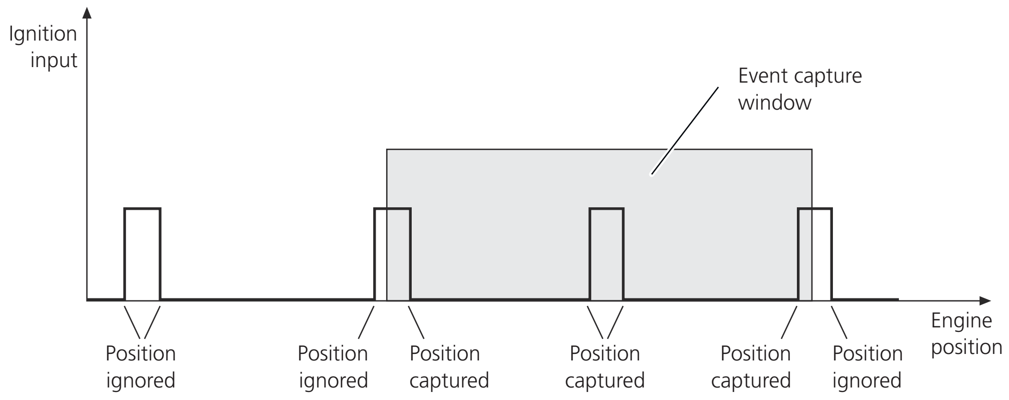
Capture modes

Two capture modes are available.

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



Multiple event capture mode Within each event capture window, the position values of all leading and trailing edges of up to 8 input pulses are evaluated. 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 64.

Related topics

Basics

APU Reference.......... 59

Injection Event Capture Unit

Introduction

The injection event capture unit is designed for injection position and fuel amount measurements with definable event capture windows for up to 16 channels. You can choose between active high or active low pulses. For fuel amount measurement (duration), two resolutions are available, 1 µs and 4 µs. Up to 8 channels per group can be captured:

- Group 1 uses the input lines INJ1 … INJ6 on P2 and PWM_IN7 and PWM_IN8 on connector P1.
- Group 2 uses the input lines IGN1 … IGN6 on P2 and AUXCAP1 and AUXCAP2 on connector P2.

Note

The following features are supported only for boards with extended functionality:

- The use of group 2 for injection capture
- Channel 7 and channel 8 of group 1
- 1 µs resolution for the duration mode

To check whether your board has extended functionality, refer to DS2210 Board Revision on page 119.

Note

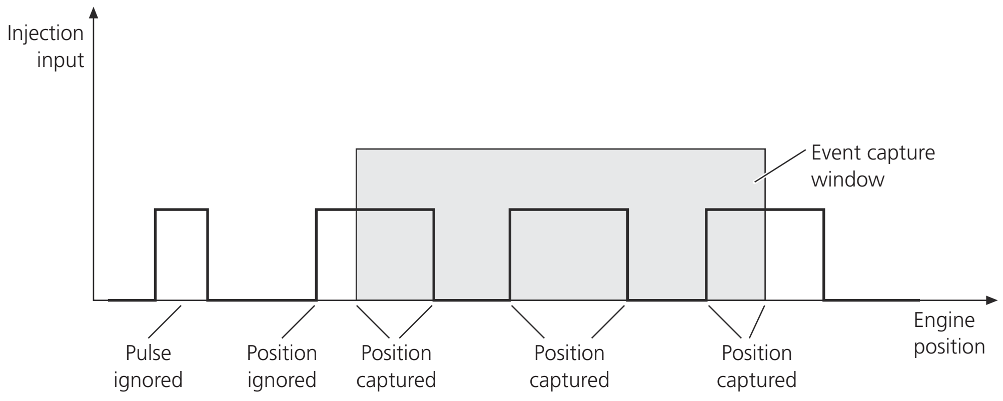
- If you use the input lines of group 2 for injection capture, ignition capture is not possible.
- The channels of the injection event capture unit can be read simultaneously in the event capture mode and the continuos mode by using RTLib functions. Refer to Continuous Value Capturing on page 58.

Event capture windows can be defined according to the 13-bit engine position resolution (0.088°) provided by the engine position bus (refer also to Event Capture Windows on page 55).

Capture modes

Two capture modes are available.

Start position/fuel amount capture mode (duration mode) Within each event capture window, the position values of up to 8 leading edges and the durations of up to 8 input pulses are evaluated (the duration is proportional to the fuel amount). Further pulses within the same event capture window are ignored. You can specify the number of expected pulses to minimize execution time.

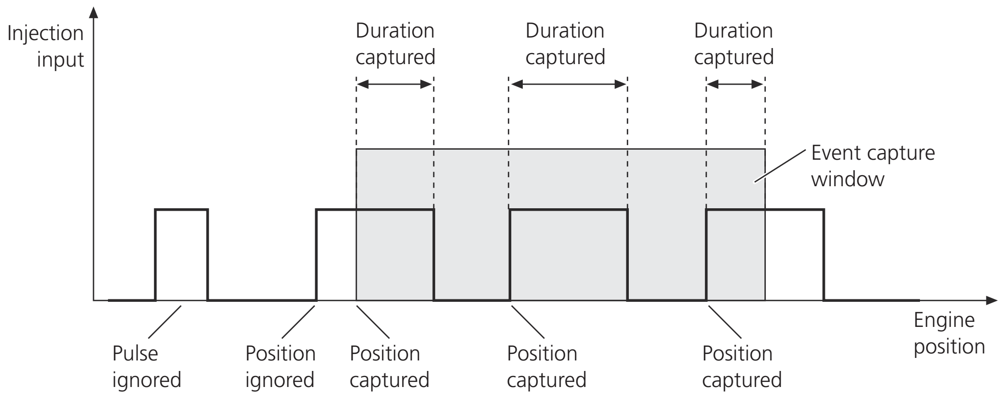


If either the leading or the trailing edge of a pulse is not in the event capture window, it is assumed that the pulse starts/ends at the corresponding border of the event capture window.

If no leading and trailing edges of the pulse are captured (permanent 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 … 719.8242°) an input pulse that overlaps the border of the event capture window is detected as two separate pulses.

Start/end position capture mode (position mode) Within each event capture window, the position values of up to 8 leading and trailing edges of up to 8 input pulses are evaluated. You can specify the number of expected pulses to minimize execution time.



If either the leading or the trailing edge of a pulse is not in the event capture window, the position value of the corresponding event capture window border is counted as an edge.

If no leading and trailing edges of the pulse are captured (permanent 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.

For reference information including the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement on page 65.

Related topics

Basics

APU Basics.. 44
APU Reference.. 59

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. 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 a spark event capture, you can use the single event capture mode or the multiple event capture mode (refer to Spark Event Capture Unit on page 51).

For an injection event capture, you can use the start position/fuel amount capture mode (duration mode) or the start/end position capture mode (position mode): refer to Injection Event Capture Unit on page 52.

Continuous value capture

For spark event and injection event capture, you can continuously measure up to 32 events per channel via the capture FIFO buffer. Using this mode, the event capture window can cover the whole engine cycle of 0 … 720° (refer to Continuous Value Capturing on page 58).

Setting event capture windows

You can use RTI (DS2210APU_IGN_Bx, DS2210APU_INJ_Bx_Gy, DS2210APU_AUXCAP_Bx_Cy, DS2210APU_IGNCONT_Bx, DS2210APU_INJCONT_Bx_Gy and DS2210APU_AUXCAP_Bx_Cy) or RTLib (ds2210_event_window_set) functions to define event capture windows.

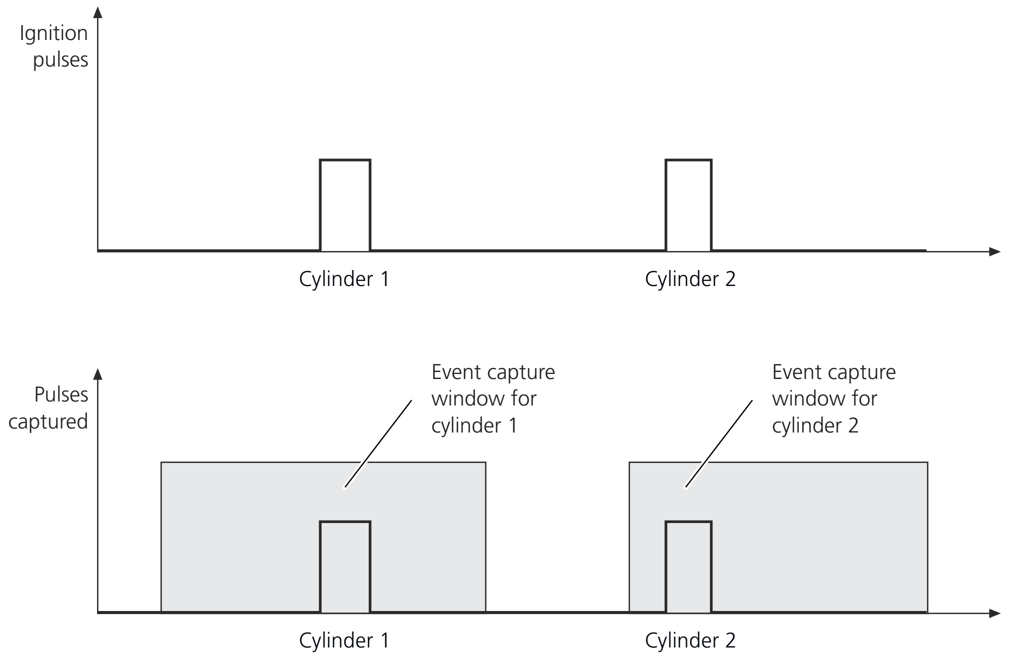
Restrictions

For event capture windows the following restrictions apply:

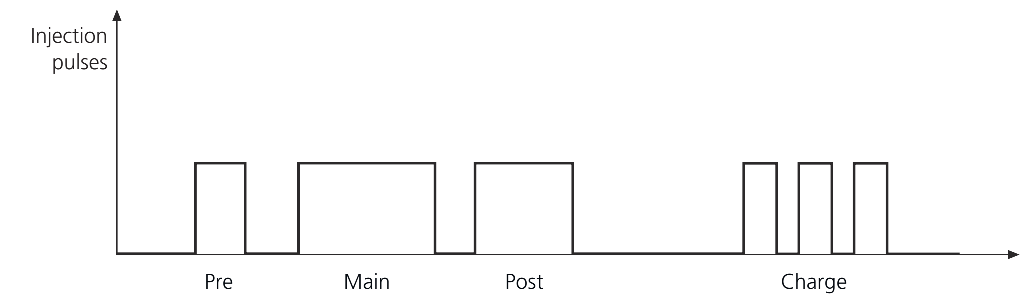
- An event window must not cover the whole engine cycle of 0 … 720°. The maximum width of an event window is 719.824° (= 12.5631 rad).
- The minimum width of an event window is 0.176° (= 0.003 rad).
- If you specify the same value for the start and the end position no event capture window will be set and an error message is issued when using RTI.
- The minimum distance between two event windows is two engine position steps ($2 \cdot 4\pi/8192$).

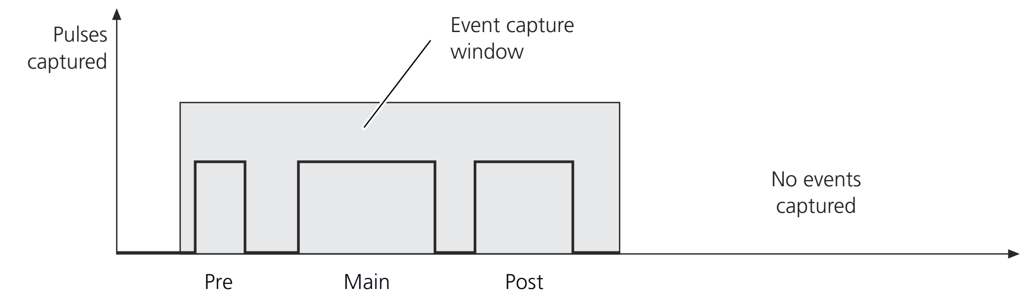
Example

Double spark ignition There are 2 ignition pulses per coil but only 1 signal for the 2 cylinders. The cylinder is identified by its engine angle. With event capture windows you can specify possible ignition pulse positions.

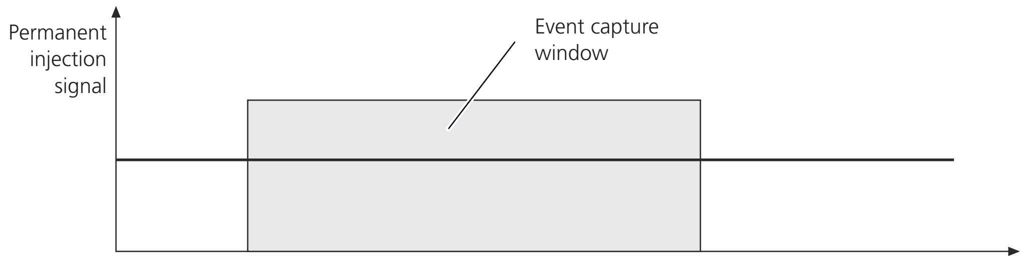

Ignition pulses
Cylinder 1
Cylinder 2
Pulses captured
Event capture window for cylinder 1
Event capture window for cylinder 2
Cylinder 1
Cylinder 2

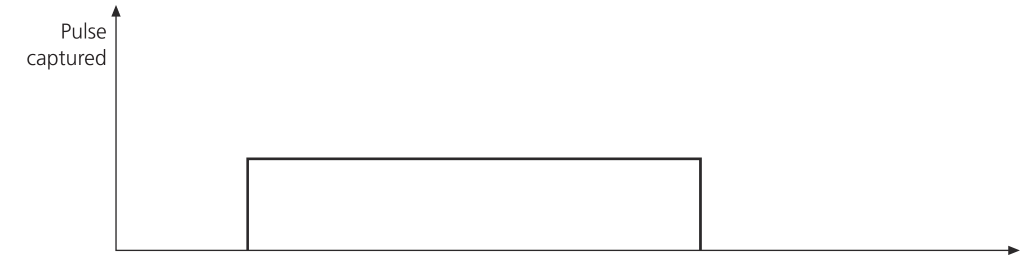
Common rail injection For several injection pulses the start positions of the pulses and the pulse durations can be captured. Additional charge pulses can be ignored with the appropriate length of the event capture window.





Permanent injection For a permanent injection signal the width of the event capture window will be captured.





Related topics

References

Event Capture Windows (DS2210 RTLib Reference 🕮)

Continuous Value Capturing

Introduction

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

Spark event capture

In single event mode, up to 32 position values per channel of the leading edge of the first input pulse within the capture window is read. In multiple event capture mode, up to 32 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 events per channel can be read within one sample hit. In duration mode, the position values of all leading edges and the duration of up to 32 events per channel within one sample hit can be read (refer to Injection Event Capture Unit on page 52).

Auxiliary event capture

2 channels are available to measure various positions. In single event mode, up to 32 position values of the leading edge of the first input pulse are evaluated. In multiple event mode, up to 32 position values of leading and trailing edges can be captured. Auxiliary event capture is similar to the spark event capture (refer to Spark Event Capture Unit on page 51).

Sample hit

The sample hit is defined by the sample time of your model, for example, 1 ms.

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 continuos mode by using RTLib functions. Refer to Angular Processing Unit (APU) (DS2210 RTLib Reference 📖).

Related topics

Basics

Event Capture Windows........ 55
Injection Event Capture Unit........ 52
Injection Pulse Position and Fuel Amount Measurement........ 65
Spark Event Capture........ 64
Spark Event Capture Unit........ 51

APU Reference

Introduction

The following topics 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........60
The crankshaft signal generator provides one analog crankshaft output and one digital crankshaft output.

Camshaft Sensor Signal Generation........61
The camshaft signal generator provides two analog camshaft outputs and two corresponding digital outputs. The digital outputs generate pulse patterns that represent the sign of the corresponding analog waveforms.

Wave Table Basics........62
Using wave tables, you can define specific crankshaft waveforms to simulate different signal types.

Generating Wave Tables........63
Wave tables must be supplied as MAT files. In MATLAB, you can create waveforms by using standard functions or you can import data measured at a real engine.

Spark Event Capture........64
The spark event capture unit provides 6 digital ignition inputs and 2 digital auxiliary capture inputs.

Injection Pulse Position and Fuel Amount Measurement........65
The injection event capture unit provides 16 digital injection inputs are split into 2 groups for injection pulse position and fuel amount measurement.

Information in other sections

APU Basics........44
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.

Features Served by the Slave DSP........69
Wheel speed sensor simulation and knock sensor simulation are performed by a ready-to-use application implemented on the slave DSP.

Crankshaft Sensor Signal Generation

Characteristics

The crankshaft signal generator provides one analog crankshaft output and one digital crankshaft output. Both the analog and the digital output carry out the same crankshaft information. The digital output generates pulse patterns that represent the sign of the analog waveform.

For basic information on how the crankshaft sensor signal generation works, refer to Crankshaft Signal Generator on page 49.

Using wave tables, you can define specific crankshaft waveforms to simulate different crankshaft types. The DS2210 can load 8 different wave tables for crankshaft sensor signal generation. The waveform 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 Basics on page 62.

The digital output has a push-pull driver running from an external source (VBAT within the range of 8 … 18 V). It is protected against overvoltage higher than 18 V. The maximum output current is ±50 mA. Short circuit proof to GND and VBAT is implemented. The digital outputs are in their high impedance states after reset. Using RTI, the output is enabled if the block is part of the model. Using RTLib functions, you can enable or disable the output.

The analog output signal is fed through an output transformer with a maximum physical signal level of 40 V_{pp}. Within this range, you can change the amplitude of the whole waveform at run time. The analog output can be enabled or disabled.

Note

Transformers do not transmit a DC voltage. In the frequency range below 500 Hz, the maximum output voltage is proportional to the frequency. The nominal operating range of the transformers is 500 Hz … 20 kHz.

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 (DS2210 RTLib Reference).

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 – 2 V); ±50 mA
Analog	P2	68	P2A	12	CRANK+	Crankshaft waveform output	(CRANK+ – CRANK–) = ±20 V
	P2	70	P2A	45	CRANK–		

Related topics

References

Crankshaft Sensor Signal Generation (DS2210 RTLib Reference 🕮)
Digital Outputs (PHS Bus System Hardware Reference 🕮)
DS2210APU_CRANK_Bx (DS2210 RTI Reference 🕮)
DS2210DIO_SETUP_Bx (DS2210 RTI Reference 🕮)
Power Supply Outputs (PHS Bus System Hardware Reference 🕮)
Transformer Outputs (APU and Slave DSP) (PHS Bus System Hardware Reference 🕮)

Camshaft Sensor Signal Generation

Characteristics

The camshaft signal generator provides two analog camshaft outputs and two corresponding digital outputs. The digital outputs generate pulse patterns that represent the sign of the corresponding analog waveforms.

For basic information on how camshaft sensor signal generation works, refer to Camshaft Signal Generator on page 50.

Using wave tables you can define specific camshaft waveforms to simulate different camshaft types. The DS2210 is capable of loading eight different wave tables for each camshaft output. The waveform to be generated can be switched during run time. For detailed information on wave tables, refer to Wave Table Basics on page 62.

The digital outputs have push-pull drivers running from an external source (VBAT within the range of 8 … 18 V). They are protected against overvoltage higher than 18 V. The maximum output current per channel is ±50 mA. Short circuit proof to GND and VBAT is implemented. The outputs are in their high impedance states after reset.

Using RTI/RTLib functions, you can enable or disable the outputs.

The analog output signals are fed through output transformers with a maximum physical amplitude of 40 V_{pp}. Within this range, you can change the amplitude of the whole waveform at run time. The analog outputs can be enabled or disabled.

Note

Transformers do not transmit a DC voltage. In the frequency range below 500 Hz, the maximum output voltage is proportional to the frequency. The nominal operating range of the transformers is 500 Hz … 20 kHz with full output voltage.

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 (DS2210 RTLib Reference 🕮).

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.

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 – 2 V); ±50 mA
	Analog	P2	72	P2A	29	CAM1+	Camshaft waveform output	(CAM1+ – CAM1–) = ±20 V
		P2	74	P2A	13	CAM1–		
2 (B)	Digital	P2	49	P2B	9	CAM2_DIG	Digital camshaft output	0.4 V … (VBAT – 2 V); ±50 mA
	Analog	P2	76	P2A	46	CAM2+	Camshaft waveform output	(CAM2+ – CAM2–) = ±20 V
		P2	78	P2A	30	CAM2–		

Related topics

References

Crankshaft Sensor Signal Generation (DS2210 RTLib Reference 🕮)
Digital Outputs (PHS Bus System Hardware Reference 🕮)
DS2210APU_CRANK_Bx (DS2210 RTI Reference 🕮)
DS2210DIO_SETUP_Bx (DS2210 RTI Reference 🕮)
Power Supply Outputs (PHS Bus System Hardware Reference 🕮)
Transformer Outputs (APU and Slave DSP) (PHS Bus System Hardware Reference 🕮)

Wave Table Basics

Wave table structure

According to the 13-bit resolution of the engine position, each wave table defines the signal waveform for 2^{13} = 8192 consecutive engine positions within the range of 0 … 720° (4π). The signal level resolution is 8-bit signed (–128 … +127). 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 waveform.

Look-up mechanism

Wave tables are downloaded to the processor board together with the real-time application. The real-time processor transfers the wave tables to the DS2210 where they are stored in the memories of the Crankshaft Signal Generator and Camshaft Signal Generator.

At run time, the wave table look-up mechanism outputs the signal value, which is defined for the current engine position (every 1 µs).

Related topics

Basics

APU Basics....................44
Camshaft Signal Generator....................50
Crankshaft Signal Generator....................49
Generating Wave Tables....................63

Generating Wave Tables

Introduction

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

You find example wave tables in the `<RCP_HIL_InstallationPath>\Demos\DS100x\IOBoards\Ds2210\APU\Wavetables` folder of your dSPACE installation.

Using MATLAB

In the folder `<RCP_HIL_InstallationPath>matlab\rtlib100x\tools` of your dSPACE installation, you also find the M-file `Mat2asm2210.m`. It collects the specified wave table MAT files into a common MAT file, and converts it to an assembly source file. For each board, up to 24 wave tables can be loaded (8 for each signal generation). Use the RTLib functions `ds2210_crank_table_load` or `ds2210_cam_table_load` to load wave tables to the real-time hardware.

Using RTI

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

Related topics

References

DS2210APU_CAM_Bx_Cy (DS2210 RTI Reference 🕮)
DS2210APU_CRANK_Bx (DS2210 RTI Reference 🕮)

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 51 and Event Capture Windows on page 55.

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 8 pulses are allowed.

You can read up to 32 events out of the capture FIFO continuously within one sample hit. You can use the whole event capture window within the range of 0 … 720°.

The inputs are 12 V compatible. They are protected against overvoltage higher than 18 V. After software initialization, the input threshold is set to 2.5 V. Using RTI/RTLib functions, you can set the input threshold within the range of 1 … 7 V.

Note

RTLib functions and the DS2210APU_AUXCAP_Bx_Cy and DS2210APU_AUXCAPCONT_Bx_Cy blocks refer to the absolute engine position.
The DS2210APU_IGN_Bx and DS2210APU_IGNCONT_Bx blocks relate to the top dead center (TDC).

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 (DS2210 RTLib Reference 📖).

I/O mapping

The following table shows the mapping of the cylinder/channel numbers to the related I/O pins of the DS2210 I/O connectors P1 and P2, as used in RTI and RTLib. The I/O features of the DS2210 conflict with each other. For details, see Conflicting I/O Features on page 119.

RTI Cylinder/ Channel	RTLib Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage Range
1	1	P2	19	P2B	4	IGN1	Ignition capture	12 V compatible
2	2	P2	20	P2A	4	IGN2	Ignition capture	12 V compatible
3	3	P2	23	P2B	21	IGN3	Ignition capture	12 V compatible
4	4	P2	24	P2A	21	IGN4	Ignition capture	12 V compatible
5	5	P2	27	P2B	38	IGN5	Ignition capture	12 V compatible
6	6	P2	28	P2A	38	IGN6	Ignition capture	12 V compatible

RTI Cylinder/ Channel	RTLib Channel	Connector Pin		Sub-D Pin		Signal	Description	Voltage Range
1	7	P1	34	P1A	39	AUXCAP1	Auxiliary capture	12 V compatible
2	8	P1	36	P1A	23	AUXCAP2	Auxiliary capture	12 V compatible

Related topics

Basics

Spark Event Capture Unit.. 51

References

Digital Inputs (PHS Bus System Hardware Reference 🕮)
DS2210APU_AUXCAP_Bx_Cy (DS2210 RTI Reference 🕮)
DS2210APU_AUXCAPCONT_Bx_Cy (DS2210 RTI Reference 🕮)
DS2210APU_CAM_Bx_Cy (DS2210 RTI Reference 🕮)
DS2210APU_IGN_Bx (DS2210 RTI Reference 🕮)
DS2210APU_IGNCONT_Bx (DS2210 RTI Reference 🕮)
DS2210DIO_SETUP_Bx (DS2210 RTI Reference 🕮)
Spark Event Capture (DS2210 RTLib Reference 🕮)

Injection Pulse Position and Fuel Amount Measurement

Introduction

The injection event capture unit provides 16 digital injection inputs are 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 55 and Injection Event Capture Unit on page 52.

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 8 pulses are allowed.

You can read up to 32 events out of the capture FIFO continuously within one sample hit. You can use the whole event capture window within the range of 0 … 720°.

Note

The following features are supported only for boards with extended functionality:
- The use of group 2 for injection capture
- Channel 7 and channel 8 of group 1
- 1 µs resolution for the duration mode

To check which revision your board is, refer to DS2210 Board Revision on page 119.

The digital inputs are 12 V compatible (operational up to 18 V). After software initialization, the input threshold is set to 2.5 V. Using RTI/RTLib functions, you can set the input threshold within the range of 1 … 7 V.

Note

RTLib functions refer to the absolute engine position. The DS2210APU_INJ_Bx_Gy and DS2210APU_INJCONT_Bx_Gy blocks relate to the top dead center (TDC).

I/O mapping

The following table shows the mapping of the cylinder/channel numbers to the related I/O pins of the DS2210 I/O connector P1, as used in RTI and RTLib. The I/O features of the DS2210 conflict with each other. Refer to Conflicting I/O Features on page 119.

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 V compatible
2	2	P2	18	P2A	20	INJ2	Injection capture	12 V compatible
3	3	P2	21	P2B	37	INJ3	Injection capture	12 V compatible
4	4	P2	22	P2A	37	INJ4	Injection capture	12 V compatible
5	5	P2	25	P2B	5	INJ5	Injection capture	12 V compatible
6	6	P2	26	P2A	5	INJ6	Injection capture	12 V compatible
7	7	P1	27	P1B	38	INJ7 (PWM_IN7)	Injection capture	12 V compatible
8	8	P1	28	P1A	38	INJ8 (PWM_IN8)	Injection capture	12 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 V compatible
2	2	P2	20	P2A	4	IGN2	Ignition capture	12 V compatible
3	3	P2	23	P2B	21	IGN3	Ignition capture	12 V compatible
4	4	P2	24	P2A	21	IGN4	Ignition capture	12 V compatible
5	5	P2	27	P2B	38	IGN5	Ignition capture	12 V compatible
6	6	P2	28	P2A	38	IGN6	Ignition capture	12 V compatible
7	7	P1	34	P1B	39	AUXCAP1	Auxiliary capture	12 V compatible
8	8	P1	36	P1A	23	AUXCAP2	Auxiliary capture	12 V compatible

Related topics

Basics

Spark Event Capture Unit.. 51

References

Digital Inputs (PHS Bus System Hardware Reference 🕮)
DS2210APU_INJ_Bx_Gy (DS2210 RTI Reference 🕮)
DS2210DIO_SETUP_Bx (DS2210 RTI Reference 🕮)
Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference 🕮)

Features Served by the Slave DSP

Introduction

The following topics provide information on the features served by the slave DSP of the DS2210.

Where to go from here

Information in this section

Slave DSP TMS320C31 Basics....69
Providing general information on the slave DSP.

Knock Sensor Simulation....70
Providing information on the ready-to-use application (knock processor) implemented on the slave DSP. This application is regarded as part of the angular processing unit.

Wheel Speed Sensor Simulation....73
Providing information on the ready-to-use application implemented on the slave DSP. This application is regarded as part of the sensor and actuator interface.

Slave DSP TMS320C31 Basics

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.

Output circuits

The resolution of the four analog outputs is 12 bit. For information on the I/O circuits, refer to Digital Outputs (PHS Bus System Hardware Reference 🕮).

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 the *DS2210 RTLib Reference*.
Download slave applications	The slave DSP applications are downloaded to the master processor board together with the master application, and then transferred to the DS2210 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 µs). The master processor board can interrupt the slave DSP by writing to a predefined location of the slave DSP's dual-port memory (DPMEM).
Emulation/debug port	The emulation/debug port connector P6 can be used for slave DSP debugging. This requires additional software and hardware.
Serial interface	In addition to the analog outputs, the slave DSP provides a serial interface (DS2210 connector P5) that can be used for connecting a DDS board (DS2302). For the location of the connector, refer to Serial Interface (PHS Bus System Hardware Reference 🕮).
TMS320C31	For detailed information on the TMS320C31, refer to the Texas Instruments web site at http://www.ti.com and search for TMS320C31.

Related topics

References

DS2210APU_INT_Bx_Iy (DS2210 RTI Reference 🕮)
DS2210SL_KNSG_Bx_Cy (DS2210 RTI Reference 🕮)
DS2210SL_WSSG_Bx_Cy (DS2210 RTI Reference 🕮)
Interrupts (DS2210 RTLib Reference 🕮)
Slave DSP Access Functions (DS2210 RTLib Reference 🕮)
Slave DSP Functions and Macros (DS2210 RTLib Reference 🕮)

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.

Knock sensor simulation is triggered by the engine position, which is supplied by the engine position bus (see APU Overview on page 45). Like a knock sensor in a real engine 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 can be changed at run time by using RTI or RTLib functions.

Note

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

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

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

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

Knock length Specifies the end of signal generation. The signal generation ends after this length (given in degrees).

Amplitude Specifies the amplitude a of the cosine signal in V_{pp}.

Frequency Specifies the frequency f of the cosine signal in Hz.

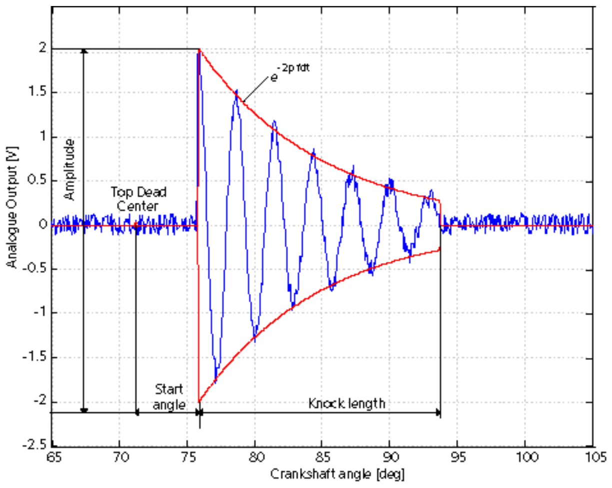
Damping Specifies the damping factor of the cosine signal.

Knock number Specifies 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 Specifies after how many engine cycles the knock signal generation will start again for the given cylinder.

Noise Specifies the amplitude of the noise in V_{pp}. The additional noise signal can be selected independently for knock sensor 1 and 2.

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



Outputs

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

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 (DS2210 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. The I/O features of the DS2210 conflict with each other. Refer to Conflicting I/O Features on page 119.

Channel Number	Connector Pin		Sub-D Pin		Signal	Description	Voltage Range/Output Current
1	P2	63	P2B	44	C31-Waveform 0+	Knock sensor or wheel speed sensor signal	(C31-Waveform 0+ – C31-Waveform 0–) = ±20 V
	P2	65	P2B	28	C31-Waveform 0–		
2	P2	67	P2B	12	C31-Waveform 1+	Knock sensor or wheel speed sensor signal	(C31-Waveform 1+ – C31-Waveform 1–) = ±20 V
	P2	69	P2B	45	C31-Waveform 1–		
3	P2	71	P2B	29	C31-Waveform 2+	Knock sensor or wheel speed sensor signal	(C31-Waveform 2+ – C31-Waveform 2–) = ±20 V
	P2	73	P2B	13	C31-Waveform 2–		
4	P2	75	P2B	46	C31-Waveform 3+	Knock sensor or wheel speed sensor signal	(C31-Waveform 3+ – C31-Waveform 3–) = ±20 V
	P2	77	P2B	30	C31-Waveform 3–		

Related topics

References

DS2210SL_KNSG_Bx_Cy (DS2210 RTI Reference 📖)
Transformer Outputs (APU and Slave DSP) (PHS Bus System Hardware Reference 📖)

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.

Note

You cannot use wheel speed sensor simulation and Knock Sensor Simulation at the same time.

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

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

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

Where

a	is the amplitude
f_{Wheel}	is the wheel speed signal frequency which is calculated as follows:
	$f_{Wheel} = speed \cdot teeth \cdot 1[min]/60[s]$
	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}. The outputs can be enabled or disabled.

Note

- Transformers do not transmit a DC voltage.
- In the frequency range below 500 Hz, the maximum output voltage is proportional to the frequency.
- The nominal operating range of the transformers is 500 Hz … 20 kHz.

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 (DS2210 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. The I/O features of the DS2210 conflict with each other. Refer to Conflicting I/O Features on page 119.

Channel Number	Connector Pin		Sub-D Pin		Signal	Description	Voltage Range/Output Current
1	P2	63	P2B	44	C31-Waveform 0+	Knock sensor or wheel speed sensor signal	(C31-Waveform 0+ – C31-Waveform 0–) = ±20 V
	P2	65	P2B	28	C31-Waveform 0–		
2	P2	67	P2B	12	C31-Waveform 1+	Knock sensor or wheel speed sensor signal	(C31-Waveform 1+ – C31-Waveform 1–) = ±20 V
	P2	69	P2B	45	C31-Waveform 1–		
3	P2	71	P2B	29	C31-Waveform 2+	Knock sensor or wheel speed sensor signal	(C31-Waveform 2+ – C31-Waveform 2–) = ±20 V
	P2	73	P2B	13	C31-Waveform 2–		
4	P2	75	P2B	46	C31-Waveform 3+	Knock sensor or wheel speed sensor signal	(C31-Waveform 3+ – C31-Waveform 3–) = ±20 V
	P2	77	P2B	30	C31-Waveform 3–		

Related topics

References

DS2210SL_WSSG_Bx_Cy (DS2210 RTI Reference 🕮)
Slave DSP Access Functions (DS2210 RTLib Reference 🕮)

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 76
Explains how to set up a CAN controller to use a dSPACE board with CAN bus interface.

Using the RTI CAN MultiMessage Blockset 89
Provides information on using the RTI CAN MultiMessage Blockset.

CAN Signal Mapping 112
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 129
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....................76
The CAN controller performs serial communication according to the CAN protocol. You must configure the CAN controller according to the application.

CAN Transceiver Types....................78
The way in which CAN messages are transmitted on a CAN bus depends on the CAN transceiver used.

Defining CAN Messages....................82
The dSPACE CAN software lets you easily define CAN messages to be transmitted or received.

Implementing a CAN Interrupt....................84
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....................84
RTI CAN Blockset provides two concepts for receiving CAN messages.

Removing a CAN Controller (Go Bus Off)....................86
You can remove the CAN controller that is being used from the bus when you use several CAN controllers.

Getting CAN Status Information....................87
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.

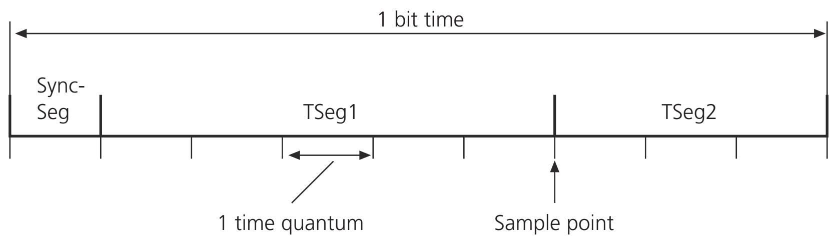
Advanced configuration (bit timing parameters)

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 🕮).

Related topics

References

RTICAN CONTROLLER SETUP (RTI CAN Blockset Reference 🕮)

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
▪ DS2202 ▪ DS2210 ▪ DS2211	ISO11898
DS4302	The following transceiver types are supported: ▪ ISO11898 ▪ RS485 ▪ C252 ▪ Piggyback[1)] **Note** 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.
MicroAutoBox II	The following transceiver types are supported: ▪ ISO11898 ▪ ISO11898-6[2), 3)]
MicroLabBox	The following transceiver types are supported: ▪ ISO11898 ▪ ISO11898-6[2)]

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 kBd. 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 kBd. 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 kΩ resistor for many connected bus members, or a 1.6 kΩ 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 🕮).

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 🕮).

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

Defining CAN Messages

Introduction

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

Message types

You can define a message as a TX, RX, RQ, or RM message:

Message Type	Description
Transmit (TX)	This message is transmitted with a specific identifier. A TX message contains up to 8 bytes of data.
Receive (RX)	This message is *not* 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 *remote transmission request*1). An RQ message is transmitted with a specific identifier to request data. An RQ message does not contain data.

Message Type	Description
Remote (RM)	Second part of a *remote transmission request*[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 📖).

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 🕮)

Removing a CAN Controller (Go Bus Off)

Introduction

If you use several CAN controllers, you can remove the one currently in use from the bus. Data transfer from the master to the slave processor is then stopped. You can select the CAN controller you want to remove from the bus via the **RTICAN Go Bus Off** block.

You can restart data transfer with another CAN controller or the same one with the **RTICAN Bus Off Recovery** block.

RTI support

- To remove a CAN controller from the bus, use the **RTICAN Go Bus Off** block. Refer to RTICAN Go Bus Off (RTI CAN Blockset Reference 🕮).
- To restart data transfer, use the **RTICAN Bus Off Recovery** block. Refer to RTICAN Bus Off Recovery (RTI CAN Blockset Reference 🕮).

Related topics

References

RTICAN Bus Off Recovery (RTI CAN Blockset Reference 🕮)
RTICAN Go Bus Off (RTI CAN Blockset Reference 🕮)

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.

Number of ...	Description
Cyclic redundancy check (CRC) errors	Each time the CRC checksum of the received message is incorrect, the appropriate counter is incremented. The EML also checks the CRC checksum of each message (see 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 *RX lost error* 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 (DS2210 RTLib Reference 📖)
RTICAN Status (RTI CAN Blockset Reference 📖)

Using the RTI CAN MultiMessage Blockset

Introduction

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

Where to go from here

Information in this section

Basics on the RTI CAN MultiMessage Blockset..89
Gives you an overview of the features of the RTI CAN MultiMessage Blockset.

Basics on Working with CAN FD..94
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..............................99
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..105
Large CAN message bundles can be managed by a single Simulink block provided by the RTI CAN MultiMessage Blockset.

Manipulating Signals to be Transmitted...108
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 94.

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 🕮).

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

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:

- Trigger options
- Ports and displays
- Message ID and length adjustable during run time

The following features are not supported:

- Checksum generation
- Custom signal manipulation

Capturing messages

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.

The captured messages can be made available as outports of the MainBlock or in the TRC file.

CAN partial networking

With CAN partial networking, you can set selected ECUs in a network to sleep mode or shut them down if they do not have to run continuously. Wake-up messages then activate specific ECUs as and when required, and for as long as required.

Note

Partial networking is possible for the following dSPACE real-time hardware:

- MicroAutoBox II equipped with the DS1513 I/O Board
- MicroLabBox
- dSPACE hardware that supports working with CAN FD messages and that is equipped with DS4342 CAN FD Interface Modules, such as:
  - PHS-bus-based systems (DS1006 or DS1007 modular systems) with DS4505 Interface Board
  - MicroAutoBox II variants with DS1507
  - MicroAutoBox II variants with DS1514

The RTI CAN MultiMessage Blockset lets you specify the CAN partial networking wake-up messages by filtering message IDs and message data.

The CAN transceiver of the dSPACE real-time hardware is switched to sleep mode via the real-time application. You can use partial networking messages to wake up dSPACE real-time hardware after its CAN transceiver is switched to sleep mode.

(Relevant for MicroAutoBox II only) If a MicroAutoBox II's CAN transceiver is woken up via a partial networking message, the MicroAutoBox II behaves as if it was powered up manually. Depending on where the real-time application is loaded (flash memory or RAM), the MicroAutoBox II starts the application or waits for further input.

For more information, refer to Partial Networking Page (RTICANMM ControllerSetup) (RTI CAN MultiMessage Blockset Reference 📖).

TRC file entries with initial data

TRC/SDF files generated for Simulink models including blocks from the RTI CAN MultiMessage Blockset contain initial data. The RTI CAN MultiMessage Blockset supplies all variables with initial values when they are included in the TRC file. TRC files with initial data lets you to perform offline calibration with ControlDesk.

Visualization with the Bus Navigator

The RTI CAN MultiMessage Blockset supports visualization with the Bus Navigator. Layouts/instruments are generated on demand and provide access to all CAN signals and all switches required to configure CAN communication during run time. You do not have to preconfigure layouts by hand.

RTI CAN Blockset and RTI CAN MultiMessage Blockset

(Not relevant for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board) You can use the RTI CAN Blockset and the RTI CAN MultiMessage Blockset in parallel for different CAN controllers. However, you cannot use the RTI CAN MultiMessage Blockset and the RTI CAN Blockset for the same CAN controller.

Further information on the RTI CAN MultiMessage Blockset

The following documents provide further information on the RTI CAN MultiMessage Blockset:

- RTI CAN MultiMessage Blockset Reference 📖

  This RTI reference provides a full description of the RTI CAN MultiMessage Blockset.
- RTI CAN MultiMessage Blockset Tutorial 📖

  This tutorial guides you through your first steps with the RTI CAN MultiMessage Blockset.

Basics on Working with CAN FD

Introduction

Using the CAN FD protocol allows data rates higher than 1 Mbit/s and payloads longer than 8 bytes per message.

Basics on CAN FD

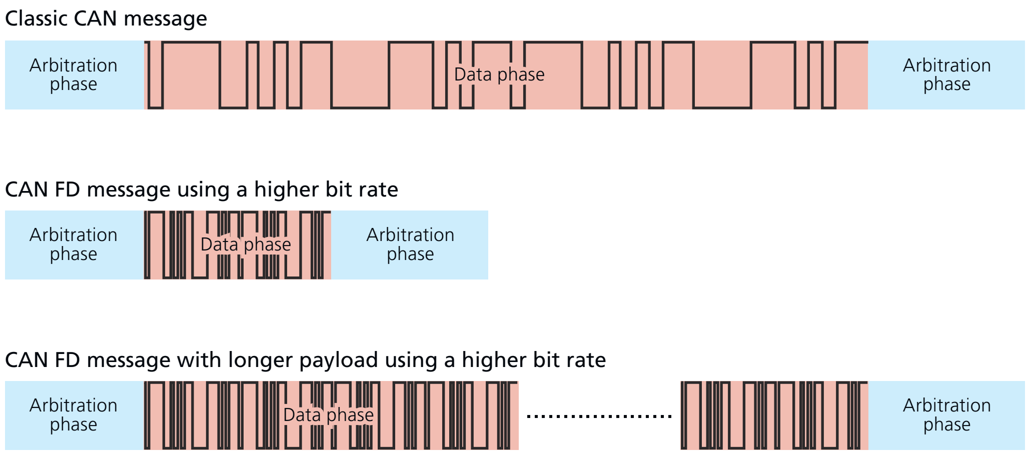
CAN FD stands for *CAN with Flexible Data Rate*. The CAN FD protocol is based on the CAN protocol as specified in ISO 11898-1. Compared with the classic CAN protocol, CAN FD comes with an increased bandwidth for the serial communication. The improvement is based on two factors:

- The CAN FD protocol allows you to use CAN messages with longer data fields (up to 64 bytes).
- The CAN FD protocol allows you to use a higher bit rate (typically higher by a factor of 8). It is possible to switch inside the message to the faster bit rate.

Arbitration phase and data phase CAN FD messages consist of two phases: a data phase and an arbitration phase. The data phase spans the phase where the data bits, CRC and length information are transferred. The rest of the frame, outside the data phase, is the arbitration phase. The data phase can be configured to have a higher bit rate than the arbitration phase.

CAN FD still uses the CAN bus arbitration method. During the arbitration process, the standard data rate is used. After CAN bus arbitration is decided, the data rate can be increased. The data bits are transferred with the preconfigured higher bit rate. At the end of the data phase, CAN FD returns to the standard data rate.

The following illustration shows a classic CAN message, a CAN FD message using a higher bit rate during the data phase, and a CAN FD message with longer payload using a higher bit rate. You can see the implications of the CAN FD features: The arbitration phases are identical in all cases, because the standard bit rate is always used. The lengths of the data phases differ depending on the payload length and bit rate used.



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.

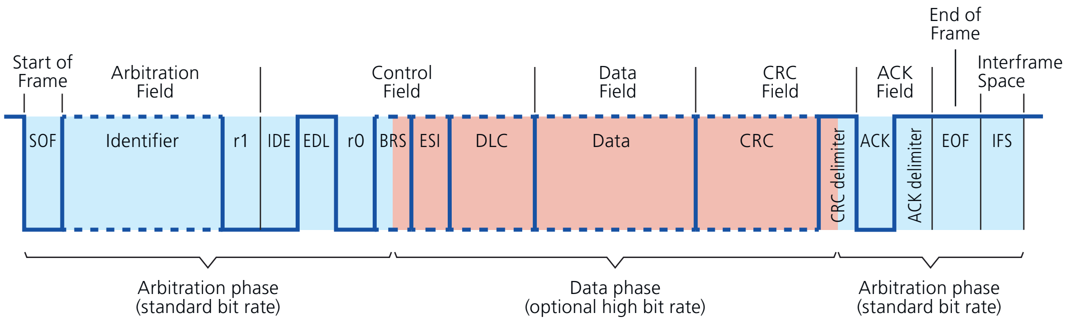
The RTI CAN MultiMessage Blockset supports both CAN FD protocols.

CAN FD message characteristics

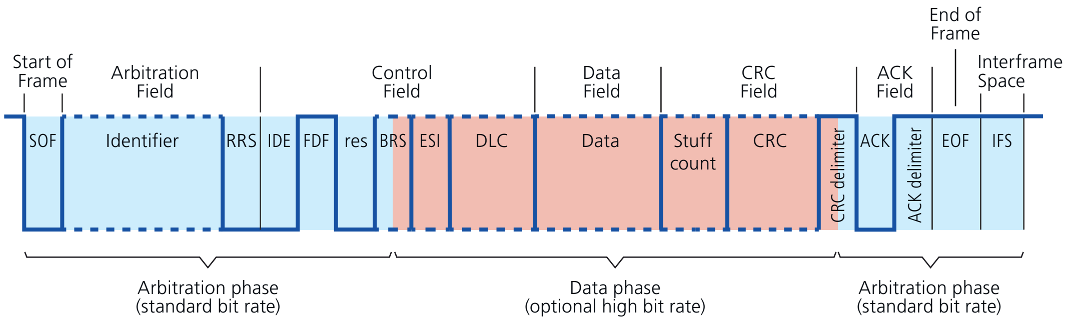
In principle, CAN FD messages and CAN messages consist of the same elements and have the same message structure.

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 📖)
- MicroAutoBox II: DS4342 Connections in Different Topologies (MicroAutoBox II Hardware Installation and Configuration Guide 📖)

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 📖)

Basics on Working with a J1939-Compliant DBC File

Introduction

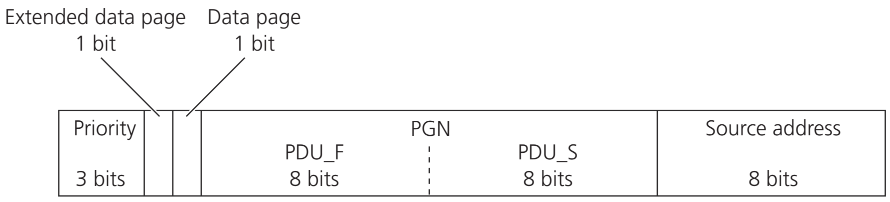
J1939 is a vehicle CAN bus standard defined by the Society of Automotive Engineers (SAE). It is used for communication in heavy-duty vehicles, for example, for communication between a tractor and its trailer.

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):



29-bit CAN message ID

- The 3-bit *priority* is used during the arbitration process. A value of 0 represents the highest priority, a value of 7 represents the lowest priority.
- The 1-bit *extended data page* can be used as an extension of the PGN. The RTI CAN MultiMessage Blockset lets you specify whether to use the extended data page bit this way. Refer to Code Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖).
- The 1-bit *data page* is a selector for the PDU_F in the PGN. It can be taken as an extension of the PGN. The RTI CAN MultiMessage Blockset uses the data page bit in this way.
- The 16-bit *PGN* is the parameter group number. It is described in this section.
- The 8-bit *source address* is the address of the transmitting network node.

Parameter group number (PGN) A 16-bit number in the 29-bit message identifier of a CAN message defined in a J1939-compliant DBC file. Each PGN references a *parameter group* that groups parameters and assigns them to the 8-byte data field of the message. A parameter group can be the engine temperature including the engine coolant temperature, the fuel temperature, etc. PGNs and parameter groups are defined by the SAE (see SAE J1939/71 Vehicle Application Layer).

The first 8 bits of the PGN represent the *PDU_F* (Protocol Data Unit format). The PDU_F value specifies the communication mode of the message (peer-to-peer or broadcast). The interpretation of the *PDU_S* value (PDU-specific) depends on the PDU_F value. For messages with a PDU_F < 240 (peer-to-peer communication, also called PDU1 messages), PDU_S is not relevant for the PGN, but contains the destination address of the network node that receives the message. For messages with a PDU_F ≥ 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.

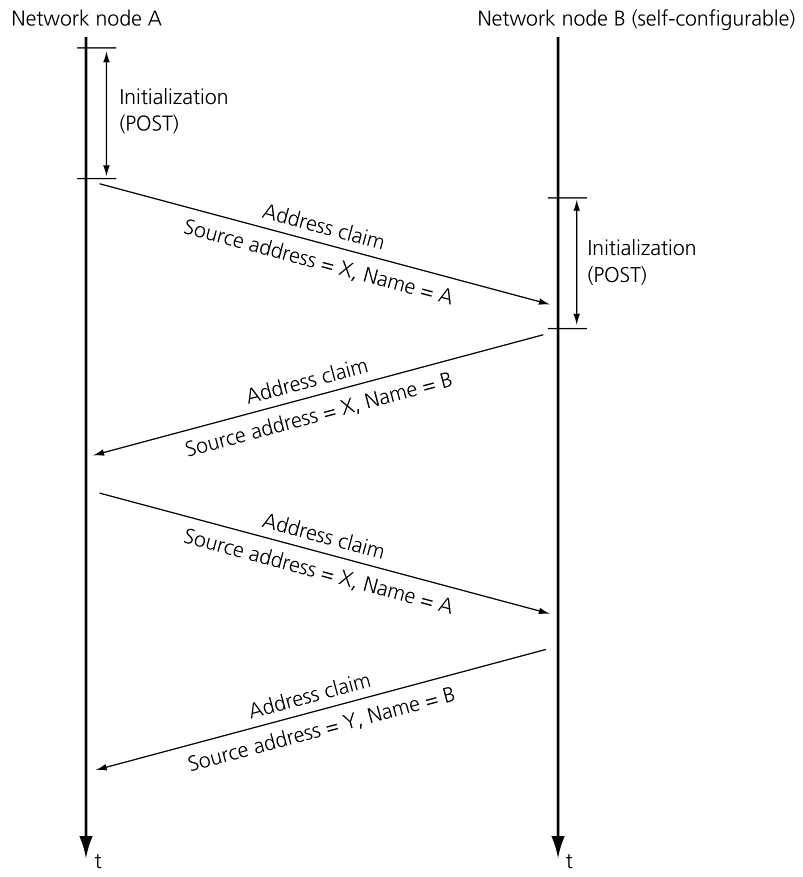
Name The J1939 standard defines a 64-bit *name* to identify each network node. The name indicates the main function of the network node with the associated address and provides information on the manufacturer.

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.


Network node A
Network node B (self-configurable)
Initialization (POST)
Address claim
Source address = X, Name = A
Initialization (POST)
Address claim
Source address = X, Name = B
Address claim
Source address = X, Name = A
Address claim
Source address = Y, Name = B
t
t

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.

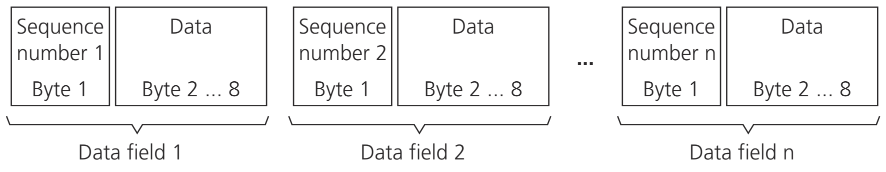
Note

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 📖)

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 📖). You can also define CAN communication without using a database file.

DBC file as the database The Data Base Container (DBC) file format was developed by Vector Informatik GmbH, Stuttgart, Germany. For the RTI CAN MultiMessage Blockset, you can use all the DBC files that pass the consistency check of Vector Informatik's CANdb++ Editor.

FIBEX file as the database The Field Bus Exchange (FIBEX) format is an XML exchange file format. It is used for data exchange between different tools that work with message-oriented bus communication. A FIBEX file usually describes more than one bus system. You therefore have to select one of the available bus systems if you work with a FIBEX file as the database.

AUTOSAR system description file as the database You can use an AUTOSAR system description file as the database for CAN communication. AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership that aims to develop and establish an open standard for automotive electric/electronic (E/E) architectures.

AUTOSAR system description files are files of AUTOSAR XML file type that describe a system according to AUTOSAR. A system is a combination of a hardware topology, a software architecture, a network communication, and information on the mappings between these elements. AUTOSAR system description files are instances of the AUTOSAR System Template.

An AUTOSAR system description file usually describes more than one bus system. You therefore have to select one of the available bus systems if you work with an AUTOSAR XML file as the database.

MAT file as the database You can also use the MAT file format as the database for CAN communication, or specify other database file formats as the database. You must convert your specific database files into the MAT file format for this purpose. Because the MAT file requires a particular structure, it must be generated by M-script.

Working without a database file If you want to work without a database file, you can use free raw messages and/or capture messages. These messages are independent of DBC and MAT files.

Changing database defaults When you work with a database file, you can change its default settings via the following dialog pages of the RTICANMM MainBlock:

- Cycle Time Defaults Page (RTICANMM MainBlock)
- Base/Update Time Page (RTICANMM MainBlock)
- TX Message Length Page (RTICANMM MainBlock)
- Message Defaults Page (RTICANMM MainBlock)
- Signal Defaults Page (RTICANMM MainBlock)
- Signal Ranges Page (RTICANMM MainBlock)
- Signal Errors Page (RTICANMM MainBlock)

Defining RX messages and TX messages

You can receive and/or transmit each message defined in the database file that you specify for CAN communication.

Defining RX messages You can define RX messages on the RX Messages Page (RTICANMM MainBlock).

Defining TX messages You can define TX messages on the TX Messages Page (RTICANMM MainBlock).

Triggering TX message transmission

You can specify different options to trigger the transmission of TX messages. For example, message transmission can be triggered cyclically or by an event.

For details, refer to Triggering Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Triggering reactions to the reception of RX messages

You can specify the reactions to receiving a specific RX message. One example of a reaction is the immediate transmission of a TX message.

For details, refer to Raw Data Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Working with raw data

The RTI CAN MultiMessage Blockset lets you to work with the raw data of messages. You have to select the messages for this purpose. The RTICANMM

MainBlock then provides the raw data of these messages to the model byte-wise for further manipulation. You can easily access the raw data of RX messages via a Simulink Bus Selector block.

For details, refer to Raw Data Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Implementing checksum algorithms

You can implement checksum algorithms for the checksum calculation of TX messages and checksum verification of RX messages.

Checksum header file You have to specify the checksum algorithms in a checksum header file. This needs to have a C-coded switch-case directive to switch between algorithms.

Checksum calculation for TX messages You can assign a checksum algorithm to each TX message. A checksum is calculated according to the algorithm and assigned to the TX message. Then the message is transmitted together with the calculated checksum.

Checksum check for RX messages You can assign a checksum algorithm to each RX message. A checksum is calculated for the message and compared to the checksum in the received message. If they differ, this is indicated at the error ports for RX messages if these ports are enabled.

Checksum algorithms based on end-to-end communication protection The RTI CAN MultiMessage Blockset supports run-time access to E2E protection parameters from AUTOSAR communication matrices and DBC files. This means that you can implement checksum algorithms based on end-to-end communication (E2E protection) parameters. E2E protection checksum algorithms are implemented in the same checksum header file as the checksum algorithms without E2E protection data.

For details, refer to Checksum Definition Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Gatewaying messages

Gatewaying means exchanging CAN messages between two CAN buses. Gatewaying also applies to messages that are not specified in the database file. You can also exclude individual messages specified in the database file from being exchanged.

You can gateway CAN messages in two ways:

Controller gateway This is a gateway between two CAN controllers. The gateway is between two RTICANMM ControllerSetup blocks and is independent of the active CAN controller variant.

MainBlocks gateway This is a gateway between different variants of two CAN controllers. The gateway is between two RTICANMM MainBlocks. The MainBlocks gateway is active only if the variants of both CAN controllers are active at the same time.

For details, refer to RTICANMM Gateway (RTI CAN MultiMessage Blockset Reference 🕮).

Related topics

References

General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Manipulating Signals to be Transmitted

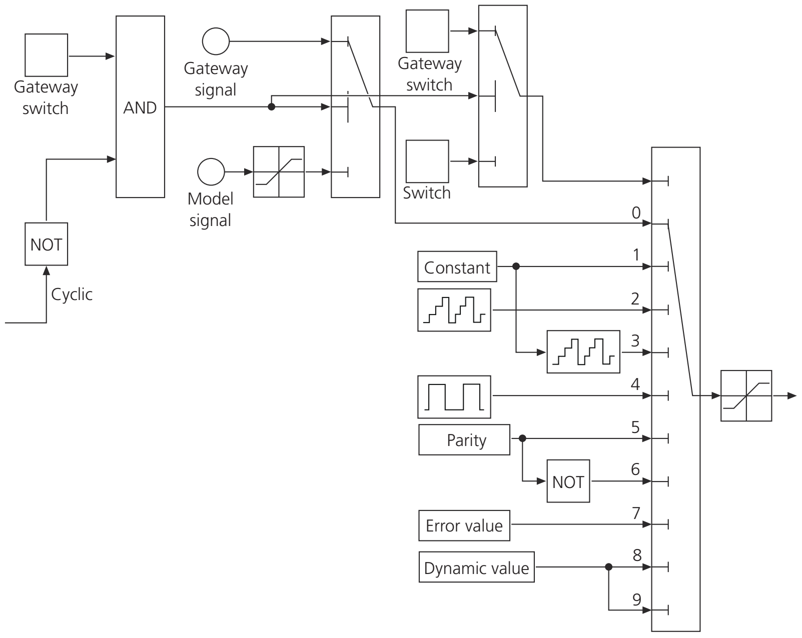
Introduction

All the signals of all the RX and TX messages (see Defining RX messages and TX messages on page 106) automatically get corresponding entries in the generated TRC file. This allows you to analyze them (signals of RX messages) or change their values (signals of TX messages) with the Bus Navigator in ControlDesk.

Manipulating signals to be transmitted

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

The illustration below visualizes the options.



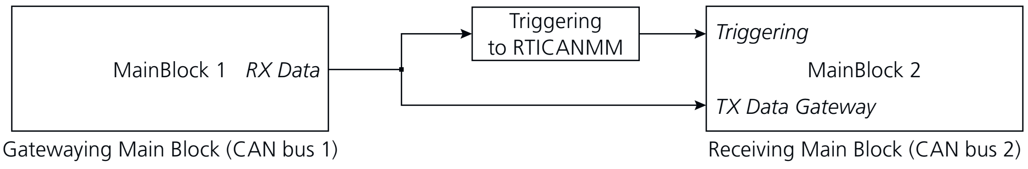
You can switch between these options in ControlDesk.

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

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

You can specify TX model signals on the Model Signals (TX) Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

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



MainBlock1 gateways messages and their signals to MainBlock2. Specifying gateway signals for MainBlock2 adds a **TX Data Gateway** inport 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 inports and therefore reduces performance. If you want to exchange messages between CAN controllers and do not want to perform signal manipulation, you should use the RTICANMM Gateway block instead.

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

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

You can specify toggle signals on the Toggle Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

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

You can specify parity signals on the Parity Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖).

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

- Behavior of counter signals

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

  You can specify counter signals to check for correct transmission of a message or trigger the transmission of signals in a message.
  - *Checking correct message transmission*: The receiver of a message expects a certain counter signal value. For example, if the transmission of a message was stopped for two transmissions, the expected counter signal value and the real counter signal value can differ, which indicates an error. Counter signals used in this way are often called alive counters.
  - *Triggering the transmission of signals*: You can trigger the transmission of signals if you specify a mode signal as a counter signal. The signal value of the mode signal triggers the transmission of mode-dependent signals. Because the counter signal value changes with every message transmission, this triggers the transmission of the mode-dependent signals. By using counter signals in this way, you can work with signals which use the same bytes of a message. Counter signals used in this way are often called mode counters.
- Using counter signals in ControlDesk

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

You can specify counter signals on the Counter Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖).

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

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

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

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

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

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

CAN Signal Mapping

Introduction

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

CAN Signal Mapping

Introduction

The CAN subsystem of the DS2210 provides two CAN bus interfaces that meet the ISO/DIS 11898 specifications.

I/O signals

The following table lists the CAN signals of the DS2210 and the mapping of the signals to RTI blocks and RTLib functions.

The table also provides the mapping of the I/O signals to the I/O pins on the DS2210 and on the Sub-D connectors (P1A, P1B, P2A, P2B).

Signal	Channel/Bit Numbers of Related RTI Blocks/RTLib Functions				I/O Pin on ...	
	Related RTI Block(s)	Ch/Bit (RTI)	Related RTLib Functions	Ch/Bit (RTLib)	DS2210	Sub-D Conn.
CAN Support						
▪ CANxL: CAN dominant low ▪ CANxH: CAN dominant high ▪ Electrical characteristics: see CAN Bus Interface (PHS Bus System Hardware Reference 📖)						
CAN1L	▪ 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

Introduction

The DS2210 provides interrupts, which you can use in your Simulink model and your handcoded model.

Where to go from here

Information in this section

DS2210 Interrupts......113
The DS2210 provides different interrupt types.

Interrupt Handling......114
The interrupt handling is different if you use RTI or RTLib functions for your application.

Information in other sections

Introduction to the Features of the DS2210......9

DS2210 Interrupts

Introduction

The DS2210 provides different interrupt types.

PHS bus interrupt

The PHS bus provides several interrupt lines for communication between I/O boards and processor board (one for each I/O board). The PHS bus interrupt controller of the DS2210 I/O board provides 8 interrupts that can be requested by the serial interface, the angular processing unit (angle position interrupts), and the slave CAN. These interrupts can be masked. A global interrupt enable/disable is available. Only enabled interrupt sources generate interrupts.

Serial interface interrupt

The serial interface allocates one hardware interrupt. A subinterrupt handler allows you to specify different subinterrupts that support sending (Tx register empty) and receiving, for example.

Angle position interrupts

Depending on the engine position (angle), the angular processing unit can generate angle position interrupts for up to 6 cylinders. You can generate interrupts when the cylinder has passed the top dead center (TDC), for example. Any interrupt position can be chosen while several interrupt positions are possible for each cylinder. Use RTI (DS2210APU_INT_Bx_Iy) or RTLib (`ds2210_int_position_set`) functions to define the interrupt positions.

Slave CAN MC interrupt

The slave CAN MC can request an interrupt by writing to a predefined location in its dual-ported memory (DPMEM). You can define subinterrupts for different message events or CAN bus events.

Slave DSP interrupts

The slave DSP receives two different interrupts. One is issued by the angle processing unit when the engine position has been updated (every 1 µs). You can use this interrupt to synchronize the slave DSP and angular processing unit. Second, 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.

Interrupt sources

The following interrupt sources are available:

Interrupt Line	Interrupt Source
IRQ0 … IRQ5	Angle position interrupts for up to 6 cylinders
IRQ6	Interrupt from the CAN controller
IRQ7	Interrupt from the serial interface (UART)

Related topics

References

ds2210_int_position_set (DS2210 RTLib Reference 🕮)
DS2210APU_INT_Bx_Iy (DS2210 RTI Reference 🕮)

Interrupt Handling

Introduction

The interrupt handling is different if 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 of your Simulink model. When the task in this system has finished, the interrupt handling automatically creates an "End of interrupt" (EOI) message to indicate the state for 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.

Related topics

References

DS2210 Interrupts..113
ds2210_int_position_set (DS2210 RTLib Reference 🕮)
DS2210APU_INT_Bx_Iy (DS2210 RTI Reference 🕮)

Limitations

Introduction

There are some limitations you have to take into account when working with the DS2210.

Where to go from here

Information in this section

Quantization Effects...118
Quantization effects occur in signal generation or measurement.

DS2210 Board Revision...119
Some features are only available for DS2210 boards with higher board revision.

Conflicting I/O Features..119
Some I/O features of the DS2210 conflict with other I/O features of the board.

Limited Number of CAN Messages..129
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..130
There are a number of general limitations with RTICANMM.

Limitations with J1939-Support...134
There are a number of limitations regarding the J1939 support of RTICANMM.

Information in other sections

Introduction to the Features of the DS2210...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:

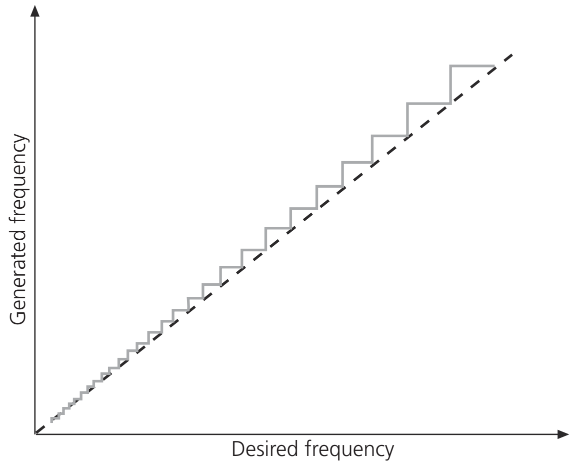
$$f = \frac{1}{n \cdot R}$$

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.

DS2210 Board Revision

Introduction

Several features are supported only for DS2210 boards with specific revisions and higher, for example, if you want to use the ignition capture unit for injection capture. The following boards are extended in functionality:

- DS2210 boards with board revision 4 or higher.
- DS2210 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>.0.<FPGA>
```

For example, 3.0.3 means you have a DS2210 board with the revision 3 and a FPGA revision number of 3 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 📖).

Related topics

References

Board Details Properties (ControlDesk Platform Management 📖)

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 DS2210

The following tables list the I/O features of the DS2210 that conflict with other I/O features, and the related RTI blocks/RTLib functions.

- Conflicts for the sensor and actuator interface
  - Conflicts for PWM Signal Generation on page 120
  - Conflicts for Square-Wave Signal Generation (D2F) on page 121
  - Conflicts for PWM Signal Measurement (PWM2D) on page 121
  - Conflicts for Square-Wave Signal Measurement (F2D) on page 122
  - Conflicts for Wheel Speed Sensor Simulation on page 122
- Conflicts for the angular processing unit (APU)
  - Conflicts for Spark Event Capture (Last Event Window Read) on page 123
  - Conflicts for Spark Event Capture (Last Event Window Read) on page 123
  - Conflicts for Injection Pulse Position and Fuel Amount Measurement (Last Event Window Read) on page 126
  - Conflicts for Injection Pulse Position and Fuel Amount Measurement (Continuous Read) on page 127
  - Conflicts for Knock Sensor Simulation on page 123
- Conflicts for the serial interface
  - Conflicts for the Serial Interface on page 128

Conflicts for PWM Signal Generation

The following I/O features of the DS2210 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 1	PWM_OUT1	Square-wave signal generation (D2F)	Ch 1	Ch 1
Ch 2	Ch 2	PWM_OUT2	Square-wave signal generation (D2F)	Ch 2	Ch 2
Ch 3	Ch 3	PWM_OUT3	Square-wave signal generation (D2F)	Ch 3	Ch 3
Ch 4	Ch 4	PWM_OUT4	Square-wave signal generation (D2F)	Ch 4	Ch 4
Ch 5	Ch 5	PWM_OUT5	Square-wave signal generation (D2F)	Ch 5	Ch 5
Ch 6	Ch 6	PWM_OUT6	Square-wave signal generation (D2F)	Ch 6	Ch 6
*) Related RTI blocks and RTLib functions: ▪ DS2210PWM_Bx_Cy ▪ PWM Signal Generation (DS2210 RTLib Reference 🕮)			**) Related RTI blocks and RTLib functions: ▪ DS2210D2F_Bx_Cy ▪ Square-Wave Signal Generation (DS2210 RTLib Reference 🕮)		

Conflicts for Square-Wave Signal Generation (D2F)

The following I/O features of the DS2210 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 1	PWM_OUT1	PWM signal generation	Ch 1	Ch 1
Ch 2	Ch 2	PWM_OUT2	PWM signal generation	Ch 2	Ch 2
Ch 3	Ch 3	PWM_OUT3	PWM signal generation	Ch 3	Ch 3
Ch 4	Ch 4	PWM_OUT4	PWM signal generation	Ch 4	Ch 4
Ch 5	Ch 5	PWM_OUT5	PWM signal generation	Ch 5	Ch 5
Ch 6	Ch 6	PWM_OUT6	PWM signal generation	Ch 6	Ch 6
*) Related RTI blocks and RTLib functions: ▪ DS2210D2F_Bx_Cy ▪ *Square-Wave Signal Generation (DS2210 RTLib Reference 🕮)*			**) Related RTI blocks and RTLib functions: ▪ DS2210PWM_Bx_Cy ▪ *PWM Signal Generation (DS2210 RTLib Reference 🕮)*		

Conflicts for PWM Signal Measurement (PWM2D)

The following I/O features of the DS2210 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 1	PWM_IN1	Square-wave signal measurement (F2D)	Ch 1	Ch 1
Ch 2	Ch 2	PWM_IN2	Square-wave signal measurement (F2D)	Ch 2	Ch 2
Ch 3	Ch 3	PWM_IN3	Square-wave signal measurement (F2D)	Ch 3	Ch 3
Ch 4	Ch 4	PWM_IN4	Square-wave signal measurement (F2D)	Ch 4	Ch 4
Ch 5	Ch 5	PWM_IN5	Square-wave signal measurement (F2D)	Ch 5	Ch 5
Ch 6	Ch 6	PWM_IN6	Square-wave signal measurement (F2D)	Ch 6	Ch 6
Ch 7	Ch 7	PWM_IN7	Square-wave signal measurement (F2D)	Ch 7	Ch 7
		(INJ7)	Injection pulse position and fuel amount measurement	Group 1, ch 7	Group 1, ch 7
Ch 8	Ch 8	PWM_IN8	Square-wave signal measurement (F2D)	Ch 8	Ch 8
		(INJ8)	Injection pulse position and fuel amount measurement	Group 1, ch 8	Group 1, ch 8
*) Related RTI blocks and RTLib functions: ▪ DS2210PWM2D_Bx_Cy ▪ *PWM Signal Measurement (DS2210 RTLib Reference 🕮)*			**) Related RTI blocks and RTLib functions: ▪ Square-wave signal measurement (F2D): ▪ DS2210F2D_Bx_Cy ▪ *Frequency Measurement (DS2210 RTLib Reference 🕮)* ▪ Injection pulse position and fuel amount measurement: ▪ DS2210APU_INJ_Bx_Gy DS2210APU_INJCONT_Bx_Gy		

PWM Signal Measurement *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
			▪ *Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference 📖)*		

Conflicts for Square-Wave Signal Measurement (F2D)

The following I/O features of the DS2210 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 1	PWM_IN1	PWM signal measurement (PWM2D)	Ch 1	Ch 1
Ch 2	Ch 2	PWM_IN2	PWM signal measurement (PWM2D)	Ch 2	Ch 2
Ch 3	Ch 3	PWM_IN3	PWM signal measurement (PWM2D)	Ch 3	Ch 3
Ch 4	Ch 4	PWM_IN4	PWM signal measurement (PWM2D)	Ch 4	Ch 4
Ch 5	Ch 5	PWM_IN5	PWM signal measurement (PWM2D)	Ch 5	Ch 5
Ch 6	Ch 6	PWM_IN6	PWM signal measurement (PWM2D)	Ch 6	Ch 6
Ch 7	Ch 7	PWM_IN7	PWM signal measurement (PWM2D)	Ch 7	Ch 7
		(INJ7)	Injection pulse position and fuel amount measurement	Group 1, ch 7	Group 1, ch 7
Ch 8	Ch 8	PWM_IN8	PWM signal measurement (PWM2D)	Ch 8	Ch 8
		(INJ8)	Injection pulse position and fuel amount measurement	Group 1, ch 8	Group 1, ch 8
*) Related RTI blocks and RTLib functions: ▪ DS2210F2D_Bx_Cy ▪ *Frequency Measurement (DS2210 RTLib Reference 📖)*			**) Related RTI blocks and RTLib functions: ▪ PWM signal measurement (PWM2D) ▪ DS2210PWM2D_Bx_Cy ▪ *PWM Signal Measurement (DS2210 RTLib Reference 📖)* ▪ Injection Pulse Position and Fuel Amount Measurement: ▪ DS2210APU_INJ_Bx_Gy DS2210APU_INJCONT_Bx_Gy ▪ *Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference 📖)*		

Conflicts for Wheel Speed Sensor Simulation

The following I/O features of the DS2210 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					
If you perform wheel speed sensor simulation, you cannot perform knock sensor simulation at the same time.					
*) Related RTI blocks and RTLib functions: ▪ DS2210SL_WSSG_Bx_Cy			**) Related RTI blocks and RTLib functions: ▪ DS2210SL_KNSG_Bx_Cy ▪ Knock Sensor Simulation (DS2210 RTLib Reference 📖)		

Wheel Speed Sensor Simulation *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
▪ Wheel Speed Sensor Simulation (DS2210 RTLib Reference 🕮)					

Conflicts for Knock Sensor Simulation

The following I/O features of the DS2210 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					
If you perform knock sensor simulation, you cannot perform wheel speed sensor simulation at the same time.					
*) Related RTI blocks and RTLib functions: ▪ DS2210SL_KNSG_Bx_Cy ▪ *Knock Sensor Simulation (DS2210 RTLib Reference 🕮)*			**) Related RTI blocks and RTLib functions: ▪ DS2210SL_WSSG_Bx_Cy ▪ *Wheel Speed Sensor Simulation (DS2210 RTLib Reference 🕮)*		

Conflicts for Spark Event Capture (Last Event Window Read)

The following I/O features of the DS2210 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 Blocks/Functions for All Channels					
All channels	All channels	IGN1 … 6, AUXCAP1, AUXCAP2	▪ Spark event capture (continuous read)	▪ All channels	
			▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	▪ Group 2 (all channels)	▪ Ch 1 … 8 (all channels)
			▪ Spark event capture, auxiliary input (continuous read)	▪ Aux. ch 1/ Aux. ch 2	
Ch 7	Ch 7	AUXCAP1	Spark event capture, auxiliary input (continuous read)	Aux. ch 1	
Ch 8	Ch 8	AUXCAP2	Spark event capture, auxiliary input (continuous read)	Aux. ch 2	
*) Related RTI blocks and RTLib functions: ▪ DS2210APU_IGN_Bx ▪ *Spark Event Capture (DS2210 RTLib Reference 🕮)*			**) Related RTI blocks and RTLib functions: ▪ Spark event capture (continuous read): ▪ DS2210APU_IGNCONT_Bx ▪ Injection pulse position and fuel amount measurement (last event window and continuous read): ▪ DS2210APU_INJ_Bx_Gy DS2210APU_INJCONT_Bx_Gy(group 2)		

Spark Event Capture (Last Event Window Read) *)		**Signal**	**Conflicting I/O Feature **)**		
Ch (RTI)	**Ch (RTLib)**			**Ch (RTI)**	**Ch (RTLib)**
			▪ *Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference)* ▪ Spark event capture, auxiliary input (continuous read): ▪ DS2210APU_AUXCAPCONT_Bx_Cy		
Conflicts Concerning Blocks/Functions for Auxiliary Inputs Only					
Aux. ch 1	Aux. ch 1	AUXCAP1	▪ Spark event capture (continuous read)	▪ All channels	
			▪ Spark event capture, auxiliary input (continuous read)	▪ Aux. ch 1/ Aux. ch 2	
			▪ Injection pulse position and fuel amount measurement (continuous read)	▪ Group 2 (all channels)	▪ Ch 1 … 8 (all channels)
			▪ Spark event capture (last event window read)	▪ Ch 7	
			▪ Injection pulse position and fuel amount measurement (last event window read)	▪ Group 2, ch 7	▪ Ch 7
Aux. ch 2	Aux. ch 2	AUXCAP2	▪ Spark event capture (continuous read)	▪ All channels	
			▪ Spark event capture, auxiliary input (continuous read)	▪ Aux. ch 1/ Aux. ch 2	
			▪ Injection pulse position and fuel amount measurement (continuous read)	▪ Group 2 (all channels)	▪ Ch 1 … 8 (all channels)
			▪ Spark event capture (last event window read)	▪ Ch 8	
			▪ Injection pulse position and fuel amount measurement (last event window read)	▪ Group 2, ch 8	▪ Ch 8
*) Related RTI blocks and RTLib functions: ▪ DS2210APU_AUXCAP_Bx_Cy ▪ *Spark Event Capture (DS2210 RTLib Reference)*			**) Related RTI blocks and RTLib functions: ▪ Spark event capture (continuous read): ▪ DS2210APU_IGNCONT_Bx ▪ Spark event capture, auxiliary input (continuous read): ▪ DS2210APU_AUXCAPCONT_Bx_Cy ▪ Injection pulse position and fuel amount measurement (continuous read): ▪ DS2210APU_INJCONT_Bx_Gy(group 2) ▪ *Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference)* ▪ Spark event capture (last event window read): ▪ DS2210APU_IGN_Bx ▪ Injection pulse position and fuel amount measurement (last event window read): ▪ DS2210APU_INJ_Bx_Gy ▪ *Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference)*		

Conflicts for Spark Event Capture (Continuous Read)

The following I/O features of the DS2210 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 Blocks/Functions for All Channels					
All channels	All channels	IGN1 … 6, AUXCAP1, AUXCAP2	▪ 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)	▪ All channels ▪ Group 2 (all channels) ▪ Aux. ch 1/ Aux. ch 2	▪ Ch 1 … 8 (all channels)
Ch 7	Ch 7	AUXCAP1	Spark event capture, auxiliary input (last event window read)	Aux. ch 1	
Ch 8	Ch 8	AUXCAP2	Spark event capture, auxiliary input (last event window read)	Aux. ch 2	
*) Related RTI blocks and RTLib functions: ▪ DS2210APU_IGNCONT_Bx ▪ *Spark Event Capture (DS2210 RTLib Reference 🕮)*			**) Related RTI blocks and RTLib functions: ▪ Spark event capture (last event window read): ▪ DS2210APU_IGN_Bx ▪ Injection pulse position and fuel amount measurement (last event window and continuous read): ▪ DS2210APU_INJ_Bx_Gy DS2210APU_INJCONT_Bx_Gy(group 2) ▪ *Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference 🕮)* ▪ Spark event capture, auxiliary input (last event window read): ▪ DS2210APU_AUXCAP_Bx_Cy		
Conflicts Concerning Blocks/Functions for Auxiliary Inputs Only					
Aux. ch 1	Aux. ch 1	AUXCAP1	▪ 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)	▪ All channels ▪ Aux. ch 1/ Aux. ch 2 ▪ Ch 7 ▪ Ch 7	▪ Ch 7
Aux. ch 2	Aux. ch 2	AUXCAP2	▪ 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)	▪ All channels ▪ Aux. ch 1/ Aux. ch 2 ▪ Ch 8 ▪ Ch 8	▪ Ch 8

Spark Event Capture (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
*) Related RTI blocks and RTLib functions: ▪ DS2210APU_AUXCAPCONT_Bx_Cy ▪ *Spark Event Capture (DS2210 RTLib Reference 🕮)*			**) Related RTI blocks and RTLib functions: ▪ Spark event capture (last event window read): ▪ DS2210APU_IGN_Bx ▪ Spark event capture, auxiliary input (last event window read): ▪ DS2210APU_AUXCAP_Bx_Cy ▪ Spark event capture (continuous read): ▪ DS2210APU_IGNCONT_Bx ▪ Injection pulse position and fuel amount measurement (last event window and continuous read): ▪ DS2210APU_INJ_Bx_Gy DS2210APU_INJCONT_Bx_Gy(group 2) ▪ *Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference 🕮)*		

Conflicts for Injection Pulse Position and Fuel Amount Measurement (Last Event Window Read)

The following I/O features of the DS2210 conflict with injection pulse position and fuel amount measurement 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 (all channels)	Ch 1 … 8 (all channels)	INJ1 … 8	Injection pulse position and fuel amount measurement (continuous read)	Group 1 (all channels)	
Group 1, ch 7	Ch 7	INJ7	Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D)	Ch 7	Ch 7
Group 1, ch 8	Ch 8	INJ8	Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D)	Ch 8	Ch 8
*) Related RTI blocks and RTLib functions: ▪ DS2210APU_INJ_Bx_Gy(group 1) ▪ *Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference 🕮)*			**) Related RTI blocks and RTLib functions: ▪ Injection pulse position and fuel amount measurement (continuous read) ▪ DS2210APU_INJCONT_Bx_Gy(group 1) ▪ PWM2D/F2D: ▪ DS2210PWM2D_Bx_Cy DS2210F2D_Bx_Cy ▪ *PWM Signal Measurement (DS2210 RTLib Reference 🕮)* *Frequency Measurement (DS2210 RTLib Reference 🕮)*		
Conflicts Concerning Group 2					
Group 2 (all channels)	Ch 1 … 8 (all channels)	IGN1 … 6, AUXCAP1, AUXCAP2	▪ Spark event capture (last event window and continuous read) ▪ Injection pulse position and fuel amount measurement (continuous read)	▪ All channels ▪ Group 2 (all channels)	▪ 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	Spark event capture, auxiliary input (last event window and continuous read)	Aux. ch 1	Aux. ch 1
Group 2, ch 8	Ch 8	AUXCAP2	Spark event capture, auxiliary input (last event window and continuous read)	Aux. ch 2	Aux. ch 2
*) Related RTI blocks and RTLib functions: ▪ DS2210APU_INJ_Bx_Gy(group 2) ▪ *Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference 📖)*			**) Related RTI blocks and RTLib functions: ▪ Spark event capture (last event window and continuous read): ▪ DS2210APU_IGN_Bx DS2210APU_IGNCONT_Bx ▪ *Spark Event Capture (DS2210 RTLib Reference 📖)* ▪ Injection pulse position and fuel amount measurement (continuous read): ▪ DS2210APU_INJCONT_Bx_Gy(group 2) ▪ Spark event capture, auxiliary input (last event window and continuous read): ▪ DS2210APU_AUXCAP_Bx_Cy DS2210APU_AUXCAPCONT_Bx_Cy ▪ *Spark Event Capture (DS2210 RTLib Reference 📖)*		

Conflicts for Injection Pulse Position and Fuel Amount Measurement (Continuous Read)

The following I/O features of the DS2210 conflict with injection pulse position and fuel amount measurement 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 (all channels)	Ch 1 … 8 (all channels)	INJ1 … 8	Injection pulse position and fuel amount measurement (last event window read)	Group 1 (all channels)	
Group 1, ch 7	Ch 7	INJ7	Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D)	Ch 7	Ch 7
Group 1, ch 8	Ch 8	INJ8	Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D)	Ch 8	Ch 8
*) Related RTI blocks and RTLib functions: ▪ DS2210APU_INJCONT_Bx_Gy (group 1) ▪ Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference 📖)			**) Related RTI blocks and RTLib functions: ▪ Injection pulse position and fuel amount measurement (last event window read): ▪ DS2210APU_INJ_Bx_Gy (group 1) ▪ PWM2D/F2D: ▪ DS2210PWM2D_Bx_Cy DS2210F2D_Bx_Cy ▪ PWM Signal Measurement (DS2210 RTLib Reference 📖) Frequency Measurement (DS2210 RTLib Reference 📖)		

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 (all channels)	Ch 1 … 8 (all channels)	IGN1 … 6, AUXCAP1, AUXCAP2	▪ Spark event capture (last event window and continuous read) ▪ Injection pulse position and fuel amount measurement (last event window read)	▪ All channels ▪ Group 2 (all channels)	▪ All channels
Group 2, ch 7	Ch 7	AUXCAP1	Spark event capture, auxiliary input (last event window and continuous read)	Aux. ch 1	Aux. ch 1
Group 2, ch 8	Ch 8	AUXCAP2	Spark event capture, auxiliary input (last event window and continuous read)	Aux. ch 2	Aux. ch 2
*) Related RTI blocks and RTLib functions: ▪ DS2210APU_INJ_Bx_Gy(group 2) ▪ Injection Pulse Position and Fuel Amount Measurement (DS2210 RTLib Reference 📖)			**) Related RTI blocks and RTLib functions: ▪ Spark event capture (last event window and continuous read) ▪ DS2210APU_IGN_Bx DS2210APU_IGNCONT_Bx ▪ Spark Event Capture (DS2210 RTLib Reference 📖) ▪ Injection pulse position and fuel amount measurement (continuous read) ▪ DS2210APU_INJCONT_Bx_Gy(group 2) ▪ Spark event capture, auxiliary input (last event window and continuous read): ▪ DS2210APU_AUXCAP_Bx_Cy DS2210APU_AUXCAPCONT_Bx_Cy ▪ Spark Event Capture (DS2210 RTLib Reference 📖)		

Conflicts for the Serial Interface

The DS2210 supports only one serial interface. It can be configured as either RS232 or RS422 mode.

Related topics

Basics

Introduction to the Features of the DS2210..9

References

Limitations..117

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

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]

[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 84.

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 **rticanmmlib** 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
- characteristics 18
- I/O mapping 19

analog output circuits 69
angle position interrupt 46
angular processing unit
- overview 45

APU overview 45

B

bit I/O unit 24
- I/O mapping 24

C

camshaft phase 50
camshaft sensor signal generation 61
- I/O mapping 62

camshaft signal generator 50
CAN
- channel 76
- fault-tolerant transceiver 80
- interrupts 84
- physical layer 78
- service 87
- setup 76
- status information 87

CAN FD 94
CAN subsystem 10
CAN support 75
cascading DS2210 boards 46
cascading I/O boards 47
Common Program Data folder 8
comparing transceiver modes 39
conflicts of I/O features 119
continuous capture 55
crankshaft sensor signal generation 60
- I/O mapping 60

crankshaft signal generator 49

D

D/A channel 21
D/R converter 22
- I/O mapping 23

DAC unit 20
- I/O mapping 21

data file support 83
defining CAN messages 82
Documents folder 8
DS2210
- analog output circuits 69
- cascading 46
- interrupts 113
- master board 46, 48
- slave board 46, 48

DS2210 board revision 119
DS802
- partitioning PHS bus 14

DSP subsystem 10
duty cycle 26

E

engine position bus 45
engine position interrupt 46
engine position phase accumulator 46
event capture window 51
event capture windows 55

F

feature summary 12
frequency measurement 32
fuel amount measurement 52
- I/O mapping 66

I

I/O boards
- cascading 47

I/O features
- conflicts 119

I/O mapping
- ADC unit 19
- bit I/O unit 24
- camshaft sensor signal generation 62
- crankshaft sensor signal generation 60
- D/R converter 23
- DAC unit 21
- frequency measurement 34
- fuel amount measurement 66
- knock sensor simulation 72
- PWM signal generation 32
- PWM signal measurement 28
- spark event measurement 64
- square-wave signal generation 35
- wheel speed sensor simulation 74

injection event capture unit 52
injection pulse position measurement 65
- continuous capture 58
- I/O mapping 66

interrupt control 13
interrupts 113
ISO11898 79
ISO11898-6 80

J

J1939
- broadcast/peer-to-peer messages 99
- limitations 134
- working with J1939-compliant DBC files 99

K

knock sensor simulation 70
- I/O mapping 72

L

limitations
- J1939 134
- RTI CAN MultiMessage Blockset 130

Local Program Data folder 8

M

master DS2210 46, 48
messages
- defining CAN messages 82
- delay time 82
- multiple 83

multiple data files 83
multiple event capture mode 52
multiple message support 83

P

partitioning PHS bus with DS802 14
phase accumulator 46
PHS++ bus 14
piggyback support 81
PWM analysis 26
PWM period 26
PWM signal generation 29
- I/O mapping 32

PWM signal measurement 25
- I/O mapping 28

Q

quantization effects 118

R

receive buffer 41
resistor channel 23
- I/O mapping 23

resistor output circuits 22
RS485 80
RTI CAN MultiMessage Blockset
- limitations 130
- supported platforms 89

RTICANMM
- J1939 134

RX service support 84
RX SW FIFO 41

S

sample hit 58
serial interface 37
- baud rates 40
- cable length/baud rate (RS232) 39
- cable length/baud rate (RS422) 39
- data transfer 38
- oscillator frequency 40
- RS232 transceiver mode 39
- RS422 networks 39
- RS422 topologies 40
- RS422 transceiver mode 39

single event capture mode 51
slave DS2210 46, 48
slave DSP
- analog output circuits 69

slave DSP TMS320C31 69
spark event
  continuous capture 58
spark event capture unit 51
spark event measurement 64
  I/O mapping 64
square-wave signal generation 34
  I/O mapping 35
start position/fuel amount capture mode 53

T

time-base bus 45
TJA1041
  limitations 81
TJA1054
  transceiver 81
transceiver
  TJA1054 81
transmit buffer 41
TX SW FIFO 41

U

UART 37

W

wave table
  basics 62
  generating 63
wheel speed sensor simulation 73
  I/O mapping 74
working with CAN FD 94