

dSPACE Simulator Mid-Size Based on DS2211

Features

Release 2021-A – May 2021

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

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

© 2004 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

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

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

Contents

About This Document	9
Overview of the Simulator	11
Overview of the Simulator.....	12
Block Diagram of the Simulator.....	13
Functional Units.....	13
Processor Board	15
DS1006 Architecture.....	15
DS1006 Feature Overview.....	16
DS2211 HIL I/O Board	19
System Overview of the DS2211.....	19
Features of the DS2211.....	22
Sensor and Actuator Interface	25
Standard I/O.....	26
ADC Unit.....	26
DAC Unit.....	29
D/R Converter.....	31
Bit I/O Unit.....	33
Timing I/O.....	36
PWM Signal Measurement.....	36
PWM Signal Generation.....	40
Frequency Measurement (F2D).....	43
Square-Wave Signal Generation (D2F).....	46
Serial Interface.....	49
Basics on the Serial Interface.....	49
Comparing RS232 and RS422.....	50
Specifying the Baud Rate of the Serial Interface.....	52
Software FIFO Buffer.....	52

Angular Processing Unit 55

APU Basics.....	56
APU Overview.....	57
Engine Position Phase Accumulator.....	59
Cascading I/O Boards.....	60
Crankshaft Signal Generator.....	61
Reverse Crankshaft Rotation.....	62
Camshaft Signal Generator.....	65
Spark Event Capture Unit.....	67
Injection Event Capture Unit.....	68
Event Capture Windows.....	72
Continuous Value Capturing.....	76
Complex Comparators.....	77
Angular Processing Unit - Variant.....	81
Basics on Simulating Engine Variants.....	81
Building a Simulink Model for Engine Variants.....	83
Multiple Event Capture Mode of the VAR APU Blockset.....	84
Example of Capturing a Multiple Event with the VAR APU Blockset.....	85
APU Reference.....	88
Crankshaft Sensor Signal Generation.....	88
Camshaft Sensor Signal Generation.....	90
Wave Table Generation.....	92
Spark Event Capture.....	93
Injection Pulse Position and Fuel Amount Measurement.....	95

Features of the Slave DSP 99

Basics of the Slave DSP.....	100
Knock Sensor Simulation.....	101
Wheel Speed Sensor Simulation.....	103
DSP DAC Unit.....	105
DSP Bit I/O Unit and Capture Input Access.....	106
Slave DSP Interrupts.....	107

CAN Support 109

Setting Up a CAN Controller.....	110
Initializing the CAN Controller.....	110
CAN Transceiver Types.....	112
DS2211: Selecting the CAN Controller Frequency.....	116

Defining CAN Messages.....	117
Implementing a CAN Interrupt.....	118
Using RX Service Support.....	119
Removing a CAN Controller (Go Bus Off).....	121
Getting CAN Status Information.....	121
Using the RTI CAN MultiMessage Blockset.....	124
Basics on the RTI CAN MultiMessage Blockset.....	124
Basics on Working with CAN FD.....	129
Basics on Working with a J1939-Compliant DBC File.....	134
Transmitting and Receiving CAN Messages.....	140
Manipulating Signals to be Transmitted.....	143
CAN Signal Mapping.....	147
CAN Signal Mapping.....	147
Interrupts of the DS2211	149
Basics of DS2211 Interrupts.....	149
Interrupt Handling.....	151
Power Supply Unit	153
Characteristics and I/O Mapping of the Power Supply Unit.....	154
Controlling the Battery Voltage.....	156
Controlling the High Rails.....	158
How to Control the High Rails.....	160
Load Simulation	163
Connecting Loads.....	163
Failure Simulation	167
Safety Precautions for Failure Simulation.....	168
Failure Types.....	168
Failure Simulation for ECU Outputs.....	169
Make-Before-Break Switching.....	170
Failure Simulation for ECU Inputs.....	171
DS793 Sensor FIU.....	171
Diagnostics	175
Diagnostic Connector.....	175

Demo for the Simulator	177
Signal List of the Demo.....	178
Basics on Signal Lists.....	178
Structure of the Signal List.....	178
Working with the Template Signal List.....	182
How to Generate the CSV File.....	183
Demo Signal List.....	184
ECU_1 Demo Signal.....	184
SPARE ECU Demo Signal.....	187
SIMULATOR Demo Signal.....	187
MATLAB/Simulink Model of the Demo.....	188
Basics of the Simulink Model.....	188
Overview of the Simulink Model.....	188
Model User Interface.....	189
MDLUserInterface Subsystem.....	189
IO Subsystem.....	190
Overview of the IO Subsystem.....	191
ScalingToHardware.....	192
MappingToHardware.....	194
HardwareInterface.....	195
Protocols.....	196
Soft ECU.....	196
MappingFromHardware.....	197
ScalingFromHardware.....	198
I/O User Interface.....	200
IOUserInterface Subsystem.....	200
ControlDesk Experiment of the Demo.....	203
Layouts.....	203
Data Connections to ControlDesk Instruments.....	204
ECU_1 Layouts.....	206
SPARE Layouts.....	208
DS2211 Layouts.....	209
Additional Layouts.....	210
Failure Simulation.....	213
Failure Simulation.....	213

Expandability of the dSPACE Simulator Mid-Size	215
Using Additional I/O Boards or Signal Conditioning.....	215
Expanding the Processing Power.....	216
Expanding the I/O Hardware.....	216
Limitations	219
Quantization Effects.....	220
DS2211 Board Revision.....	220
Conflicting I/O Features.....	221
Limited Number of CAN Messages.....	235
Limitations with RTICANMM.....	237
Limitations with J1939-Support.....	241
Index	243









About This Document

Contents

This document provides feature-oriented access to the information you need to implement your models on dSPACE Simulator Mid-Size based on a DS2211.

Symbols

dSPACE user documentation uses the following symbols:

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files


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

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

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

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

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

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

Overview of the Simulator

Where to go from here

Information in this section

Overview of the Simulator.....	12
Giving an overview of the components features of dSPACE Simulator Mid-Size Based on DS2211.	
Block Diagram of the Simulator.....	13
The simulator consists of several hardware components.	
Functional Units.....	13
Describing the function units of the simulator.	

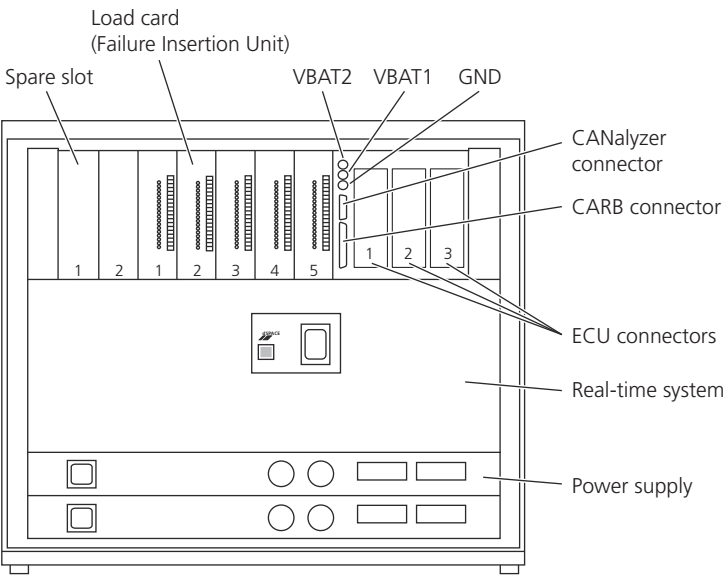
Information in other sections

Expandability of the dSPACE Simulator Mid-Size.....	215
If a standard dSPACE Simulator Mid-Size does not fulfill your requirements, it can be expanded.	
Limitations.....	219
There are some limitations you have to take into account when working with dSPACE Simulator Mid-Size Based on the DS2211.	

Overview of the Simulator

Introduction

The following gives you an overview of the components of dSPACE Simulator Mid-Size Based on DS2211.



The real-time system consists of a processor board and a HIL I/O board (DS2211).

Related topics

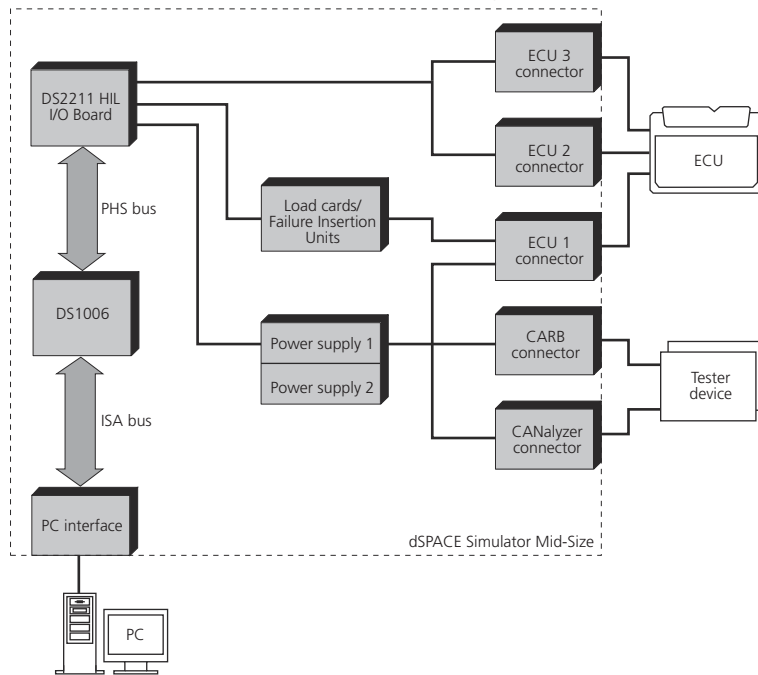
Basics

Block Diagram of the Simulator.....	13
Functional Units.....	13

Block Diagram of the Simulator

Introduction

The simulator consists of several hardware components.



dSPACE Simulator Mid-Size is mounted in a 19" desktop rack with 6, 9 or 12 HU (height units). It contains all necessary components for hardware-in-the-loop (HIL) simulation:

- DS1006 Processor Board
- DS2211 HIL I/O Board (1 or 2)
- PC interface
- Power supply (1 or 2)
- Load cards
- Failure Insertion Unit (FIU)

Functional Units

Processor board

The processor board calculates the real-time application. The dSPACE Simulator Mid-Size is equipped with a DS1006 Processor Board. For more information on the processor boards, refer to [Processor Board](#) on page 15.

DS2211 HIL I/O Board

The DS2211 HIL I/O Board is tailored to generate 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, 24 V, and 42 V automotive systems. For more information, refer to [DS2211 HIL I/O Board](#) on page 19.

PC interface

All dSPACE software for experiment setup and control runs on your PC or laptop, which is connected to the simulator via a PC interface. The simulator is equipped with the DS814 PC Interface board. The host PC has to be equipped with an interface board depending on its slot type:

Slot Type	Required Interface Board
ISA slot	DS813 plug-in card
PCI slot	DS817 plug-in card
PCMCIA slot	DS815 type 2 PCMCIA card

Power supply unit

The simulator can be equipped with one or two power supplies. The power supplies are used to simulate the battery voltage. The nominal voltages of the power supplies depend on your simulator (power supply 1 up to 30 V, power supply 2 up to 60 V). This is large enough to simulate the battery voltage of 12 V, 24 V, or 42 V automobile systems. The power supply is remote-controlled by the simulator. Each power supply is connected to a high rail system with 4 high rails, three of them are switchable (see [Power Supply Unit](#) on page 153).

Load cards

The load cards are used to connect substituted or real loads to the ECU outputs (see [Load Simulation](#) on page 163).

Failure Insertion Units

The Failure Insertion Units emulate electrical failures in the wiring of an ECU output, for example, short to ground or cable break (see [Failure Simulation](#) on page 167).

Connectors

The following connectors are available on the front:

- Three 90-pin connectors for the ECU
- One Sub-D CARB connector and one CANalyzer connector for diagnostic tools, see [Diagnostics](#) on page 175
- Jacks for the battery voltages for external devices
- Front connector of the load cards

The following connectors are available on the rear:

- Mains connector
- RS232-FIU connector

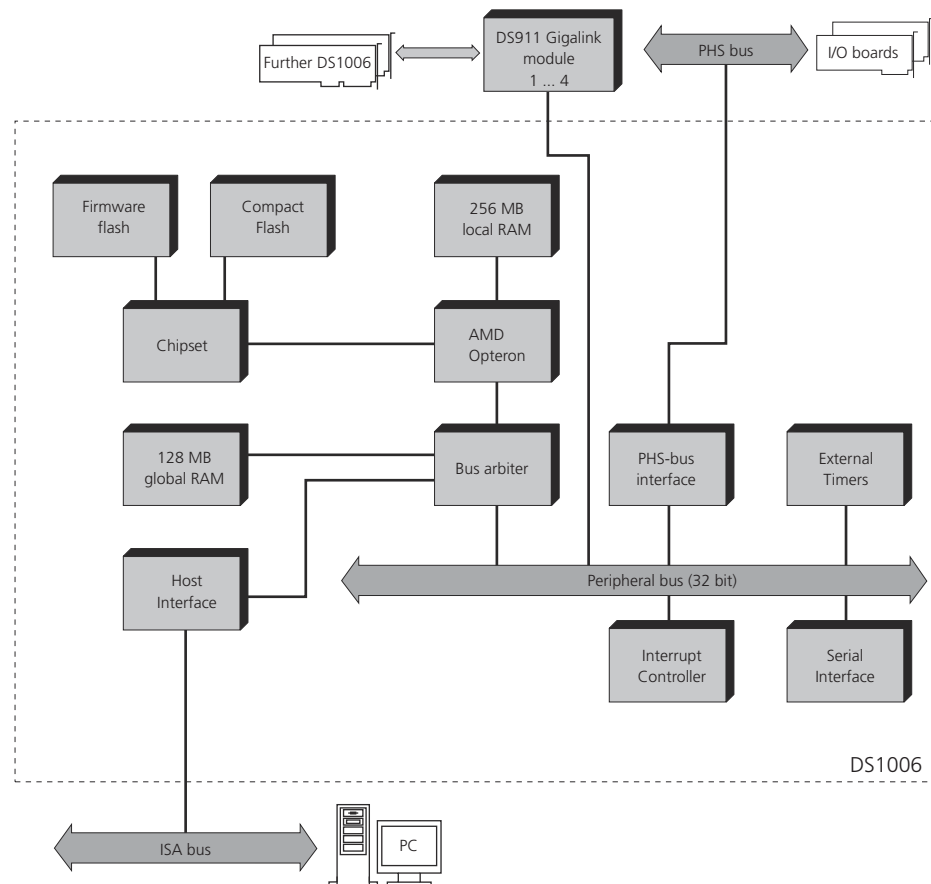
Processor Board

Introduction The processor board calculates the real-time model. dSPACE Simulator MidSize based on a DS2211 can be equipped with a DS1006 Processor Board.

Where to go from here	Information in this section
	DS1006 Architecture..... 15 Gives an overview of the functional units of a DS1006.
	DS1006 Feature Overview..... 16 Introduces the features of the DS1006 Processor Board.
	Information in other sections
	DS2211 HIL I/O Board..... 19 The simulator based on a DS2211 HIL I/O Board. This I/O board is tailored to generate and measure automotive signals.

DS1006 Architecture

Architecture The DS1006 Processor Board is based on an AMD Opteron™ processor. This real-time processor (RTP) forms the main processing unit. Via its PHS bus it can access modular I/O boards. Via the (optional) DS911 Gigalink Module it is capable of multiprocessing. The following illustration gives an overview of the functional units of the DS1006:



DS1006 Feature Overview

Introduction

The DS1006 Processor Board provides the following features.

DS1006 features

For detailed information, refer to the given section in [DS1006 Features](#) .

Processor Represents the computing power of the board. Refer to Processor.

Memory Comprises RAM, cache, and flash. Refer to Memory.

Timers and time base counters Comprises timers and time base counters:

- Timer A, Timer D: Sample rate timer with interrupt function. Refer to Timer A and Timer D.
- Timer B: Interval timer with interrupt function. Refer to Timer B.
- RTP built-in time base counter used for the time base for single-processor systems. Refer to Time-Stamp Counter.

- Synchronized time base unit used for the time base for multiprocessor systems. Refer to Global Time Base in an MP System.

Interrupt control Provides various hardware and software interrupts. Refer to Interrupt Controller.

PHS bus Allows access to up to 16 I/O boards. Refer to PHS-Bus Interface.

Gigalink module (optional) Connects DS1006 boards for setting up multiprocessor systems. Refer to DS911 Gigalink Module.

Synchronization Comprises synchronization features:

- Synchronized timer tasks: Timer A for multiprocessor systems. Refer to Synchronized Timer Tasks.
- Global time base in an MP system. Refer to Global Time Base in an MP System.
- Tracing signals in an MP system. Refer to Tracing Signals in an MP System.

Monitoring Allows monitoring of program execution and host access:

- Watchdog to monitor program execution. Refer to Watchdog.
- Force Reset to monitor host access. Refer to Force Reset.

Serial interface For setting up a user-specific communication or for debugging, etc. Refer to Serial Interface.

Host interface For setting up the DS1006, downloading programs and transferring run-time data to/from the host PC. Refer to Host Interface.

DS2211 HIL I/O Board

Introduction The DS2211 is tailored to generate 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, 24 V, and 42 V automotive systems.

Where to go from here	Information in this section
	System Overview of the DS2211..... 19 Providing an overview of the functional units and interfaces of the DS2211 HIL I/O Board.
	Features of the DS2211..... 22 Provides an alphabetical list of the features of the DS2211.
	Information in other sections
	Processor Board..... 15 The simulator contains a processor board which calculates the real-time model.

System Overview of the DS2211

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

The DS2211 is always part of dSPACE Simulator Mid-Size. While the DS2211 measures and simulates the signals required, the processor board performs the calculation of the real-time model. That is, applications using DS2211 I/O

features are implemented on the processor board. Together with a processor board, the DS2211 constitutes a basic HIL simulator.

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

Functional units

The DS2211 includes the following functional units:

Sensor and actuator interface The DS2211 contains a sensor and actuator interface that provides a typical set of automotive I/O functions, including A/D conversion, digital I/O, wheel speed sensor signal generation, and others.

Angular processing unit The most important feature of the DS2211 is the angular processing unit (APU), which provides the engine HIL core functions:

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

The angular processing unit can be used to simulate different engine variants.

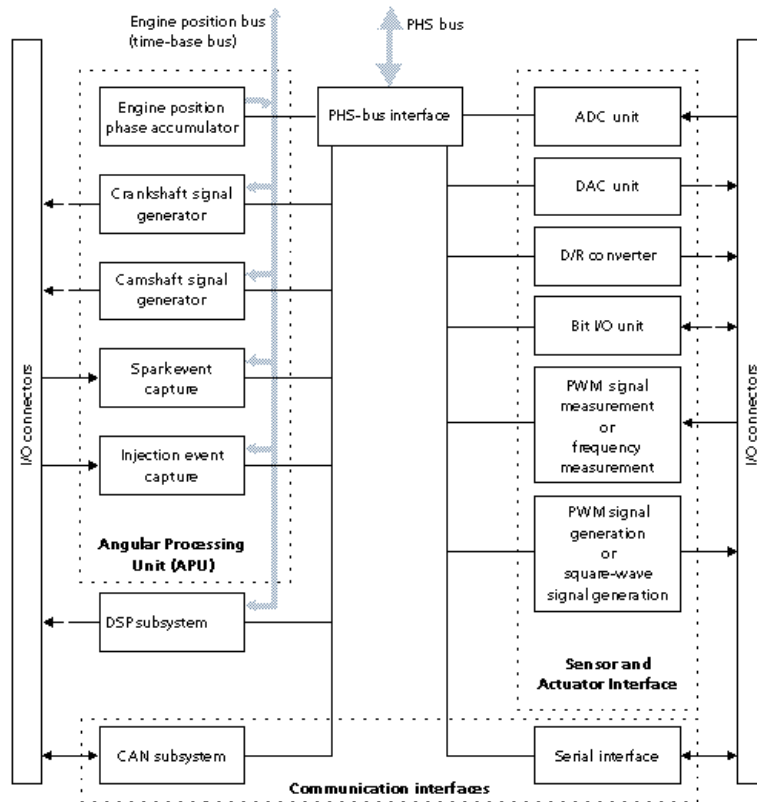
DSP subsystem The DSP subsystem is based on the TMS320VC33. It includes ready-to-use applications that allow you to generate knock sensor signals or wheel speed sensor signals. As an alternative, you can program the DSP to generate user-specific signals. The serial port of the slave DSP allows you to connect a DS2302 board, for example.

Communication interfaces In addition to a standard serial interface (RS232, RS422 based on a Texas Instruments TL 16C550 UART), the DS2211 includes a CAN subsystem that is based on the Siemens SAB 80C164 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 interprocessor communication, the processor board is the master, and the DS2211 microcontrollers are slaves.

Voltage systems The DS2211 supports two independent voltage systems with a voltage in the range 5 ... 60 V. The voltage come from the power supply unit.

The illustration shows the functional units of the DS2211.



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

Cascading DS2211 boards

Several DS2211 boards can be cascaded to expand from 8-cylinder simulation to 16-cylinder simulation, or even more via the engine position bus.

Further information

Use scenarios Use scenarios demonstrates how you can simulate complex tasks with the DS2211. Refer to [Use Scenarios \(DS2211 Features !\[\]\(830769b31eeeaca920791081939ff8ba_img.jpg\)](#)).

Limitations There are some limitations when you work with the DS2211. See [Limitations](#) on page 219.

Features of the DS2211

Introduction

The DS2211 provides the following features, summarized in alphabetical order.

DS2211 features

A/D conversion The ADC unit provides 16 unipolar A/D channels with 14-bit resolution and 20 μ s conversion time for all channels. You cannot use all A/D channels for your application because some A/D channels are already used for remote-controlling the power supply. Refer to [ADC Unit](#) on page 26.

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 33.

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 109.

D/A conversion The DAC unit provides 20 unipolar D/A channels (for user output) with 12-bit resolution and 20 μ s full-scale settling time to 1 least significant bit (LSB). The reference voltage is internally connected to 10 V and cannot be changed. You cannot use all D/A channels for your application because some D/A channels are already used for remote-controlling the power supply. Refer to [DAC Unit](#) on page 29.

D/R conversion The D/R converter provides 10 independent resistance outputs with 16-bit resolution covering a resistance range of 15.8 Ω ... ∞ . Refer to [D/R Converter](#) on page 31.

Engine HIL simulation The angular processing unit provides the following features:

- Simulation of engine core functions is based on a 16-bit engine position (angle) for an engine cycle of 0 ... 720° and a resolution of 0.011°. The engine position is updated every 250 ns. Refer to [Engine Position Phase Accumulator](#) on page 59.
- Crankshaft sensor simulation provides one crankshaft wave form output with 8 selectable wave forms. Refer to [Crankshaft Signal Generator](#) on page 61 and [Crankshaft Sensor Signal Generation](#) on page 88.
- Camshaft sensor simulation provides 2 camshaft wave form outputs with analog or digital signal and 2 camshaft wave form outputs with digital signal. Each output has 8 selectable wave forms. Refer to [Camshaft Signal Generator](#) on page 65 and [Camshaft Sensor Signal Generation](#) on page 90.
- 6 different interrupts can be generated depending on the engine position. Refer to [APU Overview](#) on page 57 and [Basics of DS2211 Interrupts](#) on page 149.

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

- For spark event capture, 8 digital ignition inputs (6 ignition and 2 auxiliary channels) for up to 8-cylinder engines are available. The two auxiliary channels can be individually configured either for various position measurements or for ignition capture. You can define up to two event capture windows for each

digital ignition input. Refer to [Spark Event Capture Unit](#) on page 67 and [Spark Event Capture](#) on page 93.

- For injection pulse position and fuel amount measurement, 8 digital injection inputs are available with up to two event capture windows for each channel. Refer to [Injection Event Capture Unit](#) on page 68 and [Injection Pulse Position and Fuel Amount Measurement](#) on page 95.

Tip

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

- You can define threshold voltages and hysteresis for the inputs of the spark event capture, injection pulse position and fuel amount measurement. Refer to [Complex Comparators](#) on page 77.

Frequency measurement For frequency measurement, 24 channels are available with a resolution of 16 bit. Refer to [Frequency Measurement \(F2D\)](#) on page 43.

Interrupt control The DS2211 PHS bus interrupt controller provides 8 hardware interrupts for the serial interface, the CAN subsystem, and the angular processing unit. Refer to [Basics of DS2211 Interrupts](#) on page 149.

Ionic current signal measurement Using the interrupt from the complex comparators, the DS2211 can be used for measuring ionic current signals. For an example, see [Simulating Ionic Current Measurement \(DS2211 Features !\[\]\(2b376d1a92330ab09dad2665d2f89bf5_img.jpg\)](#)).

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

Pulse generation For PWM signal generation, 9 independent outputs with run-time adjustable frequencies and duty cycles are available. Refer to [PWM Signal Generation](#) on page 40.

Pulse measurement For PWM signal measurement, 24 independent input channels are available. Refer to [PWM Signal Measurement](#) on page 36.

Serial interface The serial interface is based on the standard UART TL16C550C from Texas Instruments, which can be used in the RS232 or RS422 mode. Refer to [Serial Interface](#) on page 49.

Square-wave signal generation For square-wave signal generation, 9 channels are available with a resolution of 16 bit. Refer to [Square-Wave Signal Generation \(D2F\)](#) on page 46.

User-specific slave DSP applications You can program your own slave DSP applications to generate specific signals. Refer to [Basics of the Slave DSP](#) on page 100.

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

Related topics**Basics**

[System Overview of the DS2211.....](#) 19

Sensor and Actuator Interface

Introduction

The standard simulator has a DS2211, which offers a sensor and actuator interface providing standard I/O components and timing I/O components. Wheel speed sensor simulation is performed by a ready-to-use application implemented on the slave DSP.

Where to go from here

Information in this section

[Standard I/O.....26](#)

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

[Timing I/O.....36](#)

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

[Serial Interface.....49](#)

The board contains a universal asynchronous receiver and transmitter (UART) to communicate with external devices.

Information in other sections

[Features of the Slave DSP.....99](#)

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

Standard I/O

Introduction

Standard I/O includes the following components:

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

Where to go from here

Information in this section

ADC Unit.....	26
Introduction of the analog/digital converter (ADC) Unit	
DAC Unit.....	29
Introduction of the digital/analog converter (DAC) Unit	
D/R Converter.....	31
Introduction of the digital/resistance (D/R) converter	
Bit I/O Unit.....	33
Introduction of the bit (I/O) unit	

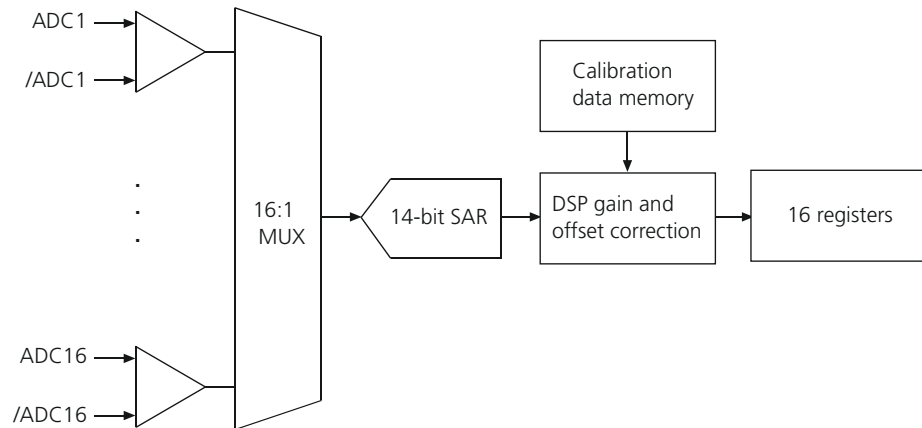
ADC Unit

Characteristics

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

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

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



Note

Note the following limitations if the DS2211 is built in the a dSPACE Simulator Mid-Size

- ADC inputs ADC1 to ADC12 are internally connected to the FIU and load cards. For more information, refer to [Load Simulation](#) on page 163.
- ADC13 is not connected to the FIU and load card. It can only be used if dSPACE Simulator Mid-Size has one power supply.
- ADC inputs ADC1 to ADC13 are differential inputs. They have ground sense lines ($\overline{\text{ADCx}}$). The ADCs measure the voltage difference of ($\text{ADCx} - \overline{\text{ADCx}}$). The $\overline{\text{ADCx}}$ lines function as individual ground sense line for each input. They must be connected to the ground of your system near the sensor, or to SGND/MIDGND inside the simulator, for all ADC channels used. SGND is a common ECU reference ground, MIDGND is the DS685 midplane reference ground.
- The ADC15 and ADC16 inputs are used for remote-controlling the power supply VBAT1.
- The ADC13 and ADC14 inputs are used for remote-controlling the power supply VBAT2.

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(e474458956c9a37fbf9586ddb60a7fa1_img.jpg\)](#)).

I/O mapping

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

A/D Channel	Signal	ECU Connector	Pin	Description	Voltage Range
1	ADC1	1	E3	14-bit ADC	(ADC1 – $\overline{\text{ADC1}}$) = 0 ... 60 V
	$\overline{\text{ADC1}}$	3	B11		

A/D Channel	Signal	ECU Connector	Pin	Description	Voltage Range
2	ADC2	1	E4	14-bit ADC	$(ADC2 - \overline{ADC2}) = 0 \dots 60 \text{ V}$
	$\overline{ADC2}$	3	C11		
3	ADC3	1	E5	14-bit ADC	$(ADC3 - \overline{ADC3}) = 0 \dots 60 \text{ V}$
	$\overline{ADC3}$	3	A12		
4	ADC4	1	E6	14-bit ADC	$(ADC4 - \overline{ADC4}) = 0 \dots 60 \text{ V}$
	$\overline{ADC4}$	3	B12		
5	ADC5	1	E7	14-bit ADC	$(ADC5 - \overline{ADC5}) = 0 \dots 60 \text{ V}$
	$\overline{ADC5}$	3	C12		
6	ADC6	1	E8	14-bit ADC	$(ADC6 - \overline{ADC6}) = 0 \dots 60 \text{ V}$
	$\overline{ADC6}$	3	A13		
7	ADC7	1	E9	14-bit ADC	$(ADC7 - \overline{ADC7}) = 0 \dots 60 \text{ V}$
	$\overline{ADC7}$	3	B13		
8	ADC8	1	E10	14-bit ADC	$(ADC8 - \overline{ADC8}) = 0 \dots 60 \text{ V}$
	$\overline{ADC8}$	3	C13		
9	ADC9	1	E11	14-bit ADC	$(ADC9 - \overline{ADC9}) = 0 \dots 60 \text{ V}$
	$\overline{ADC9}$	3	A14		
10	ADC10	1	E12	14-bit ADC	$(ADC10 - \overline{ADC10}) = 0 \dots 60 \text{ V}$
	$\overline{ADC10}$	3	B14		
11	ADC11	1	E13	14-bit ADC	$(ADC11 - \overline{ADC11}) = 0 \dots 60 \text{ V}$
	$\overline{ADC11}$	3	C14		
12	ADC12	1	E14	14-bit ADC	$(ADC12 - \overline{ADC12}) = 0 \dots 60 \text{ V}$
	$\overline{ADC12}$	3	A15		
13	ADC13	2	A1	14-bit ADC ¹⁾	$(ADC13 - \overline{ADC13}) = 0 \dots 60 \text{ V}$
	$\overline{ADC13}$	2	B1		
14	ADC14	–	–	14-bit ADC ²⁾	–
	$\overline{ADC14}$	–	–		
15	ADC15	–	–	14-bit ADC ³⁾	–
	$\overline{ADC15}$	–	–		
16	ADC16	–	–	14-bit ADC ³⁾	–
	$\overline{ADC16}$	–	–		

¹⁾ ADC13 cannot be used if the simulator has two power supplies. In this case, it is used for remote-controlling of VBAT2.

²⁾ It is used for remote-controlling of VBAT2.

³⁾ ADC15 and ADC16 are used for remote-controlling of VBAT1.

Related topics

References

[ADC Unit \(DS2211 RTLib Reference !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)\)](#)
[Analog Inputs \(PHS Bus System Hardware Reference !\[\]\(d4257ae6a3e163e6d467b3eb87960fa1_img.jpg\)\)](#)
[DS2211MUX_ADC_Bx \(DS2211 RTI Reference !\[\]\(37da042f270bb1ebdb248503fcdcdd43_img.jpg\)\)](#)

DAC Unit

Characteristics

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

Note

- The DAC12 output is used for remote-controlling the power supply for VBAT1 by the simulator.
- The DAC11 output is used for remote-controlling the power supply for VBAT2 by the simulator. It is only available if your simulator has only one power supply.

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

The DAC unit works with an internal reference of $V_{REF} = 10$ V.

Note

DAC outputs are differential outputs in the range 0 ... 10 V. Each output has an individual ground sense line (\overline{DACx}), which must be connected to the ground (signal ground) pin of the ECU, for all DAC channels used. The DAC controls the voltage difference of ($DACx - \overline{DACx}$).

Tip

The \overline{DACx} pins can be switched to the signal ground inside the simulator. Refer to [Reference Signal Jumpers \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(799877f5c2f906134441300079881630_img.jpg\)\)](#).

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

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

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

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(5eb1325dfdc3f1cad8426726c0db51cd_img.jpg\)](#)).

I/O mapping

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

D/A Channel	Signal	ECU Connector	Pin	Description	Voltage Range/Output Current
1	DAC1	2	C1	12-bit DAC/20 μ s	$(\text{DAC1} - \overline{\text{DAC1}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC1}}$	2	D1		
2	DAC2	2	E1	12-bit DAC/20 μ s	$(\text{DAC2} - \overline{\text{DAC2}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC2}}$	2	A2		
3	DAC3	2	C2	12-bit DAC/20 μ s	$(\text{DAC3} - \overline{\text{DAC3}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC3}}$	2	D2		
4	DAC4	2	E2	12-bit DAC/20 μ s	$(\text{DAC4} - \overline{\text{DAC4}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC4}}$	2	A3		
5	DAC5	2	B3	12-bit DAC/20 μ s	$(\text{DAC5} - \overline{\text{DAC5}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC5}}$	2	C3		
6	DAC6	2	D3	12-bit DAC/20 μ s	$(\text{DAC6} - \overline{\text{DAC6}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC6}}$	2	E3		
7	DAC7	2	A4	12-bit DAC/20 μ s	$(\text{DAC7} - \overline{\text{DAC7}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC7}}$	2	B4		
8	DAC8	2	C4	12-bit DAC/20 μ s	$(\text{DAC8} - \overline{\text{DAC8}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC8}}$	2	D4		
9	DAC9	2	E4	12-bit DAC/20 μ s	$(\text{DAC9} - \overline{\text{DAC9}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC9}}$	2	A5		
10	DAC10	2	B5	12-bit DAC/20 μ s	$(\text{DAC10} - \overline{\text{DAC10}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC10}}$	2	C5		
11	DAC11	2	D5	12-bit DAC/20 μ s ¹⁾	$(\text{DAC11} - \overline{\text{DAC11}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC11}}$	2	E5		
12	DAC12	–	–	12-bit DAC/20 μ s ²⁾	–
	$\overline{\text{DAC12}}$	–	–		
13	DAC13	3	A2	12-bit DAC/20 μ s	$(\text{DAC13} - \overline{\text{DAC13}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC13}}$	3	A3		
14	DAC14	3	B3	12-bit DAC/20 μ s	$(\text{DAC14} - \overline{\text{DAC14}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC14}}$	3	C3		
15	DAC15	3	A4	12-bit DAC/20 μ s	$(\text{DAC15} - \overline{\text{DAC15}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC15}}$	3	B4		
16	DAC16	3	C4	12-bit DAC/20 μ s	$(\text{DAC16} - \overline{\text{DAC16}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC16}}$	3	A5		

D/A Channel	Signal	ECU Connector	Pin	Description	Voltage Range/Output Current
17	DAC17	3	B5	12-bit DAC/20 μ s	$(\text{DAC17} - \overline{\text{DAC17}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC17}}$	3	C5		
18	DAC18	3	A6	12-bit DAC/20 μ s	$(\text{DAC18} - \overline{\text{DAC18}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC18}}$	3	B6		
19	DAC19	3	C6	12-bit DAC/20 μ s	$(\text{DAC19} - \overline{\text{DAC19}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC19}}$	3	A7		
20	DAC20	3	C7	12-bit DAC/20 μ s	$(\text{DAC20} - \overline{\text{DAC20}}) = 0 \dots 10 \text{ V}; \pm 5 \text{ mA}$
	$\overline{\text{DAC20}}$	3	A8		

¹⁾ DAC11 cannot be used if the simulator has two power supplies. In this case, it is used for remote-controlling of VBAT2, see [Controlling the Battery Voltage](#) on page 156

²⁾ DAC12 is used by the simulator for remote-controlling of VBAT1, see [Controlling the Battery Voltage](#) on page 156

Related topics

References

[Analog Outputs \(DAC\) \(PHS Bus System Hardware Reference !\[\]\(8d0f0e0fe25b320c33272c52aec1fbca_img.jpg\)\)](#)
[DAC Unit \(DS2211 RTLib Reference !\[\]\(c1e4487e48462435243c9e117557e045_img.jpg\)\)](#)
[DS2211DAC_Bx_Cy \(DS2211 RTI Reference !\[\]\(8823fcf8e90563a144be0b7cea058423_img.jpg\)\)](#)
[Power Supply Outputs \(PHS Bus System Hardware Reference !\[\]\(38006c13f313e4b0b98e8b2a7226a5e4_img.jpg\)\)](#)

D/R Converter

Introduction

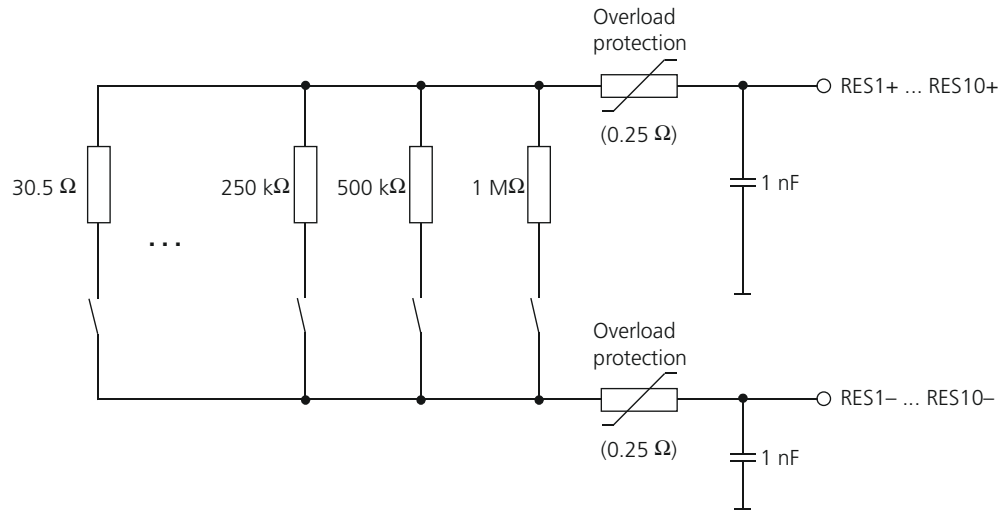
The D/R converter allows the simulation of sensors that have a resistance output, for example, thermistors or RTDs for temperature measurements. You can also use the D/R converter to simulate loose connections. To generate the desired resistance, your application writes a value to the D/R converter,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

Characteristics

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

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

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



Tip

You can connect several resistor channels

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

Note

Resistors are not grounded. Terminals must stay in the range ± 10 V to the ground (signal ground) of the ECU 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.

Tip

The RESx- pins can be switched to the signal ground inside the simulator. Refer to [Reference Signal Jumpers \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration\)](#).

Execution times

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

I/O mapping

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

Channel Number	Signal	ECU Connector	Pin	Description	Power/Output Current
1	RES1+	2	A6	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES1–	2	B6		
2	RES2+	2	C6	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES2–	2	D6		
3	RES3+	2	E6	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES3–	2	A7		
4	RES4+	2	C7	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES4–	2	D7		
5	RES5+	2	A8	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES5–	2	B8		
6	RES6+	2	C8	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES6–	2	D8		
7	RES7+	3	B8	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES7–	3	C8		
8	RES8+	3	B9	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES8–	3	C9		
9	RES9+	3	A10	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES9–	3	B10		
10	RES10+	3	C10	Resistance output	$P_{\max} = 250 \text{ mW}$; $I_{\max} = \pm 80 \text{ mA}$
	RES10–	3	A11		

Related topics**References**

[D/R Converter \(DS2211 RTLib Reference !\[\]\(e3f8612927870f2e0f9f5989e6dd3064_img.jpg\)](#))
[DS2211RES_Bx_Cy \(DS2211 RTI Reference !\[\]\(a86c7d1c9cb81c81614634a31267440d_img.jpg\)](#))

Bit I/O Unit

Characteristics

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

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

digital outputs are set by the bit I/O unit and the slave DSP using an OR operation.

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

Digital outputs The outputs have push-pull drivers running from an external source (2 independent VBAT rails, VBAT1 operable in the range 5 ... 30 V, VBAT2 operable in the range 5 ... 60 V). The maximum output current per channel is ± 50 mA. Short circuit protection to GND and VBAT or any voltage between GND and +30 V (+60 V) is implemented by self-resetting fuses, which have been designed for occasional faults only. They are therefore not suitable for failure simulation. The outputs are in their high impedance states after reset and power-up. You can enable or disable all outputs individually, using RTI/RTLib functions. A disabled output is in high impedance state. You can set the supply rail for each output individually (low side, high side VBAT1, or high side VBAT2) using RTI/RTLib functions.

Note

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

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(de95854c7ee024cfadc48187bbb781b2_img.jpg\)](#)).

I/O mapping

The following table shows the mapping of the channel numbers to the corresponding I/O pins of ECU connector, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. Refer to [Conflicting I/O Features](#) on page 221.

Channel Number	Signal	ECU Connector	Pin	Description	Voltage Range
1	DIG_IN1	1	C1	Digital input	12 – 42 V compatible
2	DIG_IN2	1	C2	Digital input	12 – 42 V compatible
3	DIG_IN3	1	C3	Digital input	12 – 42 V compatible
4	DIG_IN4	1	C4	Digital input	12 – 42 V compatible
5	DIG_IN5	1	C5	Digital input	12 – 42 V compatible

Channel Number	Signal	ECU Connector	Pin	Description	Voltage Range
6	DIG_IN6	1	C6	Digital input	12 – 42 V compatible
7	DIG_IN7	1	C7	Digital input	12 – 42 V compatible
8	DIG_IN8	1	C8	Digital input	12 – 42 V compatible
9	DIG_IN9	1	C9	Digital input	12 – 42 V compatible
10	DIG_IN10	1	C10	Digital input	12 – 42 V compatible
11	DIG_IN11	1	C11	Digital input	12 – 42 V compatible
12	DIG_IN12	1	C12	Digital input	12 – 42 V compatible
13	DIG_IN13	1	C13	Digital input	12 – 42 V compatible
14	DIG_IN14	1	C14	Digital input	12 – 42 V compatible
15	DIG_IN15	1	C15	Digital input	12 – 42 V compatible
16	DIG_IN16	1	C16	Digital input	12 – 42 V compatible
1	DIG_OUT1	2	E8	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
2	DIG_OUT2	2	B9	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
3	DIG_OUT3	2	C9	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
4	DIG_OUT4	2	D9	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
5	DIG_OUT5	2	E9	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
6	DIG_OUT6	2	A10	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
7	DIG_OUT7	2	B10	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
8	DIG_OUT8	2	C10	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
9	DIG_OUT9	2	D10	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
10	DIG_OUT10	2	E10	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
11	DIG_OUT11	2	A11	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
12	DIG_OUT12	2	B11	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
13	DIG_OUT13	2	C11	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
14	DIG_OUT14	2	D11	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
15	DIG_OUT15	2	E11	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
16	DIG_OUT16	2	A12	Digital output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA

Related topics

References

[Bit I/O Unit \(DS2211 RTI Reference !\[\]\(cbe80b694ebd74fcfe136a095b608235_img.jpg\)\)](#)

Timing I/O

Introduction

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

Where to go from here

Information in this section

PWM Signal Measurement.....	36
PWM signal measurement is used to capture digital signals in hardware-in-the-loop applications.	
PWM Signal Generation.....	40
9 independent PWM outputs are available for the generation of square-wave signals with a run-time adjustable frequency and run-time adjustable duty cycle.	
Frequency Measurement (F2D).....	43
24 independent channels are available to measure the frequency of square-wave signals.	
Square-Wave Signal Generation (D2F).....	46
9 independent channels are available to generate square-wave signals with variable frequencies.	

PWM Signal Measurement

Introduction

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

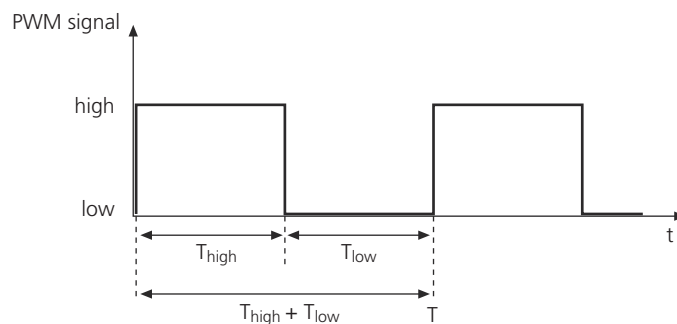
Note

- The channels can be used for either frequency measurement or PWM signal measurement.
- Due to quantization effects, you will encounter considerable deviations between the input PWM period T_P and the measured PWM period, especially for higher PWM frequencies. To avoid a poor frequency resolution, you should therefore select the frequency range with the best possible resolution (resolution values as small as possible). Refer to [Quantization Effects](#) on page 220.

Characteristics

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

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



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

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

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

Measurement range

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

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

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

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

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

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

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

Note

Signal periods and resolution

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

Update mode

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

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

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

Input voltage

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

Execution times

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

I/O mapping

The following table shows the mapping of the PWM channel numbers to the corresponding I/O pins of ECU connector, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 221.

PWM Channel	Signal	ECU Connector	Pin	Description	Voltage
1	PWM_IN1	1	C17	PWM signal measurement	12/42 V compatible
2	PWM_IN2	1	C18	PWM signal measurement	12/42 V compatible
3	PWM_IN3	1	D1	PWM signal measurement	12/42 V compatible
4	PWM_IN4	1	D2	PWM signal measurement	12/42 V compatible
5	PWM_IN5	1	D3	PWM signal measurement	12/42 V compatible
6	PWM_IN6	1	D4	PWM signal measurement	12/42 V compatible
7	PWM_IN7	1	D5	PWM signal measurement	12/42 V compatible
8	PWM_IN8	1	D6	PWM signal measurement	12/42 V compatible
9	PWM_IN9 (DIG_IN1)	1	C1	PWM signal measurement (shared with digital input)	12/42 V compatible
10	PWM_IN10 (DIG_IN2)	1	C2	PWM signal measurement (shared with digital input)	12/42 V compatible
11	PWM_IN11 (DIG_IN3)	1	C3	PWM signal measurement (shared with digital input)	12/42 V compatible
12	PWM_IN12 (DIG_IN4)	1	C4	PWM signal measurement (shared with digital input)	12/42 V compatible
13	PWM_IN13 (DIG_IN5)	1	C5	PWM signal measurement (shared with digital input)	12/42 V compatible
14	PWM_IN14 (DIG_IN6)	1	C6	PWM signal measurement (shared with digital input)	12/42 V compatible
15	PWM_IN15 (DIG_IN7)	1	C7	PWM signal measurement (shared with digital input)	12/42 V compatible
16	PWM_IN16 (DIG_IN8)	1	C8	PWM signal measurement (shared with digital input)	12/42 V compatible
17	PWM_IN17 (DIG_IN9)	1	C9	PWM signal measurement (shared with digital input)	12/42 V compatible
18	PWM_IN18 (DIG_IN10)	1	C10	PWM signal measurement (shared with digital input)	12/42 V compatible
19	PWM_IN19 (DIG_IN11)	1	C11	PWM signal measurement (shared with digital input)	12/42 V compatible

PWM Channel	Signal	ECU Connector	Pin	Description	Voltage
20	PWM_IN20 (DIG_IN12)	1	C 12	PWM signal measurement (shared with digital input)	12/42 V compatible
21	PWM_IN21 (DIG_IN13)	1	C 13	PWM signal measurement (shared with digital input)	12/42 V compatible
22	PWM_IN22 (DIG_IN14)	1	C 14	PWM signal measurement (shared with digital input)	12/42 V compatible
23	PWM_IN23 (DIG_IN15)	1	C 15	PWM signal measurement (shared with digital input)	12/42 V compatible
24	PWM_IN24 (DIG_IN16)	1	C 16	PWM signal measurement (shared with digital input)	12/42 V compatible

Related topics

Basics

[PWM Signal Generation](#)..... 40

References

[PWM Signal Measurement \(DS2211 RTI Reference !\[\]\(fa6f3af6bfa46c5d4a2d362681095beb_img.jpg\)](#))

PWM Signal Generation

Characteristics

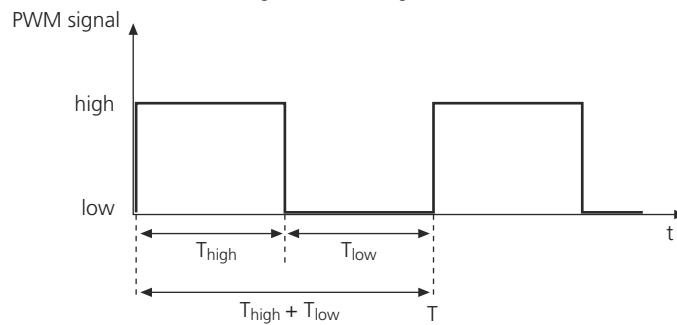
9 independent PWM outputs are available for the generation of square-wave signals with a run-time adjustable frequency and run-time adjustable duty cycle.

Note

- Before operating the digital outputs of PWM signal generation, you must connect an external power supply (V_{Bat}) to at least one of the two VBAT supply rails.
- The channels can be used for either square-wave signal generation or PWM signal generation.
- Due to quantization effects, you may encounter considerable deviations between the desired PWM period T_P and the generated PWM period, especially for higher PWM frequencies. To avoid a poor frequency resolution, you should therefore select the frequency range with the best possible resolution (resolution values as small as possible). Refer to [Quantization Effects](#) on page 220.

At 16-bit resolution, you can set the duty cycle in the range 0 ... 100% for PWM frequencies in the range 0.01 Hz ... 100 kHz. The duty cycle values 0 and 1 yield

a constant low or constant high output signal. The following illustration shows how the duty cycle = $(T_{\text{high}} / (T_{\text{low}} + T_{\text{high}}))$ is defined.



Limits and resolution

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

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

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

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

Output voltage

The outputs have push-pull drivers running from an external source (2 independent VBAT rails, VBAT1 can operate in the range 5 ... 30 V, VBAT2 can operate in the range 5 ... 60 V). The maximum output current per channel is ± 50 mA. Short circuit protection to GND and VBAT or any voltage between GND and +30 V (+60 V) is implemented by self-resetting fuses, which have been designed for occasional faults only. They are therefore not suitable for failure simulation. The outputs are in their high impedance states after reset and power-up.

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

Note

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

Update mode

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

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

Note

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

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

Execution times

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

I/O mapping

The following table shows the mapping of the logical PWM channel numbers to the related I/O pins of ECU connector, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 221.

PWM Channel	Signal	ECU Connector	Pin	Description	Voltage
1	PWM_OUT1	2	B12	PWM signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
2	PWM_OUT2	2	C12	PWM signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
3	PWM_OUT3	2	D12	PWM signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
4	PWM_OUT4	2	E12	PWM signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
5	PWM_OUT5	2	A13	PWM signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
6	PWM_OUT6	2	B13	PWM signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
7	PWM_OUT7	3	A17	PWM signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
8	PWM_OUT8	3	B17	PWM signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
9	PWM_OUT9	3	B18	PWM signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA

Related topics**References**

[PWM Signal Generation \(DS2211 RTI Reference !\[\]\(cbe2492b119e39e02a1dab2af4a4b296_img.jpg\)\)](#)

Frequency Measurement (F2D)

Characteristics

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

Note

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

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

Measurement range

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

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	100 kHz	50 ns
2	4.77 Hz	100 kHz	100 ns
3	2.39 Hz	100 kHz	200 ns
4	1.19 Hz	100 kHz	400 ns
5	0.60 Hz	100 kHz	800 ns
6	0.30 Hz	100 kHz	1.6 μ s
7	0.15 Hz	78.12 kHz	3.2 μ s
8	75 mHz	39.06 kHz	6.4 μ s
9	38 mHz	19.53 kHz	12.8 μ s
10	19 mHz	9.76 kHz	25.6 μ s
11	10 mHz	4.88 kHz	51.2 μ s
12	5.0 mHz	2.44 kHz	102 μ s
13	2.5 mHz	1.22 kHz	205 μ s
14	1.2 mHz	610.35 Hz	410 μ s
15	0.6 mHz	305.17 Hz	819 μ s
16	0.3 mHz	152.59 Hz	1.64 ms

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

Range	Restriction
Frequency < f_{\min}	The measured frequency is measured as a 0 Hz signal.
Frequency > f_{\max}	Faulty measurement because of quantization problems, refer to Quantization Effects on page 220.

Note

Signal periods and resolution

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

Input voltage

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

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(dd161862f9164df98f62b726e9846241_img.jpg\)](#)).

I/O mapping

The following table shows the mapping of the PWM channel numbers to the corresponding I/O pins of ECU connector, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 221.

PWM Channel	Signal	ECU Connector	Pin	Description	Voltage
1	PWM_IN1	1	C17	Frequency measurement	12/42 V compatible
2	PWM_IN2	1	C18	Frequency measurement	12/42 V compatible
3	PWM_IN3	1	D1	Frequency measurement	12/42 V compatible
4	PWM_IN4	1	D2	Frequency measurement	12/42 V compatible
5	PWM_IN5	1	D3	Frequency measurement	12/42 V compatible
6	PWM_IN6	1	D4	Frequency measurement	12/42 V compatible
7	PWM_IN7	1	D5	Frequency measurement	12/42 V compatible
8	PWM_IN8	1	D6	Frequency measurement	12/42 V compatible
9	PWM_IN9 (DIG_IN1)	1	C1	Frequency measurement (shared with digital input)	12/42 V compatible
10	PWM_IN10 (DIG_IN2)	1	C2	Frequency measurement (shared with digital input)	12/42 V compatible
11	PWM_IN11 (DIG_IN3)	1	C3	Frequency measurement (shared with digital input)	12/42 V compatible
12	PWM_IN12 (DIG_IN4)	1	C4	Frequency measurement (shared with digital input)	12/42 V compatible
13	PWM_IN13 (DIG_IN5)	1	C5	Frequency measurement (shared with digital input)	12/42 V compatible
14	PWM_IN14 (DIG_IN6)	1	C6	Frequency measurement (shared with digital input)	12/42 V compatible
15	PWM_IN15 (DIG_IN7)	1	C7	Frequency measurement (shared with digital input)	12/42 V compatible
16	PWM_IN16 (DIG_IN8)	1	C8	Frequency measurement (shared with digital input)	12/42 V compatible
17	PWM_IN17 (DIG_IN9)	1	C9	Frequency measurement (shared with digital input)	12/42 V compatible
18	PWM_IN18 (DIG_IN10)	1	C10	Frequency measurement (shared with digital input)	12/42 V compatible
19	PWM_IN19 (DIG_IN11)	1	C11	Frequency measurement (shared with digital input)	12/42 V compatible
20	PWM_IN20 (DIG_IN12)	1	C12	Frequency measurement (shared with digital input)	12/42 V compatible
21	PWM_IN21 (DIG_IN13)	1	C13	Frequency measurement (shared with digital input)	12/42 V compatible
22	PWM_IN22 (DIG_IN14)	1	C14	Frequency measurement (shared with digital input)	12/42 V compatible
23	PWM_IN23 (DIG_IN15)	1	C15	Frequency measurement (shared with digital input)	12/42 V compatible
24	PWM_IN24 (DIG_IN16)	1	C16	Frequency measurement (shared with digital input)	12/42 V compatible

Related topics

References

[Frequency Measurement \(DS2211 RTI Reference !\[\]\(eafc244b53721dd1ec133f0772f70fc7_img.jpg\)\)](#)

Square-Wave Signal Generation (D2F)

Characteristics

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

Note

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

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

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

Range	Minimum Frequency	Maximum Frequency	Resolution
1	9.54 Hz	100 kHz	50 ns
2	4.77 Hz	100 kHz	100 ns
3	2.39 Hz	100 kHz	200 ns
4	1.19 Hz	100 kHz	400 ns
5	0.60 Hz	100 kHz	800 ns
6	0.30 Hz	100 kHz	1.6 μ s
7	0.15 Hz	100 kHz	3.2 μ s
8	75 mHz	78.12 kHz	6.4 μ s
9	38 mHz	39.06 kHz	12.8 μ s
10	19 mHz	19.53 kHz	25.6 μ s
11	10 mHz	9.76 kHz	51.2 μ s
12	5.0 mHz	4.88 kHz	102 μ s
13	2.5 mHz	2.44 kHz	205 μ s
14	1.2 mHz	1.22 kHz	410 μ s

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

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

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

Output voltage

The outputs have push-pull drivers running from an external source (2 independent VBAT rails, VBAT1 can operate in the range 5 ... 30 V, VBAT2 can operate in the range 5 ... 60 V). The maximum output current per channel is ± 50 mA. Short circuit protection to GND and VBAT or any voltage between GND and +30 V (+60 V) is implemented by self-resetting fuses, which have been designed for occasional faults only. They are therefore not suitable for failure simulation. The outputs are in their high impedance states after reset and power-up. You can enable or disable all outputs individually, using RTI/RTLib functions. A disabled output is in high impedance state. Using RTI/RTLib functions, you can set the supply rail for each output individually (low side, high side VBAT1, or high side VBAT2).

Note

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

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(3cb60d42b10e53f9522bb0b392c1c4cd_img.jpg\)](#)).

I/O mapping

The following table shows the mapping of the logical PWM channel numbers to the corresponding I/O pins of ECU connector, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. Refer to [Conflicting I/O Features](#) on page 221.

PWM Channel	Signal	ECU Connector	Pin	Description	Voltage
1	PWM_OUT1	2	B12	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
2	PWM_OUT2	2	C12	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA

PWM Channel	Signal	ECU Connector	Pin	Description	Voltage
3	PWM_OUT3	2	D12	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
4	PWM_OUT4	2	E12	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
5	PWM_OUT5	2	A13	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
6	PWM_OUT6	2	B13	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
7	PWM_OUT7	3	A17	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
8	PWM_OUT8	3	B17	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
9	PWM_OUT9	3	B18	Square-wave signal generation	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA

Related topics

References

[Square-Wave Signal Generation \(DS2211 RTI Reference !\[\]\(6605b201d6f14d9b3bcb8ab5f274d107_img.jpg\)\)](#)

Serial Interface

Where to go from here

Information in this section

Basics on the Serial Interface	49
Provides information on board's <i>universal asynchronous receiver and transmitter</i> (UART) for performing serial asynchronous communication with external devices.	
Comparing RS232 and RS422	50
The dSPACE Simulator Mid-Size allows you to use the RS232 or RS422 transceiver mode.	
Specifying the Baud Rate of the Serial Interface	52
Provides information on the baud rate that you can specify for the board's serial interface.	
Software FIFO Buffer	52
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 50.
- 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 52.
- Selectable number of data bits, parity bit and stop bits
- Software FIFO buffer of selectable size. For details, see [Software FIFO Buffer](#) on page 52.

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

RTI/RTLib support

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

- RTI: [Serial Interface \(DS2211 RTI Reference !\[\]\(97faa0168e491544be255cfcab218e9b_img.jpg\)](#))
- RTLib: [Serial Interface Communication \(DS2211 RTLib Reference !\[\]\(b2166b76608b8499cffc130bf1b1fe60_img.jpg\)](#))

I/O mapping

The TXD, $\overline{\text{TXD}}$, RXD, and $\overline{\text{RXD}}$ signals are available at the ECU connector. For information on the I/O mapping, refer to [ECU 2 Connector Pinout \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(c50c8b7b2cc2cf9ff925edec0ee94c0d_img.jpg\)](#)).

Note

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

Comparing RS232 and RS422

Introduction

The dSPACE Simulator Mid-Size 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 mode 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

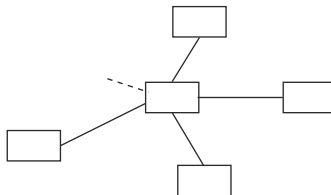
- Simple point-to-point connections



- Daisy-chain connections



- Backbone connections



Related topics

Basics

[Software FIFO Buffer..... 52](#)

Specifying the Baud Rate of the Serial Interface

Oscillator frequency

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

Baud rate range

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

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

Available baud rates

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

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

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

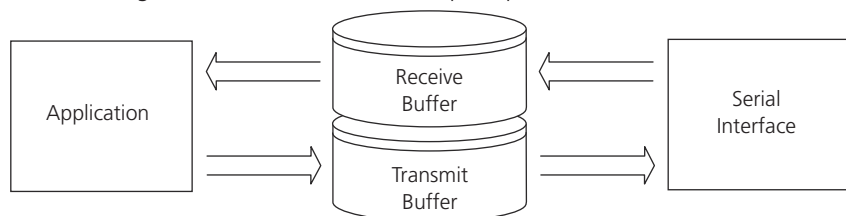
When you specify a baud rate in RTI or RTLib, the closest available baud rate is actually used for serial communication. For example, if you specify 70,000 baud as the baud rate, the baud rate used is 71,429 baud.

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.....](#) 50

Angular Processing Unit

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

Where to go from here

Information in this section

APU Basics.....	56
You can find basic information on how simulation of engine core functions works. The most important terms when working with RTI and RTLib are explained.	
Angular Processing Unit - Variant.....	81
If you use the VAR APU blockset, you can change the engine variant without the need to rebuild and download the real-time application.	
APU Reference.....	88
You can find the reference data required to implement the APU functions. This includes restrictions and limitations as well as the I/O mappings.	

APU Basics

Introduction

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

Where to go from here

Information in this section

APU Overview.....	57
Presenting the functional units of the APU and their interconnections.	
Engine Position Phase Accumulator.....	59
Explaining the unit that supplies the engine position.	
Cascading I/O Boards.....	60
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.....	61
Explaining how the crankshaft signal is generated and what wave tables are for.	
Reverse Crankshaft Rotation.....	62
Explaining how a reverse crankshaft signal is generated.	
Camshaft Signal Generator.....	65
Explaining how the camshaft signals are generated and what wave tables are for.	
Spark Event Capture Unit.....	67
Explaining how ignition pulses are captured.	
Injection Event Capture Unit.....	68
Explains how injection pulses are captured and how the fuel amount is evaluated.	
Event Capture Windows.....	72
Giving information on the definition of event capture windows.	
Continuous Value Capturing.....	76
Explaining how you can determine ignition position, injection pulse position and duration values continuously.	
Complex Comparators.....	77
Explains the complex comparator functionality which is needed for capturing complex signals, for example, the current through an injector.	

Information in other sections

[Features of the Slave DSP..... 99](#)

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

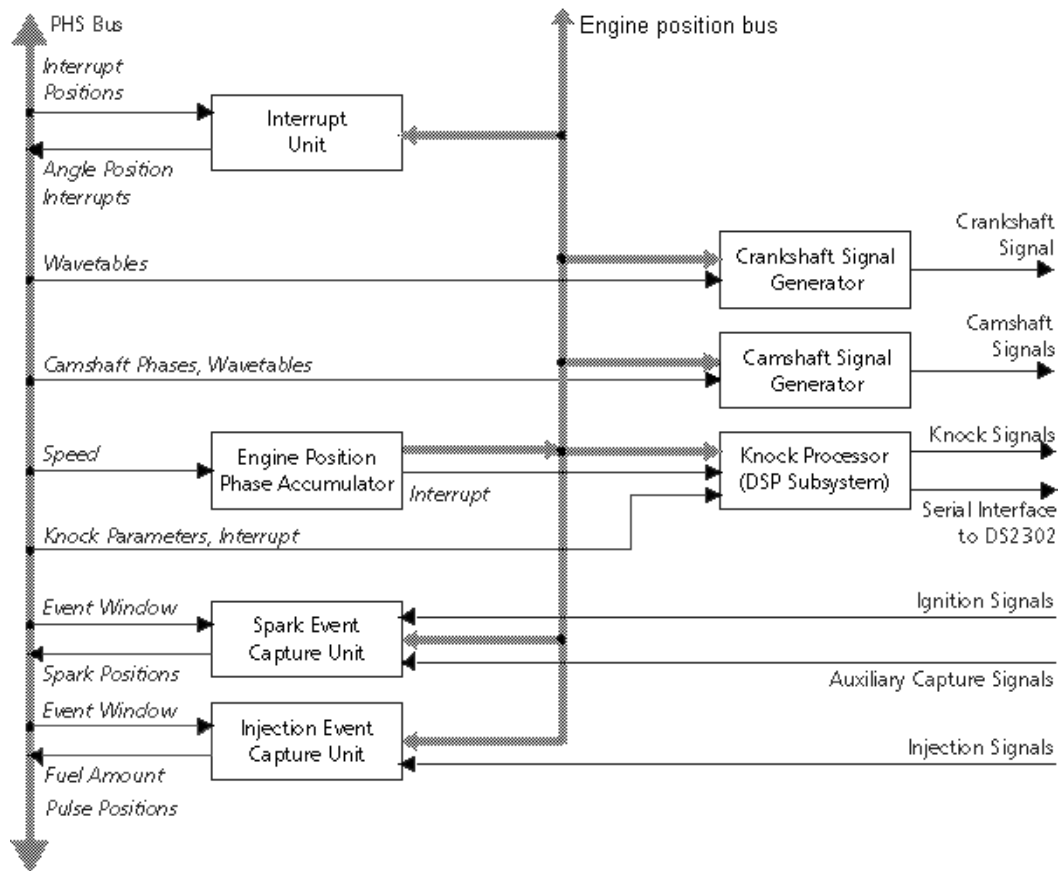
[Angular Processing Unit - Variant..... 81](#)

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

APU Overview

Introduction

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



Engine position bus

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

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

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

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

The boards are connected via the time-base connector. Refer to [Board Overview \(PHS Bus System Hardware Reference !\[\]\(a870788d6ed9b8fd294b7654a8c8526b_img.jpg\)](#)).

PHS bus

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

Angle position interrupt

Depending on the engine position (angle), the angular processing unit can generate interrupts for up to 6 angle positions. You can use these lines to request interrupts when the cylinder has passed the top dead center (TDC), for example. Any interrupt position can be chosen, and several interrupt positions are possible. For further information on interrupts, refer to [Interrupts of the DS2211](#) on page 149.

Related topics**Basics**

[APU Basics.....](#) 56

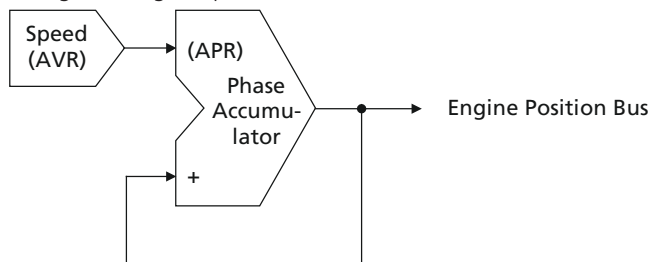
References

[APU Reference.....](#) 88
[DS2211 I/O Board \(PHS Bus System Hardware Reference !\[\]\(f1c5da15572e3e09d343161be98f508d_img.jpg\)](#))

Engine Position Phase Accumulator

Introduction

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



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

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

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

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

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

Related topics

References

APU Reference.....	88
--------------------	----

Cascading I/O Boards

Introduction

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

Functionality

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

Usable I/O boards

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

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


Note

The DS2302 board cannot be used as the bus master.

Limitation

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

Hardware settings

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

Software settings

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

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

Related topics

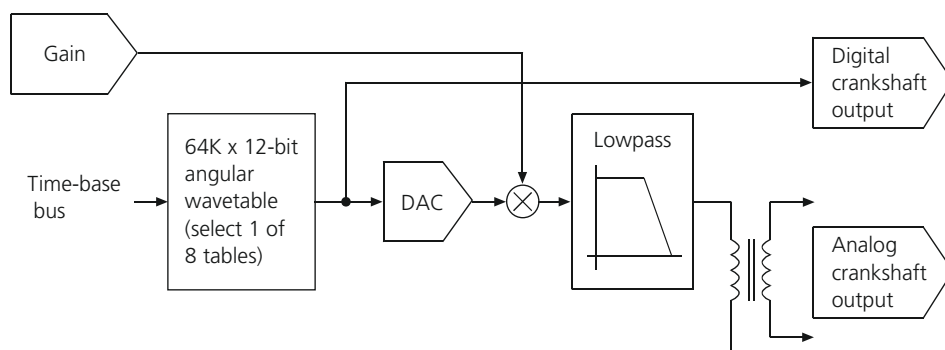
References

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

Crankshaft Signal Generator

Introduction

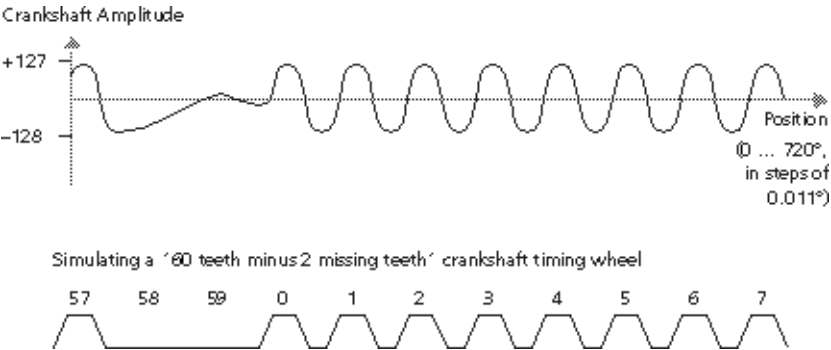
The crankshaft signal generator converts the engine position input to a crankshaft signal with analog and digital outputs. Conversion is done by using a wave table look-up mechanism. The analog output is fed through output transformers. The level of the analog output can be adapted to your application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



The angular wave table defines a signal level for each 0.011° engine position. The resolution is 12-bit signed ($-2048 \dots +2047$). Due to the output transformers, the wave table data must be without a mean value. The data is used to define the wave form. For more details on wave tables, refer to [Wave Table Generation](#) on page 92.

If you need to simulate forward *and* reverse cranking, refer to [Reverse Crankshaft Rotation](#) on page 62.

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



For reference information, including the I/O mapping, refer to [Crankshaft Sensor Signal Generation](#) on page 88.

Related topics

References

APU Reference.....	88
--------------------	----

Reverse Crankshaft Rotation

Introduction

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

Hardware requirements

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

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

Reverse crank sensor signal

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

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

Note

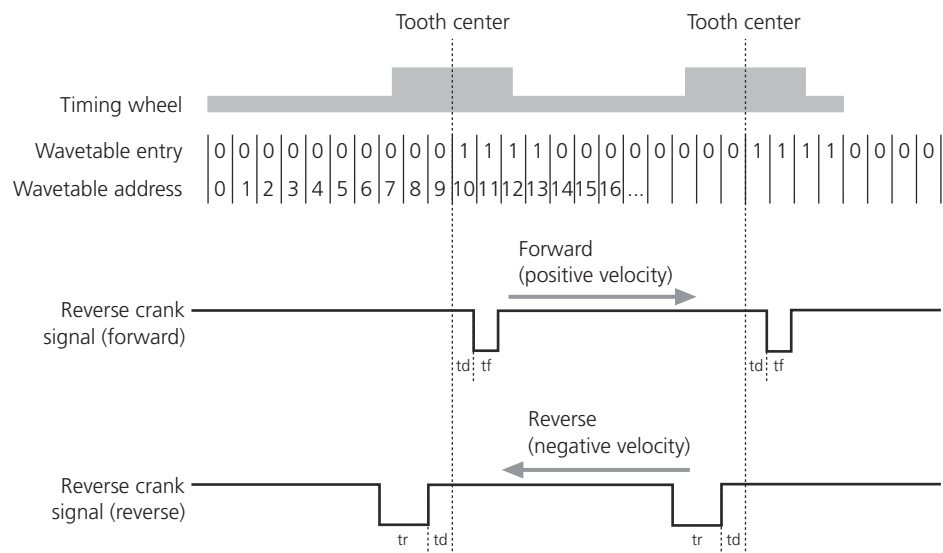
The reverse crank signal is generated as a digital signal. However, the analog crankshaft output is not automatically disabled and might generate an irrelevant analog signal.

Reverse crankshaft wave table

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

Note

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



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

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

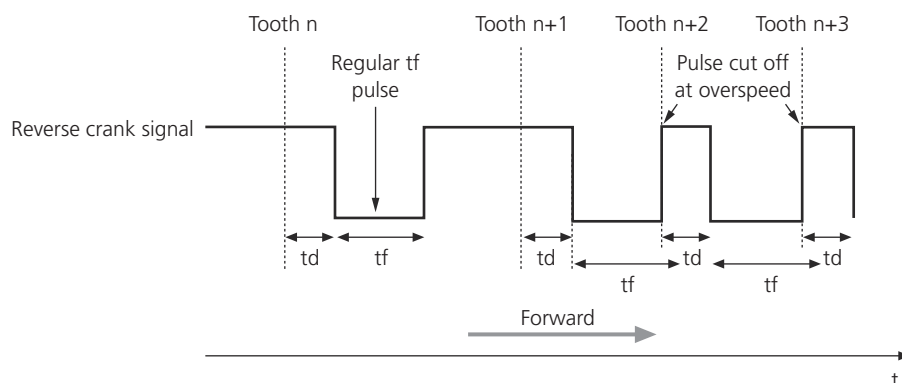
resulting engine positions then determine forward or backward stepping through the wave table addresses.

Minimum length of inactive pulses

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

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

Fast forward rotation

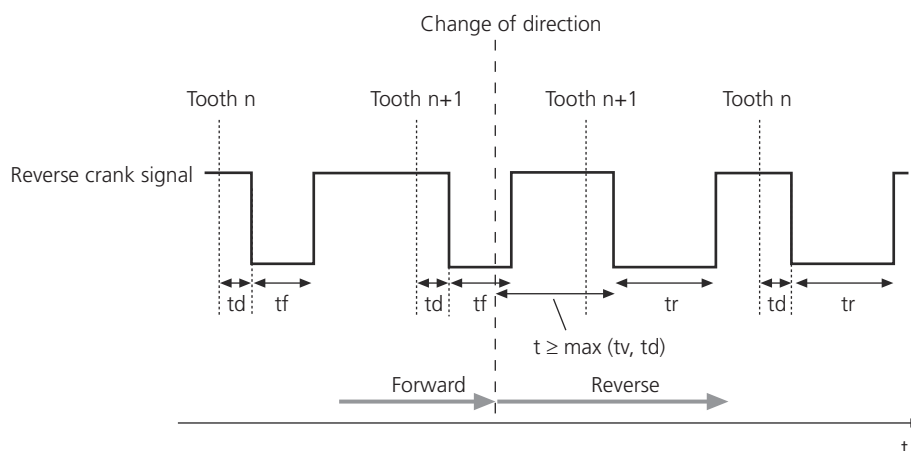


The illustration shows a low active reverse crank sensor signal. The rotation direction is always the same, and the time delay (t_d) is kept after each trigger point.

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

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

Change of direction



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

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

Related topics

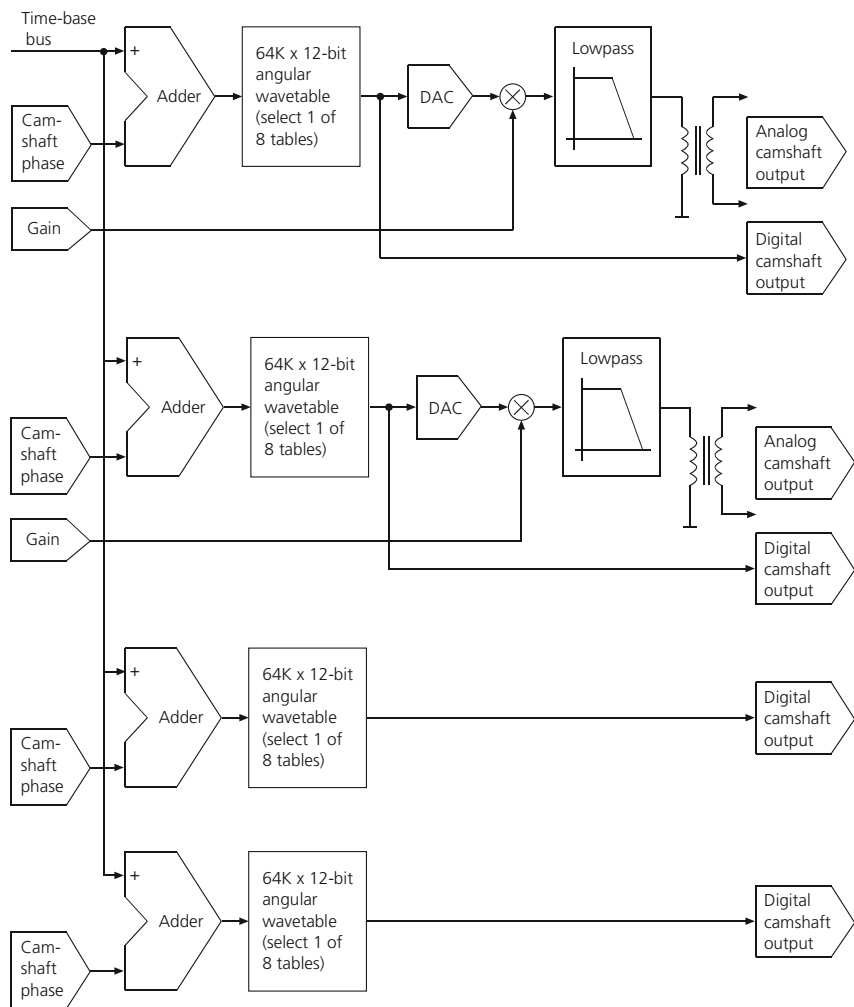
References

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

Camshaft Signal Generator

Introduction

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



The angular wave table defines a signal level for each engine position. The signal level resolution is 12-bit signed (–2048 ... +2047). Due to the output transformers, the wave table data must be without a mean value. The data is used to define the wave form. For more details on wave tables, refer to [Wave Table Generation](#) on page 92.

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

For reference information, including the I/O mapping, refer to [Camshaft Sensor Signal Generation](#) on page 90.

Related topics

References

APU Reference..... 88

Spark Event Capture Unit

Introduction

The spark event capture unit is designed for ignition position measurement with one or two definable event capture windows for up to 8 cylinders. Ignition pulses can be captured from the input lines IGN1 ... IGN6, AUXCAP1 and AUXCAP2. You can choose between active high and active low pulses. The voltage level when a pulse is captured can be defined with a complex comparator (refer to [Complex Comparators](#) on page 77).

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

Note

- If you do not use the channels for ignition capture, you can use them for injection capture.
- The channels of the spark event capture unit can be read simultaneously in the event capture mode and the continuous mode by using RTLib functions. Refer to [Continuous Value Capturing](#) on page 76.

Event capture windows can be defined according to the 16-bit engine position resolution (0.011°) provided by the engine position bus (refer also to [Event Capture Windows](#) on page 72).

Interrupt

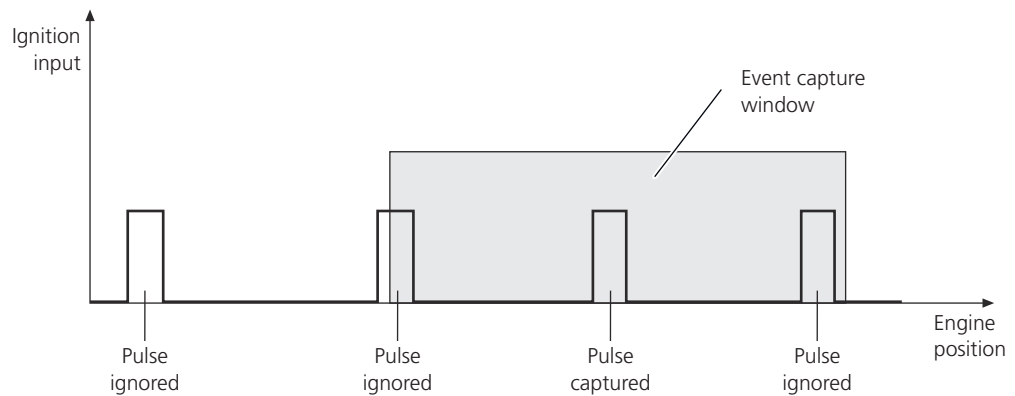
The spark event capture unit can trigger an interrupt on the slave DSP when an edge is detected. Refer to [Basics of DS2211 Interrupts](#) on page 149.

Capture modes

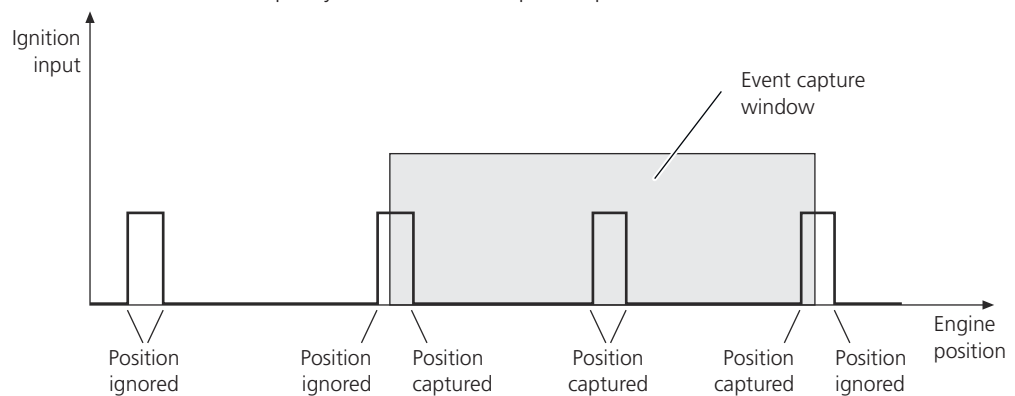
Three capture modes are available:

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

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



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



For reference information, including the I/O mapping, refer to [Spark Event Capture](#) on page 93.

Related topics

References

[APU Reference](#).....88

Injection Event Capture Unit

Introduction

The injection event capture unit is designed for injection position and fuel amount measurements with one or two definable event capture windows for up to 16 channels. You can choose between active high and active low pulses. The voltage level when a pulse is captured can be defined by a complex comparator (refer to [Complex Comparators](#) on page 77). The resolution for fuel amount

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

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

Note

- If you use the input lines of group 2 for injection capture, ignition capture is not possible.
- The channels of the injection event capture unit can be read simultaneously in the event capture mode and the continuous mode by using RTLib functions. Refer to [Continuous Value Capturing](#) on page 76.

Event capture windows can be defined according to the 16-bit engine position resolution (0.011°) provided by the engine position bus (refer also to [Event Capture Windows](#) on page 72).

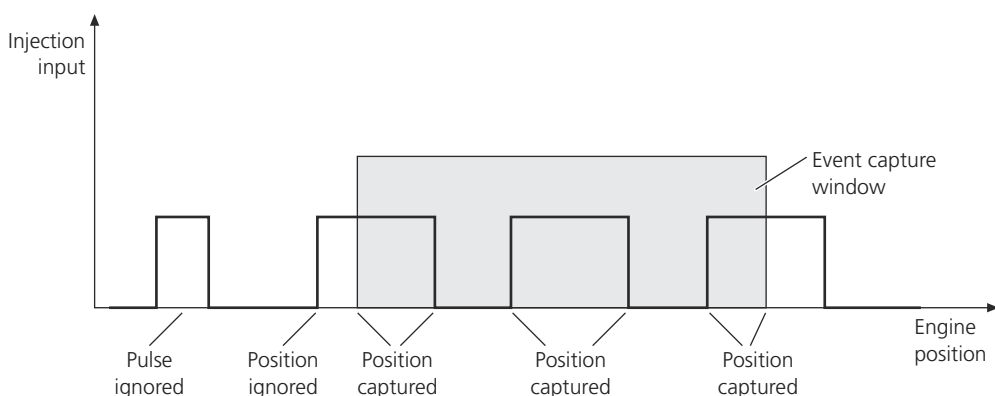
Interrupt

The injection event capture unit can trigger an interrupt on the slave DSP when an edge is detected. Refer to [Basics of DS2211 Interrupts](#) on page 149.

Capture modes

The following capture modes are available:

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



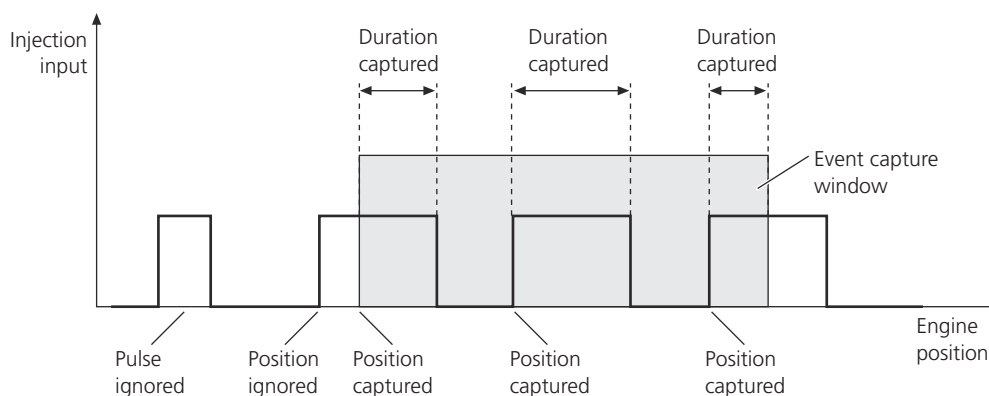
If either the leading or the trailing edge of a pulse is outside the event capture window, the position value of the corresponding window border is counted as an edge.

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

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

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

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

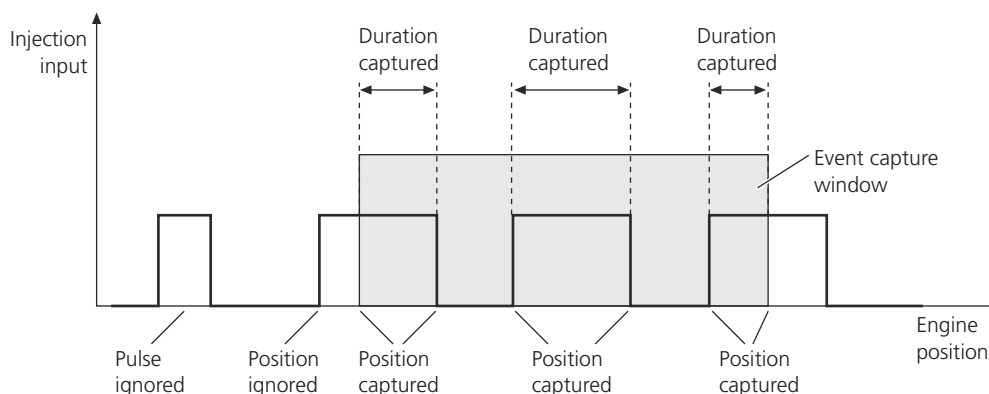


If either the leading or the trailing edge of a pulse is outside the event capture window, it is assumed that the pulse starts/ends at the respective border of the window.

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

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

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



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

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

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

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

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

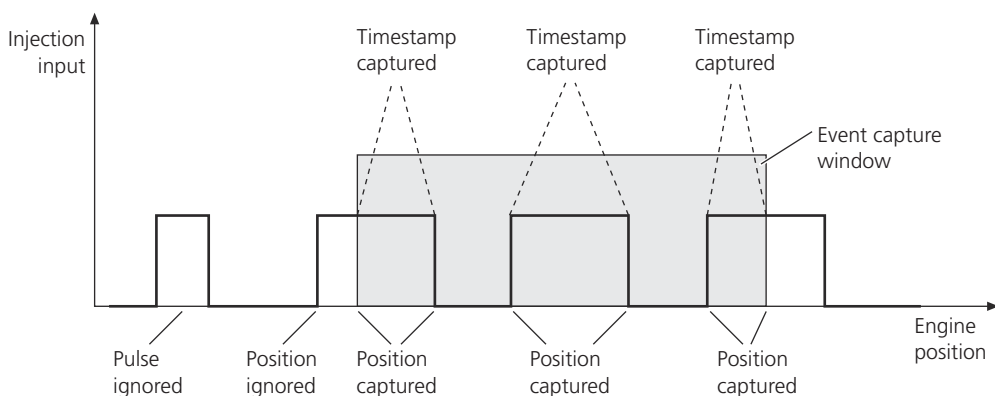
This mode can be used starting from board revision 3 and FPGA revision 3. The position values and time stamps of leading and trailing edges of up to 64 input pulses are evaluated within each event capture window.

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

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

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

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



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

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

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

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

For reference information including the I/O mapping, refer to [Injection Pulse Position and Fuel Amount Measurement](#) on page 95.

Related topics

Basics

[DS2211 Board Revision](#)..... 220

References

[APU Reference](#)..... 88

[DS2211_ABS_COUNTER_RESET \(DS2211 RTLib Reference !\[\]\(17acf1afa8cdf0b67c53d4865a5ed469_img.jpg\)\)](#)

[DS2211APU_ABS_CNT_RESET_Bx \(DS2211 RTI Reference !\[\]\(e8fb589d58dad1692debababa5e928b6_img.jpg\)\)](#)

Event Capture Windows

Introduction

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

Capture modes

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

- Single event capture mode
- Multiple event capture mode

Refer to [Spark Event Capture Unit](#) on page 67.

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

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

Refer to [Injection Event Capture Unit](#) on page 68.

Continuous value capture

For spark event and injection event capture, you can continuously measure up to 64 events per channel via the capture FIFO buffer. Continuous value capturing can cover the whole engine cycle of 0 ... 720° using one event capture window. Refer to [Continuous Value Capturing](#) on page 76.

Setting event capture windows

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

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

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

Restrictions

The following restrictions apply to event capture windows:

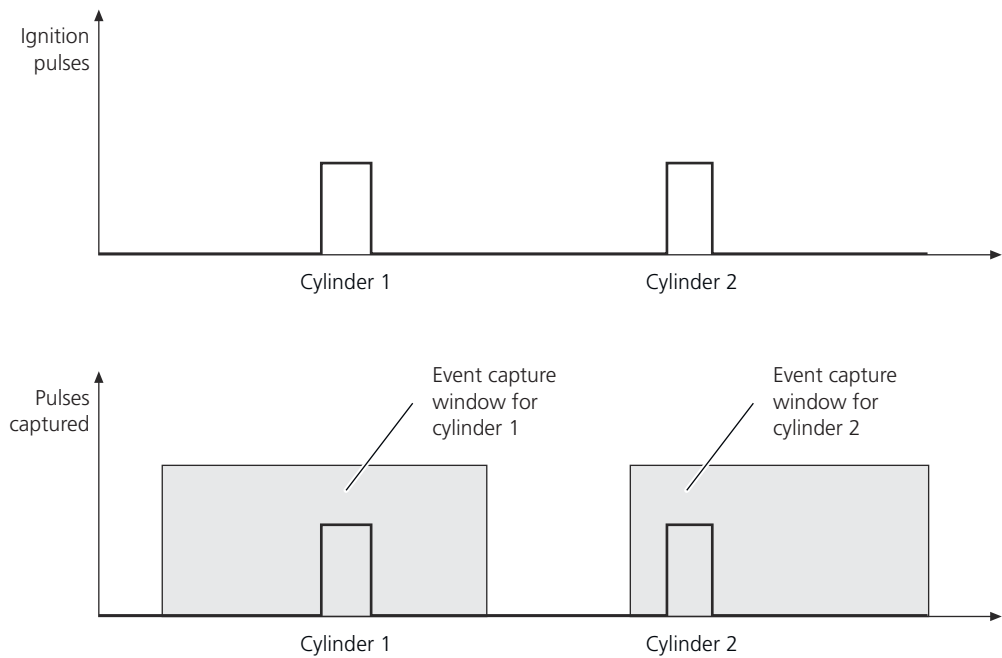
- An event window must not cover the whole engine cycle of 0 ... 720°. The maximum width of an event window is 719.824° (= 12.5633 rad).
Continuous value capturing is an exception. It allows you to cover the whole engine cycle of 0 ... 720°.
- The minimum width of an event window is 0.176° (= 0.003 rad).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

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

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

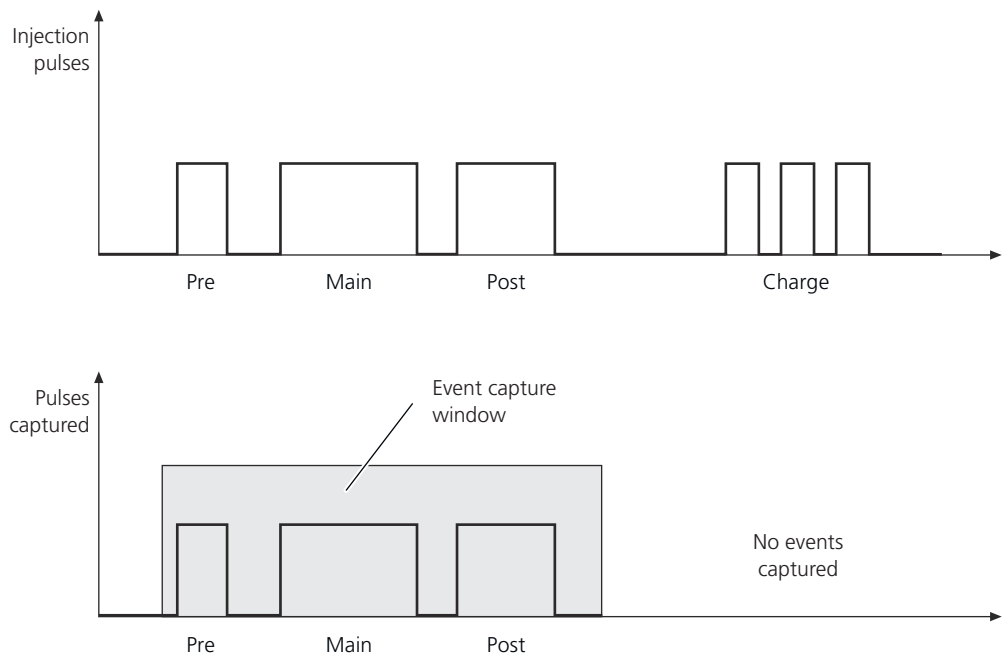
Example of Double Spark Ignition

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

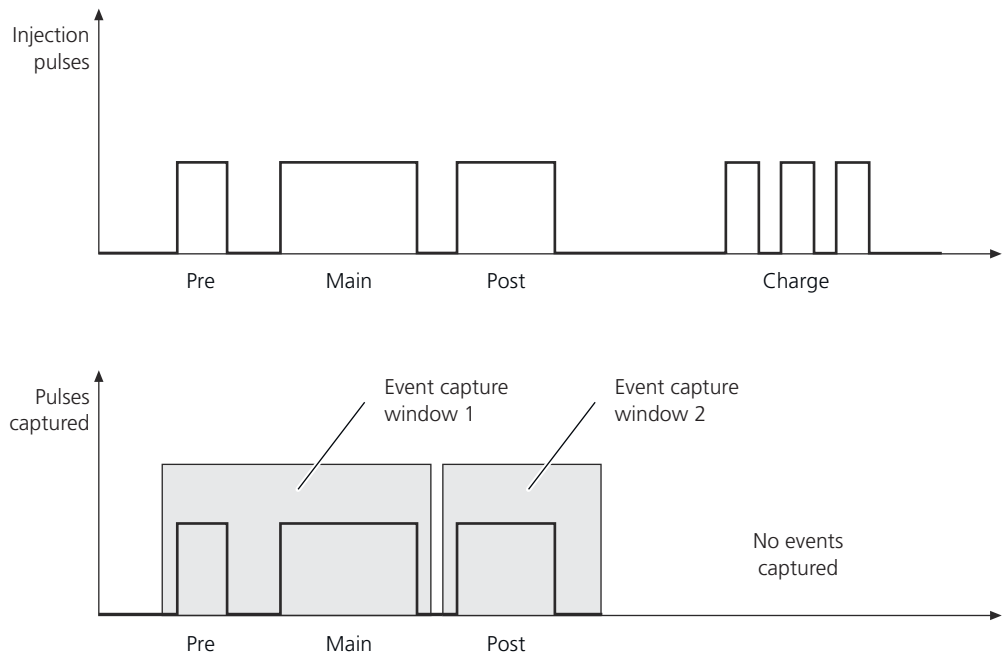


Example of Common Rail Injection

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

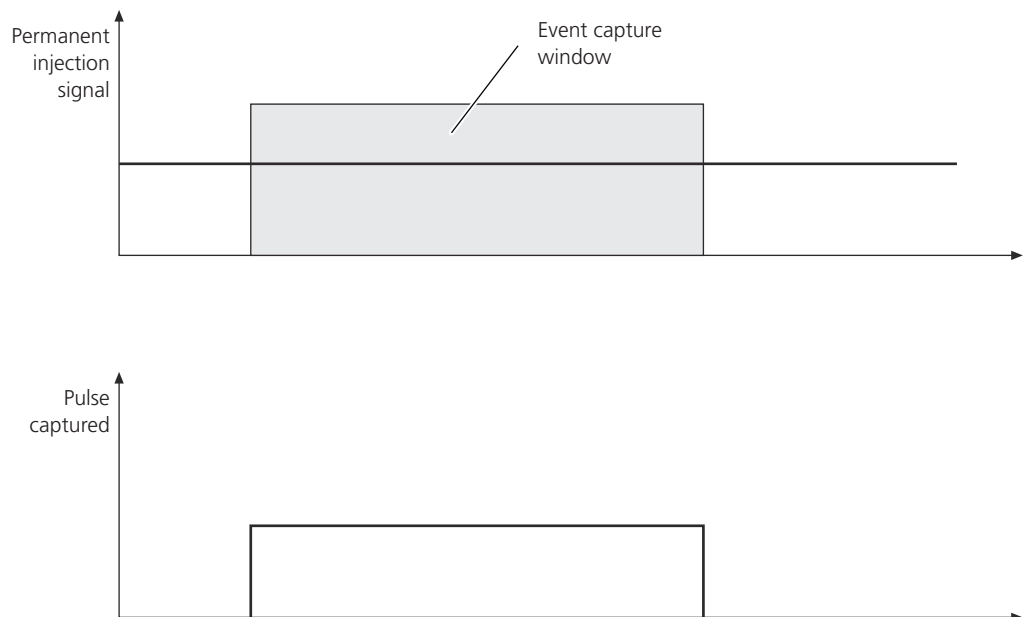


You can use two event capture windows to distinguish between torque and exhaust relevant injection.



Example of Permanent Injection

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



Related topics**Basics**

Continuous Value Capturing.....	76
Injection Event Capture Unit.....	68
Spark Event Capture Unit.....	67

Continuous Value Capturing

Introduction

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

Spark event capture

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

Injection pulse position and duration

In position mode, the position values of all trailing and leading edges of up to 32 pulses (= 64 events) per channel can be read within one sample hit (the sample hit is defined by the sample time of your model). In duration mode, the position values of all leading edges and the duration of up to 32 pulses per channel within one sample hit can be read (refer to [Injection Event Capture Unit](#) on page 68).

Auxiliary event capture

Two channels are available to measure various positions. In single event mode, up to 64 position values of the leading edge of the first input pulse are evaluated. In multiple event mode, up to 64 position values of leading and trailing edges can be captured. Auxiliary event capture is similar to spark event capture (refer to [Spark Event Capture Unit](#) on page 67).

Note

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

Related topics**Basics**

[Event Capture Windows..... 72](#)

References

[Injection Pulse Position and Fuel Amount Measurement..... 95](#)
[Spark Event Capture..... 93](#)

Complex Comparators

Introduction

The complex comparators can be used to capture complex signals.

Supported units

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

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

Characteristics

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

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

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

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

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

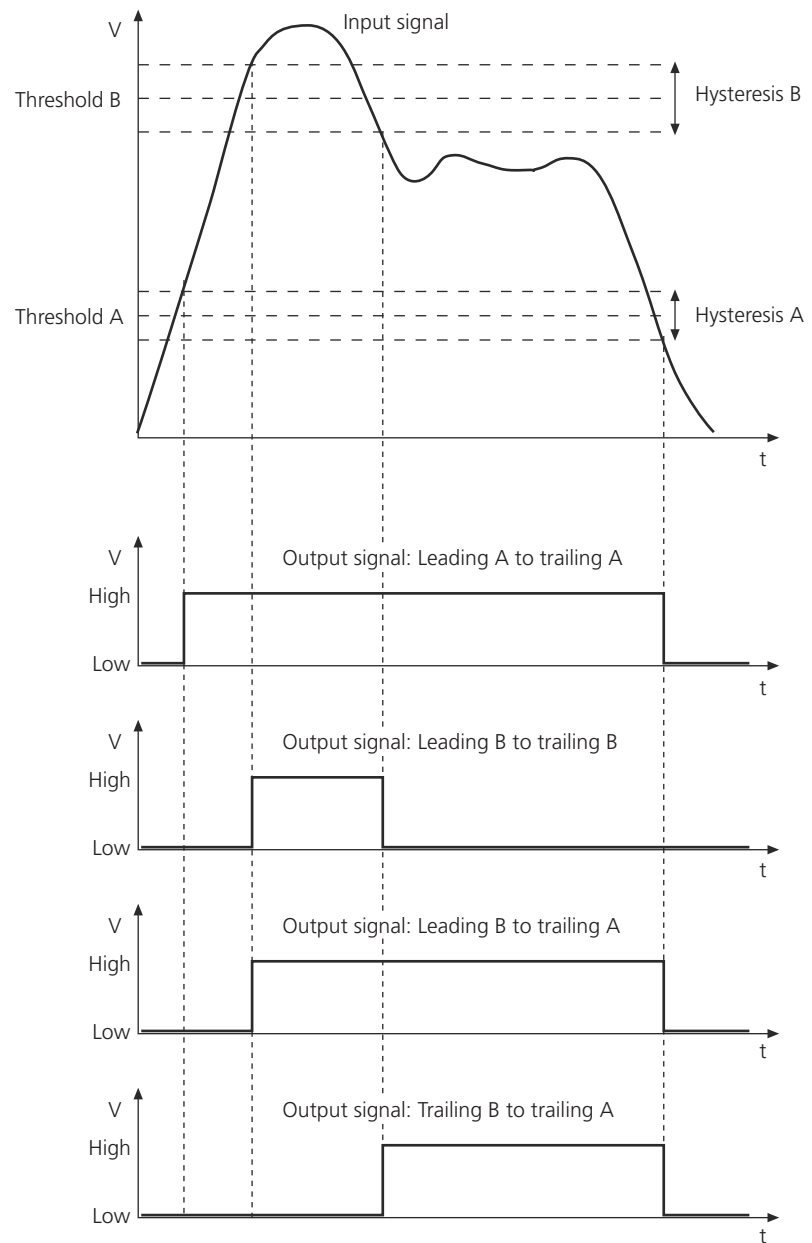
$$TH_B - 0.5 \cdot Hyst_B > TH_A + 0.5 \cdot Hyst_A$$

Signal capture modes

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

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

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

**Related RTI block**

DS2211DIO_SETUP_Bx

Related RTLib functionsds2211_digin_threshold_set, ds2211_apu_ignition_cc_setup,
ds2211_apu_injection_cc_setup

Related topics

References

[ds2211_apu_ignition_cc_setup \(DS2211 RTLib Reference !\[\]\(3d8c13c92b853674f749aac6fa869926_img.jpg\)\)](#)
[ds2211_apu_injection_cc_setup \(DS2211 RTLib Reference !\[\]\(ce455c990c00145a2dda1d9a310cb682_img.jpg\)\)](#)
[ds2211_digin_threshold_set \(DS2211 RTLib Reference !\[\]\(de9e6664b8ceb5519927d73e240a55d9_img.jpg\)\)](#)
[DS2211DIO_SETUP_Bx \(DS2211 RTI Reference !\[\]\(f7025958c2763d977981ad2aefd8cb1b_img.jpg\)\)](#)

Angular Processing Unit - Variant

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

Where to go from here

Information in this section

Basics on Simulating Engine Variants.....	81
A dSPACE simulator or a processor board with a DS2211 can be used to simulate different engine variants.	
Building a Simulink Model for Engine Variants.....	83
You should use the blocks of the VAR APU blockset to build a Simulink model for engine variants.	
Multiple Event Capture Mode of the VAR APU Blockset.....	84
In contrast to the blocks of the APU blockset, the capture blocks of the VAR APU blockset behave differently.	
Example of Capturing a Multiple Event with the VAR APU Blockset.....	85
The example demonstrates how the output values of a capture block change when a signal is captured in the multiple capture mode.	

Basics on Simulating Engine Variants

Introduction	A dSPACE simulator or a processor board with a DS2211 can be used to simulate different engine variants.
APU blockset	If you work with the APU (Angular Processing Unit) blockset, you can build the model for one engine variant only. You have to rebuild and download the real-time application if you want to simulate another engine variant. This requires a lot of time, especially when working with large engine models.
VAR APU blockset	The Angular Processing Unit - Variant (VAR APU) blockset is designed for simulating engine variants. All relevant parameters can be modified during run time via the experiment software. The changed values take effect only when the simulation state of the model changes from STOP to RUN or PAUSE.

Capture modes

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

Modifiable parameters

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

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

Limitations

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

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

Related RTI blocks

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

Related RTLib function

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

Related topics**Basics**

[Simulating Engine Variants \(DS2211 Features !\[\]\(f1c5da15572e3e09d343161be98f508d_img.jpg\)\)](#)

Building a Simulink Model for Engine Variants

Introduction

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

Note


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

Differences of VAR APU and APU blocks

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

RTI Block ¹⁾	Differences
DS2211VARAPU_CRANK_Bx	This block specifies the engine parameters: <ul style="list-style-type: none"> ▪ Number of cylinders ▪ Cylinder sequence and first TDC (top dead center) ▪ Firing sequence These parameters can be modified via the experiment software.
DS2211VARAPU_CAM_Bx_Cy	No difference (shared with the APU blockset)
DS2211APU_ANG_Bx	No difference (shared with the APU blockset)
DS2211APU_ANG_REL_Bx	The block's output depends on the engine parameters specified by DS2211VARAPU_CRANK_Bx. No difference in the functionality.
DS2211VARAPU_IGN_Bx	The block's parameters depend on the engine parameters specified by DS2211VARAPU_CRANK_Bx. The start and end positions of the first and second event windows are relative to the position of the TDC. In addition you can modify the trigger mode. Event capturing uses an extended method. This method supplies the captured events on based on event windows and count the events continuously. Refer to Multiple Event Capture Mode of the VAR APU Blockset on page 84.
DS2211VARAPU_INJ_Bx_Gy	
DS2211VARAPU_AUXCAP_Bx_Cy	
DS2211APU_ABS_CNT_RESET_Bx	No difference (shared with the APU blockset)
DS2211APU_INT_Bx_Iy	No difference (shared with the APU blockset)
DS2211VARSL_KNSG_Bx_Cy	The block's parameters depend on the engine parameters specified by DS2211VARAPU_CRANK_Bx. All engine-related parameters are tunable and can be changed at run time.

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

Implementing the model	Implement the engine model using the VAR APU blockset. The real-time model must be implemented for the highest number of cylinders. As the width of the in- and outports are fixed and cannot be changed during run time, you can simulate only engine variants with equal or less number of cylinders. In addition, you can only simulate with two event capture windows when it was specified in the real-time model.
Build and download	Build and download the real-time application as usual. This must be done only once for the real-time application.
Varying the engine parameters	The engine parameters are available in the SDF file, so they can be changed in the real-time application via the experiment software. The changes on the parameter values get valid during the transition of the simulation state from STOP to PAUSE or RUN. If new parameter values are invalid, appropriate error messages are displayed and the last values of these parameters remain valid. Nevertheless, the simulation starts. It cannot be stopped due to technical reason. The simulation may run with a mix of old and new parameter values.
Related topics	<div>Basics</div> <div> Simulating Engine Variants (DS2211 Features ) </div>

Multiple Event Capture Mode of the VAR APU Blockset

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

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

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

Related topics

Examples

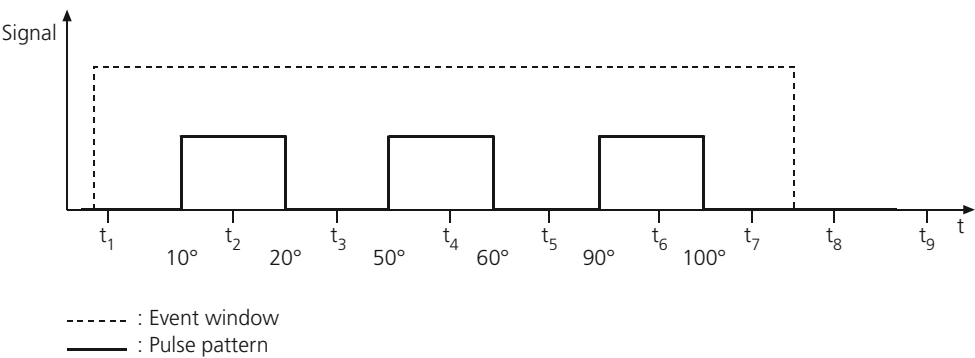
Example of Capturing a Multiple Event with the VAR APU Blockset.....	85
--	----

Example of Capturing a Multiple Event with the VAR APU Blockset

Introduction

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

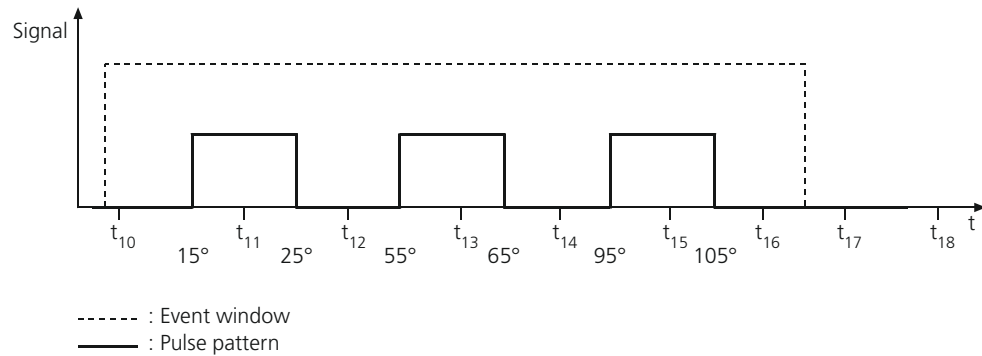
The following illustration shows the captured signal. Three pulses are expected within one defined capture event window. The initial value of the start and end position is set to 999.



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

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

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



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

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

Related topics

Basics

[Multiple Event Capture Mode of the VAR APU Blockset..... 84](#)

APU Reference

Introduction

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

Where to go from here

Information in this section

[Crankshaft Sensor Signal Generation..... 88](#)

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

[Camshaft Sensor Signal Generation..... 90](#)

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

[Wave Table Generation..... 92](#)

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

[Spark Event Capture..... 93](#)

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

[Injection Pulse Position and Fuel Amount Measurement..... 95](#)

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

Information in other sections

[APU Basics..... 56](#)

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

[Knock Sensor Simulation..... 101](#)

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

Crankshaft Sensor Signal Generation

Characteristics

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

of the analog wave form. For basic information on how crankshaft sensor signal generation works, refer to [Crankshaft Signal Generator](#) on page 61.

Using wave tables, you can define specific crankshaft wave forms to simulate different crankshaft types. The DS2211 can load 8 different wave tables for crankshaft sensor signal generation. The wave form to be generated can be switched during run time by using RTI or RTLib functions. For detailed information on wave tables, refer to [Wave Table Generation](#) on page 92.

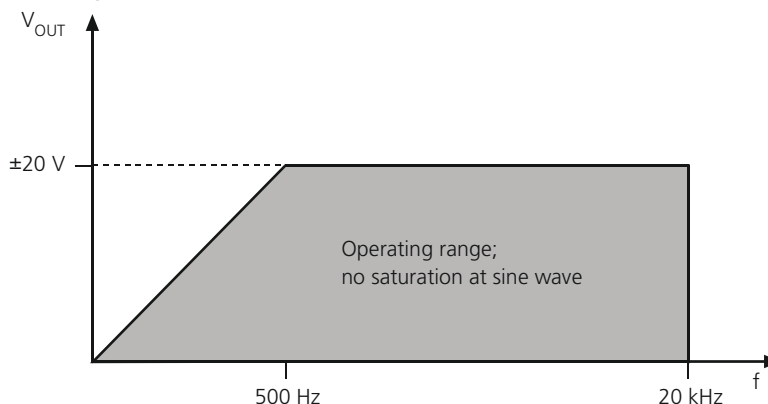
Digital output The digital output has a push-pull driver running from an external source (VBAT1 within the range 5 ... 30 V, VBAT2 within the range 5 ... 60 V). It is protected against overvoltage up to 60 V. The maximum output current is ± 50 mA. Short circuit protection to GND and VBAT is implemented. The digital outputs are in their high impedance state after reset or after power-up. When RTI is used, the output is enabled if the block is part of the model. You can enable or disable the output using RTLib functions.

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

Note

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

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



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

Execution times

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

I/O mapping

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

Channel	Signal	ECU Connector	Pin	Description	Voltage Range/Output Current
Digital	CRANK_DIG	2	C13	Digital crankshaft output	0.4 V ... (VBAT(x) – 1.2 V); ± 50 mA
Analog	CRANK+	2	D13	Crankshaft wave form output	With transformer (AC output mode): (CRANK+ – CRANK–) = ± 20 V Bypassed transformer (DC output mode): (CRANK+ – GND) = ± 10 V; ± 5 mA
	CRANK–	2	E13		

Related topics**References**

[Crankshaft Sensor Signal Generation \(DS2211 RTLib Reference !\[\]\(3211b5d1d968fc1665909b34f9f16010_img.jpg\)](#))

[Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(d47ad152ec3d86a04ad64c8049e1f17f_img.jpg\)](#))

[DS2211APU_CRANK_Bx \(DS2211 RTI Reference !\[\]\(6b7fbb0b7bdb78cadf73d50851a4dfb1_img.jpg\)](#))

[DS2211DIO_SETUP_Bx \(DS2211 RTI Reference !\[\]\(cd0f39e2b8d76d7e84d5eed1ed02b2df_img.jpg\)](#))

[Power Supply Outputs \(PHS Bus System Hardware Reference !\[\]\(11e014e490252374320bfedb9a33c896_img.jpg\)](#))

[Transformer Outputs \(APU and Slave DSP\) \(PHS Bus System Hardware Reference !\[\]\(4971f4a68f0e91a16206aa5c8215b1e8_img.jpg\)](#))

Camshaft Sensor Signal Generation

Characteristics

The camshaft signal generator provides four camshaft sensor signals. Two signals are digital and analog, two signals are digital only. The digital signals are pulse patterns that represent the sign of the corresponding analog wave forms. For basic information on how camshaft sensor signal generation works, refer to [Camshaft Signal Generator](#) on page 65.

Using wave tables, you can define specific camshaft wave forms to simulate different camshaft types. The DS2211 is capable of loading 8 different wave tables for each camshaft output. The wave form to be generated can be switched during run time. For detailed information on wave tables, refer to [Wave Table Generation](#) on page 92.

Digital outputs The digital outputs have push-pull drivers running from an external source (VBAT1 within the range 5 ... 30 V, VBAT2 within the range 5 ... 60 V). The maximum output current per channel is ± 50 mA. Short circuit protection to GND and VBAT is implemented. The outputs are in their high

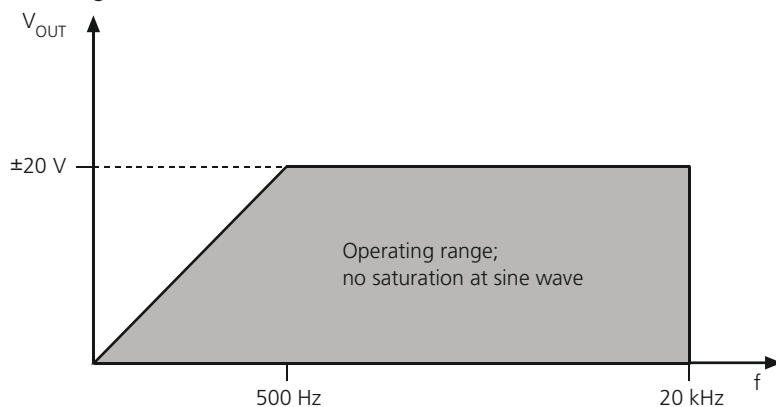
impedance state after reset or after power-up. You can enable or disable the outputs using RTI/RTLib functions.

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

Note

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

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



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

Execution times

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

I/O mapping

The following table shows the mapping of the camshaft outputs (analog and digital) to the related I/O pins of the ECU connector. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 221.

Channel		ECU Connector	Pin	Signal	Description	Voltage Range/Output Current
1 (A)	Digital	2	C14	CAM1_DIG	Digital camshaft output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
	Analog	2	A14	CAM1+	Camshaft wave form output	With transformer (AC output mode): (CAM1+ – CAM1–) = ±20 V Bypassed transformer (DC output mode): (CAM1+ – GND) = ±10 V; ±5 mA
		2	B14	CAM1–		
2 (B)	Digital	2	A15	CAM2_DIG	Digital camshaft output	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
	Analog	2	D14	CAM2+	Camshaft wave form output	With transformer (AC output mode): (CAM2+ – CAM2–) = ±20 V Bypassed transformer (DC output mode): (CAM2+ – GND) = ±10 V; ±5 mA
		2	E14	CAM2–		
3 (C)	Digital	2	E11	CAM3_DIG (DIG_OUT15)	Digital camshaft output (shared with digital output)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA
4 (D)	Digital	2	A12	CAM4_DIG (DIG_OUT16)	Digital camshaft output (shared with digital output)	0.4 V ... (VBAT(x) – 1.2 V); ±50 mA

Related topics

References

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

Wave Table Generation

Wave table

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

Wave table basics

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

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

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

Generating wave tables

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

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

Generating wave tables using MATLAB

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

Generating wave tables using RTI

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

Related topics

References

Camshaft Sensor Signal Generation.....	90
Crankshaft Sensor Signal Generation.....	88

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 67 and [Event Capture Windows](#) on page 72.

You can choose between active high and active low pulses.

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

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

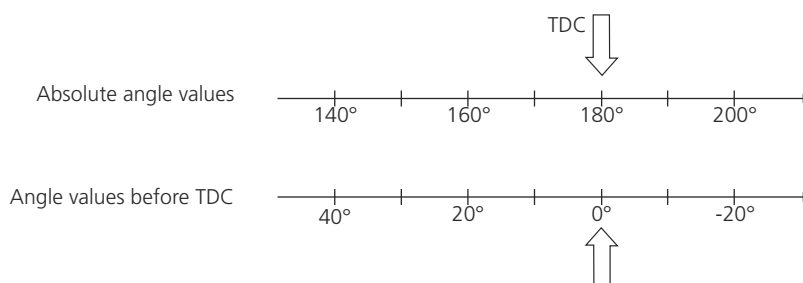
The inputs are 12/42 V compatible. They are protected against overvoltage up to 60 V. The voltage level when a pulse is captured can be defined with a complex comparator (refer to [Complex Comparators](#) on page 77). After software initialization, the input threshold is set to 2.5 V.

Note

There are different reference points for measuring engine positions:

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

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



Execution times

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

I/O mapping

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

features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 221.

RTI Cylinder/ Channel	RTLib Channel	Signal	ECU Connector	Pin	Description	Voltage Range
1	1	IGN1	1	D13	Ignition capture	12/42 V compatible
2	2	IGN2	1	D14	Ignition capture	12/42 V compatible
3	3	IGN3	1	D15	Ignition capture	12/42 V compatible
4	4	IGN4	1	D16	Ignition capture	12/42 V compatible
5	5	IGN5	1	D17	Ignition capture	12/42 V compatible
6	6	IGN6	1	D18	Ignition capture	12/42 V compatible
1	7	AUXCAP1	1	E1	Auxiliary capture	12/42 V compatible
2	8	AUXCAP2	1	E2	Auxiliary capture	12/42 V compatible

Related topics

Basics

Event Capture Windows.....	72
Spark Event Capture Unit.....	67

References

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

Injection Pulse Position and Fuel Amount Measurement

Characteristics

The injection event capture unit provides 16 digital injection inputs split into 2 groups for injection pulse position and fuel amount measurement. For detailed information on event capture windows and capture modes, refer to [Event Capture Windows](#) on page 72 and [Injection Event Capture Unit](#) on page 68.

You can choose between active high and active low pulses.

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

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

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

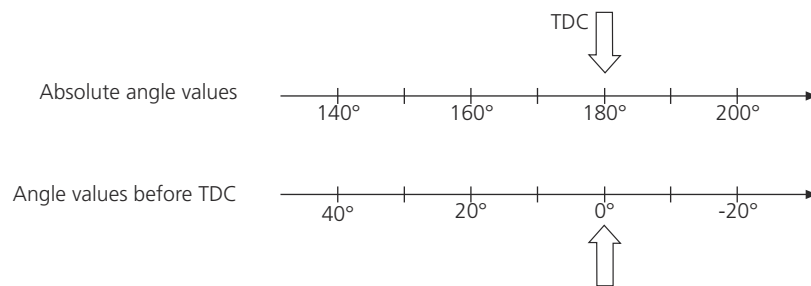
to [Complex Comparators](#) on page 77). After software initialization, the input threshold is set to 2.5 V.

Note

There are different reference points for measuring engine positions:

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

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



I/O mapping

The following table shows the mapping of the cylinder/channel numbers to the related I/O pins of the ECU connector, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 221.

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

RTI Cylinder/ Channel	RTLib Channel	ECU Connector	Pin	Signal	Description	Voltage Range
1	1	1	D7	INJ1	Injection capture	12/42 V compatible
2	2	1	D8	INJ2	Injection capture	12/42 V compatible
3	3	1	D9	INJ3	Injection capture	12/42 V compatible
4	4	1	D10	INJ4	Injection capture	12/42 V compatible
5	5	1	D11	INJ5	Injection capture	12/42 V compatible
6	6	1	D12	INJ6	Injection capture	12/42 V compatible
7	7	1	D5	INJ7 (PWM_IN7)	Injection capture (shared with PWM input)	12/42 V compatible
8	8	1	D6	INJ8 (PWM_IN8)	Injection capture (shared with PWM input)	12/42 V compatible

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

RTI Cylinder/ Channel	RTLib Channel	ECU Connector	Pin	Signal	Description	Voltage Range
1	1	1	D13	IGN1	Injection capture	12/42 V compatible
2	2	1	D14	IGN2	Injection capture	12/42 V compatible
3	3	1	D15	IGN3	Injection capture	12/42 V compatible
4	4	1	D16	IGN4	Injection capture	12/42 V compatible
5	5	1	D17	IGN5	Injection capture	12/42 V compatible
6	6	1	D18	IGN6	Injection capture	12/42 V compatible
7	7	1	E1	AUXCAP1	Auxiliary capture	12/42 V compatible
8	8	1	E2	AUXCAP2	Auxiliary capture	12/42 V compatible

Related topics

Basics

Event Capture Windows.....	72
Spark Event Capture Unit.....	67

References

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

Features of the Slave DSP

Introduction	The slave DSP is used for wheel speed sensor simulation and knock sensor simulation. It provides a DAC unit, has access to the Bit I/O unit and can be triggered by interrupts.
Where to go from here	<div>Information in this section<div><div>Basics of the Slave DSP..... 100</div><div>The DS2211 is equipped with a TMS320VC33 DSP from Texas Instruments as the slave DSP.</div><div>Knock Sensor Simulation..... 101</div><div>Knock sensor simulation is performed by a ready-to-use application implemented on the slave DSP.</div><div>Wheel Speed Sensor Simulation..... 103</div><div>Wheel speed sensor simulation is performed by a ready-to-use application implemented on the slave DSP.</div><div>DSP DAC Unit..... 105</div><div>The slave DSP contains eight digital analog converters. Four digital analog converters are used for knock sensor simulation or wheel speed sensor simulation. The other four channels can be used for user output.</div><div>DSP Bit I/O Unit and Capture Input Access..... 106</div><div>The slave DSP has access to the 16 digital inputs and the 16 digital outputs of the bit I/O unit.</div><div>Slave DSP Interrupts..... 107</div><div>The slave DSP receives APU update interrupts, DPMEM interrupts, and edge detection interrupts.</div></div></div>

Basics of the Slave DSP

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

Related topics

References

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

Knock Sensor Simulation

Introduction

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

Basics

Knock sensor simulation is triggered by the engine position, which is supplied by the engine position bus (see [APU Overview](#) on page 57). Like a knock sensor in a real engine that measures several cylinders, multiple knock signals can be generated for up to 8 cylinders in one engine cycle (0 ... 720°). Knock sensor signals are generated as cosine wave signals plus Gaussian noise. All parameters to determine the characteristics of the generated signal can be changed at run time by using RTI or RTLib functions.

Knock signal calculation

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

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

Where

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

Knock signal parameters

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

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

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

Amplitude Amplitude of the cosine signal in V_{pp}

Frequency Frequency of the cosine signal in Hz

Damping Damping factor of the cosine signal

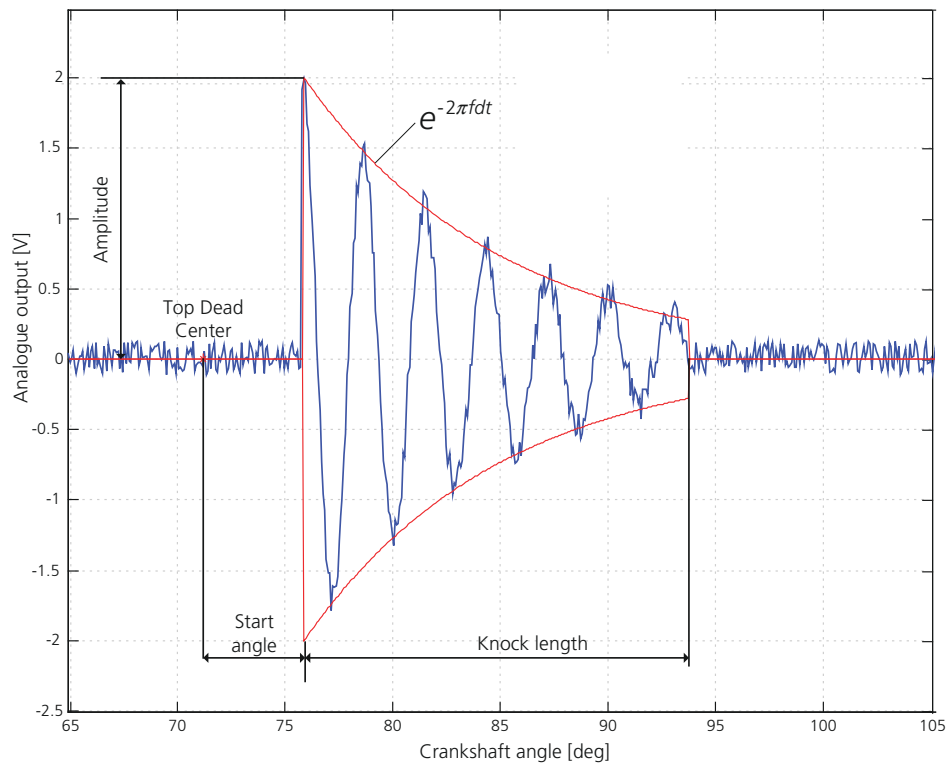
Knock number This parameter defines the maximum number of knock signals which are generated for each cylinder while the application is running. When the limit is reached, no further pulses are generated. To specify an infinity number of knock signals, set Knock number = 0 and Knock rate = 1.

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

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

Noise Amplitude of the noise in V_{pp}

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



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

Transformers do not transmit a DC voltage. If the signal has a DC voltage component, you can bypass the transformer via jumper settings. For details, refer to [Jumper Settings J1 ... J7 \(PHS Bus System Hardware Reference !\[\]\(ab4e2b3fc7e7887b7a72f548aa6f5e60_img.jpg\)](#)).

Note

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

Execution times

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

I/O mapping

The following table shows the mapping of the channel numbers to the related I/O pins of the ECU connector, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 221.

Channel Number	Signal	ECU Connector	Pin	Description	Voltage Range/Output Current
1	VC33-Waveform 0+	2	B15	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 0+ – VC33-Waveform 0–) = ±20 V
	VC33-Waveform 0–	2	C15		
2	VC33-Waveform 1+	2	B16	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 1+ – VC33-Waveform 1–) = ±20 V
	VC33-Waveform 1–	2	C16		
3	VC33-Waveform 2+	2	D16	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 2+ – VC33-Waveform 2–) = ±20 V
	VC33-Waveform 2–	2	E16		
4	VC33-Waveform 3+	2	A17	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 3+ – VC33-Waveform 3–) = ±20 V
	VC33-Waveform 3–	2	B17		

Related topics**References**

[DS2211SL_KNSG_Bx_Cy \(DS2211 RTI Reference\)](#)

[DS2211VARSL_KNSG_Bx_Cy \(DS2211 RTI Reference\)](#)

[Knock Sensor Simulation \(DS2211 RTLib Reference\)](#)

[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.

Wheel speed signal calculation

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

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

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

Where

a is the amplitude

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

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

speed: Wheel speed

teeth: Number of wheel teeth

t is the time

Outputs

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

Transformers do not transmit a DC voltage. If the signal has a DC voltage component, you can bypass the transformer via jumper settings, refer to [Jumper Settings J1 ... J7 \(PHS Bus System Hardware Reference !\[\]\(e3f255517d37bb309a3a931ec4849e6a_img.jpg\)](#)).

The outputs can be enabled or disabled.

Note

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

Execution times

The execution times required by the RTLib functions have been measured. For details on the results and the corresponding measurement setup, refer to [Function Execution Times \(DS2211 RTLib Reference !\[\]\(9a795c4c0c43d0827b424565265fc8e6_img.jpg\)](#)).

I/O mapping

The following table shows the mapping of the channel numbers to the related I/O pins of the ECU connector, as used in RTI and RTLib. Some I/O features of the DS2211 conflict with each other. For details, see [Conflicting I/O Features](#) on page 221.

Channel Number	Signal	ECU Connector	Pin	Description	Voltage Range/Output Current
1	VC33-Waveform 0+ VC33-Waveform 0–	2 2	B15 C15	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 0+ – VC33-Waveform 0–) = ± 20 V
2	VC33-Waveform 1+ VC33-Waveform 1–	2 2	B16 C16	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 1+ – VC33-Waveform 1–) = ± 20 V
3	VC33-Waveform 2+ VC33-Waveform 2–	2 2	D16 E16	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 2+ – VC33-Waveform 2–) = ± 20 V
4	VC33-Waveform 3+ VC33-Waveform 3–	2 2	A17 B17	Knock sensor / Wheel speed sensor signal	(VC33-Waveform 3+ – VC33-Waveform 3–) = ± 20 V

Related topics

References

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

DSP DAC Unit

Characteristics

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

Execution times

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

I/O mapping

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

D/A Channel	Signal	ECU Connector	Pin	Description	Voltage Range/Output Current
5	DSP_DAC4	3	B15	12-bit DAC/10 μ s	(DSP_DAC5 – GND) = ± 10 V; ± 5 mA
6	DSP_DAC5	3	C15	12-bit DAC/10 μ s	(DSP_DAC6 – GND) = ± 10 V; ± 5 mA
7	DSP_DAC6	3	B16	12-bit DAC/10 μ s	(DSP_DAC7 – GND) = ± 10 V; ± 5 mA
8	DSP_DAC7	3	C16	12-bit DAC/10 μ s	(DSP_DAC8 – GND) = ± 10 V; ± 5 mA

Related topics**Basics**

Knock Sensor Simulation.....	101
Wheel Speed Sensor Simulation.....	103

DSP Bit I/O Unit and Capture Input Access

Characteristics

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

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

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

For a description of the characteristics, refer to [Bit I/O Unit](#) on page 33.

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

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



Execution times

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

Related topics**Basics**

Bit I/O Unit.....	33
-------------------	----

References

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

Slave DSP Interrupts

Introduction

The slave DSP receives 4 different interrupts.

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

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

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

Interrupt sources

The following interrupt sources are available:

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

CAN Support

Introduction

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

Where to go from here

Information in this section

- [Setting Up a CAN Controller.....](#) 110
Explains how to set up a CAN controller to use a dSPACE board with CAN bus interface.
- [Using the RTI CAN MultiMessage Blockset.....](#) 124
Provides information on using the RTI CAN MultiMessage Blockset.
- [CAN Signal Mapping.....](#) 147
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.....](#) 235
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..... 110](#)

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

[CAN Transceiver Types..... 112](#)

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

[DS2211: Selecting the CAN Controller Frequency..... 116](#)

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

[Defining CAN Messages..... 117](#)

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

[Implementing a CAN Interrupt..... 118](#)

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..... 119](#)

RTI CAN Blockset provides two concepts for receiving CAN messages.

[Removing a CAN Controller \(Go Bus Off\)..... 121](#)

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

[Getting CAN Status Information..... 121](#)

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

Initializing the CAN Controller

Introduction

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

Standard configuration

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

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

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

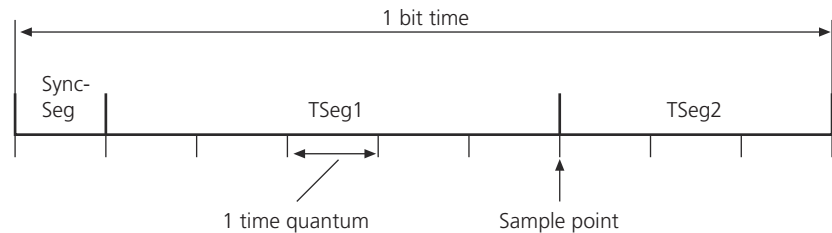
Advanced configuration (bit timing parameters)

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

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

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

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



The following parameters are also part of the advanced configuration:

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

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

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

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

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

Note

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

RTI support

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

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

Related topics**References**

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

CAN Transceiver Types

Introduction

To communicate with other bus members in a CAN bus, each bus member is equipped with a CAN transceiver. The transceiver defines the type of wire used for the bus (coaxial, two-wire line, or fiber-optic cables), the voltage level, and the pulse forms used for 0-bit and 1-bit values. The way in which CAN messages are transmitted on a CAN bus therefore significantly depends on the CAN transceiver used.

Note

Make sure that the CAN transceiver type used on the CAN bus matches the type on the dSPACE board you use to connect to the bus.

Terminating the CAN bus

Depending on the CAN transceiver type, you must terminate each CAN bus with resistors at both ends of the bus.

Note

Failure to terminate the bus will cause bit errors due to reflections. These reflections can be detected with an oscilloscope.

Supported transceivers

The following table lists dSPACE hardware and the supported transceivers:

dSPACE Hardware	Transceiver Type
<ul style="list-style-type: none"> DS2202 DS2210 DS2211 	ISO11898
DS4302	<p>The following transceiver types are supported:</p> <ul style="list-style-type: none"> ISO11898 RS485 C252 Piggyback¹⁾ <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>The RTI CAN Blockset does not support transceiver types with different modes, for example single-wire and two-wire operation. Nevertheless, such transceiver types can be applied to the DS4302, but additional user-written S-functions are required to implement the communication between the piggyback module and the CAN controller.</p> </div>
MicroAutoBox II	<p>The following transceiver types are supported:</p> <ul style="list-style-type: none"> ISO11898 ISO11898-6^{2), 3)}
MicroLabBox	<p>The following transceiver types are supported:</p> <ul style="list-style-type: none"> ISO11898 ISO11898-6²⁾

¹⁾ If none of the above transceivers matches your application or if a TJA1041 transceiver is used, "piggyback" must be selected as the transceiver type.

²⁾ Selecting the ISO11898-6 transceiver type is required to perform partial networking.

³⁾ Supported only by MicroAutoBox II with DS1513 I/O board.

ISO11898 transceiver

ISO11898 defines a high-speed CAN bus that supports baud rates of up to 1 MBd. This is the most commonly used transceiver, especially for the engine management electronics in automobiles.

CAN-H, CAN-L ISO11898 defines two voltage levels:

Level	Description
CAN-H	High if the bit is dominant (3.5 V), floating (2.5 V) if the bit is recessive.
CAN-L	Low if the bit is dominant (1.0 V), floating (2.5 V) if the bit is recessive.

Termination To terminate the CAN bus lines, ISO11898 requires a 120-Ω resistor at both ends of the bus.

ISO11898-6 transceiver

High-speed transceiver that supports partial networking.

Termination To terminate the CAN bus lines, ISO11898-6 requires a 120-Ω resistor at both ends of the bus.

Note

There are some limitations when you use the optional ISO11898-6 transceiver:

- No wake-up interrupt is implemented.
- Partial networking is supported only for the following baud rates:
 - 125 kbit/s
 - 250 kbit/s
 - 500 kbit/s
 - 1000 kbit/s

Other baud rates can be used for normal CAN operation, but detecting wake-up messages for partial networking is supported only for the baud rates listed above.

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

RS485 transceiver

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

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


C252 fault-tolerant transceiver

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

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

Termination There are two ways to terminate the CAN bus lines: Use a 10 k Ω resistor for many connected bus members, or a 1.6 k Ω resistor if the number of bus members is equal to or less than five. The termination resistors are located between CAN-L and RTL and CAN-H and RTH (refer also to the "PCA82C252 Fault-tolerant Transceiver Data Sheet" issued by Philips Semiconductors).

Note

The TJA1054 transceiver is pin and downward compatible with the C252 transceiver. If the TJA1054 transceiver is on board the DS4302 and you want to use the fault-tolerant transceiver functionality, select "C252" in the RTI CAN CONTROLLER SETUP block. Refer to [Unit Page \(RTICAN CONTROLLER SETUP\)](#) (RTI CAN Blockset Reference .

Custom transceivers

The DS4302 allows you to mount up to four customization modules to use transceivers that are not on the DS4302.

Connecting customization modules For instructions on connecting customization modules, refer to [Customization Modules \(PHS Bus System Hardware Reference !\[\]\(e474458956c9a37fbf9586ddb60a7fa1_img.jpg\)](#)).

Optional TJA1041 transceiver dSPACE provides the optional TJA1041 that you can use as a custom transceiver for the DS4302. For a detailed description of the transceiver and the available transceiver modes, refer to the data sheet of the TJA1041 transceiver.

For details on the RTI support for the TJA1041 transceiver, refer to [TJA1041 Support Blocks \(RTI CAN Blockset Reference !\[\]\(21199eb166cc97331a0c54c649195dcc_img.jpg\)\)](#).

Note

There are some limitations when you use the optional TJA1041 transceiver:

- No wake-up interrupt is implemented.
- You have to enable Automatic Wake Up in the [DS4302_TJA1041_SST \(RTI CAN Blockset Reference !\[\]\(cdf2842d82858164c68c92720a337fb9_img.jpg\)\)](#) block before you build the model. You cannot enable automatic wake-up during run time.
- If the transceiver is in *power on / listen only* mode, the CAN controller does not send an acknowledge message to the transmitter. The transmitter therefore continues to send the message until it receives the acknowledge signal. This might cause a task overrun if an RX interrupt is configured for the CAN controller.
- If the transceiver is in *power on / listen only* mode, it is not able to send CAN messages. Automatic wake-up is not possible if the transceiver is in *power on / listen only* mode. Because no message is sent on the CAN bus by the transceiver in *power on / listen only* mode, CAN arbitration fails. The CAN controller changes to the BUS OFF state. It is not possible to set the BUS state automatically to BUS ON via an interrupt, because the reason for the BUS OFF state still remains. You must set the CAN controller to BUS ON after you have switched the transceiver state to normal, standby, or sleep mode.

DS2211: Selecting the CAN Controller Frequency

Introduction

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

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

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

Highest possible frequency automatically selected

The real-time application built with RTI CAN Blockset is compatible with both board versions and frequencies: During board initialization, the highest frequency that is available for the controller is automatically selected, together with the corresponding bit timing values. This applies regardless of the frequency you select in the Advanced Configuration dialog. Refer to [Advanced Configuration Dialog \(RTICAN CONTROLLER SETUP\) \(RTI CAN Blockset Reference !\[\]\(758ebdf4629c903da74c2e079717ae32_img.jpg\)\)](#).

Related topics**References**

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

Defining CAN Messages

Introduction

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

Message types

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

Message Type	Description
Transmit (TX)	This message is transmitted with a specific identifier. A TX message contains up to 8 bytes of data.
Receive (RX)	This message is <i>not</i> transmitted over the bus. An RX message is used only to define how the CAN controller processes a received message. An RX message transfers the incoming data from the CAN controller to the master processor.
Request (RQ)	First part of a <i>remote transmission request</i> ¹⁾ . An RQ message is transmitted with a specific identifier to request data. An RQ message does not contain data.
Remote (RM)	Second part of a <i>remote transmission request</i> ¹⁾ . An RM message is a TX message that is sent only if the CAN controller has received a corresponding RQ message. The RM message contains the data requested by the RQ message.

¹⁾ With RTI CAN Blockset, the remote transmission request is divided into an RQ message and an RM message. The meanings of the words “remote” and “request” used in this document do not correspond to those used in CAN specifications.

Message configuration

With RTI CAN Blockset, you have to implement one message block for each message. To define a message to be transmitted, for example, you must implement an RTICAN Transmit (TX) block.

Message configuration by hand You can perform message configuration by hand. In this case, you must specify the message identifier and identifier format (STD, XTD), the length of the data field, and the signals for each message. You also have to specify the start bit and length of each signal.

Message configuration from data file (data file support) You can load a data file containing the configuration of one or more messages. Then you can assign a message defined in the data file to a message block. Refer to [Configuring CAN Messages via Data Files \(RTI CAN Blockset Reference !\[\]\(799877f5c2f906134441300079881630_img.jpg\)\)](#).

Multiple message access

Multiple message access allows you to place several RX or TX blocks with the same identifier and identifier format in one model. You can decode the signals of





an RX message in several ways, or place TX blocks in several enabled subsystems to send data in various ways.

Delay time for message transmission

To distribute messages over time and avoid message bursts, you can specify delay times. A message is sent after the delay time. The delay time is a multiple of the time needed to send a CAN message at a given baud rate and identifier format. You can only enter a factor to increase or decrease the delay time.

RTI support

With RTI CAN Blockset, you have to implement one message block for each message. Refer to:

Message Type	RTI Block
Transmit (TX)	RTICAN Transmit (TX) (RTI CAN Blockset Reference )
Receive (RX)	RTICAN Receive (RX) (RTI CAN Blockset Reference )
Request (RQ)	RTICAN Request (RQ) (RTI CAN Blockset Reference )
Remote (RM)	RTICAN Remote (RM) (RTI CAN Blockset Reference )

Related topics**Basics**

[Configuring CAN Messages via Data Files \(RTI CAN Blockset Reference !\[\]\(73002692dd5e7a64e60946be3158e719_img.jpg\)](#))

Implementing a CAN Interrupt

Introduction

The CAN controller transmits and receives messages and handles error management. It is also responsible for generating interrupts to the master processor. You can specify the events on which these interrupts are generated.

A special Bus Failure interrupt and a wake-up interrupt are available for the DS4302.

RTI support

You can implement a CAN interrupt with the RTICAN Interrupt block. Refer to [RTICAN Interrupt \(RTI CAN Blockset Reference !\[\]\(b538fe54c1f3a7343e37e85cc2d00497_img.jpg\)](#)).

Related topics**References**

[RTICAN Interrupt \(RTI CAN Blockset Reference !\[\]\(097cdd6c9c875b64d9b8c9a2409491c4_img.jpg\)](#))

Using RX Service Support

Concepts for receiving CAN messages

When CAN messages are received, RX blocks access the DPMEM between the master processor and the slave processor.

RTI CAN Blockset provides two concepts for receiving CAN messages:

- Common receive concept
- RX service receive concept

Common receive concept

According to the common receive concept, one data object is created in the DPMEM for each received CAN message. Due to the limited DPMEM size, the number of RX blocks you can use in a model is limited to 100 (200 for the DS4302).


RX service receive concept

When you enable RX service support, one data object is created in the DPMEM for all received CAN messages, and memory on the master processor is used to receive CAN messages. The RX service fills this memory with new CAN data. This concept improves run-time performance.

Tip

In contrast to the common receive concept, the number of RX blocks for which RX service support is enabled is unlimited.

Specifying a message filter When you use RX service, you have to specify a filter to select the messages to receive via RX service. To define the filter, you have to set up a bitmap that represents the message. Each bit position can be assigned 0 (must be matched), 1 (must be matched), or X (don't care). A message is received via RX service only if it matches the bitmap.

You can define the message filter on the [RX Service Page \(RTICAN CONTROLLER SETUP\)](#) (RTI CAN Blockset Reference .

Specifying the queue size When you use RX service, you have to specify the maximum number of CAN messages that you expect to receive in a sample step. The memory allocated on the master processor used to queue CAN messages is calculated from the specified maximum number of CAN messages.

Note

If more CAN messages than the specified Queue size are received in a sample step, the oldest CAN messages are lost. You should therefore specify the queue size so that no CAN messages are lost.

Example:

A CAN controller is configured to use the baud rate 500 kBd. The slowest RX block assigned to this CAN controller is sampled every 10 ms. At the specified baud rate, a maximum of about 46 CAN messages (STD format) might be received during two consecutive sample steps. To ensure that no CAN message is lost, set the queue size to 46.


Triggering an interrupt when a message is received via RX service You can let an interrupt be triggered when a message is received via RX service.

Note

You cannot let an interrupt be triggered when a message *with a specific ID* is received. An interrupt is triggered each time a message is received via RX service.

You can define the interrupt on the [Unit Page \(RTI CAN Interrupt\)](#) (RTI CAN Blockset Reference .

Precondition for gatewaying messages Enabling the RX service is a precondition for *gatewaying messages* between CAN controllers.

Refer to [Gatewaying Messages Between CAN Buses](#) (RTI CAN Blockset Reference .



Precondition for the TX loop back feature RX service allows you to use the *TX loop back feature*. The feature lets you observe whether message transfer over the bus was successful.

You can enable TX loop back on the [Options Page \(RTICAN Transmit \(TX\)\)](#) (RTI CAN Blockset Reference .

Enabling RX service support

You have to enable RX service support for each CAN controller and for each RX block.

RTI support

- For a CAN controller, you enable the RX service on the RX Service page of the RTICAN CONTROLLER SETUP block. Refer to [RX Service Page \(RTICAN CONTROLLER SETUP\)](#) (RTI CAN Blockset Reference .
- For an RX block, you enable the RX service on the Options page of the RTICAN Receive (RX) block of the RTICAN CONTROLLER. Refer to [Options Page \(RTICAN Receive \(RX\)\)](#) (RTI CAN Blockset Reference .

Related topics**Basics**

[Gatewaying Messages Between CAN Buses \(RTI CAN Blockset Reference !\[\]\(666e09182d4cd268646ea700ea60dcdf_img.jpg\)](#))

Removing a CAN Controller (Go Bus Off)

Introduction

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

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

RTI support

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

Related topics**References**

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

Getting CAN Status Information

Introduction

You can use the Error Management Logic (EML) of a CAN controller to get error and status information on the CAN bus and the controller. Errors occur, for example, if a CAN controller fails to transmit a message successfully.

CAN controller status information

The controller's EML has two counters: the Receive Error counter and the Transmit Error counter. According to their values, the EML can set the CAN controller to one of the following states:

Counter Value	Error State	Description
Each counter value < 128	Error active	The CAN controller is active. Before turning to the error passive state, the controller sets an error warn (EWRN) bit if one of the counter values is ≥ 96 .
At least one counter value ≥ 128	Error passive	The CAN controller is still active. The CAN controller can recover from this state itself.
Transmit Error counter value ≥ 256	Bus off	The CAN controller disconnects itself from the bus. To recover, an external action is required (bus off recovery).

CAN bus status information

You can get the following CAN bus status information:

Number of ...	Description
Stuff bit errors	Each time more than 5 equal bits in a sequence occurred in a part of a received message where this is not allowed, the appropriate counter is incremented.
Form errors	Each time the format of a received message deviates from the fixed format, the appropriate counter is incremented.
Acknowledge errors	Each time a message sent by the CAN controller is not acknowledged, the appropriate counter is incremented.
Bit 0 errors	Each time the CAN controller tries to send a dominant bit level and a recessive bus level is detected instead, the appropriate counter is incremented. During bus off recovery, the counter is incremented each time a sequence of 11 recessive bits is detected. This enables the controller to monitor the bus off recovery sequence, indicating that the bus is not permanently disturbed.
Bit 1 errors	Each time the CAN controller tries to send a recessive bit level and a dominant bus level is detected instead, the appropriate counter is incremented.
Cyclic redundancy check (CRC) errors	Each time the CRC checksum of the received message is incorrect, the appropriate counter is incremented. The EML also checks the CRC checksum of each message (see Message fields (RTI CAN Blockset Reference)).
Lost RX messages	Each time a message cannot be stored in the buffer of the CAN controller, the message is lost and an <i>RX lost error</i> is detected.
Successfully received RX messages	Each time an RX message is received successfully, the appropriate counter is incremented.
Successfully sent TX messages	Each time a TX message is sent successfully, the appropriate counter is incremented.
(DS4302 only) Status of fault tolerant receiver	The error state of the fault tolerant receiver is reported.
(DS4302 only) Fault tolerant transceiver	The value of the output is increased if a CAN bus events occurs.

RTI support

To get status information, use the RTICAN Status block. Refer to [RTICAN Status \(RTI CAN Blockset Reference\)](#).

Related topics

References

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

Using the RTI CAN MultiMessage Blockset

Introduction

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

Where to go from here

Information in this section

[Basics on the RTI CAN MultiMessage Blockset](#)..... 124

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

[Basics on Working with CAN FD](#)..... 129

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](#)..... 134

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](#)..... 140

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

[Manipulating Signals to be Transmitted](#)..... 143

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 129.

Support of AUTOSAR features

The RTI CAN MultiMessage Blockset supports miscellaneous AUTOSAR features, such as secure onboard communication and global time synchronization. For more information, refer to [Aspects of Miscellaneous Supported AUTOSAR Features](#) (RTI CAN MultiMessage Blockset Reference )

Manipulating signals to be transmitted

The RTI CAN MultiMessage Blockset provides several options to manipulate the values of signals before they are transmitted. You can switch between them with the Bus Navigator in ControlDesk via entries in the generated TRC file. In

addition, you can calculate checksum, parity, toggle, counter, and mode counter values.

Updating a model

The RTI CAN MultiMessage Blockset creates an S-function for the specified database file. You can easily update the CAN configuration of a model by replacing the database file and updating the S-function.

Tip

You do not have to recreate the S-function for the RTI CAN MultiMessage Blockset if you switch from one processor board to another, e.g., from a DS1006 to a DS1007, and vice versa.

When you switch to or from a MicroAutoBox II or MicroLabBox, you only have to recreate the ControllerSetup block. You also have to recreate the ControllerSetup block if you change the controller name.

Modifying model parameters during run time



Model parameters such as messages or signal values can be modified during run time either via model input or via the **Bus Navigator** in ControlDesk. For modifying model parameters via ControlDesk, a variable description file (TRC) is automatically generated each time you create an S-function for the RTICANMM MainBlock. The entries of the TRC file let you analyze received signals, change the values of signals to be transmitted, etc. In ControlDesk, you can access the settings specified in the TRC file via the model's system description file (SDF). The SDF file bundles all the TRC files and additional information for the application.

(Relevant only for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board) For information on where to find the signals of the CAN bus in the TRC file, refer to [Available TRC File Variable Entries and Their Locations in the TRC File \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(73002692dd5e7a64e60946be3158e719_img.jpg\)](#)).

User-defined variables

You can include user-defined variables in a TRC file in addition to the parameters of the RTI CAN MultiMessage Blockset.

Working with variants of CAN communication

You can work with different CAN communication variants on one CAN controller, and switch between them during run time. Only one variant for each CAN controller can be active at a time. In the **Bus Navigator**, the active variant is labeled  when an application is running on the real-time hardware. An inactive variant is labeled . If you open layouts of an inactive variant, the headers of all the RX and TX layouts are red.

Gatewaying messages between CAN buses

You can easily exchange messages between different CAN buses. In addition, you can use the gatewaying feature to modify messages in gateway mode during run time.

Online modification of gateway exclude list	You can modify the exclude list of RTICANMM Gateways during run time, i.e., specify messages not to be gatewayed.
Dynamic message triggering	You can modify the cycle times and initiate a spontaneous transmission (interactive or model-based) during run time.
Defining free raw messages	<p>You can define free raw messages as additional messages that are independent of the database file. Once they are defined, you can use them like standard database messages and specify various options, for example:</p> <ul style="list-style-type: none"> ▪ Trigger options ▪ Ports and displays ▪ Message ID and length adjustable during run time <p>The following features are not supported:</p> <ul style="list-style-type: none"> ▪ Checksum generation ▪ Custom signal manipulation
Capturing messages	<p>You can process the raw data of messages whose IDs match a given filter via the capture messages feature. This can be a specific ID, a range of IDs, or even all IDs. Optionally, you can exclude the messages contained in the database file.</p> <p>The captured messages can be made available as outports of the MainBlock or in the TRC file.</p>
CAN partial networking	<p>With CAN partial networking, you can set selected ECUs in a network to sleep mode or shut them down if they do not have to run continuously. Wake-up messages then activate specific ECUs as and when required, and for as long as required.</p> <div data-bbox="592 1341 659 1371" data-label="Section-Header">Note</div> <p>Partial networking is possible for the following dSPACE real-time hardware:</p> <ul style="list-style-type: none"> ▪ MicroAutoBox II equipped with the DS1513 I/O Board ▪ MicroLabBox ▪ dSPACE hardware that supports working with CAN FD messages and that is equipped with DS4342 CAN FD Interface Modules, such as: <ul style="list-style-type: none"> ▪ PHS-bus-based systems (DS1006 or DS1007 modular systems) with DS4505 Interface Board ▪ MicroAutoBox II variants with DS1507 ▪ MicroAutoBox II variants with DS1514 <p>The RTI CAN MultiMessage Blockset lets you specify the CAN partial networking wake-up messages by filtering message IDs and message data.</p>

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

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

For more information, refer to [Partial Networking Page \(RTICANMM ControllerSetup\)](#) (RTI CAN MultiMessage Blockset Reference .

TRC file entries with initial data

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

Visualization with the Bus Navigator

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

RTI CAN Blockset and RTI CAN MultiMessage Blockset

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

Further information on the RTI CAN MultiMessage Blockset

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

- [RTI CAN MultiMessage Blockset Reference](#) 

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

- [RTI CAN MultiMessage Blockset Tutorial](#) 

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

Basics on Working with CAN FD

Introduction

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

Basics on CAN FD

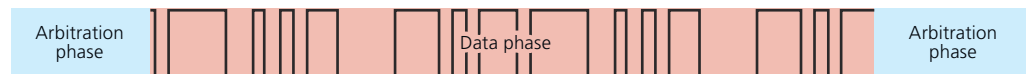
CAN FD stands for *CAN with Flexible Data Rate*. The CAN FD protocol is based on the CAN protocol as specified in ISO 11898-1. Compared with the classic CAN protocol, CAN FD comes with an increased bandwidth for the serial communication. The improvement is based on two factors:

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

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

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

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

Classic CAN message**CAN FD message using a higher bit rate****CAN FD message with longer payload using a higher bit rate****CAN FD protocols**

Currently, there are two CAN FD protocols on the market, which are not compatible with each other.

- The *non-ISO CAN FD protocol* represents the original CAN FD protocol from Bosch.
- The *ISO CAN FD protocol* represents the CAN FD protocol according to the ISO 11898-1:2015 standard.

Compared to the non-ISO CAN FD protocol, the ISO CAN FD protocol comes with an improved failure detection capability.

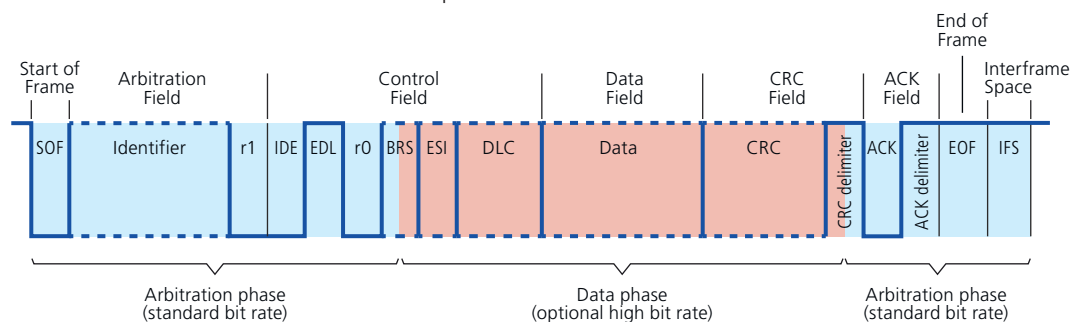
The RTI CAN MultiMessage Blockset supports both CAN FD protocols.

CAN FD message characteristics

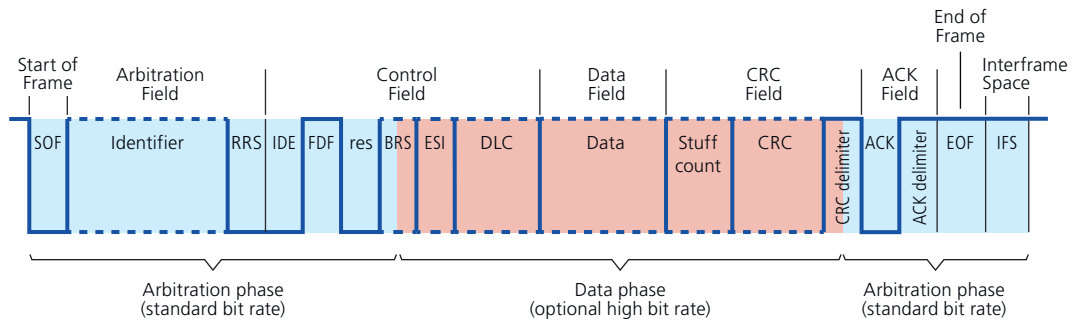
In principle, CAN FD messages and CAN messages consist of the same elements and have the same message structure.

For an overview of the fields of a CAN FD message and the message components for each of the two CAN FD protocols, refer to the following illustration:

- Non-ISO CAN FD protocol:



■ ISO CAN FD protocol:



There are some differences between CAN FD messages and CAN messages:

- The Data field and the CRC field of CAN FD messages can be longer. The maximum payload length of a CAN FD message is 64 bytes.
- The Control fields are different:
 - A classic CAN message always has a dominant (= 0) reserved bit immediately before the data length code.

In a CAN FD message, this bit is always transmitted with a recessive level (= 1) and is called *EDL* (Extended Data Length) or *FDF* (FD Format), depending on the CAN FD protocol you are working with. So CAN messages and CAN FD messages are always distinguishable by looking at the EDL/FDF bit. A recessive EDL/FDF bit indicates a CAN FD message, a dominant EDL/FDF bit indicates a CAN message.

In CAN FD messages, the EDL/FDF bit is always followed by a dominant reserved bit (*r0/res*), which is reserved for future use.
- A CAN FD message has the additional *BRS* (Bit Rate Switching) bit, which allows you to switch the bit rate for the data phase. A recessive BRS bit switches from the standard bit rate to the preconfigured alternate bit rate. A dominant BRS bit means that the bit rate is not switched and the standard bit rate is used.
- A CAN FD message has the additional *ESI* (Error State Indicator) bit. The ESI bit is used to identify the error status of a CAN FD node. A recessive ESI bit indicates a transmitting node in the 'error active' state. A dominant ESI bit indicates a transmitting node in the 'error passive' state.
- Since CAN FD messages can contain up to 64 data bytes, the coding of the DLC (data length code) has been expanded. The following table shows the possible data field lengths of CAN FD messages and the corresponding DLC values.

DLC	Number of Data Bytes
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6

DLC	Number of Data Bytes
0111	7
1000	8
1001	12
1010	16
1011	20
1100	24
1101	32
1110	48
1111	64



If necessary, padding bytes are used to fill the data field of a CAN FD message to the next greater possible DLC value.

For classic CAN messages, the DLC values 1000 ... 1111 are interpreted as 8 data bytes.

- (Valid for the ISO CAN FD protocol only) The CRC fields are different:
 - The CRC field of a CAN FD message was extended by a stuff count before the actual CRC sequence. The stuff count consists of a 3-bit stuff bit counter (reflects the number of the data-dependent dynamic stuff bits (modulo 8)), and an additional parity bit to secure the counter.
 - The start value for the CRC calculation was changed from '0...0' to '10...0'.

Activating CAN FD mode in the RTI CAN MultiMessage Blockset

To ensure that CAN FD messages are properly transmitted and received during run time, the CAN FD mode must be enabled at two places in the RTI CAN MultiMessage Blockset: in the MainBlock and at the CAN controller selected in the MainBlock. To do so, perform the following steps:

- In the ControllerSetup block, select the CAN FD protocol to be used. Refer to [Setup Page \(RTICANMM ControllerSetup\)](#) (RTI CAN MultiMessage Blockset Reference )
- In the MainBlock, select the CAN FD support checkbox. Refer to [General Settings Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference )

To monitor CAN FD messages, it is sufficient to enable CAN FD support in the ControllerSetup block.

Supported database file types

To work with CAN FD messages, you need a suitable database file containing descriptions in the CAN FD format. The RTI CAN MultiMessage Blockset supports CAN FD for the following database file types:

- DBC file
- AUTOSAR system description file
- FIBEX file (FIBEX 4.1.2 only)

CanFrameTxBehavior and CanFrameRxBehavior attributes In AUTOSAR and FIBEX files, the **CanFrameTxBehavior** and/or **CanFrameRxBehavior**

attributes of a message can be defined to specify whether the message is to be treated as a CAN FD message or classic CAN 2.0 message. The RTI CAN MultiMessage Blockset evaluates the attribute as follows:

- If the `CanFrameTxBehavior` attribute is defined for a message in the database file, RTICANMM uses this setting for the message on the CAN bus for both directions, i.e., for sending and receiving the message.
- If the `CanFrameTxBehavior` attribute is not defined in the database for a message, RTICANMM uses the `CanFrameRxBehavior` setting of the message for sending and receiving the message.

Supported dSPACE platforms

The RTI CAN MultiMessage Blockset supports working with CAN FD messages for the following dSPACE hardware:

- SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, or DS6342 CAN Board
- The following dSPACE platforms if they are equipped with DS4342 CAN FD Interface Modules:
 - DS1006 modular system with DS4505 Interface Board
 - DS1007 modular system with DS4505 Interface Board
 - MicroAutoBox II in the following variants:
 - 1401/1507
 - 1401/1511/1514
 - 1401/1513/1514

When you connect a DS4342 CAN FD Module to a CAN bus, you must note some specific aspects (such as bus termination and using feed-through bus lines). For more information, refer to:

- PHS-bus-based system with DS4505: [DS4342 Connections in Different Topologies \(PHS Bus System Hardware Reference !\[\]\(223f1a84e0bc2cacb9c165f716817dcc_img.jpg\)](#))
- MicroAutoBox II: [DS4342 Connections in Different Topologies \(MicroAutoBox II Hardware Installation and Configuration Guide !\[\]\(c437123967ec19fa50ef7951237304ba_img.jpg\)](#))

Working with CAN messages and CAN FD messages

Both messages in CAN format and in CAN FD format can be transmitted and received via the same network.

Related topics

References

[Setup Page \(RTICANMM ControllerSetup\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(bd3b31712ad9bab5a241210fa6925cdd_img.jpg\)](#))

Basics on Working with a J1939-Compliant DBC File

Introduction

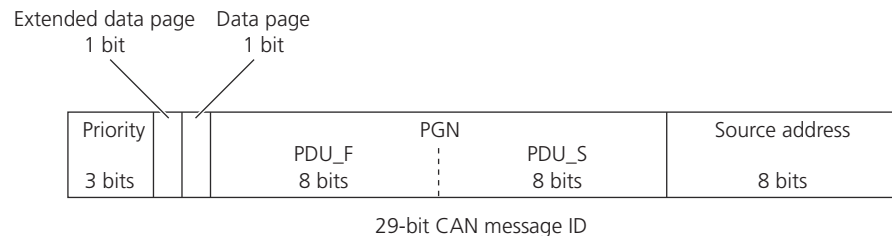
J1939 is a vehicle CAN bus standard defined by the Society of Automotive Engineers (SAE). It is used for communication in heavy-duty vehicles, for example, for communication between a tractor and its trailer.

The RTI CAN MultiMessage Blockset supports you in working with J1939-compliant DBC files.

Broadcast and peer-to-peer communication

CAN message identifier for J1939 Standard CAN messages use an 11-bit message identifier (CAN 2.0 A). J1939 messages use an extended 29-bit message identifier (CAN 2.0 B).

The 29-bit message identifier is split into different parts (according to SAE J1939/21 Data Link Layer):



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

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

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

messages with a $PDU_F \geq 240$ (broadcast messages, also called PDU2 messages), PDU_S specifies the second 8 bits of the PGN and represents the group extension. A group extension is used to increase the number of messages that can be broadcast in the network.

PDU_F (first 8 bits)	PDU_S (second 8 bits)	Communication Mode
< 240	Destination address	Peer-to-peer (message is transmitted to one destination network node)
≥ 240	Group extension	Broadcast (message is transmitted to any network node connected to the network)

Message attributes in J1939-compliant DBC files

A message described in a J1939-compliant DBC file is described by the ID attribute and others.

DBC files created with CANalyzer 5.1 or earlier In a DBC file created with CANalyzer 5.1 or earlier, the ID attribute describing a message actually is the *message PGN*. Thus, the ID provides no information on the source and destination of the message. The source and destination can be specified by the *J1939PGSrc* and *J1939PGDest* attributes.

DBC files created with CANalyzer 5.2 or later In a DBC file created with CANalyzer 5.2 or later, the ID attribute describing a message actually is the *CAN message ID, which consists of the priority, PGN, and source address*. Thus, the ID provides information on the source and destination of the message. Further senders can be specified for a message in Vector Informatik's CANdb++ Editor (*_BO_TX_BU* attribute). When a DBC file is read in, RTICANMM automatically creates new instances of the message for its other senders.

Source/destination mapping

Messages in a J1939-compliant DBC file that are described only by the PGN have no *source/destination mapping*. Messages that are described by the CAN message ID (consisting of the priority, PGN, and source address) have source/destination mapping.

Tip

The RTI CAN MultiMessage Blockset lets you specify source/destination mapping for messages that are described only by the PGN. The mapping can be specified in the RTICANMM MainBlock or in the DBC file using the *J1939Mapping* attribute.

Container and instance messages

The RTI CAN MultiMessage Blockset distinguishes between *container* and *instance messages* when you work with a J1939-compliant DBC file:

Container message A J1939 message defined by its PGN (and Data Page bit). A container can receive all the messages with its PGN in general. If several messages are received in one sampling step, only the last received message is

held in the container. Container messages can be useful, for example, when you configure a gateway with message manipulation.

Instance message A J1939 message defined by its PGN (and Data Page bit), for which the source (transmitting) network node and the destination (receiving) network node (only for peer-to-peer communication) are defined.

Note

The RTI CAN MultiMessage Blockset only imports instances for which valid source node and destination node specifications are defined in the DBC file. In contrast to instances, containers are imported regardless of whether or not valid source node and destination node specifications are known for them during the import. This lets you configure instances in the RTI CAN MultiMessage Blockset.

There is one container for each PGN (except for proprietary PGNs). If you work with DBC files created with CANalyzer 5.1 or earlier, the container can be clearly derived from the DBC file according to its name. With DBC files created with CANalyzer 5.2 or later, several messages with the same PGN might be defined. In this case, either the message with the shortest name or an additionally created message (named `CONT_<shortest_message_name>`) is used as the container for the PGN. The RTI CAN MultiMessage Blockset lets you specify the container type in the RTICANMM MainBlock. If several messages fulfill the condition of the shortest name, the one that is listed first in the DBC file is used. For messages with proprietary PGNs, each message is its own container (because proprietary PGNs can have different contents according to their source and destination nodes).

Network management

The J1939 CAN standard defines a multimaster communication system with decentralized network management. J1939 network management defines the automatic assignment of network node addresses via address claiming, and provides identification for each network node and its primary function.

Each network node must hold exactly one 64-bit name and one associated address for identification.

Address The 8-bit network node *address* defines the source or destination for J1939 messages in the network. The address of a network node must be unique. If there is an address conflict, the network nodes try to perform dynamic network node addressing (address claiming) to ensure unique addresses, if this is enabled for the network nodes.

The J1939 standard reserves the following addresses:

- Address 0xFE (254) is reserved as the 'null address' that is used as the source address by network nodes that have not yet claimed an address or have failed to claim an address.
- Address 0xFF (255) is reserved as the 'global address' and is exclusively used as a destination address in order to support message broadcasting (for example, for address claims).

The RTI CAN MultiMessage Blockset does not allow J1939 messages to be sent from the null or global addresses.

Note

The RTI CAN MultiMessage Blockset interprets attributes in the DBC file like this:

- In a DBC file created with CANalyzer 5.1 or earlier, the *name* network node attributes and the *J1939PGSrc* and *J1939PGDest* message attributes are read in. The *J1939PGSrc* attribute is interpreted as the address of the node that sends the message, the *J1939PGDest* attribute is interpreted as the address of the node that receives the message.
- In a DBC file created with CANalyzer 5.2 or later, the *name* and *NMStationAddress* network node attributes are read in. The *NMStationAddress* attribute is interpreted as the network node address.

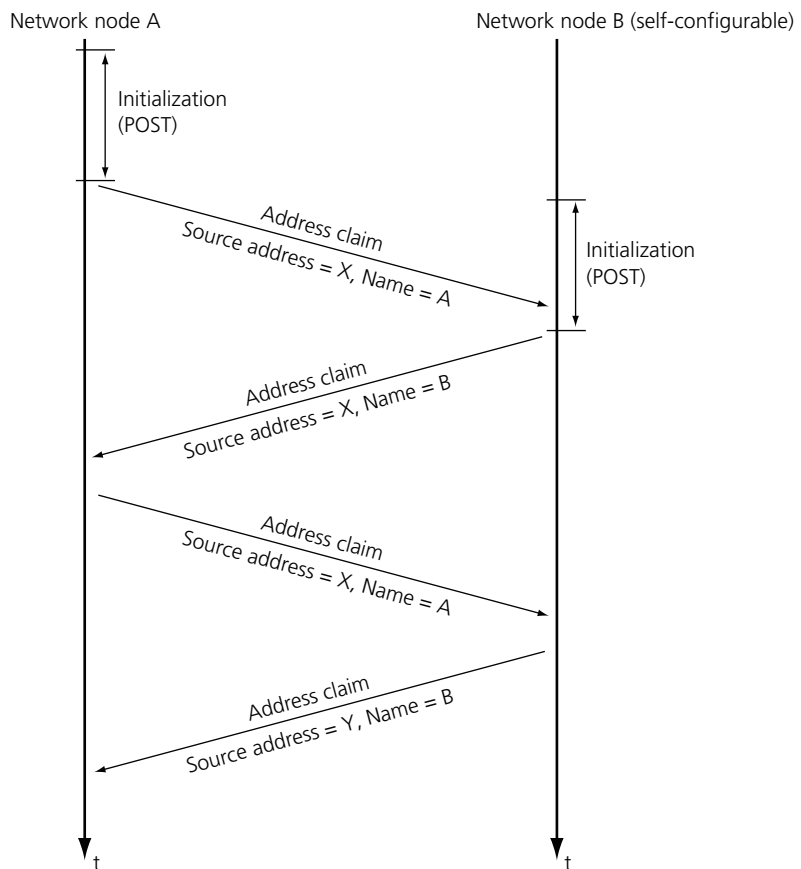
Name The J1939 standard defines a 64-bit *name* to identify each network node. The name indicates the main function of the network node with the associated address and provides information on the manufacturer.

Arbitrary Address Capable	Industry Group	Vehicle System Instance	Vehicle System	Reserved	Function	Function Instance	ECU Instance	Manufacturer Code	Identity Number
1 bit	3 bit	4 bit	7 bit	1 bit	8 bit	5 bit	3 bit	11 bit	21 bit

Address claiming The J1939 standard defines an address claiming process in which addresses are autonomously assigned to network nodes during network initialization. The process ensures that each address is unique.

Each network node sends an address claim to the CAN bus. The nodes receiving an address claim verify the claimed address. If there is an address conflict, the network node with the lowest 64-bit name value (highest priority) gets the claimed address. The other network nodes must claim different addresses.

The following illustration shows the address claiming process with two network nodes claiming the same address. Network node A has a 64-bit name of higher priority.



The following steps are performed in the address claiming procedure:

- Node A starts initialization and the power-on self-test (POST).
- While node B performs initialization and POST, node A sends its address claim message.
- After performing initialization and POST, node B sends its address claim message, trying to claim the same source address as node A.
- In response to the address claim message of node B, the 64-bit names of the network nodes are compared. Because the name of network node A has a higher priority, network node A succeeds and can use the claimed address. Node A sends its address claim message again.
- Network node B receives the address claim message and determines that node A's name has higher priority. Node B claims a different address by sending another address claim message.

The RTI CAN MultiMessage Blockset supports J1939 network management including address claiming for self-configurable address network nodes. This allows network nodes simulated by the RTI CAN MultiMessage Blockset to change their addresses, if necessary, and to update their internal address assignments if addresses of external network nodes are changed.

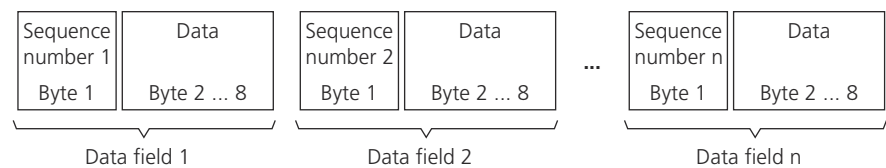
Note

The RTI CAN MultiMessage Blockset supports network management only for network nodes for which network addresses are contained in the DBC file and that have unique 64-bit name identifiers. The node configuration is taken directly from the DBC file and can be adapted on the RTI CAN MultiMessage Blockset dialog pages.

Messages > 8 bytes (message packaging)

Standard CAN messages have a data field of variable length (0 ... 8 bytes). The J1939 protocol defines transport protocol functions which allow the transfer of up to 1785 bytes in one message.

A multipacket message contains up to 255 data fields, each of which has a length of 8 bytes. Each data field is addressed by the 1-byte sequence number, and contains 7 bytes of data. This yields a maximum of 1785 (= 255 · 7) bytes in one message.



The RTI CAN MultiMessage Blockset supports J1939 message packaging via BAM and RTS/CTS:

Broadcasting multipacket messages via BAM To broadcast a multipacket message, the sending network node first sends a *Broadcast Announce Message* (BAM). It contains the PGN and size of the multipacket message, and the number of packages. The BAM allows all receiving network nodes to prepare for the reception. The sender then starts the actual data transfer.

Peer-to-peer communication of multipacket messages via RTS/CTS To transfer a multipacket message to a specific destination in the network, the sending network node sends a *request to send* (RTS). The receiving network node responds with either a *clear to send* (CTS) message or a connection abort message if the connection cannot be established. When the sending network node receives the CTS message, it starts the actual data transfer.

By default, the number of CTS packets is set to 1. To allow peer-to-peer communication for multipacket J1939 messages longer than 8 bytes via RTS/CTS, you can change the default number of CTS packets. The RTI CAN MultiMessage Blockset provides the special MATLAB preference `set_j1939_cts_packet_number`. To change the default number of CTS packets, you must type the following command in the MATLAB Command Window:

```
rtimmsu_private('fcnlib', 'set_j1939_cts_packet_number', 'can', <n>);
```

The argument <n> describes the number of CTS packets. The value must be in the range [1, 255].

Related topics

Basics

[Lesson 13 \(Advanced\): Working with a J1939-Compliant DBC File \(RTI CAN MultiMessage Blockset Tutorial !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)\)](#)

Transmitting and Receiving CAN Messages

Introduction

Large CAN message bundles (> 200 messages) can be managed by a single Simulink block provided by the RTI CAN MultiMessage Blockset.

Defining CAN communication

To define the CAN communication of a CAN controller, you can specify a DBC, MAT, FIBEX, or AUTOSAR system description file as the database file on the [General Settings Page \(RTICANMM MainBlock\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(aa53ad6fea213b8b2226d3077e30533a_img.jpg\)\)](#). You can also define CAN communication without using a database file.

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

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

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

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

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

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

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

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

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

Defining RX messages and TX messages

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

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

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


Triggering TX message transmission

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

For details, refer to [Triggering Options Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference .

Triggering reactions to the reception of RX messages


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

For details, refer to [Raw Data Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference .

Working with raw data

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

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

For details, refer to [Raw Data Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference )

Implementing checksum algorithms

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

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

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

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

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

For details, refer to [Checksum Definition Page \(RTICANMM MainBlock\)](#) (RTI CAN MultiMessage Blockset Reference )

Gatewaying messages

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

You can gateway CAN messages in two ways:

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

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

For details, refer to [RTICANMM Gateway](#) (RTI CAN MultiMessage Blockset Reference )

Related topics

References

[General Settings Page \(RTICANMM MainBlock\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(feabb98897b440bc8695a03336a6e2df_img.jpg\)\)](#)

Manipulating Signals to be Transmitted

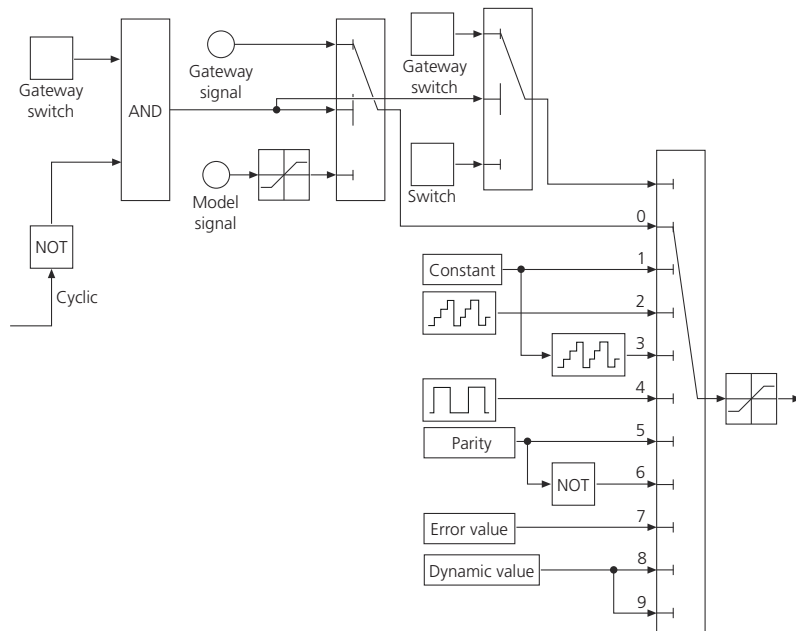
Introduction

All the signals of all the RX and TX messages (see [Defining RX messages and TX messages](#) on page 141) automatically get corresponding entries in the generated TRC file. This allows you to analyze them (signals of RX messages) or change their values (signals of TX messages) with the Bus Navigator in ControlDesk.

Manipulating signals to be transmitted

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

The illustration below visualizes the options.



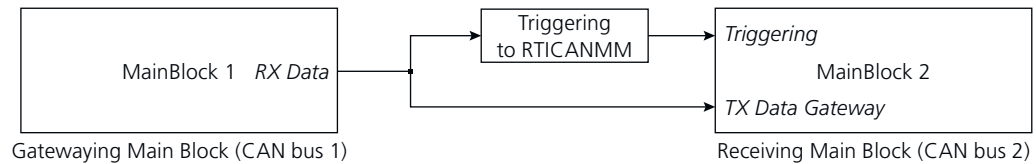
You can switch between these options in ControlDesk.

TX model signal A signal of a TX message whose value can be changed from within the model. By default, the values of TX model signals cannot be changed in ControlDesk. If you also want to manipulate TX model signals from ControlDesk, you have to select them on the [Input Manipulation Page \(RTICANMM MainBlock\) \(RTI CAN MultiMessage Blockset Reference !\[\]\(c444627dab9fee9a1550c053ffaaaae2_img.jpg\)\)](#).

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

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

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



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

Note

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

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

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

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

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

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

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

- Behavior of counter signals

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

- Use of counter signals

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

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

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

- Using counter signals in ControlDesk

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

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

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

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

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

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

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

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

CAN Signal Mapping

Introduction

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

CAN Signal Mapping

Introduction

The CAN subsystem of dSPACE Simulator Mid-Size Based on DS2211 provides two CAN bus interfaces that meet the ISO/DIS 11898 specifications.

I/O mapping

The following table lists the CAN signals of dSPACE Simulator Mid-Size Based on DS2211 and the mapping of the signals to RTI blocks and RTLib functions.

Signal	Channel/Bit Numbers of Related RTI Blocks/RTLib Functions				ECU Connector	Pin
	Related RTI Block(s)	Ch/Bit (RTI)	Related RTLib Functions	Ch/Bit (RTLib)		
CAN Support						
<ul style="list-style-type: none">ISO 11898CANxL: CAN dominant lowCANxH: CAN dominant high						
CAN1L	<ul style="list-style-type: none">RTICAN CONTROLLER SETUPRTICANMM ControllerSetup	CAN1	Slave CAN Access Functions	CAN1	2	D17
CAN1H					2	C17
GND					2	A18
CAN2L		CAN2		CAN2	2	E18
CAN2H					2	D18
GND					2	E15

Interrupts of the DS2211

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

Where to go from here

Information in this section

[Basics of DS2211 Interrupts.....](#) 149

Providing information on the available interrupts.

[Interrupt Handling.....](#) 151

Explaining how interrupts are handled with RTI or RTLib.

Information in other sections

[Slave DSP Interrupts.....](#) 107

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

Basics of DS2211 Interrupts

Basics

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

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

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

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

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

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

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

Interrupt sources

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

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

Interrupt Handling

Introduction

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

Interrupt-driven subsystems in RTI

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

Handcoded models

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

Power Supply Unit

Introduction

To generate the battery voltage, dSPACE Simulator Mid-Size is equipped with one or two remote-controlled power supply units. If you want to simulate a two-voltage system, your simulator must contain two independent power supply units.

Where to go from here

Information in this section

[Characteristics and I/O Mapping of the Power Supply Unit..... 154](#)

The power supply unit consists of a switched-mode power supply and a high rail system to provide the battery voltage for the ECU.

[Controlling the Battery Voltage..... 156](#)

The battery voltage is controlled remotely by dSPACE Simulator Mid-Size.

[Controlling the High Rails..... 158](#)

The high rails are connected to relays for each power supply, so they can be switched on or off.

[How to Control the High Rails..... 160](#)

The high rails can be controlled by the ECU or by the simulator.

Information in other sections

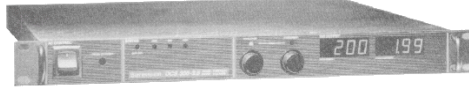
[Components of dSPACE Simulator Mid-Size \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration](#)

Gives an overview of the simulator's components.

Characteristics and I/O Mapping of the Power Supply Unit

Introduction

A power supply unit of the simulator consists of a switched-mode power supply and four high rails (one direct high rail and three switchable high rails) to provide the battery voltage for the ECU.



Switched-mode power supply

The switched-mode power supply is manufactured by Sorenson, Sorenson DCS series 1kW, for example, DCS20-50E (20 V/50 A) and/or DCS60-18E (60 V/18 A). The type which is built-in the simulator, depends on the battery voltage, which must be simulated:

- Output voltage
 - VBAT1 = 0 ... 30 V DC (depending on the type used)
 - VBAT2 = 0 ... 60 V DC (depending on the type used)
 - Controlled by the real-time system. For more information, refer to [Controlling the Battery Voltage](#) on page 156
- Output current
 - 50 A maximum (depending on the type used)
 - Measured by the DS2211 HIL I/O Board
 - Maximum current adjustable via the front knob of the power supply

High rail system

The high rail system connects the power supplies to the ECU. They have the following characteristics:

- Two direct high rail (VBAT1 and VBAT2)
- Six switchable high rails (Three for each power supply). For more information on switching the high rails, refer to [Controlling the High Rails](#) on page 158
- Each high rail has a maximum current of 16 A RMS (total 50 A RMS)
- The high rails of the ECU 1 connector are protected by self-resetting fuses
- Power supply 1 and 2 have the same ground

Voltage control

The voltage are controlled by the real-time system. The simulator uses ADC input channels to measure the voltage and current and DAC output channels control the voltage. Therefore, these channels are reserved and cannot be used for interfacing to the ECU.

The values of power supply 1 are measured and controlled by the following channels:

Channel	Function
DAC12	Sets the output value of VBAT1
ADC15	Measures the VBAT1 current output
ADC16	Measures the actual value of VBAT1

The values of power supply 2 are measured and controlled by the following channels:

Channel	Function
DAC11	Sets the output value of VBAT2
ADC13	Measures the VBAT2 current output
ADC14	Measures the actual value of VBAT2

If the simulator has only one power supply, the DAC11 and ADC13 signals can be used as ordinary signals. To do this, some jumpers must be set, refer to [Reference Signal Jumpers \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(a03a7eb2f4046e1d3c76772003e549ea_img.jpg\)](#)).

For information on how to control the battery voltage, refer to [Controlling the Battery Voltage](#) on page 156.

I/O mapping

The following tables show the mapping of the power supply input and output related to the I/O pins of the ECU 1 connector.

The following table shows the signals for power supply 1:

Signal	ECU 1 Connector Pin	Description
VBAT1A	A6	Direct high rail output
VBAT1B	A7	
VSW11A	A8	Switchable high rail 1 output
VSW11B	A9	
VSW12A	A10	Switchable high rail 2 output
VSW12B	A11	
VSW13A	A12	Switchable high rail 3 output
VSW13B	A13	
GND	B2, B3, B14, B15, B16, B17, B18, E15	Ground
SWREL11–	A15	Control switchable high rail 1
SWREL11+	A14	Control switchable high rail 1
SWREL12–	A17	Control switchable high rail 2
SWREL12+	A16	Control switchable high rail 2
SWREL13–	B1	Control switchable high rail 3
SWREL13+	A18	Control switchable high rail 3

The following table shows the signals for power supply 2:

Signal	ECU 1 Connector Pin	Description
VBAT2	B4	Direct high rail output
VSW21	B5	Switchable high rail 1 output
VSW22	B6	Switchable high rail 2 output
VSW23	B7	Switchable high rail 3 output
GND	B2, B3, B14, B15, B16, B17, B18, E15	Ground
SWREL21–	B9	Control switchable high rail 1
SWREL21+	B8	Control switchable high rail 1
SWREL22–	B11	Control switchable high rail 2
SWREL22+	B10	Control switchable high rail 2
SWREL23–	B13	Control switchable high rail 3
SWREL23+	B12	Control switchable high rail 3

Related topics

Basics

Controlling the Battery Voltage.....	156
Controlling the High Rails.....	158

Controlling the Battery Voltage

Introduction

The battery voltage is controlled remotely by dSPACE Simulator Mid-Size. The range of the battery voltage depends on the type of power supply used. The simulator is prepared for voltages up to 30 V (VBAT1) and 60 V (VBAT2). This range is large enough to simulate all battery voltages that can occur in a typical automobiles.

Use scenario

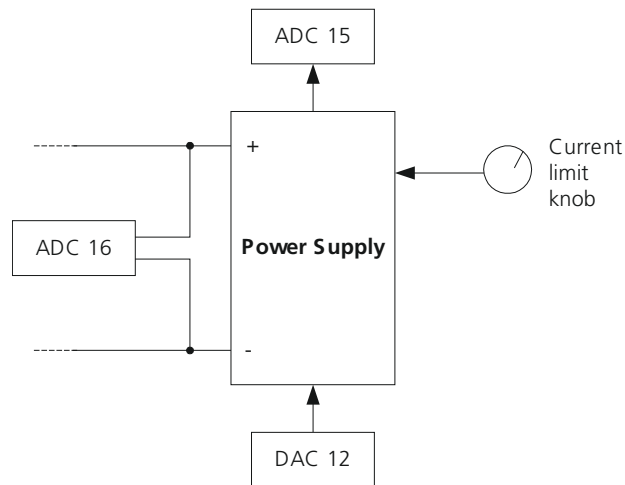
The following table shows some examples of battery voltages of a typical passenger car.

For testing ...	Set the battery voltage to ...
undervoltage	under 9 V Note that the relays for switching the high rails need a voltage above 8 V for switching. Refer to Controlling the High Rails on page 158.
engine cranking	8 ... 11 V

For testing ...	Set the battery voltage to ...
engine running	13.8 ... 14 V
overvoltage	over 17 V

Controlling the voltage

The following illustration shows how the battery voltage is controlled via simulator.



The following table provides information about channels and their functions.

Channel	Function
DAC12	Sets the output value of VBAT1
ADC15	Measures the VBAT1 current output (low-pass filtered by 1.6 kHz)
ADC16	Measures the actual value of VBAT1 (at the DS685 Midplane)

If your simulator has two power supplies, the power supply 2 is controlled by the following channels:

Channel	Function
DAC11	Sets the output value of VBAT2
ADC13	Measures the VBAT2 current output (low-pass filtered by 1.6 kHz)
ADC14	Measures the actual value of VBAT2 (at the DS685 Midplane)

Electrical specification

The following table shows the formula to calculate the voltages and currents:

Value	Formula
Set value of VBAT1	$U = U_{Nom} \cdot U_{DAC12} / 5 \text{ V}$
Actual value of VBAT1	$U = U_{ADC16}$
Current of VBAT1	$I = I_{Nom} \cdot U_{ADC15} / 5 \text{ V}$
Set value of VBAT2	$U = U_{Nom} \cdot U_{DAC11} / 5 \text{ V}$
Actual value of VBAT2	$U = U_{ADC14}$
Current of VBAT2	$I = I_{Nom} \cdot U_{ADC13} / 5 \text{ V}$

Where

U_{DACxx} : Output voltage of DACxx of the DS2211

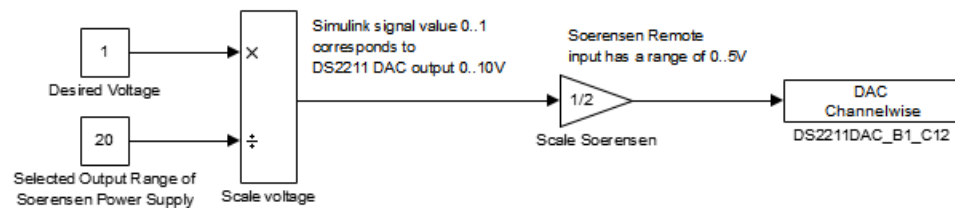
U_{ADCxx} : Input voltage of ADCxx of the DS2211

U_{Nom} : Nominal maximum output voltage of the power supply, for example, for a SORENSEN DCS20-50 is $U_{Nom} = 20 \text{ V}$.

I_{Nom} : Nominal maximum output current of the power supply, for example, for a SORENSEN DCS20-50 is $I_{Nom} = 50 \text{ A}$.

Example

The following example shows a Simulink model that controls power supply 1.

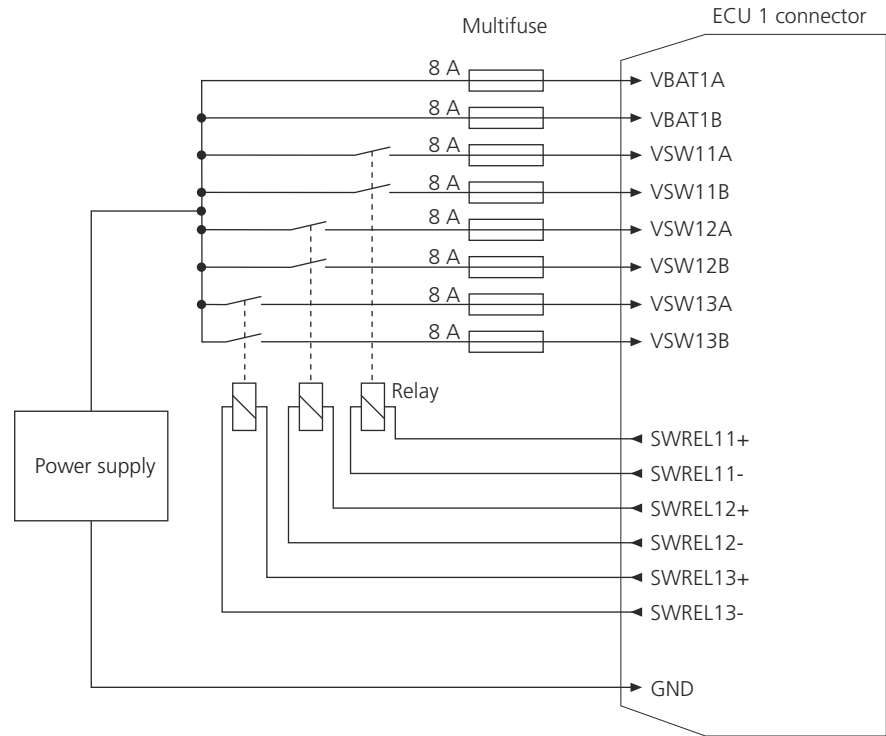
**Related topics****References**

Characteristics and I/O Mapping of the Power Supply Unit..... 154

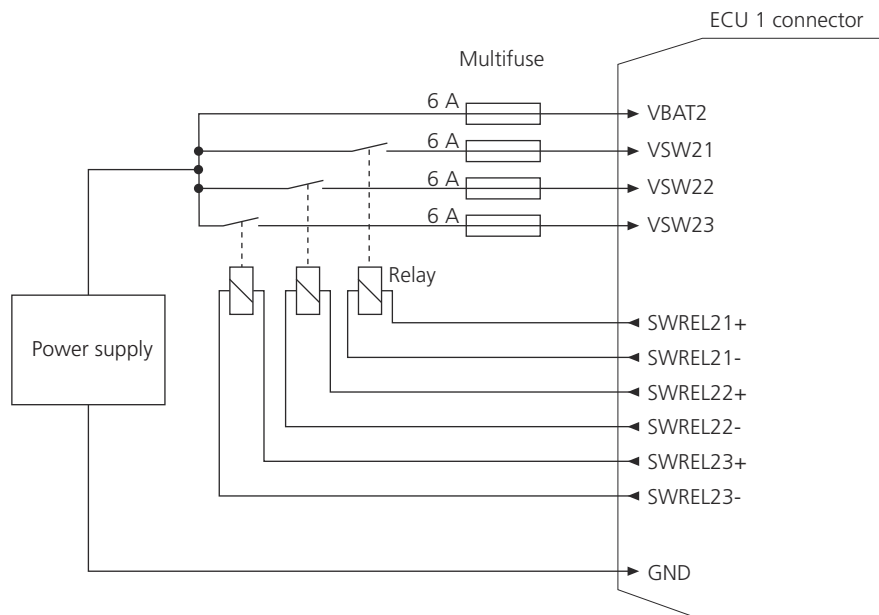
Controlling the High Rails

Introduction

Three high rails are connected to relays for each power supply, so they can be switched on or off. The following illustration shows the principle schematics of the power supply unit for power supply 1.



The following illustration shows the principle schematics of the power supply unit for power supply 2.



Description

All the high rails and the control lines for the switchable high rails are connected to the ECU 1 connector (see the I/O mapping in [Characteristics and I/O Mapping](#)

of the [Power Supply Unit](#) on page 154). The control lines are differential inputs, which are connected to the relays. The relays switch on at a voltage above 8 V. You can control the rails with the ECU or with the simulator, see below.

Related topics

References

[ECU 1 Connector Pinout \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(6605b201d6f14d9b3bcb8ab5f274d107_img.jpg\)](#))

How to Control the High Rails

Introduction

The high rails can be controlled by the ECU or by the simulator.

Possible Methods

You have two possibilities to control the high rails

- If you want to control the high rails with the ECU, refer to Method 1.
- If you want to control a switchable high rail with the simulator, for example, to simulate the ignition key, use a digital output block in your application. Refer to Method 2.

Method 1

To control a high rail with the ECU

- 1 Connect the ECU outputs which control the high rail to the corresponding pins SWRELxy+ and SWRELxy-, x = power supply unit (1 or 2), y = high rail (1, 2, or 3).

For more information, refer to [Configuring the Simulator \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(e3f255517d37bb309a3a931ec4849e6a_img.jpg\)](#)).

Result

The ECU controls the high rails.

Method 2

To control a high rail with the simulator

- 1 Open your real-time model in Simulink.
- 2 Place a DS2211BIT_OUT_Bx_Cy block into your real-time application, and select an unused channel.
- 3 Connect the block input to the signal which controls the high rail.
In addition two bridges have to be installed in the ECU connectors:
- 4 Install a bridge between the output signal of the block (pin at the ECU 2 connector) and the SWRELxy+ pin at the ECU 1 connector.

- 5 Install a bridge between a ground pin and the SWRELxy- pin at the ECU 1 connector.

Result The simulator controls the high rails.

Related topics

Basics	
Controlling the High Rails.....	158

Load Simulation

Where to go from here

Information in this section

[Connecting Loads..... 163](#)

ECUs monitor the current of the actuators, for example, to ensure that they work properly. Therefore, in a hardware-in-the-loop simulation loads must be connected to ECU outputs to obtain the necessary current.

Information in other sections

[Installing Loads \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration \)](#)

You can install loads for the simulator inputs on the load cards inside the simulator or connect them to the front panel.

Connecting Loads

Introduction

ECUs monitor the current of the actuators, for example, to ensure that they work properly. Therefore, in a hardware-in-the-loop simulation loads must be connected to ECU outputs to obtain the necessary current.

In dSPACE Simulator Mid-Size, all the ECU's outputs are connected to load cards, which means you can integrate substitute loads, for example, resistive or inductive loads. If substitute loads do not fulfill your load requirements, you can connect real loads to the simulator instead. External loads can be connected to a load boards connector on the front of the simulator.

Characteristics

A Load/FIU unit has 5 load cards. Each load card supports:

- 10 single-ended loads or
- 5 double-ended loads by using two channels for each load or
- A mixture of single-ended and double-ended loads
- Maximum load current of:
 - Each signal of a load card has a maximum current of 6 A RMS.
 - Each high rail (VBATx, VSWxy) and GND of a load card has a maximum current of 8 A RMS.
 - Each high rail (VBATx, VSWxy) of one Load/FIU unit has a maximum current of 16 A RMS.
 - The GND line of one Load/FIU unit has a maximum current of 40 A RMS.
- Maximum continuous power dissipation:
 - 2 W per load
 - 50 W per load/FIU unit
- Connecting loads between one ECU output and ground or one of the high rails. Refer to [Power Supply Unit](#) on page 153.
- Each load channel is connected to a Failure Insertion Unit. Refer to [Failure Simulation](#) on page 167.
- Each load channel can be disconnected from the DS2211 via a jumper. Refer to [Input Disconnect Jumpers \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration\)](#).

The load cards have front panels with status LEDs and external connectors.

- Each load card has the following status LEDs:
 - 8 rows with red LEDs indicating the status of the high rails
 - 10 rows with yellow LEDs indicating the load states (can be disabled via jumper)
- Each load card has the following external connectors:
 - Connector to the ground and high rails (VBATx, VSWxy) of both power supplies
 - Connector to external loads, for example, original loads

WARNING

High voltage at the front panels of the load card

The ECU outputs are directly connected to the front panels of the load cards. Therefore, voltages higher than 60 V may be connected to the front panels.

The following table shows the assignments of the load cards and channels to the signals.

Channel	1. Load Card	2. Load Card	3. Load Card	4. Load Card	5. Load Card
1	ADC12	ADC11	ADC10	ADC9	ADC8
2	ADC7	ADC6	ADC5	ADC4	ADC3

Channel	1. Load Card	2. Load Card	3. Load Card	4. Load Card	5. Load Card
3	ADC2	ADC1	DIG_IN16	DIG_IN15	DIG_IN14
4	DIG_IN13	DIG_IN12	DIG_IN11	DIG_IN10	DIG_IN9
5	DIG_IN8	DIG_IN7	DIG_IN6	DIG_IN5	DIG_IN4
6	DIG_IN3	PWM_IN8	PWM_IN7	PWM_IN6	PWM_IN5
7	DIG_IN2	PWM_IN4	PWM_IN3	PWM_IN2	PWM_IN1
8	DIG_IN1	INJ6	INJ3	IGN6	IGN3
9	AUX_CAP2	INJ5	INJ2	IGN5	IGN2
10	AUX_CAP1	INJ4	INJ1	IGN4	IGN1

Related topics

Basics

[Installing Loads \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(e2376d476d06eb31946dc01a69a4403a_img.jpg\)\)](#)

HowTos

[How to Label Load Cards \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(0aff635c4179ba9e710b00f4b01d3b20_img.jpg\)\)](#)

Failure Simulation

Introduction

Using the Failure Insertion Units, you can simulate failures in the wiring of an ECU.

Where to go from here

Information in this section

[Safety Precautions for Failure Simulation](#)..... 168

You must note some points when you simulate electrical errors.

[Failure Types](#)..... 168

Cable break or short circuit to a fail potential can be simulated.

[Failure Simulation for ECU Outputs](#)..... 169

To simulate failures for ECU outputs, a Load/FIU unit has five DS791s, which are directly connected to the load cards.

[Make-Before-Break Switching](#)..... 170

New FIU boards support make-before-break switching. This avoids a cable-break detection when a short circuit to a fail potential is switched.

[Failure Simulation for ECU Inputs](#)..... 171

The standard version of dSPACE Simulator Mid-Size Based on DS2211 has no failure simulation unit for the ECU inputs. However, you can also failure-simulate ECU inputs.

[DS793 Sensor FIU](#)..... 171

The DS793 is optionally available for dSPACE Simulator Mid-Size. It is used for failure simulation of ECU inputs.

Information in other sections

[Components of dSPACE Simulator Mid-Size \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration](#)

Gives an overview of the simulator's components.


ControlDesk Electrical Error Simulation via XIL API EESPort

Electrical error simulation is used to disturb the signals that are transmitted between a hardware-in-the-loop (HIL) simulator and a device under test (DUT), which is usually an ECU.

Safety Precautions for Failure Simulation

Safety precautions for failure simulation

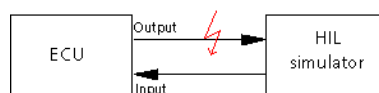
Failure simulation simulates electrical failures in the wiring of an ECU, for example, a short circuit of a signal to the battery voltage. To simulate the failures, the wires are electrical connected to the failure potential. Note the following points for failure simulation:

- Currents can be higher than during normal operation.
- High voltages can occur at pins where they are not expected.
- If the circuit which is failure-simulated contains an inductance, a high voltage can be induced (see [Safety Precautions When Using Inductive Loads \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration](#) )).
- Overvoltage pulses in relays can cause electric arcs at the contacts of the relays. The electric arcs can weld the relay's contacts so that the contacts are permanently short-circuited. A permanent short-circuit can cause further damages, for example, overheating.

Failure Types

Introduction

You can simulate different failure types with dSPACE Simulator Mid-Size.



The wiring of an ECU can be corrupted in various ways:

- Open circuit
An open circuit result from a broken wire or when a wire has lost connection.
- Direct short circuit to another potential
In direct short circuit a wire is connected to another wire without any additional resistance. The wire may be the signal line for ground, one of the battery voltages, or another signal.

Failure Simulation for ECU Outputs

Introduction

To simulate failures for ECU outputs, a Load/FIU unit has five DS791 FIU cards (FIU = Failure Insertion Unit), which are directly connected to the load cards.

Architecture

- Each FIU card has 10 FIU channels for 10 ECU outputs.
- All FIU cards are directly connected to load cards via the DS685 midplane.
- All FIU cards are directly connected to the pins of the ECU 1 connector which contains the signals for the ECU outputs.

Failure simulation

- Each channel can be set to
 - Normal (no failure simulation)
 - Opened
 - Connection to fail plane 1 or fail plane 2
- The fail planes 1 and 2 of all FIU cards are connected to a common fail plane 1 and 2. The common fail planes 1 and 2 can be set to:
 - Open
 - Switched to VBAT1, VBAT2, or ground

Thus, if a channel is connected to a fail plane, it can be connected to VBAT1, VBAT2, ground or another channel, which is connected to the same fail plane (see the example below). For more information on the connection of channels and fail planes, refer to [DS791 Actuator FIU Module \(dSPACE XIL API Implementation Guide\)](#).

- The failure simulation can be controlled with the Failure Simulation module of ControlDesk, refer to [Basics on Failure Simulation \(dSPACE XIL API Implementation Guide\)](#).

Example

The following examples shows how two channels can simulate failure (the values describes the potential where the channels or fail planes are connected to):

Channel 1	Channel 2	Fail Plane 1	Fail Plane 2	Description
Open	Open	–	–	Open circuit (cable break) for channel 1 and channel 2
Fail plane 1	Fail plane 2	VBAT1	Ground	Channel 1 is connected to VBAT1, channel 2 is connected to ground
Fail plane 1	Fail plane 1	Open	–	Short circuit between channel 1 and channel 2
Normal	Fail plane 1	Open	–	Open circuit for channel 2 (if no other channel is connected to fail plane 1)

Switching characteristics

- Loads are disconnected during short circuit simulation (see [Make-Before-Break Switching](#) on page 170)

- The FIU channels can be independently switched for multiple synchronized failure simulation.

Electrical characteristics

- Maximum continuous current
 - Per channel: 6 A
 - Of ground line (total): 8 A
 - Of fail plane (total): 8 A
- Maximum peak current: 14 A
- Relays:
 - Maximum DC load breaking capacity (resistive load, environment temperature of 27°C): 60 V / 8 A
 - Minimum mechanical life: 10^5
- All channels individually overload-protected by self-resetting fuses

Make-Before-Break Switching

Introduction

New FIU boards support make-before-break switching. This improves the characteristic when failures are switched.

Previous switching characteristics Switching failures to the fail planes leads to unwanted behavior. When the failures are simulated, the loads are disconnected from the ECU outputs. When a failure is switched, there is an “open wire” state at the switched ECU output for a short time. This time is sufficient for modern ECUs to mistake it for a cable break.

Make-before-break switching To avoid cable break detection, the FIU cards have been slightly modified. When failures to the fail planes are simulated, the ECU channel is connected to the fail plane first and, then the load (for a DS791) or the DS2211 (for a DS793) is disconnected after a defined transition period.

DS791 The DS791 FIUs support make-before-break switching since firmware version 1.2 (see their labels for the firmware version). The transition period time is typically 22 ms, at maximum 28 ms.

DS793 DS793 Sensor FIUs support make-before-break switching. The transition period time is typically 4 μ s, at maximum 8 μ s.

Failure Simulation for ECU Inputs

Introduction

The standard version of dSPACE Simulator Mid-Size Based on DS2211 has no failure simulation unit for the ECU inputs. However, you can also failure-simulate ECU inputs:

1. Simulating the failure in the model
The easiest way is to implement the failure simulation in the real-time model. You only have to set the ECU inputs (= simulator outputs) to the fail potential.
2. Using a channel for an ECU output
This required a change of the cable harness, refer to [How to Configure the Standard Simulator for Failure Simulation of ECU Inputs \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration\)](#).
3. Extend the simulator
The DS793 Sensor FIU is optionally available from dSPACE. DS793s are connected to the ECU 2 and ECU 3 connectors inside the simulator. See below.

DS793 Sensor FIU

Introduction

The DS793 Input FIU is optionally available for dSPACE Simulator Mid-Size. The DS793 is used for failure simulation of ECU inputs.

Architecture

A DS793 Sensor FIU is a base board for a maximum of nine DS794 FIU modules. Each DS794 FIU module has nine channels for failure simulation. A DS793 Sensor FIU can therefore have a maximum of 81 channels. A simulator can have two DS793 Sensor FIUs which are connected between the ECU 2 and ECU 3 connectors and the outputs of the DS2211. The DS793 connected to the ECU 2 connector must have nine DS794 modules (81 channels). The DS793 connected to the ECU 3 connector must have at least five DS794 modules (45 channels). If you need more channels for signal conditioning, the rest of the second DS793 can also be equipped.

Failure simulation

- Each channel can be set to
 - Normal (no failure simulation)
 - Opened
 - Connection to fail plane 1 or fail plane 2.
The fail planes can be switched to a failure potential: VBAT1, VBAT2, or ground, see [Failure Simulation for ECU Outputs](#) on page 169.
- The failure simulation can be controlled with the Failure Simulation Package of ControlDesk, refer to [Overview of the Graphical User Interface in ControlDesk \(ControlDesk Electrical Error Simulation via XIL API EESPort\)](#).

Switching characteristics

- The DS2211 channels are disconnected during short circuit simulation (see [Make-Before-Break Switching](#) on page 170).
- The FIU channels can be independently switched for multiple synchronized failure simulation.

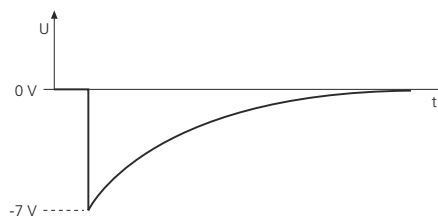
Electrical characteristics

- Failrail potentials: minimum -20 V , maximum 60 V
- Signal voltage: minimum -20 V , maximum 60 V
- Signal current:
 - Specified by DS2211 output signals
 - No signal current protection to minimize the resistance of the signal path
 - Maximal continuous current: $I_{\text{RMS}} = 0.5\text{ A}$
- Failure current between fail plane and output pin:
 - Per channel: $I_{\text{RMS}} = 140\text{ mA}$
 - Total: $I_{\text{RMS}} = 900\text{ mA}$
- Resistances:
 - Of signal path: $R_{\text{avg}} = 1.5\ \Omega$ (at 25°C)
 - Of failure path: $R_{\text{max}} = 9\ \Omega$ (at 25°C)
- Crosstalk between in- and output: $A \geq 60\text{ dB}$ ($f = 1\text{ MHz}$, $R_{\text{load}} = 10\text{ k}\Omega$)
- Capacitance from signal to ground per channel: $C_{\text{typ}} = 300\text{ pF}$
- The failures are switched by semiconductor switches (MOSFETs) which have a parasitic capacitance. This leads to an unwanted behavior, see the following example.

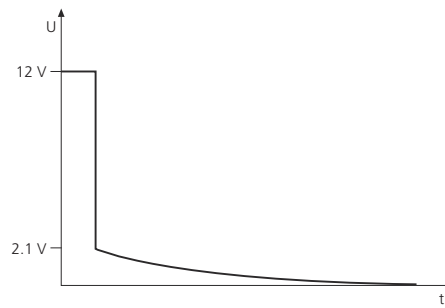
Example

The following examples shows how the parasitic capacitance of the semiconductor switches effects the switching process from the default state to the break state. The total value of the capacitance (break, fail plane 1, and fail plane switch) increases, when the break switch switches to break state from about 240 pF to 300 pF . Therefore, a spike will be generated. The total capacitance is discharged with a resistor of $R_{\text{load}} = 10\text{ M}\Omega$ only (input resistance of the oscilloscope). Typically an ECU input has a lower resistance, which causes a faster discharge of the capacitance.

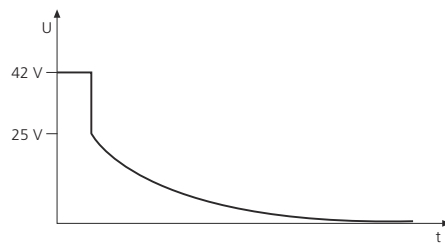
- The first curve shows the switching process when the output value of the DS2211 is 0 V .



- The second curve shows the switching process when the output value of the DS2211 is 12 V.



- The third curve shows the switching process when the output value of the DS2211 is 42 V.



If the behavior shown above disturbs an ECU input, you can do the following:

- To reduce the amplitude of the spike, you can connect a capacitor between ECU pin and GND. However, this extends the time constant of the discharge.
- To reduce the time constant, you can connect a resistor between ECU pin and GND.

Diagnostics

Diagnostic Connector

Introduction

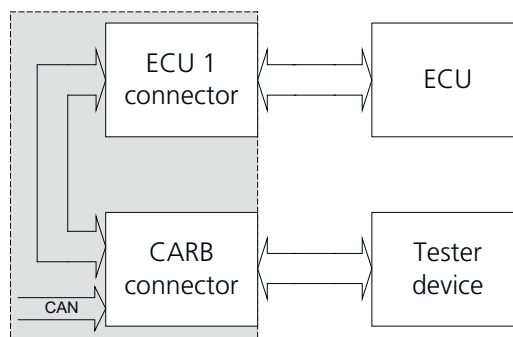
The front of dSPACE Simulator Mid-Size based on DS2211 has a CARB diagnostic and a CANalyzer connector that can be used to connect external testers.

CARB connector

The CARB (California Air Resources Board) diagnostic connector is a connector for the onboard diagnostic standard OBD II. The CARB diagnostic connector is internally connected to the ECU 1 connector, VBAT1 and CAN bus 1. When wiring the cable harness, the ECU can be connected to the ECU 1 connector, and consequently it is also connected to the CARB connector. In addition, the CARB connector contains a connection to the CAN bus 1.

If your tester device requires the battery voltage, you can tap the battery voltage at the Vbat jacks on the front of the simulator. The voltages of both power supplies are available.

The following illustration shows the data flow from the ECU to an external tester device via the ECU and CARB connectors and vice versa.



The CARB connector is placed on the front side of the simulator. It is a Sub-D connector with 15 pins. An interface to the standard 16 pin OBD/EOBD connector is available by dSPACE.

Note

If you use the standard 16 pin OBD/EOBD connector, the CAN bus is connected to pin 6 and pin 14 according to the CARB standard. If the connected diagnostic device uses these pins for other signals, the communication on the CAN bus is destroyed.

For information on the connector pinouts, refer to [CARB Connector Pinout \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(3d8c13c92b853674f749aac6fa869926_img.jpg\)](#)).

For more information on the onboard diagnostic, refer to the OBD II Homepage at <http://www.obdii.com>.

CANalyzer connector

For the diagnostics of the CAN bus, the CAN_L and CAN_H signals are available at the CANalyzer connector. It is a female Sub-D connector with 9 pins.

For information on the CANalyzer connector pinouts, refer to [CANalyzer Connector Pinout \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(17acf1afa8cdf0b67c53d4865a5ed469_img.jpg\)](#)).

Demo for the Simulator

Introduction

This demo is designed for a standard simulator based on DS2211 or DS2202. It consists of 3 parts representing essential milestones to get an HIL simulator system.

You can download a demo from dSPACE:
<http://www.dspace.com/go/MidSizeDemos>.

Where to go from here

Information in this section

[Signal List of the Demo..... 178](#)

The signal list describes the configuration of the simulator, i.e., the jumper settings, and internal and external wiring harnesses. The signal list contains the routing of each signal from the simulator to an external device like an ECU.

[MATLAB/Simulink Model of the Demo..... 188](#)

The signal names from the signal list are used in the MATLAB/Simulink model. The model contains an I/O part that handles the signal flow from and to the simulator. The interfaces to the physical part of the model (i.e., a gasoline engine model) are already prepared.

[ControlDesk Experiment of the Demo..... 203](#)

The ControlDesk experiment with its related layouts provides access to all signals on the real-time hardware of the simulator. It is possible to observe, set, or even manipulate all signal values.

Signal List of the Demo

Introduction

The signal list describes the configuration of the simulator, i.e., the jumper settings, and internal and external wiring harnesses. The signal list contains the routing of each signal from the simulator to an external device like an ECU.

Where to go from here

Information in this section

[Basics on Signal Lists..... 178](#)

The signal list contains all the information on the hardware configuration for the Simulator Mid-Size and the wiring harness.

[Demo Signal List..... 184](#)

The template signal list contains some typical ECU signals in the demo_xx columns. The corresponding Simulink model and the project in ControlDesk are based on these demo signals.

Basics on Signal Lists

Where to go from here

Information in this section

[Structure of the Signal List..... 178](#)

The signal list contains all the information on the hardware configuration for the dSPACE Simulator Mid-Size and the wiring harness.

[Working with the Template Signal List..... 182](#)

The template signal list is used to configure the hardware settings and wiring harness of an off-the-shelf dSPACE Simulator MidSize.

[How to Generate the CSV File..... 183](#)

A CSV file (comma separated value file) is necessary for Failure Simulation module of ControlDesk.

Structure of the Signal List

Introduction

The signal list contains all the information on the hardware configuration for the dSPACE Simulator Mid-Size and the wiring harness. It is created as an Excel

sheet, where each row represents the signal flow from the Simulator to the ECU (electronic control unit) and vice versa.

Template signal list

This demo contains a template signal list for a dSPACE Simulator Mid-Size based on a DS2211 or DS2202 I/O board (`\Signalists\DS22xx_SignalList.xls`). It can be used to configure the simulator and the corresponding wiring harness for an ECU. How to fill the template signal list is explained in [Working with the Template Signal List](#) on page 182.

Demo columns

Additionally this template signal list provides three extra demo columns that are filled with typical ECU signals. For a more detailed signal description and instructions on setting set up a possible Mid-Size configuration, refer to [Demo Signal List](#) on page 184.


Structure

The template signal contains four worksheets, see the following table.



Worksheet	Description
Project_Infos	Here you have to select the hardware components that your dSPACE Simulator Mid-Size comprises (single/double Mid-Size, DS2211/DS2202, FIU/no FIU).
SignalList	The actual signal list contains all the information on the hardware configuration.
LoadCard Label	You can print out this sheet after filling out the signal list to label the load cards of your dSPACE Simulator Mid-Size.
Ref_2211_2202	For internal use only. Do not change this sheet.

Each column of the signal list is explained below.

Column	Description
IO_Index	This is just the numbering of the rows. Each signal has a number to simplify signal list debugging.
Signal_name(1)	The signal name is used in the MATLAB/Simulink model.
Signal_name_group(1)	The signals are combined in signal groups inside the Simulink model for greater clarity. Examples of signal name groups are: sensors, actuators, relays, etc.
ECU_Name(1)	The signal name groups are themselves combined to form an ECU group named in this column. This makes the Simulink model easier to understand. All the internal signals of the simulator are combined in an ECU group named SIMULATOR. Examples of ECU names are: ESP53, EDC16, ECU_1, etc. To provide an FIU demo, this column and the next four columns are partly filled with demo data. You have to edit all the cells according to your own application.
PIN_Number(1)	Contains the pin numbers of the ECU plug for the ECU signal.

Column	Description
PIN_Name(1)	Usually, the ECU pin name does not differ from the signal name. The pin name is used by the failure simulation module of ControlDesk.
Signal_Description(1)	Contains a more detailed description of the signal. The column can contain special electrical characteristics, for example.
Faults_ECU(1)	Defines the failures for the relevant ECU pin. The dSPACE Simulator Mid-Size DS22xx contains five DS791 Failure Insertion Units (FIUs) by default. The DS793 Sensor FIU is optionally available. The permitted failures are configured via a number code. <ul style="list-style-type: none"> ▪ 300: Open circuit ▪ 200: Short circuit to another signal ▪ 100: Short circuit to fail plane 1 (i.e. VBAT1) ▪ 110: Short circuit to fail plane 2 (i.e. VBAT2) ▪ 120: Short circuit to GND
Demo_Signal_name(1)	The demo signal names provide typical signal names by way of example.
Demo_Signal_name_group(1)	The demo signal name groups provide typical signal name groups by way of example.
Demo_ECU_Name(1)	The demo ECU names provide typical ECU names by way of example.
dSPACE_IO_channel	This column contains the electrical type of the dSPACE I/O channel, i.e., DAC (digital analog converter), PWM IN (pulse width modulated signal). This information can be used to combine the dSPACE RTI blocks in the MATLAB/Simulink model (Real-Time Interface) to form groups with the corresponding electrical properties. Differential I/O channels have two signals, so they need two rows in the signal list (i.e. ADC+ and ADC-, RES+ and RES-).
dSPACE_Board	The I/O board is selected from this column, for example, the DS2211_1 as the default I/O board for dSPACE Simulator Mid-Size. The index “_1” indicates the board number. This numbering is necessary if the simulator contains two or more boards of the same type. For more detailed information about the I/O board, refer to DS2211 HIL I/O Board on page 19. dSPACE Simulator Mid-Size can be expanded with additional DS2211 or DS2202 boards or several other I/O boards (refer to Setting Up and Installing Further I/O Boards (dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration )).
Loadboard_Channel	Indicates the load board channel. Almost all simulator input channels are routed to the load boards. The load boards allow an equivalent load (i.e., resistance) or a real load to be connected to an ECU output channel (refer to Installing Loads (dSPACE Simulator Mid-Size Based on

Column	Description
	<p>DS2211 Hardware Installation and Configuration (PDF) and Load Simulation on page 163).</p> <p>For example, a cell entry like "2-3" addresses the second load board and its third channel.</p>
Load_Resistance	If a resistance is used as an equivalent load for this I/O channel, the value is entered here. "real" indicates a real load located outside of the simulator enclosure.
Maximal Load Power	Gives information about the maximum load power of the resistance load used, i.e., 1 W, 2 W, etc.
Load connected to	<p>If an equivalent load is placed at the load board, the potential the load is connected to can be configured with jumpers. The possible connections are:</p> <ul style="list-style-type: none"> ▪ GND: Ground of VBAT ▪ VBAT1: Potential of power supply 1 ▪ VSW11, VSW12, VSW13: Switched rails with the potential of VBAT1 ▪ VBAT2: Potential of power supply 2 ▪ VSW21, VSW22, VSW23: Switched rails with the potential of VBAT2 ▪ CAS: The adjacent load channel to configure double ended loads. <p>For detailed information about configuring load board channels, refer to Configuring the Simulator (dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration) (PDF).</p>
FIU_Module_ID	Contains the FIU module ID in hex. Usually there is no necessity to change the values because the simulator is completely wired internally. The values are used for accessing the FIU boards via the RS232 protocol.
FIU_Module_Channel	Used for accessing the FIU boards via the RS232 protocol. As with the FIU_Module_ID column, there is usually no necessity to change the values.
FIU_Type	This column is predefined for an off-the-shelf dSPACE Simulator Mid-Size based on DS2211 or DS2202. The entries give information about the type of FIU board (DS791 for actuator FIU, DS793 for sensor FIU).
GND_Jumper_State	Contains the setting of ground jumpers for differential RES, DAC and ADC channels. A detailed explanation can be found in Signal Reference Overview (dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration) (PDF).
Disconnect_Jumper	Sometimes it is necessary to disconnect a dSPACE input channel from the load/FIU unit, for example, if the simulator is expanded with current measurement. A detailed explanation can be found in Input Disconnect Jumpers (dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration) (PDF).
HYP-Connector	Contains the number of the Hypertac connector (ECU connector) located at the simulator front. This information is necessary for the wiring harness.

Column	Description
HYP-Connector Pin	Contains the corresponding Hypertac connector pin.
Bridge in HYP Plug	Defines bridges to another Hypertac pin in the wiring harness outside of the simulator. These bridges are necessary, for example, to connect the relay coils (SWREL) with a simulator output channel. For detailed information, refer to Connecting the ECU (dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration ).
CARB Connector PIN	Some pins of the Hypertac connectors at the front of the simulator are directly routed to the CARB connector for diagnostic purposes. This column defines the corresponding signals. Refer to CARB Connector Pinout (dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration ) and Diagnostics on page 175.

Related topics

Basics

[Defining Failure Classes with Signal Files \(dSPACE XIL API Implementation Guide !\[\]\(d3fb9f94af8b26d1c844efa9a98805b0_img.jpg\)](#))

Working with the Template Signal List

Introduction

The template signal list is used to configure the hardware settings and wiring harness of an off-the-shelf dSPACE Simulator MidSize based on DS2211 or DS2202 (DS22xx_SignalList.xls). It also includes an example of a simulator configuration. An explanation on the columns of the signal list can be found in [Structure of the Signal List](#) on page 178.

Workflow

Configuration consists of the following steps:

1. Fill the **ECU_Name**, **PIN_Number**, and **PIN_Name** columns, and assign the ECU signals to dSPACE I/O channels. Fill the **Signal_Name** column and combine signals in groups with the **Signal_name_group** column.
2. Choose adequate failure simulation for each ECU channel to ensure that the ECU will not be damaged.
3. Define adequate loads for ECU actuator channels (real load, resistance, etc.) and set the configuration of the load boards.
4. Define the jumper configuration, including ground jumpers and input disconnect jumpers.

All cells of the template signal list containing red hashes (" # ") or red demo data (for example, "ECU1 ") must be edited during configuration. The character color

changes while the cells are filled with data. Cells containing black characters do not have to be edited. This color scheme makes it possible to see at a glance which cells must still be edited to complete the configuration.

It is also possible to configure a double Simulator Mid-Size DS2211/DS2202 with the template signal list: Select **Double Mid-Size** as the simulator type in the **Project Infos** worksheet. The rows for the I/O channels of the second DS22xx I/O board can be shown via the + buttons on the left side of the signal list.

How to Generate the CSV File

Introduction	A CSV file (comma separated value file) is necessary for Failure Simulation module of ControlDesk.
CSV file	The signal list must be saved as a CSV file so that the Failure Simulation module can read the FIU configuration. It is important that the list separator in the CSV file is a semicolon ';'. The separator sign can be selected under the system settings of Microsoft Windows.
Restrictions	<p>The allowed failure classes for an ECU are defined in the signal file. If you change these settings, it is at your own risk.</p> <p>Signal files delivered with older simulators cannot be used with the XIL API EESPort. dSPACE can upgrade the signal files so that you can use the component.</p>
Preconditions	A signal file in Microsoft Excel format must exist.
Method	<p>To generate a signal file</p> <ol style="list-style-type: none"> 1 In Microsoft Excel, open the signal file which was delivered with your simulator. 2 Change the pin table as required. Be careful when editing the table, as the failure simulation does not work if entries in the table are wrong. Do not change cells that are described as read-only. For a description of the table format, refer to Structure of the Signal List on page 178. 3 Export the table as a CSV file (comma separated values). Ensure that the separator in the file is a semicolon. You can select the separator under System settings.
Result	The signal file is created.

Demo Signal List

Introduction

The template signal list (`DS22xx_SignalList.xls`) contains some typical ECU signals in the `demo_xx` columns. The corresponding Simulink model and the ControlDesk project are based on these demo signals. The following topics use a DS2211 as example, but they are also valid for a DS2202.

Where to go from here

Information in this section

ECU_1 Demo Signal.....	184
ECU_1 is a fictive example ECU with several sensor and actuator channels to demonstrate the tool chain to set up dSPACE Simulator Mid-Size.	
SPARE ECU Demo Signal.....	187
The SPARE ECU contains the I/O channels which are not used for ECU_1 or the SIMULATOR.	
SIMULATOR Demo Signal.....	187
This SIMULATOR ECU contains signals to control the simulator itself.	

Information in other sections

MATLAB/Simulink Model of the Demo.....	188
The signal names from the signal list are used in the MATLAB/Simulink model. The model contains an I/O part that handles the signal flow from and to the simulator. The interfaces to the physical part of the model (i.e., a gasoline engine model) are already prepared.	
ControlDesk Experiment of the Demo.....	203
The ControlDesk experiment with its related layouts provides access to all signals on the real-time hardware of the simulator. It is possible to observe, set, or even manipulate all signal values.	

ECU_1 Demo Signal

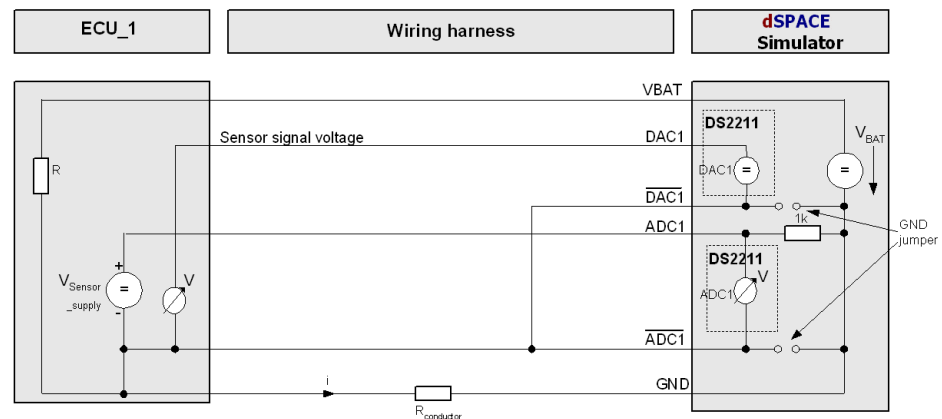
Introduction

ECU_1 is a fictive example ECU with several sensor and actuator channels to demonstrate the tool chain to set up dSPACE Simulator Mid-Size.

Sensor signals

ECU_1 sensor signals are combined in the demo signal name group 'Sensors' (Signal_name_group column).

- Crankshaft and camshaft sensor channels are differential and usually used for engine ECUs. For detailed information, refer to [Angular Processing Unit](#) on page 55.
- Wheel speed sensor channels are differential and usually used for vehicle dynamics stability ECUs, for example, ESP. For detailed information, refer to [Features of the Slave DSP](#) on page 99.
- Resistor channels are used to simulate temperature sensors (NTC or PTC thermistor), etc. For detailed information, refer to [D/R Converter](#) on page 31.
- Two analog input channels (ADC) are used in combination with ECU_1 to measure sensor supply voltages. To simulate a correct supply current for the **sensor1_supply** channel, an equivalent resistive load of 1 kΩ has to be placed at the load board (ADC1). This is specified in the **Load_Resistance** column. The ECU_1 **sensor1_supply_ref** channel is connected to the DS2211 reference channel $\overline{\text{ADC1}}$. The ground jumper must therefore not be connected. In this case, choose "no" in the **GND_jumper_state** column. The following schematic shows the wiring (see also [ADC Unit](#) on page 26).



- An analog output channel (DAC1) is used to simulate an analog sensor, for example, a potentiometer as contained in an acceleration pedal. To simulate a correct sensor voltage, the analog output channel must receive its reference from the sensor supply. The $\overline{\text{DAC1}}$ reference channel is therefore connected to $\overline{\text{ADC1}}$ with a bridge inside the wiring harness. Specify this by writing, for example, "HYP 3.B11", in the **Bridge in HYP plug** column. Channel DAC2 simulates another analog sensor, and its reference is connected to the common sensor ground with the reference jumper. In this case, choose "SGND" in the **GND_jumper_state** column. For details, refer to [DAC Unit](#) on page 29.
- Digital output channels (DIG_OUT) are used to simulate switches, etc. For details, refer to [Bit I/O Unit](#) on page 33.

- PWM out channels are used to simulate PWM sensors of the ECU_1 Unit, for example, rotation speed sensors. For details, refer to [PWM Signal Generation](#) on page 40.

Actuator signals

The ECU_1 actuator signals are combined in the demo signal name group 'Actuators' (**Demo_Signal_name_group** column).

- Simple actuators like fans or heaters can be measured with digital input channels (DIG_IN). It might be necessary to apply equivalent loads to these ECU_1 actuator channels (demo signal name group is 'Switches'). The load resistors are connected to a switched rail (VSW11), which means that ECU_1 provides a down side switch to activate this actuator. The switching of the high rail is done by means of the simulator, refer to [Bit I/O Unit](#) on page 33. This configuration is specified in the **Load_Resistance** and **Load connected to** columns.
- Injection and ignition signals from ECU_1 are measured with injection and ignition channels (INJ, IGN). Both actuator types, ignition coils and injection valves, are normally connected to the simulator as real loads. It depends on the wiring harness of the external real loads which potential the loads are connected to. No jumper setting is necessary for the load boards. Use the keyword "real" in the **Load_Resistance** and **Load connected to** columns for specifying real loads. For details, refer to [APU Basics](#) on page 56.
- The PWM input channels (PWM_IN) are used in combination with ECU_1 to measure the activation of PWM actuators like heaters or electrical fan motors. For details, refer to [PWM Signal Measurement](#) on page 36.

Power supply

In addition, the fictional control unit ECU_1 is connected to the power supply of the Simulator Mid-Size (channels VBAT1 and GNDVBAT1 for continuous supply and channel VSW1 for switched supply). The relay for the switched power supply rail (VSW11) is controlled with a digital output channel of the simulator (channels SWREL11+ and SWREL11-). Channel SWREL11+ is therefore connected to the power supply channel VBAT1, and SWREL- is connected to a digital output channel (DIG_OUT1) with bridges inside the wiring harness. Specify the correct connections in the **Bridges inside HYP plug** column. For details, refer to [Connecting the ECU \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(a870788d6ed9b8fd294b7654a8c8526b_img.jpg\)](#)).

ECU_1 provides a common sensor ground (channel SGND). Inside the simulator the sensor ground is optionally connected to power supply ground via an RC network. For details, refer to [Signal Reference Overview \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(de95854c7ee024cfadc48187bbb781b2_img.jpg\)](#).

SPARE ECU Demo Signal

SPARE ECU

The SPARE ECU contains all other I/O channels which are not used for ECU_1 or the SIMULATOR. The spare channels are used to access every I/O channel of the DS2211 board.

SIMULATOR Demo Signal

Introduction

This SIMULATOR ECU contains signals to control the simulator itself.

SIMULATOR ECU

A SIMULATOR ECU is also defined in the signal list (**Demo_ECU_Name** column). This ECU contains signals to control the simulator itself. The power supply voltage is controlled with analog output channel 12 (DAC12). Voltage and current measurement of the power supply are performed via analog input channels 15 and 16 (ADC15 and ADC16). For a simulator with a second power supply, the channels DAC11, ADC13 and ADC14 are also used for power supply (VBAT2) control. (Refer to [Power Supply Unit](#) on page 153) In that case, change the entries in the **Load connected to** column from ADC13/ADC13 or DAC11/DAC11 to IBAT2 or VBAT2.

The SIMULATOR ECU also controls the relay (SWREL11) for the switched rail VSW11 with a digital output channel (DIG_OUT1).

MATLAB/Simulink Model of the Demo

Introduction

The signal names from the signal list are used in the MATLAB/Simulink model. The model contains an I/O part that handles the signal flow from and to the simulator. The interfaces to the physical part of the model (i.e., a gasoline engine model) are already prepared.

Where to go from here

Information in this section

Basics of the Simulink Model..... 188

Gives an overview of the Simulink model for the demo.

Model User Interface..... 189

This block is introduced to define a location from which access to the dynamic model is possible.

IO Subsystem..... 190

The IO subsystem contains the RTI blocks to access the hardware and additional blocks for scaling signals, providing test automation access, and switching between different configurations.

I/O User Interface..... 200

This block defines a location from which you can access the entire I/O model, and therefore all simulator inputs and outputs.

Basics of the Simulink Model

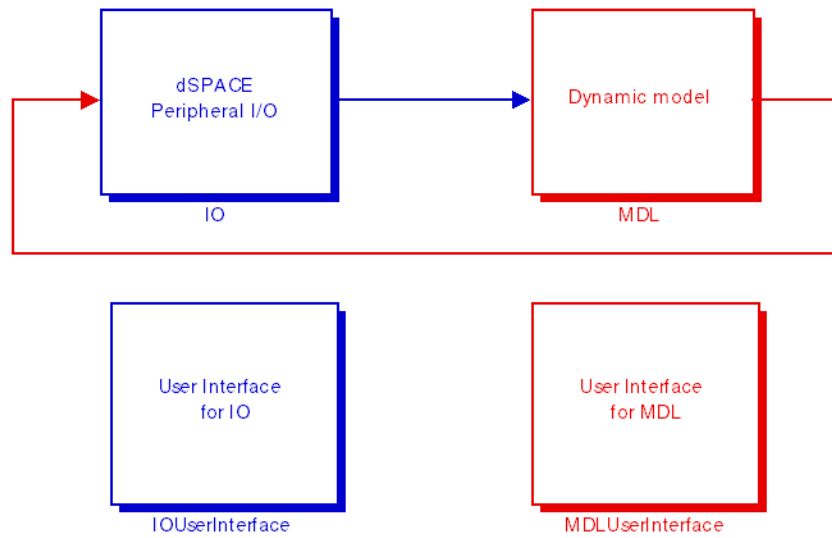
Overview of the Simulink Model

Introduction

The Simulink model for the demo is divided in the four subsystems, which are described briefly in this topic.

Simulink model

The topmost level of the Simulink model is shown below. The models for this demo are: `DemoDS2211MidSize.mdl` and `DemoDS2202MidSize.mdl` in `\Simulation.current`.



Subsystem	Task
IO subsystem	Contains the dSPACE RTI blocks to access the hardware and additional blocks for scaling signals, providing test automation access, and switching between different configurations. Refer to IO Subsystem on page 190.
MDL subsystem	Contains the dynamic model, for example the engine or vehicle model.
IOUserInterface subsystem	Defines a location from which you can access the entire I/O model, and therefore all simulator in- and outputs. Refer to IOUserInterface Subsystem on page 200.
MDLUserInterface subsystem	Defines a location from which access to the dynamic model is possible. Refer to MDLUserInterface Subsystem on page 189.

Model User Interface

MDLUserInterface Subsystem

Introduction

This block defines a location from which access to the dynamic model is possible.

Tasks

The MDLUserInterface block performs several tasks:

- The dynamic model components used only for user access are separated from the overall dynamic model necessary for proper HIL operation.
- You can create an individual structure with cascaded subsystems entirely independently of the structure of the real model. The benefit is a convenient variable tree in ControlDesk for easy navigation.
- Signals can be converted into user-specific units (for example, kph, bar or rpm) without disturbing the SI unit based signal flow in the real model (for example, mps, P, or radps).

Communication to MDL subsystem

The communication between the MDL and the MDLUserInterface block is realized by Simulink's From/Goto connection blocks.

IO Subsystem

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

[Overview of the IO Subsystem](#)..... 191

The IO subsystem contains the dSPACE RTI blocks to access the hardware and additional blocks for scaling signals, providing test automation access, and switching between different configurations.

[ScalingToHardware](#)..... 192

The ScalingToHardware subsystem resides inside the IO subsystem.

[MappingToHardware](#)..... 194

The subsystem maps the ECU-related signals to the hardware interface (discrete signals) and to the Protocols subsystem (bus signals).

[HardwareInterface](#)..... 195

The subsystem contains all standard RTI blocks which control discrete hardware channels.

[Protocols](#)..... 196

The subsystem usually contains one separate subsystem for each bus type.

[Soft ECU](#)..... 196

A software ECU (soft ECU) is a way to reproduce a particular functionality or behavior of real ECUs which are not available yet but are necessary for proper operation of the ECUs connected to the simulator.

[MappingFromHardware](#)..... 197

The subsystem is used to assign a real hardware or protocol signal to an ECU-related signal.

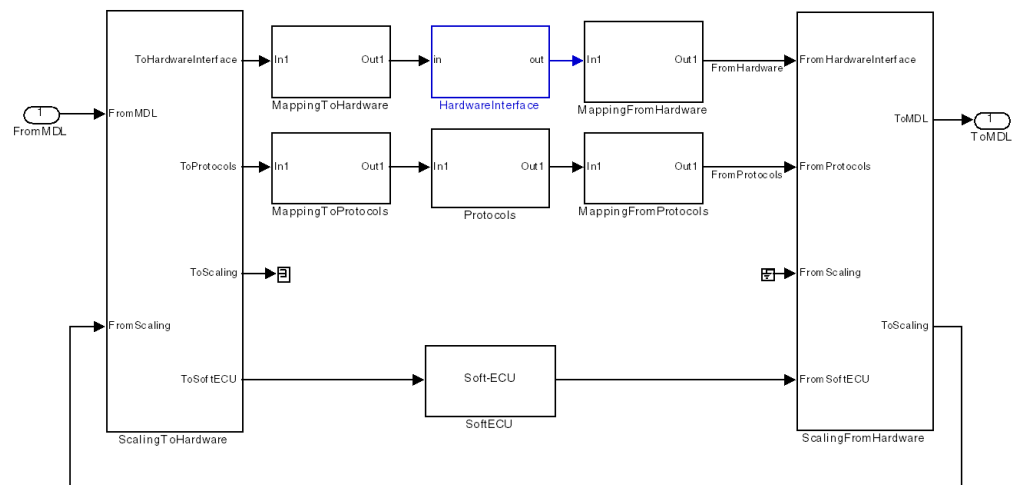
ScalingFromHardware..... 198

The subsystem contains subsystems for each supported ECU and the simulator itself.

Overview of the IO Subsystem

IO subsystem overview

The IO subsystem contains the dSPACE RTI blocks to access the hardware and additional blocks for scaling signals, providing test automation access, and switching between different configurations.



From left to right, the data flow to or from hardware is categorized by separate blocks for scaling, mapping and hardware access. The innermost blocks represent the real access point to the dSPACE hardware, using RTI blocks or S-functions.

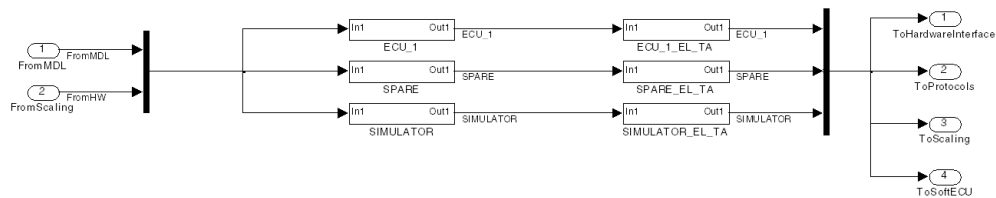
- [ScalingToHardware](#) on page 192
- [MappingToHardware](#) on page 194
- [HardwareInterface](#) on page 195
- [Protocols](#) on page 196
- [Soft ECU](#) on page 196

- [MappingFromHardware](#) on page 197
- [ScalingFromHardware](#) on page 198

ScalingToHardware

ScalingToHardware subsystem

The ScalingToHardware subsystem resides inside the IO subsystem.



The block has two input vectors:

- **FromMDL** contains all the values sent by the dynamic model (MDL) block.
- **FromHardware** contains signals received by dSPACE hardware. This can be used to achieve direct, short data flow that gets signals from hardware into the output part of the simulator. A typical application for this is to transmit the measured reference signal values for realizing ratiometric sensors.

All data fed into the ScalingToHardware subsystem is combined in one array by using the Simulink Bus Selectors. The signals are also grouped. One group is normally for controlling the HIL simulator itself, usually named **SIMULATOR** (refer to [SIMULATOR Demo Signal](#) on page 187). There is also at least one additional group, with the same name as the ECU that the signals leads to (i.e. **ECU_1**, refer to [ECU_1 Demo Signal](#) on page 184).

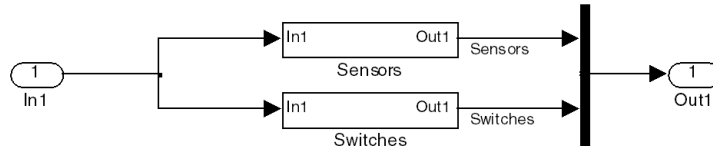
The Test Automation blocks (TA) (for example, **ECU_1_EL_TA**) contain an optional manipulation access to each single signal in electrical (EL) quantities. Typically, the model contains TA blocks not only after scaling, but also before. These provide an entry point for physical test automation that manipulates the signal flow from the dynamic model to the hardware based on engineering units (rpm, bar, km/h, etc). Whether TA blocks are present depends on the actual application. This demo has no physical model, so physical test automation is only implemented for power supply control signals in the **SIMULATOR** group.

Finally, all ECU group signal vectors are themselves combined in one array and sent to the 4 outputs:

- **ToHardwareInterface**
- **ToProtocols**
- **ToScaling**
- **ToSoftECU**

Function groups

Underneath an ECU subsystem (i.e., ECU_1), signals are usually separated into functional groups. The following illustration shows the functional groups of ECU_1 (model path: `./ScalingToHardware/ECU_1`).



The aim of introducing function groups is to achieve a clear and application-oriented view of the model. The number and names of the additional group blocks depend on the available functionality of the ECU and on how it is subdivided.

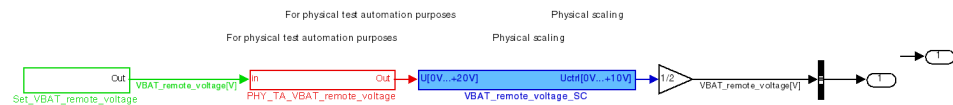
Examples of function groups:

- IgnitionInjection
- ThrottleValve
- PowerSupply
- Sensors
- Actuators

Signal scaling

The signal scaling is done in the functional groups.

The following Simulink block diagram shows the contents of the Power_supply functional group of the SIMULATOR group (model path: `./ScalingToHardware/SIMULATOR/Power_supply`).



The green block (on the left) represents a test automation block with signal write access function (Constant block) to stimulate the channel. Because this demo does not contain a physical model, these set blocks are necessary for each output channel. The red block provides test automation manipulation access to the signal. The corresponding control block resides inside the `IOUserInterface` block. The blue block and the Gain block do the signal scaling.

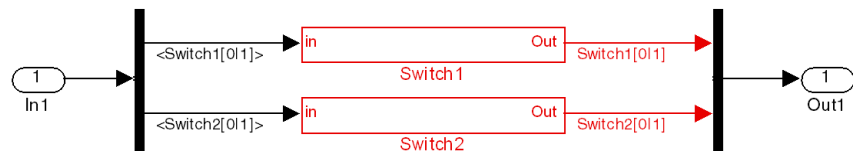
The scaling underneath the function group is the innermost level, which:

- Receives physical value signals from the dynamic model if available.
- Creates nondynamic physical signals. Even if a dynamic model is available, a lot of signals are of a static nature and must be provided to the ECU only to prevent the ECU from detecting a fault. This can easily be achieved by using Simulink's Constant blocks.
- Converts the physical values to suitable electrical values. For example, a look-up table is often used to convert the coolant temperature (in Kelvin or °Celsius) into electrical quantities which then can be linked to the associated resistor channel.

- Creates an output structure for all ECU variants controlled by the simulator. Depending on whether different ECU variants must be taken into account, one or more groups of signals are implemented by using Simulink's Mux blocks.

TA groups

Within the test automation (TA) groups (i.e. ECU_1_EL_TA), signals are combined and labeled in the same way as described previously, and the same subdivision is used. The illustration shows the test automation subsystem for ECU_1 switch signals (model path: `./ScalingToHardware/ECU_1_EL_TA/switches`)

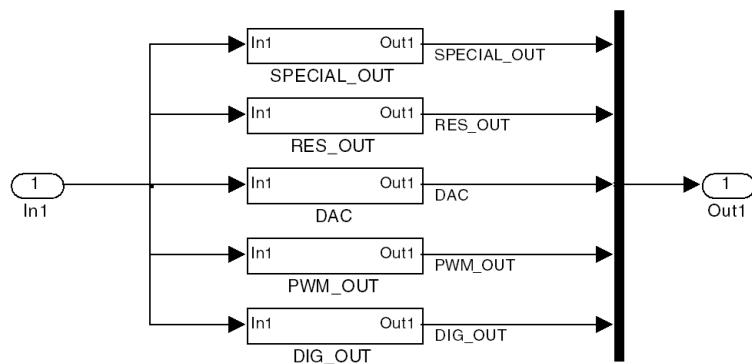


The red blocks provide test automation manipulation access to the signals. The corresponding control blocks reside inside the IOUserInterface subsystem.

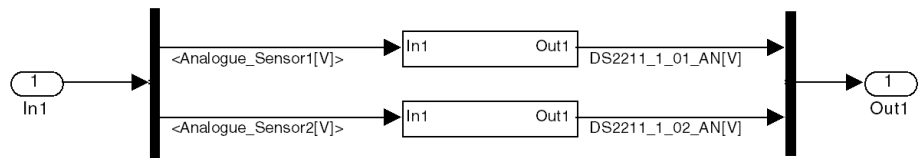
MappingToHardware

MappingToHardware subsystem

The subsystem maps the ECU-related signals to the hardware interface (discrete signals) and to the Protocols subsystem (bus signals, for example, CAN, LIN, etc). For example, the water temperature signal for the engine controller is mapped to an analog output of the DS2211 board, channel DAC1, because the simulator has to generate the appropriate sensor signal. The following illustration shows the contents of the MappingToHardware subsystem.



The subsystems shown are for classifying the signals in terms of their electrical characteristics (digital output, analog output, etc). The following diagram shows an excerpt from the DAC subsystem of the demo DS2211 Simulator Mid-Size model.



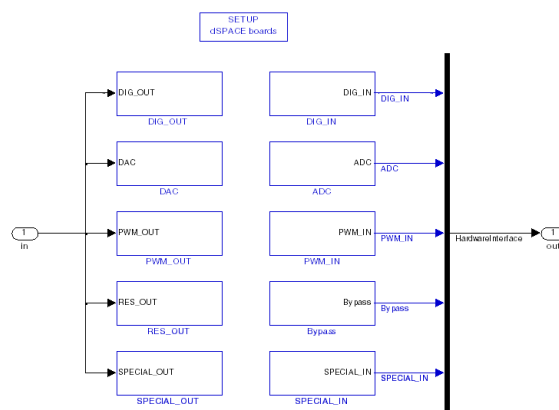
The labeling provides information on which board and which channel a signal is connected to.

HardwareInterface

HardwareInterface subsystem

The subsystem contains all standard RTI blocks which control discrete hardware channels. Note that RTI drivers or S-functions which are necessary to create protocols for realizing data transfer via buses reside in the Protocols folder.

The following block diagram shows the contents of the HardwareInterface subsystem.

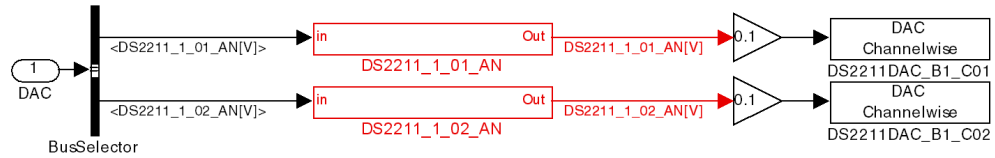


To simplify the handling, the signals are classified using a subsystem for each group of signals which have the same characteristics (DIG_IN, ADC etc.).

For more information about RTI blocks of the DS2211/DS2202 I/O board used in this Mid-Size demo, refer to [DS2211 RTI Reference](#) or [DS2202 RTI Reference](#).

The hardware subsystems can be divided into data outputs and data inputs. The actual contents of the subsystems can vary and depend on the application. Apart from dSPACE RTI blocks, inverse scaling may be implemented due to signal conditioning modules (SC), and built-in user access (test automation) for test purposes might also be available.

The illustration below shows a typical signal flow in the hardware subsystems (analog output channels). It is an excerpt from the demo Simulator Mid-Size MATLAB/Simulink model.



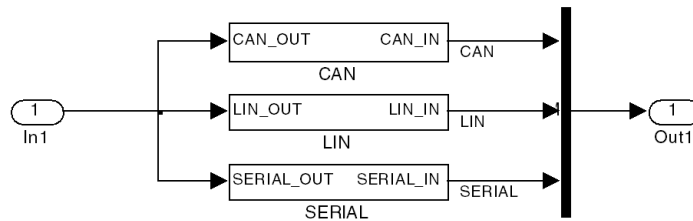
The gain blocks do the electrical signal conditioning (value: 0.1). The red blocks are another means of test automation signal intervention at dSPACE signal level. They are used to perform stimulus tests with a dSPACE Simulator Mid-Size.

Protocols

Protocols subsystem

The subsystem usually contains one separate subsystem for each bus type. The default model is prepared for CAN, LIN and serial interface support.

The following illustration shows the contents of the Protocols subsystem.



The contents of the subsystems can be very different and depend entirely on the bus system which must be supported. The model shipped with this demo contains a link to other demo models for CAN, LIN and serial protocols in each protocol subsystem.

Soft ECU

Software ECU

A software ECU (soft ECU) is a way to reproduce a particular functionality or behavior of real ECUs which are not available yet but are necessary for proper operation of the ECUs connected to the simulator. A software ECU might also be used to emulate plausible dynamic CAN bus traffic.

The Simulink subsystem which accommodates the soft ECU is located in the center of the IO subsystem. If this block is used to simulate an ECU, the corresponding control algorithms can be found here, assuming such a model component is available.

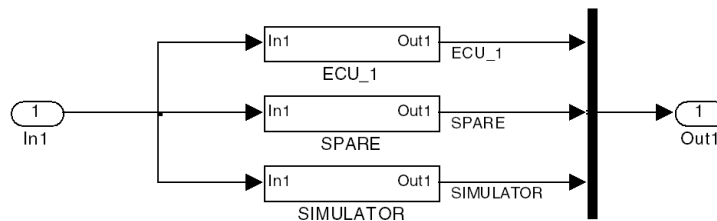
With skillfully implemented variant handling, soft ECUs can even be used as an alternative to real ECUs, with the ability to switch back and forth between them in real time (model-in-the-loop).

MappingFromHardware

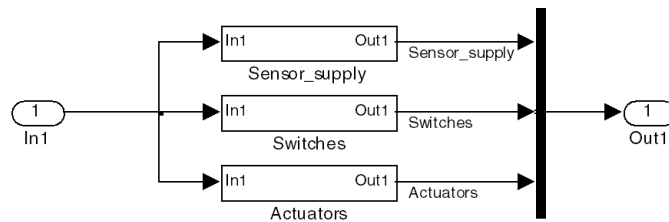
MappingFromHardware subsystem

The aim of this block is to assign a real hardware or protocol signal to an ECU-related signal. For example, the frequency signal provided by a PWM RTI block of a DS2211 is mapped to the cooling fan speed control signal.

The following illustration shows the inner model structure of the MappingFromHardware subsystem.



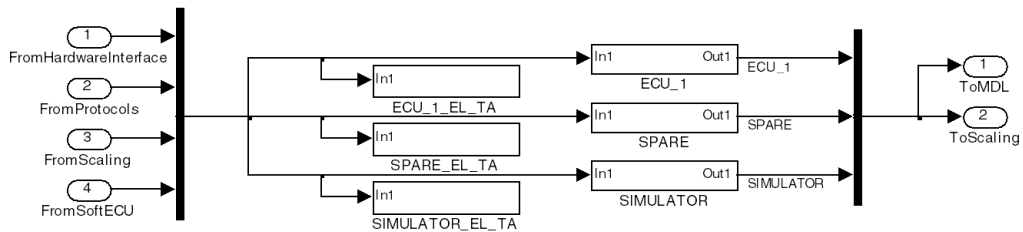
The MappingFromHardware subsystem includes one subsystem for each supported ECU. Since some simulator inputs are usually controlled by the model, there is usually one subsystem called SIMULATOR. Beneath the ECU groups the signals are combined in functional groups like Sensor_supply or Actuators, as the following block diagram excerpt from the demo model shows (model path: `./IO/MappingFromHardware/ECU_1`).



ScalingFromHardware

ScalingFromHardware subsystem

The block diagram excerpt from the demo model shows the structure of the ScalingFromHardware subsystem.



The block has four input vectors:

- **FromHardwareInterface** contains the signals received by the dSPACE hardware.
- **FromProtocols** contains the signals received on CAN or LIN network nodes.
- **FromScaling** is a direct line from the **ScalingToHardware** subsystem.
- **FromSoftECU** contains signals generated by the **SoftECU** subsystem.

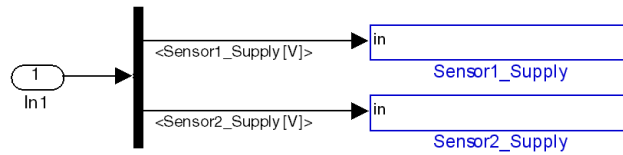
All the values are combined in one array to simplify access to single values using the Simulink Bus Selectors. You will find a separate pair of subsystems for each supported ECU (i.e., **ECU_1_EL_TA** and **ECU_1**). The simulator itself, and its generated signals, is also seen as an ECU, and is usually labeled **SIMULATOR**. The subsystems with the suffix **"_EL_TA"** on the left side provide read access to the signals fed into the subsystem by means of test automation blocks. The connected subsystems on the right side are for accessing the physically processed signals.

Finally, the ECU-related signal vectors are combined in one array that is sent to the two outputs:

- **ToMDL** is a main line fed directly to the dynamic model (MDL).
- **ToScaling** is for looping signals back to the **ScalingToHardware** subsystem.

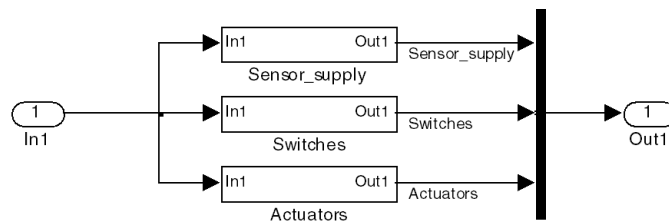
Test automation

The Test Automation blocks (for example, **ECU_1_EL_TA**) introduced in **ScalingFromHardware** comprise access blocks to the measured hardware inputs. The following model excerpt shows the blue Test Automation blocks as signal sinks (model path: `./IO/ScalingFromHardware/ECU_1_EL_TA/Sensor_supply`). The blue blocks contain a Simulink Goto block. The corresponding From block is inside the **IOMUserInterface** subsystem.



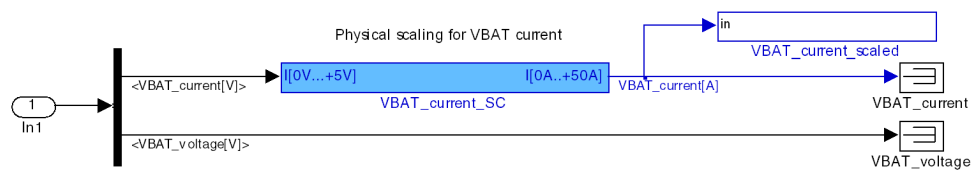
Functional groups

As with the `ScalingToHardware` system, functional groups are specified. The example block diagram shows the contents of the `ECU_1` subsystem of the `ScalingFromHardware` subsystem.



The aim of grouping the ECU signals into functional groups is to create a clear and user-friendly view of the model. The number and names of the additional group blocks depend on the available functionality of the ECU and on how it is subdivided.

The functional group block is the innermost structure level, and mainly contains Simulink Bus Selector blocks which provide access to the signals from the I/O hardware. Appropriate transfer functions or look-up tables for signal processing must be located in the scaling subsystem. The following block diagram shows the scaling of power supply current measurement (model path: `/IO/ScalingFromHardware/SIMULATOR/Power_supply`).



The upper right blue block again represents a test automation access to the physically scaled signal. Its corresponding block is inside the `IOUserInterface` subsystem at model root level. By default, the signals are connected to Simulink's termination blocks, except those which are actually connected to a dynamic model subsystem. This entirely depends on the actual application.

I/O User Interface

IOUserInterface Subsystem

Introduction

This block is introduced to define a location from which users can access the entire I/O model, and therefore all simulator inputs and outputs.

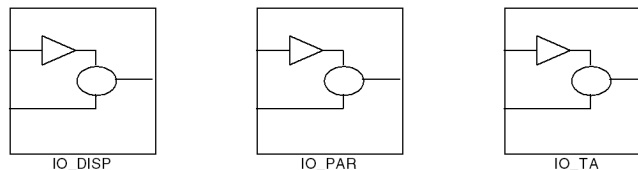
Tasks

Like the model user interface, the `IOUserInterface` subsystem performs several tasks

- The I/O model components used only for user access are separated from the overall model necessary for proper HIL operation.
- You can create an individual structure with cascaded subsystems entirely independently of the structure of the real model. The benefit is a convenient variable tree in ControlDesk for easy navigation.
- Signals can be converted into user-specific units, for example, kph, bar, rpm, without disturbing the SI unit based signal flow in the real model, for example, mps, P, radps.

Structure

The `IOUserInterface` subsystem of this demo has the following structure.

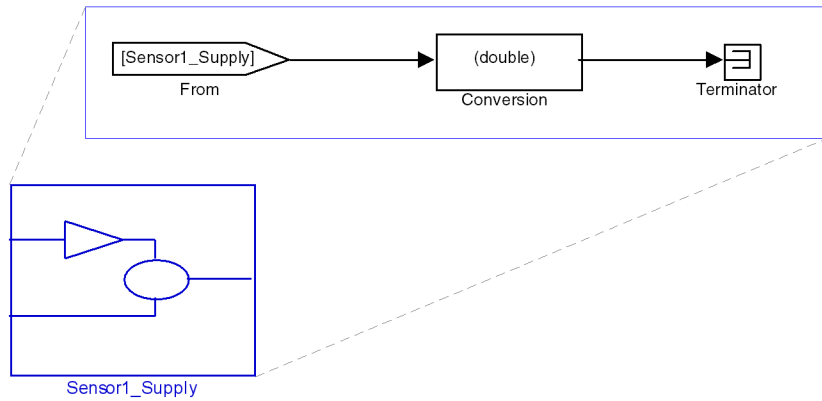


Every subsystem, `IO_DISP`, `IO_PAR` and `IO_TA`, provides a different type of access to signals inside the I/O subsystem. Their substructure is similar to the `ScalingToHardware` subsystem. All signals are first combined in ECU groups (`ECU_1`, `SPARE`, `SIMULATOR`) and afterwards in functional groups (sensors, actuators). All communication between the IO and the `IOUserInterface` subsystem is realized with Simulink's From/Goto connection blocks.

The `IO_DISP` subsystem contains all test automation blocks that grant read access to dSPACE input channels. The corresponding test automation blocks reside inside the `ScalingFromHardware` subsystem (blocks with blue frame color, refer to [ScalingFromHardware](#) on page 198).

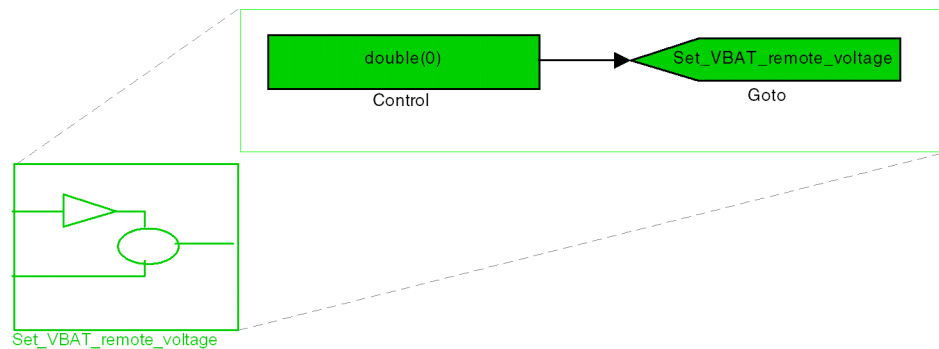
Sensor_supply

This example test automation block is located here: `./IOUserInterface/IO_DISP/ECU_1/Sensor_supply/`



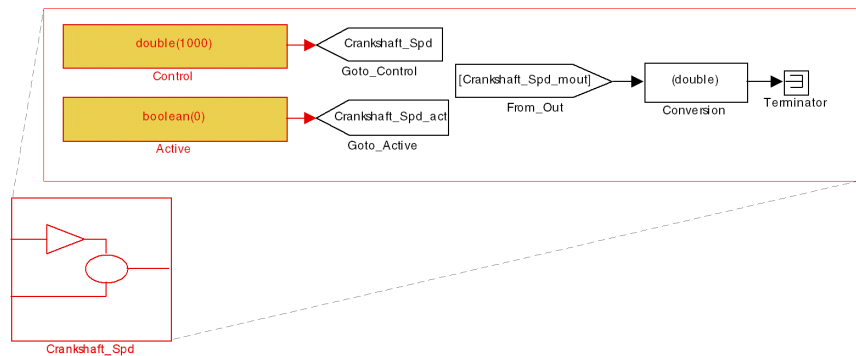
The `IO_PAR` subsystem contains all test automation blocks that grant write access to signals inside the `ScalingToHardware` subsystem (i.e. signals inside `./IO/ScalingToHardware/ECU_1/Sensors`). These blocks have a green border and their names begin with "set", refer to [ScalingToHardware](#) on page 192.

This example test automation block is located here: `./IOUserInterface/IO_PAR/SIMULATOR/Power_supply/`



The `IO_TA` subsystem contains all test automation blocks that grant manipulation access to signals inside the TA part of the `ScalingToHardware` subsystem (i.e., signals in `./IO/ScalingToHardware/ECU_1_EL_TA/Sensors`). These blocks have a red border.

This example test automation block is located here: `./IOUserInterface/IO_TA/ECU_1/Sensors/`



The Control constant block contains the manipulation value for this signal. The Active constant block holds a digital value which controls whether the signal value (for example, crankshaft speed) is passed through or set to the manipulation value. The From_Out block reads back the current signal value.

ControlDesk Experiment of the Demo

Introduction	<p>The ControlDesk experiment provides access to all signals on the real-time hardware. It is possible to observe, set, or even manipulate all signal values.</p> <p>The ControlDesk experiment combines the variable description file, the layout files and the failure simulation files.</p>
--------------	--

Where to go from here	<div><div>Information in this section</div><div><div><div>Layouts.....203</div><div>The experiment in ControlDesk contains three groups of layouts.</div><div>Failure Simulation.....213</div><div>The ControlDesk experiment also contains a failure simulation file.</div></div></div></div>
-----------------------	--

Layouts

Introduction	<p>In general the experiment in ControlDesk contains three groups of layouts.</p>
--------------	---

Where to go from here	<div><div>Information in this section</div><div><div><div>Data Connections to ControlDesk Instruments.....204</div><div>All layout data connections to ECU_1, SPARE and SIMULATOR signals point to Test Automation blocks inside the IOUserInterface subsystem.</div><div>ECU_1 Layouts.....206</div><div>The ECU_1 layouts provide access to all ECU signals.</div><div>SPARE Layouts.....208</div><div>The SPARE layouts provide access to all spare signals.</div><div>DS2211 Layouts.....209</div><div>The DS2211 layouts provide direct access to each I/O channel of the DS2211 board.</div><div>Additional Layouts.....210</div><div>There are additional layouts for controlling the power supply, controlling the simulation, accessing the complex comparator of the DS2211, and an example of a cockpit.</div></div></div></div>
-----------------------	---

Data Connections to ControlDesk Instruments

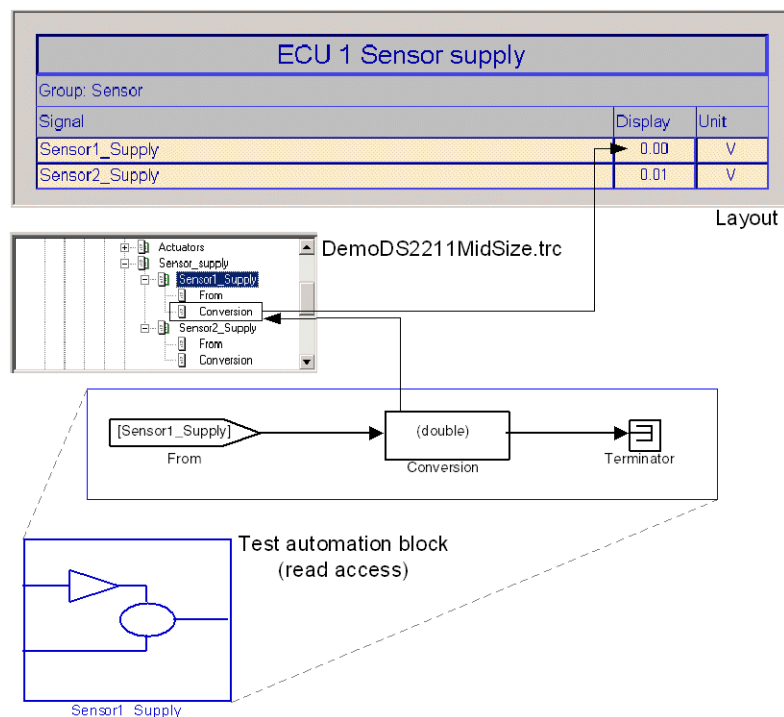
Introduction

All layout data connections to ECU_1, SPARE and SIMULATOR signals point to test automation Simulink blocks inside the IOUserInterface subsystem (see [IOUserInterface Subsystem](#) on page 200). The data connections for direct DS2211 I/O access point to Test Automation blocks that reside inside the HardwareInterface subsystem of the IO subsystem (refer to [HardwareInterface](#) on page 195).

The layout data connections to all three kinds of Test Automation blocks are described below.

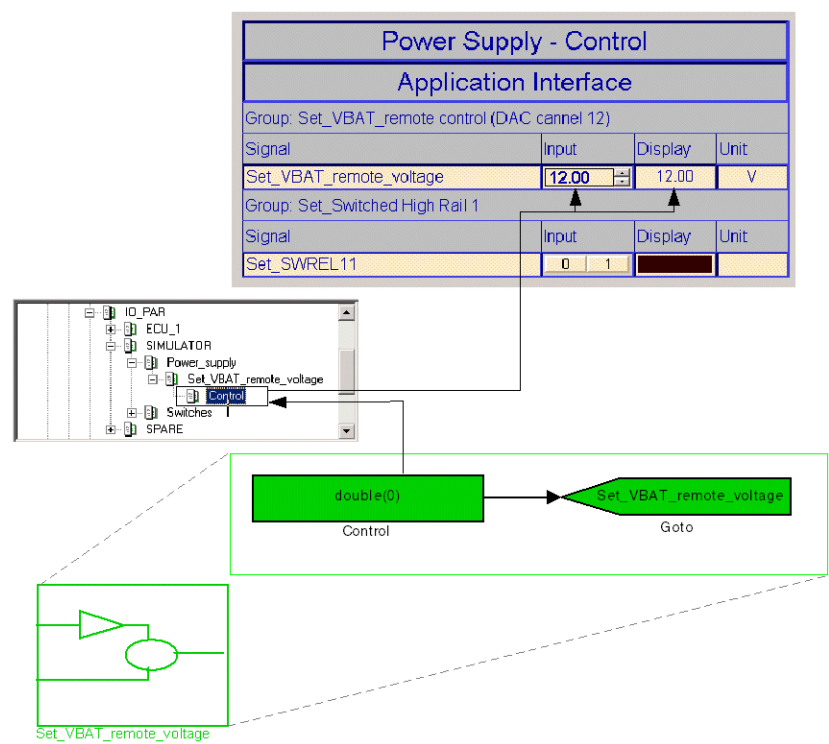
Data connections for read access

The following illustration shows the ControlDesk data connection to a Test Automation block (read access) in the IO_DISP subsystem. These data connections can be found inside actuator layouts for ECU_1, for example, and in all layouts with dSPACE input signals (see [ScalingFromHardware](#) on page 198).



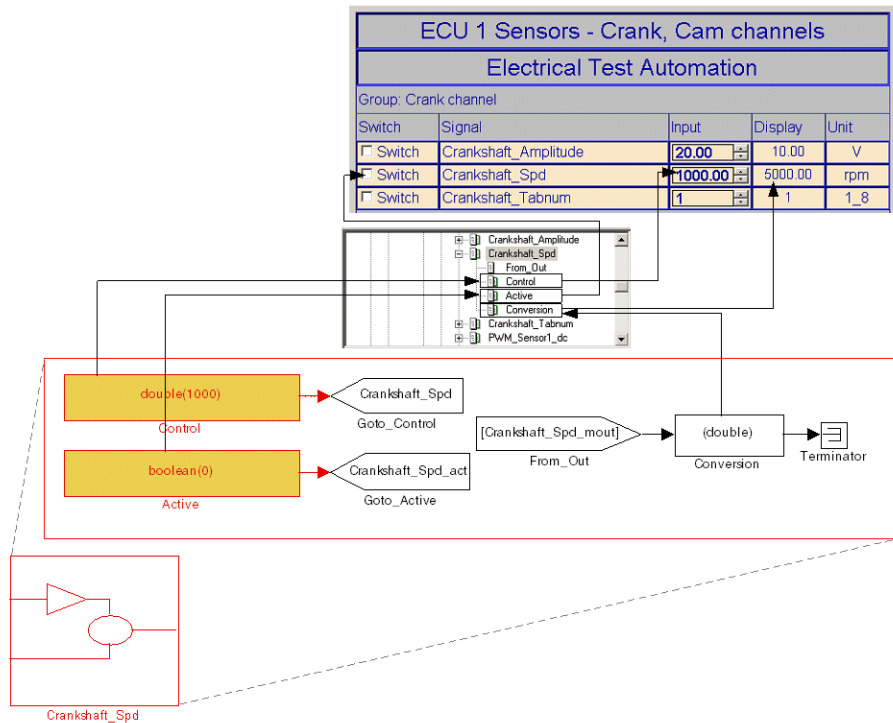
Data connections for write access

The following illustration shows how Test Automation blocks for write access are connected to layouts. The Control constant block holds the value of the signal, which can be adjusted by means of the Numeric Input instrument. This data connection can be found in sensor layouts of ECU_1, for example, and in all layouts with dSPACE output signals (refer to [ScalingToHardware](#) on page 192).



Data connection for signal manipulation

Test automation blocks for signal manipulation access are connected to the ControlDesk layouts in the following manner:



These data connections can be found in ECU_1 sensor layouts, for example, and in all layouts with dSPACE output signals (refer to [ScalingToHardware](#) on page 192 and [IOUserInterface Subsystem](#) on page 200). The checkbox instrument labeled "Switch" decides whether the signal is manipulated or not. The manipulation value can be set with the numeric instrument of the "Input" column.

ECU_1 Layouts

Introduction

The ECU_1 layouts provide access to all ECU signals. The layout file names (*.lay) for ECU_1 begin with ECU_1_ followed by a group indicator, **actuators** or **sensors**.

Sensors group

All layouts of the sensors group, for example, `ecu_1_sensors_crankcam.lay`, provide access to signals that flow from the physical model to the I/O model (see [ScalingToHardware](#) on page 192).

ECU 1 Sensors - Crank, Cam channels					
Application Interface			Electrical Test Automation		
Group: Set_Crank channel			Group: Crank channel		
Signal	Input	Display Unit	Switch	Signal	Input Display Unit
Set_Crankshaft_Amplitude	10.00	10.00 V	<input type="checkbox"/> Switch	Crankshaft_Amplitude	20.00 10.00 V
Set_Crankshaft_Spd	5000.00	5000.00 rpm	<input type="checkbox"/> Switch	Crankshaft_Spd	1000.00 5000.00 rpm
Set_Crankshaft_Tabnum	1	1 1_8	<input type="checkbox"/> Switch	Crankshaft_Tabnum	1 1 1_8
Group: Set_Cam 1 channel			Group: Cam 1 channel		
Signal	Input	Display Unit	Switch	Signal	Input Display Unit
Set_Camshaft1_Amp	40.00	40.00 V	<input checked="" type="checkbox"/> Switch	Camshaft1_Amp	40.00 40.00 V
Set_Camshaft1_Phase	0.00	0.00 deg	<input checked="" type="checkbox"/> Switch	Camshaft1_Phase	60.00 60.00 deg
Set_Camshaft1_Tab	1	1 1_8	<input checked="" type="checkbox"/> Switch	Camshaft1_Tab	1 1 1_8
Group: Set_Cams 2 channel			Group: Cam 2 channel		
Signal	Input	Display Unit	Switch	Signal	Input Display Unit
Set_Camshaft2_Amp	40.00	40.00 V	<input checked="" type="checkbox"/> Switch	Camshaft2_Amp	40.00 40.00 V
Set_Camshaft2_Phase	0.00	0.00 deg	<input checked="" type="checkbox"/> Switch	Camshaft2_Phase	0.00 0.00 deg
Set_Camshaft2_Tab	1	1 1_8	<input checked="" type="checkbox"/> Switch	Camshaft2_Tab	1 1 1_8

The Application Interface column on the left side of the example layout contains the data connections to Test Automation blocks with signal write access (refer to [ScalingToHardware](#) on page 192 and [Data Connections to ControlDesk Instruments](#) on page 204). The Application Interface represents the connection between physical model and IO subsystem.

The Electrical Test Automation column contains instruments that provide access to Test Automation blocks with signal manipulation capability (refer to [ScalingToHardware](#) on page 192 and [Data Connections to ControlDesk Instruments](#) on page 204). You can manipulate electrical ECU signals here.

Actuators group

Layouts of the actuators group, for example, `ecu_1_actuators_injection.lay`, provide access to signals that flow from dSPACE I/O hardware (DS2211) to the physical model (refer to [ScalingToHardware](#) on page 192).

ECU 1 Actuators - Injection channels					
Group: Injector channel 1			Group: Injector4		
Signal	Display	Unit	Signal	Display	Unit
Injector1_INJ_DU1	0.001	s	Injector4_INJ_DU1	0.000	s
Injector1_INJ_LE1	240.00	deg	Injector4_INJ_LE1	0.00	deg
Injector1_Pulses	28.00		Injector4_Pulses	0.00	
Group: Injector2			Group: Injector5		
Signal	Display	Unit	Signal	Display	Unit
Injector2_INJ_DU1	0.000	s	Injector5_INJ_DU1	0.000	s
Injector2_INJ_LE1	0.00	deg	Injector5_INJ_LE1	0.00	deg
Injector2_Pulses	0.00		Injector5_Pulses	0.00	
Group: Injector3			Group: Injector6		
Signal	Display	Unit	Signal	Display	Unit
Injector3_INJ_DU1	0.000	s	Injector6_INJ_DU1	0.000	s
Injector3_INJ_LE1	0.00	deg	Injector6_INJ_LE1	0.00	deg
Injector3_Pulses	0.00		Injector6_Pulses	0.00	

Here is a list of all ECU_1 related layouts:

- ecu_1_sensors_analog.lay
- ecu_1_sensors_crankcam.lay
- ecu_1_sensors_pwm.lay
- ecu_1_sensors_res.lay
- ecu_1_sensors_supply.lay
- ecu_1_sensors_switches.lay
- ecu_1_sensors_wheel.lay
- ecu_1_actuators_ignition.lay
- ecu_1_actuators_injection.lay
- ecu_1_actuators_pwm.lay
- ecu_1_actuators_switches.lay

SPARE Layouts

Introduction

The SPARE layouts provide access to all spare signals.

SPARE layouts

The design of the layouts is similar to the ECU_1 layouts. Layouts with signals that reside inside ScalingToHardware subsystem are:

- spare_dac.lay
- spare_dig_out.lay
- spare_pwm_out.lay
- spare_res_out.lay

Spare - Analogue output channels				
Application Interface				Electrical Test Automation
Group: Set_Spare_DAC Channels				Group: Spare_DAC Channels
Channel	Input	Display	Unit	Switch Channel Input Display Unit
Set_DAC_Channel3	5.70	5.70	V	<input checked="" type="checkbox"/> Switch DAC_Channel3 4.30 4.30 V
Set_DAC_Channel4	5.50	5.50	V	<input checked="" type="checkbox"/> Switch DAC_Channel4 4.70 4.70 V
Set_DAC_Channel5	4.60	4.60	V	<input checked="" type="checkbox"/> Switch DAC_Channel5 5.20 5.20 V
Set_DAC_Channel6	5.40	5.40	V	<input checked="" type="checkbox"/> Switch DAC_Channel6 5.40 5.40 V
Set_DAC_Channel7	4.80	4.80	V	<input checked="" type="checkbox"/> Switch DAC_Channel7 4.80 4.80 V
Set_DAC_Channel8	4.70	4.70	V	<input checked="" type="checkbox"/> Switch DAC_Channel8 4.80 4.80 V
Set_DAC_Channel9	5.30	5.30	V	<input checked="" type="checkbox"/> Switch DAC_Channel9 4.20 4.20 V
Set_DAC_Channel10	3.90	3.90	V	<input checked="" type="checkbox"/> Switch DAC_Channel10 5.20 5.20 V
Set_DAC_Channel11	5.40	5.40	V	<input checked="" type="checkbox"/> Switch DAC_Channel11 4.70 4.70 V
Set_DAC_Channel13	5.30	5.30	V	<input checked="" type="checkbox"/> Switch DAC_Channel13 4.20 4.20 V
Set_DAC_Channel14	4.50	4.50	V	<input checked="" type="checkbox"/> Switch DAC_Channel14 4.60 4.60 V
Set_DAC_Channel15	4.40	4.40	V	<input checked="" type="checkbox"/> Switch DAC_Channel15 5.40 5.40 V
Set_DAC_Channel16	4.60	4.60	V	<input checked="" type="checkbox"/> Switch DAC_Channel16 5.30 5.30 V
Set_DAC_Channel17	5.30	5.30	V	<input checked="" type="checkbox"/> Switch DAC_Channel17 4.70 4.70 V
Set_DAC_Channel18	5.20	5.20	V	<input checked="" type="checkbox"/> Switch DAC_Channel18 5.30 5.30 V
Set_DAC_Channel19	4.60	4.60	V	<input checked="" type="checkbox"/> Switch DAC_Channel19 5.30 5.30 V
Set_DAC_Channel20	5.30	5.30	V	<input checked="" type="checkbox"/> Switch DAC_Channel20 4.60 4.60 V

Layouts with signals that reside inside `ScalingFromHardware` subsystem are:

- `spare_adc.lay`
- `spare_dig_in.lay`
- `spare_pwm_in.lay`

Spare - Analogue input channels		
Group: ADC_Channels		
Channel	Display	Unit
ADC_Channel3	0.00	V
ADC_Channel4	0.00	V
ADC_Channel5	0.00	V
ADC_Channel6	0.00	V
ADC_Channel7	0.00	V
ADC_Channel8	0.00	V
ADC_Channel9	0.00	V
ADC_Channel10	0.00	V
ADC_Channel11	0.00	V
ADC_Channel12	0.00	V
ADC_Channel13	0.00	V
ADC_Channel14	0.00	V

Related topics

Basics	
ScalingFromHardware.....	198
ScalingToHardware.....	192

DS2211 Layouts

Introduction

The DS2211 layouts provide direct access to each I/O channel of the DS2211 board. They can be used to stimulate and test the simulator channels without an ECU connected at the front connectors. Direct access is provided by Test Automation blocks with signal manipulation access inside the MATLAB/Simulink model (refer to [HardwareInterface](#) on page 195).

Electrical Test Automation - DS2211 PWM out channels				
Group: DS2211_1_PWM_OUT_Channel Stimulus				
Switch	Signal	Input	Display	Unit
<input type="checkbox"/> Switch	DS2211_1_01_DC	10.00	10.00	%
<input type="checkbox"/> Switch	DS2211_1_01_f	1000.00	1000.00	Hz
<input type="checkbox"/> Switch	DS2211_1_02_DC	10.00	10.00	%
<input type="checkbox"/> Switch	DS2211_1_02_f	1000.00	1000.00	Hz
<input type="checkbox"/> Switch	DS2211_1_03_DC	10.00	10.00	%
<input type="checkbox"/> Switch	DS2211_1_03_f	1000.00	1000.00	Hz
<input type="checkbox"/> Switch	DS2211_1_04_DC	10.00	10.00	%
<input type="checkbox"/> Switch	DS2211_1_04_f	1000.00	1000.00	Hz
<input type="checkbox"/> Switch	DS2211_1_05_DC	10.00	10.00	%
<input type="checkbox"/> Switch	DS2211_1_05_f	1000.00	1000.00	Hz
<input type="checkbox"/> Switch	DS2211_1_06_DC	10.00	10.00	%
<input type="checkbox"/> Switch	DS2211_1_06_f	1000.00	1000.00	Hz
<input checked="" type="checkbox"/> Switch	DS2211_1_07_DC	64.60	64.60	%
<input checked="" type="checkbox"/> Switch	DS2211_1_07_f	1000.00	1000.00	Hz
<input type="checkbox"/> Switch	DS2211_1_08_DC	10.00	10.00	%
<input type="checkbox"/> Switch	DS2211_1_08_f	1000.00	1260.00	Hz
<input type="checkbox"/> Switch	DS2211_1_09_DC	10.00	10.00	%
<input type="checkbox"/> Switch	DS2211_1_09_f	1000.00	1000.00	Hz

The DS2211 layout files are:

- ds2211_adc.lay
- ds2211_crankcam.lay
- ds2211_dac.lay
- ds2211_dig_in.lay
- ds2211_dig_out.lay
- ds2211_ignition.lay
- ds2211_injection.lay
- ds2211_pwm_in.lay
- ds2211_pwm_out.lay
- ds2211_res.lay
- ds2211_wheelspeed.lay

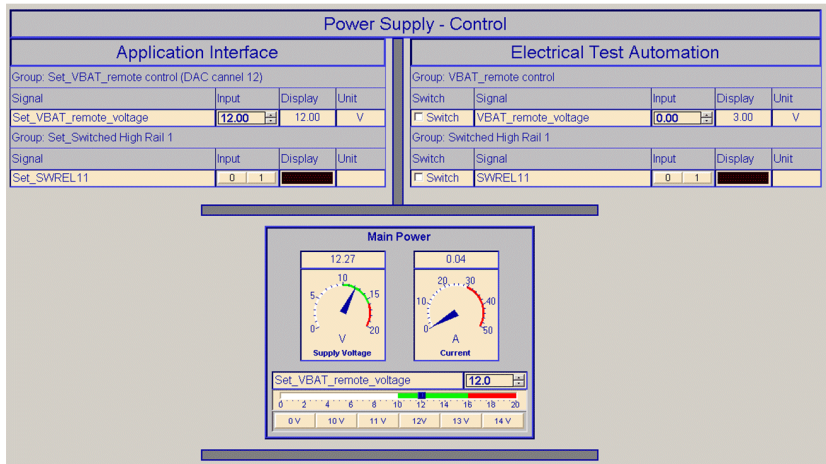
Additional Layouts

Introduction

There are additional layouts for controlling the power supply, controlling the simulation, accessing the complex comparator of the DS2211, and an example of a cockpit.

Power supply control

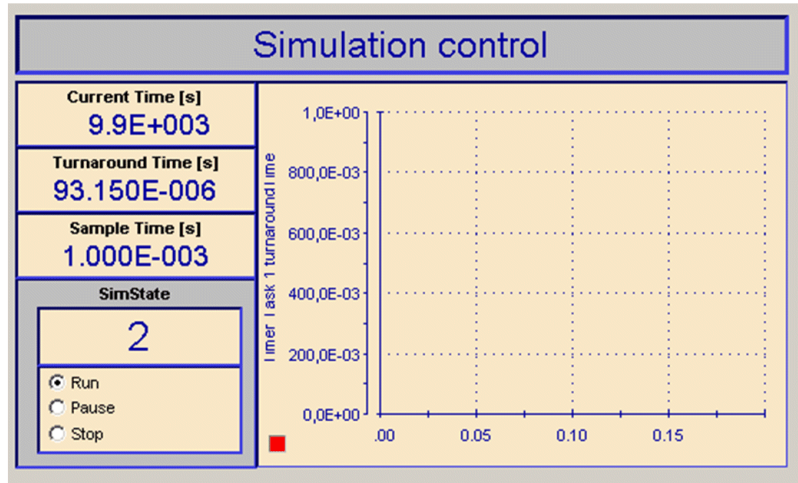
One additional layout is for controlling the power supply and the switched rail of the dSPACE Simulator Mid-Size.



Note that bridges between some ECU connectors of the simulator are necessary to control the switched rail (SWREL11), refer to [Demo Signal List](#) on page 184.

Sim_control layout

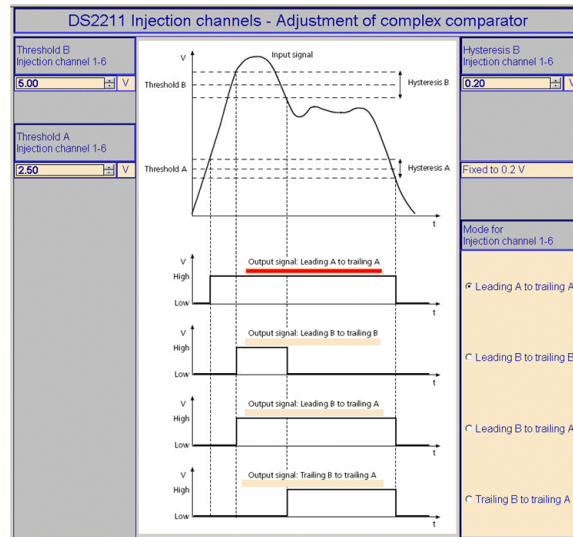
The sim_control layout demonstrates how to control and observe the real-time application running on the processor board.



The turnaround time is the time the processor board takes to do all the necessary calculations within one sample step.

ds2211_ignition_setCC and ds2211_injection_setCC layouts

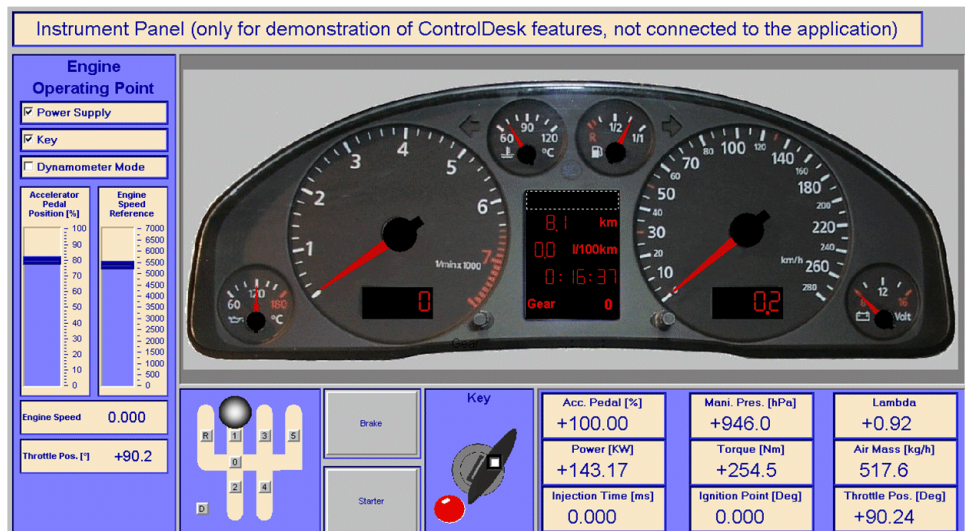
These layouts provide access to the complex comparator configuration. Complex comparators of the DS2211 I/O board are used to evaluate injection and ignition signals, etc. For details, refer to [Complex Comparators](#) on page 77.



The layouts provide a means of adjusting the mode of the complex comparator, the levels for thresholds A and B, and the hysteresis level of comparator B.

Cockpit layout

This layout (instr_panel.lay) is an example of a main layout of an automotive HIL simulator. The physical signals displayed are characteristic of an engine ECU.



Failure Simulation

Failure Simulation

Introduction

The ControlDesk experiment contains a failure simulation file (FIU Mid-Size DS22xx.fss). This is based on the generated CSV file of the signal list (DS22xx_SignalList.csv).

Failure simulation for simulator output signals as sensor or switch signals is possible only if the simulator was equipped with the optional DS793 Sensor FIU.

The failure simulation file contains five demo failure patterns in which some actuator signals are selected. For basic and advanced information about failure simulation on a dSPACE Simulator Mid-Size, refer to [Failure Simulation](#) on page 167.

Related topics	Basics
	Signal List of the Demo..... 178

Expandability of the dSPACE Simulator Mid-Size

Introduction

The following notes give you some ideas how you can expand dSPACE Simulator Mid-Size if it does not fulfill your requirements.

Where to go from here

Information in this section

[Using Additional I/O Boards or Signal Conditioning.....215](#)

You can expand the standard version of dSPACE Simulator Mid-Size.

[Expanding the Processing Power.....216](#)

When the model is too large to run on one processor board, you can use a multiprocessor system to enlarge the processing power.

[Expanding the I/O Hardware.....216](#)

The engine to be simulated has 10 or 12 cylinders but there are not enough I/O channels.

Using Additional I/O Boards or Signal Conditioning

Introduction

The standard version of dSPACE Simulator Mid-Size can be expanded:

- You can use additional I/O boards, for example, ECU Interface or CAN Interface boards. Refer to [Setting Up and Installing Further I/O Boards \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(6b2ce2ef0aa0acafe24dd5ed94556dce_img.jpg\)](#)).
- You can use additional or customer-specific signal conditioning (e.g., current measurement) or real loads. Refer to [Additional Signal Conditioning \(dSPACE Simulator Mid-Size Based on DS2211 Hardware Installation and Configuration !\[\]\(2277423912c64094fa85b84c0d40e3dd_img.jpg\)](#)).

Expanding the Processing Power

Large model

Problem The model is too large to run on one processor board and it is not possible to increment the sample rate.

Solution Use an additional processor board. Processor boards can be linked together to a multiprocessor system. The number of additional processor boards is limited by the number of spare slots, which in turns depends on the number and types of the I/O boards.

Expanding the I/O Hardware

10 or 12 cylinders

Problem The engine to be simulated has 10 or 12 cylinders.

Solution


- Connect two DS2211 via the engine position bus so that the Angular Processing Units run synchronously. If not all of the Load/FIU channels of the first DS2211 are used, the remaining channels can be used for the second DS2211. If a second Load/FIU unit is necessary, it can be included.
- If you simulate a diesel engine and ignition does not have to be captured, the ignition channels can be used for measuring the additional injection.

Alternative use of pins

Problem A few ECU pins are remaining, but the matching DS2211 channels are already in use.

Solution Map those ECU pins to alternative DS2211 pins. For example,

- Use analog inputs instead of digital inputs and detect threshold crossing by the model
- Use digital outputs instead of PWM outputs
- Use analog outputs instead of digital outputs (if the amplitude range matches the requirements)

Refer to the [DS2211 RTLib Reference](#)  to see how the hardware can be accessed.

Special crankshaft signal

Problem A special crankshaft signal simulation is required, which cannot be realized by using waveform tables with the DS2211 crank signal output.

Solution Use the slave DSP on the DS2211. Refer to [Features of the Slave DSP](#) on page 99.

Frequency range

Problem A frequency signal with a wide frequency range and a constant frequency resolution over the wide frequency range has to be generated.

Solution Use the slave DSP on the DSP2211. Refer to [Features of the Slave DSP](#) on page 99.

Limitations

Introduction	There are some limitations you have to take into account when working with dSPACE Simulator Mid-Size Based on the DS2211.
--------------	---

Where to go from here	Information in this section
	<div><div>Quantization Effects.....220</div><div>Quantization effects occur in signal generation or measurement.</div><div>DS2211 Board Revision.....220</div><div>Some features are only available for DS2211 boards with higher board revision.</div><div>Conflicting I/O Features.....221</div><div>Describing the types of I/O conflicts and lists the conflicted functional units.</div><div>Limited Number of CAN Messages.....235</div><div>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.</div><div>Limitations with RTICANMM.....237</div><div>There are a number of general limitations with RTICANMM.</div><div>Limitations with J1939-Support.....241</div><div>There are a number of limitations regarding the J1939 support of RTICANMM.</div></div>

Quantization Effects

Introduction

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

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

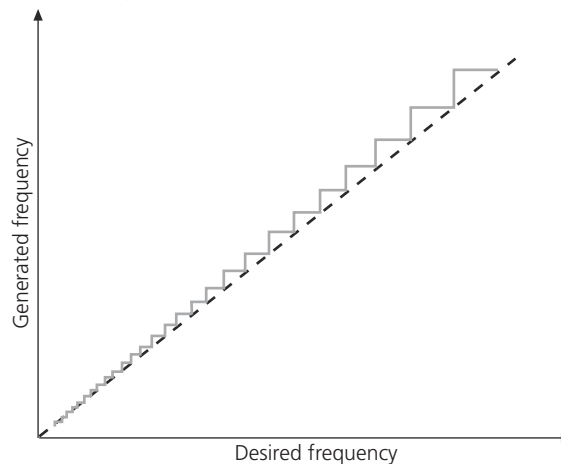
$$f = \frac{1}{n \cdot R}$$

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

Example

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

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



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

DS2211 Board Revision

Introduction

Several features are supported only for DS2211 boards with specific revisions and higher, for example, if you want to use the position and duration mode or

absolute mode for injection event capturing. The following board is extended in functionality:

- DS2211 boards with board revision 3 and FPGA revision 3 or higher (FPGA = field programmable gate array).

Revision numbers

The revision numbers are displayed with the following syntax:

<board>.<FPGA>

For example, 3.2 means you have a DS2211 board with the revision 3 and a FPGA revision number of 2 installed.

The revision number is printed on the board. You can also read out the number with ControlDesk, refer to [Board Details Properties \(ControlDesk Platform Management !\[\]\(830769b31eeeaca920791081939ff8ba_img.jpg\)](#)).

Conflicting I/O Features

Types of I/O conflicts

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

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

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

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

Conflicts for the DS2211





The following tables list the I/O features of the DS2211 that conflict with other I/O features, and the related RTI blocks/RTLib functions.

- Conflicts for the sensor and actuator interface
 - [Conflicts for Digital Inputs](#) on page 222
 - [Conflicts for Digital Outputs](#) on page 223
 - [Conflicts for PWM Signal Generation](#) on page 223
 - [Conflicts for Square-Wave Signal Generation \(D2F\)](#) on page 223
 - [Conflicts for PWM Signal Measurement \(PWM2D\)](#) on page 224
 - [Conflicts for Square-Wave Signal Measurement \(F2D\)](#) on page 225
 - [Conflicts for Wheel Speed Sensor Simulation](#) on page 226
 - [Conflicts for Camshaft Sensor Signal Generation](#) on page 226

- Conflicts for the angular processing unit (APU)
 - [Conflicts for Spark Event Capture \(Last Event Window Read\)](#) on page 227
 - [Conflicts for Spark Event Capture, Auxiliary Inputs \(Last Event Window Read\)](#) on page 228
 - [Conflicts for Spark Event Capture \(Continuous Read\)](#) on page 229
 - [Conflicts for Spark Event Capture, Auxiliary Inputs \(Continuous Read\)](#) on page 230
 - [Conflicts for Injection Pulse Position and Fuel Amount Measurement \(Last Event Window Read\)](#) on page 231
 - [Conflicts for Injection Pulse Position and Fuel Amount Measurement \(Continuous Read\)](#) on page 232
 - [Conflicts for Knock Sensor Simulation](#) on page 226
- Conflicts for communication channels
 - [Conflicts for Single Edge Nibble Transmission \(SENT\)](#) on page 234
 - [Conflicts for the Serial Interface](#) on page 235




Conflicts for Digital Inputs

The following I/O features of the DS2211 conflict with digital inputs provided by the sensor and actuator interface:

Digital Inputs *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	Ch (RTLib)
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 16	Ch 1 ... ch 16	DIG_IN1 ... DIG_IN16	<ul style="list-style-type: none">▪ Square-wave signal measurement (F2D)▪ PWM signal measurement (PWM2D)	Ch 9 ... ch 24	Ch 9 ... ch 24
Although DIG_IN channels 1 ... 16 are shared with PWM input channels 9 ... 24, they can be used at the same time.					
Ch 1 ... ch 4	Ch 1 ... ch 4	DIG_IN1 ... DIG_IN4	Single edge nibble transmission (SENT receiver)	Ch 1 ... ch 4	Ch 1 ... ch 4
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211BIT_IN_Bx_Cy DS2211BIT_IN16_Bx▪ Bit I/O Unit (DS2211 RTLib Reference 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Square-wave signal measurement<ul style="list-style-type: none">▪ DS2211F2D_Bx_Cy▪ Frequency Measurement (DS2211 RTLib Reference ▪ PWM signal measurement<ul style="list-style-type: none">▪ DS2211PWM2D_Bx_Cy▪ PWM Signal Measurement (DS2211 RTLib Reference ▪ Single edge nibble transmission (SENT receiver):<ul style="list-style-type: none">▪ DS2211SENT_RX_BLx▪ Single Edge Nibble Transmission (SENT) (DS2211 RTLib Reference 		



Conflicts for Digital Outputs

The following I/O features of the DS2211 conflict with digital outputs provided by the sensor and actuator interface:

Digital Outputs *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 15 and ch 16	Ch 15 and ch 16	DIG_OUT15 and DIG_OUT16	Camshaft sensor signal generation	Ch 3 and ch 4	Ch 3 and ch 4
Ch 1 ... ch 5	Ch 1 ... ch 5	DIG_OUT1 ... DIG_OUT5	Single edge nibble transmission	Ch 1 ... ch 5	Ch 1 ... ch 5
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211BIT_OUT_Bx_Cy DS2211BIT_OUT16_BxBit I/O Unit (DS2211 RTLib Reference 			**) Related RTI blocks and RTLib functions: Camshaft sensor signal generation <ul style="list-style-type: none">DS2211APU_CAM_Bx_CyCamshaft Sensor Signal Generation (DS2211 RTLib Reference  Single edge nibble transmission <ul style="list-style-type: none">DS2211SENT_TX_BLxSingle Edge Nibble Transmission (SENT) (DS2211 RTLib Reference 		

Conflicts for PWM Signal Generation

The following I/O features of the DS2211 conflict with PWM signal generation provided by the sensor and actuator interface:

PWM Signal Generation *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 9	Ch 1 ... ch 9	PWM_OUT1 ... PWM_OUT9	Square-wave signal generation (D2F)	Ch 1 ... ch 9	Ch 1 ... ch 9
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211PWM_Bx_CyPWM Signal Generation (DS2211 RTLib Reference 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211D2F_Bx_CySquare-Wave Signal Generation (DS2211 RTLib Reference 		

Conflicts for Square-Wave Signal Generation (D2F)

The following I/O features of the DS2211 conflict with square-wave signal generation provided by the sensor and actuator interface:

Square-Wave Signal Generation *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 9	Ch 1 ... ch 9	PWM_OUT1 ... PWM_OUT9	PWM signal generation	Ch 1 ... ch 9	Ch 1 ... ch 9

Square-Wave Signal Generation *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	Ch (RTLib)
Ch (RTI)	Ch (RTLib)				
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> DS2211D2F_Bx_Cy Square-Wave Signal Generation (DS2211 RTLib Reference [1]) 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> DS2211PWM_Bx_Cy PWM Signal Generation (DS2211 RTLib Reference [1]) 		






Conflicts for PWM Signal Measurement (PWM2D)

The following I/O features of the DS2211 conflict with PWM signal measurement provided by the sensor and actuator interface:

PWM Signal Measurement *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	Ch (RTLib)
Ch (RTI)	Ch (RTLib)				
Conflicts Concerning Single Channels					
Ch 1 ... ch 6	Ch 1 ... ch 6	PWM_IN1 ... PWM_IN6	Square-wave signal measurement (F2D)	Ch 1 ... ch 6	Ch 1 ... ch 6
Ch 7 and ch 8	Ch 7 and ch 8	PWM_IN7 and PWM_IN8	<ul style="list-style-type: none">Square-wave signal measurement (F2D)Injection pulse position and fuel amount measurement	<ul style="list-style-type: none">Ch 7 and ch 8Group 1, ch 7 and group 1, ch 8	<ul style="list-style-type: none">Ch 7 and ch 8Group 1, ch 7 and group 1 ,ch 8
Ch 9 ... ch 24	Ch 9 ... ch 24	PWM_IN9 ... PWM_IN24	<ul style="list-style-type: none">Square-wave signal measurement (F2D)Digital input	<ul style="list-style-type: none">Ch 9 ... ch 24Ch 1 ... ch 16	<ul style="list-style-type: none">Ch 9 ... ch 24Ch 1 ... ch 16
Ch 9 ... ch 12	Ch 9 ... ch 12	PWM_IN9 ... PWM_IN12	Single edge nibble transmission (SENT receiver)	Ch 1 ... ch 4	Ch 1 ... ch 4
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211PWM2D_Bx_CyPWM Signal Measurement (DS2211 RTLib Reference 📖)			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">Square-wave signal measurement (F2D):<ul style="list-style-type: none">DS2211F2D_Bx_CyFrequency Measurement (DS2211 RTLib Reference 📖)Injection pulse position and fuel amount measurement:<ul style="list-style-type: none">DS2211APU_INJ_Bx_GyDS2211APU_INJCONT_Bx_GyDS2211VARAPU_INJ_Bx_GyInjection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference 📖)Digital input:<ul style="list-style-type: none">DS2211BIT_IN16_BxDS2211BIT_IN_Bx_CyBit I/O Unit (DS2211 RTLib Reference 📖)Single edge nibble transmission (SENT receiver):<ul style="list-style-type: none">DS2211SENT_RX_BLxSingle Edge Nibble Transmission (SENT) (DS2211 RTLib Reference 📖)		



Conflicts for Square-Wave Signal Measurement (F2D)

The following I/O features of the DS2211 conflict with square-wave signal measurement provided by the sensor and actuator interface:

Square-Wave Signal Measurement *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 6	Ch 1 ... ch 6	PWM_IN1 ... PWM_IN6	PWM signal measurement (PWM2D)	Ch 1 ... ch 6	Ch 1 ... ch 6
Ch 7 and ch 8	Ch 7 and ch 8	PWM_IN7 and PWM_IN8	<ul style="list-style-type: none"> PWM signal measurement (PWM2D) Injection pulse position and fuel amount measurement 	<ul style="list-style-type: none"> Ch 7 and ch 8 Group 1, ch 7 and group 1, ch 8 	<ul style="list-style-type: none"> Ch 7 and ch 8 Group 1, ch 7 and group 1, ch 8
Ch 9 ... ch 24	Ch 9 ... ch 24	PWM_IN9 ... PWM_IN24	<ul style="list-style-type: none"> PWM signal measurement (PWM2D) Digital input 	<ul style="list-style-type: none"> Ch 9 ... ch 24 Ch 1 ... ch 16 	<ul style="list-style-type: none"> Ch 9 ... Ch 24 Ch 1 ... ch 16
Ch 9 ... ch 12	Ch 9 ... ch 12	PWM_IN9 ... PWM_IN12	Single edge nibble transmission (SENT receiver)	Ch 1 ... ch 4	Ch 1 ... ch 4
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> DS2211F2D_Bx_Cy Frequency Measurement (DS2211 RTLib Reference ) 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> PWM2D: <ul style="list-style-type: none"> DS2211PWM2D_Bx_Cy PWM Signal Measurement (DS2211 RTLib Reference ) Injection Pulse Position and Fuel Amount Measurement: <ul style="list-style-type: none"> DS2211APU_INJ_Bx_Gy DS2211APU_INJCONT_Bx_Gy DS2211VARAPU_INJ_Bx_Gy Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference ) Digital input: <ul style="list-style-type: none"> DS2211BIT_IN16_Bx DS2211BIT_IN_Bx_Cy Bit I/O Unit (DS2211 RTLib Reference ) Single edge nibble transmission (SENT receiver): <ul style="list-style-type: none"> DS2211SENT_RX_Bx Single Edge Nibble Transmission (SENT) (DS2211 RTLib Reference ) 		



Conflicts for Wheel Speed Sensor Simulation

The following I/O features of the DS2211 conflict with wheel speed sensor simulation provided by the sensor and actuator interface:

Wheel Speed Sensor Simulation *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	Ch (RTLib)				
Ch (RTI)	Ch (RTLib)								
Conflicts Concerning Wheel Speed Sensor Simulation as a Whole									
<ul style="list-style-type: none">If you perform wheel speed sensor simulation, you cannot perform knock sensor simulation at the same time.									
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SL_WSSG_Bx_CyWheel Speed Sensor Simulation (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SL_KNSG_Bx_CyDS2211VARSL_KNSG_Bx_CyKnock Sensor Simulation (DS2211 RTLib Reference )						



Conflicts for Camshaft Sensor Signal Generation

The following I/O features of the DS2211 conflict with wheel speed sensor simulation provided by the sensor and actuator interface:

Camshaft Sensor Signal Generation *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	Ch (RTLib)
Ch (RTI)	Ch (RTLib)				
Conflicts Concerning Single Channels					
Ch 3 and ch 4	Ch 3 and ch 4	CAM3_DIG and CAM4_DIG	Digital outputs	Ch 15 and ch 16	Ch 15 and ch 16
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211APU_CAM_Bx_CyCamshaft Sensor Signal Generation (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211BIT_OUT16_Bx_DS2211BIT_OUT_Bx_CyBit I/O Unit (DS2211 RTLib Reference )		

Conflicts for Knock Sensor Simulation

The following I/O features of the DS2211 conflict with knock sensor simulation provided by the angular processing unit:

Knock Sensor Simulation *)		Signal	Conflicting I/O Feature **)	Ch (RTI)	Ch(RTLib)
Ch (RTI)	Ch (RTLib)				
Conflicts Concerning Knock Sensor Simulation as a Whole					
<ul style="list-style-type: none">If you perform knock sensor simulation, you cannot perform wheel speed sensor simulation at the same time.					
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SL_KNSG_Bx_CyDS2211VARSL_KNSG_Bx_CyKnock Sensor Simulation (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SL_WSSG_Bx_CyWheel Speed Sensor Simulation (DS2211 RTLib Reference )		



Conflicts for Spark Event Capture (Last Event Window Read)

The following I/O features of the DS2211 conflict with spark event capture provided by the angular processing unit:

Spark Event Capture (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 6	Ch 1 ... ch 6	IGN1... IGN6	<ul style="list-style-type: none">▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Ch 1 ... ch 6▪ Group 2, ch 1 ... ch 6	<ul style="list-style-type: none">▪ –▪ Ch 1 ... ch 6
Ch 7 and ch 8	Ch 7 and ch 8	AUXCAP1 and AUXCAP2	<ul style="list-style-type: none">▪ Spark event capture, auxiliary input (continuous read and last event window read)▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Aux. ch 1 and Aux. ch 2▪ Ch 7 and ch 8▪ Group 2, ch 7 and group 2, ch 8	<ul style="list-style-type: none">▪ –▪ –▪ Ch 7 and ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211APU_IGN_Bx DS2211VARAPU_IGN_Bx▪ <i>Spark Event Capture (DS2211 RTLib Reference </i>)		**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Spark event capture (continuous read):<ul style="list-style-type: none">▪ DS2211APU_IGNCONT_Bx▪ Injection pulse position and fuel amount measurement (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_INJ_Bx_Gy DS2211APU_INJCONT_Bx_Gy(group 2) DS2211VARAPU_INJ_Bx_Gy▪ <i>Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference </i>)▪ Spark event capture, auxiliary input (continuous read and last event window read):<ul style="list-style-type: none">▪ DS2211APU_AUXCAPCONT_Bx_Cy DS2211APU_AUXCAP_Bx_Cy DS2211VARAPU_AUXCAP_Bx_Cy			



Conflicts for Spark Event Capture, Auxiliary Inputs (Last Event Window Read)

The following I/O features of the DS2211 conflict with spark event capture provided by the angular processing unit when using the auxiliary inputs:

Spark Event Capture (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Aux. ch 1	Aux. ch 1	AUXCAP1	<ul style="list-style-type: none">▪ Spark event capture, auxiliary input (continuous read)▪ Spark event capture (continuous read)▪ Spark event capture (last event window)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Aux. ch 1/ Aux. ch 2▪ All channels▪ Ch 7▪ Group 2, ch 7	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 7
Aux. ch 2	Aux. ch 2	AUXCAP2	<ul style="list-style-type: none">▪ Spark event capture, auxiliary input (continuous read)▪ Spark event capture (continuous read)▪ Spark event capture (last event window)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ Aux. ch 1/ Aux. ch 2▪ All channels▪ Ch 8▪ Group 2, ch 8	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211APU_AUXCAP_Bx_Cy▪ DS2211VARAPU_AUXCAP_Bx_Cy▪ <i>Spark Event Capture (DS2211 RTLib Reference</i> 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_IGN_Bx▪ DS2211APU_IGNCONT_Bx▪ DS2211VARAPU_IGN_Bx▪ Injection pulse position and fuel amount measurement (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_INJ_Bx_Gy▪ DS2211APU_INJCONT_Bx_Gy(group 2)▪ DS2211VARAPU_INJ_Bx_Gy▪ <i>Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference</i> ▪ Spark event capture, auxiliary input (continuous read):<ul style="list-style-type: none">▪ DS2211APU_AUXCAPCONT_Bx_Cy		



Conflicts for Spark Event Capture (Continuous Read) *

The following I/O features of the DS2211 conflict with spark event capture provided by the angular processing unit:

Spark Event Capture (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 6	Ch 1 ... ch 6	IGN1 ... IGN6	<ul style="list-style-type: none">Spark event capture (last event window read)Injection pulse position and fuel amount measurement (last event window and continuous read)Spark event capture, auxiliary input (last event window read)	<ul style="list-style-type: none">Ch 1 ... ch 6Group 2, ch 1 ... ch 6Aux. ch 1/ Aux. ch 2	<ul style="list-style-type: none">–Ch 1 ... ch 8–
Ch 7	Ch 7	AUXCAP1	<ul style="list-style-type: none">Spark event capture (last event window read)Spark event capture, auxiliary input (last event window read)Spark event capture (continuous read)Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">Ch 1 ... ch 8Aux. ch 1/ Aux. ch 2Ch 7Ch 7	<ul style="list-style-type: none">–––Ch 7
Ch 8	Ch 8	AUXCAP2	<ul style="list-style-type: none">Spark event capture (last event window read)Spark event capture, auxiliary input (last event window read)Spark event capture (continuous read)Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">Ch 1 ... ch 8Aux. ch 1/ Aux. ch 2Ch 8Ch 8	<ul style="list-style-type: none">–––Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211APU_IGNCONT_BxSpark Event Capture (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">Spark event capture (last event window read):<ul style="list-style-type: none">DS2211APU_IGN_BxDS2211VARAPU_IGN_BxSpark event capture, auxiliary input (last event window read):<ul style="list-style-type: none">DS2211APU_AUXCAP_Bx_CyDS2211VARAPU_AUXCAP_Bx_CyInjection pulse position and fuel amount measurement (last event window and continuous read):<ul style="list-style-type: none">DS2211APU_INJ_Bx_GyDS2211APU_INJCONT_Bx_Gy(group 2)DS2211VARAPU_INJ_Bx_GyInjection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference )		




Conflicts for Spark Event Capture, Auxiliary Inputs (Continuous Read)

The following I/O features of the DS2211 conflict with spark event capture provided by the angular processing unit when using the auxiliary inputs:

Spark Event Capture (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)				
Conflicts Concerning Single Channels					
Aux. ch 1	Aux. ch 1	AUXCAP1	<ul style="list-style-type: none">▪ Spark event capture (last event window read)▪ Spark event capture, auxiliary input (last event window read)▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ All channels▪ Aux. ch 1/ Aux. ch 2▪ Ch 7▪ Ch 7	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 7
Aux. ch 2	Aux. ch 2	AUXCAP2	<ul style="list-style-type: none">▪ Spark event capture (last event window read)▪ Spark event capture, auxiliary input (last event window read)▪ Spark event capture (continuous read)▪ Injection pulse position and fuel amount measurement (last event window and continuous read)	<ul style="list-style-type: none">▪ All channels▪ Aux. ch 1/ Aux. ch 2▪ Ch 8▪ Ch 8	<ul style="list-style-type: none">▪ –▪ –▪ –▪ Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211APU_AUXCAPCONT_Bx_Cy▪ <i>Spark Event Capture (DS2211 RTLib Reference</i> )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Spark event capture (last event window read):<ul style="list-style-type: none">▪ DS2211APU_IGN_Bx▪ DS2211VARAPU_IGN_Bx▪ Spark event capture, auxiliary input (last event window read):<ul style="list-style-type: none">▪ DS2211APU_AUXCAP_Bx_Cy▪ DS2211VARAPU_AUXCAP_Bx_Cy▪ Spark event capture (continuous read):<ul style="list-style-type: none">▪ DS2211APU_IGNCONT_Bx▪ Injection pulse position and fuel amount measurement (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_INJ_Bx_Gy▪ DS2211APU_INJCONT_Bx_Gy(group 2)▪ DS2211VARAPU_INJ_Bx_Gy▪ <i>Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference</i> )		

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

The following I/O features of the DS2211 conflict with injection pulse position (group 1) provided by the angular processing unit:

Injection Pulse Position and Fuel Amount Measurement (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)		Group (RTI)	Ch (RTLib)	
Conflicts Concerning Group 1					
Group 1, ch 1 ... ch 6	Ch 1 ... ch 6	INJ1 ... INJ6	Injection pulse position and fuel amount measurement (continuous read)	Group 1 (all channels)	–
Group 1, ch 7	Ch 7	INJ7	<ul style="list-style-type: none">Injection pulse position and fuel amount measurement (continuous read)Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D)	<ul style="list-style-type: none">Ch 7Ch 7	<ul style="list-style-type: none">Ch 7Ch 7
Group 1, ch 8	Ch 8	INJ8	<ul style="list-style-type: none">Injection pulse position and fuel amount measurement (continuous read)Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D)	<ul style="list-style-type: none">Ch 8Ch 8	<ul style="list-style-type: none">Ch 8Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211APU_INJ_Bx_Gy (group 1) DS2211VARAPU_INJ_Bx_Gy (group 1)Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">Injection pulse position and fuel amount measurement (continuous read)<ul style="list-style-type: none">DS2211APU_INJCONT_Bx_Gy (group 1)Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D):<ul style="list-style-type: none">DS2211PWM2D_Bx_Cy DS2211F2D_Bx_CyPWM Signal Measurement (DS2211 RTLib Reference ) Frequency Measurement (DS2211 RTLib Reference )		

The following I/O features of the DS2211 conflict with fuel amount measurement (group 2) provided by the angular processing unit:




Injection Pulse Position and Fuel Amount Measurement (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)			
Group (RTI)	Ch (RTLib)				Group (RTI)	Ch (RTLib)
Conflicts Concerning Group 2						
Group 2, ch 1 ... ch 6	Ch 1 ... ch 6	IGN1 ... IGN6	<ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)▪ Injection pulse position and fuel amount measurement (continuous read)		<ul style="list-style-type: none">▪ All channels▪ Group 2 (all channels)	<ul style="list-style-type: none">▪ All channels▪ –

Injection Pulse Position and Fuel Amount Measurement (Last Event Window Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)			Group (RTI)	Ch (RTLib)
Group 2, ch 7	Ch 7	AUXCAP1	<ul style="list-style-type: none"> Spark event capture (last event window and continuous read) Injection pulse position and fuel amount measurement (continuous read) Spark event capture, auxiliary input (last event window and continuous read) 	<ul style="list-style-type: none"> All channels Group 2 (all channels) Aux. ch 1 	<ul style="list-style-type: none"> All channels – Aux. ch 1
Group 2, ch 8	Ch 8	AUXCAP2	<ul style="list-style-type: none"> Spark event capture (last event window and continuous read) Injection pulse position and fuel amount measurement (continuous read) Spark event capture, auxiliary input (last event window and continuous read) 	<ul style="list-style-type: none"> All channels Group 2 (all channels) Aux. ch 2 	<ul style="list-style-type: none"> All channels – Aux. ch 2
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> DS2211APU_INJ_Bx_Gy (group 2) DS2211VARAPU_INJ_Bx_Gy (group 2) Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference) 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> Spark event capture (last event window and continuous read): <ul style="list-style-type: none"> DS2211APU_IGN_Bx DS2211APU_IGNCONT_Bx DS2211VARAPU_IGN_Bx Spark Event Capture (DS2211 RTLib Reference) Injection pulse position and fuel amount measurement (continuous read): <ul style="list-style-type: none"> DS2211APU_INJCONT_Bx_Gy (group 2) Spark event capture, auxiliary input (last event window and continuous read): <ul style="list-style-type: none"> DS2211APU_AUXCAP_Bx_Cy DS2211APU_AUXCAPCONT_Bx_Cy DS2211VARAPU_AUXCAP_Bx_Cy Spark Event Capture (DS2211 RTLib Reference) 		

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




The following I/O features of the DS2211 conflict with injection pulse position (group 1) provided by the angular processing unit:

Injection Pulse Position and Fuel Amount Measurement (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)			Group (RTI)	Ch (RTLib)
Conflicts Concerning Group 1					
Group 1, ch 1 ... ch 6	Ch 1 ... ch 6	INJ1 ... INJ6	Injection pulse position and fuel amount measurement (last event window read)	Group 1 (all channels)	—

Injection Pulse Position and Fuel Amount Measurement (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)			Group (RTI)	Ch (RTLib)
Group 1, ch 7	Ch 7	INJ7	<ul style="list-style-type: none"> Injection pulse position and fuel amount measurement (last event window read) Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D) 	<ul style="list-style-type: none"> Group 1 (all channels) Ch 7 	<ul style="list-style-type: none"> – Ch 7
Group 1, ch 8	Ch 8	INJ8	<ul style="list-style-type: none"> Injection pulse position and fuel amount measurement (last event window read) Square-wave signal measurement (F2D)/ PWM signal measurement (PWM2D) 	<ul style="list-style-type: none"> Group 1 (all channels) Ch 8 	<ul style="list-style-type: none"> – Ch 8
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> DS2211APU_INJCONT_Bx_Gy (group 1) Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference ) 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none"> Injection pulse position and fuel amount measurement (last event window read): <ul style="list-style-type: none"> DS2211APU_INJ_Bx_Gy (group 1) DS2211VARAPU_INJ_Bx_Gy (group 1) PWM2D/F2D: <ul style="list-style-type: none"> DS2211PWM2D_Bx_Cy DS2211F2D_Bx_Cy PWM Signal Measurement (DS2211 RTLib Reference )/ Frequency Measurement (DS2211 RTLib Reference ) 		



The following I/O features of the DS2211 conflict with fuel amount measurement (group 2) provided by the angular processing unit:

Injection Pulse Position and Fuel Amount Measurement (Continuous Read) *)		Signal	Conflicting I/O Feature **)		
Group (RTI)	Ch (RTLib)			Group (RTI)	Ch (RTLib)
Conflicts Concerning Group 2					
Group 2, ch 1 ... ch 6	Ch 1 ... ch 6	IGN1 ... IGN6	<ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)▪ Injection pulse position and fuel amount measurement (last event window read)	<ul style="list-style-type: none">▪ All channels▪ Group 2 (all channels)	<ul style="list-style-type: none">▪ All channels▪ –
Group 2, ch 7	Ch 7	AUXCAP1	<ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)▪ Injection pulse position and fuel amount measurement (last event window read)▪ Spark event capture, auxiliary input (last event window and continuous read)	<ul style="list-style-type: none">▪ All channels▪ Group 2 (all channels)▪ Aux. ch 1	<ul style="list-style-type: none">▪ All channels▪ –▪ Aux. ch 1

Injection Pulse Position and Fuel Amount Measurement (Continuous Read) *)		Signal	Conflicting I/O Feature **)	
Group (RTI)	Ch (RTLib)		Group (RTI)	Ch (RTLib)
Group 2, ch 8	Ch 8	AUXCAP2	<ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)▪ Injection pulse position and fuel amount measurement (last event window read)▪ Spark event capture, auxiliary input (last event window and continuous read)	<ul style="list-style-type: none">▪ All channels▪ Group 2 (all channels)▪ Aux. ch 2 <ul style="list-style-type: none">▪ All channels▪ –▪ Aux. ch 2
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211APU_INJCONT_Bx_Gy (group 2)▪ Injection Pulse Position and Fuel Amount Measurement (DS2211 RTLib Reference )			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ Spark event capture (last event window and continuous read)<ul style="list-style-type: none">▪ DS2211APU_IGN_Bx▪ DS2211APU_IGNCONT_Bx▪ DS2211VARAPU_IGN_Bx▪ Spark Event Capture (DS2211 RTLib Reference )▪ Injection pulse position and fuel amount measurement (last event window read)<ul style="list-style-type: none">▪ DS2211APU_INJ_Bx_Gy (group 2)/▪ DS2211VARAPU_INJ_Bx_Gy (group 2)▪ Spark event capture, auxiliary input (last event window and continuous read):<ul style="list-style-type: none">▪ DS2211APU_AUXCAP_Bx_Cy▪ DS2211APU_AUXCAPCONT_Bx_Cy▪ DS2211VARAPU_AUXCAP_Bx_Cy▪ Spark Event Capture (DS2211 RTLib Reference )	

Conflicts for Single Edge Nibble Transmission (SENT)

The following I/O features of the DS2211 conflict with a SENT transmitter:

SENT Transmitter *)		Signal	Conflicting I/O Feature **)		
Ch (RTI)	Ch (RTLib)			Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels					
Ch 1 ... ch 5	Ch 1 ... ch 5	DIG_OUT1 ... DIG_OUT5	Digital output	Ch 1 ... ch 5	Ch 1 ... ch 5
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211SENT_TX_BLx▪ Single Edge Nibble Transmission (SENT) (DS2211 RTLib Reference 			**) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">▪ DS2211BIT_OUT_Bx_Cy▪ DS2211BIT_OUT16_Bx▪ Bit I/O Unit (DS2211 RTLib Reference 		

The following I/O features of the DS2211 conflict with a SENT receiver:

SENT Receiver *)		Signal	Conflicting I/O Feature **)	
Ch (RTI)	Ch (RTLib)		Ch (RTI)	Ch (RTLib)
Conflicts Concerning Single Channels				
Ch 1 ... ch 4	Ch 1 ... ch 4	DIG_IN1 ... DIG_IN4	<ul style="list-style-type: none">Digital inputsSquare-wave signal measurement (F2D)PWM signal measurement (PWM2D)	<ul style="list-style-type: none">Ch 1 ... ch 4Ch 9 ... ch 12Ch 9 ... ch 12
*) Related RTI blocks and RTLib functions: <ul style="list-style-type: none">DS2211SENT_RX_BLxSingle Edge Nibble Transmission (SENT) (DS2211 RTLib Reference 📖)			**) Related RTI blocks and RTLib functions: Digital inputs <ul style="list-style-type: none">DS2211BIT_IN_Bx_Cy DS2211BIT_IN16_BxBit I/O Unit (DS2211 RTLib Reference 📖) Square-wave signal measurement (F2D) <ul style="list-style-type: none">DS2211F2D_Bx_CyFrequency Measurement (DS2211 RTLib Reference 📖) PWM signal measurement (PWM2D) <ul style="list-style-type: none">DS2211PWM2D_Bx_CyPWM Signal Measurement (DS2211 RTLib Reference 📖)	

Conflicts for the Serial Interface

The DS2211 supports only one serial interface. It can be configured to either RS232 or RS422 mode.

Limited Number of CAN Messages

Limitation

When you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions, the number of CAN messages in an application is limited.

This applies to the following message types:

- Transmit (TX) messages
- Receive (RX) messages
- Request (RQ) messages

An RQ message and the corresponding RX message are interpreted as a single (RQ) message. You cannot enable RX service support for the corresponding RX message.

- Remote (RM) messages

The sum of these messages is n_{sum} :

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

Maximum number of CAN messages

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

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

n_{max} in one application depends on:

- Whether you implement CAN communication with RTI CAN Blockset or with RTLib's CAN access functions.
- Whether you use RX service support.

The maximum number of CAN messages n_{max} is listed in the table below:

Platform	n_{max} with RTLib	n_{max} with RTI CAN Blockset							
		RX Service Support Disabled				RX Service Support Enabled			
		1 ¹⁾	2 ¹⁾	3 ¹⁾	4 ¹⁾	1 ¹⁾	2 ¹⁾	3 ¹⁾	4 ¹⁾
DS2202 (2 CAN controllers)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
DS2210 (2 CAN controllers)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
DS2211 (2 CAN controllers)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
MicroAutoBox II ³⁾ (2 CAN controllers per CAN_Type1)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
MicroLabBox (2 CAN controllers)	100	98	96	–	–	96 ²⁾	92 ²⁾	–	–
DS4302 (4 CAN controllers)	200	198	196	194	192	196 ²⁾	192 ²⁾	188 ²⁾	184 ²⁾

¹⁾ Number of CAN controllers used in the application

²⁾ It is assumed that RX service support is enabled for all the CAN controllers used, and that both CAN message identifier formats (STD, XTD) are used.


³⁾ Depending on the variant, the MicroAutoBox II contains up to three CAN_Type1 modules, each with 2 CAN controllers. The values in the list apply to a single CAN_Type1 module.

Ways to implement more CAN messages

There are two ways to implement more CAN messages in an application.

Using RX service support If you use RTI CAN Blockset's RX service support, the number of receive (RX) messages n_{RX} in the equations above applies only to RTICAN Receive (RX) blocks for which RX service support is not enabled.

The number of RTICAN Receive (RX) blocks for which RX service support is enabled is unlimited. Refer to [Using RX Service Support](#) on page 119.

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 26
 - I/O mapping 27
- analog/digital converter 26
- angle position interrupt 58
- angular processing unit
 - overview 57
- angular processing unit - variant 81

B

- battery voltage
 - controlling 156
- bit I/O unit 33
 - I/O mapping 34
- block diagram of simulator 13

C

- camshaft phase 65
- camshaft sensor signal generation 90
 - I/O mapping 91
- camshaft signal generator 65
- CAN
 - channel 110
 - fault-tolerant transceiver 114
 - interrupts 118
 - physical layer 112
 - service 121
 - setup 110
 - status information 121
- CAN FD 129
- CAN subsystem 20
- CAN support 109
- CANalyzer connector 176
- car battery 154
- CARB connector 175
- cascading I/O boards 60
- Common Program Data folder 10
- comparing transceiver modes 50
- complex comparator 77
- conflicting I/O features 221
- connecting time-base 60
- continuous capture 72
- continuous capture mode 67
- controlling
 - high rails 158
- crankshaft sensor signal generation 88
 - I/O mapping 90
- crankshaft signal generator 61
- CSV file for electrical error simulation 183

D

- D/A channel 105
- D/R converter 31
 - I/O mapping 33
- DAC unit 29
 - I/O mapping 30
- data file support 117
- defining CAN messages 117

- demo 177
- demo layouts
 - actuator group 207
 - cockpit 212
 - DS2211 signals 209
 - ds2211_ignition_setCC 211
 - ds2211_injection_setCC 211
 - power supply control 211
 - sensors group 206
 - sim_control 211
 - spare signals 208
- diagnostic
 - CARB connector 175
- digital/analog converter 29
- digital/resistance converter 31
- Documents folder 10
- DS1006
 - block diagram 15
 - feature overview 16
- DS2211
 - feature overview 22
 - master board 61
 - overview 19
 - slave board 61
- DS2211 board revision 220
- DS2211 interrupts
 - basics 149
- DS793 Sensor FIU 171
- DSP DAC unit 105
 - I/O mapping 105
- DSP subsystem 20
- duty cycle
 - measuring 36

E

- engine position bus 58
- engine position interrupt 58
- engine position phase accumulator 59
- event capture window 67, 72
- expanding the simulator 215

F

- failure insertion 167
- feature overview
 - DS1006 16
 - DS2211 22
- FIU 169
- frequency measurement 43
- fuel amount measurement 68
 - I/O mapping 96

H

- high rails 154
 - controlling 158
- hysteresis 77

I

- I/O mapping
 - ADC unit 27

- bit I/O unit 34
- camshaft sensor signal generation 91
- crankshaft sensor signal generation 90
- D/R converter 33
- DAC unit 30, 105
- frequency measurement 45
- fuel amount measurement 96
- knock sensor simulation 103
- PWM signal generation 43
- PWM signal measurement 39
- spark event measurement 94
- square-wave signal generation 47
- wheel speed sensor simulation 104
- ignition pulses 67
- injection event capture unit 68
- injection pulse position measurement 95
 - continuous capture 76
 - I/O mapping 96
- input FIU 171
- interrupts 149
 - basics 149
- IO subsystem 190
- IOUserInterface subsystem 200
- ISO11898 113
- ISO11898-6 114

J

- J1939
 - broadcast/peer-to-peer messages 134
 - limitations 241
 - working with J1939-compliant DBC files 134

K

- knock sensor simulation 101
 - I/O mapping 103

L

- limitations
 - J1939 241
 - RTI CAN MultiMessage Blockset 237
- load
 - real 163
 - substitute 163
- load simulation 163
- Local Program Data folder 10

M

- MappingFromHardware subsystem 197
- master DS2211 61
- MDLUserInterface subsystem 189
- measuring duty cycle 36
- messages
 - defining CAN messages 117
 - delay time 117
 - multiple 117
- multiple data files 117
- multiple event capture mode 68
- multiple message support 117

O

overview
 dSPACE Simulator Mid-Size 12

P

PC interface 14
 phase accumulator 59
 PHS++ bus 19
 piggyback support 115
 power supply
 I/O mapping 155
 power supply unit 154
 processor board 15
 PWM analysis 36
 PWM signal generation 40
 I/O mapping 43
 PWM signal measurement 36
 I/O mapping 39

Q

quantization effects 220

R

real load 163
 receive buffer 53
 resistor channel
 I/O mapping 33
 resistor output circuits 31
 reverse cranking 62
 RS485 114
 RTI CAN MultiMessage Blockset
 limitations 237
 supported platforms 124
 RTICANMM
 J1939 241
 RX service support 119
 RX SW FIFO 53

S

SAI 25
 ScalingFromHardware subsystem 198
 sensor and actuator interface 25
 serial interface 49
 baud rates 52
 cable length/baud rate (RS232) 51
 cable length/baud rate (RS422) 51
 oscillator frequency 52
 RS232 transceiver mode 50
 RS422 networks 51
 RS422 topologies 51
 RS422 transceiver mode 51
 signal capture mode 78
 signal list
 template 182
 simulate
 failures 167
 loads 163
 simulating

 car battery 154
 engine variants 81
 single event capture mode 67
 slave DS2211 61
 slave DSP
 basics 100
 slave DSP TMS320C31 100
 spark event
 continuous capture 76
 spark event capture unit 67
 spark event measurement 93
 I/O mapping 94
 square-wave signal generation 46
 I/O mapping 47
 start position/fuel amount capture mode 70
 substitute load 163
 switching
 high rails 158

T

TDC 94
 threshold voltage 77
 time-base bus 58
 time-base connector 60
 time-base master 60
 time-base slave 60
 TJA1041
 limitations 115
 TJA1054
 transceiver 115
 top dead center (TDC) 94
 transceiver
 TJA1054 115
 transmit buffer 53
 TX SW FIFO 53

U

UART 49
 using VAR APU blockset 83

V

VAR APU blockset 81
 differences to APU blockset 83
 example of capturing multiple event 85
 related RTI blocks 82
 related RTLib functions 82
 using 83

W

wave table
 basics 92
 generation 92
 wheel speed sensor simulation 103
 I/O mapping 104
 working with CAN FD 129