

DS2202 HIL I/O Board

RTLib Reference

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.

© 2005 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

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

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

Contents

About This Document	7
Initialization and Setup Functions	9
ds2202_init.....	10
ds2202_digin_threshold_set.....	11
ds2202_digout_mode_set.....	12
ds2202_digout_ls_write.....	14
ds2202_digout_ls_set.....	15
ds2202_digout_ls_clear.....	16
ds2202_digout_hs_vbat1_write.....	17
ds2202_digout_hs_vbat1_set.....	19
ds2202_digout_hs_vbat1_clear.....	20
ds2202_digout_hs_vbat2_write.....	21
ds2202_digout_hs_vbat2_set.....	22
ds2202_digout_hs_vbat2_clear.....	24
Multi I/O Interface	27
ADC Unit.....	29
Basics on Accessing the ADC Unit via RTLib.....	30
Example of Fast Read and Standard Read.....	30
Example of Single Read.....	31
Example of Multiple Read.....	33
ds2202_adc_start.....	34
ds2202_adc_block_in_fast.....	35
ds2202_adc_single_in.....	36
ds2202_adc_block_init.....	38
ds2202_adc_block_start.....	39
ds2202_adc_block_in.....	40
DAC Unit.....	42
Example of the DAC Unit.....	42
ds2202_dac_out.....	43
Bit I/O Unit.....	45
Example of Using the Bit I/O Unit.....	46
ds2202_bit_io_in.....	47
ds2202_bit_io_out.....	49

ds2202_bit_io_set.....	50
ds2202_bit_io_clear.....	51
Timing Mode.....	53
ds2202_timing_out_mode_set.....	53
ds2202_timing_in_mode_set.....	56
PWM Signal Generation.....	59
Example of PWM Signal Generation.....	59
ds2202_pwm_out.....	61
PWM Signal Measurement.....	63
Example of PWM Signal Measurement.....	63
ds2202_pwm_in.....	65
Square-Wave Signal Generation.....	67
Example of Square-Wave Signal Generation.....	67
ds2202_d2f.....	69
Frequency Measurement.....	71
Example of Frequency Measurement.....	71
ds2202_f2d.....	73

Serial Interface Communication 75

Basic Principles of Serial Communication.....	76
Trigger Levels.....	76
How to Handle Subinterrupts in Serial Communication.....	77
How to Write Portable Applications.....	78
Example of Serial Interface Communication.....	79
Data Types for Serial Communication.....	81
ds2202_ISR.....	81
ds2202_LSR.....	83
ds2202_MSR.....	84
ds2202_subint_handler_t.....	85
ds2202Channel.....	87
Generic Serial Interface Communication Functions.....	89
ds2202_init.....	90
ds2202_config.....	91
ds2202_transmit.....	94
ds2202_receive.....	95
ds2202_receive_term.....	97
ds2202_fifo_reset.....	98
ds2202_enable.....	99
ds2202_disable.....	100

ds2202_error_read.....	101
ds2202_transmit_fifo_level.....	102
ds2202_receive_fifo_level.....	103
ds2202_status_read.....	103
ds2202_set.....	104
ds2202_subint_handler_inst.....	106
ds2202_subint_enable.....	107
ds2202_subint_disable.....	108
ds2202_word2bytes.....	109
ds2202_bytes2word.....	111

Slave CAN Access Functions 113

Basics on Slave CAN Access Functions.....	115
Basics on the CAN Subsystem.....	115
Basic Communication Principles.....	116
CAN Error Message Types.....	117
Data Structures for CAN.....	118
ds2202_canChannel.....	118
ds2202_canService.....	119
ds2202_canMsg.....	123
Initialization.....	128
ds2202_can_communication_init.....	128
CAN Channel Handling.....	130
ds2202_can_channel_init.....	130
ds2202_can_channel_init_advanced.....	133
ds2202_can_channel_start.....	136
ds2202_can_channel_all_sleep.....	137
ds2202_can_channel_all_wakeup.....	138
ds2202_can_channel_BOff_go.....	139
ds2202_can_channel_BOff_return.....	140
ds2202_can_channel_set.....	141
ds2202_can_channel_txqueue_clear.....	143
CAN Message Handling.....	145
ds2202_can_msg_tx_register.....	146
ds2202_can_msg_rx_register.....	150
ds2202_can_msg_rqt_x_register.....	154
ds2202_can_msg_rqr_x_register.....	157
ds2202_can_msg_rm_register.....	160
ds2202_can_msg_set.....	164
ds2202_can_msg_rqt_x_activate.....	167

ds2202_can_msg_write.....	168
ds2202_can_msg_send.....	169
ds2202_can_msg_send_id.....	171
ds2202_can_msg_queue_level.....	172
ds2202_can_msg_txqueue_init.....	173
ds2202_can_msg_send_id_queued.....	175
ds2202_can_msg_txqueue_level_read.....	177
ds2202_can_msg_sleep.....	178
ds2202_can_msg_wakeup.....	179
ds2202_can_msg_read.....	180
ds2202_can_msg_trigger.....	182
ds2202_can_msg_clear.....	183
ds2202_can_msg_processed_register.....	184
ds2202_can_msg_processed_request.....	185
ds2202_can_msg_processed_read.....	186
CAN Service Functions.....	188
ds2202_can_service_register.....	188
ds2202_can_service_request.....	190
ds2202_can_service_read.....	191
CAN Subinterrupt Handling.....	193
Defining a Callback Function.....	193
ds2202_can_subint_handler_install.....	194
Utilities.....	196
ds2202_can_all_data_clear.....	196
ds2202_can_error_read.....	197
Examples of Using CAN.....	198
Example of Handling Transmit and Receive Messages.....	198
Example of Handling Request and Remote Messages.....	200
Example of Using Subinterrupts.....	202
Example of Using Service Functions.....	204
Example of Receiving Different Message IDs.....	205

Function Execution Times 207

Information on the Test Environment.....	207
Measured Execution Times.....	208









Index 211

About This Document

Content This reference provides a description of the C functions and macros you need to program the DS2202 HIL I/O Board.

Symbols

dSPACE user documentation uses the following symbols:

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files


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

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

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

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

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.

Initialization and Setup Functions

Introduction

Before you can use the DS2202 you have to perform an initialization process and set the working modes of some I/O units.

Where to go from here

Information in this section

ds2202_init	10
To initialize the DS2202 board.	
ds2202_digin_threshold_set	11
To set the threshold level for the digital inputs DIG_IN and PWM_IN of the DS2202 board channelwise.	
ds2202_digout_mode_set	12
To enable or disable the digital outputs.	
ds2202_digout_ls_write	14
To enable or disable the low-side switch of all digital outputs.	
ds2202_digout_ls_set	15
To enable the low-side switch of individual digital outputs.	
ds2202_digout_ls_clear	16
To disable the low-side switch of individual digital outputs.	
ds2202_digout_hs_vbat1_write	17
To enable or disable the high-side switch to VBAT1 of all digital outputs.	
ds2202_digout_hs_vbat1_set	19
To enable the high-side switch to VBAT1 of individual digital outputs.	
ds2202_digout_hs_vbat1_clear	20
To disable the high-side switch to VBAT1 of individual digital outputs.	
ds2202_digout_hs_vbat2_write	21
To enable and disable the high-side switch to VBAT2 of all digital outputs.	

ds2202_digout_hs_vbat2_set	22
To enable the high-side switch to VBAT2 of individual digital outputs.	
ds2202_digout_hs_vbat2_clear	24
To disable the high-side switch to VBAT2 of individual digital outputs.	

ds2202_init

Syntax

```
void ds2202_init(Int32 base)
```

Purpose

To perform the basic initialization of the DS2202 board.

Description

This function must be executed at the beginning of each application and initializes the board as follows:

- Checks for board presence.
- Resets all board functions to their default settings (writes `0xA5000000` to the ID register).

Function	Default Setting
Digital inputs	Sets 2.5 V threshold voltage
Digital outputs	Disabled
	Enables all low-side switches
	Enables all high-side switches to VBAT1
	Disables all high-side switches to VBAT2
Analog outputs (DAC)	Sets 0 V output voltage
Slave MC CAN-80C167	Reset

- Depending on the `DEBUG_INIT` global debug status flag, the function outputs an info message indicating the completion of the initialization.

Refer to:

- [Initialization \(DS1006 RTLib Reference !\[\]\(adb0331d22f78481623cc605df40612a_img.jpg\)](#)) or
- [Initialization \(DS1007 RTLib Reference !\[\]\(7e3a264c08e10137510d1aa76522412b_img.jpg\)](#))

Note

The `ds2202_init` function avoids multiple execution of the initialization code.

Parameters

base PHS-bus base address of the DS2202 board

Return value None

Messages

The following message is defined:

Type	Message	Meaning
Error	ds2202_init(board_offset): Board not found!	There is no DS2202 at the given board index (offset of the PHS-bus address).

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

[Initialization \(DS1006 RTLib Reference !\[\]\(5361750c22c4e047a52f4eac1ec2d4cc_img.jpg\)](#))
[Initialization \(DS1007 RTLib Reference !\[\]\(f276343e5e0d2402c20fdc9e8443c0dd_img.jpg\)](#))

ds2202_digin_threshold_set

Syntax

```
void ds2202_digin_threshold_set(
    Int32 base,
    Int32 channel,
    dsfloat value)
```

Include file

ds2202.h

Purpose

To set the threshold level for the digital inputs DIG_IN and PWM_IN of the DS2202 board channelwise.

Description

The threshold level for the digital inputs DIG_IN 1 ... 16 and for PWM_IN 1 ... 8 can be set channelwise. The threshold level for the digital inputs DIG_IN 25 ... 32 is set groupwise for all 8 inputs. The threshold level for the digital inputs DIG_IN 33 ... 38 is set groupwise for all 6 inputs.

For further information, refer to [Digital Inputs \(PHS Bus System Hardware Reference !\[\]\(21199eb166cc97331a0c54c649195dcc_img.jpg\)](#)).

Note

After initialization, the threshold of the digital input pins is 2.5 V.

Parameters

base PHS-bus base address of the DS2202 board

channel Channel which threshold level is to be set. The following symbols are predefined

Predefined Symbol	Description
DS2202_THRESH_DIGIN1 ... DS2202_THRESH_DIGIN38	Digital inputs 1 ... 38
DS2202_THRESH_PWMIN1 ... DS2202_THRESH_PWMIN8	PWM signal inputs 1 ... 8 ¹⁾
DS2202_THRESH_PWMIN9 ... DS2202_THRESH_PWMIN24	PWM signal inputs 9 ... 24 ²⁾

¹⁾ Shared with digital inputs 17 ... 24

²⁾ Shared with digital inputs 1 ... 16

value Threshold level within the range 1 ... 23.8 V. You can set the value in steps of 100 mV.

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

[ds2202_init](#) 10

ds2202_digout_mode_set

Syntax

```
void ds2202_digout_mode_set(
    Int32 base,
    Int32 mode)
```

Include file

ds2202.h

Purpose To enable or disable the digital outputs.

Description Writes the specified mode of the digital output circuits to the setup register of the DS2202. The setting refers to the bit I/O unit and PWM signal generation. For information on the digital outputs, refer to [Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(2e897e890e69d81eae4503a8342c36b0_img.jpg\)](#)).

To set an external voltage (VBAT1 or VBAT2) for a digital output you have to perform the following steps:

1. Call `ds2202_digout_mode_set` to enable all digital outputs.
2. Call `ds2202_digout_hs_vbat1_set` or `ds2202_digout_hs_vbat2_set` to select a supply rail for the respective digital output.
3. Invoke `ds2202_digout_ls_set` to ensure push-pull-driver functionality (thus enabling the corresponding low-side switch).

Parameters

base PHS-bus base address of the DS2202 board

mode Enables or disables the digital output circuits. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_DIGOUT_ENABLE	Enables the digital output circuits.
DS2202_DIGOUT_DISABLE	Disables the digital output circuits.


Note

If the digital output drivers are disabled, writing to or reading from the digital I/O ports has no effect.

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics **References**

Digital Outputs (PHS Bus System Hardware Reference 	
ds2202_digout_hs_vbat1_set	19
ds2202_digout_hs_vbat2_set	22
ds2202_digout_ls_set	15
ds2202_init	10

ds2202_digout_ls_write

Syntax

```
void ds2202_digout_ls_write(  
    Int32 base,  
    Int32 channel)
```

Include file

ds2202.h

Purpose

To enable or disable the low-side switch of all digital outputs.

Description

This function affects all digital outputs and resets the output bits that are not explicitly set. Use `ds2202_digout_ls_set` to set individual digital outputs without affecting other outputs.

Note

If the digital output drivers are disabled, writing or reading to the I/O port has no effect.

Parameters

base PHS-bus base address of the DS2202 board

channel Defines the channels whose low-side switch is to be enabled or disabled in the range 0x00000000 ... 0xFFFFFFFF. 1 enables the low-side switch, 0 disables the low-side switch. You can use the following predefined symbols. To enable more than one digital output, you must specify a list of predefined symbols combined by the logical operator OR:

Predefined Symbol	Bit	Signal
DS2202_OUTSTP_DIGOUT1 ... DS2202_OUTSTP_DIGOUT16	Bit 0 ... Bit 15	DIG_OUTx
DS2202_OUTSTP_PWMOUT1 ... DS2202_OUTSTP_PWMOUT9	Bit 16 ... Bit 24	PWM_OUTx

Default value

All low-side switches are enabled after reset.

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

ds2202_digout_ls_clear	16
ds2202_digout_ls_set	15
ds2202_digout_mode_set	12
ds2202_init	10

ds2202_digout_ls_set

Syntax

```
void ds2202_digout_ls_set(
    Int32 base,
    Int32 channel)
```

Include file

ds2202.h

Purpose

To enable the low-side switch of individual digital outputs.

Description

This function enables only the digital outputs specified by the **channel** parameter. Use **ds2202_digout_ls_write** to set individual digital outputs and to reset all other digital outputs.

To use the digital outputs, you must additionally enable their output drivers using the function **ds2202_digout_mode_set**.

Note

If the digital output drivers are disabled, writing or reading to the I/O ports has no effect.

Parameters

base PHS-bus base address of the DS2202 board

channel Defines the channels which low side is to be enabled in the range 0x00000000 ... 0xFFFFFFFF. '1' enables the low-side switch, '0' means that the channel is not affected. You can use the following predefined symbols. To enable more than one digital output, you must specify a list of predefined symbols combined by the logical operator OR:

Predefined Symbol	Bit	Signal
DS2202_OUTSTP_DIGOUT1 ... DS2202_OUTSTP_DIGOUT16	Bit 0 ... Bit 15	DIG_OUTx
DS2202_OUTSTP_PWMOUT1 ... DS2202_OUTSTP_PWMOUT9	Bit 16 ... Bit 24	PWM_OUTx

Default value All low side switches are enabled after reset.

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

ds2202_digout_ls_clear	16
ds2202_digout_ls_write	14
ds2202_digout_mode_set	12
ds2202_init	10

ds2202_digout_ls_clear

Syntax

```
void ds2202_digout_ls_clear(
    Int32 base,
    Int32 channel)
```

Include file ds2202.h

Purpose To disable the low-side switch of individual digital outputs.

Description

This function disables only the digital outputs specified by the **channel** parameter. Use **ds2202_digout_ls_write** to set individual digital outputs and to reset all other digital outputs.

Note

If the digital output drivers are disabled, writing or reading to the I/O ports has no effect.

Parameters

base PHS-bus base address of the DS2202 board

channel Defines the channels whose low-side switch is to be disabled in the range 0x00000000 ... 0x0FFFFFFF. '1' disables the low-side switch, '0' means that the channel is not affected. You can use the following predefined symbols. To disable more than one digital output, you must specify a list of predefined symbols combined by the logical operator OR:

Predefined Symbol	Bit	Signal
DS2202_OUTSTP_DIGOUT1 ... DS2202_OUTSTP_DIGOUT16	Bit 0 ... Bit 15	DIG_OUTx
DS2202_OUTSTP_PWMOUT1 ... DS2202_OUTSTP_PWMOUT9	Bit 16 ... Bit 24	PWM_OUTx

Default value All low side switches are enabled after reset.

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

ds2202_digout_ls_set	15
ds2202_digout_ls_write	14
ds2202_digout_mode_set	12
ds2202_init	10

ds2202_digout_hs_vbat1_write

Syntax

```
void ds2202_digout_hs_vbat1_write(
    Int32 base,
    Int32 channel)
```

Include file ds2202.h

Purpose To enable or disable the high-side switch to VBAT1 of all digital outputs.

Description

This function affects all digital outputs and resets the digital outputs that are not explicitly set. Use `ds2202_digout_hs_vbat1_set` to set individual digital outputs without affecting other outputs.

Note

If the digital output drivers are disabled, writing or reading to the I/O port has no effect.

Parameters

base PHS-bus base address of the DS2202 board

channel Defines the channels whose high-side switch is to be enabled or disabled in the range 0x00000000 ... 0xFFFFFFFF. '1' enables the high-side switch, '0' disables the high-side switch. You can use the following predefined symbols. To enable more than one digital output, you must specify a list of predefined symbols combined by the logical operator OR:

Predefined Symbol	Bit	Signal
DS2202_OUTSTP_DIGOUT1 ... DS2202_OUTSTP_DIGOUT16	Bit 0 ... Bit 15	DIG_OUTx
DS2202_OUTSTP_PWMOUT1 ... DS2202_OUTSTP_PWMOUT9	Bit 16 ... Bit 24	PWM_OUTx

Default value

All high-side switches to VBAT1 are enabled after reset.

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics**References**

ds2202_digout_hs_vbat1_clear	20
ds2202_digout_hs_vbat1_set	19
ds2202_digout_mode_set	12
ds2202_init	10

ds2202_digout_hs_vbat1_set

Syntax

```
void ds2202_digout_hs_vbat1_set(
    Int32 base,
    Int32 channel)
```

Include file

ds2202.h

Purpose

To enable the high-side switch to VBAT1 of individual digital outputs.

Description

This function enables only the digital outputs specified by the **channel** parameter. Use **ds2202_digout_hs_vbat1_write** to set individual digital outputs and to reset all other digital outputs.

To use the digital outputs, you must additionally enable their output drivers using the function **ds2202_digout_mode_set**.

Note

If the digital output drivers are disabled, writing or reading to the I/O ports has no effect.

Parameters

base PHS-bus base address of the DS2202 board

channel Defines the channels whose high-side switch is to be enabled in the range 0x00000000 ... 0xFFFFFFFF. '1' enables the high-side switch, '0' means that the channel is not affected. You can use the following predefined symbols. To enable more than one digital output, you must specify a list of predefined symbols combined by the logical operator OR:

Predefined Symbol	Bit	Signal
DS2202_OUTSTP_DIGOUT1 ... DS2202_OUTSTP_DIGOUT16	Bit 0 ... Bit 15	DIG_OUTx
DS2202_OUTSTP_PWMOUT1 ... DS2202_OUTSTP_PWMOUT9	Bit 16 ... Bit 24	PWM_OUTx

Default value

All high side switches to VBAT1 are enabled after reset.

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

ds2202_digout_hs_vbat1_clear	20
ds2202_digout_hs_vbat1_write	17
ds2202_digout_mode_set	12
ds2202_init	10

ds2202_digout_hs_vbat1_clear

Syntax

```
void ds2202_digout_hs_vbat1_clear(  
    Int32 base,  
    Int32 channel)
```

Include file

`ds2202.h`

Purpose

To disable the high-side switch to VBAT1 of individual digital outputs.

Description

This function disables only the digital outputs specified by the **channel** parameter. Use `ds2202_digout_hs_vbat1_write` to set individual digital outputs and to reset all other digital outputs.

Note

If the digital output drivers are disabled, writing or reading to the I/O ports has no effect.

Parameters

base PHS-bus base address of the DS2202 board

channel Defines the channels whose high-side switch is to be disabled in the range 0x00000000 ... 0xFFFFFFFF. '1' disables the high-side switch, '0' means that the channel is not affected. You can use the following predefined symbols. To disable more than one digital output, you must specify a list of predefined symbols combined by the logical operator OR:

Predefined Symbol	Bit	Signal
DS2202_OUTSTP_DIGOUT1 ... DS2202_OUTSTP_DIGOUT16	Bit 0 ... Bit 15	DIG_OUTx
DS2202_OUTSTP_PWMOUT1 ... DS2202_OUTSTP_PWMOUT9	Bit 16 ... Bit 24	PWM_OUTx

Default value All high-side switches to VBAT1 are enabled after reset.

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

ds2202_digout_hs_vbat1_set	19
ds2202_digout_hs_vbat1_write	17
ds2202_digout_mode_set	12
ds2202_init	10

ds2202_digout_hs_vbat2_write

Syntax

```
void ds2202_digout_hs_vbat2_write(
    Int32 base,
    Int32 channel)
```

Include file ds2202.h

Purpose To enable/disable the high-side switch to VBAT2 of all digital outputs.

Description

This function affects all digital outputs and resets the digital outputs that are not explicitly set. Use [ds2202_digout_hs_vbat2_set](#) to set individual digital outputs without affecting other outputs.

Note

If the digital output drivers are disabled, writing or reading to the I/O port has no effect.

Parameters**base** PHS-bus base address of the DS2202 board

channel Defines the channels whose high-side switch is to be enabled or disabled in the range 0x00000000 ... 0xFFFFFFFF. '1' enables the high-side switch, '0' disables the high-side switch. You can use the following predefined symbols. To enable more than one digital output, you must specify a list of predefined symbols combined by the logical operator OR:

Predefined Symbol	Bit	Signal
DS2202_OUTSTP_DIGOUT1 ... DS2202_OUTSTP_DIGOUT16	Bit 0 ... Bit 15	DIG_OUTx
DS2202_OUTSTP_PWMOUT1 ... DS2202_OUTSTP_PWMOUT9	Bit 16 ... Bit 24	PWM_OUTx

Default value

All high-side switches to VBAT2 are disabled after reset.

Return value

None

Execution timesFor information, refer to [Function Execution Times](#) on page 207.**Related topics****References**

ds2202_digout_hs_vbat2_clear	24
ds2202_digout_hs_vbat2_set	22
ds2202_digout_mode_set	12
ds2202_init	10

ds2202_digout_hs_vbat2_set

Syntax

```
void ds2202_digout_hs_vbat2_set(
    Int32 base,
    Int32 channel)
```

Include file

ds2202.h

Purpose

To enable the high-side switch to VBAT2 of individual digital outputs.

Description

This function enables only the digital outputs specified by the **channel** parameter. Use **ds2202_digout_hs_vbat2_write** to set individual digital outputs and to reset all other digital outputs.

To use the digital outputs, you must additionally enable their output drivers using the function **ds2202_digout_mode_set**.

Note

If the digital output drivers are disabled, writing or reading to the I/O ports has no effect.

Parameters

base PHS-bus base address of the DS2202 board

channel Defines the channels whose high-side switch is to be enabled in the range 0x00000000 ... 0xFFFFFFFF. '1' enables the high-side switch, '0' means that the channel is not affected. You can use the following predefined symbols. To enable more than one digital output, you must specify a list of predefined symbols combined by the logical operator OR:

Predefined Symbol	Bit	Signal
DS2202_OUTSTP_DIGOUT1 ... DS2202_OUTSTP_DIGOUT16	Bit 0 ... Bit 15	DIG_OUTx
DS2202_OUTSTP_PWMOUT1 ... DS2202_OUTSTP_PWMOUT9	Bit 16 ... Bit 24	PWM_OUTx

Default value

All high-side switches to VBAT2 are disabled after reset.

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics**References**

ds2202_digout_hs_vbat2_clear	24
ds2202_digout_hs_vbat2_write	21
ds2202_digout_mode_set	12
ds2202_init	10

ds2202_digout_hs_vbat2_clear

Syntax

```
void ds2202_digout_hs_vbat2_clear(  
    Int32 base,  
    Int32 channel)
```

Include file

ds2202.h

Purpose

To disable the high-side switch to VBAT2 of individual digital outputs.

Description

This function disables only the digital outputs specified with the **channel** parameter. Use **ds2202_digout_hs_vbat2_write** to set individual digital outputs and reset all other digital outputs.

Note

If the digital output drivers are disabled, writing or reading to the I/O ports has no effect.

Parameters

base PHS-bus base address of the DS2202 board

channel Defines the channels whose high-side switch is to be disabled in the range 0x00000000 ... 0xFFFFFFFF. '1' disables the high-side switch, '0' means that the channel is not affected. You can use the following predefined symbols. To disable more than one digital output, you must specify a list of predefined symbols combined by the logical operator OR:

Predefined Symbol	Bit	Signal
DS2202_OUTSTP_DIGOUT1 ... DS2202_OUTSTP_DIGOUT16	Bit 0 ... Bit 15	DIG_OUTx
DS2202_OUTSTP_PWMOUT1 ... DS2202_OUTSTP_PWMOUT9	Bit 16 ... Bit 24	PWM_OUTx

Default value

All high-side switches to VBAT2 are disabled after reset.

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

ds2202_digout_hs_vbat2_set.....	22
ds2202_digout_hs_vbat2_write.....	21
ds2202_digout_mode_set.....	12
ds2202_init.....	10

Multi I/O Interface

Introduction

The multi I/O interface consists of several components comprising the functions for ADC, DAC, bit I/O, PWM generation and measurement.

Where to go from here

Information in this section

ADC Unit.....	29
The analog digital converter unit comprises all functions to start and read the A/D channels.	
DAC Unit.....	42
The digital analog converter unit comprises all functions to program the digital/analog converters.	
Bit I/O Unit.....	45
The bit I/O unit comprises the functions to program the digital I/O ports.	
Timing Mode.....	53
The timing mode functions let you enable PWM or frequency measurement/generation.	
PWM Signal Generation.....	59
With the PWM signal generation functions, you can generate standard PWM signals.	
PWM Signal Measurement.....	63
The PWM signal measurement functions let you analyze PWM signals.	
Square-Wave Signal Generation.....	67
Using these functions, you can generate square-wave signals.	
Frequency Measurement.....	71
The frequency measurement functions let you measure the frequency of square-wave signals.	

Information in other sections

[Multi I/O Interface \(DS2202 Features !\[\]\(2bdfe261b986065ee0ac76460d6528c9_img.jpg\)](#))

The DS2202 has channels for A/D conversion, D/A conversion and for digital and timing I/O.

ADC Unit

Where to go from here

Information in this section

Basics on Accessing the ADC Unit via RTLib.....	30
The analog digital converter unit comprises all functions to start and read the A/D channels.	
Example of Fast Read and Standard Read.....	30
This example shows how to implement a fast read/standard read access for four A/D channels.	
Example of Single Read.....	31
This example shows how to implement a single read access for one A/D channel.	
Example of Multiple Read.....	33
The example shows how to implement a multiple read access for five A/D channels.	
ds2202_adc_start.....	34
To start the A/D conversion.	
ds2202_adc_block_in_fast.....	35
To wait until ADC conversion finished and read ADC input values.	
ds2202_adc_single_in.....	36
To read an input value when the A/D conversion is finished.	
ds2202_adc_block_init.....	38
To initialize the read function for several A/D channels for a multiple read access.	
ds2202_adc_block_start.....	39
To start the A/D conversion process for the channels 1 ... 4, 1 ... 8, 1 ... 12, or 1 ... 16.	
ds2202_adc_block_in.....	40
To read the input values of several A/D channels.	

Information in other sections

[ADC Unit \(DS2202 Features \)](#)

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

Basics on Accessing the ADC Unit via RTLib

Introduction

The analog digital converter unit comprises all functions to start and read the A/D channels.

Methods to access the A/D converters

There are three different methods to access the A/D converters:

Fast read/standard read To read 4, 8, 12, or 16 A/D channels blockwise at the same time. This method allows you to read several channels of the selected block.

Single read To read only one A/D channel. This method allows you to access only one A/D channel.

Multiple read To read several A/D channels at the same time. This method allows you to read several selected channels (not blockwise), but has the highest execution time.

Related topics

Examples

Example of Fast Read and Standard Read.....	30
Example of Multiple Read.....	33
Example of Single Read.....	31

Example of Fast Read and Standard Read

Introduction

This example shows how to implement a fast read/standard read access for the A/D channels 1 ... 4:

Example

```

1  #include <Brtenv.h>           /* basic real-time environment */
2  #include <Ds2202.h>
3  /*-----*/
4
5  #define DT 1e-3               /* 1 ms simulation step size */
6
7  dsfloat adc_data[4] = {0, 0, 0, 0}; /* A/D channel values */
8
9  /* variables for execution time profiling */
10 dsfloat exec_time;           /* execution time */
11
12 /*-----*/
13

```

```

14 void isr_t1()           /* timer1 interrupt service routine */
15 {
16     ts_timestamp_type ts;
17     RTLIB_SRT_ISR_BEGIN();           /* overload check */
18     RTLIB_TIC_START();               /* start time measurement */
19     ts_timestamp_read(&ts);
20     host_service(1, &ts);           /* data acquisition service */
21     /* start A/D channels 1 ... 4 for conversion */
22     ds2202_adc_start(DS2202_1_BASE, DS2202_ADC_START4);
23
24     /* read A/D channels 1 ... 4 */
25     ds2202_adc_block_in_fast(DS2202_1_BASE, adc_data);
26     exec_time = RTLIB_TIC_READ(); /* calculate execution time*/
27
28     RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
29 }
30
31 /*-----*/
32 void main()
33 {
34     init();           /* initialize hardware system */
35     ds2202_init(DS2202_1_BASE); /* initialize DS2202 board */
36
37     msg_info_set(MSG_SM_RTLib, 0, "System started.");
38
39     RTLIB_SRT_START(DT, isr_t1); /*init sampling clock timer*/
40     RTLIB_TIC_INIT();
41
42     while(1)           /* background process */
43     {
44
45         RTLIB_BACKGROUND_SERVICE();
46
47     } /* while(1) */
48
49 } /* main() */

```

Related topics

Examples

Example of Multiple Read.....	33
Example of Single Read.....	31

Example of Single Read

Introduction

This example shows how to implement a single read access for A/D channel 6:

Example

```

1 #include <Brtenv.h>          /* basic real-time environment */
2 #include <Ds2202.h>
3
4 /*-----*/
5 #define DT 1e-3              /* 1 ms simulation step size */
6
7 Int32  channel = 6;          /* A/D channel number */
8 dsfloat adc_data;           /* A/D channel value */
9
10 /* variables for execution time profiling */
11 dsfloat exec_time;           /* execution time */
12
13 /*-----*/
14 void isr_t1()                /* timer1 interrupt service routine */
15 {
16     ts_timestamp_type ts;
17     RTLIB_SRT_ISR_BEGIN();    /* overload check */
18     RTLIB_TIC_START();        /* start time measurement */
19     ts_timestamp_read(&ts);
20     host_service(1, &ts);     /* data acquisition service */
21     /* start A/D channels 1-8 for conversion */
22     ds2202_adc_start(DS2202_1_BASE, DS2202_ADC_MASK(6));
23
24     /* read A/D channel 6 */
25     ds2202_adc_single_in(DS2202_1_BASE, channel, &adc_data);
26
27     exec_time = RTLIB_TIC_READ(); /* calc. execution time */
28     RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
29 }
30 /*-----*/
31 void main()
32 {
33
34     init();                  /* initialize hardware system */
35
36     ds2202_init(DS2202_1_BASE); /* initialize DS2202 board */
37
38     msg_info_set(MSG_SM_RTLib, 0, "System started.");
39
40
41     /* initialize sampling clock timer */
42     RTLIB_SRT_START(DT, isr_t1);
43     RTLIB_TIC_INIT();
44
45     while(1)                 /* background process */
46     {
47
48         RTLIB_BACKGROUND_SERVICE();
49
50     } /* while(1) */

```



```

51
52 } /* main() */

```

Related topics

Examples

Example of Fast Read and Standard Read.....	30
Example of Multiple Read.....	33

Example of Multiple Read

Introduction

This example shows how to implement a multiple read access for the A/D channels 1, 5, 6, 12 and 13:

Example

```

1  #include <Brtenv.h>           /* basic real-time environment */
2  #include <Ds2202.h>
3
4  /*-----*/
5
6  #define DT 1e-3               /* 1 ms simulation step size */
7
8  Int32  count = 5;             /* number of A/D channels to read */
9  Int32  channels[5] = {1, 5, 6, 12, 13}; /* A/D channel num.*/
10 dsfloat adc_data[5] = {0, 0, 0, 0, 0}; /* A/D channel value */
11
12 /* variables for execution time profiling */
13 dsfloat exec_time;             /* execution time */
14 /*-----*/
15
16 void isr_t1()                 /* timer1 interrupt service routine */
17 {
18     ts_timestamp_type ts;
19     RTLIB_SRT_ISR_BEGIN();     /* overload check */
20     RTLIB_TIC_START();         /* start time measurement */
21     ts_timestamp_read(&ts);
22     host_service(1, &ts);      /* data acquisition service */
23     /* start A/D channels for conversion */
24     ds2202_adc_block_start(DS2202_1_BASE);
25
26
27     /* read A/D channels */
28     ds2202_adc_block_in(DS2202_1_BASE, adc_data);
29
30     exec_time = RTLIB_TIC_READ(); /* calc. execution time */
31

```

```

32   RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
33 }
34 /*-----*/
35
36 void main()
37 {
38
39     init(); /* initialize hardware system */
40
41     ds2202_init(DS2202_1_BASE); /* initialize DS2202 board */
42
43     ds2202_adc_block_init(DS2202_1_BASE, count, channels);
44
45     msg_info_set(MSG_SM_RTLib, 0, "System started.");
46
47     /* initialize sampling clock timer */
48     RTLIB_SRT_START(DT, isr_t1);
49     RTLIB_TIC_INIT();
50
51     while(1) /* background process */
52     {
53
54         RTLIB_BACKGROUND_SERVICE();
55
56     } /* while(1) */
57
58 } /* main() */

```

Related topics

Examples

Example of Fast Read and Standard Read.....	30
Example of Single Read.....	31

ds2202_adc_start

Syntax

```

void ds2202_adc_start(
    Int32 base,
    Int32 mask)

```

Include file

ds2202.h

Purpose

To start the A/D conversion for the channels 1 ... 4, 1 ... 8, 1 ... 12 or 1 ... 16.

I/O mapping For information on the I/O mapping, refer to [ADC Unit \(DS2202 Features\)](#).

Parameters

base PHS-bus base address of the DS2202 board

mask Input channels to be started. The following symbols are predefined:

Predefined Symbol	Meaning	Value
DS2202_ADC_START4	Starts channels 1 ... 4.	1...4
DS2202_ADC_START8	Starts channels 1 ... 8.	1...8
DS2202_ADC_START12	Starts channels 1 ... 12.	1...12
DS2202_ADC_START16	Starts channels 1 ... 16.	1...16
DS2202_ADC_MASK(channel)	Starts the block the given channel belongs to. Use this symbol in connection with <code>ds2202_adc_single_in</code> .	1...4, 8, 12 or 16

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

Examples

Example of Fast Read and Standard Read	30
Example of Single Read	31

References

ds2202_adc_block_in_fast	35
ds2202_adc_single_in	36


ds2202_adc_block_in_fast

Syntax

```
void ds2202_adc_block_in_fast(
    Int32 base,
    dsfloat *data)
```

Include file `ds2202.h`

Purpose To wait until ADC conversion finished and read ADC input values.

I/O mapping	For information on the I/O mapping, refer to ADC Unit (DS2202 Features ).
Description	<p>Polls the ADC busy flag until conversion is complete and then reads the input values of the channels selected by <code>ds2202_adc_start</code>. Input values are scaled to the range 0..1.0 and are written to memory starting at the address given by the <code>data</code> parameter.</p> <div> <p>Note</p> <p>ADC inputs are differential inputs. Each input has an individual ground sense line (\overline{ADCx}), which must be connected to the plant near the sensor, or connected to GND at the DS2202 connector, for all ADC channels used. The voltage difference of ($ADCx - \overline{ADCx}$) is measured by the ADC.</p> </div>
Parameters	<p>base PHS-bus board base address</p> <p>data Address where ADC input data is written. You have to allocate the target buffer with the appropriate length. The length depends on the number of channels that shall be converted.</p>
Return value	None
Execution times	For information, refer to Function Execution Times on page 207.
Related topics	<p>Examples</p> <p>Example of Fast Read and Standard Read..... 30</p> <p>References</p> <p>ds2202_adc_start..... 34</p>

ds2202_adc_single_in

Syntax

```
void ds2202_adc_single_in(
    Int32 base,
    Int32 channel,
    dsfloat *value)
```

Include file `ds2202.h`

Purpose To read an input value when the A/D conversion is finished.

I/O mapping For information on the I/O mapping, refer to [ADC Unit \(DS2202 Features !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)](#)).

Description Polls the ADC busy flag until conversion is complete and then reads the input value of the selected channel. It is assumed that the channel is started by using `ds2202_adc_start`. The input value is scaled to the range 0 ... 1.0 and is written to memory at the address given by the `value` parameter.

Note

ADC inputs are differential inputs. Each input has an individual ground sense line (\overline{ADCx}), which must be connected to the plant near the sensor, or connected to GND at the DS2202 connector, for all ADC channels used. The voltage difference of ($ADCx - \overline{ADCx}$) is measured by the ADC.

Parameters **base** PHS-bus base address of the DS2202 board

channel Channel number within the range of 1 ... 16. You can also use the following predefined symbols:

Predefined Symbol	Meaning
DS2202_ADC_CH1	A/D channel 1
...	...
DS2202_ADC_CH16	A/D channel 16

value Input value. The value is scaled as follows:

Input Voltage Range	Return Value Range
0 ... 60 V	0 ... 1.0

Return value None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics**Examples**

[Example of Single Read.....](#) 31

References

[ds2202_adc_start.....](#) 34

ds2202_adc_block_init

Syntax

```
void ds2202_adc_block_init(  
    Int32 base,  
    Int32 count,  
    Int32 *channels)
```

Include file

ds2202.h

Purpose

To initialize the read function for several A/D channels for a multiple read access.

I/O mapping

For information on the I/O mapping, refer to [ADC Unit \(DS2202 Features\)](#).

Description

`ds2202_adc_block_init` determines the blocks of channels that must be converted in correspondence to the channel numbers specified in the `channels` array.

Note

Due to the hardware used, the conversion will always be started for the appropriate block of channels. For example, if you select only channel "5" in `ds2202_adc_block_init` the conversion will be started for channels 1 ... 8.

Parameters

base PHS-bus base address of the DS2202 board

count Number of selected channels, that is, the size of the `channels` array

channels Array of channel numbers (1 ... 16). You can also use the following predefined symbols:

Predefined Symbol	Meaning
DS2202_ADC_CH1	A/D channel 1
...	...
DS2202_ADC_CH16	A/D channel 16

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

Examples

[Example of Multiple Read](#)..... 33

References

[ds2202_adc_block_in](#)..... 40
[ds2202_adc_block_start](#)..... 39

ds2202_adc_block_start

Syntax `void ds2202_adc_block_start(Int32 base)`

Include file `ds2202.h`

Purpose To start the A/D conversion process for the channels 1 ... 4, 1 ... 8, 1 ... 12, or 1 ... 16.

I/O mapping For information on the I/O mapping, refer to [ADC Unit \(DS2202 Features !\[\]\(274fd520e03b61c1b9ffc861754cacdc_img.jpg\)](#)).

Description

The A/D conversion is started for the channels selected by the `ds2202_adc_block_init` function.

Note

Due to the hardware used, the conversion will always be started for the appropriate block of channels. For example, if you select only channel "5" in `ds2202_adc_block_init` the conversion will be started for channels 1 ... 8.

Parameters

base PHS-bus base address of the DS2202 board

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics**Examples**

[Example of Multiple Read.....](#) 33

References

[ds2202_adc_block_in.....](#) 40
[ds2202_adc_block_init.....](#) 38

ds2202_adc_block_in

Syntax

```
void ds2202_adc_block_in(
    Int32 base,
    dsfloat *data)
```

Include file

`ds2202.h`

Purpose

To read the input values of several A/D channels.

I/O mapping

For information on the I/O mapping, refer to [ADC Unit \(DS2202 Features !\[\]\(346f5b9c8222e44e815e44b5dc7c53e5_img.jpg\)](#)).

Description You have to start the channels with `ds2202_adc_block_start` first. Then `ds2202_adc_block_in` polls the ADC busy flag until conversion is complete and reads the input values of the selected channels. The input values are scaled to the range of 0 ... 1.0 and written to the memory starting at the address given by the `data` parameter.

Parameters

base PHS-bus base address of the DS2202 board

data Start address where the input values of the selected channels are written. The values are scaled as follows:

Input Voltage Range	Return Value Range
0 ... 60 V	0 ... 1.0

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

Examples

[Example of Multiple Read.....](#) 33

References

[ds2202_adc_block_init.....](#) 38

[ds2202_adc_block_start.....](#) 39

DAC Unit

Introduction

The digital analog converter unit comprises all functions to program the digital/analog converters.

Where to go from here

Information in this section

Example of the DAC Unit.....	42
The example shows how to write two values to the D/A channels 1 and 2.	
ds2202_dac_out.....	43
To write a value to the DAC output.	

Information in other sections

[DAC Unit \(DS2202 Features \)](#)

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

Example of the DAC Unit

Introduction

This example shows how to write two values to the D/A channels 1 and 2:

Example

```

1  #include <Brtenv.h>           * basic real-time environment */
2  #include <Ds2202.h>
3
4  /*-----*/
5
6  #define DT 1e-3               /* 1 ms simulation step size */
7
8  dsfloat val = 0.0;           /* D/A output value */
9  UInt32 count = 0;
10
11 /* variables for execution time profiling */
12 dsfloat exec_time;           /* execution time */
13 /*-----*/
14
15 void isr_t1()                /* timer1 interrupt service routine */
16 {

```

```

17  ts_timestamp_type ts;
18  RTLIB_SRT_ISR_BEGIN();           /* overload check */
19  RTLIB_TIC_START();               /* start time measurement */
20  ts_timestamp_read(&ts);
21  host_service(1, &ts);           /* data acquisition service */
22  ds2202_dac_out(DS2202_1_BASE, DS2202_DAC_CH1, value);
23
24  val = (val == 0.0) ? 0.5 : 0.0; /* toggle D/A value */
25
26  ds2202_dac_out(DS2202_1_BASE, DS2202_DAC_CH2, val);
27
28  exec_time = RTLIB_TIC_READ();    /* calc. execution time */
29
30  RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
31 }
32 /*-----*/
33
34 void main()
35 {
36     init();                       /* initialize hardware system */
37     ds2202_init(DS2202_1_BASE);   /* initialize DS2202 board */
38
39     msg_info_set(MSG_SM_RTLIB, 0, "System started.");
40
41     /* initialize sampling clock timer */
42     RTLIB_SRT_START(DT, isr_t1);
43     RTLIB_TIC_INIT();
44
45     while(1)                      /* background process */
46     {
47
48         RTLIB_BACKGROUND_SERVICE();
49
50     } /* while(1) */
51
52 } /* main() */

```

Related topics

References

ds2202_dac_out..... 43

ds2202_dac_out

Syntax

```

void ds2202_dac_out(
    Int32 base,
    Int32 channel,
    dsfloat value)

```

Include file ds2202.h

Purpose To update the DAC output of the specified channel.

I/O mapping For information on the I/O mapping, refer to [DAC Unit \(DS2202 Features !\[\]\(dfbd6b3763a6d1d9afaa974f64e2e4b5_img.jpg\)](#)).

Description Writes the output value to the specified DAC channel. The value must be in the range 0 ... 1.0.

Parameters
base PHS-bus base address of the DS2202 board
channel Channel number within the range of 1 ... 20. You can also use the following predefined symbols:

Predefined Symbol	Meaning
DS2202_DAC_CH1	DAC channel 1
...	...
DS2202_DAC_CH20	DAC channel 20

value Output value within the range of 0 ... 1.0. The value is scaled as follows:

Value Range	Output Voltage Range
0 ... 1.0	0 V...VREF

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics Examples[Example of the DAC Unit..... 42](#)

Bit I/O Unit

Introduction

The bit I/O unit comprises the functions you need to program the digital I/O port.

Note

Use `ds2202_digout_mode_set` to enable the digital output ports before using the outputs.

Where to go from here

Information in this section

Example of Using the Bit I/O Unit.....	46
The example shows how to set and clear bits periodically and how to read an input bitmap.	
ds2202_bit_io_in.....	47
To read from the digital input port and write the input value to the memory.	
ds2202_bit_io_out.....	49
To write a value to the digital output port.	
ds2202_bit_io_set.....	50
To set specific bits of the digital output port.	
ds2202_bit_io_clear.....	51
To clear specific bits of the digital output port.	

Information in other sections

[Bit I/O Unit \(DS2202 Features !\[\]\(6bb0e4f14c4133b37d2887cb37e67ddd_img.jpg\)](#))

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

Example of Using the Bit I/O Unit

Introduction

This example shows how to set and clear bits periodically and how to read an input bitmap.

Example

```

1  #include <Brtenv.h>           /* basic real-time environment */
2  #include <Ds2202.h>
3
4  /*-----*/
5
6  #define DT 1e-3               /* 1 ms simulation step size */
7
8  /* variables for ControlDesk */
9  UInt32 bitmap = 0;
10 UInt32 mask_clear = 0;
11 UInt32 mask_set = 0;
12
13 /* variables for execution time profiling */
14 dsfloat exec_time;           /* execution time */
15 /*-----*/
16
17 void isr_t1()                /* timer1 interrupt service routine */
18 {
19     static UInt32 i = 0;
20     ts_timestamp_type ts;
21     RTLIB_SRT_ISR_BEGIN();    /* overload check */
22     RTLIB_TIC_START();        /* start time measurement */
23     ts_timestamp_read(&ts);
24     host_service(1, &ts);     /* data acquisition service */
25     /* clears I/O port i and sets I/O port i to "1" */
26     mask_clear = i;
27     mask_set = i - 1;
28     ds2202_bit_io_clear(DS2202_1_BASE, (0x0001 << mask_clear));
29     ds2202_bit_io_set(DS2202_1_BASE, (0x0001 << mask_set));
30
31     /* increments i until channel 16 is reached */
32     i++;
33     if (i == 16)
34         i = 0;
35
36     /* reads the 16 bits I/O port of group 1 */
37     ds2202_bit_io_in(DS2202_1_BASE, DS2202_BITIO_IN_G1, &bitmap);
38
39     exec_time = RTLIB_TIC_READ(); /* calculate execution time */
40
41     RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
42 }
43 /*-----*/
44
45 void main()

```

```

46 {
47
48     init();                /* initialize hardware system */
49
50     ds2202_init(DS2202_1_BASE); /* initialize DS2202 board */
51
52     msg_info_set(MSG_SM_RTLIB, 0, "System started.");
53
54     /* enable digital outputs */
55     ds2202_digout_mode_set(DS2202_1_BASE, DS2202_DIGOUT_ENABLE);
56
57     /* sets the Bit I/O ports 0 ... 15 to "1" */
58     ds2202_bit_io_out(DS2202_1_BASE, 0x0000FFFF);
59
60     /* initialize sampling clock timer */
61     RTLIB_SRT_START(DT, isr_t1);
62     RTLIB_TIC_INIT();
63
64     while(1)                /* background process */
65     {
66
67         RTLIB_BACKGROUND_SERVICE();
68
69     } /* while(1) */
70
71 } /* main() */

```

Related topics

References

ds2202_bit_io_clear.....	51
ds2202_bit_io_in.....	47
ds2202_bit_io_out.....	49
ds2202_bit_io_set.....	50

ds2202_bit_io_in

Syntax

```

void ds2202_bit_io_in(
    Int32 base,
    Int32 group,
    UInt32 *value)

```

Include file

ds2202.h

Purpose

To read from the digital input port and write the input value to the memory at the address given by the **value** parameter.

Description

This function reads from different groups of digital input channels, corresponding to the **group** parameter. For group 1 and 2, 16 digital input bits are read, so the returned value is in the range 0 ... 0xFFFF. For group 3, 6 digital input bits are read, so the returned value is in the range 0 ... 0x3F.

I/O mapping

For information on the I/O mapping, refer to [Bit I/O Unit \(DS2202 Features !\[\]\(6605b201d6f14d9b3bcb8ab5f274d107_img.jpg\)](#)).

Note

The inputs are 12 ... 42 V compatible. After initialization, the input threshold is set to 2.5 V. Use `ds2202_digin_threshold_set` to set the threshold within the range of 1.0 ... 23.8 V. For further information, refer to [Digital Inputs \(PHS Bus System Hardware Reference !\[\]\(fa6f3af6bfa46c5d4a2d362681095beb_img.jpg\)](#)).

Parameters

base PHS-bus base address of the DS2202 board

group Group of digital input channels whose statuses are to be read. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_BITIO_IN_G1	Reads channels DIG_IN1 ... DIG_IN16
DS2202_BITIO_IN_G2	Reads channels DIG_IN17 ... DIG_IN32
DS2202_BITIO_IN_G3	Reads channels DIG_IN33 ... DIG_IN38

value Address where the input value is written

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics**Examples**

[Example of Using the Bit I/O Unit.....](#) 46

References

[ds2202_bit_io_out.....](#) 49
[ds2202_digin_threshold_set.....](#) 11

ds2202_bit_io_out

Syntax

```
void ds2202_bit_io_out(
    Int32 base,
    UInt32 value)
```

Include file

ds2202.h

Purpose

To write the given output value to the 16-bit digital output port.

Description

This function affects all outputs and resets the output bits that are not explicitly set. Use `ds2202_bit_io_set` to set individual output bits without affecting other bits.

I/O mapping

For information on the I/O mapping, refer to [Bit I/O Unit \(DS2202 Features !\[\]\(d3102649f02e825ddb76dc3de0190154_img.jpg\)](#)).

Note

By default the outputs are disabled. Use `ds2202_digout_mode_set` to enable the digital outputs. For further information, refer to [Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(95b425611cbd2b8716a140cf67c81822_img.jpg\)](#)).

Parameters

base PHS-bus base address of the DS2202 board

value Output value within the range 0x0000 ... 0xFFFF: "0" clears the bit, "1" sets the bit. You can also use the following predefined symbols. To set more than one bit, you must specify a list of predefined symbols combined by the logical operator OR.

Predefined Symbol	Value	Meaning
DS2202_BITIO_OUT1	0x0001	Sets bit 1
DS2202_BITIO_OUT2	0x0002	Sets bit 2
DS2202_BITIO_OUT3	0x0004	Sets bit 3
...
DS2202_BITIO_OUT16	0x8000	Sets bit 16

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics

Examples

[Example of Using the Bit I/O Unit](#)..... 46

References

[ds2202_bit_io_in](#)..... 47
[ds2202_bit_io_set](#)..... 50

ds2202_bit_io_set

Syntax

```
void ds2202_bit_io_set(
    Int32 base,
    UInt32 mask)
```

Include file

ds2202.h

Purpose

To set individual digital output bits. The digital output bits specified by the parameter **mask** are set to high ("1") without affecting the other digital output bits.

I/O mapping

For information on the I/O mapping, refer to [Bit I/O Unit \(DS2202 Features\)](#).

Note

After initialization, the outputs are disabled. Use `ds2202_digout_mode_set` to enable the digital output ports. For further information, refer to [Digital Outputs \(PHS Bus System Hardware Reference\)](#).

Parameters

base PHS-bus base address of the DS2202 board

mask Set mask within the range 0x0000 ... 0xFFFF: "1" sets the bit, "0" has no effect. You can also use the following predefined symbols. To set more than one bit, you must specify a list of predefined symbols combined by the logical operator OR.

Predefined Symbol	Value	Meaning
DS2202_BITIO_OUT1	0x0001	Sets bit 1
DS2202_BITIO_OUT2	0x0002	Sets bit 2

Predefined Symbol	Value	Meaning
DS2202_BITIO_OUT3	0x0004	Sets bit 3
...
DS2202_BITIO_OUT16	0x8000	Sets bit 16

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

Examples

[Example of Using the Bit I/O Unit](#)..... 46

References

[ds2202_bit_io_clear](#)..... 51
[ds2202_digout_mode_set](#)..... 12

ds2202_bit_io_clear

Syntax

```
void ds2202_bit_io_clear(
    Int32 base,
    UInt32 mask)
```

Include file ds2202.h

Purpose

To clear individual digital output bits. The digital output bits specified by the **mask** parameter are set to low ("0") without affecting the other digital output bits.

I/O mapping

For information on the I/O mapping, refer to [Bit I/O Unit \(DS2202 Features !\[\]\(2bdfe261b986065ee0ac76460d6528c9_img.jpg\)](#)).

Note

After initialization, the outputs are disabled. Use `ds2202_digout_mode_set` to enable the digital output ports. For further information, refer to [Digital Outputs \(PHS Bus System Hardware Reference !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)](#)).

Parameters

base PHS-bus base address of the DS2202 board

mask Clear mask within the range 0x0000 ... 0xFFFF: "1" clears the bit, "0" has no effect. You can also use the following predefined symbols. To clear more than one bit, you must specify a list of predefined symbols combined by the logical operator OR.

Predefined Symbol	Value	Meaning
DS2202_BITIO_OUT1	0x0001	Sets bit 1
DS2202_BITIO_OUT2	0x0002	Sets bit 2
DS2202_BITIO_OUT3	0x0004	Sets bit 3
...
DS2202_BITIO_OUT16	0x8000	Sets bit 16

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics**Examples**

[Example of Using the Bit I/O Unit..... 46](#)

References

[ds2202_bit_io_set..... 50](#)
[ds2202_digout_mode_set..... 12](#)

Timing Mode

Introduction

With the timing mode functions you can set whether you want to perform PWM or frequency measurement/generation.

Where to go from here

Information in this section

ds2202_timing_out_mode_set.....	53
To set the output mode and range to PWM or square-wave signal generation.	
ds2202_timing_in_mode_set.....	56
To set the input mode and range to PWM signal or frequency measurement.	

ds2202_timing_out_mode_set

Syntax

```
void ds2202_timing_out_mode_set(
    Int32 base,
    Int32 channel,
    Int32 range,
    Int32 mode)
```

Include file

ds2202.h

Purpose

To set the output mode of the specified channel and the clock prescaler.

Note

Only one mode can be set for each output channel. Using the functions for PWM and square-wave signal generation simultaneously for the same channel can cause unpredictable results.

Parameters**base** PHS-bus base address of the DS2202 board**channel** PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_PWMOUT_CH1	PWM channel 1
...	...
DS2202_PWMOUT_CH9	PWM channel 9

range Period range/frequency of the timer unit within the range of 1 ... 16. The following symbols are predefined in frequency mode:

Predefined Symbol	Frequency Mode		
	Minimum	Maximum	Resolution
DS2202_TIMING_RANGE1	9.54 Hz	100 kHz	50 ns
DS2202_TIMING_RANGE2	4.77 Hz	100 kHz	100 ns
DS2202_TIMING_RANGE3	2.39 Hz	100 kHz	200 ns
DS2202_TIMING_RANGE4	1.20 Hz	100 kHz	400 ns
DS2202_TIMING_RANGE5	0.60 Hz	100 kHz	800 ns
DS2202_TIMING_RANGE6	0.30 Hz	100 kHz	1.6 μ s
DS2202_TIMING_RANGE7	0.15 Hz	100 kHz	3.2 μ s
DS2202_TIMING_RANGE8	75 mHz	78.12 kHz	6.4 μ s
DS2202_TIMING_RANGE9	38 mHz	39.06 kHz	12.8 μ s
DS2202_TIMING_RANGE10	19 mHz	19.53 kHz	25.6 μ s
DS2202_TIMING_RANGE11	10 mHz	9.76 kHz	51.2 μ s
DS2202_TIMING_RANGE12	5.0 mHz	4.88 kHz	103 μ s
DS2202_TIMING_RANGE13	2.5 mHz	2.44 kHz	205 μ s
DS2202_TIMING_RANGE14	1.2 mHz	1.22 kHz	410 μ s
DS2202_TIMING_RANGE15	0.6 mHz	610.35 Hz	820 μ s
DS2202_TIMING_RANGE16	0.3 mHz	305.17 Hz	1.64 ms

The following symbols are predefined in PWM mode:

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
DS2202_TIMING_RANGE1	100 ns	10 μ s	3.27 ms	50 ns
DS2202_TIMING_RANGE2	200 ns	10 μ s	6.55 ms	100 ns
DS2202_TIMING_RANGE3	400 ns	10 μ s	13.1 ms	200 ns
DS2202_TIMING_RANGE4	800 ns	10 μ s	26.2 ms	400 ns
DS2202_TIMING_RANGE5	1.6 μ s	10 μ s	52.4 ms	800 ns
DS2202_TIMING_RANGE6	3.2 μ s	10 μ s	104 ms	1.6 μ s
DS2202_TIMING_RANGE7	6.4 μ s	10 μ s	209 ms	3.2 μ s
DS2202_TIMING_RANGE8	12.8 μ s	12.8 μ s	419 ms	6.4 μ s

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
DS2202_TIMING_RANGE9	25.6 μ s	25.6 μ s	838 ms	12.8 μ s
DS2202_TIMING_RANGE10	51.2 μ s	51.2 μ s	1.67 s	25.6 μ s
DS2202_TIMING_RANGE11	103 μ s	103 μ s	3.35 s	51.2 μ s
DS2202_TIMING_RANGE12	205 μ s	205 μ s	6.71 s	103 μ s
DS2202_TIMING_RANGE13	410 μ s	410 μ s	13.4 s	205 μ s
DS2202_TIMING_RANGE14	820 μ s	820 μ s	26.8 s	410 μ s
DS2202_TIMING_RANGE15	1.64 ms	1.64 ms	53.6 s	820 μ s
DS2202_TIMING_RANGE16	3.28 ms	3.28 ms	107.3 s	1.64 ms

mode Mode of the timing generation unit. The following modes are available:

Mode	Meaning
DS2202_D2PWM	PWM signal generation
DS2202_D2PWM_SYNCH_UPDATE	PWM signal generation with synchronous update
DS2202_D2F_LOW	Square-wave signal generation, the output is set to low level.
DS2202_D2F_HIGH	Square-wave signal generation, the output is set to high level.
DS2202_D2F_HOLD	Square-wave signal generation, the output keeps the current signal level (low or high).

Note

- The **mode** parameter for square-wave signal generation defines the output behavior when frequency $< f_{\min}$.
- For PWM signal generation with *synchronous* update, the output period should be constant. It is constant if $T = T_{\text{high}} + T_{\text{low}}$ is constant. If you change the period during run time, synchronous PWM update cannot be ensured.

Return value

None

Execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics**References**

[ds2202_timing_in_mode_set](#)..... 56

ds2202_timing_in_mode_set

Syntax

```
void ds2202_timing_in_mode_set(  
    Int32 base,  
    Int32 channel,  
    Int32 range,  
    Int32 mode)
```

Include file

ds2202.h

Purpose

To set the input mode of the specified channel and the clock prescaler.

Note

Only one mode for each input channel is adjustable. Using the functions for PWM and frequency measurement simultaneously for the same channel can cause unpredictable results.

Parameters

base PHS-bus base address of the DS2202 board

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_PWMIN_CH1	PWM channel 1
...	...
DS2202_PWMIN_CH24	PWM channel 24

range Period range/frequency of the timer unit within the range 1 ... 16. The following symbols are predefined in frequency mode:

Predefined Symbol	Frequency Mode		
	Minimum	Maximum	Resolution
DS2202_TIMING_RANGE1	9.54 Hz	100 kHz	50 ns
DS2202_TIMING_RANGE2	4.77 Hz	100 kHz	100 ns
DS2202_TIMING_RANGE3	2.39 Hz	100 kHz	200 ns
DS2202_TIMING_RANGE4	1.20 Hz	100 kHz	400 ns
DS2202_TIMING_RANGE5	0.60 Hz	100 kHz	800 ns
DS2202_TIMING_RANGE6	0.30 Hz	100 kHz	1.6 µs
DS2202_TIMING_RANGE7	0.15 Hz	100 kHz	3.2 µs
DS2202_TIMING_RANGE8	75 mHz	78.12 kHz	6.4 µs
DS2202_TIMING_RANGE9	38 mHz	39.06 kHz	12.8 µs
DS2202_TIMING_RANGE10	19 mHz	19.53 kHz	25.6 µs
DS2202_TIMING_RANGE11	10 mHz	9.76 kHz	51.2 µs
DS2202_TIMING_RANGE12	5.0 mHz	4.88 kHz	103 µs
DS2202_TIMING_RANGE13	2.5 mHz	2.44 kHz	205 µs
DS2202_TIMING_RANGE14	1.2 mHz	1.22 kHz	410 µs
DS2202_TIMING_RANGE15	0.6 mHz	610.35 Hz	820 µs
DS2202_TIMING_RANGE16	0.3 mHz	305.17 Hz	1.64 ms

The following symbols are predefined in PWM mode:

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
DS2202_TIMING_RANGE1	100 ns	10 µs	3.27 ms	50 ns
DS2202_TIMING_RANGE2	200 ns	10 µs	6.55 ms	100 ns
DS2202_TIMING_RANGE3	400 ns	10 µs	13.1 ms	200 ns
DS2202_TIMING_RANGE4	800 ns	10 µs	26.2 ms	400 ns
DS2202_TIMING_RANGE5	1.6 µs	10 µs	52.4 ms	800 ns
DS2202_TIMING_RANGE6	3.2 µs	10 µs	104 ms	1.6 µs
DS2202_TIMING_RANGE7	6.4 µs	10 µs	209 ms	3.2 µs
DS2202_TIMING_RANGE8	12.8 µs	12.8 µs	419 ms	6.4 µs
DS2202_TIMING_RANGE9	25.6 µs	25.6 µs	838 ms	12.8 µs
DS2202_TIMING_RANGE10	51.2 µs	51.2 µs	1.67 s	25.6 µs
DS2202_TIMING_RANGE11	103 µs	103 µs	3.35 s	51.2 µs
DS2202_TIMING_RANGE12	205 µs	205 µs	6.71 s	103 µs
DS2202_TIMING_RANGE13	410 µs	410 µs	13.4 s	205 µs
DS2202_TIMING_RANGE14	820 µs	820 µs	26.8 s	410 µs

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
DS2202_TIMING_RANGE15	1.64 ms	1.64 ms	53.6 s	820 μ s
DS2202_TIMING_RANGE16	3.28 ms	3.28 ms	107.3 s	1.64 ms

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.

mode Mode of the timing measurement unit. The following modes are available:

Mode	Meaning
DS2202_PWM2D	PWM signal measurement
DS2202_PWM2D_SYNCH_UPDATE	PWM signal measurement with synchronous update
DS2202_F2D	Frequency measurement

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

[ds2202_timing_out_mode_set.....](#) 53

PWM Signal Generation

Introduction

The PWM signal generation functions let you generate standard PWM signals.

Where to go from here

Information in this section

[Example of PWM Signal Generation..... 59](#)

The example shows how to generate a PWM signal with a period of 1 ms and a duty cycle of 25 % on PWM channel 1.

[ds2202_pwm_out..... 61](#)

To update the period and duty cycle for PWM signal generation.

Information in other sections

[PWM Signal Generation \(DS2202 Features \)](#)

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

[ds2202_timing_out_mode_set..... 53](#)

To set the output mode and range to PWM or square-wave signal generation.

Example of PWM Signal Generation

Introduction

This example shows how to generate a PWM signal with a period of 1 ms and a duty cycle of 25 % on PWM channel 1.

Example

```

1  #include <Brtenv.h>           /* basic real-time environment */
2  #include <Ds2202.h>
3
4  /*-----*/
5
6  #define DT 1e-3               /* 1 ms simulation step size */
7
8  dsfloat duty   = 0.25;        /* set duty cycle to 25% */
9  dsfloat period = 0.001;      /* set output period to 1 ms */
10
11 /* variables for execution time profiling */
12 dsfloat exec_time;            /* execution time */
13 /*-----*/

```

```

14
15 void isr_t1()           /* timer1 interrupt service routine */
16 {
17     ts_timestamp_type ts;
18     RTLIB_SRT_ISR_BEGIN();           /* overload check */
19     RTLIB_TIC_START();               /* start time measurement */
20     ts_timestamp_read(&ts);
21     host_service(1, &ts);           /* data acquisition service */
22     /* update period or duty-cycle */
23     ds2202_pwm_out(DS2202_1_BASE, DS2202_PWMOUT_CH1,
24                     period, duty);
25
26     exec_time = RTLIB_TIC_READ(); /* calculate execution time*/
27
28     RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
29 }
30 /*-----*/
31 void main()
32 {
33
34     init();           /* initialize hardware system */
35
36     ds2202_init(DS2202_1_BASE); /* initialize DS2202 board */
37
38     msg_info_set(MSG_SM_RTILIB, 0, "System started.");
39
40     /* enable digital output driver */
41     ds2202_digout_mode_set(DS2202_1_BASE,
42                             DS2202_DIGOUT_ENABLE);
43
44     /* set PWM range for output channel 1 */
45     ds2202_timing_out_mode_set(DS2202_1_BASE,
46                                 DS2202_PWMOUT_CH1,
47                                 DS2202_TIMING_RANGE5,
48                                 DS2202_D2PWM);
49
50     /* set values for PWM signal generation */
51     ds2202_pwm_out(DS2202_1_BASE, DS2202_PWMOUT_CH1,
52                     period, duty);
53
54     /* initialize sampling clock timer */
55     RTLIB_SRT_START(DT, isr_t1);
56     RTLIB_TIC_INIT();
57
58     while(1)           /* background process */
59     {
60
61         RTLIB_BACKGROUND_SERVICE();
62
63     } /* while(1) */
64
65 } /* main() */

```

Related topics

References

[ds2202_pwm_out](#)..... 61

ds2202_pwm_out

Syntax

```
void ds2202_pwm_out(
    Int32 base,
    Int32 channel,
    dsfloat period,
    dsfloat duty)
```

Include file

ds2202.h

Purpose

To update the PWM period and duty cycle of the specified PWM output channel during run time.

I/O mapping

For information on the I/O mapping, refer to [PWM Signal Generation \(DS2202 Features\)](#).

Note

- After initialization, the outputs are disabled. Use `ds2202_digout_mode_set` to enable the digital output ports.
- To minimize the quantization effect on the frequency resolution and the duty cycle, you should select the smallest possible frequency range. For detailed information, refer to [PWM Signal Generation \(DS2202 Features\)](#).

Parameters

base PHS-bus base address of the DS2202 board

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Description
DS2202_PWMOUT_CH1	PWM channel 1
...	...
DS2202_PWMOUT_CH9	PWM channel 9

period PWM period in seconds. The values should remain within the selected period range (refer to `ds2202_timing_out_mode_set`). For information on PWM signal generation and its restrictions, refer to [PWM Signal Generation \(DS2202 Features !\[\]\(d0a1791f26d167e866e44ebbf83efebe_img.jpg\)](#)).

duty PWM duty cycle within the range 0 ... 1.0. The following table shows the relation to the duty cycle given in percent:

Range	Duty Cycle
0 ... 1.0	0 ... 100%

Return value None

Execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

Examples

[Example of PWM Signal Generation.....](#) 59

References

[ds2202_digout_mode_set.....](#) 12
[ds2202_timing_out_mode_set.....](#) 53

PWM Signal Measurement

Introduction

The PWM signal measurement functions let you analyze PWM signals.

Where to go from here

Information in this section

[Example of PWM Signal Measurement](#)..... 63

The example measures a PWM signal in the frequency range of 100 Hz ... 10 kHz.

[ds2202_pwm_in](#)..... 65

To read the period and duty cycle from the specified PWM input channel.

Information in other sections

[PWM Signal Measurement \(DS2202 Features\)](#) 

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

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

[ds2202_timing_in_mode_set](#)..... 56

To set the input mode and range to PWM signal or frequency measurement.

Example of PWM Signal Measurement

Introduction

The example measures a PWM signal in the frequency range of 100 Hz ... 10 kHz.

Example

```

1 #include <Brtenv.h>           /* basic real-time environment */
2 #include <ds2202.h>
3
4 /*-----*/
5
6 #define DT 1e-3               /* 1 ms simulation step size */
7
8 dsfloat in_duty;              /* measured duty cycle */
9 dsfloat in_period;           /* measured period */
10
11 /* variables for execution time profiling */

```

```

12 dsfloat exec_time;                                /* execution time */
13 /*-----*/
14
15 void isr_t1()                                     /* timer1 interrupt service routine */
16 {
17     ts_timestamp_type ts;
18     RTLIB_SRT_ISR_BEGIN();                        /* overload check */
19     RTLIB_TIC_START();                            /* start time measurement */
20     ts_timestamp_read(&ts);
21     host_service(1, &ts);                        /* data acquisition service */
22     /* read PWM input channel 1 */
23     ds2202_pwm_in(DS2202_1_BASE, DS2202_PWMIN_CH1,
24                   &in_period, &in_duty);
25
26     exec_time = RTLIB_TIC_READ(); /* calculate execution time*/
27
28     RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
29 }
30 /*-----*/
31 void main()
32 {
33
34     init();                                       /* initialize hardware system */
35
36     ds2202_init(DS2202_1_BASE); /* initialize DS2202 board */
37
38     msg_info_set(MSG_SM_RTLIB, 0, "System started.");
39
40     /* set PWM range for expected signal periods
41     on input channel 1 */
42     ds2202_timing_in_mode_set(DS2202_1_BASE,
43                               DS2202_PWMIN_CH1,
44                               DS2202_TIMING_RANGE5,
45                               DS2202_PWM2D);
46
47     /* initialize sampling clock timer */
48     RTLIB_SRT_START(DT, isr_t1);
49
50     RTLIB_TIC_INIT();
51
52     while(1)                                     /* background process */
53     {
54         RTLIB_BACKGROUND_SERVICE();
55
56     } /* while(1) */
57
58 } /* main() */

```

Related topics

References

ds2202_pwm_in.....65

ds2202_pwm_in

Syntax

```
void ds2202_pwm_in(
    Int32 base,
    Int32 channel,
    dsfloat *period,
    dsfloat *duty)
```

Include file

ds2202.h

Purpose

To capture the PWM period and duty cycle of the specified PWM input channel.

I/O mapping

For information on the I/O mapping, refer to [PWM Signal Measurement \(DS2202 Features\)](#).

Note

- After initialization, the input threshold is set to 2.5 V.
- To minimize the quantization effect on the frequency resolution and duty cycle, you should select the smallest possible frequency range. For detailed information, refer to [PWM Signal Measurement \(DS2202 Features\)](#).
- The PWM input channels 1 ... 8 are shared with DIG_IN17 ... 24. The PWM input channels 9 ... 24 are shared with DIG_IN1 ... 16.

Parameters

base PHS-bus base address of the DS2202 board

channel PWM input channel. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_PWMIN_CH1	PWM input channel 1
...	...
DS2202_PWMIN_CH24	PWM input channel 24

period Address where the measured period is written. The value is given in seconds.

duty Address where the measured duty cycle is written. The duty cycle is scaled to the range 0 ... 1.0. The following table shows the relation to the duty cycle given in percent:

Range	Duty Cycle
0 ... 1.0	0 ... 100%

Return value	None
Execution times	For information, refer to Function Execution Times on page 207.
Related topics	<div>Examples<div>Example of PWM Signal Measurement..... 63</div></div> <div>References<div>ds2202_digin_threshold_set..... 11</div><div>ds2202_timing_in_mode_set..... 56</div></div>


Square-Wave Signal Generation

Where to go from here

Information in this section

Example of Square-Wave Signal Generation.....	67
The example shows how to generate a square-wave signal.	
ds2202_d2f.....	69
To generate a square-wave signal.	

Information in other sections

Square-Wave Signal Generation (D2F) (DS2202 Features )	
The DS2202 provides 9 independent output channels for the generation of square-wave signals with variable frequencies.	
ds2202_timing_out_mode_set.....	53
To set the output mode and range to PWM or square-wave signal generation.	

Example of Square-Wave Signal Generation

Introduction

This example shows how to generate a square-wave signal.

Example

```

1  #include <Brtenv.h>           /* basic real-time environment */
2  #include <Ds2202.h>
3
4  /*-----*/
5
6  #define DT 1e-3               /* 1 ms simulation step size */
7
8  dsfloat frequency = 1000;     /* set output frequency to 1kHz */
9
10 /* variables for execution time profiling */
11 dsfloat exec_time;           /* execution time */
12 /*-----*/
13
14 void isr_t1()                 /* timer1 interrupt service routine */
15 {
16     ts_timestamp_type ts;
17     RTLIB_SRT_ISR_BEGIN();    /* overLoad check */
18     RTLIB_TIC_START();        /* start time measurement */

```

```

19  ts_timestamp_read(&ts);
20  host_service(1, &ts);      /* data acquisition service */
21  /* update output frequency */
22  ds2202_d2f(DS2202_1_BASE, DS2202_PWMOUT_CH1, frequency);
23
24  exec_time = RTLIB_TIC_READ(); /* calc. execution time */
25
26  RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
27 }
28 /*-----*/
29
30 void main()
31 {
32
33     init();                /* initialize hardware system */
34
35     ds2202_init(DS2202_1_BASE); /* initialize DS2202 board */
36
37     msg_info_set(MSG_SM_RTLIB, 0, "System started.");
38
39     /* enable digital output driver */
40     ds2202_digout_mode_set(DS2202_1_BASE,
41                             DS2202_DIGOUT_ENABLE);
42
43     /* set frequency range for output channel 1,
44        output(0Hz) = Low */
45     ds2202_timing_out_mode_set(DS2202_1_BASE,
46                                 DS2202_PWMOUT_CH1,
47                                 DS2202_TIMING_RANGE5,
48                                 DS2202_D2F_LOW);
49
50     /* set values for frequency generation */
51     ds2202_d2f(DS2202_1_BASE, DS2202_PWMOUT_CH1, frequency);
52
53     /* initialize sampling clock timer */
54     RTLIB_SRT_START(DT, isr_t1);
55     RTLIB_TIC_INIT();
56
57     while(1)                /* background process */
58     {
59         RTLIB_BACKGROUND_SERVICE();
60
61     } /* while(1) */
62
63 } /* main() */

```

Related topics

Basics

[ds2202_d2f.....](#) 69

ds2202_d2f

Syntax

```
void ds2202_d2f(
    Int32 base,
    Int32 channel,
    dsfloat frequency)
```

Include file

ds2202.h

Purpose

To generate a square-wave signal with a specified frequency.

I/O mapping

For information on the I/O mapping, refer to [Square-Wave Signal Generation \(D2F\) \(DS2202 Features !\[\]\(8bba887393ca45b761e5cb49e755e762_img.jpg\)](#)).

Note

- All digital outputs are high-Z after reset. Outputs are enabled using the ds2202_digout_mode_set function.
- To minimize the quantization effect on the frequency resolution, you should select the smallest possible frequency range. For detailed information, refer to [Square-Wave Signal Generation \(D2F\) \(DS2202 Features !\[\]\(c44db1e92ba1244b2894d325c806ff8a_img.jpg\)](#)).

Description

The function outputs a digital signal with the specified frequency on the appropriate output channel. The resolution of the frequency signal is 21 bit and depends on the selected prescaler setting. For information on the available range, refer to [Square-Wave Signal Generation \(D2F\) \(DS2202 Features !\[\]\(0fb13ad0bfa3d86868cdd3883e5665b3_img.jpg\)](#)).

The frequency ranges can be set using the ds2202_timing_out_mode_set function.

Parameters

base PHS-bus base address of the DS2202 board

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_PWMOUT_CH1	PWM channel 1
...	...
DS2202_PWMOUT_CH9	PWM channel 9

frequency Frequency of the generated signal in Hz

Return value	None
Execution times	For information, refer to Function Execution Times on page 207.
Related topics	<div>Examples<div>Example of Square-Wave Signal Generation..... 67</div></div> <div>References<div>ds2202_digout_mode_set..... 12</div><div>ds2202_timing_out_mode_set..... 53</div></div>

Frequency Measurement

Introduction


The frequency measurement functions let you measure the frequency of square-wave signals.

Where to go from here

Information in this section

Example of Frequency Measurement.....	71
The example shows how to measure the frequency of an input signal.	
ds2202_f2d.....	73
To measure the frequency of a square-wave signal.	

Information in other sections

Frequency Measurement (F2D) (DS2202 Features )	
The DS2202 provides 24 independent input channels for the frequency measurement of square-wave signals.	
ds2202_timing_in_mode_set.....	56
To set the input mode and range to PWM signal or frequency measurement.	

Example of Frequency Measurement

Introduction

This example shows how to measure the frequency of an input signal.

Example

```

1  #include <Brtenv.h>           /* basic real-time environment */
2  #include <Ds2202.h>
3
4  /*-----*/
5
6  #define DT 1e-3               /* 1 ms simulation step size */
7
8  dsfloat frequency;           /* input frequency */
9
10 /* variables for execution time profiling */
11 dsfloat exec_time;           /* execution time */
12
13 /*-----*/
14

```

```

15 void isr_t1()          /* timer1 interrupt service routine */
16 {
17     ts_timestamp_type ts;
18     RTLIB_SRT_ISR_BEGIN();          /* overload check */
19     RTLIB_TIC_START();              /* start time measurement */
20     ts_timestamp_read(&ts);
21     host_service(1, &ts);          /* data acquisition service */
22     /* read signal frequency from input 1 */
23     ds2202_f2d(DS2202_1_BASE, DS2202_PWMOUT_CH1, &frequency);
24
25     exec_time = RTLIB_TIC_READ(); /* calculate execution time*/
26
27     RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
28 }
29
30 /*-----*/
31 void main()
32 {
33
34     init();          /* initialize hardware system */
35
36     ds2202_init(DS2202_1_BASE); /* initialize DS2202 board */
37
38     msg_info_set(MSG_SM_RTLIB, 0, "System started.");
39
40     /* set frequency range for input channel 1 */
41     ds2202_timing_in_mode_set(DS2202_1_BASE,
42                               DS2202_PWMIN_CH1,
43                               DS2202_TIMING_RANGE5,
44                               DS2202_F2D);
45
46     /* initialize sampling clock timer */
47     RTLIB_SRT_START(DT, isr_t1);
48     RTLIB_TIC_INIT();
49
50     while(1)          /* background process */
51     {
52
53         RTLIB_BACKGROUND_SERVICE();
54
55     } /* while(1) */
56
57 } /* main() */

```

Related topics

References

ds2202_f2d..... 73

ds2202_f2d

Syntax

```
void ds2202_f2d(
    Int32 base,
    Int32 channel,
    dsfloat* frequency)
```

Include file

ds2202.h

Purpose

To measure the frequency of a square-wave signal.

I/O mapping

For information on the I/O mapping, refer to [Frequency Measurement \(F2D\) \(DS2202 Features !\[\]\(faf942dc3e59ce8eb64b4ac481eca7e0_img.jpg\)](#)).

Note

- After initialization, the input threshold is set to 2.5 V.
- To minimize the quantization effect on the frequency resolution, you should select the smallest possible frequency range. For detailed information, refer to [Frequency Measurement \(F2D\) \(DS2202 Features !\[\]\(564903337f30b845a5f6979939a95fe6_img.jpg\)](#)).

Description

The function measures the signal frequency of the specified input channel. The frequency value is scaled to Hz and written to the memory at the address specified by the **frequency** parameter. The resolution of the frequency signal is 21 bit and depends on the selected prescaler setting. For information on the available range, refer to [Frequency Measurement \(F2D\) \(DS2202 Features !\[\]\(95b425611cbd2b8716a140cf67c81822_img.jpg\)](#)).

The frequency ranges can be set using the `ds2202_timing_in_mode_set` function.

Parameters

base PHS-bus base address of the DS2202 board

channel PWM input channel. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_PWMIN_CH1	PWM input channel 1
...	...
DS2202_PWMIN_CH24	PWM input channel 24

frequency Address where frequency is written (Hz)

Return value	None
Execution times	For information, refer to Function Execution Times on page 207.
Related topics	<div>Examples<div>Example of Frequency Measurement..... 71</div></div> <div>References<div>ds2202_digin_threshold_set..... 11</div><div>ds2202_timing_in_mode_set..... 56</div></div>

Serial Interface Communication

Introduction	The serial interface contains generic functions for serial communication.
---------------------	---

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

Basic Principles of Serial Communication.....	76
Information on serial communication using generic RTLib functions.	
Data Types for Serial Communication.....	81
There are some specific data structures specified for the serial communication interface.	
Generic Serial Interface Communication Functions.....	89
Information on the generic functions for serial interface communication.	


Basic Principles of Serial Communication

Introduction	Information on serial communication using generic RTLib functions.
---------------------	--

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

Trigger Levels.....	76
You get information about the trigger levels.	
How to Handle Subinterrupts in Serial Communication.....	77
Subinterrupt handling in serial communication.	
How to Write Portable Applications.....	78
You get information about writing applications for different processor platforms.	
Example of Serial Interface Communication.....	79
The example shows how to initialize the serial interface.	

Information in other sections

Serial Interface (DS2202 Features )
The board contains a universal asynchronous receiver and transmitter (UART) to communicate with external devices.

Trigger Levels

Introduction	Two different trigger levels can be configured.
---------------------	---

UART trigger level	The UART trigger level is hardware dependent. After the specified number of bytes is received, the UART generates an interrupt and the bytes are copied into the receive buffer.
---------------------------	--

User trigger level	The user trigger level is hardware independent and can be adjusted in smaller steps than the UART trigger level. After a specified number of bytes is received in the receive buffer, the subinterrupt handler is called.
---------------------------	---

Related topics**HowTos**

[How to Handle Subinterrupts in Serial Communication..... 77](#)

How to Handle Subinterrupts in Serial Communication

Objective

Subinterrupt handling in serial communication.

Available subinterrupts

The following subinterrupts can be passed to your application:

Subinterrupt	Meaning
DSSER_TRIGGER_LEVEL_SUBINT	Generated when the receive buffer is filled with the number of bytes specified as the trigger level (see Trigger Levels on page 76)
DSSER_TX_FIFO_EMPTY_SUBINT	Generated when the transmit buffer has no data
DSSER_RECEIVER_LINE_SUBINT	Line status interrupt provided by the UART
DSSER_MODEM_STATE_SUBINT	Modem status interrupt provided by the UART
DSSER_NO_SUBINT	Generated after the last subinterrupt. This subinterrupt tells your application that no further subinterrupts were generated.

Restrictions

You can use interrupt functions only in handcoded applications. Using them in Simulink applications (User-Code or S-functions) conflicts with the internal interrupt handling.

Method**To install a subinterrupt handler within your application**

- 1 Write a function that handles your subinterrupt, such as:

```
void my_subint_handler(dsserChannel* serCh, Int32 subint)
{
    switch (subint)
    {
        case DSSER_TRIGGER_LEVEL_SUBINT:
            /* do something */
            break;
        case DSSER_TX_FIFO_EMPTY_SUBINT:
            /* do something */
            break;
        case DSSER_NO_SUBINT:
            /* no further subinterrupts */
            break;
        default:
            break;
    }
}
```

- 2 Initialize your subinterrupt handler:

```
dsser_subint_handler_inst(serCh,
    (dsser_subint_handler_t) my_subint_handler);
```

- 3 Enable the required subinterrupts:

```
dsser_subint_enable(serCh,
    DSSER_TRIGGER_LEVEL_SUBINT_MASK |
    DSSER_TX_FIFO_EMPTY_SUBINT_MASK);
```

Related topics**Basics**

Trigger Levels..... 76

How to Write Portable Applications

Objective

If your application has to run on different processor boards, you must consider the available data types.

Method**To make an application data type compatible**

- 1 Use the functions `dsser_word2bytes` and `dsser_bytes2word` to convert the parameter into the required format.

Related topics**References**

ds2202_bytes2word	111
ds2202_word2bytes	109

Example of Serial Interface Communication

Introduction

The example shows how to initialize the serial interface.

The serial interface is initialized with 9600 baud, 8 data bits, 1 stop bit and no parity. The receiver FIFO generates a subinterrupt when 32 bytes were received and the subinterrupt handler `callback` is called. The subinterrupt handler `callback` reads the received bytes and sends the bytes back immediately.

Example

```

1  #include <brtENV.h>
2  #include <ds2202.h>
3  void callback(ds2202Channel* serCh, UInt32 subint)
4  {
5      UInt32 count;
6      UInt8 data[32];
7      switch (subint)
8      {
9          case DSSER_TRIGGER_LEVEL_SUBINT:
10             msg_info_set(0,0,"DSSER_TRIGGER_LEVEL_SUBINT");
11             ds2202_receive(serCh,32,data,&count);
12             ds2202_transmit(serCh,count,data,&count);
13             break;
14             case DSSER_TX_FIFO_EMPTY_SUBINT:
15                 msg_info_set(0,0,"DSSER_TX_FIFO_EMPTY_SUBINT");
16                 break;
17             default:
18                 break;
19         }
20     }
21     main()
22     {
23         ds2202Channel* serCh;
24         init();
25         ds2202_init(DS2202_1_BASE);
26
27         /* allocate a new 1024 byte SW-FIFO */
28         serCh = ds2202_init(DS2202_1_BASE, 0, 1024);
29         ds2202_subint_handler_inst(serCh,
30             (ds2202_subint_handler_t)callback);

```

```
31 dsser_subint_enable(serCh,  
32     DSSER_TRIGGER_LEVEL_SUBINT_MASK |  
33     DSSER_TX_FIFO_EMPTY_SUBINT_MASK);  
34  
35 /* config and start the UART */  
36 dsser_config(serCh, DSSER_FIFO_MODE_OVERWRITE,  
37     9600, 8, DSSER_1_STOPBIT, DSSER_NO_PARITY,  
38     DSSER_14_BYTE_TRIGGER_LEVEL, 32,  
39     DSSER_RS232 | DSSER_AUTOFLOW_DISABLE);  
40 RTLIB_INT_ENABLE();  
41 for(;;)  
42 {  
43     RTLIB_BACKGROUND_SERVICE();  
44 }  
45 }
```

Related topics

Basics

[Trigger Levels.....](#) 76

HowTos

[How to Handle Subinterrupts in Serial Communication.....](#) 77

[How to Write Portable Applications.....](#) 78

Data Types for Serial Communication

Introduction There are some specific data structures specified for the serial communication interface.

Where to go from here	Information in this section
	dsser_ISR.....81 To provide information about the interrupt identification register (IIR).
	dsser_LSR.....83 To provide information about the status of data transfers.
	dsser_MSR.....84 To provide information about the state of the control lines.
	dsser_subint_handler_t.....85 This type definition is required for installing a subinterrupt handler.
	dsserChannel.....87 To provide information about the serial channel.

dsser_ISR

Syntax

```
typedef union
{
    UInt32    Byte;
    struct
    {
        unsigned dummy : 24;
        unsigned DSSER_FIFO_STATUS_BIT1 : 1;
        unsigned DSSER_FIFO_STATUS_BIT0 : 1;
        unsigned DSSER_BIT5 : 1;
        unsigned DSSER_BIT4 : 1;
        unsigned DSSER_INT_PRIORITY_BIT2 : 1;
        unsigned DSSER_INT_PRIORITY_BIT1 : 1;
        unsigned DSSER_INT_PRIORITY_BIT0 : 1;
        unsigned DSSER_INT_STATUS : 1;
    }Bit;
}dsser_ISR;
```

Include file dsserdef.h

Description

The `dsser_ISR` structure provides information about the interrupt identification register (IIR). Call `dsser_status_read` to read the status register.

The `dsser_ISR`, `dsser_LSR`, and `dsser_MSR` data types contain the values of the UART's register. For further information, refer to the UART's datasheet: *TEXAS INSTRUMENTS, TL16C550C, Asynchronous Communications Element with Autoflow Control, SLLS177C - March 1994 - Revised March 1997*

Members

DSSER_INT_STATUS 0 if interrupt pending

DSSER_INT_PRIORITY_BIT0 Interrupt ID bit 1

DSSER_INT_PRIORITY_BIT1 Interrupt ID bit 2

DSSER_INT_PRIORITY_BIT2 Interrupt ID bit 3

DSSER_BIT4 Not relevant

DSSER_BIT5 Not relevant

DSSER_FIFO_STATUS_BIT0 UART FIFOs enabled

DSSER_FIFO_STATUS_BIT1 UART FIFOs enabled

For more information about the predefined constants, refer to the datasheet of the *TEXAS INSTRUMENTS, TL16C550C*.

Related topics**References**

[dsser_status_read..... 103](#)

dsser_LSR

Syntax

```
typedef union
{
    UInt32 Byte;
    struct
    {
        unsigned dummy : 24;
        unsigned DSSER_FIFO_DATA_ERR : 1;
        unsigned DSSER_THR_TSR_STATUS : 1;
        unsigned DSSER_THR_STATUS : 1;
        unsigned DSSER_BREAK_STATUS : 1;
        unsigned DSSER_FRAMING_ERR : 1;
        unsigned DSSER_PARITY_ERR : 1;
        unsigned DSSER_OVERRUN_ERR : 1;
        unsigned DSSER_RECEIVE_DATA_RDY : 1;
    }Bit;
} dsser_LSR;
```

Include file

dsserdef.h

Description

The `dsser_LSR` structure provides information about the status of data transfers. Call `dsser_status_read` to read the status register.

The `dsser_ISR`, `dsser_LSR`, and `dsser_MSR` data types contain the values of the UART's register. For further information, refer to the UART's datasheet: *TEXAS INSTRUMENTS, TL16C550C, Asynchronous Communications Element with Autoflow Control, SLLS177C - March 1994 - Revised March 1997*

Members

DSSER_RECEIVE_DATA_RDY Data ready (DR) indicator

DSSER_OVERRUN_ERR Overrun error (OE) indicator

DSSER_PARITY_ERR Parity error (PE) indicator

DSSER_FRAMING_ERR Framing error (FE) indicator

DSSER_BREAK_STATUS Break interrupt (BI) indicator

DSSER_THR_STATUS Transmitter holding register empty (THRE)

DSSER_THR_TSR_STATUS Transmitter empty (TEMT) indicator

DSSER_FIFO_DATA_ERR Error in receiver FIFO

For more information about the predefined constants, refer to the datasheet of the *TEXAS INSTRUMENTS, TL16C550C*.

Related topics**References**

[dsser_status_read](#)..... 103

dsser_MSR

Syntax

```
typedef union
{
    UInt32 Byte;
    struct
    {
        unsigned dummy : 24;
        unsigned DSSER_OP2_STATUS : 1;
        unsigned DSSER_OP1_STATUS : 1;
        unsigned DSSER_DTR_STATUS : 1;
        unsigned DSSER_RTS_STATUS : 1;
        unsigned DSSER_CD_STATUS : 1;
        unsigned DSSER_RI_STATUS : 1;
        unsigned DSSER_DSR_STATUS : 1;
        unsigned DSSER_CTS_STATUS : 1;
    }Bit;
}dsser_MSR;
```

Include file

dsserdef.h

Description

The **dsser_MSR** structure provides information about the state of the control lines. Call **dsser_status_read** to read the status register.

The **dsser_ISR**, **dsser_LSR**, and **dsser_MSR** data types contain the values of the UART's register. For further information, refer to the UART's datasheet: *TEXAS INSTRUMENTS, TL16C550C, Asynchronous Communications Element with Autoflow Control, SLLS177C - March 1994 - Revised March 1997*

Members	DSSER_CTS_STATUS	Clear-to-send (CTS) changed state
	DSSER_DSR_STATUS	Data-set-ready (DSR) changed state
	DSSER_RI_STATUS	Ring-indicator (RI) changed state
	DSSER_CD_STATUS	Data-carrier-detect (CD) changed state
	DSSER_RTS_STATUS	Complement of CTS
	DSSER_DTR_STATUS	Complement of DSR
	DSSER_OP1_STATUS	Complement of RI
	DSSER_OP2_STATUS	Complement of DCD

For more information about the predefined constants, refer to the datasheet of the *TEXAS INSTRUMENTS, TL16C550C*.

Related topics**References**

[dsser_status_read](#)..... 103

dsser_subint_handler_t

Syntax

```
typedef void (*dsser_subint_handler_t)
(void* serCh, Int32 subint)
```

Include file

dsserdef.h

Description

You must use this type definition if you install a subinterrupt handler (see [How to Handle Subinterrupts in Serial Communication](#) or [dsser_subint_handler_inst](#)).

Members

serCh Pointer to the serial channel structure (see [dsser_init](#))

subint Identification number of the related subinterrupt. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_TRIGGER_LEVEL_SUBINT	Interrupt triggered when the user trigger level is reached (see Trigger Levels on page 76)
DSSER_TX_FIFO_EMPTY_SUBINT	Interrupt triggered when the transmit buffer is empty
DSSER_RECEIVER_LINE_SUBINT	Line status interrupt of the UART

Predefined Symbol	Meaning
DSSER_MODEM_STATE_SUBINT	Modem status interrupt of the UART
DSSER_NO_SUBINT	Flag that is sent after the last triggered subinterrupt

Related topics

HowTos

How to Handle Subinterrupts in Serial Communication.....	77
--	--------------------

References

dsrser_init.....	90
dsrser_subint_handler_inst.....	106

dsserChannel

Syntax

```
typedef struct
{
    /*--- public -----*/
    /* interrupt status register */
    dsser_ISR intStatusReg;
    /* line status register */
    dsser_LSR lineStatusReg;
    /* modem status register */
    dsser_MSR modemStatusReg;

    /*--- protected -----*/

    /*--- serial channel allocation ---*/
    UInt32 module;
    UInt32 channel;
    Int32 board_bt;
    UInt32 board;
    UInt32 fifo_size;
    UInt32 frequency;

    /*--- serial channel configuration ---*/
    UInt32 baudrate;
    UInt32 databits;
    UInt32 stopbits;
    UInt32 parity;
    UInt32 rs_mode;
    UInt32 fifo_mode;
    UInt32 uart_trigger_level;
    UInt32 user_trigger_level;
    dsser_subint_handler_t subint_handler;
    dsserService* serService;
    dsfifo_t* txFifo;
    dsfifo_t* rxFifo;
    UInt32 queue;
    UInt8 isr;
    UInt8 lsr;
    UInt8 msr;
    UInt32 interrupt_mode;
    UInt8 subint_mask;
    Int8 subint;
}dsserChannel
```

Include file

dsserdef.h

Description

This structure provides information about the serial channel. You can call `dsser_status_read` to read the values of the status registers. All protected variables are only for internal use.

Members	intStatusReg	Interrupt status register (see <code>dsser_ISR</code>)
	lineStatusReg	Line status register (see <code>dsser_LSR</code>)
	modemStatusReg	Modem status register (see <code>dsser_MSR</code>)
Related topics	References	
	dsser_status_read..... 103	

Generic Serial Interface Communication Functions

Introduction

Information on the generic functions for serial interface communication.

Where to go from here

Information in this section

dsser_init	90
To initialize the serial interface.	
dsser_config	91
To configure the serial interface.	
dsser_transmit	94
To transmit data with the serial interface.	
dsser_receive	95
To receive data via the serial interface.	
dsser_receive_term	97
To receive data until a termination character is received.	
dsser_fifo_reset	98
To reset the serial interface.	
dsser_enable	99
To enable the serial interface.	
dsser_disable	100
To disable the serial interface.	
dsser_error_read	101
To read the error flag of the serial interface.	
dsser_transmit_fifo_level	102
To get the number of bytes in the transmit buffer.	
dsser_receive_fifo_level	103
To get the number of bytes in the receive buffer.	
dsser_status_read	103
To get the values of status registers.	
dsser_set	104
To set a new quartz frequency.	
dsser_subint_handler_inst	106
To install a subinterrupt handler.	
dsser_subint_enable	107
To enable one or several subinterrupts.	
dsser_subint_disable	108
To disable one or several subinterrupts.	

dsser_word2bytes.....	109
To convert a word (max. 4 bytes long) into a byte array.	
dsser_bytes2word.....	111
To convert a byte array with a maximum of 4 elements into a single word.	

dsser_init

Syntax

```
dsserChannel* dsser_init(
    UInt32 base,
    UInt32 channel,
    UInt32 fifo_size)
```

Include file

`dsser.h`

Purpose

To initialize the serial interface and install the interrupt handler.

Note

Pay attention to the initialization sequence: Initialize the processor board before the I/O boards, and the serial interface at the end.

Parameters

base Specifies the base address of the serial interface. This value has to be set to DS2202_y_BASE, with y as a consecutive number within the range of 1 ... 16. For example, if there is only one DS2202 board use DS2202_1_BASE.

channel Specifies the number of the channel that should be used for the serial interface. The permitted value is 0.

fifo_size Specifies the size of the transmit and receive buffer in bytes. The size must be a power of two (2^n) and at least 64 bytes great. The maximum size depends on the available memory.

Return value

This function returns the pointer to the serial channel structure.

Messages

The following messages are defined (x = base address of the I/O board, y = number of the channel):

ID	Type	Message	Description
100	Error	x, ch=y, Board not found!	I/O board was not found.
501	Error	x, ch=y, memory: Allocation error on master.	Memory allocation error. No free memory on the master.
508	Error	x, ch=y, channel: out of range!	The <code>channel</code> parameter is out of range.
700	Error	x, ch=y, Buffersize: Illegal	The <code>fifo_size</code> parameter is out of range.

Related topics**Basics**

[Basic Principles of Serial Communication..... 76](#)

References

[Data Types for Serial Communication..... 81](#)
[ds_ser_config..... 91](#)

ds_ser_config

Syntax

```
void ds_ser_config(
    ds_serChannel* serCh,
    const UInt32 fifo_mode,
    const UInt32 baudrate,
    const UInt32 databits,
    const UInt32 stopbits,
    const UInt32 parity,
    const UInt32 uart_trigger_level,
    const Int32 user_trigger_level,
    const UInt32 uart_mode)
```

Include file

`ds_ser.h`

Purpose

To configure and start the serial interface.

Note

- This function starts the serial interface, so all dSPACE real-time boards must be initialized and the interrupt vector must be installed before calling this function.
- Calling this function again, reconfigures the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see `dsser_init`).

fifo_mode Specifies the mode of the receive buffer (see [Serial Interface \(DS2202 Features !\[\]\(dfbd6b3763a6d1d9afaa974f64e2e4b5_img.jpg\)](#))):

Value	Mode	Meaning
DSSER_FIFO_MODE_BLOCKED	Blocked mode	If the receive buffer is full, new data is rejected.
DSSER_FIFO_MODE_OVERWRITE	Overwrite mode	If the receive buffer is full, new data replaces the oldest data in the buffer.

baudrate Specifies the baud rate in bits per second:

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

For further information, refer to [Specifying the Baud Rate of the Serial Interface \(DS2202 Features !\[\]\(ec9132f1d27c8919987d92907322654d_img.jpg\)](#))

databits Specifies the number of data bits. Values are: 5, 6, 7, 8

stopbits Specifies the number of stop bits. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_1_STOPBIT	1 stop bit
DSSER_2_STOPBIT	The number of stop bits depends on the number of the specified data bits: 5 data bits: 1.5 stop bits 6 data bits: 2 stop bits 7 data bits: 2 stop bits 8 data bits: 2 stop bits

parity Specifies whether and how parity bits are generated. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_PARITY	No parity bits
DSSER_ODD_PARITY	Parity bit is set so that there is an odd number of "1" bits in the byte, including the parity bit
DSSER_EVEN_PARITY	Parity bit is set so that there is an even number of "1" bits in the byte, including the parity bit
DSSER_FORCED_PARITY_ONE	Parity bit is forced to a logic 1
DSSER_FORCED_PARITY_ZERO	Parity bit is forced to a logic 0

uart_trigger_level Sets the UART trigger level (see [Trigger Levels](#) on page 76). The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_1_BYTE_TRIGGER_LEVEL	1-byte trigger level
DSSER_4_BYTE_TRIGGER_LEVEL	4-byte trigger level
DSSER_8_BYTE_TRIGGER_LEVEL	8-byte trigger level
DSSER_14_BYTE_TRIGGER_LEVEL	14-byte trigger level

Note

Use the highest UART trigger level possible to generate fewer interrupts.

user_trigger_level Sets the user trigger level within the range of 1 ... (fifo_size-1) for the receive interrupt (see `dsser_init` and [Trigger Levels](#) on page 76):

Value	Meaning
DSSER_DEFAULT_TRIGGER_LEVEL	Synchronizes the UART trigger level and the user trigger level
1 ... (fifo_size-1)	Sets the user trigger level
DSSER_TRIGGER_LEVEL_DISABLE	No receive subinterrupt handling for the serial interface

uart_mode Sets the mode of the UART transceiver.

The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_RS232	RS232 mode
DSSER_RS422	RS422 mode

Messages

The following messages are defined (x = base address of the I/O board, y = number of the channel):

ID	Type	Message	Description
601	Error	x, serCh: The UART channel was not initialized.	The <code>dsser_config</code> function was called before the serial interface was initialized with <code>dsser_init</code> .
602	Error	x, ch=y, baudrate: Illegal!	The <code>baudrate</code> parameter is out of range.
603	Error	x, ch=y, databits: Use range 5 ... 8 bits!	The <code>databits</code> parameter is out of range.
604	Error	x, ch=y, stopbits: Illegal number (1-2 bits allowed)!	The <code>stopbits</code> parameter is out of range.
605	Error	x, ch=y, parity: Illegal parity!	The <code>parity</code> parameter is out of range.
606	Error	x, ch=y, trigger_level: Illegal UART trigger level!	The <code>uart_trigger_level</code> parameter is out of range.

ID	Type	Message	Description
607	Error	x, ch=y, trigger_level: Illegal user trigger level!	The <code>user_trigger_level</code> parameter is out of range.
608	Error	x, ch=y, fifo_mode: Use range 0 ... (fifo_size-1) bytes!	The <code>uart_mode</code> parameter is out of range.
609	Error	x, ch=y, uart_mode: Transceiver not supported!	The selected UART mode does not exist for this serial interface.
611	Error	x, ch=y, uart_mode: Autoflow is not supported!	Autoflow does not exist for this serial interface.

Related topics**Basics**

[Basic Principles of Serial Communication..... 76](#)

References

[dsser_init..... 90](#)

dsser_transmit

Syntax

```
Int32 dsser_transmit(
    dsserChannel* serCh,
    UInt32 datalen,
    UInt8* data,
    UInt32* count)
```

Include file

`dsser.h`

Purpose

To transmit data through the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see `dsser_init`).

datalen Specifies the number of bytes to be transmitted.

data Specifies the pointer to the data to be transmitted.

count Specifies the pointer to the number of transmitted bytes. When this function is finished, the variable contains the number of bytes that were transmitted. If the function could send all the data, the value is equal to the value of the `datalen` parameter.

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_FIFO_OVERFLOW	The FIFO is filled or not all the data could be copied into the FIFO.
DSSER_COMMUNICATION_FAILED	The function failed with no effect on the input or output data. No data is written to the FIFO. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.

Example This example shows how to check the transmit buffer for sufficient free memory before transmitting data.

```
UInt32 count;
UInt8 block[5] = {1, 2, 3, 4, 5};
if(dsser_transmit_fifo_level(serCh) < serCh->fifo_size - 5)
{
    dsser_transmit(serCh, 5, block, &count);
}
```

Related topics

Basics

[Basic Principles of Serial Communication..... 76](#)

HowTos

[How to Write Portable Applications..... 78](#)

References

[dsser_init..... 90](#)

dsser_receive

Syntax

```
Int32 dsser_receive(
    dsserChannel* serCh,
    UInt32 datalen,
    UInt8* data,
    UInt32* count)
```

Include file

dsser.h

Purpose

To receive data through the serial interface.

Tip

It is better to receive a block of bytes instead of several single bytes because the processing speed is faster.

Parameters

serCh Specifies the pointer to the serial channel structure (see `dsser_init`).

datalen Specifies the number of data bytes that should be read. The value must not be greater than the fifo size defined with `dsser_init`.

data Specifies the pointer to the destination buffer.

count Specifies the pointer to the number of received bytes. When this function is finished the variable contains the number of bytes that were received.

Return value

This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_NO_DATA	No new data is read from the FIFO.
DSSER_FIFO_OVERFLOW	The FIFO is filled, the behavior depends on the <code>fifo_mode</code> adjusted with <code>dsser_config</code> : <code>fifo_mode = DSSER_FIFO_MODE_BLOCKED</code> : Not all new data could be placed into the FIFO. <code>fifo_mode = DSSER_FIFO_MODE_OVERWRITE</code> : The old data is rejected.
DSSER_COMMUNICATION_FAILED	The function failed with no effect on the input or output data. No data is read from the FIFO. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.

Example

The following example shows, how to receive 4 bytes.

```
UInt8 data[4];
UInt32 count;
Int32 error;
/* receive four bytes over serCh */
error = dsser_receive(serCh, 4, data, &count);
```


Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 76

HowTos

[How to Write Portable Applications.....](#) 78

References

[ds-ser_config.....](#) 91

[ds-ser_init.....](#) 90

ds-ser_receive_term

Syntax

```
Int32 ds-ser_receive_term(
    ds-serChannel* serCh,
    UInt32 datalen,
    UInt8* data,
    UInt32* count,
    const UInt8 term)
```

Include file

`ds-ser.h`

Purpose

To receive data through the serial interface.

Description

This is terminated when the character `term` is received. The character `term` is stored as the last character in the buffer, so you can check if the function was successful.

Parameters

serCh Specifies the pointer to the serial channel structure (see `ds-ser_init`).

datalen Specifies the number of data bytes that should be read. The value must not be greater than the fifo size defined with `ds-ser_init`.

data Specifies the pointer to the destination buffer.

count Specifies the pointer to the number of received bytes. When this function is finished the variable contains the number of bytes that were received.

term Specifies the character that terminates the receiving.

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_NO_DATA	No new data is read from the FIFO.
DSSER_FIFO_OVERFLOW	The FIFO is filled; the behavior depends on the <code>fifo_mode</code> adjusted with <code>ds-ser-config</code> : <code>fifo_mode = DSSER_FIFO_MODE_BLOCKED</code> : Not all new data could be placed into the FIFO. <code>fifo_mode = DSSER_FIFO_MODE_OVERWRITE</code> : The old data is rejected.
DSSER_COMMUNICATION_FAILED	The function failed with no effect on the input or output data. No data is read from the FIFO. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.

Example The following example shows how to receive a maximum of 4 bytes over the serial channel until the terminating character '\r' occurs:

```
UInt8 data[4];
UInt32 count;
Int32 error;

error = ds-ser_receive_term(serCh, 4, data, &count, '\r');
```

Related topics

Basics

[Basic Principles of Serial Communication..... 76](#)

References

[ds-ser-config..... 91](#)
[ds-ser_init..... 90](#)

ds-ser_fifo_reset

Syntax

```
Int32 ds-ser_fifo_reset(ds-serChannel* serCh)
```

Include file

`ds-ser.h`

Purpose To reset the serial interface.

Description The channel is disabled and the transmit and receive buffers are cleared.

Note

If you still want to use the serial interface, the channel has to be enabled with `dsser_enable`.

Parameters **serCh** Specifies the pointer to the serial channel structure (see `dsser_init`).

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.

Related topics

Basics

[Basic Principles of Serial Communication..... 76](#)

References

[dsser_enable..... 99](#)
[dsser_init..... 90](#)

dsser_enable

Syntax `Int32 dsser_enable(const dsserChannel* serCh)`

Include file `dsser.h`

Purpose	To enable the serial interface.
Description	The UART interrupt is enabled, the serial interface starts transmitting and receiving data.
Parameters	serCh Specifies the pointer to the serial channel structure (see <code>dsser_init</code>).

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.

Related topics

Basics

[Basic Principles of Serial Communication..... 76](#)

References

[dsser_disable..... 100](#)
[dsser_init..... 90](#)

dsser_disable

Syntax	<code>Int32 dsser_disable(const dsserChannel* serCh)</code>
Include file	<code>dsser.h</code>
Purpose	To disable the serial interface.
Description	The serial interface stops transmitting data, incoming data is no longer stored to the receive buffer and the UART sub-interrupts are disabled.

Parameters **serCh** Specifies the pointer to the serial channel structure (see `dsser_init`).

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.

Related topics

Basics

[Basic Principles of Serial Communication..... 76](#)

References

[dsser_enable..... 99](#)
[dsser_init..... 90](#)

dsser_error_read

Syntax `Int32 dsser_error_read(const dsserChannel* serCh)`

Include file `dsser.h`

Purpose To read an error flag of the serial interface.

Description Because only one error flag is returned, you have to call this function as long as the value `DSSER_NO_ERROR` is returned in order to get all error flags.

Parameters **serCh** Specifies the pointer to the serial channel structure (see `dsser_init`).

Return value

This function returns an error flag.

The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error flag set
DSSER_FIFO_OVERFLOW	Too many bytes for the buffer

Related topics**Basics**

[Basic Principles of Serial Communication..... 76](#)

References

[dserr_config..... 91](#)
[dserr_init..... 90](#)

dserr_transmit_fifo_level

Syntax

```
Int32 dserr_transmit_fifo_level(const dserrChannel* serCh)
```

Include file

`dserr.h`

Purpose

To get the number of bytes in the transmit buffer.

Parameters

serCh Specifies the pointer to the serial channel structure (see `dserr_init`).

Return value

This function returns the number of bytes in the transmit buffer.

Related topics**Basics**

[Basic Principles of Serial Communication..... 76](#)

References

[dserr_init..... 90](#)
[dserr_receive_fifo_level..... 103](#)

dsr_receive_fifo_level

Syntax	<code>Int32 dsr_receive_fifo_level(const dsrChannel* serCh)</code>
Include file	<code>dsr.h</code>
Purpose	To get the number of bytes in the receive buffer.
Parameters	serCh Specifies the pointer to the serial channel structure (see <code>dsr_init</code>).
Return value	This function returns the number of bytes in the receive buffer.
Related topics	<p>Basics</p> <p>Basic Principles of Serial Communication..... 76</p> <p>References</p> <p>dsr_init..... 90</p> <p>dsr_transmit_fifo_level..... 102</p>

dsr_status_read

Syntax	<code>Int32 dsr_status_read(dsrChannel* serCh, const UInt8 register_type)</code>
Include file	<code>dsr.h</code>
Purpose	To read the value of one or more status registers and store the values in the corresponding fields of the channel structure.
Parameters	serCh Specifies the pointer to the serial channel structure (see <code>dsr_init</code>).

register_type Specifies the register that is read. You can combine the predefined symbols with the logical operator OR to read several registers. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_STATUS_IIR_FCR	Interrupt status register, see <code>dsser_ISR</code>
DSSER_STATUS_LSR	Line status register, see <code>dsser_LSR</code>
DSSER_STATUS_MSR	Modem status register, see <code>dsser_MSR</code>

Return value

This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.

Example

This example shows how to check if the clear-to-send bit has changed:

```
UInt8 cts;
dsser_status_read(serCh, DSSER_STATUS_MSR);
cts = serCh->modemStatusReg.Bit.DSSER_CTS_STATUS;
```

Related topics

Basics

[Basic Principles of Serial Communication..... 76](#)

References

[dsser_init..... 90](#)
[dsser_ISR..... 81](#)
[dsser_LSR..... 83](#)
[dsser_MSR..... 84](#)

dsser_set

Syntax

```
Int32 dsser_set(
    dsserChannel *serCh,
    UInt32 type,
    const void *value_p)
```


Include	<code>dsser.h</code>						
Purpose	To set a property of the UART.						
Description	<p>The DS2202 board is delivered with a standard quartz working with the frequency of $1.8432 \cdot 10^6$ Hz. You can exchange this quartz for another with a different frequency. Then you have to set the new quartz frequency using <code>dsser_set</code> followed by executing <code>dsser_config</code>.</p> <div> <p>Note</p> <p>You must execute <code>dsser_config</code> after <code>dsser_set</code>; otherwise <code>dsser_set</code> has no effect.</p> </div>						
Parameters	<p>serCh Specifies the pointer to the serial channel structure.</p> <p>type Specifies the property to be changed (<code>DSSER_SET_UART_FREQUENCY</code>).</p> <p>value_p Specifies the pointer to a UInt32-variable with the new value, for example, a variable which contains the quartz frequency.</p>						
Return value	<p>This function returns an error code. The following symbols are predefined:</p> <table> <tr> <th>Predefined Symbol</th><th>Meaning</th></tr> <tr> <td><code>DSSER_NO_ERROR</code></td><td>No error occurred during the operation.</td></tr> <tr> <td><code>DSSER_COMMUNICATION_FAILED</code></td><td>The function failed. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.</td></tr> </table>	Predefined Symbol	Meaning	<code>DSSER_NO_ERROR</code>	No error occurred during the operation.	<code>DSSER_COMMUNICATION_FAILED</code>	The function failed. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.
Predefined Symbol	Meaning						
<code>DSSER_NO_ERROR</code>	No error occurred during the operation.						
<code>DSSER_COMMUNICATION_FAILED</code>	The function failed. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.						
Example	<p>This example sets a new value for the frequency.</p> <pre>UInt32 freq = 1843200; /* 1.8432 MHz */ Int32 error; error = dsser_set(serCh, DSSER_SET_UART_FREQUENCY, &freq);</pre>						

Related topics**Basics**

[Basic Principles of Serial Communication..... 76](#)

References

[dsser_config..... 91](#)

dsser_subint_handler_inst

Syntax

```
dsser_subint_handler_t dsser_subint_handler_inst(
    dsserChannel* serCh,
    dsser_subint_handler_t subint_handler)
```

Include file

dsser.h

Purpose

To install a subinterrupt handler for the serial interface.

Description

After installing the handler the specified subinterrupt type must be enabled (see `dsser_subint_enable`).

Note

The interrupt functions may be used only in handcoded applications. Using them in Simulink applications (User-Code or S-functions) conflicts with the internal interrupt handling.

Parameters

serCh Specifies the pointer to the serial channel structure (see `dsser_init`).
subint_handler Specifies the pointer to the subinterrupt handler.

Return value

This function returns the pointer to the previously installed subinterrupt handler.

Related topics**Basics**

[Basic Principles of Serial Communication..... 76](#)

References

[dsser_init..... 90](#)
[dsser_subint_disable..... 108](#)
[dsser_subint_enable..... 107](#)

dsser_subint_enable

Syntax

```
Int32 dsser_subint_enable(
    dsserChannel* serCh,
    const UInt8 subint)
```

Include file

`dsser.h`

Purpose

To enable one or several subinterrupts of the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see `dsser_init`).

subint Specifies the subinterrupts to be enabled. You can combine the predefined symbols with the logical operator OR to enable several subinterrupts. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_TRIGGER_LEVEL_SUBINT_MASK	Interrupt triggered when user trigger level is reached (see Trigger Levels on page 76)
DSSER_TX_FIFO_EMPTY_SUBINT_MASK	Interrupt triggered when transmit buffer is empty

Predefined Symbol	Meaning
DSSER_RECEIVER_LINE_SUBINT_MASK	Line status interrupt of the UART
DSSER_MODEM_STATE_SUBINT_MASK	Modem status interrupt of the UART

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.

Related topics

Basics

[Basic Principles of Serial Communication..... 76](#)

References

[dsser_init..... 90](#)
[dsser_subint_disable..... 108](#)
[dsser_subint_handler_inst..... 106](#)

dsser_subint_disable

Syntax

```
Int32 dsser_subint_disable(
    dsserChannel* serCh,
    const UInt8 subint)
```

Include file

`dsser.h`

Purpose

To disable one or several subinterrupts of the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see `dsser_init`).

subint Specifies the subinterrupts to be disabled. You can combine the predefined symbols with the logical operator OR to disable several subinterrupts. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_TRIGGER_LEVEL_SUBINT_MASK	Interrupt triggered when the user trigger level is reached (see Trigger Levels on page 76)
DSSER_TX_FIFO_EMPTY_SUBINT_MASK	Interrupt triggered when the transmit buffer is empty
DSSER_RECEIVER_LINE_SUBINT_MASK	Line status interrupt of the UART
DSSER_MODEM_STATE_SUBINT_MASK	Modem status interrupt of the UART

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART is probably overloaded. Do not poll this function because it may lead to an endless loop.

Related topics

Basics

[Basic Principles of Serial Communication](#)..... 76

References

[dsser_init](#)..... 90
[dsser_subint_enable](#)..... 107
[dsser_subint_handler_inst](#)..... 106

dsser_word2bytes

Syntax

```
UInt8* dsser_word2bytes(
    const UInt32* word,
    UInt8* bytes,
    const int bytesInWord)
```

Include file

dsser.h

Purpose

To convert a word (max. 4 bytes long) into a byte array.

Parameters

word Specifies the pointer to the input word.

bytes Specifies the pointer to the byte array. The byte array must have enough memory for **bytesInWord** elements.

bytesInWord Specifies the number of elements in the byte array. Possible values are: 2, 3, 4.

Return value

This function returns the pointer to a byte array.

Example

The following example shows how to write a processor-independent function that transmits a 32-bit value:

```
void word_transmit(dsSerChannel* serCh, UInt32* word, UInt32* count)
{
    UInt8    bytes[4];
    UInt8*   data_p;
    if(dsSer_transmit_fifo_level(serCh) < serCh->fifo_size - 4)
    {
        data_p = dsSer_word2bytes(word, bytes, 4);
        dsSer_transmit(serCh, 4, data_p, count);
    }
    else
    {
        *count = 0;
    }
}
```

Use of the function:

```
UInt32 word = 0x12345678;
UInt32 count;
word_transmit(serCh, &word, &count);
```

Related topics**Basics**

[Basic Principles of Serial Communication..... 76](#)

References

[dsSer_bytes2word..... 111](#)
[dsSer_transmit..... 94](#)

dsser_bytes2word

Syntax	<pre>UInt32* dsser_bytes2word(UInt8* bytes_p, UInt32* word_p, const int bytesInWord)</pre>
Include file	dsser.h
Purpose	To convert a byte array with a maximum of 4 elements into a single word.
Parameters	<p>bytes_p Specifies the pointer to the input byte array.</p> <p>word_p Specifies the pointer to the converted word.</p> <p>bytesInWord Specifies the number of elements in the byte array. Possible values are 2, 3, 4.</p>
Return value	This function returns the pointer to the converted word.

Example The following example shows how to write a processor-independent function that receives a 32-bit value:

```
void word_receive(dsserChannel* serCh, UInt32* word_p, UInt32* count)
{
    UInt8 bytes[4];
    if (dsser_receive_fifo_level(serCh) > 3)
    {
        dsser_receive(serCh, 4, bytes, count);
        word_p = dsser_bytes2word(bytes, word_p, 4);
    }
    else
    {
        *count = 0;
    }
}
```

Use of the function:

```
UInt32 word;
UInt32 count;
word_receive(serCh, &word, &count);
```

Related topics

Basics

[Basic Principles of Serial Communication.....](#) 76

References

[dsr_receive.....](#) 95
[dsr_word2bytes.....](#) 109

Slave CAN Access Functions

Where to go from here

Information in this section

Basics on Slave CAN Access Functions.....	115
Information on the communication principles between the master processor board and the slaves.	
Data Structures for CAN.....	118
Information on channels, services and messages to be used by other functions.	
Initialization.....	128
Before you can use a CAN controller you have to perform an initialization process to set up the communication channels between the master and slave.	
CAN Channel Handling.....	130
Information on handling CAN interfaces, called <i>CAN channels</i> .	
CAN Message Handling.....	145
Information on how to deal with all kind of CAN messages.	
CAN Service Functions.....	188
Information on errors and status information.	
CAN Subinterrupt Handling.....	193
Information on defining subinterrupts caused by certain events.	
Utilities.....	196
Information on setting the time base to a defined value, clearing CAN data on the master, and reading the current error code.	
Examples of Using CAN.....	198
Examples showing how to use the CAN functions.	

Information in other sections

[CAN Support \(DS2202 Features !\[\]\(3dfb8d66e81160ad61421a3452093d1b_img.jpg\)\)](#)

The board has a CAN interface to communicate with CAN devices.

Basics on Slave CAN Access Functions

Introduction Information on the communication principles between the master processor board and the slaves.

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

[Basics on the CAN Subsystem](#)..... 115

The CAN subsystem of the DS2202 provides two independent CAN bus interfaces – CAN channels – that meet the ISO/DIS 11898 specification.

[Basic Communication Principles](#)..... 116

This section provides general information on the communication principles.

[CAN Error Message Types](#)..... 117

The functions of the CAN environment report error, warning and information messages if a problem occurs.

Basics on the CAN Subsystem

Introduction The CAN subsystem of the DS2202 provides two independent CAN bus interfaces – CAN channels – that meet the ISO/DIS 11898 specification.

The CAN subsystem supports the following features:

- Registration of up to 100 CAN messages
- Time stamping for all messages
- Functions to read the CAN bus status
For more information, refer to [Getting CAN Status Information \(DS2202 Features !\[\]\(e1bdc70a9006e3802acd56af7aa337d8_img.jpg\)](#)).
- Interrupt generation by message events or CAN bus events
For more information, refer to [Implementing a CAN Interrupt \(DS2202 Features !\[\]\(6ae057bca7ac6a248ab7813081463b17_img.jpg\)](#)).

Related topics

Basics

[Getting CAN Status Information \(DS2202 Features !\[\]\(b64b40baaee5acddc1eab8538ba84754_img.jpg\)](#))

[Implementing a CAN Interrupt \(DS2202 Features !\[\]\(84f47badaad7772cd95667a7c387a639_img.jpg\)](#))

[Initializing the CAN Controller \(DS2202 Features !\[\]\(28f72b996fc97883dfd9d4e8b1b16b4e_img.jpg\)](#))

Basic Communication Principles

Introduction

With the help of the slave access functions, the master processor board controls the actions of the slave CAN and exchanges data with the slave interface.

Note

You have to initialize the communication between the master and slaves (see `ds2202_can_communication_init`).

Basic principles

- The master application initializes the necessary slave function or a group of slave functions based on the CAN controller.
- The message register functions write all needed values to the appropriate handle (`ds2202_canMsg`, for example). The corresponding request and read functions will get the information from this handle later on.
- In order to perform a read operation, the master processor board requests that the previously registered slave function is carried out. The slave then performs the required functions independently and writes the results back into the dual-port memory. If more than one function is required simultaneously – for example, as a result of different tasks on the processor board – priorities must be considered.
- The master processor board application reads/writes the input/output data from/to the slave.

Note

Remember that the master processor board reads the slave results from the dual-port memory in the order in which they occur, and then reads them into a buffer, regardless of whether a particular result is needed. The read functions are the ones that copy data results from the buffer into the processor board application variables.

The slave applications are based on communication functions that are divided into separate classes as follows:

- *Initialization functions* initialize the slave functions.
- *Register functions* make the slave functions known to the slave.
- *Request functions* require that the previously registered slave function is carried out by the slave.
- *Read functions* fetch data from the dual-port memory and convert or scale the data, if necessary.
- *Write functions* convert or scale the data if necessary and write them into the dual-port memory.

When an error occurs with initialization or register functions, an error message appears from the global message module. Then the program ends.

Request, read and write functions return an error code. The application can then deal with the error code.

This communication method along with the command table and the transfer buffer can be initialized in parallel for the statically defined communication channels with fixed priorities (0 ... 6). Just as communication buffers, each communication channel has access to memory space in the dual-port memory so that slave error codes can be transferred.

Related topics

References

[ds2202_can_communication_init..... 128](#)

CAN Error Message Types

Introduction

The functions of the CAN environment report error, warning and information messages if a problem occurs.

Message display

The error, warning, and information messages are displayed by the Log Viewer of the experiment software. A message consists of an error number, the function name, the board index (offset of the PHS-bus address) and the message text.

Message Number	Message Type
100 ... 249	Error
250 ... 349	Warning
400 ... 500	Information

Example

```
Error[121]: ds2202_can_channel_init(6,..) baudrate: too low (min. 10 kBaud)!
```

Data Structures for CAN

Introduction

Information on channels, services and messages to be used by other functions.

Tip

Using RTLib CAN functions, you access the data structures for CAN automatically and you do not have to access them in your application explicitly.

Where to go from here

Information in this section

[ds2202_canChannel..... 118](#)

This structure contains information on the CAN channel capabilities.

[ds2202_canService..... 119](#)

The ds2202_canService structure contains information on the CAN services. The CAN service provides information on errors and status information.

[ds2202_canMsg..... 123](#)

This structure contains information on the CAN message capabilities.

ds2202_canChannel

Introduction

The ds2202_canChannel structure contains information on the CAN channel capabilities.

Syntax

```
typedef struct
{
    UInt32 base;
    Int32 index;
    UInt32 channel;
    UInt32 btr0;
    UInt32 btr1;
    UInt32 frequency;
    UInt32 mb15_format;
    UInt32 busoff_int_number;
} ds2202_canChannel;
```

Include file

Can2202.h

Members

base The PHS-bus base address is provided by the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced`. This parameter is read-only.

index Table index allocated by the message register function. This parameter is read-only.

channel Number of the used CAN channel. This parameter is provided by the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced`. This parameter is read-only.

btr0 Value of Bit Timing Register 0. This parameter is provided by the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced`. This parameter is read-only.

btr1 Value of Bit Timing Register 1. This parameter is provided by the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced`. This parameter is read-only.

frequency Frequency of the CAN controller. This parameter is provided by the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced`. This parameter is read-only.

mb15_format Format of mailbox 15. Mailbox 15 is a double-buffered receive unit of the CAN. Use this mailbox for the message type most frequently used in your application. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_STD	11-bit standard format, CAN 2.0A
DS2202_CAN_EXT	29-bit extended format, CAN 2.0B

This parameter is provided by the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced`. This parameter is read-only.

busoff_int_number Subinterrupt generated when the CAN channel goes bus off. This parameter is provided by the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced`. This parameter is read-only.

Related topics**References**

ds2202_can_channel_init.....	130
ds2202_can_channel_init_advanced.....	133

ds2202_canService

Introduction

The `ds2202_canService` structure contains information on the CAN services. The CAN service provides information on errors and status information.

Syntax

```
typedef struct
{
    UInt32 busstatus;
    UInt32 stdmask;
    UInt32 extmask;
    UInt32 msg_mask15;
    UInt32 tx_ok;
    UInt32 rx_ok;
    UInt32 crc_err;
    UInt32 ack_err;
    UInt32 form_err;
    UInt32 stuffbit_err;
    UInt32 bit1_err;
    UInt32 bit0_err;
    UInt32 rx_lost;
    UInt32 data_lost;
    UInt32 version;
    UInt32 mailbox_err;
    UInt32 data0;
    UInt32 data1;
    UInt16 txqueue_overflowcnt_std;
    UInt16 txqueue_overflowcnt_ext;
    UInt32 module;
    UInt32 queue;
    UInt32 type;
    Int32 index;
} ds2202_canService;
```

Include file

Can2202.h

Members

data0 Contains returned data from the function **ds2202_can_service_read**.

data1 Contains returned data from the function **ds2202_can_service_read**.

Note

For each service the structure now provides its own member. For the meaning of the services, refer to the parameter **type**. The members **data0** and **data1** remain in the structure for compatibility reasons.

module The CAN module is provided by the function **ds2202_can_service_register**. This parameter is read-only.

queue This parameter is provided by the function **ds2202_can_service_register**. This parameter is read-only.

type Type of the service already allocated by the previously performed register function. Once a service is registered on the slave it can deliver a value. The return value will be stored in the structure members **data0** and **data1**. This

parameter is provided by the function **ds2202_can_service_register**. This parameter is read-only.

Note

Start the CAN channel with the enabled status interrupt to use the following predefined services (see **ds2202_can_channel_start**).

Predefined Symbol	Meaning
DS2202_CAN_SERVICE_TX_OK	Number of successfully sent TX/RM/RQTX messages
DS2202_CAN_SERVICE_RX_OK	Number of successfully received RX/RQRX messages
DS2202_CAN_SERVICE_CRC_ERR	Number of CRC errors
DS2202_CAN_SERVICE_ACK_ERR	Number of acknowledge errors
DS2202_CAN_SERVICE_FORM_ERR	Number of format errors
DS2202_CAN_SERVICE_BIT1_ERR	Number of Bit1 errors
DS2202_CAN_SERVICE_BIT0_ERR	Number of Bit0 errors
DS2202_CAN_SERVICE_STUFFBIT_ERR	Number of stuff bit errors

Note

It is not necessary to start the CAN channel with the enabled status interrupt if you are using only the following predefined services (see **ds2202_can_channel_start**).

Predefined Symbol	Meaning
DS2202_CAN_SERVICE_RX_LOST	Number of lost RX messages. The RX lost counter will be incremented when a received message is overwritten in the receive mailbox before the message has been read.
DS2202_CAN_SERVICE_DATA_LOST	Number of data lost errors. The data lost counter will be incremented when the data of a message is overwritten before the data has been written to the communication queue.
DS2202_CAN_SERVICE_MAILBOX_ERR	Number of mailbox errors. If a message to be sent can not be assigned to a mailbox, the mailbox error counter will be increased by one. For possible error reasons, refer to the additional information on DS2202_CAN_SERVICE_MAILBOX_ERR below this table.
DS2202_CAN_SERVICE_BUSSTATUS	Status of the CAN controller. For the predefined values, refer to the additional information on DS2202_CAN_SERVICE_BUSSTATUS below this table.
DS2202_CAN_SERVICE_STDMASK	Status of the global standard mask register
DS2202_CAN_SERVICE_EXTMASK	Status of the global extended mask register
DS2202_CAN_SERVICE_MSG_MASK15	Status of the message 15 mask register
DS2202_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT	Overflow counter of the transmit queue. The overflow counter (STD or XTD message format) is incremented, if the queue is filled (64 messages) and a new message

Predefined Symbol	Meaning
DS2202_CAN_SERVICE_VERSION	<p>arrives. Depending on the <code>overrun_policy</code> parameter set with <code>ds2202_can_msg_txqueue_init</code> the new message overwrites the oldest message entry or is ignored.</p> <p>The overflow counters are 16-bit counters. The wraparound occurs after 65535 overflows.</p> <p>Version number of the CAN firmware.</p>

index Table index already allocated by the register function `ds2202_can_service_register`. This parameter is read-only.

Parameter type

Additional information on the service functions provided by the **type** parameter:

DS2202_CAN_SERVICE_MAILBOX_ERR Provides the number of mailbox errors. The following table describes possible error reasons and how you can avoid these errors:

Error reason	Description	Workaround
All mailboxes are filled.	The messages are not removed fast enough out of a mailbox.	Decrease the timeout value of all messages of the corresponding CAN channel and restart the application.
Conflict between two message IDs.	This error can occur if standard and extended messages are used on a CAN channel simultaneously. Check whether all messages are sent according to your requirements. It is not possible to remove remote messages temporarily from a mailbox. Check for a possible problem with a registered remote message.	<p>Try the first element of the following list. If the error counter still increases, try the next one:</p> <ul style="list-style-type: none"> Decrease the timeout value for messages with the same format as mailbox 14 – that is, the opposite format of mailbox 15 (refer to <code>ds2202_can_channel_init</code>). Initialize the parameter <code>mb15_format</code> with the other format when calling <code>ds2202_can_channel_init</code> or <code>ds2202_can_channel_init_advanced</code>. Choose different message IDs for messages corresponding to the format of mailbox 14. Do not use standard and extended messages on one CAN channel simultaneously.

DS2202_CAN_SERVICE_BUSSTATUS Provides bus status information; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_BUSOFF_STATE	The CAN channel disconnects itself from the CAN bus. Use <code>ds2202_can_channel_Boff_return</code> to recover from the bus off state.
DS2202_CAN_WARN_STATE	The CAN controller is still active. The CAN controller recovers from this state automatically.
DS2202_CAN_ACTIVE_STATE	The CAN controller is active.

Note

After calling `ds2202_can_channel_BOff_return`, the service `DS2202_CAN_SERVICE_BUSSTATUS` will not return `DS2202_CAN_BUSOFF_STATE`.

Example

The following example shows how to use the CAN service with an overflow counter:

```
ds2202_canService* service;
UInt16 overflow;
...
service = ds2202_can_service_register(
            txCh, DS2202_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT);
...
ds2202_can_service_request( service );
ds2202_can_service_read( service );
overflow = service->txqueue_overflowcnt_std;
```

Related topics

References

ds2202_can_channel_BOff_go	139
ds2202_can_channel_BOff_return	140
ds2202_can_channel_init	130
ds2202_can_channel_init_advanced	133
ds2202_can_channel_start	136
ds2202_can_msg_txqueue_init	173
ds2202_can_service_read	191
ds2202_can_service_register	188

ds2202_canMsg

Introduction

The `ds2202_canMsg` structure contains information on the CAN message capabilities.

Syntax

```
typedef struct
{
    double timestamp;
    Float32 deltatime;
    Float32 delaytime;
    Int32 processed;
    UInt32 datalen;
    UInt32 data[8];
    UInt32 identifier;
    UInt32 format;
    UInt32 module;
    UInt32 queue;
    Int32 index;
    UInt32 msg_no;
    UInt32 type;
    UInt32 inform;
    UInt32 timecount;
    ds2202_canChannel* canChannel;
    ds2202_canService* msgService;
} ds2202_canMsg;
```

Include file

Can2202.h

Members

timestamp This parameter contains the following values:

- For transmit or remote messages: The point of time the last message was successfully sent (given in seconds).
- For receive messages: The point of time the last message was received (given in seconds).

This parameter is updated by the function **ds2202_can_msg_read** if the message was registered using the **inform** parameter DS2202_CAN_TIMECOUNT_INFO.

deltatime Time difference in seconds between the old and the new timestamp

This parameter is updated by the function **ds2202_can_msg_read** if the message was registered with the **inform** parameter DS2202_CAN_TIMECOUNT_INFO.

Note

If several CAN identifiers are received with a single RX message, the **deltatime** parameter delivers useless values. For this reason, it is recommended to use the **deltatime** parameter only if one CAN identifier is received per registered CAN message.

delaytime Time difference between the update and the sending of a message (for TX, RQTX and RM messages only). For cyclic sending, the delaytime

between the update and the sending of a message is used. For acyclic sending, the delaytime between the trigger and the successful sending of a message is used. Valid range is 0.0 ... 100.0 seconds.

This parameter is updated by the function **ds2202_can_msg_read** if the message was registered with the **inform** parameter DS2202_CAN_DELAYCOUNT_INFO.

processed Processed flag of the message. This parameter is updated by the function **ds2202_can_msg_read**; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_PROCESSED	The message has been sent/received since the last execution call.
DS2202_CAN_NOT_PROCESSED	The message has not been sent/received since the last execution call.

datalen Length of the data in the CAN message in byte. This parameter is updated by the function **ds2202_can_msg_read** if the message was registered with the **inform** parameter DS2202_CAN_DATA_INFO.

data[8] Buffer for CAN message data. This data is updated by the function **ds2202_can_msg_read** if the message was registered with the **inform** parameter DS2202_CAN_DATA_INFO.

identifier Identifier of the message. This parameter is provided by the message register functions and is read-only.

format Format of the identifier. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_STD	11-bit standard format, CAN 2.0A
DS2202_CAN_EXT	29-bit extended format, CAN 2.0B

This parameter is provided by the message register functions and is read-only.

module Address of the registered message. This parameter is provided by the message register functions and is read-only.

queue Communication channel within the range of 0 ... 5. This parameter is provided by the message register functions and is read-only.

index Table index already allocated by the previously performed register function. This parameter is provided by the message register functions and is read-only.

msg_no Number of the message. This parameter is provided by the message register functions and is read-only.

type Type of the CAN message. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_TX	Transmit message registered by ds2202_can_msg_tx_register

Predefined Symbol	Meaning
DS2202_CAN_RX	Receive message registered by ds2202_can_msg_rx_register
DS2202_CAN_RM	Remote message registered by ds2202_can_msg_rm_register
DS2202_CAN_RQTX	RQTX message registered by ds2202_can_msg_rqtx_register
DS2202_CAN_RQRX	RQRX message registered by ds2202_can_msg_rqr_register

This parameter is provided by the message register functions and is read-only.

inform Specifies the kind of information returned by the function **ds2202_can_msg_read**. You have to register a message with the appropriate **inform** parameter to get the requested information. You can combine the predefined symbols with the logical operator OR. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_INFO	Returns no information.
DS2202_CAN_DATA_INFO	Updates the parameters data and datalen (needed for receive and request (RQRX) messages).
DS2202_CAN_MSG_INFO	Updates the message identifier and the message format for RM, RQ, TX and RX messages.
DS2202_CAN_TIMECOUNT_INFO	Updates the parameters timestamp and the deltatime.
DS2202_CAN_DELAYCOUNT_INFO	Updates the parameter delaytime.

NOTICE

Do not modify the **inform** parameter after the message has been registered or your message data will be corrupted.

This parameter is provided by the message register functions and is read-only.

timecount Internally used parameter. This parameter is read-only.

canChannel Pointer to the used **ds2202_canChannel1** structure where the message object is installed. This parameter is read-only. (See [ds2202_canChannel](#) on page 118.)

msgService Only used by the message processed functions to read the processed status (sent or received) of a message. This parameter is read-only.

Related topics

References

ds2202_can_msg_read.....	180
ds2202_can_msg_rm_register.....	160
ds2202_can_msg_rqr_register.....	157
ds2202_can_msg_rqt_register.....	154
ds2202_can_msg_rx_register.....	150
ds2202_can_msg_tx_register.....	146
ds2202_canChannel.....	118

Initialization

Introduction

Before you can use a CAN controller you have to perform an initialization process that sets up the communication channels between the master and slave (**queue** parameter).

For the initialization of the DS2202 board, refer to **ds2202_init**.

ds2202_can_communication_init

Syntax

```
void ds2202_can_communication_init(  
    const UInt32 base,  
    const UInt32 bufferwarn)
```

Include file

Can2202.h

Purpose

To initialize the communication between the master and the slave DS2202.

Description

This function also initializes seven communication channels with fixed queues (0 ... 6) for the master-to-slave communication. The communication channel QUEUE0 has the highest priority. The slave initializes the communication with the master itself and sends an acknowledgment code if the initialization was successful. If the master does not receive this acknowledgment code within one second, the program will be aborted.

Parameters

base Specifies the PHS-bus base address of the DS2202 board.

bufferwarn Enables the bufferwarn subinterrupt. The subinterrupt handler will be installed automatically. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_INT_DISABLE	The bufferwarn subinterrupt is disabled.
DS2202_CAN_INT_ENABLE	The bufferwarn subinterrupt is enabled.

Return value

None

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2202_can_communication_init(x,..) memory: allocation error on master	Memory allocation error. No free memory on the master.
104	Error	ds2202_can_communication_init(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation will be aborted.
105	Error	ds2202_can_communication_init(x,..) subint: init failed by master	Master subinterrupt initialization failed. There is not enough memory available.
106	Error	ds2202_can_communication_init(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second due to a wrong firmware version or a hardware failure.
107	Error	ds2202_can_communication_init(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2202_can_communication_init(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with ds2202_can_msg_sleep or ds2202_can_channel_all_sleep when registering messages or services.
109	Error	ds2202_can_communication_init(x,..) slave: wrong firmware version	The firmware version of the CAN controller is incompatible with the Real-Time Library that is used.
200	Error	ds2202_can_communication_init(x,..) slave: not connected to HwInterrupt	There may be a hardware failure or the initialization process is not correct.
400	Info	ds2202_can_communication_init(x,..) DS2202 CAN firmware version x.y.z detected.	Information on the firmware version running on the DS2202.

Related topics**Examples**

Example of Handling Request and Remote Messages.....	200
Example of Handling Transmit and Receive Messages.....	198
Example of Using Subinterrupts.....	202

References

ds2202_can_channel_all_sleep.....	137
ds2202_can_msg_sleep.....	178

CAN Channel Handling

Introduction

Information on handling CAN interfaces, called *CAN channels*.

Where to go from here

Information in this section

ds2202_can_channel_init	130
To initialize a CAN channel.	
ds2202_can_channel_init_advanced	133
To initialize a CAN channel with additional parameters.	
ds2202_can_channel_start	136
To start a CAN channel.	
ds2202_can_channel_all_sleep	137
To deactivate all CAN messages of a channel.	
ds2202_can_channel_all_wakeup	138
To reactivate all CAN messages of a channel.	
ds2202_can_channel_BOff_go	139
To set the bus off state.	
ds2202_can_channel_BOff_return	140
To return from the bus off state.	
ds2202_can_channel_set	141
To set an attribute for a CAN channel.	
ds2202_can_channel_txqueue_clear	143
To clear the content of the transmit queues of the selected CAN channel.	

ds2202_can_channel_init

Syntax

```
ds2202_canChannel* ds2202_can_channel_init(
    const UInt32 base,
    const UInt32 channel,
    const UInt32 baudrate,
    const UInt32 mb15_format,
    const Int32 busoff_subinterrupt,
    const UInt32 termination);
```

Include File

Can2202.h

Purpose

To perform the basic initialization of the specified CAN channel, that is, to reset the CAN controller and set its baud rate.

Note

To complete CAN channel initialization, you have to call `ds2202_can_channel_start`.

Description

If no error occurs, `ds2202_can_channel_init` returns a pointer to the `ds2202_canChannel` structure.

If an interrupt should be sent for the bus off state of the CAN controller, you have to specify a subinterrupt number and a subinterrupt handler.

Parameters

base Specifies the PHS-bus base address of the DS2202 board.

channel Specifies the CAN channel within the range 0 ... 1.

baudrate Specifies the baud rate of the CAN bus within the range 10 kBd ... 1 MBd

mb15_format Specifies the format for mailbox 15. Mailbox 15 is a double-buffered receive unit of the CAN. Use this mailbox for the message type most frequently used in your application. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_STD	11-bit standard format, CAN 2.0A
DS2202_CAN_EXT	29-bit extended format, CAN 2.0B

busoff_subinterrupt Specifies the Subinterrupt number for the bus off state. Valid range is 0 ... 14. Use the following predefined symbol to disable the bus off interrupt:

Predefined Symbol	Meaning
DS2202_CAN_NO_SUBINT	No interrupt for bus off

termination Activates the bus termination (120 W). The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_TERMINATION_ON	Bus termination activated
DS2202_CAN_TERMINATION_OFF	Bus termination deactivated

Return value

canChannel Pointer to the `ds2202_canChannel` structure. (See [ds2202_canChannel](#) on page 118.)

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2202_can_channel_init_advanced(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
104	Error	ds2202_can_channel_init_advanced(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
106	Error	ds2202_can_channel_init_advanced(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second due to a wrong firmware version or a hardware failure.
107	Error	ds2202_can_channel_init_advanced(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2202_can_channel_init_advanced(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with ds2202_can_msg_sleep or ds2202_can_channel_all_sleep when registering messages or services.
121	Error	ds2202_can_channel_init(x,..) baudrate: too low (min. 10 kBaud)!	The parameter baud rate is too low. The minimum baud rate is limited by DS2202_CAN_MIN_BAUDRATE.
122	Error	ds2202_can_channel_init(x,..) baudrate: too high (max. 1 MBaud)!	The parameter baud rate is too high. The maximum baud rate is limited by DS2202_CAN_MAX_BAUDRATE.
123	Error	ds2202_can_channel_init_advanced(x,..) channel: use range 0..1!	Use a CAN channel within the range of 0 ... 1.
140	Error	ds2202_can_channel_init_advanced(x,..) format: wrong format.	Only the symbols DS2202_CAN_STD and DS2202_CAN_EXT are allowed for the parameter mb15_format.
141	Error	ds2202_can_channel_init_advanced(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range of 0 ... 14.

Information message

The following info message is defined:

ID	Type	Message	Description
250	Info	ds2202_can_channel_init(x,..) baudrate: Doesn't match the desired baudrate. (baudrate = X bit/s)	The given baud rate differs from the default baud rate of X bit/s.

Related topics**Examples**

Example of Handling Request and Remote Messages.....	200
Example of Handling Transmit and Receive Messages.....	198
Example of Using Subinterrupts.....	202

References

ds2202_can_channel_all_sleep.....	137
ds2202_can_channel_all_wakeup.....	138
ds2202_can_channel_BOff_go.....	139
ds2202_can_channel_BOff_return.....	140
ds2202_can_channel_init_advanced.....	133
ds2202_can_channel_set.....	141
ds2202_can_channel_start.....	136
ds2202_can_msg_rm_register.....	160
ds2202_can_msg_rqr_register.....	157
ds2202_can_msg_rqt_register.....	154
ds2202_can_msg_rx_register.....	150
ds2202_can_msg_sleep.....	178
ds2202_can_msg_tx_register.....	146
ds2202_can_service_register.....	188

ds2202_can_channel_init_advanced

Syntax

```
ds2202_canChannel* ds2202_can_channel_init_advanced(
    const UInt32 base,
    const UInt32 channel,
    const UInt32 bit_timing0,
    const UInt32 bit_timing1,
    const UInt32 mb15_format,
    const Int32 busoff_subinterrupt,
    const UInt32 termination);
```

Include file

Can2202.h

Purpose

To perform the initialization of a CAN channel with parameters.

If no error occurs, **ds2202_can_channel_init_advanced** returns a pointer to the ds2202_canChannel structure.

Note

You have to call **ds2202_can_channel_start** to complete the CAN channel initialization.

Description

Use the returned handle when calling one of the following functions:

- **ds2202_can_channel_start**
- **ds2202_can_channel_all_sleep**
- **ds2202_can_channel_all_wakeup**
- **ds2202_can_channel_BOff_go**
- **ds2202_can_channel_BOff_return**
- **ds2202_can_channel_set**
- **ds2202_can_msg_tx_register**
- **ds2202_can_msg_rx_register**
- **ds2202_can_msg_rqtz_register**
- **ds2202_can_msg_rqrx_register**

If an interrupt should be sent for the bus off state of the CAN controller, you have to specify a subinterrupt number.

The function **ds2202_can_channel_start** completely initializes the CAN controller. All mailbox-independent initializations are done by this function. After the hardware-dependent registers are set, the CAN controller interrupts are disabled.

Parameters

base Specifies the PHS-bus base address of the DS2202 board

channel Specifies the CAN channel 0 ... 1

bit_timing0 Specifies the value for the bit timing register 0

bit_timing1 Specifies the value for the bit timing register 1

mb15_format Specifies the format for mailbox 15. Mailbox 15 is a double-buffered receive unit of the CAN. Use this mailbox for the message type most frequently used in your application. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_STD	11-bit standard format, CAN 2.0A
DS2202_CAN_EXT	29-bit extended format, CAN 2.0B

busoff_subinterrupt Specifies the Subinterrupt number for bus off. Valid range is 0 ... 14. Use the following predefined symbol to disable the bus off interrupt:

Predefined Symbol	Meaning
DS2202_CAN_NO_SUBINT	No interrupt for bus off

termination Activates the bus termination (120 W). The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_TERMINATION_ON	Bus termination activated
DS2202_CAN_TERMINATION_OFF	Bus termination deactivated

Return value `canChannel` Specifies the pointer to the `ds2202_canChannel` structure. (See [ds2202_canChannel](#) on page 118.)

Messages The following messages are defined:

ID	Type	Message	Description
101	Error	<code>ds2202_can_channel_init_advance(x,..)</code> memory allocation error on master	Memory allocation error. No free memory on the master.
104	Error	<code>ds2202_can_channel_init_advanced(x,..)</code> queue: master to slave overflow.	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns <code>DS2202_CAN_NO_ERROR</code> .
106	Error	<code>ds2202_can_channel_init_advanced(x,..)</code> slave: not responding	The slave did not finish the initialization of the communication in the time <code>DSCOMDEF_TIMEOUT</code> (in seconds).
107	Error	<code>ds2202_can_channel_init_advanced(x,..)</code> slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	<code>ds2202_can_channel_init_advanced(x,..)</code> queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with <code>ds2202_can_msg_sleep</code> or <code>ds2202_can_channel_all_sleep</code> when registering messages or services.
123	Error	<code>ds2202_can_channel_init_advanced(x,..)</code> channel: use range 0..1 !	Use a CAN channel within the range of 0 ... 1.
140	Error	<code>ds2202_can_channel_init_advanced(x,..)</code> format: wrong format	Only the symbols <code>DS2202_CAN_STD</code> and <code>DS2202_CAN_EXT</code> are allowed for the parameter <code>mb15_format</code> .
141	Error	<code>ds2202_can_channel_init_advanced(x,..)</code> subint: use range 0..14 !	The subinterrupt number must be within the range of 0 ... 14.

Example

```
ds2202_canChannel* CH;
CH = ds2202_can_channel_init_advanced(
    DS2202_1_BASE,      /* PHS-bus base address */
    0,                  /* channel 0 */
    0x80,               /* BTR0 */
    0x6F,               /* BTR1 */
    DS2202_CAN_STD,     /* use mailbox 15 to receive only */
                        /* CAN messages with standard format */
    DS2202_CAN_NO_SUBINT, /* generate no subinterrupt when */
                        /* the CAN controller goes in the */
                        /* bus off state */
    DS2202_CAN_TERMINATION_ON /* Bus termination activated */
);
```

Related topics

References

ds2202_can_channel_all_sleep	137
ds2202_can_channel_all_wakeup	138
ds2202_can_channel_BOFF_go	139
ds2202_can_channel_BOFF_return	140
ds2202_can_channel_init	130
ds2202_can_channel_set	141
ds2202_can_channel_start	136
ds2202_can_msg_rqr_register	157
ds2202_can_msg_rqt_register	154
ds2202_can_msg_rx_register	150
ds2202_can_msg_sleep	178
ds2202_can_msg_tx_register	146

ds2202_can_channel_start

Syntax

```
void ds2202_can_channel_start(
    const ds2202_canChannel* canCh,
    const UInt32 status_int);
```

Include file

Can2202.h

Purpose

To complete the initialization and start the CAN channel referenced by the pointer canCh.

Description

The CAN channel will change to the bus on state and the DS2202 slave interrupts will be enabled. Use the returned handle from the function **ds2202_can_channel_init** or **ds2202_can_channel_init_advanced** to call this function.

Parameters

canCh Specifies the pointer to the **ds2202_canChannel** structure.

status_int Enables the status change interrupt; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_INT_DISABLE	No status interrupt will be generated.
DS2202_CAN_INT_ENABLE	A status change interrupt can be generated when a CAN bus event is detected in the Status Register. A status change interrupt will occur on each successful reception or transmission on the CAN bus, regardless of whether the DS2202 slave has configured a message object to receive that particular message identifier.

Predefined Symbol	Meaning
	This interrupt is useful to detect bus errors caused by physical layer issues, such as noise. In most applications, this bit should not be set. Since this interrupt occurs for each message, the DS2202 slave will be unnecessarily burdened.

Return value None

Messages The following messages are defined:

ID	Type	Message	Description
104	Error	ds2202_can_channel_start(x,...) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.

Related topics

Examples

Example of Handling Request and Remote Messages.....	200
Example of Handling Transmit and Receive Messages.....	198
Example of Using Subinterrupts.....	202

References

ds2202_can_channel_init.....	130
ds2202_can_channel_init_advanced.....	133
ds2202_canChannel.....	118

ds2202_can_channel_all_sleep

Syntax

```
Int32 ds2202_can_channel_all_sleep(
    const ds2202_canChannel* canCh);
```

Include file

Can2202.h

Purpose

To stop the transmission of all previously registered transmit, request transmission and remote messages, and the data transfer from all registered messages to the master processor board.

Description

The messages are deactivated and set into the sleep mode until they will be reactivated by `ds2202_can_channel_all_wakeup`.

Use the returned handle from the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2202_canChannel` structure. (See [ds2202_canChannel](#) on page 118.)

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	The communication buffer from master to slave has overflowed. Repeat the function until it returns DS2202_CAN_NO_ERROR.

Function execution times

For information, refer to [Function Execution Times](#) on page 207.

Example

```
ds2202_can_channel_all_sleep(canCh);
```

Related topics**References**

ds2202_can_channel_all_wakeup	138
ds2202_can_channel_init	130
ds2202_can_channel_init_advanced	133

ds2202_can_channel_all_wakeup

Syntax

```
Int32 ds2202_can_channel_all_wakeup(
    const ds2202_canChannel* canCh);
```

Include file

Can2202.h

Purpose

To reactivate all messages that were deactivated by calling the functions `ds2202_can_channel_all_sleep` and `ds2202_can_msg_sleep`.

Description	Use the returned handle from the function <code>ds2202_can_channel_init</code> or <code>ds2202_can_channel_init_advanced</code> to call this function.										
Parameters	canCh Specifies the pointer to the <code>ds2202_canChannel</code> structure. (See ds2202_canChannel on page 118.)										
Return value	This function returns the error code; the following symbols are predefined: <table border="1"> <thead> <tr> <th>Predefined Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DS2202_CAN_NO_ERROR</td><td>The function has been performed without error.</td></tr> <tr> <td>DS2202_CAN_BUFFER_OVERFLOW</td><td>The communication buffer from master to slave has overflowed. Repeat the function until it returns DS2202_CAN_NO_ERROR.</td></tr> </tbody> </table>	Predefined Symbol	Meaning	DS2202_CAN_NO_ERROR	The function has been performed without error.	DS2202_CAN_BUFFER_OVERFLOW	The communication buffer from master to slave has overflowed. Repeat the function until it returns DS2202_CAN_NO_ERROR.				
Predefined Symbol	Meaning										
DS2202_CAN_NO_ERROR	The function has been performed without error.										
DS2202_CAN_BUFFER_OVERFLOW	The communication buffer from master to slave has overflowed. Repeat the function until it returns DS2202_CAN_NO_ERROR.										
Function execution times	For information, refer to Function Execution Times on page 207.										
Example	<pre>ds2202_can_channel_all_wakeup(canCh);</pre>										
Related topics	References <table> <tr> <td>ds2202_can_channel_all_sleep.....</td><td>137</td></tr> <tr> <td>ds2202_can_channel_init.....</td><td>130</td></tr> <tr> <td>ds2202_can_channel_init_advanced.....</td><td>133</td></tr> <tr> <td>ds2202_can_msg_sleep.....</td><td>178</td></tr> <tr> <td>ds2202_canChannel.....</td><td>118</td></tr> </table>	ds2202_can_channel_all_sleep	137	ds2202_can_channel_init	130	ds2202_can_channel_init_advanced	133	ds2202_can_msg_sleep	178	ds2202_canChannel	118
ds2202_can_channel_all_sleep	137										
ds2202_can_channel_init	130										
ds2202_can_channel_init_advanced	133										
ds2202_can_msg_sleep	178										
ds2202_canChannel	118										

ds2202_can_channel_BOff_go

Syntax	<pre>Int32 ds2202_can_channel_BOff_go(const ds2202_canChannel* canCh);</pre>
Include file	Can2202.h
Purpose	To set the CAN channel to the bus off state. All bus operations performed by the CAN channel are canceled.

Description Use the returned handle from the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced` to call this function.

Parameters `canCh` Specifies the pointer to the `ds2202_canChannel` structure. (See [ds2202_canChannel](#) on page 118.)

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.

Function execution times For information, refer to [Function Execution Times](#) on page 207.

Example

```
ds2202_can_channel_BOff_go(canCh);
```

Related topics

References

ds2202_can_channel_BOff_return	140
ds2202_can_channel_init	130
ds2202_can_channel_init_advanced	133
ds2202_canChannel	118

ds2202_can_channel_BOff_return

Syntax

```
Int32 ds2202_can_channel_BOff_return(  
    const ds2202_canChannel* canCh);
```

Include file `Can2202.h`

Purpose To reset the slave DS2202 CAN channel from the bus off state.

Use the returned handle from the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced` to call this function.

Parameters	canCh Specifies the pointer to the <code>ds2202_canChannel</code> structure. (See ds2202_canChannel on page 118.)								
Return value	<p>This function returns the error code; the following symbols are predefined:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DS2202_CAN_NO_ERROR</td><td>The function has been performed without error.</td></tr> <tr> <td>DS2202_CAN_BUFFER_OVERFLOW</td><td>The communication buffer from master to slave has overflowed. Repeat the function until it returns DS2202_CAN_NO_ERROR.</td></tr> </tbody> </table>	Predefined Symbol	Meaning	DS2202_CAN_NO_ERROR	The function has been performed without error.	DS2202_CAN_BUFFER_OVERFLOW	The communication buffer from master to slave has overflowed. Repeat the function until it returns DS2202_CAN_NO_ERROR.		
Predefined Symbol	Meaning								
DS2202_CAN_NO_ERROR	The function has been performed without error.								
DS2202_CAN_BUFFER_OVERFLOW	The communication buffer from master to slave has overflowed. Repeat the function until it returns DS2202_CAN_NO_ERROR.								
Function execution times	For information, refer to Function Execution Times on page 207.								
Example	<pre>ds2202_can_channel_BOff_return(canCh);</pre>								
Related topics	<p>References</p> <table> <tbody> <tr> <td>ds2202_can_channel_BOff_go</td><td>139</td></tr> <tr> <td>ds2202_can_channel_init</td><td>130</td></tr> <tr> <td>ds2202_can_channel_init_advanced</td><td>133</td></tr> <tr> <td>ds2202_canChannel</td><td>118</td></tr> </tbody> </table>	ds2202_can_channel_BOff_go	139	ds2202_can_channel_init	130	ds2202_can_channel_init_advanced	133	ds2202_canChannel	118
ds2202_can_channel_BOff_go	139								
ds2202_can_channel_init	130								
ds2202_can_channel_init_advanced	133								
ds2202_canChannel	118								

ds2202_can_channel_set

Syntax	<pre>Int32 ds2202_can_channel_set(const ds2202_canChannel* canCh, const UInt32 mask_type, const UInt32 mask_value);</pre>
Include file	Can2202.h
Purpose	To set a mask value or attribute for the specified CAN channel. Use this function to write the value to the specified CAN controller memory area. Use the returned

handle from the function `ds2202_can_channel_init` or `ds2202_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2202_canChannel` structure. (See [ds2202_canChannel](#) on page 118.)

mask_type Specifies the mask type; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_CHANNEL_SET_MASK15	Sets the Message 15 Mask Register.
DS2202_CAN_CHANNEL_SET_ARBMASK15	Sets the Arbitration Register for mailbox 15.
DS2202_CAN_CHANNEL_SET_TERMINATION	Set the termination resistor for the channel.
DS2202_CAN_CHANNEL_SET_BAUDRATE	Sets the baud rate of the selected channel during run time.

mask_value Specifies the value of the mask to be written; 0 = "don't care", 1 = "must match".

mask_type	mask_value
DS2202_CAN_CHANNEL_SET_ARBMASK15	Arbitration field for mailbox 15. Bit0 (on the right in mask_value) corresponds to bit ID0 in the arbitration field, Bit1 = ID1, ..., Bit28 = ID28.
DS2202_CAN_CHANNEL_SET_MASK15	For mailbox 15 only: Message 15 Mask Register. Bit0 (on the right in mask_value) corresponds to bit ID0 in the arbitration field, Bit1 = ID1, ..., Bit28 = ID28.
DS2202_CAN_CHANNEL_SET_TERMINATION	Use one of the following symbols to set the termination resistor: DS2202_CHANNEL_TERMINATION_ON or DS2202_CHANNEL_TERMINATION_OFF
DS2202_CAN_CHANNEL_SET_BAUDRATE	Sets the baud rate (in Baud). Valid range: 10,000 ... 1,000,000. NOTE: Some baud rates in the allowed range cannot be met. If the actual baud rate differs more than 1% from the one you specify, the function outputs a warning with the actual baud rate settings. Using CAN service functions, you can check the current bus status and whether the new baud rate parameters were changed correctly. Refer to CAN Service Functions on page 188.

For further information on the registers, refer to the CAN controller manual *C167 Derivatives User's Manual*.

Return value

This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	The communication buffer from master to slave has overflowed. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_BAUDRATE_L_ERROR	The baud rate is too low. The operation is aborted.

Predefined Symbol	Meaning
DS2202_CAN_BAUDRATE_H_ERROR	The baud rate is too high. The operation is aborted.
DS2202_CAN_BAUDRATE_SET_BAUDR_ERROR	Error during calculation of the new bit timing parameters: The operation is aborted.

Messages

The following messages are defined:

Type	Message	Description
Warning	CAN2202 (0x y,...): baudrate on channel ... doesn't match the desired baudrate. New baudrate = ... bit/s (y: board index)	The actual baud rate differs more than 1% from the one you specify.

Example

```
ds2202_can_channel_set(
    canCh,
    DS2202_CAN_CHANNEL_SET_MASK15,
    0xFFFFFFFF);
/* Set the lowest bit of the Message 15 Mask Register */
/* to "don't care" */
```

Related topics**References**

```
ds2202_can_channel_init..... 130
ds2202_can_channel_init_advanced..... 133
ds2202_canChannel..... 118
```

ds2202_can_channel_txqueue_clear

Syntax

```
Int32 ds2202_can_channel_txqueue_clear(
    const ds2202_canChannel* canCh);
```

Include file

Can2202.h

Purpose

To clear the content of the transmit queues of the selected CAN channel.

Description

The function clears the content of the transmit queues of the selected CAN channel.

Note

When you use this function, all the TX messages in the transmit queues are deleted.

Parameters

canCh Specifies the pointer to the **ds2202_canChannel1** structure.

Return value

This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2220_CAN_BUFFER_OVERFLOW	The communication buffer from master to slave has overflowed. Repeat the function until it returns DS2202_CAN_NO_ERROR.

Related topics**References**

[ds2202_canChannel](#)..... 118

CAN Message Handling

Introduction

Information on how to deal with all kind of CAN messages.

Where to go from here

Information in this section

ds2202_can_msg_tx_register.....	146
To register a transmit message.	
ds2202_can_msg_rx_register.....	150
To register a receive message.	
ds2202_can_msg_rqtz_register.....	154
To register a request transmission message.	
ds2202_can_msg_rqrz_register.....	157
To register a request-receive message.	
ds2202_can_msg_rm_register.....	160
To register a remote message.	
ds2202_can_msg_set.....	164
To set the properties of a CAN message.	
ds2202_can_msg_rqtz_activate.....	167
To activate a request transmission message for cyclic or acyclic sending.	
ds2202_can_msg_write.....	168
To write the related message to the dual-port memory and start the cyclic sending.	
ds2202_can_msg_send.....	169
To write the related message to the dual-port memory and send it immediately or after the delay time.	
ds2202_can_msg_send_id.....	171
To write the one specific message to the dual-port memory and send it immediately or after the delay time.	
ds2202_can_msg_queue_level.....	172
To get the number of messages in a message queue.	
ds2202_can_msg_txqueue_init.....	173
To initialize transmit message queues.	
ds2202_can_msg_send_id_queued.....	175
To build a transmit order.	
ds2202_can_msg_txqueue_level_read.....	177
To read the fill level of the transmit queue for the specified TX message on the CAN slave.	
ds2202_can_msg_sleep.....	178
To set a message into sleep mode.	

ds2202_can_msg_wakeup	179
To reactivate a message.	
ds2202_can_msg_read	180
To read data, data length, and status information from the dual-port memory.	
ds2202_can_msg_trigger	182
To send a transmit or request message immediately or after the delay time.	
ds2202_can_msg_clear	183
To clear message data.	
ds2202_can_msg_processed_register	184
To register the message processed function.	
ds2202_can_msg_processed_request	185
To request the message processed information from the slave.	
ds2202_can_msg_processed_read	186
To read the message processed information from the slave.	

ds2202_can_msg_tx_register

Syntax

```
ds2202_canMsg* ds2202_can_msg_tx_register(
    const ds2202_canChannel* canCh,
    const Int32 queue,
    const UInt32 identifier,
    const UInt32 format,
    const UInt32 inform,
    const Int32 subinterrupt,
    const Float32 start_time,
    const Float32 repetition_time,
    const Float32 timeout);
```

Include file

Can2202.h

Purpose

To register a transmit message on the slave DS2202.

Description

If no error occurs, `ds2202_can_msg_tx_register` returns a pointer to the `ds2202_canMsg` structure. (See [ds2202_canMsg](#) on page 123.)

Use the returned handle when calling one of the following functions:

- **ds2202_can_msg_write** to write new data to the message
- **ds2202_can_msg_read** to read the returned timestamps
- **ds2202_can_msg_send** to send the message with new data
- **ds2202_can_msg_trigger** to send the message
- **ds2202_can_msg_sleep** to deactivate the message
- **ds2202_can_msg_wakeup** to reactivate the message
- **ds2202_can_msg_clear** to clear the message object data
- **ds2202_can_msg_processed_register** to register the processed function
- **ds2202_can_msg_processed_request** to request the processed function
- **ds2202_can_msg_processed_read** to read the returned data

Note

You must call **ds2202_can_msg_write** to make the message valid for the CAN channel.

Parameters

canCh Specifies the Pointer to the **ds2202_canChannel1** structure. (See [ds2202_canChannel](#) on page 118.)

queue Specifies the communication channel within the range 0 ... 5.

identifier Specifies the identifier of the message.

format Specifies the message format. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_STD	11-bit standard format, CAN 2.0A
DS2202_CAN_EXT	29-bit extended format, CAN 2.0B

inform Specifies the information values to be updated. You can combine the predefined symbols with the logical operator OR. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_INFO	Returns no information.
DS2202_CAN_MSG_INFO	Updates the message identifier and the message format.

Predefined Symbol	Meaning
DS2202_CAN_TIMECOUNT_INFO	Updates the timestamp and the deltatime.
DS2202_CAN_DELAYCOUNT_INFO	Updates the parameter delaytime.

subinterrupt Specifies the subinterrupt number for a received message. Valid range is 0 ... 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (DS2202_CAN_SUBINT_BUFFERWARN). You must not use this number for any other interrupt.

Use the following predefined symbol to select no interrupt for the TX message:

Predefined Symbol	Meaning
DS2202_CAN_NO_SUBINT	No interrupt for the TX message

start_time Specifies the point of time for the first sending after the timer start, given in seconds. Valid range is 0 ... 420.

repetition_time Specifies the time for repeating the message automatically. Enter the value in seconds within the range of 0 ... 100. Use the following predefined symbol to define a message sent only once with **ds2202_can_msg_trigger**:

Predefined Symbol	Meaning
DS2202_CAN_TRIGGER_MSG	Call ds2202_can_msg_trigger to send the message.

timeout The message will occupy the mailbox that is used only up to this time. When the threshold is exceeded, the message is released from the mailbox. Enter the value in seconds within the range of 0 ... 100. Use the following predefined symbol to calculate the timeout value internally:

Predefined Symbol	Meaning
DS2202_CAN_TIMEOUT_NORMAL	The timeout value is calculated internally when registering the message. In most cases this timeout value will work.

Return value

canMsg Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2202_can_msg_tx_register(x,...) memory allocation error on master	Memory allocation error. No free memory on the master.

ID	Type	Message	Description
102	Error	ds2202_can_msg_tx_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2202_can_msg_tx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2202_CAN_AUTO_INDEX.
104	Error	ds2202_can_msg_tx_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	ds2202_can_msg_tx_register(x,..) slave: not responding	The slave did not finish the initialization within one second.
107	Error	ds2202_can_msg_tx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2202_can_msg_tx_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with ds2202_can_msg_sleep on page 178 or ds2202_can_channel_all_sleep when registering messages or services.
140	Error	ds2202_can_msg_tx_register(x,..) format: wrong format	Only the symbols DS2202_CAN_STD and DS2202_CAN_EXT are allowed for the parameter format .
141	Error	ds2202_can_msg_tx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range of 0 ... 14.
142	Error	ds2202_can_msg_tx_register(x,..) subint: used for busoff!	The given subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2202_can_msg_tx_register(x,..) id: Illegal id or id conflict	The CAN controller does not install the identifier given in the program. For further information, see ds2202_canService .
144	Error	ds2202_can_msg_tx_register(x,..) Too many messages (max. 100)!	The total number of registered messages is limited to 100. The program will be aborted.
145	Error	ds2202_can_msg_tx_register(x,..) starttime: too high (max. 420s)!	The start_time value must not be higher than 420 seconds. Exceeding this value causes an error and the program will be aborted.
146	Error	ds2202_can_msg_tx_register(x,..) rep. time: too high (max. 100s)!	The repetition_time value must not be higher than 100 seconds. Exceeding this value causes an error and the program will be aborted.
147	Error	ds2202_can_msg_tx_register(x,..) rep. time: too low !	At least CAN_FRAME_TIME. A lower value causes an error and the program will be aborted. Note that CAN_FRAME_TIME = (136 / Baud rate).
148	Error	ds2202_can_msg_tx_register(x,..) timeout: too high (max. 100s)!	The timeout value must not be higher than 100 seconds. Exceeding this value causes an error and the program will be aborted.
149	Error	ds2202_can_msg_tx_register(x,..) timeout: too low !	The timeout value has to be at least 3 · CAN_FRAME_TIME. A lower value causes an error and the program will be aborted. Note that CAN_FRAME_TIME = (136 / Baud rate).
152	Error	ds2202_can_msg_tx_register(x,..) canCh: the CAN channel wasn't initialized	This message will be displayed if: <ul style="list-style-type: none"> you try to register a CAN message on an uninitialized CAN channel.

ID	Type	Message	Description
			<ul style="list-style-type: none"> ▪ you try to register a CAN service on an uninitialized CAN channel. Use <code>ds2202_can_channel_init</code> or <code>ds2202_can_channel_init_advanced</code> to initialize the CAN channel.

Related topics

Examples

Example of Handling Transmit and Receive Messages.....	198
Example of Using Subinterrupts.....	202

References

<code>ds2202_can_channel_all_sleep</code>	137
<code>ds2202_can_channel_init</code>	130
<code>ds2202_can_channel_init_advanced</code>	133
<code>ds2202_can_msg_clear</code>	183
<code>ds2202_can_msg_processed_read</code>	186
<code>ds2202_can_msg_processed_register</code>	184
<code>ds2202_can_msg_processed_request</code>	185
<code>ds2202_can_msg_read</code>	180
<code>ds2202_can_msg_send</code>	169
<code>ds2202_can_msg_sleep</code>	178
<code>ds2202_can_msg_trigger</code>	182
<code>ds2202_can_msg_wakeup</code>	179
<code>ds2202_can_msg_write</code>	168
<code>ds2202_canChannel</code>	118
<code>ds2202_canMsg</code>	123
<code>ds2202_canService</code>	119

ds2202_can_msg_rx_register

Syntax

```
ds2202_canMsg* ds2202_can_msg_rx_register(
    const ds2202_canChannel* canCh,
    const Int32 queue,
    const UInt32 identifier,
    const UInt32 format,
    const UInt32 inform,
    const Int32 subinterrupt);
```

Include file

Can2202.h

Purpose

To register a receive message on the slave DS2202.

Description

If no error occurs `ds2202_can_msg_rx_register` returns a pointer to the `ds2202_canMsg` structure. (See [ds2202_canMsg](#) on page 123.)

Use the returned handle when calling one of the following functions:

- `ds2202_can_msg_read` to read the returned data and timestamps
- `ds2202_can_msg_sleep` to deactivate the message
- `ds2202_can_msg_wakeup` to reactivate the message
- `ds2202_can_msg_clear` to clear the message data
- `ds2202_can_msg_processed_register` to register the processed function
- `ds2202_can_msg_processed_request` to request the processed function
- `ds2202_can_msg_processed_read` to read the returned data

Parameters

canCh Specifies the pointer to the `ds2202_canChannel` structure. (See [ds2202_canChannel](#) on page 118.)

queue Specifies the communication channel within the range 0 ... 5.

identifier Specifies the identifier of the message.

format Specifies the message format. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_STD	11-bit standard format, CAN 2.0A
DS2202_CAN_EXT	29-bit extended format, CAN 2.0B

inform Specifies the information values to be updated. You can combine the predefined symbols with logical operators (OR); the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_INFO	Returns no information.
DS2202_CAN_DATA_INFO	Updates the parameters data and datalen (needed for receive and request (RQRX) messages).
DS2202_CAN_MSG_INFO	Updates the message identifier and the message format.
DS2202_CAN_TIMECOUNT_INFO	Updates the parameters timestamp and the deltatime.

subinterrupt Specifies the subinterrupt number for a received message. Valid range is 0 ... 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (`DS2202_CAN_SUBINT_BUFFERWARN`). You must not use this number for any other interrupt.

Use the following predefined symbol to select no interrupt for the receive message:

Predefined Symbol	Meaning
DS2202_CAN_NO_SUBINT	No interrupt for the receive message.

Return value

canMsg This function returns the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2202_can_msg_rx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2202_can_msg_rx_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2202_can_msg_rx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2202_CAN_AUTO_INDEX.
104	Error	ds2202_can_msg_rx_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	ds2202_can_msg_rx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2202_can_msg_rx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2202_can_msg_rx_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with ds2202_can_msg_sleep or ds2202_can_channel_all_sleep when registering messages or services.
140	Error	ds2202_can_msg_rx_register(x,..) format: wrong format	Only the symbols DS2202_CAN_STD and DS2202_CAN_EXT are allowed for the parameter format .
141	Error	ds2202_can_msg_rx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range of 0 ... 14.
142	Error	ds2202_can_msg_rx_register(x,..) subint: used for busoff!	The given subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2202_can_msg_rx_register(x,..) id: Illegal id or id conflict	The CAN controller does not install the identifier given in the program. For further information, see ds2202_canService .
144	Error	ds2202_can_msg_rx_register(x,..): Too many messages (max. 100)!	The total number of registered messages is limited to 100. The program will be aborted.
152	Error	ds2202_can_msg_rx_register(x,..) canCh: the CAN channel wasn't initialized	This message will be displayed if: <ul style="list-style-type: none"> you try to register a CAN message on an uninitialized CAN channel.

ID	Type	Message	Description
			<ul style="list-style-type: none">▪ you try to register a CAN service on an uninitialized CAN channel. Use <code>ds2202_can_channel_init</code> or <code>ds2202_can_channel_init_advanced</code> to initialize the CAN channel.

Example

```
ds2202_canMsg* rxMsg = ds2202_can_msg_rx_register(  
    canCh,  
    0,  
    0x123,  
    DS2202_CAN_STD,  
    DS2202_CAN_DATA_INFO,  
    DS2202_CAN_NO_SUBINT);
```

Related topics

Examples

Example of Handling Transmit and Receive Messages.....	198
Example of Using Subinterrupts.....	202

References

ds2202_can_channel_all_sleep.....	137
ds2202_can_channel_init.....	130
ds2202_can_channel_init_advanced.....	133
ds2202_can_msg_clear.....	183
ds2202_can_msg_processed_read.....	186
ds2202_can_msg_processed_register.....	184
ds2202_can_msg_processed_request.....	185
ds2202_can_msg_read.....	180
ds2202_can_msg_sleep.....	178
ds2202_can_msg_trigger.....	182
ds2202_can_msg_wakeup.....	179
ds2202_canChannel.....	118
ds2202_canMsg.....	123
ds2202_canService.....	119

ds2202_can_msg_rqtx_register

Syntax

```
ds2202_canMsg* ds2202_can_msg_rqtx_register(
    const ds2202_canChannel* canCh,
    const Int32 queue,
    const UInt32 identifier,
    const UInt32 format,
    const UInt32 inform,
    const Int32 subinterrupt,
    const Float32 start_time,
    const Float32 repetition_time,
    const Float32 timeout);
```

Include file

Can2202.h

Purpose

To register a request transmission (RQTX) message on the slave DS2202.

Description

Use this function to register a request message. Use the function **ds2202_can_msg_rqrx_register** to register a function that receives the requested data. If no error occurs **ds2202_can_msg_rqtx_register** returns a pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Use the returned handle when calling one of the following functions:

- **ds2202_can_msg_rqtx_activate** to activate the message
- **ds2202_can_msg_read** to read the returned timestamps
- **ds2202_can_msg_sleep** to deactivate the message
- **ds2202_can_msg_wakeup** to reactivate the message
- **ds2202_can_msg_trigger** to send the request message
- **ds2202_can_msg_clear** to clear the message object data
- **ds2202_can_msg_processed_register** to register the processed function
- **ds2202_can_msg_processed_request** to request the processed function
- **ds2202_can_msg_processed_read** to read the returned data

Note

You must call **ds2202_can_msg_rqtx_activate** to make the message valid for the CAN channel.

Parameters

canCh Specifies the pointer to the **ds2202_canChannel** structure. (See [ds2202_canChannel](#) on page 118.)

queue Specifies the communication channel within the range of 0 ... 5.

identifier Specifies the identifier of the message.

format Specifies the message format; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_STD	11-bit standard format, CAN 2.0A
DS2202_CAN_EXT	29-bit extended format, CAN 2.0B

inform Specifies the information values to be updated. You can combine the predefined symbols with the logical operator OR; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_INFO	Returns no information.
DS2202_CAN_MSG_INFO	Updates the message identifier and the message format.
DS2202_CAN_TIMECOUNT_INFO	Updates the parameters timestamp and the deltatime.
DS2202_CAN_DELAYCOUNT_INFO	Updates the parameter delaytime.

subinterrupt Specifies the subinterrupt number for a received message. Valid range is 0 ... 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (DS2202_CAN_SUBINT_BUFFERWARN). You must not use this number for any other interrupt.

Use the following predefined symbol to select no interrupt for the RQTX message:

Predefined Symbol	Meaning
DS2202_CAN_NO_SUBINT	No interrupt for the RQTX messages.

start_time Specifies the point of time for the first sending after timerstart. Enter the value in seconds within the range of 0 ... 420.

repetition_time Specifies the time for repeating the message automatically. Enter the value in seconds within the range of 0 ... 100. Use the following predefined symbol to define a message sent only once with **ds2202_can_msg_trigger**:

Predefined Symbol	Meaning
DS2202_CAN_TRIGGER_MSG	Calls ds2202_can_msg_trigger to send the message.

timeout The message will occupy the mailbox that is used only up to this time. When the threshold is exceeded, the message will be released from the mailbox. Valid range is 0 ... 100 s. Use the following predefined symbol to calculate the timeout value internally:

Predefined Symbol	Meaning
DS2202_CAN_TIMEOUT_NORMAL	The timeout value is calculated internally when registering the message. In most cases this timeout value will work.

Return value `canMsg` This function returns the pointer to the `ds2202_canMsg` structure. (See `ds2202_canMsg` on page 123.)

Messages The following messages are defined:

ID	Type	Message	Description
101	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> index: illegal function index	The index does not exist in the command table and is not equal to <code>DS2202_CAN_AUTO_INDEX</code> .
104	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with <code>ds2202_can_msg_sleep</code> or <code>ds2202_can_channel_all_sleep</code> when registering messages or services.
140	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> format: wrong format	Only the symbols <code>DS2202_CAN_STD</code> and <code>DS2202_CAN_EXT</code> are allowed for the parameter format.
141	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> subint: use range 0..14 !	The subinterrupt number must be within the range of 0 ... 14.
142	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> subint: used for busoff!	The given subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> id: Illegal id or id conflict	The CAN controller does not install the identifier given in the program. For further information, refer to <code>DS2202_CAN_SERVICE_MAILBOX_ERR</code> .
144	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> : Too many messages (max. 100)!	The total number of registered messages is limited to 100. The program will be aborted.
152	Error	<code>ds2202_can_msg_rqt_x_register(x,...)</code> canCh: the CAN channel wasn't initialized	This message will be displayed if: <ul style="list-style-type: none"> you try to register a CAN message on an uninitialized CAN channel.

ID	Type	Message	Description
			<ul style="list-style-type: none"> you try to register a CAN service on an uninitialized CAN channel. Use <code>ds2202_can_channel_init</code> or <code>ds2202_can_channel_init_advanced</code> to initialize the CAN channel.

Example

```
ds2202_canMsg* rqtMsg = ds2202_can_msg_rqt_register(
    canCh,
    0,
    0x123,
    DS2202_CAN_STD,
    DS2202_CAN_TIMECOUNT_INFO,
    DS2202_CAN_NO_SUBINT,
    1.5,
    0.3,
    DS2202_CAN_TIMEOUT_NORMAL);
```

Related topics**Examples**

Example of Handling Request and Remote Messages..... 200

References

[ds2202_can_channel_all_sleep](#)..... 137
[ds2202_can_channel_init](#)..... 130
[ds2202_can_channel_init_advanced](#)..... 133
[ds2202_can_msg_clear](#)..... 183
[ds2202_can_msg_processed_read](#)..... 186
[ds2202_can_msg_processed_register](#)..... 184
[ds2202_can_msg_processed_request](#)..... 185
[ds2202_can_msg_read](#)..... 180
[ds2202_can_msg_rqr_register](#)..... 157
[ds2202_can_msg_rqt_activate](#)..... 167
[ds2202_can_msg_sleep](#)..... 178
[ds2202_can_msg_trigger](#)..... 182
[ds2202_can_msg_wakeup](#)..... 179
[ds2202_canChannel](#)..... 118
[ds2202_canMsg](#)..... 123

ds2202_can_msg_rqr_register

Syntax

```
ds2202_canMsg* ds2202_can_msg_rqr_register(
    const ds2202_canMsg* rqtMsg,
    const UInt32 inform,
    const Int32 subinterrupt);
```

Include file	Can2202.h										
Purpose	To register a RQRX message on the slave DS2202.										
Description	<p>Use this message to receive the data requested with an RQTX message. If no error occurs, <code>ds2202_can_msg_rqr_register</code> returns a pointer to the <code>ds2202_canMsg</code> structure. (See ds2202_canMsg on page 123.)</p> <p>Use the returned handle when calling one of the following functions:</p> <ul style="list-style-type: none"> ▪ <code>ds2202_can_msg_read</code> to read the returned data and timestamps ▪ <code>ds2202_can_msg_sleep</code> to deactivate the message ▪ <code>ds2202_can_msg_wakeup</code> to reactivate the message ▪ <code>ds2202_can_msg_clear</code> to clear the message object data ▪ <code>ds2202_can_msg_processed_register</code> to register the processed function ▪ <code>ds2202_can_msg_processed_request</code> to request the processed function ▪ <code>ds2202_can_msg_processed_read</code> to read the returned data 										
Parameters	<p>rqrMsg Specifies the pointer to the related RQTX message.</p> <p>inform Specifies the information values to be updated. You can combine the predefined symbols with the logical operator OR. The following symbols are predefined:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DS2202_CAN_NO_INFO</td><td>Returns no information.</td></tr> <tr> <td>DS2202_CAN_DATA_INFO</td><td>Updates the parameters data and datalen (needed for receive and request (RQRX) messages).</td></tr> <tr> <td>DS2202_CAN_MSG_INFO</td><td>Updates the message identifier and the message format.</td></tr> <tr> <td>DS2202_CAN_TIMECOUNT_INFO</td><td>Updates the parameters timestamp and the deltatime.</td></tr> </tbody> </table> <p>subinterrupt Specifies the subinterrupt number for a received message. Valid range is 0 ... 14.</p>	Predefined Symbol	Meaning	DS2202_CAN_NO_INFO	Returns no information.	DS2202_CAN_DATA_INFO	Updates the parameters data and datalen (needed for receive and request (RQRX) messages).	DS2202_CAN_MSG_INFO	Updates the message identifier and the message format.	DS2202_CAN_TIMECOUNT_INFO	Updates the parameters timestamp and the deltatime.
Predefined Symbol	Meaning										
DS2202_CAN_NO_INFO	Returns no information.										
DS2202_CAN_DATA_INFO	Updates the parameters data and datalen (needed for receive and request (RQRX) messages).										
DS2202_CAN_MSG_INFO	Updates the message identifier and the message format.										
DS2202_CAN_TIMECOUNT_INFO	Updates the parameters timestamp and the deltatime.										
<p>Note</p> <p>The interrupt number 15 is occupied by the buffer overflow warning interrupt (DS2202_CAN_SUBINT_BUFFERWARN). You must not use this number for any other interrupt.</p>											

Use the following predefined symbol to select no interrupt for the RQRX message:

Predefined Symbol	Meaning
DS2202_CAN_NO_SUBINT	No interrupt for the RQRX message.

Return value **canMsg** Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Messages The following messages are defined:

ID	Type	Message	Description
101	Error	ds2202_can_msg_rqr_register	Memory allocation error. No free memory on the master.
102	Error	ds2202_can_msg_rqr_register(x,...) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2202_can_msg_rqr_register(x,...) index: illegal function index	The index does not exist in the command table and is not equal to DS2202_CAN_AUTO_INDEX.
104	Error	ds2202_can_msg_rqr_register(x,...) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	ds2202_can_msg_rqr_register(x,...) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2202_can_msg_rqr_register(x,...) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2202_can_msg_rqr_register(x,...) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with ds2202_can_msg_sleep or ds2202_can_channel1_all_sleep when registering messages or services.
140	Error	ds2202_can_msg_rqr_register(x,...) format: wrong format	Only the symbols DS2202_CAN_STD and DS2202_CAN_EXT are allowed for the parameter format.
141	Error	ds2202_can_msg_rqr_register(x,...) subint: use range 0..14 !	The subinterrupt number must be within the range of 0 ... 14.
142	Error	ds2202_can_msg_rqr_register(x,...) subint: used for busoff!	The given subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2202_can_msg_rqr_register(x,...) id: Illegal id or id conflict	The CAN controller does not install the identifier given in the program. For further information, see ds2202_canService .
144	Error	ds2202_can_msg_rqr_register(x,...): Too many messages (max. 100)!	The total number of registered messages is limited to 100. The program will be aborted.
152	Error	ds2202_can_msg_rqr_register(x,...) canCh: the CAN channel wasn't initialized	This message will be displayed if: <ul style="list-style-type: none"> ▪ you try to register a CAN message on an uninitialized CAN channel.

ID	Type	Message	Description
			<ul style="list-style-type: none"> you try to register a CAN service on an uninitialized CAN channel. Use <code>ds2202_can_channel_init</code> or <code>ds2202_can_channel_init_advanced</code> to initialize the CAN channel.

Example

```
ds2202_canMsg* rqrMsg = ds2202_can_msg_rqr_register(
    rqtMsg,
    DS2202_CAN_DATA_INFO,
    DS2202_CAN_NO_SUBINT);
```

Related topics**Examples**

[Example of Handling Request and Remote Messages.....](#) 200

References

[ds2202_can_channel_all_sleep.....](#) 137
[ds2202_can_channel_init.....](#) 130
[ds2202_can_channel_init_advanced.....](#) 133
[ds2202_can_msg_clear.....](#) 183
[ds2202_can_msg_processed_read.....](#) 186
[ds2202_can_msg_processed_register.....](#) 184
[ds2202_can_msg_processed_request.....](#) 185
[ds2202_can_msg_read.....](#) 180
[ds2202_can_msg_sleep.....](#) 178
[ds2202_can_msg_wakeup.....](#) 179
[ds2202_canMsg.....](#) 123
[ds2202_canService.....](#) 119

ds2202_can_msg_rm_register

Syntax

```
ds2202_canMsg* ds2202_can_msg_rm_register(
    const ds2202_canChannel* canCh,
    const Int32 queue,
    const UInt32 identifier,
    const UInt32 format,
    const UInt32 inform,
    const Int32 subinterrupt);
```

Include file

Can2202.h

Purpose

To register a remote message on the slave DS2202.

Description

If no error occurs, `ds2202_can_msg_rm_register` returns a pointer to the `ds2202_canMsg` structure. (See [ds2202_canMsg](#) on page 123.)

Use the returned handle when calling one of the following functions:

- `ds2202_can_msg_write` to support the remote message with data
- `ds2202_can_msg_read` to read the returned timestamps
- `ds2202_can_msg_sleep` to deactivate the message
- `ds2202_can_msg_wakeup` to reactivate the message
- `ds2202_can_msg_clear` to clear the message object data
- `ds2202_can_msg_processed_register` to register the processed function
- `ds2202_can_msg_processed_request` to request the processed function
- `ds2202_can_msg_processed_read` to read the returned data

A remote message is a special kind of a transmit message. It is sent only if the CAN controller has received a corresponding request message and carries the requested data.

Note

A remote message permanently occupies a mailbox on the slave DS2202 CAN channel. Therefore, for each CAN channel only 10 remote messages are allowed within the same model to guarantee secure CAN operation. If this is not done, the function outputs an error and aborts the program.

Parameters

canCh Specifies the pointer to the `ds2202_canChannel` structure. (See [ds2202_canChannel](#) on page 118.)

queue Specifies the communication channel within the range of 0 ... 5.

identifier Specifies the identifier of the message.

format Specifies the message format. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_STD	11-bit standard format, CAN 2.0A
DS2202_CAN_EXT	29-bit extended format, CAN 2.0B

inform Specifies the information values to be updated. You can combine the predefined symbols with the logical operator OR. The following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_INFO	Returns no information.
DS2202_CAN_MSG_INFO	Updates the message identifier and the message format.

Predefined Symbol	Meaning
DS2202_CAN_TIMECOUNT_INFO	Updates the parameters timestamp and the deltatime.
DS2202_CAN_DELAYCOUNT_INFO	Updates the parameter delaytime.

subinterrupt Specifies the subinterrupt number for a received message. Valid range is 0 ... 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (DS2202_CAN_SUBINT_BUFFERWARN). You must not use this number for any other interrupt.

Use the following predefined symbol to select no interrupt for the RM message:

Predefined Symbol	Meaning
DS2202_CAN_NO_SUBINT	No interrupt for the RM message

Return value

canMsg This function returns the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Messages

The following error and warning messages are defined:

ID	Type	Message	Description
101	Error	ds2202_can_msg_rm_register(x,...) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2202_can_msg_rm_register(x,...) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2202_can_msg_rm_register(x,...) index: illegal function index	The index does not exist in the command table and is not equal to DS2202_CAN_AUTO_INDEX.
104	Error	ds2202_can_msg_rm_register(x,...) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	ds2202_can_msg_rm_register(x,...) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2202_can_msg_rm_register(x,...) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2202_can_msg_rm_register(x,...) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with ds2202_can_msg_sleep or ds2202_can_channel_all_sleep when registering messages or services.
140	Error	ds2202_can_msg_rm_register(x,...) format: wrong format	Only the symbols DS2202_CAN_STD and DS2202_CAN_EXT are allowed for the parameter format.

ID	Type	Message	Description
141	Error	ds2202_can_msg_rm_register(x,...) subint: use range 0..14 !	The subinterrupt number must be within the range of 0 ... 14.
142	Error	ds2202_can_msg_rm_register(x,...) subint: used for busoff!	The given subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2202_can_msg_rm_register(x,...) id: Illegal id or id conflict	The CAN controller does not install the identifier given in the program. For further information, see ds2202_canService .
144	Error	ds2202_can_msg_rm_register(x,...) Too many messages (max. 100)!	The total number of registered messages is limited to 100. The program will be aborted.
150	Error	ds2202_can_msg_rm_register(x,...) no rm mailbox free (max. 10)!	For each channel only 10 remote messages are allowed within the same model to guarantee secure CAN operation. If this is not done, the function outputs an error and the program will be aborted.
152	Error	ds2202_can_msg_rm_register(x,...) canCh: the CAN channel wasn't initialized	<p>This message will be displayed if:</p> <ul style="list-style-type: none"> ▪ you try to register a CAN message on an uninitialized CAN channel. ▪ you try to register a CAN service on an uninitialized CAN channel. <p>Use ds2202_can_channel_init or ds2202_can_channel_init_advanced to initialize the CAN channel.</p>

Example

```
ds2202_canMsg* rmMsg = ds2202_can_msg_rm_register(
    canCh,
    0,
    0x123,
    DS2202_CAN_STD,
    DS2202_CAN_TIMECOUNT_INFO,
    DS2202_CAN_NO_SUBINT);
```

Related topics**Examples**

[Example of Handling Request and Remote Messages.....](#) 200

References

[ds2202_can_channel_all_sleep.....](#) 137
[ds2202_can_channel_init.....](#) 130
[ds2202_can_channel_init_advanced.....](#) 133
[ds2202_can_msg_clear.....](#) 183
[ds2202_can_msg_processed_read.....](#) 186
[ds2202_can_msg_processed_register.....](#) 184
[ds2202_can_msg_processed_request.....](#) 185
[ds2202_can_msg_read.....](#) 180
[ds2202_can_msg_sleep.....](#) 178
[ds2202_can_msg_wakeup.....](#) 179
[ds2202_can_msg_write.....](#) 168
[ds2202_canChannel.....](#) 118
[ds2202_canMsg.....](#) 123
[ds2202_canService.....](#) 119

ds2202_can_msg_set

Syntax

```
Int32 ds2202_can_msg_set(
    ds2202_canMsg* msg,
    const UInt32 type,
    const void* value);
```

Include file

Can2202.h

Purpose

To set the properties of a CAN message.

Description

This function allows you to

- Receive different message IDs with one message via a bitmask (type = DS2202_CAN_MSG_MASK),
- Set the send period for a TX or RQ message (type = DS2202_CAN_MSG_PERIOD),
- Set the identifier for a TX or RQ message (type = DS2202_CAN_MSG_ID) or
- Set the queue depth for a message (type = DS2202_CAN_MSG_QUEUE).
- Set the message length for a message (type = DS2202_CAN_MSG_LEN).

Note

For DS2202_CAN_MSG_MASK the following rules apply:

- For each CAN channel only one mask for STD and one mask for EXT messages is allowed.
- If you call `ds2202_can_msg_set` for another message, the bitmask will be removed from the first message.
- Using the bitmask may cause conflicts with messages installed for one message ID. In this case, message data will be received via the message installed for this ID.
- You can skip the bitmask setting all bits to "must match" (0xFFFFFFFF) again.

Parameters

msg Specifies the pointer to the message structure.

type Defines the property to be specified. Use one of the predefined symbols:

Predefined Symbol	Meaning
DS2202_CAN_MSG_MASK	To set the arbitrary mask for an RX message
DS2202_CAN_MSG_PERIOD	To set the send period for a TX or RQ message
DS2202_CAN_MSG_ID	To set the identifier for a TX or RQ message
DS2202_CAN_MSG_QUEUE	To set the queue depth for a message
DS2202_CAN_MSG_LEN	To set the data length code (DLC) for a TX, RQTX, or RM message

value Specifies the value to be set for the defined **type**.

For the DS2202_CAN_MSG_LEN type, you can specify the data length code (DLC) value (UInt32) in the range 0 ... 8 bytes.

Note

If the specified length exceeds 8 bytes, the function sets the length to 8 bytes.

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_TYPE_ERROR	The operation is not allowed for the given message object.
DS2202_CAN_MSG_TYPE_ERROR	The function is not available for the specified message type. It is available only for TX, RQTX, and RM messages.

Example of receiving different message IDs with one message

This example shows how to receive different message IDs with one message:

Install one message with a bitmask that allows you to set some bits of the mask to "don't care" via `ds2202_can_msg_set`.

```
UInt32 mask = 0xFFFFFFFF; // Sets the last four bits to
// "Don't Care".
ds2202_can_msg_set(msg, DS2202_CAN_MSG_MASK, &mask);
```

Example of receiving different message IDs with one message via a bitmask

This example shows how to receive different message IDs with one message via a bitmask:

- A message with ID 0x120 was registered. When you set the bitmask via:

```
ds2202_can_msg_set(msg, DS2202_CAN_MSG_MASK, &mask);
```

with `mask = 0xFFFFFFFF0`, you can receive the message IDs 0x120, 0x121, ..., 0x12F.

- A message with ID 0x120 was registered. When you set the bitmask to `0x1FFFFFFF`, you can receive the message IDs 0x120 and 0x130.

Example of applying the DS2202_CAN_MSG_QUEUE option

This example shows how to apply the `DS2202_CAN_MSG_QUEUE` option.

You can define a buffer for each message to receive several messages. Otherwise, only the most recently received message will be available.

- Register the message as usual

```
myMsg = ds2202_can_msg_xx_register(...)
```

By default, `myMsg` stores only one message.

- Define a message queue of length *n* for `myMsg`:

```
ds2202_can_msg_set(myMsg, DS2202_CAN_MSG_QUEUE, &n)
```

- Call `ds2202_can_msg_read(myMsg)` repeatedly until the function returns `DS2202_CAN_NO_DATA`.

```
UInt32 n;
canMsg = ds2202_can_msg_rx_register( canCh, ...
n = 5000;
ds2202_can_msg_set(canMsg, DS2202_CAN_MSG_QUEUE, &n);
...
while(DS2202_CAN_NO_DATA != (error = ds2202_can_msg_read(canMsg)))
{
    if(DS2202_CAN_DATA_LOST == error)
    {
        /* error handling */
    }
    else if(DS2202_CAN_NO_ERROR == error)
    {
        /* process the message */
    }
    else /* DS2202_CAN_NO_DATA == error */
    {
        /* no further CAN-messages */
    }
}
```

Related topics

References

[ds2202_can_msg_read](#)..... 180

ds2202_can_msg_rqt_x_activate

Syntax

```
Int32 ds2202_can_msg_rqt_x_activate(
    const ds2202_canMsg* canMsg);
```

Include file

Can2202.h

Purpose

To activate the request transmission message on the slave DS2202 registered by **ds2202_can_msg_rqt_x_register**.

Description

This function does not send the message. Sending the message is done by the timer for cyclic sending, or by calling **ds2202_can_msg_trigger** for acyclic sending. Use the returned handle from the function **ds2202_can_msg_rqt_x_register** to call this function.

Parameters

canMsg Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_TYPE_ERROR	The operation is not allowed for the given message object.

Related topics**Examples**

[Example of Handling Request and Remote Messages](#)..... 200

References

[ds2202_can_msg_rqt_x_register](#)..... 154
[ds2202_can_msg_trigger](#)..... 182
[ds2202_canMsg](#)..... 123

ds2202_can_msg_write

Syntax

```
Int32 ds2202_can_msg_write(
    const ds2202_canMsg* canMsg,
    const UInt32 datalen,
    const UInt32* data);
```

Include file

Can2202.h

Purpose

To write CAN message data.

Description

There are differences for the following message types:

- TX message

Calling this function for the first time prepares the message to be sent with the specified parameters in the message register function. A TX message with a repetition time will be sent automatically with the given value. A TX message registered by DS2202_CAN_TRIGGER_MSG will be sent only when calling **ds2202_can_msg_trigger** or **ds2202_can_msg_send**.

Calling this function again will update CAN message data and data length.

- RM message

Calling this function for the first time prepares and activates the remote message to be sent with the given data and data length. The remote message will be sent when a corresponding request message is received.

Calling this function again will update CAN message data and data length.

Use the returned handle from the function **ds2202_can_msg_tx_register** or **ds2202_can_msg_rm_register** to call this function.

Parameters

canMsg Specifies the pointer to the **ds2202_canMsg** structure. See [ds2202_canMsg](#) on page 123.

datalen Specifies the length of the message data. Valid range is 0 ... 8 bytes.

data Specifies the buffer for CAN message data.

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_TYPE_ERROR	The operation is not allowed for the given message object.

Function execution times For information, refer to [Function Execution Times](#) on page 207.

Related topics

Examples

Example of Handling Request and Remote Messages.....	200
Example of Handling Transmit and Receive Messages.....	198
Example of Using Subinterrupts.....	202

References

ds2202_can_msg_rm_register.....	160
ds2202_can_msg_send.....	169
ds2202_can_msg_trigger.....	182
ds2202_can_msg_tx_register.....	146
ds2202_canMsg.....	123

[ds2202_can_msg_send](#)

Syntax

```
Int32 ds2202_can_msg_send(  
    const ds2202_canMsg* canMsg,  
    const UInt32 datalen,  
    const UInt32* data,  
    const Float32 delay);
```

Include file Can2202.h

Purpose To write CAN message data and send the data immediately after the delaytime. To send the transmit message with new data.

Description The transmit message must have been registered by calling **ds2202_can_msg_tx_register**. Then **ds2202_can_msg_send** writes the CAN message data to the dual-port memory. After this, the message is set up on the CAN controller and the sending of the message is started. The message is sent according to the specified parameters in the register function.

Use the returned handle from the function **ds2202_can_msg_tx_register** to call this function.

Note

Suppose the **ds2202_can_msg_send** function is called twice. If the interval between the function calls is short, the second function call may occur *before* the TX message was sent by the first function call. In this case, the TX message is sent only once, with the data of the second function call.

Parameters

canMsg Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

datalen Specifies the length of the CAN message data. Valid range is 0 ... 8 bytes.

data Specifies the buffer for CAN message data.

delay Send the message after the delay time. Valid range is 0.0 ... 100.0 seconds.

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_TYPE_ERROR	The operation is not allowed for the given message object.

Function execution times For information, refer to [Function Execution Times](#) on page 207.

Example

```
UInt32 txData[8] = {1,2,3,4,5,6,7,8};
ds2202_can_msg_send(txMsg, 8, txData, 0.005);
```

Related topics**References**

ds2202_can_msg_tx_register	146
ds2202_canMsg	123

ds2202_can_msg_send_id

Syntax

```
Int32 ds2202_can_msg_send_id(
    ds2202_canMsg* canMsg,
    const UInt32 id,
    const UInt32 datalen,
    const UInt8* data,
    const Float32 delay);
```

Include file

Can2202.h

Purpose

To send a message with a modified identifier. This allows you to send any message ID with one registered message.

Parameters

canMsg Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

id Specifies the ID of the message to be modified.

datalen Specifies the length of the CAN message data. Valid range is 0 ... 8 bytes.

data Specifies the buffer for CAN message data.

delay Send the message after the delaytime. Valid range is 0.0 ... 100.0 seconds.

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_TYPE_ERROR	The operation is not allowed for the given message object.

Example

The `ds2202_can_msg_send_id` function allows you to send any message ID with one registered message.

```
ds2202_can_msg_send_id(msg, 0x123, data, 8, 0.001)
```

Note

- The message format is determined by the format the message was installed when it was used for the first time.
- You have to use a handshake mechanism to send a message via `ds2202_can_msg_send_id` to make sure that a message installed for the message object has already been sent.
- Each message object is buffered only once on the slave. This may cause conflicts when you try to send several message objects with different IDs.

Related topics**References**

ds2202_can_msg_queue_level	172
ds2202_can_msg_send	169
ds2202_canMsg	123

ds2202_can_msg_queue_level

Syntax

```
Int32 ds2202_can_msg_queue_level(  
    ds2202_canMsg* canMsg);
```

Include file

Can2202.h

Purpose

To return the number of messages stored in the message queue allocated on the master with `ds2202_can_msg_set(msg, DS2202_CAN_MSG_QUEUE, &size)`.

Description

Use `ds2202_can_msg_read` to copy the messages from the communication channel to the message buffer.

Note

This is not the number of messages in the DPMEM.

Parameters	canMsg Specifies the pointer to the ds2202_canMsg structure. (See ds2202_canMsg on page 123.)
Return value	This function returns the number of messages in the message queue.
Related topics	References <div>ds2202_can_msg_read..... 180</div> <div>ds2202_can_msg_set..... 164</div> <div>ds2202_canMsg..... 123</div>

ds2202_can_msg_txqueue_init

Syntax	<pre>Int32 ds2202_can_msg_txqueue_init(ds2202_canMsg* canMsg, const UInt32 overrun_policy, Float32 delay);</pre>						
Include file	Can2202.h						
Purpose	To initialize the transmit queue that is used to queue messages sent by the ds2202_can_msg_send_id_queued function.						
Description	The function allocates a circular buffer on the slave with the specified overrun policy, where the transmit orders from the ds2202_can_msg_send_id_queued function are stored. The queue stores up to 64 message entries.						
Parameter	canMsg Specifies the address where the ds2202_canMsg structure is stored (only one transmit queue for STD and one for EXT for each channel are possible). (See ds2202_canMsg on page 123.) overrun_policy Selects the overrun policy of the transmit queue. The following symbols are predefined: <table><tr><th>Predefined Symbol</th><th>Meaning</th></tr><tr><td>DS2202_CAN_TXQUEUE_OVERRUN_OVERWRITE</td><td>The oldest message is overwritten.</td></tr><tr><td>DS2202_CAN_TXQUEUE_OVERRUN_IGNORE</td><td>The oldest message is kept. The new message is lost.</td></tr></table>	Predefined Symbol	Meaning	DS2202_CAN_TXQUEUE_OVERRUN_OVERWRITE	The oldest message is overwritten.	DS2202_CAN_TXQUEUE_OVERRUN_IGNORE	The oldest message is kept. The new message is lost.
Predefined Symbol	Meaning						
DS2202_CAN_TXQUEUE_OVERRUN_OVERWRITE	The oldest message is overwritten.						
DS2202_CAN_TXQUEUE_OVERRUN_IGNORE	The oldest message is kept. The new message is lost.						

delay Specifies the delay between the messages of the transmit queue within the range 0.0 ... 10 s.

Note

- Even if a delay of 0 seconds is specified the distance between two message frames is greater than 0. The length of this gap depends on the load of the slave. If the delay is smaller than 0 the function sets the delay to 0. The real delay between two message frames may not be constant due to jitter. The jitter of the delay also depends on the load of the slave.
- Only two message objects (one STD and one EXT format message) for every channel can be used for queuing. Nevertheless **ds2202_can_msg_send_id_queued** allows the identifier of the message object to be changed.
- The function can be called again to change the delay or to assign the transmit queue to another message. The old messages in the transmit queue are lost (not transmitted) if the transmit queue is initialized again.

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The transmit queue was initialized successfully.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_TXQUEUE_INIT_NOT_REG_ERROR	The message (canMsg) was not registered. The operation is aborted.
DS2202_CAN_TXQUEUE_INIT_MSG_TYPE_ERROR	The message (canMsg) is not a TX message. The operation is aborted.

Messages

The following messages are defined:

ID	Type	Message	Description
154	Error	ds2202_can_msg_txqueue_init(): TX message is not registered	The message was not registered successfully.
155	Error	ds2202_can_msg_txqueue_init(): Not a TX message	The specified message is not a TX message.
301	Warning	ds2202_can_msg_txqueue_init(): delay time: Too high (max. 10s) Set to maximum	The delay time must be within the range 0 ... 10 s.

Example

The following example shows how to initialize a TX queue.

```
void main()
{
    ds2202_canMsg* txMsg;
    ...
    txMsg = ds2202_can_msg_tx_register( txCh,
                                       2, 0x1, DS2202_CAN_STD,
                                       DS2202_CAN_TIMECOUNT_INFO |
                                       DS2202_CAN_MSG_INFO,
                                       1, 0.0,
                                       DS2202_CAN_TRIGGER_MSG, 0 );
    ds2202_can_msg_txqueue_init (
        txMsg, DS2202_CAN_TXQUEUE_OVERRUN_OVERWRITE, 0.01);
    ...
}
```

Related topics**References**

ds2202_can_msg_send_id_queued	175
ds2202_canMsg	123

ds2202_can_msg_send_id_queued

Syntax

```
Int32 ds2202_can_msg_send_id_queued(
    ds2202_canMsg* canMsg,
    const UInt32 id,
    const UInt32 data_len,
    const UInt32* data);
```

Include file

Can2202.h

Purpose

To build a transmit order and transmit it in the same order as the function is called.

Description

If no queue overflow occurs each message is transmitted. In the case of queue overflow (number of messages is greater than 64), the newest message overwrites the oldest one or the oldest messages are kept while new messages are lost. See **ds2202_can_msg_txqueue_init**.

The service DS2202_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT allows the overflow counter of the transmit queue to be requested to check whether an overflow occurred.

Parameter	<p>canMsg Specifies the address where the <code>ds2202_canMsg</code> structure is stored. Must be the same as that used with <code>ds2202_can_msg_txqueue_init</code>. (See ds2202_canMsg on page 123.)</p> <p>id Specifies the CAN message identifier type (STD/EXT). The identifier type must correspond to the type (STD/EXT) of the registered message object. This allows the identifier of the message object to be changed during run time.</p> <p>data_len Specifies the length of data within the range 0 ... 8.</p> <p>data Specifies the message data.</p>
------------------	---

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The transmit queue was initialized successfully.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_SEND_ID_QUEUED_INIT_ERROR	The transmit queue for TX messages was not initialized.

Messages The following messages are defined:

ID	Type	Message	Description
153	Error	ds2202_can_msg_send_id_queued(): TX queue: Not initialized!	The transmit queue was not initialized.

Example The following example shows how to build a transmit sequence for a TX queue.

```
void main()
{
    ds2202_canMsg* txMsg;
    UInt32 txMsgData[8];
    ...
    txMsg = ds2202_can_msg_tx_register( txCh,
                                       2, 0x1, DS2202_CAN_STD,
                                       DS2202_CAN_TIMECOUNT_INFO |
                                       DS2202_CAN_MSG_INFO,
                                       1, 0.0,
                                       DS2202_CAN_TRIGGER_MSG, 0 );
    /* initialize a transmit-queue with delay = 0.01 s */
    ds2202_can_msg_txqueue_init (
        txMsg, DS2202_CAN_TXQUEUE_OVERRUN_OVERWRITE; 0.01);
}
```



```
...
/* Write three messages to the transmit-queue.*/
/* The first message is transmitted immediately. */
/* The following messages are transmitted with */
/* a timely distance of 0.01 s. */
txMsgData[0] = 0x01;
ds2202_can_msg_send_id_queued(txMsg, 0x12, 1, txMsgData);
txMsgData[0] = 0x02;
ds2202_can_msg_send_id_queued(txMsg, 0x13, 1, txMsgData);
txMsgData[0] = 0x03;
ds2202_can_msg_send_id_queued(txMsg, 0x14, 1, txMsgData);
...
}
```

Related topics

References

ds2202_can_msg_txqueue_init	173
ds2202_canMsg	123

ds2202_can_msg_txqueue_level_read

Syntax

```
UInt32 ds2202_can_msg_txqueue_level_read(
    ds2202_canMsg* canMsg);
```

Include file

Can2202.h

Purpose

To read the fill level of the transmit queue for the specified TX message on the CAN slave.

Description

The function reads the fill level of the transmit queue for the specified TX message on the CAN slave.

Note

The TX messages pending in the command queue between the CAN master and the CAN slave are not taken into account.

Parameter

canMsg Specifies the address where the **ds2202_canMsg** structure is stored (only one transmit queue for STD and one for EXT for each channel are possible). (See [ds2202_canMsg](#) on page 123.)

Return value **Level of TX-queue** The number of TX messages in the transmit queue on the CAN slave (0 ... 64).

Related topics**References**

[ds2202_canMsg](#)..... 123

ds2202_can_msg_sleep

Syntax

```
Int32 ds2202_can_msg_sleep(
    const ds2202_canMsg* canMsg);
```

Include file

Can2202.h

Purpose

The purpose depends on the message type:

- TX, RQTX, and RM messages
To stop the transmission of the message to the CAN bus.
 - RX and RQRX messages
To stop the transmission of the message data from the slave to the master.
-

Description

The message is deactivated and remains in the sleep mode until it will be reactivated by calling **ds2202_can_msg_wakeup** or **ds2202_can_channel_all_wakeup**.

Parameters

canMsg Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_TYPE_ERROR	The operation is not allowed for the given message object.

Function execution times For information, refer to [Function Execution Times](#) on page 207.

Example `ds2202_can_msg_sleep(txMsg);`

Related topics

References

ds2202_can_channel_all_sleep	137
ds2202_can_channel_all_wakeup	138
ds2202_can_msg_wakeup	179
ds2202_canMsg	123

ds2202_can_msg_wakeup

Syntax `Int32 ds2202_can_msg_wakeup(
 const ds2202_canMsg* canMsg);`

Include file `Can2202.h`

Purpose To reactivate a message that has been deactivated by calling the function `ds2202_can_msg_sleep` or `ds2202_can_channel_all_sleep`.

Parameters **canMsg** Specifies the pointer to the `ds2202_canMsg` structure. (See [ds2202_canMsg](#) on page 123.)

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_TYPE_ERROR	The operation is not allowed for the given message object.

Function execution times For information, refer to [Function Execution Times](#) on page 207.

Example

```
ds2202_can_msg_wakeup(txMsg);
```

Related topics**References**

ds2202_can_channel_all_sleep.....	137
ds2202_can_channel_all_wakeup.....	138
ds2202_can_msg_sleep.....	178
ds2202_canMsg.....	123

ds2202_can_msg_read

Syntax

```
Int32 ds2202_can_msg_read(  
    ds2202_canMsg* canMsg);
```

Include file

Can2202.h

Purpose

To read the data length, the data and the status information from the dual-port memory.

Description

The return value provides information on whether or not the data is new. If not, the existing parameter values remain unchanged.

You can call this function several times for one message object to read all the messages available in the message buffer (see also **ds2202_can_msg_set**). By default, only one message can be received.

Use the function **ds2202_can_msg_clear** to clear the message data and timestamps. This is useful for simulation start/stop transitions.

Note

The status information being returned depends on the previously specified parameter inform in the register function corresponding to the message.

Parameters

canMsg Specifies the pointer to the updated **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.) The following data will be updated if available:

Parameter	Meaning
data	Buffer with the updated data
datalen	Data length of the message
deltatime	Deltatime of the message
timestamp	Timestamp of the message
delaytime	Delaytime of the message
processed	Processed flag of the message
identifier	identifier of the message
format	format of the identifier

Return value

This function returns the error code; the following symbols are predefined:

Symbols	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_NO_DATA	No data was updated.
DS2202_CAN_DATA_LOST	The input data of a previous request for the specified function has been overwritten.

Function execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics**Examples**

Example of Handling Request and Remote Messages.....	200
Example of Handling Transmit and Receive Messages.....	198
Example of Using Subinterrupts.....	202

References

ds2202_can_msg_clear.....	183
ds2202_can_msg_set.....	164
ds2202_canMsg.....	123

ds2202_can_msg_trigger

Syntax

```
Int32 ds2202_can_msg_trigger(
    const ds2202_canMsg* canMsg,
    const Float32 delay);
```

Include file

Can2202.h

Purpose

To send a transmit or request message immediately after the specified delay time.

Description

This function can be used for acyclic message sending. Use the returned handle from the function **ds2202_can_msg_tx_register** or **ds2202_can_msg_rqt_tx_register** to call this function.

Parameters

canMsg Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

delay Send the message after the delaytime. Valid range is 0.0 ... 100.0 seconds.

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_TYPE_ERROR	The operation is not allowed for the given message object.

Function execution times

For information, refer to [Function Execution Times](#) on page 207.

Example

```
ds2202_can_msg_trigger(txMsg, 0.005); /* 5 ms delay */
```

Related topics

References

ds2202_can_msg_rqt_tx_register	154
ds2202_can_msg_tx_register	146
ds2202_canMsg	123

ds2202_can_msg_clear

Syntax `void ds2202_can_msg_clear(
ds2202_canMsg* canMsg);`

Include file Can2202.h

Purpose To clear the following message data: data[8], datalen, timestamp, deltatime, timecount, delaytime and processed.

Description This is useful for simulation start/stop transitions.
Use the returned handle from the message register functions to call this function.

Note

The structure members identifier, format, module, queue, index, msg_no, type, inform, canChannel and msgService are untouched, since any manipulation of these structure members will corrupt the message object.

Parameters **canMsg** Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Return value None

Function execution times For information, refer to [Function Execution Times](#) on page 207.

Example `ds2202_can_msg_clear(rxMsg);`

Related topics	References
	ds2202_can_all_data_clear 196
	ds2202_can_msg_rm_register 160
	ds2202_can_msg_rqr_register 157
	ds2202_can_msg_rqt_register 154
	ds2202_can_msg_rx_register 150
	ds2202_can_msg_tx_register 146
	ds2202_canMsg 123

ds2202_can_msg_processed_register

Syntax

```
void ds2202_can_msg_processed_register(
    ds2202_canMsg* canMsg);
```

Include file

Can2202.h

Purpose

To register the processed function in the command table. Use **ds2202_can_msg_processed_read** to read the processed flag and timestamp without registering the message with the inform parameter DS2202_CAN_TIMECOUNT_INFO.

Parameters

canMsg Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Return value

None.

Messages

The following error and warning messages are defined:

ID	Type	Description	Message
101	Error	ds2202_can_msg_processed_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2202_can_msg_processed_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2202_can_msg_processed_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2202_CAN_AUTO_INDEX.
104	Error	ds2202_can_msg_processed_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	ds2202_can_msg_processed_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2202_can_msg_processed_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2202_can_msg_processed_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with ds2202_can_msg_sleep or ds2202_can_channel_all_sleep when registering messages or services.

Function execution times For information, refer to [Function Execution Times](#) on page 207.

Example `ds2202_can_msg_processed_register(rxMsg);`

Related topics

References

ds2202_can_channel_all_sleep	137
ds2202_can_channel_init	130
ds2202_can_channel_init_advanced	133
ds2202_can_msg_processed_read	186
ds2202_can_msg_processed_request	185
ds2202_can_msg_sleep	178
ds2202_canMsg	123

ds2202_can_msg_processed_request

Syntax `Int32 ds2202_can_msg_processed_request(
const ds2202_canMsg* canMsg);`

Include file `Can2202.h`

Purpose To request the message processed information from the slave DS2202.

Parameters **canMsg** Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_NO_DATA	<code>ds2202_can_msg_processed_request</code> was called without registering the function with <code>ds2202_can_msg_processed_register</code> , or an empty <code>canMsg</code> structure was handled.

Function execution times For information, refer to [Function Execution Times](#) on page 207.

Example `ds2202_can_msg_processed_request(rxMsg);`

Related topics

References

[ds2202_can_msg_processed_read](#)..... 186
[ds2202_can_msg_processed_register](#)..... 184
[ds2202_canMsg](#)..... 123

ds2202_can_msg_processed_read

Syntax

```
Int32 ds2202_can_msg_processed_read(
    ds2202_canMsg* canMsg,
    double* timestamp,
    UInt32* processed);
```

Include file

Can2202.h

Purpose

To read the message processed information from the slave DS2202.

Description

Prior to this, this information must have been requested by the master calling the function **ds2202_can_msg_processed_request** that demands the processed flag and the time stamp from the slave DS2202.

Parameters

canMsg Specifies the pointer to the **ds2202_canMsg** structure. (See [ds2202_canMsg](#) on page 123.)

timestamp Specifies the timestamp when the message was last sent or received.

processed Specifies the processed flag of the message. The following symbols are predefined:

Symbols	Meaning
DS2202_CAN_PROCESSED	Message has been sent/received since the last execution call.
DS2202_CAN_NOT_PROCESSED	Message has not been sent/received since the last execution call.

Return value

This function returns the error code; the following symbols are predefined:

Symbols	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_NO_DATA	No data was updated.
DS2202_CAN_DATA_LOST	The input data of a previous request for the specified function has been overwritten.

Function execution times

For information, refer to [Function Execution Times](#) on page 207.

Related topics

References

ds2202_can_msg_processed_register	184
ds2202_can_msg_processed_request	185
ds2202_canMsg	123

CAN Service Functions

Introduction Information on errors and status information.

Where to go from here Information in this section

ds2202_can_service_register	188
To register the service read function in the command table.	
ds2202_can_service_request	190
To request the service information from the slave.	
ds2202_can_service_read	191
To read the service information from the slave.	

ds2202_can_service_register

Syntax

```
ds2202_canService* ds2202_can_service_register(
    const ds2202_canChannel* canCh,
    const UInt32 service_type);
```

Include file Can2202.h

Purpose To register the service function.

Description Use **ds2202_can_service_read** to read a registered service specified by the **service_type** parameter. With the **ds2202_can_service_read** function you can read a registered service specified by the **service_type** parameter.

Parameters

canCh Specifies the pointer to the **ds2202_canChannel** structure. (See [ds2202_canChannel](#) on page 118.)

service_type Specifies the service to be installed. For additional information, see **ds2202_canService**, parameter **type**. You can use the logical OR to combine several services.

Return value **canService** This function returns the pointer to the **ds2202_canService** structure. (See [ds2202_canService](#) on page 119.)

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2202_can_service_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2202_can_service_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2202_can_service_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2202_CAN_AUTO_INDEX.
104	Error	ds2202_can_service_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	ds2202_can_service_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2202_can_service_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2202_can_service_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data in a filled queue. To prevent this error deactivate all messages with ds2202_can_msg_sleep or ds2202_can_channel_all_sleep when registering messages or services.
152	Error	ds2202_can_service_register(x,..) canCh: the CAN channel wasn't initialized	This message will be displayed if: <ul style="list-style-type: none"> ▪ you try to register a CAN message on an uninitialized CAN channel. ▪ you try to register a CAN service on an uninitialized CAN channel. ▪ you try to start an uninitialized CAN channel with ds2202_can_channel_start. Use ds2202_can_channel_init or ds2202_can_channel_init_advanced to initialize the CAN channel.

Example

```
ds2202_canService* service;
...
service = ds2202_can_service_register(txCh,
    DS2202_CAN_SERVICE_TX_OK |
    DS2202_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT );
```

Related topics

References

ds2202_can_channel_all_sleep	137
ds2202_can_channel_init	130
ds2202_can_channel_init_advanced	133
ds2202_can_channel_start	136
ds2202_can_msg_sleep	178
ds2202_can_service_read	191
ds2202_can_service_request	190
ds2202_canChannel	118
ds2202_canService	119

ds2202_can_service_request

Syntax

```
Int32 ds2202_can_service_request(
    ds2202_canService* service);
```

Include file

Can2202.h

Purpose

To request the service information from the slave DS2202. Use **ds2202_can_service_read** to read the registered service.

Description

Use the returned handle from the function **ds2202_can_service_register** to call this function.

Parameters

service Specifies the pointer to the **ds2202_canService** structure. (See [ds2202_canService](#) on page 119.)

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns DS2202_CAN_NO_ERROR.
DS2202_CAN_NO_DATA	ds2202_can_service_request was called without registering the function with ds2202_can_service_register , or an empty service structure was handled.

Function execution times	For information, refer to Function Execution Times on page 207.
Example	For an example of how to use this function, refer to Example of Using Service Functions on page 204.
Related topics	<div>Examples</div> <div>Example of Using Service Functions..... 204</div> <div>References</div> <div>ds2202_can_service_read..... 191 ds2202_can_service_register..... 188 ds2202_canService..... 119</div>

ds2202_can_service_read

Syntax	<pre>Int32 ds2202_can_service_read(ds2202_canService* service);</pre>
Include file	Can2202.h
Purpose	To read the service information from the slave DS2202.
Description	<p>Prior to this, this information must have been requested by the master calling the function ds2202_can_service_request that asks for the service information from the slave DS2202.</p> <p>Use the returned handle from the function ds2202_can_service_register.</p>
Parameters	<p>service Specifies the pointer to the updated ds2202_canService structure. (See ds2202_canService on page 119.)</p> <p>The following data will be updated if available: busstatus, stdmask, extmask, msg_mask15, tx_ok, rx_ok, crc_err, ack_err, form_err, stuffbit_err, bit1_err, bit0_err, rx_lost, data_lost, version, mailbox_err, txqueue_overflowcnt_std, txqueue_overflowcnt_ext.</p>

Return value

This function returns the error code; the following symbols are predefined:

Symbols	Meaning
DS2202_CAN_NO_ERROR	The function has been performed without error.
DS2202_CAN_NO_DATA	No data was updated.
DS2202_CAN_DATA_LOS	The input data of a previous request for the specified function has been overwritten.

Function execution times

For information, refer to [Function Execution Times](#) on page 207.

Examples

For an example of how to use this function, refer to [Example of Using Service Functions](#) on page 204.

```
ds2202_canService* service;
...
service = ds2202_can_service_register(txCh,
    DS2202_CAN_SERVICE_TX_OK |
    DS2202_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT);
ds2202_can_service_request(service);
ds2202_can_service_read(service);
/* output */
txok = service->tx_ok;
queueoverflow = service->txqueue_overflowcnt_std;
```

Related topics**References**

ds2202_can_service_register	188
ds2202_can_service_request	190
ds2202_canService	119

CAN Subinterrupt Handling

Where to go from here	Information in this section
	Defining a Callback Function..... 193
	You can define actions to be performed when a specific subinterrupt occurs. You have to define these actions in a <i>callback function</i> .
	ds2202_can_subint_handler_install..... 194
	To install a subinterrupt handler.

Defining a Callback Function

Actions to be performed when a subinterrupt occurs	<p>You can define actions to be performed when a specific subinterrupt occurs. You have to define these actions in a <i>callback function</i>.</p> <p>Each time a CAN subinterrupt occurs, the subinterrupt handling then passes the information to the callback function.</p>
Installing a callback function	<p>You have to install the callback function with the <code>ds2202_can_subint_handler_install</code> function.</p>
Defining a callback function	<p>Define your callback function as follows:</p> <pre>void can_callback_fcn(void* subint_data, Int32 subint);</pre> <p>with the following parameters:</p> <p>subint_data Pointer to the board index of the related board within the range of 0 ... 15</p> <p>subint Subinterrupt number within the range of 0 ... 14</p> <div><p>Note</p><p>The last subinterrupt number to be generated is always "-1". This value indicates that there is no further subinterrupt pending.</p></div>

Related topics**Examples**[Example of Using Subinterrupts.....](#) 202**References**[ds2202_can_subint_handler_install.....](#) 194

ds2202_can_subint_handler_install

Syntax

```
ds2202_can_subint_handler_t ds2202_can_subint_handler_install(  
    const UInt32 base,  
    const ds2202_can_subint_handler_t handler);
```

Include file

Can2202.h

Purpose

To install a subinterrupt handler for all CAN interrupts.

Parameters**base** Specifies the PHS-bus base address of the DS2202 board.**handler** Specifies the pointer to your callback function.For information on defining a callback function, refer to [Defining a Callback Function](#) on page 193.**Return value**

This function returns the following value:

Symbol	Meaning
ds2202_can_subint_handler_t	Pointer to the previously installed callback function

ExampleFor an example of how to use this function, refer to [Example of Using Subinterrupts](#) on page 202.

Related topics

Basics

Defining a Callback Function.....	193
-----------------------------------	-----

Examples

Example of Using Subinterrupts.....	202
-------------------------------------	-----

Utilities

Introduction Information on setting the time base to a defined value, clearing CAN data on the master, and reading the current error code.

Where to go from here

Information in this section

[ds2202_can_all_data_clear..... 196](#)
To clear CAN data on the master.

[ds2202_can_error_read..... 197](#)
To read the error code.

ds2202_can_all_data_clear

Syntax

```
void ds2202_can_all_data_clear(const UInt32 base);
```

Include file

Can2202.h

Purpose

To clear the data buffer of the master. This is required by the RTI environment to clear all data when restarting the simulation.

Parameters

base Specifies the PHS-bus base address of the DS2202 board.

Return value

None

Example

```
ds2202_can_all_data_clear(DS2202_1_BASE);
```

Related topics

References

[ds2202_can_msg_clear..... 183](#)

ds2202_can_error_read

Syntax

```
Int32 ds2202_can_error_read(
    const UInt32 base,
    const Int32 queue);
```

Include file

Can2202.h

Purpose

To read the current error of the slave DS2202 from the dual-port memory.

Parameters

base Specifies the PHS-bus base address of the DS2202 board.
queue Specifies the communication channel within the range 0 ... 6.

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
DS2202_CAN_NO_ERROR	No error on the slave DS2202
DS2202_CAN_SLAVE_ALLOC_ERROR	Memory allocation error on the slave DS2202. There are too many functions registered.
DS2202_CAN_SLAVE_BUFFER_OVERFLOW	Not enough memory space between the slave write pointer and the master read pointer.
DS2202_CAN_INIT_ACK	Acknowledge code. This is no error.
DS2202_CAN_SLAVE_UNDEF_ERROR	Undefined error, an error that can not be assigned to one of the previous errors.

Example

```
#define QUEUE0 0
Int32 slave_error;
slave_error = ds2202_can_error_read(DS2202_1_BASE, QUEUE0);
/* */
/* error handling */
/* */
```

Examples of Using CAN

Introduction

Examples showing how to use the CAN functions.

Where to go from here

Information in this section

[Example of Handling Transmit and Receive Messages..... 198](#)

The example shows how to register a transmit and a receive message.

[Example of Handling Request and Remote Messages..... 200](#)

The example shows how to register a request and a remote message.

[Example of Using Subinterrupts..... 202](#)

The example shows how to register messages that can generate a subinterrupt.

[Example of Using Service Functions..... 204](#)

The example shows how to use service functions.

[Example of Receiving Different Message IDs..... 205](#)

The example shows how to set up a CAN controller to receive different message IDs.

Example of Handling Transmit and Receive Messages

Introduction

This example shows how to register a transmit and a receive message. After a delay of 4.0 seconds the transmit message is sent periodically every 1.0 second. If you connect the two CAN channels with each other you can receive the transmitted CAN message on the other CAN channel. After the CAN message is received successfully an info message is reported to the message module.

Example

```

1  #include <Brtenv.h>
2  #include <Ds2202.h>
3  #include <Can2202.h>
4  ds2202_canChannel* txCh;
5  ds2202_canChannel* rxCh;
6  ds2202_canMsg* txMsg;
7  ds2202_canMsg* rxMsg;
8  UInt32 txMsgData[8] = {1,2,3,4,5,6,7,8};
9  main()
10 {
11     init(); /* initialize hardware system */

```

```

12 ds2202_init(DS2202_1_BASE);
13 ds2202_can_communication_init(DS2202_1_BASE,
14     DS2202_CAN_INT_DISABLE);
15 txCh = ds2202_can_channel_init(DS2202_1_BASE, 0,
16     500000,
17     DS2202_CAN_STD,
18     DS2202_CAN_NO_SUBINT,
19     DS2202_CAN_TERMINATION_ON);
20 rxCh = ds2202_can_channel_init(DS2202_1_BASE, 1,
21     500000,
22     DS2202_CAN_STD,
23     DS2202_CAN_NO_SUBINT,
24     DS2202_CAN_TERMINATION_ON);
25 txMsg = ds2202_can_msg_tx_register(txCh,
26     2,
27     0x123,
28     DS2202_CAN_STD,
29     DS2202_CAN_TIMECOUNT_INFO,
30     DS2202_CAN_NO_SUBINT,
31     4.0,
32     1.0,
33     DS2202_CAN_TIMEOUT_NORMAL);
34 rxMsg = ds2202_can_msg_rx_register(rxCh,
35     3,
36     0x123,
37     DS2202_CAN_STD,
38     DS2202_CAN_DATA_INFO | DS2202_CAN_TIMECOUNT_INFO,
39     DS2202_CAN_NO_SUBINT);
40 ds2202_can_msg_write(txMsg, 8, txMsgData);
41 ds2202_can_channel_start(rxCh, DS2202_CAN_INT_DISABLE);
42 ds2202_can_channel_start(txCh, DS2202_CAN_INT_DISABLE);
43 for(;;)
44 {
45     ds2202_can_msg_read(txMsg);
46     if (txMsg->processed == DS2202_CAN_PROCESSED)
47     {
48         msg_info_printf(MSG_SM_RTLIB, 0,
49             "TX CAN message, time: %f, deltatime: %f ",
50             txMsg->timestamp, txMsg->deltatime);
51     }
52     ds2202_can_msg_read(rxMsg);
53     if (rxMsg->processed == DS2202_CAN_PROCESSED)
54     {
55         msg_info_printf(MSG_SM_RTLIB, 0,
56             "RX CAN message, time: %f,deltatime: %f ",
57             rxMsg->timestamp, rxMsg->deltatime);
58     }
59     RTLIB_BACKGROUND_SERVICE();
60 }
61 }

```

Related topics

Examples

Example of Handling Request and Remote Messages.....	200
Example of Receiving Different Message IDs.....	205
Example of Using Service Functions.....	204
Example of Using Subinterrupts.....	202

Example of Handling Request and Remote Messages

Introduction

This example shows how to register a request and a remote message. After a delay of 4.0 seconds the request message is sent periodically every 2.0 second. If you connect the two CAN channels with each other you can receive the request message on the other CAN channel. After the requested data is received successfully an info message is reported to the message module.

Example

```

1  #include <Brtenv.h>
2  #include <Ds2202.h>
3  #include <Can2202.h>
4  ds2202_canChannel* rqCh;
5  ds2202_canChannel* rmCh;
6  ds2202_canMsg* rqtMsg;
7  ds2202_canMsg* rqrMsg;
8  ds2202_canMsg* rmMsg;
9  UInt32 rmMsgData[8] = {1,2,3,4,5,6,7,8};
10 main()
11 {
12     init(); /* initialize hardware system */
13     ds2202_init(DS2202_1_BASE);
14     ds2202_can_communication_init(DS2202_1_BASE,
15                                   DS2202_CAN_INT_DISABLE);
16     rqCh = ds2202_can_channel_init(DS2202_1_BASE,
17                                    0,
18                                    500000,
19                                    DS2202_CAN_STD,
20                                    DS2202_CAN_NO_SUBINT,
21                                    DS2202_CAN_TERMINATION_ON);
22     rmCh = ds2202_can_channel_init(DS2202_1_BASE,
23                                    1,
24                                    500000,
25                                    DS2202_CAN_STD,
26                                    DS2202_CAN_NO_SUBINT,
27                                    DS2202_CAN_TERMINATION_ON);
28     rqtMsg = ds2202_can_msg_rqt_register(rqCh,
29                                           2,
30                                           0x123,

```



```

31         DS2202_CAN_STD,
32         DS2202_CAN_TIMECOUNT_INFO,
33         DS2202_CAN_NO_SUBINT,
34         4.0,
35         2.0,
36         DS2202_CAN_TIMEOUT_NORMAL);
37 rqrMsg = ds2202_can_msg_rqr_register(rqtMsg,
38         DS2202_CAN_DATA_INFO | DS2202_CAN_TIMECOUNT_INFO,
39         DS2202_CAN_NO_SUBINT);
40 rMMsg = ds2202_can_msg_rm_register(rMCh,
41         3,
42         0x123,
43         DS2202_CAN_STD,
44         DS2202_CAN_TIMECOUNT_INFO,
45         DS2202_CAN_NO_SUBINT);
46 ds2202_can_msg_write(rMMsg, 8, rMMsgData);
47 ds2202_can_msg_rqt_activate(rqtMsg);
48 ds2202_can_channel_start(rqCh, DS2202_CAN_INT_DISABLE);
49 ds2202_can_channel_start(rMCh, DS2202_CAN_INT_DISABLE);
50 for(;;)
51 {
52     ds2202_can_msg_read(rqrMsg);
53     ds2202_can_msg_read(rqtMsg);
54     ds2202_can_msg_read(rMMsg);
55     if (rqrMsg->processed == DS2202_CAN_PROCESSED)
56     {
57         msg_info_printf(MSG_SM_RTLIB, 0,
58             "RQRX CAN message, time: %f,deltatime: %f ",
59             rqrMsg->timestamp, rqrMsg->deltatime);
60     }
61     if (rqtMsg->processed == DS2202_CAN_PROCESSED)
62     {
63         msg_info_printf(MSG_SM_RTLIB, 0,
64             "RQTX CAN message, time: %f, deltatime: %f ",
65             rqtMsg->timestamp, rqtMsg->deltatime);
66     }
67     if (rMMsg->processed == DS2202_CAN_PROCESSED)
68     {
69         msg_info_printf(MSG_SM_RTLIB, 0,
70             "RM CAN message, time: %f, deltatime: %f ",
71             rMMsg->timestamp, rMMsg->deltatime);
72     }
73     RTLIB_BACKGROUND_SERVICE();
74 }
75 }

```

Related topics

Examples

Example of Handling Transmit and Receive Messages.....	198
Example of Receiving Different Message IDs.....	205
Example of Using Service Functions.....	204
Example of Using Subinterrupts.....	202

Example of Using Subinterrupts

Introduction

This example shows how to register messages that can generate a subinterrupt. The CAN controller will be started and a CAN message will be sent immediately. If the CAN message was sent successfully, a subinterrupt is generated to call the installed callback function. The callback function in this example evaluates the given subinterrupt and sends the CAN message again with a time delay of 0.1 s. After the CAN message is received another subinterrupt is generated to read the CAN message and pass an info message to the message module.

Note

The CAN channels 0 and 1 have to be connected.

Example

```

1  #include <Brtenv.h>
2  #include <Ds2202.h>
3  #include <Can2202.h>
4  #define tx_subint 2
5  #define rx_subint 3
6  ds2202_canChannel* txCh;
7  ds2202_canChannel* rxCh;
8  ds2202_canMsg* txMsg;
9  ds2202_canMsg* rxMsg;
10 UInt32 txMsgData[8] = { 1,2,3,4,5,6,7,8 };
11 void can_user_callback(void* subint_data, Int32 subint)
12 {
13     switch(subint)
14     {
15         case tx_subint:
16             txMsgData[0] = (txMsgData[0]+1) & 0xFF;
17             /* send the message delayed */
18             ds2202_can_msg_send( txMsg, 8, txMsgData, 0.1);
19             msg_info_printf(MSG_SM_RTLIB,
20                 0,
21                 "TX Subint:%d", subint);
22             break;
23         case rx_subint:
24             /* read the message from the communication buffer */
25             ds2202_can_msg_read(rxMsg);
26             msg_info_printf(MSG_SM_RTLIB,
27                 0,
28                 "RX Subint:%d, time:%fs, deltatime:%fs data[0]:%x",
29                 subint,
30                 rxMsg->timestamp,
31                 rxMsg->deltatime,
32                 rxMsg->data[0]);
33             break;
34         default:

```

```

35     break;
36 }
37 }
38 main()
39 {
40     init(); /* initialize hardware system */
41     ds2202_init(DS2202_1_BASE);
42     ds2202_can_communication_init(DS2202_1_BASE,
43                                   DS2202_CAN_INT_ENABLE);
44     ds2202_can_subint_handler_install(DS2202_1_BASE,
45                                       can_user_callback);
46     txCh = ds2202_can_channel_init (DS2202_1_BASE,
47                                     1,
48                                     500000,
49                                     DS2202_CAN_STD,
50                                     DS2202_CAN_NO_SUBINT,
51                                     DS2202_CAN_TERMINATION_ON);
52     txMsg = ds2202_can_msg_tx_register(txCh,
53                                       0,
54                                       0x123,
55                                       DS2202_CAN_STD,
56                                       DS2202_CAN_NO_INFO,
57                                       tx_subint,
58                                       0.0,
59                                       0.0,
60                                       DS2202_CAN_TIMEOUT_NORMAL);
61     rxCh = ds2202_can_channel_init(DS2202_1_BASE,
62                                    0,
63                                    500000,
64                                    DS2202_CAN_STD,
65                                    DS2202_CAN_NO_SUBINT,
66                                    DS2202_CAN_TERMINATION_ON);
67     rxMsg = ds2202_can_msg_rx_register(rxCh,
68                                       0,
69                                       0x123,
70                                       DS2202_CAN_STD,
71                                       DS2202_CAN_DATA_INFO |
72                                       DS2202_CAN_TIMECOUNT_INFO,
73                                       rx_subint);
74     ds2202_can_channel_start(rxCh, DS2202_CAN_INT_DISABLE);
75     ds2202_can_channel_start(txCh, DS2202_CAN_INT_DISABLE);
76     ds2202_can_msg_send( txMsg, 8, txMsgData, 0.0);
77     RTLIB_INT_ENABLE();
78     for(;;)
79     {
80         RTLIB_BACKGROUND_SERVICE();
81     }
82 }

```

Related topics**Basics**

Defining a Callback Function..... 193

Examples

Example of Handling Request and Remote Messages..... 200
 Example of Handling Transmit and Receive Messages..... 198
 Example of Receiving Different Message IDs..... 205
 Example of Using Service Functions..... 204

Example of Using Service Functions

Introduction

This example shows how to use the service functions DS2202_CAN_SERVICE_TX_OK and DS2202_CAN_SERVICE_RX_OK. Note that no message is installed on the slave DS2202 in this example.

Example

```

1  #include <Brtenv.h>
2  #include <Ds2202.h>
3  #include <Can2202.h>
4  ds2202_canChannel* canCh0;
5  ds2202_canChannel* canCh1;
6  ds2202_canService* txokServ;
7  ds2202_canService* rxokServ;
8  main()
9  {
10     init();
11     ds2202_init(DS2202_1_BASE);
12     ds2202_can_communication_init(DS2202_1_BASE,
13                                   DS2202_CAN_INT_DISABLE);
14     canCh0 = ds2202_can_channel_init(DS2202_1_BASE, 0,
15                                     500000, DS2202_CAN_STD, DS2202_CAN_NO_SUBINT,
16                                     DS2202_CAN_TERMINATION_ON);
17     canCh1 = ds2202_can_channel_init(DS2202_1_BASE, 1,
18                                     500000, DS2202_CAN_STD, DS2202_CAN_NO_SUBINT,
19                                     DS2202_CAN_TERMINATION_ON);
20     /* register the tx_ok function which delivers the count */
21     /* of the tx-ok counter for CAN-Channel 0 */
22     txokServ = ds2202_can_service_register(canCh0,
23                                           DS2202_CAN_SERVICE_TX_OK);
24     /* register the rx_ok function which delivers the count */
25     /* of the rx-ok counter for CAN-Channel 1 */
26     rxokServ = ds2202_can_service_register(canCh1,
27                                           DS2202_CAN_SERVICE_RX_OK);
28     for(;;)
  
```

```

29 {
30     /* request the tx-ok counter from the slave DS2202 */
31     ds2202_can_service_request(txokServ);
32     /* request the rx-ok counter from the slave DS2202 */
33     ds2202_can_service_request(rxokServ);
34     /* read the tx-ok counter from the slave DS2202 */
35     /* the data will be available in txokServ->data0 */
36     ds2202_can_service_read(txokServ);
37     /* read the rx-ok counter from the slave DS2202 */
38     /* the data will be available in rxokServ->data0 */
39     ds2202_can_service_read(rxokServ);
40     RTLIB_BACKGROUND_SERVICE();
41 }
42 }

```

Related topics

Examples

Example of Handling Request and Remote Messages.....	200
Example of Handling Transmit and Receive Messages.....	198
Example of Receiving Different Message IDs.....	205
Example of Using Subinterrupts.....	202

Example of Receiving Different Message IDs

Introduction

This example shows how to set up a CAN controller to receive the message IDs 0x100 ... 0x1FF via one message queue.

Example

```

1  #include <Brtenv.h>
2  #include <Ds2202.h>
3  #include <Can2202.h>
4  ds2202_canChannel* rxCh;
5  ds2202_canMsg* canMonitor;
6  UInt32 data[8];
7  UInt32 mask = 0x1FFFFF00;
8  UInt32 queue_size = 64;
9  main()
10 {
11     init();
12     ds2202_init(DS2202_1_BASE);
13     ds2202_can_communication_init(DS2202_1_BASE,
14                                   DS2202_CAN_INT_DISABLE);
15     rxCh = ds2202_can_channel_init(DS2202_1_BASE,
16                                    0,
17                                    500000,
18                                    DS2202_CAN_STD,

```

```

19         DS2202_CAN_NO_SUBINT,
20         DS2202_CAN_TERMINATION_ON);
21     canMonitor = ds2202_can_msg_rx_register (rxCh,
22         1,
23         0x100,
24         DS2202_CAN_STD,
25         DS2202_CAN_TIMECOUNT_INFO |
26         DS2202_CAN_MSG_INFO,
27         DS2202_CAN_NO_SUBINT);
28     ds2202_can_msg_set(canMonitor,
29         DS2202_CAN_MSG_MASK,
30         &mask);
31     ds2202_can_msg_set(canMonitor,
32         DS2202_CAN_MSG_QUEUE,
33         &queue_size);
34     ds2202_can_channel_start(rxCh, DS2202_CAN_INT_DISABLE);
35     for(;;)
36     {
37         ds2202_can_msg_read(canMonitor);
38         if (canMonitor->processed == DS2202_CAN_PROCESSED)
39         {
40             msg_info_printf(0,0,"id: %d time: %f",
41                 canMonitor->identifier, canMonitor->timestamp);
42         }
43         RTLIB_BACKGROUND_SERVICE();
44     }
45 }

```

Related topics

Examples

Example of Handling Request and Remote Messages.....	200
Example of Handling Transmit and Receive Messages.....	198
Example of Using Service Functions.....	204
Example of Using Subinterrupts.....	202

Function Execution Times

Introduction	The execution times of the C functions can vary, since they depend on different factors. This section gives you information on the test environment and contains the mean function execution times.
--------------	---

Where to go from here	<div>Information in this section</div> <div><div>Information on the Test Environment.....207</div><div>Information on the measurement test environment of the execution times.</div><div>Measured Execution Times.....208</div><div>Function execution times are measured for the RTLib units.</div></div>
-----------------------	--

Information on the Test Environment

Test environment	<p>The execution time of a function can vary, since it depends on different factors, for example:</p> <ul style="list-style-type: none">▪ CPU clock and bus clock frequency of the processor board used▪ Optimization level of the compiler▪ Use of inlining parameters <p>The test programs that are used to measure the execution time of the functions listed below have been generated and compiled with the default settings of the <code>down<xxxx></code> tool (optimization and inlining). The execution times in the tables below are always the mean measurement values.</p>
------------------	--

The properties of the processor boards used are:

	DS1006	DS1006 Multicore
CPU clock	2.6 GHz / 3.0 GHz	2.8 GHz
Bus clock	133 MHz	133 MHz

Measured Execution Times

Introduction

Function execution times are measured for the RTLib units.

Note

The following execution times contain mean values for a sequence of I/O accesses. The execution time of a single call might be lower because of buffered I/O access.

Initialization and setup

The following execution times were measured for initialization and setup functions:

Function	Execution Time	
	DS1006	DS1006 Multicore
ds2202_digin_threshold_set	0.82 μ s	0.80 μ s
ds2202_digout_mode_set	0.72 μ s	0.67 μ s
ds2202_digout_hs_vbat1_clear	0.72 μ s	0.67 μ s
ds2202_digout_hs_vbat1_set	0.72 μ s	0.67 μ s
ds2202_digout_hs_vbat1_write	0.04 μ s	0.06 μ s
ds2202_digout_hs_vbat2_clear	0.72 μ s	0.67 μ s
ds2202_digout_hs_vbat2_set	0.72 μ s	0.67 μ s
ds2202_digout_hs_vbat2_write	0.04 μ s	0.06 μ s
ds2202_digout_ls_write	0.04 μ s	0.06 μ s
ds2202_digout_ls_set	0.72 μ s	0.67 μ s
ds2202_digout_ls_clear	0.73 μ s	0.67 μ s

ADC

The following execution times were measured for ADC functions:

Function	Execution Time	
	DS1006	DS1006 Multicore
ds2202_adc_start	0.03 μ s	0.06 μ s
ds2202_adc_block_in_fast	$(0.738 + n \cdot 1.144) \mu\text{s}^1)$	$(0.816 + n \cdot 1.136) \mu\text{s}^1)$

Function	Execution Time	
	DS1006	DS1006 Multicore
ds2202_adc_single_in (channel 16)	1.38 μ s	1.37 μ s
ds2202_adc_block_init (16 channels)	0.23 μ s	0.22 μ s
ds2202_adc_block_start (16 channels)	0.03 μ s	0.05 μ s
ds2202_adc_block_in	$(0.696 + n \cdot 0.595) \mu$ s ¹⁾	$(0.777 + n \cdot 0.589) \mu$ s ¹⁾

¹⁾ n: number of channels

DAC

The following execution times were measured for DAC functions:

Function	Execution Time	
	DS1006	DS1006 Multicore
ds2202_dac_out	0.08 μ s	0.16 μ s

Bit I/O

The following execution times were measured for Bit I/O functions:

Function	Execution Time	
	DS1006	DS1006 Multicore
ds2202_bit_io_in	0.74 μ s	0.74 μ s
ds2202_bit_io_out	0.03 μ s	0.05 μ s
ds2202_bit_io_set	0.72 μ s	0.67 μ s
ds2202_bit_io_clear	0.72 μ s	0.67 μ s

Timing mode

The following execution times were measured for timing mode functions:

Function	Execution Time	
	DS1006	DS1006 Multicore
ds2202_timing_in_mode_set	0.75 μ s	0.76 μ s
ds2202_timing_out_mode_set	0.75 μ s	0.76 μ s

PWM signal measurement and generation

The following execution times were measured for PWM signal measurement and generation functions:

Function	Execution Time	
	DS1006	DS1006 Multicore
ds2202_pwm_in	0.80 μ s	0.79 μ s
ds2202_pwm_out	0.12 μ s	0.18 μ s

Square-wave signal measurement and generation

The following execution times were measured for square-wave signal measurement and generation functions:

Function	Execution Time	
	DS1006	DS1006 Multicore
ds2202_f2d	0.75 μ s	0.74 μ s
ds2202_d2f	0.09 μ s	0.10 μ s

Related topics**References**

[Information on the Test Environment.....](#) 207

A

ADC unit 29

C

CAN

- access functions 113
- basic communication principles 116
- communication channel 116
- initialization functions 116
- read functions 116
- register functions 116
- request functions 116
- slave access functions 116
- write functions 116

Common Program Data folder 8

D

data type

- ds2202_ISR 81
- ds2202_LSR 83
- ds2202_MSR 84
- ds2202_subint_handler_t 85
- ds2202_Channel 87

Documents folder 8

DS2202

- real-time library 7
- ds2202_adc_block_in 40
- ds2202_adc_block_in_fast 35
- ds2202_adc_block_init 38
- ds2202_adc_block_start 39
- ds2202_adc_single_in 36
- ds2202_adc_start 34
- ds2202_bit_io_clear 51
- ds2202_bit_io_in 47
- ds2202_bit_io_out 49
- ds2202_bit_io_set 50
- ds2202_can_all_data_clear 196
- ds2202_can_channel_all_sleep 137
- ds2202_can_channel_all_wakeup 138
- ds2202_can_channel_BOff_go 139
- ds2202_can_channel_BOff_return 140
- ds2202_can_channel_init 130
- ds2202_can_channel_init_advanced 133
- ds2202_can_channel_set 141
- ds2202_can_channel_start 136
- ds2202_can_channel_txqueue_clear 143
- ds2202_can_communication_init 128
- ds2202_can_error_read 197
- ds2202_can_msg_clear 183
- ds2202_can_msg_processed_read 186
- ds2202_can_msg_processed_register 184
- ds2202_can_msg_processed_request 185
- ds2202_can_msg_queue_level 172
- ds2202_can_msg_read 180
- ds2202_can_msg_rm_register 160
- ds2202_can_msg_rqrq_register 157
- ds2202_can_msg_rqrq_activate 167
- ds2202_can_msg_rqrq_register 154
- ds2202_can_msg_rx_register 150

- ds2202_can_msg_send 169
- ds2202_can_msg_send_id 171
- ds2202_can_msg_send_id_queued 175
- ds2202_can_msg_set 164
- ds2202_can_msg_sleep 178
- ds2202_can_msg_trigger 182
- ds2202_can_msg_tx_register 146
- ds2202_can_msg_txqueue_init 173
- ds2202_can_msg_txqueue_level_read 177
- ds2202_can_msg_wakeup 179
- ds2202_can_msg_write 168
- ds2202_can_service_read 191
- ds2202_can_service_register 188
- ds2202_can_service_request 190
- ds2202_can_subint_handler_install 194
- ds2202_canChannel 118
- ds2202_canMsg 123
- ds2202_canService 119
- ds2202_d2f 69
- ds2202_dac_out 43
- ds2202_digin_threshold_set 11
- ds2202_digout_hs_vbat1_clear 20
- ds2202_digout_hs_vbat1_set 19
- ds2202_digout_hs_vbat1_write 17
- ds2202_digout_hs_vbat2_clear 24
- ds2202_digout_hs_vbat2_set 22
- ds2202_digout_hs_vbat2_write 21
- ds2202_digout_ls_clear 16
- ds2202_digout_ls_set 15
- ds2202_digout_ls_write 14
- ds2202_digout_mode_set 12
- ds2202_f2d 73
- ds2202_init 10
- ds2202_pwm_in 65
- ds2202_pwm_out 61
- ds2202_timing_in_mode_set 56
- ds2202_timing_out_mode_set 53
- ds2202_bytes2word 111
- ds2202_config 91
- ds2202_disable 100
- ds2202_enable 99
- ds2202_error_read 101
- ds2202_fifo_reset 98
- ds2202_init 90
- ds2202_ISR 81
- ds2202_LSR 83
- ds2202_MSR 84
- ds2202_receive 95
- ds2202_receive_fifo_level 103
- ds2202_receive_term 97
- ds2202_set 104
- ds2202_status_read 103
- ds2202_subint_disable 108
- ds2202_subint_enable 107
- ds2202_subint_handler_inst 106
- ds2202_subint_handler_t 85
- ds2202_transmit 94
- ds2202_transmit_fifo_level 102
- ds2202_word2bytes 109
- ds2202_Channel 87

E

execution times
DS2202 207

F

frequency measurement 71
function execution times
DS2202 207

I

initialization functions 9

L

Local Program Data folder 8

R

real-time library
DS2202 7

S

setup functions 9
slave CAN
communication channel 116
slave CAN access functions 116
square-wave signal generation 67
subinterrupt
type of 77

T

timing mode 53
trigger level 76

