DS2211 HIL I/O Board

RTLib Reference

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

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

Contents

Slave CAN Access Functions 455

About This Reference

About this reference

The DS2211 Real-Time Library (RTLib) provides the C functions and macros you need to program the DS2211 HIL I/O Board.

Demo examples

There are examples for some features included in this documentation. You will find the relevant files after the installation of your dSPACE software in `<RCP_HIL_InstallationPath>\Demos\Ds100x\IOBoards\Ds2211`. Use ControlDesk to load and start the real-time application on your processor board.

Symbols

dSPACE user documentation uses the following symbols:

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.
To access the Web version, you must have a *mydSPACE* account.

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

Overall Functions

Purpose

To get information on overall functions, initialization and setup functions, and conversion macros.

Where to go from here

Information in this section

Standard Definitions

Standard Definitions

Introduction

To be independent from the processor board used, some board-related functions are mapped to overall functions.

Standard definitions

You can find the following standard definitions in the include file `Dsstd.h`:

- `RTLIB_BACKGROUND_SERVICE`
- `RTLIB_INT_ENABLE`
- `RTLIB_SLAVE_LOAD_ACKNOWLEDGE`
- `RTLIB_SRT_DISABLE`
- `RTLIB_SRT_ENABLE`
- `RTLIB_SRT_ISR_BEGIN`
- `RTLIB_SRT_ISR_END`
- `RTLIB_SRT_START`
- `RTLIB_TIC_INIT`
- `RTLIB_TIC_READ`
- `RTLIB_TIC_START`
- `init`

Initialization and Setup Functions

Introduction

Before you can use the DS2211 you have to perform an initialization process, set the master or slave mode of the board and the working modes of some I/O units.

Where to go from here

Information in this section

ds2211_init

Syntax

```
void ds2211_init(Int32 base)
```

Include file

`ds2211.h`

Purpose

To perform the basic initialization of the DS2211.

Description

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

- Checks for board presence
- Sets the board to master mode (refer to ds2211_mode_set on page 21) and resets all board functions to their default settings (writes 0xA5000000 to ID register).

Function	Default Setting
Board mode	Master
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
Transformer outputs (APU, slave DSP)	Disabled
Digital wave form outputs	Enables clearing
Complex comparator	Sets threshold A to 2.5 V Sets threshold B to 5.0 V Sets hysteresis to 0.2 V Sets capture mode to "A leading to A trailing edge"
Analog outputs (DAC)	Sets 0 V output voltage
Resistor outputs (RES)	High-Z (1 MΩ)
Slave DSP VC33	Reset
Slave MC CAN-80C167	Reset

- Allocates and initializes the temporary ignition and injection capture buffers and the corresponding data structures. The buffers are cleared and the number of expected events is set to the default value `"0"` for all channels.
- Clears the interrupt position flags, the dual-port memories, and clears the wave form data buffer.
- Depending on the global debug status flag DEBUG_INIT, the function outputs an info message signaling the completion of the initialization. Refer to Initialization (DS1006 RTLib Reference 🕮) or Initialization (DS1007 RTLib Reference 🕮).

The **`ds2211_init`** function avoids multiple execution of the initialization code.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

Return value

None

Messages

The following message is defined:

Type	Message	Meaning
Error	ds2211_init(board_offset): Board not found!	There is no DS2211 at the given board index (offset of the PHS-bus address).

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_mode_set 21

ds2211_mode_set

Syntax

```
void ds2211_mode_set(
      Int32 base,
      Int32 mode)
```

Include file

`ds2211.h`

Purpose

To set the DS2211 board to master or slave mode.

Description

The APU of the master board calculates the engine position and supplies the information to slave DS2211 boards via the time-base connector.

> **Note**
>
> When cascading several boards, exactly one board has to be specified as the master, the other ones as slaves. Otherwise, the angular processing unit will not work correctly.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

mode The following symbols are predefined to set the mode:

Predefined Symbol	Meaning
DS2211_SLAVE_MODE	Sets the board to slave mode.
DS2211_MASTER_MODE	Sets the board to master mode.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_init..20

ds2211_digin_threshold_set

Syntax

```
void ds2211_digin_threshold_set(
      Int32 base,
      Int32 channel,
      dsfloat value)
```

Include file

`ds2211.h`

Purpose

To set the threshold level for the digital inputs DIG_IN and PWM_IN of the DS2211 board channelwise.

Description

For further information, refer to Digital Inputs (PHS Bus System Hardware Reference 📖).

Note

- After initialization, the threshold of the digital input pins is 2.5 V.
- The pin PWM_IN7 is shared with INJ7 and DIG_IN23. The pin PWM_IN8 is shared with INJ8 and DIG_IN24. Therefore, the settings can also be made via the setup function for the complex comparator of the injection capture unit (see `ds2211_apu_injection_cc_setup` on page 178).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

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

Predefined Symbol	Description
DS2211_THRESH_DIGIN1 (= DS2211_THRESH_PWMIN9) … DS2211_THRESH_DIGIN16 (= DS2211_THRESH_PWMIN24)	Digital inputs 1 … 16 (shared with PWM signal inputs 9 … 24)
DS2211_THRESH_DIGIN17 (= DS2211_THRESH_PWMIN1) … DS2211_THRESH_DIGIN24 (= DS2211_THRESH_PWMIN8)	Digital inputs 17 … 24 (shared with PWM signal inputs 1 … 8)
DS2211_THRESH_PWMIN1 (= DS2211_THRESH_DIGIN17) … DS2211_THRESH_PWMIN8 (= DS2211_THRESH_DIGIN24)	PWM signal inputs 1 … 8 (shared with digital inputs 17 … 24)
DS2211_THRESH_PWMIN9 (= DS2211_THRESH_DIGIN1) … DS2211_THRESH_PWMIN24 (= DS2211_THRESH_DIGIN16)	PWM signal inputs 9 … 24 (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 551.

Related topics

References

ds2211_init..20

ds2211_digout_mode_set

Syntax

```
void ds2211_digout_mode_set(
      Int32 base,
      Int32 mode)
```

Include file

`ds2211.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 DS2211. The setting refers to the bit I/O unit, PWM signal generation, and the digital outputs of the crankshaft and the camshaft sensor. For information on the digital outputs, refer to Digital Outputs (PHS Bus System Hardware Reference 🕮).

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

1. Call **`ds2211_digout_mode_set`** to enable all digital outputs.
2. Call **`ds2211_digout_hs_vbat1_set`** or **`ds2211_digout_hs_vbat2_set`** to select a supply rail for the respective digital output.
3. Invoke **`ds2211_digout_ls_set`** to ensure push-pull-driver functionality (thus enabling the corresponding low-side switch).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

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

Predefined Symbol	Meaning
DS2211_DIGOUT_ENABLE	Enables the digital output circuits.
DS2211_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 551.

Related topics

References

Digital Outputs (PHS Bus System Hardware Reference 📖)
ds2211_digout_hs_vbat1_set........ 31
ds2211_digout_hs_vbat2_set........ 35
ds2211_digout_ls_set........ 26
ds2211_init........ 20

ds2211_digout_ls_write

Syntax

```
void ds2211_digout_ls_write(
   Int32 base,
   Int32 channel)
```

Include file

`ds2211.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 **`ds2211_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 DS2211 board

channel Defines the channels whose low-side switch is to be enabled or disabled in the range 0x00000000 … 0x0FFFFFFF. '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
DS2211_OUTSTP_DIGOUT1 … DS2211_OUTSTP_DIGOUT16	Bit 0 … Bit 15	DIG_OUTx
DS2211_ OUTSTP _PWMOUT1 … DS2211_ OUTSTP _PWMOUT9	Bit 16 … Bit 24	PWM_OUTx
DS2211_ OUTSTP _DIGCRANK	Bit 25	CRANK_DIG
DS2211_ OUTSTP _DIGCAM_A	Bit 26	CAM1_DIG
DS2211_ OUTSTP _DIGCAM_B	Bit 27	CAM2_DIG
DS2211_ OUTSTP _DIGCAM_C (DS2211_OUTSTP_DIGOUT15)	Bit 14	CAM3_DIG (shared with DIG_OUT15)
DS2211_ OUTSTP _DIGCAM_D (DS2211_OUTSTP_DIGOUT16)	Bit 15	CAM4_DIG (shared with DIG_OUT16)

Default value

All low-side switches are enabled after reset.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_digout_ls_clear........28
ds2211_digout_ls_set........26
ds2211_digout_mode_set........24
ds2211_init........20

ds2211_digout_ls_set

Syntax

```
void ds2211_digout_ls_set(
   Int32 base,
   Int32 channel)
```

Include file

`ds2211.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 `ds2211_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 `ds2211_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 DS2211 board

channel Defines the channels which low side is to be enabled in the range 0x00000000 … 0x0FFFFFFF. '1' enables the low-side switch, '0' 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
DS2211_OUTSTP_DIGOUT1 … DS2211_OUTSTP_DIGOUT16	Bit 0 … Bit 15	DIG_OUTx
DS2211_ OUTSTP _PWMOUT1 … DS2211_ OUTSTP _PWMOUT9	Bit 16 … Bit 24	PWM_OUTx
DS2211_ OUTSTP _DIGCRANK	Bit 25	CRANK_DIG
DS2211_ OUTSTP _DIGCAM_A	Bit 26	CAM1_DIG
DS2211_ OUTSTP _DIGCAM_B	Bit 27	CAM2_DIG
DS2211_ OUTSTP _DIGCAM_C (DS2211_OUTSTP_DIGOUT15)	Bit 14	CAM3_DIG (shared with DIG_OUT15)
DS2211_ OUTSTP _DIGCAM_D (DS2211_OUTSTP_DIGOUT16)	Bit 15	CAM4_DIG (shared with DIG_OUT16)

Default value

All low side switches are enabled after reset.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_digout_ls_clear.. 28
ds2211_digout_ls_write.. 25

ds2211_digout_mode_set.......... 24
ds2211_init.......... 20

ds2211_digout_ls_clear

Syntax

```
void ds2211_digout_ls_clear(
   Int32 base,
   Int32 channel)
```

Include file

`ds2211.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 `ds2211_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 DS2211 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
DS2211_OUTSTP_DIGOUT1 … DS2211_OUTSTP_DIGOUT16	Bit 0 … Bit 15	DIG_OUTx
DS2211_OUTSTP_PWMOUT1 … DS2211_OUTSTP_PWMOUT9	Bit 16 … Bit 24	PWM_OUTx
DS2211_OUTSTP_DIGCRANK	Bit 25	CRANK_DIG
DS2211_OUTSTP_DIGCAM_A	Bit 26	CAM1_DIG
DS2211_OUTSTP_DIGCAM_B	Bit 27	CAM2_DIG

Predefined Symbol	Bit	Signal
DS2211_OUTSTP_DIGCAM_C (DS2211_OUTSTP_DIGOUT15)	Bit 14	CAM3_DIG (shared with DIG_OUT15)
DS2211_OUTSTP_DIGCAM_D (DS2211_OUTSTP_DIGOUT16)	Bit 15	CAM4_DIG (shared with DIG_OUT16)

Default value

All low side switches are enabled after reset.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_digout_hs_vbat1_write

Syntax

```
void ds2211_digout_hs_vbat1_write(
   Int32 base,
   Int32 channel)
```

Include file

`ds2211.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 `ds2211_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 DS2211 board

channel Defines the channels whose high-side switch is to be enabled or disabled in the range 0x00000000 … 0x0FFFFFFF. '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**
DS2211_OUTSTP_DIGOUT1 … DS2211_OUTSTP_DIGOUT16	Bit 0 … Bit 15	DIG_OUTx
DS2211_ OUTSTP _PWMOUT1 … DS2211_ OUTSTP _PWMOUT9	Bit 16 … Bit 24	PWM_OUTx
DS2211_ OUTSTP _DIGCRANK	Bit 25	CRANK_DIG
DS2211_ OUTSTP _DIGCAM_A	Bit 26	CAM1_DIG
DS2211_ OUTSTP _DIGCAM_B	Bit 27	CAM2_DIG
DS2211_ OUTSTP _DIGCAM_C (DS2211_OUTSTP_DIGOUT15)	Bit 14	CAM3_DIG (shared with DIG_OUT15)
DS2211_ OUTSTP _DIGCAM_D (DS2211_OUTSTP_DIGOUT16)	Bit 15	CAM4_DIG (shared with DIG_OUT16)

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 551.

Related topics

References

ds2211_digout_hs_vbat1_clear........ 32
ds2211_digout_hs_vbat1_set........ 31
ds2211_digout_mode_set........ 24
ds2211_init........ 20

ds2211_digout_hs_vbat1_set

Syntax

```
void ds2211_digout_hs_vbat1_set(
   Int32 base,
   Int32 channel)
```

Include file

`ds2211.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 `ds2211_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 `ds2211_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 DS2211 board

channel Defines the channels whose high-side switch is to be enabled in the range 0x00000000 … 0x0FFFFFFF. '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
DS2211_OUTSTP_DIGOUT1 … DS2211_OUTSTP_DIGOUT16	Bit 0 … Bit 15	DIG_OUTx
DS2211_ OUTSTP _PWMOUT1 … DS2211_ OUTSTP _PWMOUT9	Bit 16 … Bit 24	PWM_OUTx
DS2211_ OUTSTP _DIGCRANK	Bit 25	CRANK_DIG
DS2211_ OUTSTP _DIGCAM_A	Bit 26	CAM1_DIG
DS2211_ OUTSTP _DIGCAM_B	Bit 27	CAM2_DIG

Predefined Symbol	Bit	Signal
DS2211_ OUTSTP _DIGCAM_C (DS2211_OUTSTP_DIGOUT15)	Bit 14	CAM3_DIG (shared with DIG_OUT15)
DS2211_ OUTSTP _DIGCAM_D (DS2211_OUTSTP_DIGOUT16)	Bit 15	CAM4_DIG (shared with DIG_OUT16)

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 551.

Related topics

References

ds2211_digout_hs_vbat1_clear

Syntax

```
void ds2211_digout_hs_vbat1_clear (
   Int32 base,
   Int32 channel)
```

Include file

`ds2211.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 `ds2211_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 DS2211 board

channel Defines the channels whose high-side switch is to be disabled in the range 0x00000000 … 0x0FFFFFFF. '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
DS2211_OUTSTP_DIGOUT1 … DS2211_OUTSTP_DIGOUT16	Bit 0 … Bit 15	DIG_OUTx
DS2211_ OUTSTP _PWMOUT1 … DS2211_ OUTSTP _PWMOUT9	Bit 16 … Bit 24	PWM_OUTx
DS2211_ OUTSTP _DIGCRANK	Bit 25	CRANK_DIG
DS2211_ OUTSTP _DIGCAM_A	Bit 26	CAM1_DIG
DS2211_ OUTSTP _DIGCAM_B	Bit 27	CAM2_DIG
DS2211_ OUTSTP _DIGCAM_C (DS2211_OUTSTP_DIGOUT15)	Bit 14	CAM3_DIG (shared with DIG_OUT15)
DS2211_ OUTSTP _DIGCAM_D (DS2211_OUTSTP_DIGOUT16)	Bit 15	CAM4_DIG (shared with DIG_OUT16)

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 551.

Related topics

References

ds2211_digout_hs_vbat1_set	31
ds2211_digout_hs_vbat1_write	29
ds2211_digout_mode_set	24
ds2211_init	20

ds2211_digout_hs_vbat2_write

Syntax

```
void ds2211_digout_hs_vbat2_write(
   Int32 base,
   Int32 channel)
```

Include file

`ds2211.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 `ds2211_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 DS2211 board

channel Defines the channels whose high-side switch is to be enabled or disabled in the range 0x00000000 … 0x0FFFFFFF. '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
DS2211_OUTSTP_DIGOUT1 … DS2211_OUTSTP_DIGOUT16	Bit 0 … Bit 15	DIG_OUTx
DS2211_ OUTSTP _PWMOUT1 … DS2211_ OUTSTP _PWMOUT9	Bit 16 … Bit 24	PWM_OUTx
DS2211_ OUTSTP _DIGCRANK	Bit 25	CRANK_DIG
DS2211_ OUTSTP _DIGCAM_A	Bit 26	CAM1_DIG
DS2211_ OUTSTP _DIGCAM_B	Bit 27	CAM2_DIG
DS2211_ OUTSTP _DIGCAM_C (DS2211_OUTSTP_DIGOUT15)	Bit 14	CAM3_DIG (shared with DIG_OUT15)
DS2211_ OUTSTP _DIGCAM_D (DS2211_OUTSTP_DIGOUT16)	Bit 15	CAM4_DIG (shared with DIG_OUT16)

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 551.

Related topics

References

ds2211_digout_hs_vbat2_clear	36
ds2211_digout_hs_vbat2_set	35
ds2211_digout_mode_set	24
ds2211_init	20

ds2211_digout_hs_vbat2_set

Syntax

```
void ds2211_digout_hs_vbat2_set(
   Int32 base,
   Int32 channel)
```

Include file

`ds2211.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 `ds2211_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 `ds2211_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 DS2211 board

channel Defines the channels whose high-side switch is to be enabled in the range 0x00000000 … 0x0FFFFFFF. '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
DS2211_OUTSTP_DIGOUT1 … DS2211_OUTSTP_DIGOUT16	Bit 0 … Bit 15	DIG_OUTx
DS2211_ OUTSTP _PWMOUT1 … DS2211_ OUTSTP _PWMOUT9	Bit 16 … Bit 24	PWM_OUTx
DS2211_ OUTSTP _DIGCRANK	Bit 25	CRANK_DIG
DS2211_ OUTSTP _DIGCAM_A	Bit 26	CAM1_DIG
DS2211_ OUTSTP _DIGCAM_B	Bit 27	CAM2_DIG
DS2211_ OUTSTP _DIGCAM_C (DS2211_OUTSTP_DIGOUT15)	Bit 14	CAM3_DIG (shared with DIG_OUT15)
DS2211_ OUTSTP _DIGCAM_D (DS2211_OUTSTP_DIGOUT16)	Bit 15	CAM4_DIG (shared with DIG_OUT16)

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 551.

Related topics

References

ds2211_digout_hs_vbat2_clear 36
ds2211_digout_hs_vbat2_write 34
ds2211_digout_mode_set 24
ds2211_init 20

ds2211_digout_hs_vbat2_clear

Syntax

```
void ds2211_digout_hs_vbat2_clear (
   Int32 base,
   Int32 channel)
```

Include file

`ds2211.h`

Purpose

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

Description

This function disables only the digital outputs specified with `channel` parameter. Use `ds2211_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 DS2211 board

channel Defines the channels whose high-side switch is to be disabled in the range 0x00000000 … 0x0FFFFFFF. '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
DS2211_OUTSTP_DIGOUT1 … DS2211_OUTSTP_DIGOUT16	Bit 0 … Bit 15	DIG_OUTx
DS2211_ OUTSTP _PWMOUT1 … DS2211_ OUTSTP _PWMOUT9	Bit 16 … Bit 24	PWM_OUTx
DS2211_ OUTSTP _DIGCRANK	Bit 25	CRANK_DIG
DS2211_ OUTSTP _DIGCAM_A	Bit 26	CAM1_DIG
DS2211_ OUTSTP _DIGCAM_B	Bit 27	CAM2_DIG
DS2211_ OUTSTP _DIGCAM_C (DS2211_OUTSTP_DIGOUT15)	Bit 14	CAM3_DIG (shared with DIG_OUT15)
DS2211_ OUTSTP _DIGCAM_D (DS2211_OUTSTP_DIGOUT16)	Bit 15	CAM4_DIG (shared with DIG_OUT16)

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 551.

Related topics

References

ds2211_digout_hs_vbat2_set 35
ds2211_digout_hs_vbat2_write 34
ds2211_digout_mode_set 24
ds2211_init 20

ds2211_digwform_mode_set

Syntax

```
void ds2211_digwform_mode_set(
      Int32 base,
      Int32 mode)
```

Include file

`ds2211.h`

Purpose

To enable the digital wave form outputs of the angular processing unit to be cleared.

Description

If this function is called with the parameter DS2211_DIGWFORM_CLEAR_ENABLE, the hardware clears the digital wave form outputs when the APU is stopped or the velocity is 0.

> **Note**
>
> After initialization, clearing is enabled.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

mode Mode of the digital wave form outputs. The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_DIGWFORM_CLEAR_ENABLE	The hardware clears the digital wave form outputs if the APU is stopped or the velocity is 0.
DS2211_DIGWFORM_CLEAR_DISABLE	Digital wave form outputs are not cleared.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_init..20

ds2211_apu_transformer_mode_set

Syntax

```
void ds2211_apu_transformer_mode_set(
     Int32 base,
     Int32 mode)
```

Include file

`ds2211.h`

Purpose

To enable or disable the D/A converter connected to the APU transformers.

Description

This function applies both to the crankshaft and camshaft signal generation of the APU, and to the knock signal and wheel speed signal generation of the slave DSP.

The transformers can be enabled or disabled (bypassed) via jumper settings, refer to Transformer Outputs (APU and Slave DSP) (PHS Bus System Hardware Reference 🕮).

Note

After initialization, the D/A converters connected to the APU transformers are disabled.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

mode Mode of the transformer output circuits. The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_APU_TRANSFORMER_ENABLE	Enables the D/A converters connected to the APU transformers.
DS2211_APU_TRANSFORMER_DISABLE	Disables the D/A converters connected to the APU transformers.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_init.. 20
Transformer Outputs (APU and Slave DSP) (PHS Bus System Hardware Reference 📖)

Conversion Macros

Introduction

There are some macros to convert a position and speed.

Where to go from here

Information in this section

DS2211_DEG

Syntax

```
dsfloat DS2211_DEG(dsfloat position)
```

Include file

`ds2211.h`

Purpose

To convert an engine position from radians to degrees.

Parameters

position	Engine position in radians

Return value

Engine position in degrees

Related topics

References

DS2211_RAD 42

DS2211_RAD

Syntax

```
dsfloat DS2211_RAD(dsfloat position)
```

Include file

`ds2211.h`

Purpose

To convert an engine position from degrees to radians.

Parameters

position	Engine position in degrees

Return value

Engine position in radians

Related topics

References

DS2211_DEG.. 41

DS2211_RPM

Syntax

```
dsfloat DS2211_RPM(dsfloat speed)
```

Include file

`ds2211.h`

Purpose

To convert the given speed (angle velocity) from radians per second (rad/s) to revolutions per minute (rpm).

Parameters

speed	Speed value in rad/s

Return value

Speed in rpm

Related topics

References

DS2211_RAD_S 43

DS2211_RAD_S

Syntax

```
dsfloat DS2211_RAD_S(dsfloat speed)
```

Include file

`ds2211.h`

Purpose

To convert the given speed (angle velocity) from revolutions per minute (rpm) to radians per second (rad/s).

Parameters

speed	Speed given in rpm

Return value

Speed in rad/s

Related topics

References

DS2211_RPM 42

Sensor and Actuator Interface

Introduction

The sensor and actuator interface (SAI) consists of standard I/O components and timing I/O components.

Where to go from here

Information in this section

Information in other sections

Sensor and Actuator Interface (DS2211 Features 📖)
The DS2211 has a sensor and actuator interface for standard I/O components and timing I/O components.

Wheel Speed Sensor Simulation..318
You can simulate a wheel speed sensor using a ready-to-use application implemented on the slave DSP.

ADC Unit

Introduction

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

Where to go from here

Information in this section

Basics on Accessing A/D Converters

Accessing 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

Basics

ADC Unit (DS2211 Features 🕮)

References

Analog Inputs (PHS Bus System Hardware Reference 🕮)

Example of Fast Read and Standard Read

Example

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

```
#include <Brtenv.h>
/* basic real-time environment */
#include <Ds2211.h>
/*----------------------------------------------------------*/
#define DT 1e-3                /* 1 ms simulation step size */
dsfloat adc_data[4] = {0, 0, 0, 0};   /* A/D channel values */
/* variables for execution time profiling */
dsfloat exec_time;                        /* execution time */
/*----------------------------------------------------------*/
void isr_t1()          /* timer1 interrupt service routine */
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
   /* start A/D channels 1 … 4 for conversion */
   ds2211_adc_start(DS2211_1_BASE, DS2211_ADC_START4);
   /* read A/D channels 1 … 4 */
   ds2211_adc_block_in_fast(DS2211_1_BASE, adc_data);
   exec_time = RTLIB_TIC_READ(); /* calculate execution time*/
   RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
}
/*----------------------------------------------------------*/
void main()
{
   init();                    /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);   /* initialize DS2211 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   RTLIB_TIC_START(DT, isr_t1);  /*init sampling clock timer*/
   RTLIB_TIC_INIT();
   while(1)                           /* background process */
   {
      RTLIB_BACKGROUND_SERVICE();
   } /* while(1) */
} /* main() */
```

Related topics

Examples

Example of Multiple Read.. 50
Example of Single Read.. 49

References

ds2211_adc_block_in_fast.. 53
ds2211_adc_start.. 51
ds2211_init.. 20

Example of Single Read

Example

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

```
#include <Brtenv.h>          /* basic real-time environment */
#include <Ds2211.h>
/*----------------------------------------------------------*/
#define DT 1e-3                /* 1 ms simulation step size */
Int32   channel = 6;                  /* A/D channel number */
dsfloat adc_data;                      /* A/D channel value */
/* variables for execution time profiling */
dsfloat exec_time;                        /* execution time */
/*----------------------------------------------------------*/
void isr_t1()           /* timer1 interrupt service routine */
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
   /* start A/D channels 1-8 for conversion */
   ds2211_adc_start(DS2211_1_BASE, DS2211_ADC_MASK(6));
   /* read A/D channel 6 */
   ds2211_adc_single_in(DS2211_1_BASE, channel, &adc_data);
   exec_time = RTLIB_TIC_READ();    /* calc. execution time */
   RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
}
/*------------------------------------------------------------*/
void main()
{
   init();                       /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);   /* initialize DS2211 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* initialize sampling clock timer */
   RTLIB_SRT_START(DT, isr_t1);
   RTLIB_TIC_INIT();
   while(1)                              /* background process */
   {
         RTLIB_BACKGROUND_SERVICE();
   } /* while(1) */
} /* main() */
```

Related topics

Examples

Example of Fast Read and Standard Read 48
Example of Multiple Read 50

References

ds2211_adc_single_in 54
ds2211_adc_start 51
ds2211_init 20

Example of Multiple Read

Example

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

```
#include <Brtenv.h>          /* basic real-time environment */
#include <Ds2211.h>
/*------------------------------------------------------------*/
#define DT 1e-3                    /* 1 ms simulation step size */
Int32   count = 5;         /* number of A/D channels to read */
Int32   channels[5] = {1, 5, 6, 12, 13}; /* A/D channel num.*/
dsfloat adc_data[5] = {0, 0, 0, 0, 0}; /* A/D channel value */
/* variables for execution time profiling */
dsfloat exec_time;                          /* execution time */
/*------------------------------------------------------------*/
void isr_t1()            /* timer1 interrupt service routine */
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
   /* start A/D channels for conversion */
   ds2211_adc_block_start(DS2211_1_BASE);
   /* read A/D channel 6 */
   ds2211_adc_block_in(DS2211_1_BASE, adc_data);
   exec_time = RTLIB_TIC_READ();    /* calc. execution time */
   RTLIB_SRT_ISR_END(); /* end of interrupt service routine */
}
/*----------------------------------------------------------*/
void main()
{
   init();                    /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);   /* initialize DS2211 board */
   ds2211_adc_block_init(DS2211_1_BASE, count, channels);
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* initialize sampling clock timer */
   RTLIB_SRT_START(DT, isr_t1);
   RTLIB_TIC_INIT();
   while(1)                           /* background process */
   {
      RTLIB_BACKGROUND_SERVICE();
   } /* while(1) */
} /* main() */
```

Related topics

Examples

Example of Fast Read and Standard Read 48
Example of Single Read 49

References

ds2211_adc_block_in 58
ds2211_adc_block_init 55
ds2211_adc_block_start 57
ds2211_init 20

ds2211_adc_start

Syntax

```
void ds2211_adc_start(
       Int32 base,
       Int32 mask)
```

Include file

`ds2211.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 (DS2211 Features 📖).

Description

The A/D conversion is performed sequentially for the selected channels.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

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

Predefined Symbol	Meaning	Value
DS2211_ADC_START4	Starts channels 1 … 4.	1...4
DS2211_ADC_START8	Starts channels 1 … 8.	1...8
DS2211_ADC_START12	Starts channels 1 … 12.	1...12
DS2211_ADC_START16	Starts channels 1 … 16.	1...16
DS2211_ADC_MASK (channel)	Starts the block the given channel belongs to. Use this symbol in connection with `ds2211_adc_single_in`.	1 … 4, 1 … 8, 1 … 12, or 1 … 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of Fast Read and Standard Read..48
Example of Single Read..49

References

ds2211_adc_block_in_fast..53
ds2211_adc_single_in...54

ds2211_adc_block_in_fast

Syntax

```
void ds2211_adc_block_in_fast(
      Int32 base,
      dsfloat *data)
```

Include file

`ds2211.h`

Purpose

To wait until ADC conversion finished and reads ADC input values.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS2211 Features).

Description

Polls the ADC busy flag until conversion is complete and then reads the input values of the channels selected by `ds2211_adc_start`. Input values are scaled to the range 0 … 1.0 and are written to memory starting at the address given by the `data` parameter.

> **Note**
>
> ADC inputs are differential inputs. Each input has an individual ground sense line ($\overline{\text{ADCx}}$), which must be connected to the plant near the sensor, or connected to GND at the DS2211 connector, for all ADC channels used. The voltage difference of (ADCx - $\overline{\text{ADCx}}$) is measured by the ADC.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

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.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of Fast Read and Standard Read..48

References

ds2211_adc_start..51

ds2211_adc_single_in

Syntax

```
void ds2211_adc_single_in(
      Int32 base,
      Int32 channel,
      dsfloat *value)
```

Include file

`ds2211.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 (DS2211 Features 📖).

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 `ds2211_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 ($\overline{\text{ADCx}}$), which must be connected to the plant near the sensor, or connected to GND at the DS2211 connector, for all ADC channels used. The voltage difference of (ADCx – $\overline{\text{ADCx}}$) is measured by the ADC.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

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

Predefined Symbol	Meaning
DS2211_ADC_CH1 … DS2211_ADC_CH16	A/D channel 1 … 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 551.

Related topics

Examples

Example of Single Read.. 49

References

ds2211_adc_start.. 51

ds2211_adc_block_init

Syntax

```
void ds2211_adc_block_init(
      Int32 base,
      Int32 count,
      Int32 *channels)
```

Include file

`ds2211.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 (DS2211 Features 📖).

Description

`ds2211_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 `ds2211_adc_block_init` the conversion will be started for channels 1 … 8.

Parameters

base Specifies the PHS-bus base address of the DS2211 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
DS2211_ADC_CH1 … DS2211_ADC_CH16	A/D channel 1 … A/D channel 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of Multiple Read.. 50

References

ds2211_adc_block_start.. 57

ds2211_adc_block_start

Syntax

```
void ds2211_adc_block_start(Int32 base)
```

Include file

`ds2211.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 (DS2211 Features 🕮).

Description

The A/D conversion is started for the channels selected by the `ds2211_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 `ds2211_adc_block_init`, the conversion is started for channels 1 … 8.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of Multiple Read 50

References

ds2211_adc_block_in 58
ds2211_adc_block_init 55

ds2211_adc_block_in

Syntax

```
void ds2211_adc_block_in(
      Int32 base,
      dsfloat *data)
```

Include file

`ds2211.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 (DS2211 Features 🕮).

Description

You have to start the channels with `ds2211_adc_block_start` first. Then `ds2211_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 parameter `data`.

Parameters

base Specifies the PHS-bus base address of the DS2211 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 551.

Related topics

Examples

Example of Multiple Read.. 50

References

ds2211_adc_block_init.. 55
ds2211_adc_block_start.. 57

DAC Unit

Introduction

The digital analog converter unit comprises all functions to write to D/A channels.

Where to go from here

Information in this section

Information in other sections

DAC Unit (DS2211 Features 📖)
Introduction of the digital/analog converter (DAC) Unit

Example of the DAC Unit

Example

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

```
#include <Brtenv.h>                 /* basic real-time environment */
#include <Ds2211.h>
#define DT 1e-3                       /* 1 ms simulation step size */
dsfloat val = 0.0;                           /* D/A output value */
dsfloat exec_time;                               /* execution time */
void isr_t1()                  /* timer1 interrupt service routine */
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
   ds2211_dac_out(DS2211_1_BASE, DS2211_DAC_CH1, val);
   val = (val == 0.0) ? 0.5 : 0.0;         /* toggle D/A value */
   ds2211_dac_out(DS2211_1_BASE, DS2211_DAC_CH2, val);
   exec_time = RTLIB_TIC_READ();       /* calculate execution time */
   RTLIB_SRT_ISR_END();        /* end of interrupt service routine */
}
```

```
void main()
{
   init();                           /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);          /* initialize DS2211 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* initialize sampling clock timer */
   RTLIB_SRT_START(DT, isr_t1);
   RTLIB_TIC_INIT();
   while(1)                                  /* background process */
   {
      RTLIB_BACKGROUND_SERVICE();
   } /* while(1) */
} /* main() */
```

Related topics

References

ds2211_dac_out 60
ds2211_init 20

ds2211_dac_out

Syntax

```
void ds2211_dac_out(
      Int32 base,
      Int32 channel,
      dsfloat value)
```

Include file

`ds2211.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 (DS2211 Features 📖).

Description

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

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number within the range of 1 … 20. You can also use the following predefined symbols:

Predefined Symbol	Meaning
DS2211_DAC_CH1 … DS2211_DAC_CH20	DAC channel 1 … 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...V_{REF}

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of the DAC Unit.. 59

Bit I/O Unit

Introduction

The bit I/O unit comprises all functions to read digital input signals and to write digital output signals.

Where to go from here

Information in this section

Information in other sections

Bit I/O Unit (DS2211 Features 📖)
Introduction of the bit (I/O) unit

Example of the Bit I/O Unit

Example

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

```
#include <Brtenv.h>                /* basic real-time environment */
#include <Ds2211.h>
```

```
#define DT 1e-3                      /* 1 ms simulation step size */
/* variables for ControlDesk */
UInt32 bitmap = 0;
UInt32 mask_clear = 0;
UInt32 mask_set = 0;
/* variables for execution time profiling */
dsfloat exec_time;                                /* execution time */
void isr_t1()                  /* timer1 interrupt service routine */
{
   static UInt32 i = 0;
   ts_timestamp_type ts;
   /* clears I/O port i and sets I/O port i to "1" */
   mask_clear = i;
   mask_set = i - 1;
   ds2211_bit_io_clear(DS2211_1_BASE, (0x0001 << mask_clear));
   ds2211_bit_io_set(DS2211_1_BASE, (0x0001 << mask_set));
    /* increments i until channel 16 is reached */
   i++;
   if (i == 16)
      i = 0;
   /* reads the 16 bit I/O port                                  */
   ds2211_bit_io_in(DS2211_1_BASE, &bitmap);
   exec_time = RTLIB_TIC_READ();     /* calculate execution time */
   RTLIB_SRT_ISR_END();      /* end of interrupt service routine */
}
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   rs_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
void main()
{
   init();                           /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);        /* initialize DS2211 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* enable digital outputs */
   ds2211_digout_mode_set(DS2211_1_BASE, DS2211_DIGOUT_ENABLE);
   /* sets the Bit I/O ports 0 … 15 to "1" */
   ds2211_bit_io_out(DS2211_1_BASE, 0x0000FFFF);
   /* initialize sampling clock timer */
   RTLIB_SRT_START(DT, isr_t1);
   RTLIB_TIC_INIT();
   while(1)                                  /* background process */
   {
      RTLIB_BACKGROUND_SERVICE();
   } /* while(1) */
} /* main() */
```

Related topics

References

ds2211_bit_io_in

Syntax

```
void ds2211_bit_io_in(
      Int32 base,
      UInt32 *value)
```

Include file

`ds2211.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

The function reads the same digital input ports as the `ds2211_bit_io_in_group` function when group1 is specified. The `ds2211_bit_io_in` function is therefore obsolete, use the `ds2211_bit_io_in_group` function instead.

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS2211 Features 🕮).

> **Note**
>
> The inputs are 12 ... 42 V compatible. After initialization, the input threshold is set to 2.5 V. Use `ds2211_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 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

value Specifies the address where the input value is written.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of the Bit I/O Unit 62

References

ds2211_bit_io_in_group 65
ds2211_bit_io_out 66
ds2211_digin_threshold_set 22

ds2211_bit_io_in_group

Syntax

```
void ds2211_bit_io_in_group(
      Int32 base,
      Int32 group,
      UInt32 *value)
```

Include file

`ds2211.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

The function reads from different groups of digital input channels. For group1, 16 digital inputs are read. The returned value is therefore within the range 0 … 65535 (0xFFFF). For group2, only 8 digital inputs are read and the returned value is therefore within the range 0 … 255 (0xFF). The read inputs of group1 are the same as the digital inputs read by the `ds2211_bit_io_in` function.

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS2211 Features 🕮).

Note

- The inputs are 12 … 42 V compatible. After initialization, the input threshold is set to 2.5 V. Use `ds2211_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 🕮).
- The digital input channels DIG_IN17 … 24 are shared with PWM_IN1 … PWM_IN8.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

group Specifies the group of digital input channels, which status to be read. The following symbols are predefined:

Predefined Symbol	Value	Channels
DS2211_BITIO_IN_G1	0	DIG_IN1...16 (16 channels)
DS2211_BITIO_IN_G2	1	DIG_IN17...24 / PWM_IN1...8 (8 channels)

value Specifies the address where the input value is written.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of the Bit I/O Unit.. 62

References

ds2211_bit_io_out.. 66
ds2211_digin_threshold_set.. 22

ds2211_bit_io_out

Syntax

```
void ds2211_bit_io_out(
     Int32 base,
     UInt32 value)
```

Include file

`ds2211.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 `ds2211_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 (DS2211 Features 🕮).

> **Note**
>
> By default the outputs are disabled. Use `ds2211_digout_mode_set` to enable the digital outputs. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

value Output value within the range of 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
DS2211_BITIO_OUT1	0x0001	Sets bit 1
DS2211_BITIO_OUT2	0x0002	Sets bit 2
DS2211_BITIO_OUT3	0x0004	Sets bit 3
…	…	…
DS2211_BITIO_OUT16	0x8000	Sets bit 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of the Bit I/O Unit..62

References

ds2211_bit_io_in..64
ds2211_bit_io_set..68
ds2211_digout_mode_set..24

ds2211_bit_io_set

Syntax

```
void ds2211_bit_io_set(
      Int32 base,
      UInt32 mask)
```

Include file

`ds2211.h`

Purpose

To set individual digital output bits.

Description

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

> **Note**
>
> After initialization, the outputs are disabled. Use **`ds2211_digout_mode_set`** to enable the digital output ports. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 📖).

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS2211 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

mask Set mask within the range of 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
DS2211_BITIO_OUT1	0x0001	Sets bit 1
DS2211_BITIO_OUT2	0x0002	Sets bit 2
DS2211_BITIO_OUT3	0x0004	Sets bit 3
…	…	…
DS2211_BITIO_OUT16	0x8000	Sets bit 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of the Bit I/O Unit.. 62

References

ds2211_bit_io_clear.. 69
ds2211_digout_mode_set.. 24

ds2211_bit_io_clear

Syntax

```
void ds2211_bit_io_clear(
      Int32 base,
      UInt32 mask)
```

Include file

`ds2211.h`

Purpose

To clear individual digital output bits.

Description

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

Note

After initialization, the outputs are disabled. Use **`ds2211_digout_mode_set`** to enable the digital output ports. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 📖).

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS2211 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

mask Clear mask within the range of 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
DS2211_BITIO_OUT1	0x0001	Sets bit 1
DS2211_BITIO_OUT2	0x0002	Sets bit 2
DS2211_BITIO_OUT3	0x0004	Sets bit 3
…	…	…
DS2211_BITIO_OUT16	0x8000	Sets bit 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of the Bit I/O Unit.. 62

References

ds2211_bit_io_set.. 68
ds2211_digout_mode_set.. 24

D/R Converter

Introduction

The digital/resistance converter unit comprises all functions to simulate resistors.

Where to go from here

Information in this section

Information in other sections

D/R Converter (DS2211 Features 📖)
Introduction of the digital/resistance (D/R) converter

Example of the D/R Converter

Example

This example sets the resistor channel 1 to 1.5 kΩ.

```
#include <Brtenv.h>               /* basic real-time environment */
#include <Ds2211.h>
/*--------------------------------------------------------------*/
#define DT 1e-3                     /* 1 ms simulation step size */
dsfloat val = 1500.0;                  /* resistor output value */
/* variables for execution time profiling */
dsfloat exec_time;                             /* execution time */
/*--------------------------------------------------------------*/
void isr_t1()                  /* timer1 interrupt service routine */
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
   /* update resistor value */
   ds2211_resistance_out(DS2211_1_BASE, DS2211_RES_CH1, value);
   exec_time = RTLIB_TIC_READ();     /* calculate execution time */
   RTLIB_SRT_ISR_END();      /* end of interrupt service routine */
}
/*--------------------------------------------------------------*/
void main()
{
   init();                         /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);        /* initialize DS2211 board */
   /* set resistor1 output to 1.5 kOhm */
   ds2211_resistance_out(DS2211_1_BASE, DS2211_RES_CH1, value);
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* initialize sampling clock timer */
   RTLIB_SRT_START(DT, isr_t1);
   RTLIB_TIC_INIT();
   while(1)                                  /* background process */
   {
      RTLIB_BACKGROUND_SERVICE();
   } /* while(1) */
} /* main() */
```

Related topics

References

ds2211_init..20
ds2211_resistance_out...72

ds2211_resistance_out

Syntax

```
void ds2211_resistance_out(
       Int32 base,
       Int32 channel,
       dsfloat value)
```

Include file

`ds2211.h`

Purpose

To update the resistance output of the specified channel. After initialization the resistors are set to high-Z (1 MΩ).

I/O mapping

For information on the I/O mapping, refer to D/R Converter (DS2211 Features 🕮).

Description

`ds2211_resistance_out` writes the resistance value to the specified channel. For values smaller than 15.8 Ω the output saturates at 15.8 Ω. For values higher than 1 MΩ the output resistance becomes infinity.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_RES_CH1	Resistor channel 1
…	…
DS2211_RES_CH10	Resistor channel 10

value Output value within the range of 15.8 Ω … 1 MΩ. After initialization, the value is set to high-Z (1 MΩ).

The possible resistor values are calculated based on the values of the resistance output register INVRES (16 bit) according to the following formula R = (1 MΩ / INVRES + 0.5 Ω). The following values are possible:

INVRES	Resulting R
0	Infinity
1	1 MΩ
2	500 kΩ
…	…
65,535	15.8 Ω

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

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of the D/R Converter...71

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

ds2211_timing_out_mode_set

Syntax

```
void ds2211_timing_out_mode_set(
      Int32 base,
      Int32 channel,
      Int32 range,
      Int32 mode)
```

Include file

`ds2211.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 Specifies the PHS-bus base address of the DS2211 board.

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_PWMOUT_CH1	PWM channel 1
…	…
DS2211_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**
DS2211_TIMING_RANGE1	9.54 Hz	100 kHz	50 ns
DS2211_TIMING_RANGE2	4.77 Hz	100 kHz	100 ns
DS2211_TIMING_RANGE3	2.39 Hz	100 kHz	200 ns
DS2211_TIMING_RANGE4	1.20 Hz	100 kHz	400 ns
DS2211_TIMING_RANGE5	0.60 Hz	100 kHz	800 ns
DS2211_TIMING_RANGE6	0.30 Hz	100 kHz	1.6 µs
DS2211_TIMING_RANGE7	0.15 Hz	100 kHz	3.2 µs
DS2211_TIMING_RANGE8	75 mHz	78.12 kHz	6.4 µs
DS2211_TIMING_RANGE9	38 mHz	39.06 kHz	12.8 µs
DS2211_TIMING_RANGE10	19 mHz	19.53 kHz	25.6 µs
DS2211_TIMING_RANGE11	10 mHz	9.76 kHz	51.2 µs
DS2211_TIMING_RANGE12	5.0 mHz	4.88 kHz	103 µs
DS2211_TIMING_RANGE13	2.5 mHz	2.44 kHz	205 µs
DS2211_TIMING_RANGE14	1.2 mHz	1.22 kHz	410 µs
DS2211_TIMING_RANGE15	0.6 mHz	610.35 Hz	820 µs
DS2211_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**		
DS2211_TIMING_RANGE1	200 ns	10 µs	3.28 ms	50 ns
DS2211_TIMING_RANGE2	400 ns	10 µs	6.55 ms	100 ns
DS2211_TIMING_RANGE3	800 ns	10 µs	13.1 ms	200 ns
DS2211_TIMING_RANGE4	1.6 µs	10 µs	26.2 ms	400 ns
DS2211_TIMING_RANGE5	3.2 µs	10 µs	52.4 ms	800 ns
DS2211_TIMING_RANGE6	6.4 µs	10 µs	105 ms	1.6 µs
DS2211_TIMING_RANGE7	12.8 µs	12.8 µs	210 ms	3.2 µs
DS2211_TIMING_RANGE8	25.6 µs	25.6 µs	419 ms	6.4 µs

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	**Practical**		
DS2211_TIMING_RANGE9	51.2 µs	51.2 µs	839 ms	12.8 µs
DS2211_TIMING_RANGE10	102 µs	102 µs	1.68 s	25.6 µs
DS2211_TIMING_RANGE11	205 µs	205 µs	3.36 s	51.2 µs
DS2211_TIMING_RANGE12	410 µs	410 µs	6.71 s	102 µs
DS2211_TIMING_RANGE13	819 µs	819 µs	13.4 s	205 µs
DS2211_TIMING_RANGE14	1.64 ms	1.64 ms	26.8 s	410 µs
DS2211_TIMING_RANGE15	3.28 ms	3.28 ms	53.6 s	819 µs
DS2211_TIMING_RANGE16	6.55 ms	6.55 ms	107.3 s	1.64 ms

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

Mode	Meaning
DS2211_D2PWM	PWM signal generation.
DS2211_D2PWM_SYNCH_UPDATE	PWM signal generation with synchronous update.
DS2211_D2F_LOW	Square-wave signal generation, the output is set to low level.
DS2211_D2F_HIGH	Square-wave signal generation generation, the output is set to high level.
DS2211_D2F_HOLD	Square-wave signal generation 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 $T = T_{high} + T_{low}$ is constant. If you change the period during run time, synchronous PWM update cannot be ensured.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of PWM Signal Generation.. 80

References

Timing Mode.. 74

ds2211_timing_in_mode_set

Syntax

```
void ds2211_timing_in_mode_set(
      Int32 base,
      Int32 channel,
      Int32 range,
      Int32 mode)
```

Include file

`ds2211.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 Specifies the PHS-bus base address of the DS2211 board.

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_PWMIN_CH1	PWM channel 1
…	…
DS2211_PWMIN_CH24	PWM channel 24

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
DS2211_TIMING_RANGE1	9.54 Hz	100 kHz	50 ns
DS2211_TIMING_RANGE2	4.77 Hz	100 kHz	100 ns
DS2211_TIMING_RANGE3	2.39 Hz	100 kHz	200 ns
DS2211_TIMING_RANGE4	1.20 Hz	100 kHz	400 ns
DS2211_TIMING_RANGE5	0.60 Hz	100 kHz	800 ns
DS2211_TIMING_RANGE6	0.30 Hz	100 kHz	1.6 µs
DS2211_TIMING_RANGE7	0.15 Hz	100 kHz	3.2 µs
DS2211_TIMING_RANGE8	75 mHz	78.12 kHz	6.4 µs
DS2211_TIMING_RANGE9	38 mHz	39.06 kHz	12.8 µs
DS2211_TIMING_RANGE10	19 mHz	19.53 kHz	25.6 µs

Predefined Symbol	Frequency Mode		
	Minimum	Maximum	Resolution
DS2211_TIMING_RANGE11	10 mHz	9.76 kHz	51.2 µs
DS2211_TIMING_RANGE12	5.0 mHz	4.88 kHz	103 µs
DS2211_TIMING_RANGE13	2.5 mHz	2.44 kHz	205 µs
DS2211_TIMING_RANGE14	1.2 mHz	1.22 kHz	410 µs
DS2211_TIMING_RANGE15	0.6 mHz	610.35 Hz	820 µs
DS2211_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		
DS2211_TIMING_RANGE1	100 ns	10 µs	3.27 ms	50 ns
DS2211_TIMING_RANGE2	200 ns	10 µs	6.55 ms	100 ns
DS2211_TIMING_RANGE3	400 ns	10 µs	13.1 ms	200 ns
DS2211_TIMING_RANGE4	800 ns	10 µs	26.2 ms	400 ns
DS2211_TIMING_RANGE5	1.6 µs	10 µs	52.4 ms	800 ns
DS2211_TIMING_RANGE6	3.2 µs	10 µs	104 ms	1.6 µs
DS2211_TIMING_RANGE7	6.4 µs	10 µs	209 ms	3.2 µs
DS2211_TIMING_RANGE8	12.8 µs	12.8 µs	419 ms	6.4 µs
DS2211_TIMING_RANGE9	25.6 µs	25.6 µs	838 ms	12.8 µs
DS2211_TIMING_RANGE10	51.2 µs	51.2 µs	1.67 s	25.6 µs
DS2211_TIMING_RANGE11	103 µs	103 µs	3.35 s	51.2 µs
DS2211_TIMING_RANGE12	205 µs	205 µs	6.71 s	103 µs
DS2211_TIMING_RANGE13	410 µs	410 µs	13.4 s	205 µs
DS2211_TIMING_RANGE14	820 µs	820 µs	26.8 s	410 µs
DS2211_TIMING_RANGE15	1.64 ms	1.64 ms	53.6 s	820 µs
DS2211_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 Specifies the mode of the timing measurement unit. The following modes are available:

Mode	Meaning
DS2211_PWM2D	PWM signal measurement
DS2211_PWM2D_SYNCH_UPDATE	PWM signal measurement with synchronous update
DS2211_F2D	Frequency measurement

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of PWM Signal Generation.. 80

References

Timing Mode... 74

PWM Signal Generation

Introduction

You can generate signals for pulse width modulation (PWM).

Where to go from here

Information in this section

Information in other sections

PWM Signal Generation (DS2211 Features 🕮)
9 independent PWM outputs are available for the generation of square-wave signals with a run-time adjustable frequency and run-time adjustable duty cycle.

Example of PWM Signal Generation

Example

This example generates a PWM signal with a period of 1 ms and a duty cycle of 25% on PWM channel 1.

```
#include <Brtenv.h>              /* basic real-time environment */
#include <Ds2211.h>
/*------------------------------------------------------------*/
#define DT 1e-3                    /* 1 ms simulation step size */
dsfloat duty   = 0.25;                 /* set duty cycle to 25% */
dsfloat period = 0.001;            /* set output period to 1 ms */
/* variables for execution time profiling */
dsfloat exec_time;                             /* execution time */
/*------------------------------------------------------------*/
void isr_t1()                 /* timer1 interrupt service routine */
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
   /* update period or duty-cycle */
   ds2211_pwm_out(DS2211_1_BASE, DS2211_PWMOUT_CH1, period, duty);
   exec_time = RTLIB_TIC_READ();      /* calculate execution time */
   RTLIB_SRT_ISR_END();       /* end of interrupt service routine */
}
/*---------------------------------------------------------------*/
void main()
{
   init();                      /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);   /* initialize DS2211 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* enable digital output driver */
   ds2211_digout_mode_set(DS2211_1_BASE,
                            DS2211_DIGOUT_ENABLE);
   /* set PWM range for output channel 1 */
   ds2211_timing_out_mode_set(DS2211_1_BASE,
                  DS2211_PWMOUT_CH1,
                  DS2211_TIMING_RANGE5,
                  DS2211_D2PWM);
   /* set values for PWM Signal Generation */
   ds2211_pwm_out(DS2211_1_BASE, DS2211_PWMOUT_CH1,
                            period, duty);
   /* initialize sampling clock timer */
   RTLIB_SRT_START(DT, isr_t1);
   RTLIB_TIC_INIT();
   while(1)                                   /* background process */
   {
      RTLIB_BACKGROUND_SERVICE();
   } /* while(1) */
} /* main() */
```

Related topics

References

ds2211_pwm_out

Syntax

```
void ds2211_pwm_out(
      Int32 base,
      Int32 channel,
      dsfloat period,
      dsfloat duty)
```

Include file

`ds2211.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 (DS2211 Features 📖).

> **Note**
>
> - After initialization, the outputs are disabled. Use `ds2211_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 (DS2211 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_PWMOUT_CH1	PWM channel 1
…	…
DS2211_PWMOUT_CH9	PWM channel 9

period PWM period in seconds. Values should remain within the selected period range (refer to ds2211_timing_out_mode_set on page 74). For information on PWM signal generation and its restrictions, refer to PWM Signal Generation (DS2211 Features 📖).

duty PWM duty cycle within the range of 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 551.

Related topics

Examples

Example of PWM Signal Generation..80

References

ds2211_digout_mode_set..24

PWM Signal Measurement

Introduction

You can measure signals for pulse width modulation (PWM).

Where to go from here

Information in this section

Information in other sections

PWM Signal Measurement (DS2211 Features 📖)
PWM signal measurement is used to capture digital signals in hardware-in-the-loop applications.

Example of PWM Signal Measurement

Example

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

```
#include <Brtenv.h>                /* basic real-time environment */
#include <ds2211.h>
/*--------------------------------------------------------------*/
#define DT 1e-3                     /* 1 ms simulation step size */
dsfloat in_duty;                          /* measured duty cycle */
dsfloat in_period;                            /* measured period */
/* variables for execution time profiling                        */
dsfloat exec_time;                             /* execution time */
void isr_t1()                  /* timer1 interrupt service routine */
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
   /* read PWM input channel 1 */
   ds2211_pwm_in(DS2211_1_BASE, DS2211_PWMIN_CH1,
                  &in_period, &in_duty);
   exec_time = RTLIB_TIC_READ();     /* calculate execution time */
   RTLIB_SRT_ISR_END();      /* end of interrupt service routine */
}
/*--------------------------------------------------------------*/
void main()
{
   init();                           /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);        /* initialize DS2211 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   ds2211_digout_mode_set(DS2211_1_BASE, DS2211_DIGOUT_ENABLE);
   ds2211_timing_in_mode_set(DS2211_1_BASE,
                  DS2211_PWMIN_CH1,
                  DS2211_TIMING_RANGE5,
                  DS2211_PWM2D);
   /* initialize sampling clock timer */
   RTLIB_SRT_START(DT, isr_t1);
   RTLIB_TIC_INIT();
   while(1)                                   /* background process */
   {
      RTLIB_BACKGROUND_SERVICE();
   } /* while(1) */
} /* main() */
```

Related topics

References

ds2211_pwm_out.. 81

ds2211_pwm_in

Syntax

```
void ds2211_pwm_in(
      Int32 base,
      Int32 channel,
      dsfloat *period,
      dsfloat *duty)
```

Include file

`ds2211.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 (DS2211 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 (DS2211 Features 📖).
- The PWM input channels 7 and 8 are shared with injection capture. If you use these channels for injection capture, you cannot use it for PWM signal measurement, refer to ds2211_injection_capture_read on page 183.
- The PWM input channels 9 … 24 are shared with DIG_IN1 … 16.
- The PWM input channels 1 … 8 are shared with DIG_IN17 … 24.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel PWM input channel. The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_PWMIN_CH1 … DS2211_PWMIN_CH24	PWM input channel 1 … PWM input channel 24

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

duty Specifies the address where the measured duty cycle is written. The duty cycle is scaled to the range of 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 551.

Related topics

Examples

Example of PWM Signal Measurement..84

References

ds2211_digin_threshold_set...22

Square-Wave Signal Generation

Introduction

You can generate square-wave signals.

Where to go from here

Information in this section

Information in other sections

Square-Wave Signal Generation (D2F) (DS2211 Features 📖)
9 independent channels are available to generate square-wave signals with variable frequencies.

Example of Square-Wave Signal Generation

Example

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

```
#include <Brtenv.h> /* basic real-time environment */
#include <Ds2211.h>
/*--------------------------------------------------------------*/
#define DT 1e-3                                 /* 1 ms simulation step size */
dsfloat frequency = 1000;                    /* set output frequency to 1kHz  */
                                    /* variables for execution time profiling */
dsfloat exec_time;                                          /* execution time */
/*--------------------------------------------------------------*/
void isr_t1()                             /* timer1 interrupt service routine */
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
```

```
                                             /* update output frequency */
   ds2211_d2f(DS2211_1_BASE, DS2211_PWMOUT_CH1, frequency);
   exec_time = RTLIB_TIC_READ();                   /* calculate execution time */
   RTLIB_SRT_ISR_END();                    /* end of interrupt service routine */
}
/*--------------------------------------------------------------*/
void main()
{
   init();                                      /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);                     /* initialize DS2211 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
                                              /* enable digital output driver */
   ds2211_digout_mode_set(DS2211_1_BASE,
                          DS2211_DIGOUT_ENABLE);
               /* set frequency range for output channel 1, output(0Hz) = low */
   ds2211_timing_out_mode_set(DS2211_1_BASE,
                              DS2211_PWMOUT_CH1,
                              DS2211_TIMING_RANGE5,
                              DS2211_D2F_LOW);
                                      /* set values for PWM Signal Generation */
   ds2211_d2f(DS2211_1_BASE, DS2211_PWMOUT_CH1, frequency);
                                           /* initialize sampling clock timer */
   RTLIB_SRT_START(DT, isr_t1);
   RTLIB_TIC_INIT();
   while(1)                                             /* background process */
   {
   RTLIB_BACKGROUND_SERVICE();
   }                                           /* while(1) */
}                                              /* main() */
```

Related topics

References

ds2211_d2f 89
ds2211_digout_mode_set 24
ds2211_init 20
ds2211_timing_out_mode_set 74

ds2211_d2f

Syntax

```
void ds2211_d2f(
      Int32 base,
      Int32 channel,
      dsfloat frequency)
```

Include file

`ds2211.h`

Purpose

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

I/O mapping

For information on the I/O mapping, refer to Frequency Measurement (F2D) (DS2211 Features 🕮) and Square-Wave Signal Generation (D2F) (DS2211 Features 🕮).

Note

- All digital outputs are high-Z after reset. Outputs are enabled using the `ds2211_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 Frequency Measurement (F2D) (DS2211 Features 🕮) and Square-Wave Signal Generation (D2F) (DS2211 Features 🕮).

Description

The function outputs a digital signal with the specified frequency on the appropriate output channel. The resolution of the frequency signal is 20 bit and depends on the selected prescaler setting. For information on the available range, refer to Frequency Measurement (F2D) (DS2211 Features 🕮) and Square-Wave Signal Generation (D2F) (DS2211 Features 🕮).

The frequency ranges can be set using the `ds2211_timing_out_mode_set` function.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_PWMOUT_CH1	PWM channel 1
...	...
DS2211_PWMOUT_CH9	PWM channel 9

frequency Frequency of the generated signal in Hz

Related topics

Examples

Example of Square-Wave Signal Generation..88

References

ds2211_digout_mode_set..24
ds2211_timing_out_mode_set..74

Frequency Measurement

Introduction

You can measure the frequency of a square-wave signal.

Where to go from here

Information in this section

Information in other sections

Frequency Measurement (F2D) (DS2211 Features 🕮)
24 independent channels are available to measure the frequency of square-wave signals.

Square-Wave Signal Generation (D2F) (DS2211 Features 🕮)
9 independent channels are available to generate square-wave signals with variable frequencies.

Example of Frequency Measurement

Example

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

```
#include <Brtenv.h>                            /* basic real-time environment */
#include <Ds2211.h>
/*--------------------------------------------------------------*/
#define DT 1e-3                                  /* 1 ms simulation step size */
dsfloat frequency;                                         /* input frequency */
/* variables for execution time profiling */
dsfloat exec_time;                                          /* execution time */
/*--------------------------------------------------------------*/
void isr_t1()                             /* timer1 interrupt service routine */
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
```

```
                                        /* read signal frequency from input 1 */
   ds2211_f2d(DS2211_1_BASE, DS2211_PWMOUT_CH1, &frequency);
   exec_time = RTLIB_TIC_READ();                  /* calculate execution time */
   RTLIB_SRT_ISR_END();                   /* end of interrupt service routine */
}
/*--------------------------------------------------------------*/
void main()
{
   init();                                      /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);                     /* initialize DS2211 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
                                          /* enable digital output driver */
   ds2211_digout_mode_set(DS2211_1_BASE,
                          DS2211_DIGOUT_ENABLE);
                                   /* set frequency range for input channel 1 */
   ds2211_timing_in_mode_set(DS2211_1_BASE,
                             DS2211_PWMIN_CH1,
                             DS2211_TIMING_RANGE5,
                             DS2211_F2D);
                                           /* initialize sampling clock timer */
   RTLIB_SRT_START(DT, isr_t1);
   RTLIB_TIC_INIT();
   while(1)                                             /* background process */
{
   RTLIB_BACKGROUND_SERVICE();
   }                                       /* while(1) */
}                                           /* main() */
```

Related topics

References

ds2211_f2d

Syntax

```
void ds2211_f2d(
      Int32 base,
      Int32 channel,
      dsfloat* frequency)
```

Include file

`ds2211.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) (DS2211 Features) and Square-Wave Signal Generation (D2F) (DS2211 Features).

Note

- 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) (DS2211 Features) and Square-Wave Signal Generation (D2F) (DS2211 Features).
- The PWM input channels 1 ... 8 are shared with DIG_IN17 ... 24.

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) (DS2211 Features) and Square-Wave Signal Generation (D2F) (DS2211 Features).

The frequency ranges can be set using the `ds2211_timing_in_mode_set` function.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel PWM input channel. The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_PWMIN_CH1 ... DS2211_PWMIN_CH24	PWM input channel 1 ... PWM input channel 24

frequency Specifies the address where frequency is written in Hz

Related topics

Examples

Example of Frequency Measurement.. 91

References

ds2211_timing_in_mode_set.. 77

Angular Processing Unit (APU)

Introduction

The angular processing unit (APU) is designed to simulate core engine processing functions.

Where to go from here

Information in this section

Information in other sections

Angular Processing Unit (DS2211 Features 🕮)
Provides general information on the angular processing unit, which is designed to simulate core engine processing functions.

APU Demo Application

Where to go from here

Information in this section

Description of the APU Demo Application

Introduction

This application demonstrates the use of the angular processing unit (APU).

Demo files

The demo files are located in `<RCP_HIL_InstallationPath>\Demos\Ds100x\IOBoards\Ds2211\APU\APU_demo`.

- For the code of the demo application, refer to the `Apu_2211_hc.c` file.
- For information on how to run the demo application, see How to Run the Demo Application on page 97.

Description

The following features are simulated:

- Crankshaft and camshaft signal generation
- Engine control
  - Starts and stops the APU
  - Shows how to adjust the engine speed
- Angular position recording
- Cylinder interrupt use
- Spark event capture and injection measurement of up to 6 cylinders

The demo program uses the camshaft B wave table number 1 to generate an example pulse pattern, which is used for the spark and injection event capture units.

Preparation

The following pins have to be connected:

Output	Connector, Pin	Input	Connector, Pin
CRANK_DIG	P2, Pin 45	ADC3	P1, Pin 69
CAM1_DIG	P2, Pin 47	ADC2	P1, Pin 66
CAM2_DIG	P2, Pin 49	IGN1 … 6, INJ1 … 6	P2, Pin 17 … 28
		ADC1	P1, Pin 65
+12 V	P1, Pin 91, 92	VBAT1	P2, Pin 29, 30

Connect $\overline{\text{ADC1}}$ (P1, Pin 67), $\overline{\text{ADC2}}$ (P1, Pin 68) and $\overline{\text{ADC3}}$ (P1, Pin 71) with GND.

Related topics

HowTos

How to Run the Demo Application.. 97

How to Run the Demo Application

Objective

This application demonstrates the use of the angular processing unit (APU). The demo files are located in `<RCP_HIL_InstallationPath>\Demos\Ds100x\IOBoards\Ds2211\APU\APU_demo`.

Method 1

To run the demo application

1. Start MATLAB and change to the demo folder (optional).
2. To plot the exemplary wave tables and to generate the MAT file `wav2211.mat` (optional), enter `wav2211`.
3. On the Windows Start menu, select dSPACE RCP and HIL 20xx-x – Command Prompt for dSPACE RCP and HIL 20xx-x to open a Command Prompt window in which the required paths and environment settings are preset.
4. In the Command Prompt window, change to the demo folder and convert the MAT file to a usable C file by entering `matconv wav2211` (optional).
5. To compile and download the demo application, enter the following command.

   ```
   down<xxxx> Apu_2211_hc wav2211
   ```

   `down<xxxx>` must correspond to the processor board type, for example, `down1006` for a DS1006.

Method 2

To run the demo application using the build command

1 On the Windows Start menu, select dSPACE RCP and HIL 20xx-x – Command Prompt for dSPACE RCP and HIL 20xx-x to open a Command Prompt window in which the required paths and environment settings are preset.

2 Change to the demo directory.

3 Enter `build`.

Related topics

Examples

Overall APU Functions

Where to go from here

Information in this section

Information in other sections

Angular Processing Unit (DS2211 Features 📖)
Provides general information on the angular processing unit, which is designed to simulate core engine processing functions.

ds2211_apu_position_write

Syntax

```
void ds2211_apu_position_write(
      Int32 base,
      dsfloat pos)
```

Include file

`ds2211.h`

Purpose

To set the initial engine position.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2211 Features 📖).

Description

The function checks the APU status and sets the specified engine position only if the APU is stopped. The function has no effect on boards configured as slaves.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

pos Engine position stated in radians (rad) in the range 0 … 4π.

Return value

None

Error message

The following error message is defined:

Type	Error Message	Description
Error	ds2211_apu_position_write (board_offset): No access while APU is running!	You have to stop the APU with ds2211_apu_stop before writing the APU position.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_apu_position_read.. 100
ds2211_apu_stop.. 107

ds2211_apu_position_read

Syntax

```
void ds2211_apu_position_read(
      Int32 base,
      dsfloat *pos)
```

Include file

`ds2211.h`

Purpose

To read the current engine position.

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2211 Features 🕮).

Description

For cascaded DS2211 boards, this function returns the angle position of the master board.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

pos Address where the engine position is written. The value is stated in radians (rad) in the range 0 … 4π.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_apu_position_write..99

ds2211_apu_velocity_write

Syntax

```
void ds2211_apu_velocity_write(
      Int32 base,
      dsfloat vel)
```

Include file

`ds2211.h`

Purpose

To update the APU angle velocity.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2211 Features 📖).

Description

The function has no effect on boards configured as slaves.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

vel Angle velocity in rad/s in the range –3068 … +3068. The parameter is saturated to its limits.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Examples of Camshaft Signal Generation	118
Examples of Crankshaft Signal Generation	108

References

DS2211_RAD_S	43

ds2211_apu_master_detect

Syntax

```
Int32 ds2211_apu_master_detect(int32 base)
```

Input file

`ds2211.h`

Module

`ds2211_apu`

Purpose

To search for the time-base master connected to the APU bus.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2211 Features 🕮).

Description

`ds2211_apu_master_detect` detects a board connected to the APU bus, which is initialized as the time-base master. Boards which can be connected to the APU bus and support master detection are:

- DS2211
- DS4002
- DS5001

If DS2211 and DS2210 boards are connected to the same engine position bus, a DS2210 must be the time-base master. For details, refer to Working with a Real-Time System Containing a DS2210 and a DS2211 (DS2211 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

Return value

The following values are returned:

Predefined Symbol	Value	Description
DS2211_NO_MASTER_FOUND	0	No time-base master was detected
DS2211_MASTER_FOUND	1	A time-base master was detected

Related topics

Basics

Working with a Real-Time System Containing a DS2210 and a DS2211 (DS2211 Features 🕮)

References

Crankshaft Sensor Signal Generation (DS2211 Features 🕮)

ds2211_int_position_set

Syntax

```
void ds2211_int_position_set(
      Int32 base,
      Int32 channel,
      Int32 count,
      dsfloat* pos)
```

Include file

`ds2211.h`

Purpose

To define interrupt positions for the specified cylinder.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2211 Features 🕮).

Description

`ds2211_int_position_set` checks the APU status and sets the interrupt positions for the specified cylinder if the APU is stopped. Each time one of the positions is reached, the corresponding interrupt is generated.

Note

- The function is non-reentrant.
- The interrupt position has a fixed offset error.

  If a DS2211 is the APU master, the offset error is 0.022° in forward engine rotation direction and 0.033° in reverse engine rotation direction.

  If a DS2210 is the APU master, the offset error is 0.088° in forward engine rotation direction and 0.099° in reverse engine rotation direction.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Cylinder number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_INTPOS_CYL1	Cylinder 1
…	…
DS2211_INTPOS_CYL6	Cylinder 6

count Number of interrupt positions to be specified. Up to 2048 interrupts are possible.

pos Pointer to an array of interrupt positions. The values have to be stated in rad in the range 0 … 4π. The resolution is 0.011° or 0.0012 radians (rad). Two subsequent interrupts have to be set with a distance of 0.0031 radians (rad).

Return value

None

Error message

The following error messages are defined:

Type	Error Message	Description
Error	ds2211_int_position_set (board_offset): No access while APU is running!	You have to stop the APU with ds2211_apu_stop before setting the interrupt positions.
Error	ds2211_int_position_set (board_offset): Memory access error!	The function failed due to a memory access error.

Execution times

For information, refer to Function Execution Times on page 551.

Example

```
...
dsfloat positions[5] = {DS2211_RAD(0), DS2211_RAD(90),
                        DS2211_RAD(180), DS2211_RAD(270),
                        DS2211_RAD(360)};
...
/* set interrupt positions for cylinder 1 */
ds2211_int_position_set(DS2211_1_BASE,
                        DS2211_INTPOS_CYL1,
                        5,
                        positions);
...
```

Related topics

References

ds2211_apu_stop.. 107

Engine Position Phase Accumulator

Where to go from here

Information in this section

Information in other sections

Engine Position Phase Accumulator (DS2211 Features 📖)
Explaining the unit that supplies the engine position.

ds2211_apu_start

Syntax

```
void ds2211_apu_start(Int32 base)
```

Include file

`ds2211.h`

Purpose

To start engine phase accumulation and APU signal generation.

I/O mapping

For information on the I/O mapping, refer to APU Reference (DS2211 Features 📖).

> **Note**
>
> For cascaded DS2210 or DS2211 boards, you have to start the slave APU(s) first.

Parameters

base	Specifies the PHS-bus base address of the DS2211 board.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Examples of Camshaft Signal Generation 118
Examples of Crankshaft Signal Generation 108

References

ds2211_apu_stop 107

ds2211_apu_stop

Syntax

```
void ds2211_apu_stop(Int32 base)
```

Include file

`ds2211.h`

Purpose

To stop the APU signal generation.

I/O mapping

For information on the I/O mapping, refer to APU Reference (DS2211 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_apu_start 106

Crankshaft Sensor Signal Generation

Introduction

To generate crankshaft sensor signals.

Where to go from here

Information in this section

Information in other sections

Crankshaft Sensor Signal Generation (DS2211 Features 📖)
The crankshaft signal generator has one analog and one digital crankshaft output.

Examples of Crankshaft Signal Generation

Loading wave table data manually

This example shows how to load wave table data manually without using the MATCONV tool, set the amplitude and the velocity, and start signal generation.

Note

Before you can use the crankshaft signal generator, you have to initialize the board (see ds2211_init on page 20) and enable the output transformers (see ds2211_apu_transformer_mode_set on page 39).

```
#define TBLLEN 65536
```

```
Int32 cr_tbl[TBLLEN];
...
/* initialize wave table data */
for (i=0; i<TBLLEN; i++)
{
cr_tbl[i] = ...
}
/* load data to crankshaft wave table 1 */
ds2211_crank_table_load(DS2211_1_BASE, DS2211_CRANK_TBL1, cr_tbl);
/* select crankshaft wave table 1 for signal generation */
ds2211_crank_table_select(DS2211_1_BASE, DS2211_CRANK_TBL1);
/* set amplitude of crankshaft signal to +/- 10 V */
ds2211_crank_output_ampl_set(DS2211_1_BASE, 20.0);
/* set engine velocity to 1000 rpm */
ds2211_apu_velocity_write(DS2211_1_BASE, DS2211_RAD_S(1000));
/* start signal generation */
ds2211_apu_start(DS2211_1_BASE);
...
```

Using the MATCONV tool

The wave tables used in the APU demo application (refer to APU Demo Application on page 96) are generated with MATLAB and the MATCONV tool. The following code lines are taken from the `Apu_2211_hc.c` file located in `<RCP_HIL_InstallationPath>\Demos\Ds100x\IOBoards\Ds2211\APU\APU_demo`.

Declare the global labels which allow you to access the wave table data

```
...
/* wav2211.c */
extern UInt32   wav2211_1_crank1;
...
```

Load wave table data using the global labels

```
...
/* initialize crankshaft signal generation */
ds2211_crank_table_load(DS2211_1_BASE, DS2211_CRANK_TBL1,
                  (Int32*)&wav2211_1_crank1);
...
```

Simulating reverse crank sensor signals

This example shows how to set the parameters for a reverse crank sensor signal, load and select an appropriate wave table, set the crankshaft mode, and enable the digital crankshaft output.

```
...
/* Set values: td=5µs, tf=45µs, tr=90µs, active low pulses */
ds2211_reverse_crank_setup(DS2211_1_BASE,
                            5.0e-6, 45.0e-6, 90.0e-6, 5.0e-6,
                            DS2211_POLARITY_ACTIVE_LOW);
/* Load wave table for reverse crankshaft */
ds2211_crank_table_load(DS2211_1_BASE, DS2211_CRANK_TBL1, cr_tbl);
/* Select wave table */
ds2211_crank_table_select(DS2211_1_BASE, DS2211_CRANK_TBL1);
/* Set crankshaft mode to reverse */
ds2211_crank_mode(DS2211_1_BASE, DS2211_CRANK_MODE_REVERSE);
/* Enable digital crankshaft output */
ds2211_digout_mode_set(DS2211_1_BASE, DS2211_DIGOUT_ENABLE);
```

```
...
```

ds2211_crank_table_load

Syntax

```
void ds2211_crank_table_load(
      Int32 base,
      Int32 table,
      Int32* data)
```

Include file

`ds2211.h`

Purpose

To load wave table data for crankshaft sensor signal generation.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2211 Features 🕮).

Description

This function checks the APU status and loads wave table data to one of eight crankshaft wave tables in the DS2211 memory, but only if the APU is stopped. Compressed wave tables are decompressed automatically. Wave form data is 12-bit signed, i.e., 0x800 … 0x7FF for –20 V … +20 V at transformer outputs. Digital wave form outputs are controlled by MSBs of the wave form data. You can load up to 8 wave tables and switch from one table to another and the APU is running.

The following table shows the relationship between wave table data values and output signals. Note that the internal representation on the hardware uses an inverted sign bit.

Value (Code)	Internal	Analog Output	Value	Digital Output
–2048 … –1	0x000 … 0x7FF	–20 V … –0.0097 V	–2048 … 0	0 (low)
0 … 2047	0x800 … 0xFFF	0 V … +20 V	1 … 2047	1 (high)

Note

The values for the analog output in the table above apply to the maximum amplitude of 40 V_{pp}. The analog output covers the range $+V_{max}$ … $–V_{max}$.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

table Crankshaft wave table number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CRANK_TBL1	Crankshaft wave table 1
…	…
DS2211_CRANK_TBL8	Crankshaft wave table 8

data Source address of the crankshaft wave table data

Return value

None

Message

The following messages are defined:

Type	Message	Description
Error	ds2211_crank_table_load (board_offset): No access while APU is running!	You have to stop the APU with `ds2211_apu_stop`.
Error	ds2211_crank_table_load (board_offset): Memory access error!	The function failed due to a memory access error.
Error	ds2211_crank_table_load (board_offset): Wave table decompression failed.	An error occurred during decompression of a compressed wave table.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Examples of Crankshaft Signal Generation 108

References

ds2211_apu_start 106
ds2211_apu_stop 107
ds2211_crank_mode 116
ds2211_crank_table_select 112

ds2211_crank_table_select

Syntax

```
void ds2211_crank_table_select(
      Int32 base,
      Int32 table)
```

Include file

`ds2211.h`

Purpose

To select a wave table for the crankshaft sensor signal generation.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2211 Features 🕮).

> **Note**
>
> After initialization, the analog transformer outputs are disabled. Use `ds2211_apu_transformer_mode_set` to enable the transformers. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 🕮).
> After initialization, the digital crankshaft output is disabled. Use `ds2211_digout_mode_set` to enable the output. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

table Crankshaft wave table number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CRANK_TBL1 ... DS2211_CRANK_TBL8	Crankshaft wave table 1 ... Crankshaft wave table 8

The parameter is saturated to its limits.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Examples of Crankshaft Signal Generation 108

References

ds2211_apu_start 106
ds2211_apu_transformer_mode_set 39
ds2211_crank_table_load 110
ds2211_digout_mode_set 24
ds2211_reverse_crank_setup 114

ds2211_crank_output_ampl_set

Syntax

```
void ds2211_crank_output_ampl_set(
      Int32 base,
      dsfloat value)
```

Include file

`ds2211.h`

Purpose

To set the amplitude of the crankshaft output signal.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2211 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

value Amplitude in the range 0 … 40 V_{pp}. Set the amplitude to 0 V_{pp} when using digital wave form outputs.

Return value

None

> **Note**
>
> After initialization, the analog transformer outputs are disabled. Use `ds2211_apu_transformer_mode_set` to enable the transformers. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 🕮).

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Examples of Crankshaft Signal Generation.. 108

References

ds2211_apu_start.. 106
ds2211_apu_transformer_mode_set.. 39
ds2211_crank_table_load.. 110

ds2211_reverse_crank_setup

Syntax

```
void ds2211_reverse_crank_setup(
      Int32 base,
      dsfloat td,
      dsfloat tf,
      dsfloat tr,
      dsfloat tv,
      UInt32 polarity)
```

Include file

`ds2211.h`

Purpose

To specify the settings for reverse crankshaft sensor signal generation.

> **Note**
>
> Reverse crankshaft rotation is not supported by all board revisions. For details, refer to Reverse Crankshaft Rotation (DS2211 Features 📖).

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2211 Features 📖).

Description

This function specifies the settings for the simulation of a reverse crank sensor signal. The signal is generated on the digital crankshaft output (CRANK_DIG). The sensor pulses are generated with the specified pulse width (tf, tr), time delay (td, tv), and polarity (active high or active low). A wave table for reverse crankshaft signals must be created, loaded, and selected. In the selected wave table, reverse crankshaft pulses are triggered by 0–1 or 1–0 transitions. The default values of the parameters are used if `ds2211_reverse_crank_setup` is not called before switching to reverse mode with the `ds2211_crank_mode` function.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

td Specifies the time between a trigger event of a timing wheel tooth and the beginning of the corresponding sensor pulse in the range 1 … 8191.75 µs with a resolution of 0.25 µs (the default value is 5 µs). The specified time is also used as the forced minimum inactive time between two active pulses of the same rotation direction. The value is saturated to minimum/maximum value if the range is exceeded.

tf Specifies the pulse duration that indicates a forward rotation of the crankshaft in the range 1 … 8191.75 µs with a resolution of 0.25 µs (the default value is 45 µs). The value is saturated to minimum/maximum value if the range is exceeded.

tr Specifies the pulse duration that indicates a reverse rotation of the crankshaft in the range 1 … 8191.75 µs with a resolution of 0.25 µs (the default value is 90 µs). The value is saturated to minimum/maximum value if the range is exceeded.

tv Specifies the forced minimum inactive time between two pulses of different rotation directions in the range 1 … 8191.75 µs with a resolution of 0.25 µs (the default value is 5 µs). The value is saturated to minimum/maximum value if the range is exceeded.

To ensure tv functions correctly the following condition must be met:

max((tr + tv) , (tf + tv)) < model update period

polarity Specifies the polarity of the crank sensor pulses. The following symbols are predefined:

Predefined Symbol	Description
DS2211_POLARITY_ACTIVE_LOW	Crank sensor pulses are active low (default)
DS2211_POLARITY_ACTIVE_HIGH	Crank sensor pulses are active high

Return value

None

Message

The following messages are defined:

Type	Message	Description
Error	ds2211_reverse_crank_setup (board_offset): Feature is not supported by DS2211 Board/PAL revision.	See the hardware requirements in Reverse Crankshaft Rotation (DS2211 Features 📖).

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

Reverse Crankshaft Rotation (DS2211 Features 📖)

Examples

Examples of Crankshaft Signal Generation.. 108

References

ds2211_crank_mode.. 116
ds2211_crank_table_load.. 110
ds2211_crank_table_select.. 112
ds2211_digout_mode_set.. 24

ds2211_crank_mode

Syntax

```
void ds2211_crank_mode(
      Int32 base,
      Int32 mode)
```

Include file

`ds2211.h`

Purpose

To set the normal or reverse mode of crankshaft signal generation.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2211 Features 📖).

Description

The mode of the crankshaft signal generation affects the digital crankshaft output. It can be set to:

Modus	Description
Normal mode	Normal crankshaft signal generation on the digital (CRANK_DIG) and analog crankshaft output. A "normal" crankshaft wave table must be selected.
Reverse mode	Reverse crankshaft signal generation on the digital crankshaft output (CRANK_DIG). The analog crankshaft output is not automatically disabled and might generate an irrelevant analog signal. A wave table must be created, loaded, and selected. In the selected wave table, reverse crankshaft pulses are

Modus	Description
	triggered by 0–1 or 1–0 transitions. The setup parameters for the reverse crankshaft signal are set by ds2211_reverse_crank_setup on page 114. The default values of the parameters are used, when `ds2211_reverse_crank_setup` is not called before switching to the reverse mode. Reverse crankshaft signal generation is not supported by all board revisions. For details, refer to Reverse Crankshaft Rotation (DS2211 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

mode Mode of crankshaft signal generation. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CRANK_MODE_NORMAL	Normal mode (default)
DS2211_CRANK_MODE_REVERSE	Reverse mode

Return value

None

Message

The following messages are defined:

Type	Message	Description
Error	ds2211_crank_mode (board_offset): Feature is not supported by DS2211 Board/PAL revision.	See the hardware requirements in Reverse Crankshaft Rotation (DS2211 Features 📖).

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

Reverse Crankshaft Rotation (DS2211 Features 📖)

Examples

Examples of Crankshaft Signal Generation.. 108

References

ds2211_crank_table_load.. 110
ds2211_crank_table_select.. 112
ds2211_digout_mode_set.. 24
ds2211_reverse_crank_setup.. 114

Camshaft Sensor Signal Generation

Where to go from here

Information in this section

Information in other sections

Camshaft Sensor Signal Generation (DS2211 Features 📖)
Provides general information on the camshaft signal generator and its I/O mapping.

Examples of Camshaft Signal Generation

Loading wave table data manually

This example shows how to load wave table data manually without using the MATCONV tool, set the amplitude, the camshaft phase offsets and the engine velocity, and start the signal generation.

Note

Before you can use the camshaft signal generator, you have to initialize the board (refer to ds2211_init on page 20) and enable the output transformers (refer to ds2211_apu_transformer_mode_set on page 39).

```
#define TBLLEN 65536
Int32 cam_tbl[TBLLEN];
...
/* initialize wave table data */
for (i=0; i<TBLLEN; i++)
{
    cam_tbl[i] = ...
}
/* load data to wave table 1 for both camshafts */
ds2211_cam_table_load(DS2211_1_BASE, DS2211_CAM_CHA,
                     DS2211_CAM_TBL1, cam_tbl);
ds2211_cam_table_load(DS2211_1_BASE, DS2211_CAM_CHB,
                     DS2211_CAM_TBL1, cam_tbl);
/* select camshaft wave table 1 for both camshafts */
ds2211_cam_table_select(DS2211_1_BASE, DS2211_CAM_CHA,
                        DS2211_CAM_TBL1);
ds2211_cam_table_select(DS2211_1_BASE, DS2211_CAM_CHB,
                        DS2211_CAM_TBL1);
/* set camshaft amplitude of both camshafts to +/- 10 V */
ds2211_cam_output_ampl_set(DS2211_1_BASE, DS2211_CAM_CHA, 20.0);
ds2211_cam_output_ampl_set(DS2211_1_BASE, DS2211_CAM_CHB, 20.0);
/* set camshaft phase offsets */
ds2211_cam_phase_write(DS2211_1_BASE, DS2211_CAM_CHA, 0.15);
ds2211_cam_phase_write(DS2211_1_BASE, DS2211_CAM_CHB, 0.20);
/* set engine velocity to 1000 rpm */
ds2211_apu_velocity_write(DS2211_1_BASE, DS2211_RAD_S(1000));
/* start signal generation */
ds2211_apu_start(DS2211_1_BASE);
...
```

Reading phase offset

This example shows how to read the phase offset.

```
dsfloat phase1;
...
/* read phase offset of camshaft 1 */
ds2211_cam_phase_read(DS2211_1_BASE, DS2211_CAM_CHA, &phase1);
...
```

Using the MATCONV tool

The wave tables used in the APU demo application (refer to APU Demo Application on page 96) are generated with MATLAB and the MATCONV tool. The following code lines are taken from the `Apu_2211_hc.c` file located in `<RCP_HIL_InstallationPath>\Demos\Ds100x\IOBoards\Ds2211\APU\APU_demo`.

Declare the global labels which allow you to access the wave table data

```
...
/* wav2211.c */
...
extern UInt32   wav2211_1_camA1;
extern UInt32   wav2211_1_camB1;
...
```

Load wave table data using the global labels

```
...
/* initialize camshaft signal generation */
ds2211_cam_table_load(DS2211_1_BASE, DS2211_CAM_CHA,
                      DS2211_CAM_TBL1, (Int32*)&wav2211_1_camA1);
```

Related topics

References

APU Demo Application..96

ds2211_cam_table_load

Syntax

```
void ds2211_cam_table_load(
      Int32 base,
      Int32 channel,
      Int32 table,
      Int32 *data)
```

Include file

`ds2211.h`

Purpose

To load wave table data for camshaft sensor signal generation.

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2211 Features 📖).

Description

`ds2211_cam_table_load` checks the APU status and loads wave table data to one of eight camshaft wave tables in the DS2211 memory, but only if the APU is stopped. Compressed wave tables are decompressed automatically.

Note

Camshaft channels 3 and 4 have only digital outputs. They can therefore only represent the sign of the wave table data.

The following table shows the relationship between wave table data values and output signals. Note that the internal representation on the hardware uses an inverted sign bit.

Value (Code)	Internal	Analog Output	Value	Digital Output
–2048 … –1	0x000 … 0x7FF	–20 V … –0.0097 V	–2048 … 0	0 (low)
0 … 2047	0x800 … 0xFFF	0 V … +20 V	1 … 2047	1 (high)

Note

The values for the analog output in the table above apply for the maximum amplitude of 40 V_{pp}. The analog output covers the range +V_{max} … –V_{max}.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CAM_CHA	Camshaft 1
DS2211_CAM_CHB	Camshaft 2
DS2211_CAM_CHC	Camshaft 3
DS2211_CAM_CHD	Camshaft 4

table Camshaft wave table number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CAM_TBL1	Camshaft wave table 1
…	…
DS2211_CAM_TBL8	Camshaft wave table 8

data Source address of the wave table data

Return value

None

Message

The following messages are defined:

Type	Message	Description
Error	ds2211_cam_table_load (board_offset): No access while APU is running!	You have to stop the APU with `ds2211_apu_stop` before loading wave table data.
Error	ds2211_cam_table_load (board_offset): Memory access error!	The function failed due to a memory access error.
Error	ds2211_cam_table_load (board_offset): Wave table decompression failed.	An error occurred during decompression of a compressed wave table.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Examples of Camshaft Signal Generation... 118

References

ds2211_apu_start.. 106
ds2211_apu_stop.. 107
ds2211_cam_phase_write.. 126
ds2211_cam_table_select.. 122

ds2211_cam_table_select

Syntax

```
void ds2211_cam_table_select(
     Int32 base,
     Int32 channel,
     Int32 table)
```

Include file

`ds2211.h`

Purpose

To select a wave table for camshaft sensor signal generation.

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2211 Features 📖).

> **Note**
>
> After initialization, the analog transformer outputs and the digital camshaft are disabled. Use `ds2211_apu_transformer_mode_set` to enable the transformers. Use `ds2211_digout_mode_set` to enable the output. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 📖).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CAM_CHA	Camshaft 1
DS2211_CAM_CHB	Camshaft 2

Predefined Symbol	Description
DS2211_CAM_CHC	Camshaft 3
DS2211_CAM_CHD	Camshaft 4

table Camshaft wave table number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CAM_TBL1	Camshaft wave table 1
…	…
DS2211_CAM_TBL8	Camshaft wave table 8

The parameter is saturated to its limits.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

References

ds2211_cam_output_ampl_set

Syntax

```
void ds2211_cam_output_ampl_set(
      Int32 base,
      Int32 channel,
      dsfloat value)
```

Include file

`ds2211.h`

Purpose

To set the amplitude of the camshaft output signal.

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2211 Features 🕮).

Note

After initialization, the analog transformer outputs are disabled. Use `ds2211_apu_transformer_mode_set` to enable the transformers. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CAM_CHA	Camshaft 1
DS2211_CAM_CHB	Camshaft 2

value Amplitude in the range 0 … 40 V_{pp}. Set the amplitude to 0 V_{pp} when using digital wave form outputs. You can set the value with a resolution of 9.77 mV.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Examples of Camshaft Signal Generation..........118

References

ds2211_apu_start..........106
ds2211_apu_transformer_mode_set..........39
ds2211_cam_table_select..........122

ds2211_cam_phase_offset_update_mode

Syntax

```
void ds2211_cam_phase_offset_update_mode(
      Int32 base,
      Int32 channel,
      Int32 update_mode
)
```

Include file

`ds2211.h`

Purpose

To set the phase update mode to immediate or smooth updating.

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2211 Features 🕮).

Description

The camshaft phase offset can be updated immediately or smoothly. Smooth camshaft phase updating suppresses undesired spikes when the phase offset is updated. If you update the camshaft phase according to the direction of engine rotation (for example, 20° to 40° during forward rotation), the phase is updated with a maximum of double engine speed. If you update the camshaft phase in the opposite direction to engine rotation (for example, 40° to 20° during forward rotation), the phase is updated with a maximum equal to the engine speed. This ensures that the position of the camshaft does not move back to a pulse that was already processed. It also ensures that no unprocessed pulse is skipped. Smooth phase updating is not applied with offsets greater than 180° crankshaft or a nonrotating engine.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CAM_CHA	Camshaft 1
DS2211_CAM_CHB	Camshaft 2
DS2211_CAM_CHC	Camshaft 3
DS2211_CAM_CHD	Camshaft 4

update_mode Sets the mode of the camshaft phase update. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CAM_UPDATE_MODE_IMMEDIATE	Immediate update of the phase offset
DS2211_CAM_UPDATE_MODE_SMOOTH	Smooth update of the phase offset

Return value

None

Message

The following messages are defined:

Type	Message	Description	
Error	ds2211_cam_phase_offset_update_mode (board_offset): Feature is not supported by DS2211 Board/PAL revision.	One of the following DS2211 Board/FPGA revisions are required:	
		Board revision:	*FPGA revision:*
		02	004 or higher
		03	004 or higher
		04	002 … 009 or 12 or higher
		05 or higher	001 or higher

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_cam_phase_write

Syntax

```
void ds2211_cam_phase_write(
      Int32 base,
      Int32 channel,
      dsfloat phase)
```

Include file

`ds2211.h`

Purpose

To set the phase offset between the camshaft and crankshaft signal (camshaft phase).

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2211 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CAM_CHA	Camshaft 1
DS2211_CAM_CHB	Camshaft 2
DS2211_CAM_CHC	Camshaft 3
DS2211_CAM_CHD	Camshaft 4

phase Phase offset in rad in the range 0 … 4π.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Examples of Camshaft Signal Generation.. 118

References

ds2211_apu_start.. 106
ds2211_cam_phase_offset_update_mode.. 125
ds2211_cam_phase_read.. 127
ds2211_cam_table_load.. 120
ds2211_cam_table_select.. 122

ds2211_cam_phase_read

Syntax

```
void ds2211_cam_phase_read(
     Int32 base,
     Int32 channel,
     dsfloat *phase)
```

Include file

`ds2211.h`

Purpose

To read the current phase offset between the crankshaft and camshaft sensor signal (camshaft phase).

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2211 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_CAM_CHA	Camshaft 1
DS2211_CAM_CHB	Camshaft 2
DS2211_CAM_CHC	Camshaft 3
DS2211_CAM_CHD	Camshaft 4

phase Address where phase offset is written. The value is specified in rad in the range 0 … 4π.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Examples of Camshaft Signal Generation 118

References

ds2211_apu_start 106
ds2211_cam_phase_write 126
ds2211_cam_table_load 120

Event Capture Windows

Introduction

For spark event capture and injection pulse position and fuel amount measurement, event capture windows have to be defined. These windows allow you to specify the range for which events and pulses are captured.

Where to go from here

Information in this section

Information in other sections

ds2211_event_window_set

Syntax

```
void ds2211_event_window_set(
      Int32 base,
      Int32 channel,
      dsfloat start,
      dsfloat end)
```

Include file

`ds2211.h`

Purpose

To set an event window for capturing injection and ignition signals.

Note

This function is obsolete. It sets only one event window for capturing the signals. Use `ds2211_multi_eventwin_set` instead. This function can set one or two event windows.

Description

This function checks the APU status and if the APU has stopped, the event window data for the specified channel is loaded to DS2211 memory. Legal channel numbers comprise the ignition capture channels for cylinders 1 … 6, auxiliary capture channels 1 and 2, and the injection capture channels for cylinders 1 … 8. The capture window is defined by the specified start and end positions. The function exits with an error message if the APU is running or an error occurred during the load process. It is in principle possible to specify an event window which covers the entire position range 0 … 4π rad. However, if you do so, no window borders, and therefore no pseudo pulses, will be detected.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel to set an event window for (1 … 16).

Predefined Symbol	Description
DS2211_EVWIN_IGNCAP1	Ignition capture channel 1
…	…
DS2211_EVWIN_IGNCAP6	Ignition capture channel 6
DS2211_EVWIN_AUXCAP1	Auxiliary capture channel 1
DS2211_EVWIN_AUXCAP2	Auxiliary capture channel 2
DS2211_EVWIN_INJCAP1	Ignition capture channel 1
…	…
DS2211_EVWIN_INJCAP8	Ignition capture channel 8

start Sets the window start position [rad] (0 … 4π).

end Sets the window end position [rad] (0 … 4π).

Return value

None

Messages

The following messages are defined:

Type	Message	Description
Error	ds2211_event_window_set(0x%02lX): No access while APU is running! (Parameter: Board address offset)	You have to stop the APU before setting the event windows (see `ds2211_apu_stop` on page 107).
Error	ds2211_event_window_set(0x%02lX): Memory access error! (Parameter: Board address offset)	The function failed due to a memory access error.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_multi_eventwin_set

Syntax

```
void ds2211_multi_eventwin_set(
      Int32 base,
      Int32 channel,
      Int32 eventwin_count,
      dsfloat *start,
      dsfloat *end)
```

Include file

`ds2211.h`

Purpose

To set one or two event windows for capturing injection and ignition signals.

Description

This function checks the APU status and if the APU has stopped, the event window data for the specified channel is loaded to DS2211 memory. Legal channel numbers comprise the ignition capture channels for cylinders 1 … 6, auxiliary capture channels 1 and 2, and the injection capture channels for cylinders 1 … 8. You can define 1 or 2 event windows for a channel. The capture windows are defined by the specified start and end positions. The window borders are recognized by the DS2211. To be able to recognize each window border, the window borders must have a minimum distance. The minimum distance is 16 APU position steps (16 · 4 · π / 65536 rad = 0.003068 rad = 0.176 deg). Thus, using only one event window it must not exceed 4 ·π – 0.003068 rad = 12.563 rad = 719.824 deg.

It is in principle possible to specify event windows which covers the entire position range 0 … 4π rad. However, if you do so, no window borders, and therefore no pseudo pulses, will be detected. As a result the ignition and injection capture functions do not work correctly, because it is impossible to assign the measured values to the corresponding event window. The only functions which can handle event windows covering the entire position range are the functions to read the values continuously (see `ds2211_multiwin_ign_fifo_read` on page 162 and `ds2211_multiwin_inj_fifo_read` on page 195).

If start position and end position are equal, no event window is defined.

The start position can be greater than the end position to define an event window covering the transition of 720° to 0°.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel to set an event window for (1 … 16).

Predefined Symbol	Description	Value
DS2211_EVWIN_IGNCAP1	Ignition capture channel 1	1
…	…	…
DS2211_EVWIN_IGNCAP6	Ignition capture channel 6	6
DS2211_EVWIN_AUXCAP1	Auxiliary capture channel 1	7
DS2211_EVWIN_AUXCAP2	Auxiliary capture channel 2	8
DS2211_EVWIN_INJCAP1	Ignition capture channel 1	9
…	…	…
DS2211_EVWIN_INJCAP8	Ignition capture channel 8	16

eventwin_count Specifies the number of event windows to be set. Possible values are 1 or 2. Specify the same number of event windows within all related functions, otherwise the results are incompatible.

start Sets the window start position(s) [rad] (0 … 4π). It must be a pointer to an array of length `eventwin_count`.

end Sets the window end position(s) [rad] (0 … 4π). It must be a pointer to an array of length `eventwin_count`.

Return value

None

Messages

The following messages are defined:

Type	Message	Description
Error	ds2211_multi_eventwin_set(0x%02lX): No access while APU is running! (Parameter: Board address offset)	You have to stop the APU before setting the event windows (see `ds2211_apu_stop` on page 107).
Error	ds2211_multi_eventwin_set(0x%02lX): Memory access error! (Parameter: Board address offset)	The function failed due to a memory access error.
Error	ds2211_multi_eventwin_set(0x%02lX): Parameter event window count invalid! (Parameter: Board address offset)	The `eventwin_count` parameter exceeds the maximum. Allowed values are 1 or 2.

Execution times

For information, refer to Function Execution Times on page 551.

Example

The following example shows how to specify two event windows. The first event window starts at 100° and ends at 160°. The second event window starts at 200° and ends at 260°.

```
{
   Int32 evw_count = 2;
   dsfloat start_pos = {DS2211_RAD(100), DS2211_RAD(200)};
   dsfloat end_pos   = {DS2211_RAD(160), DS2211_RAD(260)};
   ...
   ds2211_multi_eventwin_set(DS2211_1_BASE, 1, evw_count, start_pos, end_pos);
   ...
}
```

The following example shows how to specify one event window. The event window starts at 100° and ends at 160°.

```
{
   Int32 evw_count = 1;
   dsfloat start_pos = DS2211_RAD(100);
   dsfloat end_pos   = DS2211_RAD(160);
   ...
   ds2211_multi_eventwin_set(DS2211_1_BASE, 1, evw_count,
                             &start_pos, &end_pos);
   ...
}
```

Related topics

References

ds2211_multiwin_ign_cap_read........ 151
ds2211_multiwin_ign_fifo_read........ 162
ds2211_multiwin_inj_cap_read........ 185
ds2211_multiwin_inj_fifo_read........ 195

Spark Event Capture

Introduction

Note

The spark event capture unit can also be used for injection event capture on all boards.

Where to go from here

Information in this section

Information in other sections

Spark Event Capture (DS2211 Features 📖)
The spark event capture unit provides 6 digital ignition inputs for ignition position measurement and 2 digital auxiliary capture inputs for various position measurements.

ds2211_apu_ignition_cc_setup

Syntax

```
void ds2211_apu_ignition_cc_setup(
      Int32   base,
      dsfloat thresh_a_ign1_6,
      dsfloat thresh_a_aux1,
      dsfloat thresh_a_aux2,
      dsfloat thresh_b,
      dsfloat hyster_b,
      Int32 cc_mode_ign1_6,
      Int32 cc_mode_aux1,
      Int32 cc_mode_aux2)
```

Include file

`ds2211.h`

Purpose

To set the threshold level, the hysteresis level, and the complex comparator capture-mode for IGN1 … 6, AUX1 and AUX2.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 📖).

Description

The threshold levels and the hysteresis value for the complex comparator of the ignition capture inputs are set (For details on the complex comparator, refer to Complex Comparators (DS2211 Features 📖)). You can also specify the signal capture mode. The function can be called at run time. To reduce execution time, it checks the specified parameters for any changes and performs settings only if changes are detected.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

thresh_a_ign1_6 Sets the threshold level of comparator A for ignition capture inputs IGN1 … IGN6. Range: 1 … 23.8 V

thresh_a_aux1 Sets the threshold level of comparator A for ignition capture input AUX1. Range: 1 … 23.8 V

thresh_a_aux2 Sets the threshold level of comparator A for ignition capture inputs AUX2. Range: 1 … 23.8 V

thresh_b Sets the threshold level of comparator B for all ignition capture inputs IGN1 … IGN6, AUX1 and AUX2. Range 1 … 22.65 V

hyster_b Sets the hysteresis value of comparator B for all ignition capture inputs IGN1 … IGN6, AUX1 and AUX2. Range: 0.2 … 2.4 V

cc_mode_ign1_6 Capture mode of complex comparator circuit for the ignition capture inputs IGN1 … IGN6. The following modes are supported:

Predefined Symbol	Description
DS2211_CCM_AL_TO_AT	A leading to A trailing
DS2211_CCM_BL_TO_BT	B leading to B trailing
DS2211_CCM_BL_TO_AT	B leading to A trailing
DS2211_CCM_BT_TO_AT	B trailing to A trailing

cc_mode_aux1 Capture mode of complex comparator circuit for AUX1. The supported modes are the same as for cc_mode_ign1_6 on page 136.

cc_mode_aux2 Capture mode of complex comparator circuit for AUX2. The supported modes are the same as for cc_mode_ign1_6 on page 136.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

Complex Comparators (DS2211 Features 📖)

ds2211_ign_capture_mode_set

Syntax

```
void ds2211_ign_capture_mode_set(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 count)
```

Include file

`ds2211.h`

Purpose

To set the ignition capture mode of ignition channels.

> **Note**
>
> This function is obsolete. It uses only one event window. Use **`ds2211_ign_capture_mode_setup`** instead. This uses one or two event windows (see page ds2211_ign_capture_mode_setup on page 139).

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 📖).

Description

The **`edge`** parameter defines the measured input signal as either active high or active low. Depending on this setting, **`ds2211_ignition_capture_read`** and **`ds2211_ignition_fifo_read`** define either the rising or the falling edge as the leading edge. You have five modes to specify the behavior of the ignition capture unit. The number of expected events per event window is saved in a data structure for use by **`ds2211_ignition_capture_read`**.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

edge Indicates whether the input signal is active low or active high:

Predefined Symbol	Description
DS2211_IGNCAP_RISE_EDGE	Active high input pulses. Rising edges are captured as leading edges.
DS2211_IGNCAP_FALL_EDGE	Active low input pulses. Falling edges are captured as leading edges.

mode Sets the overall ignition capture mode:

Predefined Symbol	Description
DS2211_IGNCAP_ALL_EVENT	All ignition pulses in the event window are captured.
DS2211_IGNCAP_FIRST_EVENT	The position of the leading edge of the first input pulse in the event window is captured.
DS2211_IGNCAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_IGNCAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_IGNCAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

count Specifies the number of expected events (1 … 64). If you use `ds2211_ignition_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by `ds2211_ignition_fifo_read`.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_ignition_capture_read.. 149
ds2211_ignition_fifo_read.. 159

ds2211_ign_capture_mode_setup

Syntax

```
void ds2211_ign_capture_mode_setup(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 eventwin_count,
      Int32* count)
```

Include file

`ds2211.h`

Purpose

To set the ignition capture mode of ignition channels IGN1 … IGN6.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

The `edge` parameter defines the measured input signal as either active high or active low. Depending on this setting, the functions for ignition capturing define either the rising or the falling edge as the leading edge (start of the ignition pulse). Each ignition capture function has several modes that you can select from to specify the behavior of the ignition capture unit. The number of expected events per event window (`count` parameter) is saved in a data structure for use by the functions for ignition capturing (with the exception of `ds2211_multiwin_ign_fifo_read`).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

edge Indicates whether the input signal is active low or active high:

Predefined Symbol	Description
DS2211_IGNCAP_RISE_EDGE	Active high input pulses. Rising edges are captured as leading edges.
DS2211_IGNCAP_FALL_EDGE	Active low input pulses. Falling edges are captured as leading edges.

mode Sets the overall ignition capture mode. What mode you can select depends on the function you want to use for data capturing.

For the `ds2211_multiwin_ign_cap_read`, `ds2211_multiwin_ign_fifo_read`, and `ds2211_multiwin_ign_cap_read_var` functions you must use one of the following modes:

Predefined Symbol	Description
DS2211_IGNCAP_ALL_EVENT	All ignition pulses in the event windows are captured.
DS2211_IGNCAP_FIRST_EVENT	The position of the leading edge of the first input pulse in the event window is captured.
DS2211_IGNCAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_IGNCAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_IGNCAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

For the `ds2211_multiwin_ign_cap_read_ext` and `ds2211_multiwin_ign_cap_read_var_ext` functions you must use one of the following modes:

Predefined Symbol	Description
DS2211_IGNCAP_POS_DUR_EXT	Start position, end position, and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_IGNCAP_POS_DUR_EXT_LOW	Start position, end position, and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

For the `ds2211_multiwin_ign_cap_read_abs` and `ds2211_multiwin_ign_cap_read_var_abs` functions you must use the following mode:

Predefined Symbol	Description
DS2211_IGNCAP_ABSOLUTE	Start position, end position, start time, and end time are captured. All values are absolute values relative to a defined starting point. The default starting point is the start of the angular processing unit. You can define a new starting point at run time using the DS2211_ABS_COUNTER_RESET macro.

eventwin_count Specifies the number of event windows (1 or 2). Specify the same number of event windows within all related functions, otherwise the results are incompatible.

count Specifies the number of expected events for each available event window (1 … 64). It must be a pointer to an array of length `eventwin_count`. If you use `ds2211_multiwin_ign_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by `ds2211_multiwin_ign_fifo_read`.

Return value None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_multi_eventwin_set	131
ds2211_multiwin_ign_cap_read	151
ds2211_multiwin_ign_cap_read_abs	157
ds2211_multiwin_ign_cap_read_ext	154
ds2211_multiwin_ign_cap_read_var	165
ds2211_multiwin_ign_cap_read_var_abs	172
ds2211_multiwin_ign_cap_read_var_ext	168
ds2211_multiwin_ign_fifo_read	162

ds2211_aux1_capture_mode_set

Syntax

```
void ds2211_aux1_capture_mode_set(
     Int32 base,
     Int32 edge,
     Int32 mode,
     Int32 count)
```

Include file

`ds2211.h`

Purpose

To set the ignition capture mode for auxiliary1 channel.

> **Note**
>
> This function is obsolete. It uses only one event window. Use **`ds2211_aux1_capture_mode_setup`** instead. This uses one or two event windows (see page ds2211_aux1_capture_mode_setup on page 143).

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 📖).

Description

The **`edge`** parameter defines the measured input signal as either active high or active low. Depending on this setting, **`ds2211_ignition_capture_read`** and **`ds2211_ignition_fifo_read`** define either the rising or the falling edge as the leading edge. You have five modes for specifying the behavior of the auxiliary1 capture unit. The number of expected events per event window is saved in a data structure for use by **`ds2211_ignition_capture_read`**.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

edge Indicates whether the input signal is active low or active high:

Predefined Symbol	Description
DS2211_AUX1CAP_RISE_EDGE	Active high input pulses. Rising edges are captured as leading edges.
DS2211_AUX1CAP_FALL_EDGE	Active low input pulses. Falling edges are captured as leading edges.

mode Sets the overall ignition capture mode:

Predefined Symbol	Description
DS2211_AUX1CAP_ALL_EVENT	All ignition pulses in the event window is captured.
DS2211_AUX1CAP_FIRST_EVENT	The position of the leading edge of the first input pulse in the event window is captured.
DS2211_AUX1CAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_AUX1CAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_AUX1CAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

count Specifies the number of expected events (1 … 64). If you use `ds2211_ignition_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by `ds2211_ignition_fifo_read`.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_aux1_capture_mode_setup..143
ds2211_ignition_capture_read..149
ds2211_ignition_fifo_read..159

ds2211_aux1_capture_mode_setup

Syntax

```
void ds2211_aux1_capture_mode_setup(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 eventwin_count,
      Int32* count)
```

Include file

`ds2211.h`

Purpose

To set the ignition capture mode for auxiliary 1 channel.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

The `edge` parameter defines the measured input signal as either active high or active low. Depending on this setting, the functions for ignition capturing define either the rising or the falling edge as the leading edge (start of the ignition pulse). Depending on the function you want to use for ignition capturing, you can select between different modes to specify the behavior of the ignition capture unit. The number of expected events per event window (`count` parameter) is saved in a data structure for use by the functions for ignition capturing (with the exception of `ds2211_multiwin_ign_fifo_read`).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

edge Indicates whether the input signal is active low or active high:

Predefined Symbol	Description
DS2211_AUX1CAP_RISE_EDGE	Active high input pulses. Rising edges are captured as leading edges.
DS2211_AUX1CAP_FALL_EDGE	Active low input pulses. Falling edges are captured as leading edges.

mode Sets the overall ignition capture mode.

The mode you can select depends on the function you want to use.

For the `ds2211_multiwin_ign_cap_read`, `ds2211_multiwin_ign_fifo_read`, and `ds2211_multiwin_ign_cap_read_var` functions you must use one of the following modes:

Predefined Symbol	Description
DS2211_AUX1CAP_ALL_EVENT	All ignition pulses in the event windows are captured.
DS2211_AUX1CAP_FIRST_EVENT	The position of the leading edge of the first input pulse in the event window is captured.
DS2211_AUX1CAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_AUX1CAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_AUX1CAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

For the `ds2211_multiwin_ign_cap_read_ext` and `ds2211_multiwin_ign_cap_read_var_ext` functions you must use one of the following modes:

Predefined Symbol	Description
DS2211_AUX1CAP_POS_DUR_EXT	Start position, end position, and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
S2211_AUX1CAP_POS_DUR_EXT_LOW	Start position, end position, and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

For the `ds2211_multiwin_ign_cap_read_abs` and `ds2211_multiwin_ign_cap_read_var_abs` functions you must use the following mode:

Predefined Symbol	Description
DS2211_AUX1CAP_ABSOLUTE	Start position, end position, start time, and end time are captured. All values are absolute values relative to a defined starting point. The default starting point is the start of the angular processing unit. You can define a new starting point at run time using the DS2211_ABS_COUNTER_RESET macro.

eventwin_count Specifies the number of event windows (1 or 2). Specify the same number of event windows within all related functions, otherwise the results are incompatible.

count Specifies the number of expected events for each available event window (1 … 64). It must be a pointer to an array of length `eventwin_count`. If you use `ds2211_multiwin_ign_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by `ds2211_multiwin_ign_fifo_read`.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_multi_eventwin_set	131
ds2211_multiwin_ign_cap_read	151
ds2211_multiwin_ign_cap_read_abs	157
ds2211_multiwin_ign_cap_read_ext	154
ds2211_multiwin_ign_cap_read_var	165
ds2211_multiwin_ign_cap_read_var_abs	172
ds2211_multiwin_ign_cap_read_var_ext	168
ds2211_multiwin_ign_fifo_read	162

ds2211_aux2_capture_mode_set

Syntax

```
void ds2211_aux2_capture_mode_set(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 count)
```

Include file

`ds2211.h`

Purpose

To set the ignition capture mode for auxiliary2 channel.

Note

This function is obsolete. It uses only one event window. Use **`ds2211_aux2_capture_mode_setup`** instead. This uses one or two event windows.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

The **`edge`** parameter defines the measured input signal as either active high or active low. Depending on this setting, **`ds2211_ignition_capture_read`** and **`ds2211_ignition_fifo_read`** define either the rising or the falling edge as the leading edge. You have five modes for specifying the behavior of the auxiliary2 capture unit. The number of expected events per event window is saved in a data structure for use by **`ds2211_ignition_capture_read`**.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

edge Indicates whether the input signal is active low or active high:

Predefined Symbol	Description
DS2211_AUX2CAP_RISE_EDGE	Active high input pulses. Rising edges are captured as leading edges.
DS2211_AUX2CAP_FALL_EDGE	Active low input pulses. Falling edges are captured as leading edges.

mode Sets the overall ignition capture mode:

Predefined Symbol	Description
DS2211_AUX2CAP_ALL_EVENT	All ignition pulses in the event window is captured.
DS2211_AUX2CAP_FIRST_EVENT	The position of the leading edge of the first input pulse in the event window is captured.
DS2211_AUX2CAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_AUX2CAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the master.
DS2211_AUX2CAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the master and DS2211 is the slave.

count Specifies the number of expected events (1 … 64). If you use `ds2211_ignition_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by `ds2211_ignition_fifo_read`.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_aux2_capture_mode_setup..147
ds2211_ignition_capture_read..149
ds2211_ignition_fifo_read..159

ds2211_aux2_capture_mode_setup

Syntax

```
void ds2211_aux2_capture_mode_setup(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 eventwin_count,
      Int32* count)
```

Include file

`ds2211.h`

Purpose

To set the ignition capture mode for auxiliary 2 channel.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 📖).

Description

The `edge` parameter defines the measured input signal as either active high or active low. Depending on this setting, the functions for ignition capturing define either the rising or the falling edge as the leading edge (start of the ignition pulse). Depending on the function you want to use for ignition capturing, you can select between different modes to specify the behavior of the ignition capture unit. The number of expected events per event window (`count` parameter) is saved in a data structure for use by the functions for ignition capturing (with the exception of `ds2211_multiwin_ign_fifo_read`).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

edge Indicates whether the input signal is active low or active high:

Predefined Symbol	Description
DS2211_AUX2CAP_RISE_EDGE	Active high input pulses. Rising edges are captured as leading edges.
DS2211_AUX2CAP_FALL_EDGE	Active low input pulses. Falling edges are captured as leading edges.

mode Sets the overall ignition capture mode. The mode you can select depends on the function you want to use for data capturing.

For the `ds2211_multiwin_ign_cap_read`,`ds2211_multiwin_ign_fifo_read` , and `ds2211_multiwin_ign_cap_read_var` functions you must use one of the following modes:

Predefined Symbol	Description
DS2211_AUX2CAP_ALL_EVENT	All ignition pulses in the event windows are captured.
DS2211_AUX2CAP_FIRST_EVENT	The position of the leading edge of the first input pulse in the event window is captured.
DS2211_AUX2CAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_AUX2CAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_AUX2CAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

For the `ds2211_multiwin_ign_cap_read_ext` and `ds2211_multiwin_ign_cap_read_var_ext` functions you must use one of the following modes:

Predefined Symbol	Description
DS2211_AUX2CAP_POS_DUR_EXT	Start position, end position, and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_AUX2CAP_POS_DUR_EXT_LOW	Start position, end position, and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

For the `ds2211_multiwin_ign_cap_read_abs` and `ds2211_multiwin_ign_cap_read_var_abs` functions you must use the following mode:

Predefined Symbol	Description
DS2211_AUX2CAP_ABSOLUTE	Start position, end position, start time, and end time are captured. All values are absolute values relative to a defined starting point. The default starting point is the start of the angular processing unit. You can define a new starting point at run time using the DS2211_ABS_COUNTER_RESET macro.

eventwin_count Specifies the number of event windows (1 or 2). Specify the same number of event windows within all related functions, otherwise the results are incompatible.

count Specifies the number of expected events for each available event window (1 … 64). It must be a pointer to an array of length `eventwin_count`. If you use `ds2211_multiwin_ign_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by `ds2211_multiwin_ign_fifo_read`.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_multi_eventwin_set ... 131
ds2211_multiwin_ign_cap_read ... 151
ds2211_multiwin_ign_cap_read_abs ... 157
ds2211_multiwin_ign_cap_read_ext ... 154
ds2211_multiwin_ign_cap_read_var ... 165
ds2211_multiwin_ign_cap_read_var_abs ... 172
ds2211_multiwin_ign_cap_read_var_ext ... 168
ds2211_multiwin_ign_fifo_read ... 162

ds2211_ignition_capture_read

Syntax

```
void ds2211_ignition_capture_read(
      Int32 base,
      Int32 channel,
      Int32* count,
      dsfloat* start_pos,
      dsfloat* end_pos)
```

Include file

`ds2211.h`

Purpose

To read the ignition FIFO on an event window basis.

> **Note**
>
> This function is obsolete. It uses only one event window. Use `ds2211_multiwin_ign_cap_read` instead. This uses one or two event windows.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

The ignition pulse positions/durations captured during the last event window on the specified channel are returned via the `start_pos` and `end_pos` arrays. The `count` parameter returns the number of events actually captured during the last window. If the number of captured events is lower than the number of expected events specified by `ds2211_ign_capture_mode_set`, the function returns negative position values for missing pulses. Events captured after the number of expected events are ignored. Events are read from the capture FIFO and stored in a temporary buffer separated by channel numbers. Depending on the adjusted capture mode, the function behaves differently:

Capture Mode	Function Behavior
DS2211_IGNCAP_FIRST_EVENT	Only the first event in the last event window is captured. In this case the `count` parameter always returns a value of one. The first event is always the leading edge.
DS2211_IGNCAP_ALL_EVENT	If the leading edge of an input pulse is outside the event window and the trailing edge is inside the window, the start position is invalid (negative) and the end position is valid. The pulse counter is incremented. If the leading edge of an input pulse is inside the event window and the trailing edge is outside the window, the start position is valid and the end position is invalid (negative). The pulse counter is incremented. No dummy events are generated.
DS2211_IGNCAP_PULSE_POS, DS2211_IGNCAP_PULSE_DUR, DS2211_IGNCAP_PULSE_DUR_LOW	If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 deg) is detected as two separate pulses due to the pseudo edges at the window start and end positions.

Note

Inputs channels IGN1 … 6, AUX_CAP1 and AUX_CAP2 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 0.2 V. Four capture modes are supported by the complex comparator logic connected to IGN1 … 8. The threshold, hysteresis, and capture modes are set by using `ds2211_apu_ignition_cc_setup`.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Sets the channel number:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
…	…	…
DS2211_IGNCAP_CH6	Ignition capture channel 6	6

Predefined Symbol	Description	Value
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

count Address where the current event count is written.

start_pos Specifies the address where the angle positions of the leading edge of the ignition pulses are written within the range 0 … 720°.

end_pos Address where the angle positions of the trailing edges of the ignition pulses or the pulse durations are written. The values are specified in degrees (angle positions) in the range 0 … 720° or seconds (pulse durations).

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_apu_ignition_cc_setup 135
ds2211_ign_capture_mode_set 137
ds2211_multiwin_ign_cap_read 151

ds2211_multiwin_ign_cap_read

Syntax

```
void ds2211_multiwin_ign_cap_read(
      Int32 base,
      Int32 channel,
      Int32* real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64])
```

Include file

`ds2211.h`

Purpose

To read the ignition pulse positions/durations (start and end, or start and duration) from the last event capture window(s).

> **Note**
>
> Do not use the `ds2211_multiwin_ign_cap_read` function together with other ignition capture reading functions, for example `ds2211_multiwin_ign_cap_read_var`.
> There is one exception to this rule: The `ds2211_multiwin_ign_cap_read` and `ds2211_mutiwin_ign_fifo_read functions` can be used together.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

The ignition pulse positions/durations captured during the last one or two event windows on the specified channel are returned via the `start_pos` and `end_pos` arrays. The `real_count` parameter returns the number of events actually captured during the last specified event windows.

> **Note**
>
> - If the number of captured events is less than the number of expected events specified by `ds2211_ign_capture_mode_setup, ds2211_aux1_capture_mode_setup, or ds2211_aux2_capture_mode_setup`, the function returns negative position or duration values for the missing events.
> - Events are captured up to the number of expected events. Additional events are ignored.

The function supports the following injection capture modes:

Capture Mode	Function Behavior
DS2211_IGNCAP_FIRST_EVENT	Only the first event in the last event window is captured. In this case the `count` parameter always returns a value of one. The first event is always the leading edge.
DS2211_IGNCAP_ALL_EVENT	If the leading edge of an input pulse is outside the event window and the trailing edge is inside the window, the start position is invalid (negative) and the end position is valid. The pulse counter is incremented. If the leading edge of an input pulse is inside the event window and the trailing edge is outside the window, the start position is valid and the end position is invalid (negative). The pulse counter is incremented. No dummy events are generated.
DS2211_IGNCAP_PULSE_POS, DS2211_IGNCAP_PULSE_DUR, DS2211_IGNCAP_PULSE_DUR_LOW	If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is

Capture Mode	Function Behavior
	assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 deg) is detected as two separate pulses due to the pseudo edges at the window start and end positions.

Note

Inputs channels IGN1 … 6, AUX_CAP1 and AUX_CAP2 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 0.2 V. Four capture modes are supported by the complex comparator logic connected to IGN1 … 8. The threshold, hysteresis, and capture modes are set by using `ds2211_apu_ignition_cc_setup`.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Sets the channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
…	…	…
DS2211_IGNCAP_CH6	Ignition capture channel 6	6
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

real_count Pointer to an array where the current event counts per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

start_pos Pointer to a two-dimensional array where the angle positions of the leading edges of the ignition pulses are written. The values are specified in degrees in the range 0 … 720°. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions of the trailing edges of the ignition pulses or the pulse durations are written. The values are specified in degrees (angle positions) in the range 0 … 720° or seconds (pulse durations). The order of the elements is end_pos[index for event window][index for end position/duration].

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_multiwin_ign_cap_read_ext

Syntax

```
void ds2211_multiwin_ign_cap_read_ext(
      Int32 base,
      Int32 channel,
      Int32* real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64],
      dsfloat pulse_dur[][64])
```

Include file

`ds2211.h`

Purpose

To read the ignition pulse positions (start and end) and the pulse durations from the specified event window(s).

> **Note**
>
> - This function can be used starting from board revision 3 and FPGA revision 3.
> - Do not use the `ds2211_multiwin_ign_cap_read_ext` function together with other ignition capture reading functions, for example, `ds2211_multiwin_ign_cap_read_var_ext`.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

The ignition pulse positions/durations captured during the last one or two event windows on the specified channel are returned via the `start_pos`, `end_pos` and `pulse_pos` arrays. The `real_count` parameter returns the number of events actually captured during the last specified event windows.

Note

- If the number of captured events is less than the number of expected events specified by `ds2211_ign_capture_mode_setup`, `ds2211_aux1_capture_mode_setup`, or`ds2211_aux2_capture_mode_setup`, the function returns negative position or duration values for the missing events.
- Events are captured up to the number of expected events. Additional events are ignored.

The function supports the following injection capture modes:

Capture Mode	Function Behavior
DS2211_IGNCAP_POS_DUR_EXT	Start position, end position, and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_IGNCAP_POS_DUR_EXT_LOW	Start position, end position, and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 deg) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Note

Input channels IGN1 … 6, AUX_CAP1 and AUX_CAP2 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 0.2 V. Four capture modes are supported by the complex comparator logic connected to IGN1 … 8. The threshold, hysteresis, and capture modes are set by using `ds2211_apu_ignition_cc_setup`.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Sets the channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
…	…	…

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH6	Ignition capture channel 6	6
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

real_count Pointer to an array where the current event counts per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

start_pos Pointer to a two-dimensional array where the angle position values of the leading edges of the ignition pulses are written. The values are specified in degrees in the range 0 … 720°. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle position values of the trailing edges of the ignition pulses are written. The values are specified in degrees in the range 0 … 720°. The order of the elements is end_pos[index for event window][index for end position/duration].

pulse_dur Pointer to a two-dimensional array where the pulse durations of the ignition pulses are written. The values are specified in seconds. The order of the elements is pulse_dur[index for event window][index for end position/duration].

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

DS2211 Board Revision (DS2211 Features 🕮)

References

ds2211_apu_ignition_cc_setup	135
ds2211_aux1_capture_mode_setup	143
ds2211_aux2_capture_mode_setup	147
ds2211_ign_capture_mode_setup	139
ds2211_multiwin_ign_cap_read	151
ds2211_multiwin_ign_cap_read_abs	157

ds2211_multiwin_ign_cap_read_abs

Syntax

```
void ds2211_multiwin_ign_cap_read_ext(
      Int32 base,
      Int32 channel,
      Int32* real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64],
      dsfloat start_time[][64],
      dsfloat end_time[][64])
```

Include file

`ds2211.h`

Purpose

To read absolute values of the ignition pulse positions (start and end) and the corresponding time stamps from the last event capture window(s).

> **Note**
>
> - This function can be used starting from board revision 3 and FPGA revision 3.
> - Do not use the `ds2211_multiwin_ign_cap_read_abs` function together with other ignition capture reading functions, for example `ds2211_multiwin_ign_cap_read_var_abs`.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

The ignition pulse positions/durations captured during the last 1 or 2 event windows on the specified channel are returned via the `start_pos` and `end_pos` arrays. The `real_count` parameter returns the number of events actually captured during the last specified event windows.

> **Note**
>
> - If the number of captured events is less than the number of expected events specified by `ds2211_ign_capture_mode_setup, ds2211_aux1_capture_mode_setup, or ds2211_aux2_capture_mode_setup`, the function returns negative position or duration values for the missing events.
> - Events are captured up to the number of expected events. Additional events are ignored.

The function supports the following injection capture mode:

Capture Mode	Function Behavior
DS2211_IGNCAP_ABSOLUTE	Start position, end position, start time, and end time are captured. All values are absolute values relative to a defined starting point. The default starting point is the start of the angular processing unit. You can define a new starting point at run time using the `DS2211_ABS_COUNTER_RESET` macro.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 deg) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Note

Inputs channels IGN1 … 6, AUX_CAP1 and AUX_CAP2 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 0.2 V. Four capture modes are supported by the complex comparator logic connected to IGN1 … 8. The threshold, hysteresis, and capture modes are set by using `ds2211_apu_ignition_cc_setup`.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Sets the channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
…	…	…
DS2211_IGNCAP_CH6	Ignition capture channel 6	6
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

real_count Pointer to an array where the current event counts per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

start_pos Pointer to a two-dimensional array where the angle position values of the leading edges of the ignition pulses are written. The values are specified in degrees in the range 0 … 720°. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions of the trailing edges of the ignition pulses are written. The values are specified in degrees in the range 0 … 720°. The order of the elements is end_pos[index for event window][index for end position].

start_time Pointer to a two-dimensional array where the time stamps (in seconds) of the leading edges of the ignition pulses are written. The order of the elements is start_time[index for event window][index for start time].

end_time Pointer to a two-dimensional array where the time stamps (in seconds) of the trailing edges of the ignition pulses are written. The order of the elements is end_time[index for event window][index for end time].

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

DS2211 Board Revision (DS2211 Features 📖)

References

DS2211_ABS_COUNTER_RESET 209
ds2211_apu_ignition_cc_setup 135
ds2211_aux1_capture_mode_setup 143
ds2211_aux2_capture_mode_setup 147
ds2211_ign_capture_mode_setup 139
ds2211_multiwin_ign_cap_read 151
ds2211_multiwin_ign_cap_read_ext 154

ds2211_ignition_fifo_read

Syntax

```
Int32 ds2211_ignition_fifo_read(
      Int32 base,
      Int32 channel,
      Int32 count,
      Int32 *state,
      dsfloat *pos,
      Int32 *length)
```

Include file

`ds2211.h`

Purpose

To read the ignition pulse positions/durations captured during run time in the FIFO.

> **Note**
>
> This function is obsolete. It uses only one event window. Use `ds2211_multiwin_ign_fifo_read` instead. This uses one or two event windows.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

You can use ignition channels 1 … 6 and auxiliary channels 1 and 2 for ignition angle and duration capture.

The values of the specified channel are returned via the `pos` array. The `state` array contains information on whether the corresponding event was the leading or trailing edge of a pulse. The `count` parameter specifies the number of events to be read or expected. The number of events which could be read at all is returned via the `length` parameter. If all expected events are read, the following events are not read within the current function call. You can read them with the next function call. The events are read from the capture FIFO and stored in a temporary internal buffer separated by channel numbers. The temporary buffer can store up to 64 events per channel. A buffer overflow occurs if the events are not read fast enough. A buffer overflow within the last function call can be checked by the return value.

The function supports the following ignition capture modes:

Capture Modes (XXX = IGN or AUX)	Description
DS2211_XXXCAP_ALL_EVENT	All pulse positions (start and end) are captured in degrees. Regardless of the event window settings no dummy events are detected.
DS2211_XXXCAP_FIRST_EVENT	Only the first event within each event window is captured in degrees. Regardless of the event window settings no dummy events are detected. The value of the corresponding state is always 1.
DS2211_XXXCAP_PULSE_POS	Only the angle positions of the captured events are read in degrees. Under certain circumstances pseudo edges are generated.
DS2211_XXXCAP_PULSE_DUR, DS2211_XXXCAP_PULSE_DUR_LOW	All start positions (in degree) of the pulses and the pulse durations (in seconds) are read. Each start position entry in the `pos` array follows the duration entry of the corresponding pulse. The resolution depends on the mode and can be either 4 µs or 1 µs (…DUR_LOW). Under certain circumstances pseudo edges are generated.

Note

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
...	...	...
DS2211_IGNCAP_CH6	Ignition capture channel 6	6
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

count Number of events to be read. The maximum number must not exceed the internal FIFO size (64 events).

state Address where the state of the ignition events are written. Their meaning depends on the capture mode settings (active low or active high input signal, see the `ds2211_xxx_capture_mode_setup` functions).

Capture State (XXX = IGN or AUX)	State	Description
DS2211_XXXCAP_RISE_EDGE	0	Falling edge/active low
	1	Rising edge/active high
DS2211_XXXCAP_FALL_EDGE	0	Rising edge/active high
	1	Falling edge/active low

pos Address where either the angle position (in degrees) or the angle position and pulse durations (in seconds) of the captured ignition pulses is written.

length Address where the current number of returned events is written. The `length` parameter can only be less than or equal to the `count` parameter. If more data is stored in the internal FIFO than read and exceeds the maximum FIFO size, a buffer overflow occurs. The overflow can be recognized by the return value of the function.

Return value

fifo_level FIFO level of the temporary FIFO buffer. It represents the level/state of the FIFO after the previous read operations:

Return Value	Description
fifo_level = 0	The FIFO is empty, all events were read.
fifo_level > 0	The number of events remaining in the FIFO.
fifo_level = –1	A FIFO overflow occurred.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

- ds2211_aux1_capture_mode_setup........143
- ds2211_aux2_capture_mode_setup........147
- ds2211_ign_capture_mode_setup........139
- ds2211_multi_eventwin_set........131
- ds2211_multiwin_ign_fifo_read........162

ds2211_multiwin_ign_fifo_read

Syntax

```
void ds2211_multiwin_ign_fifo_read(
      Int32 base,
      Int32 channel,
      Int32 *expected_count,
      Int32 state[][64],
      dsfloat pos[][64],
      Int32 *real_count,
      Int32 *fifo_level)
```

Include file

`ds2211.h`

Purpose

To read the ignition pulse positions/durations (start and end, or start and duration) continuously.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 📖).

Description

You can use ignition channels 1 … 6 and auxiliary channels 1 and 2 for ignition angle and duration capture.

The values of the specified channel are returned via the `pos` array. The `state` array contains information on whether the corresponding event was the leading or trailing edge of a pulse. The `expected_count` parameter specifies the number of events to be read or expected for the specified event windows (1 or 2). The number of events which could be read at all is returned individually for the specified event windows via the `real_count` parameter. If all expected events are read, the following events are not read within the current function call. You can read them with the next function call. The events are read from the capture FIFO and stored in a temporary internal buffer separated by channel numbers. The temporary buffer can store up to 64 events per channel. A buffer overflow occurs if the events are not read fast enough. A buffer overflow within the last function call can be checked by the returned value of the `fifo_level` parameter.

The function supports the following ignition capture modes:

Capture Modes (XXX = IGN or AUX)	Description
DS2211_XXXCAP_ALL_EVENT	All pulse positions (start and end) are captured in degrees. Regardless of the event window settings no dummy events are detected.
DS2211_XXXCAP_FIRST_EVENT	Only the first event within each event window is captured in degrees. Regardless of the event window settings no dummy events are detected. The value of the corresponding state is always 1.
DS2211_XXXCAP_PULSE_POS	Only the angle positions of the captured events are read in degrees. Under certain circumstances pseudo edges are generated.
DS2211_XXXCAP_PULSE_DUR, DS2211_XXXCAP_PULSE_DUR_LOW	All start positions (in degree) of the pulses and the pulse durations (in seconds) are read. Each start position entry in the `pos` array follows the duration entry of the corresponding pulse. The resolution depends on the mode and can be either 4 µs or 1 µs (…DUR_LOW). Under certain circumstances pseudo edges are generated.

Note

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
…	…	…
DS2211_IGNCAP_CH6	Ignition capture channel 6	6
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

expected_count Pointer to an array containing the number of events per event window to be read. The maximum number must not exceed the internal FIFO size (64 events). The length of the array must be the number of event windows (see `ds2211_multi_eventwin_set` on page 131).

state Pointer to a two-dimensional array where the state of the ignition events per event window are written. The order of the elements is state[index for event window][index for state]. Their meaning depends on the capture mode settings (active low or active high input signal, see the `ds2211_xxx_capture_mode_setup` functions):

Capture State (XXX = IGN or AUX)	State	Description
DS2211_XXXCAP_RISE_EDGE	0	Falling edge/active low
	1	Rising edge/active high
DS2211_XXXCAP_FALL_EDGE	0	Rising edge/active high
	1	Falling edge/active low

pos Pointer to a two-dimensional array where either the angle positions (in degrees) or the angle positions and pulse durations (in seconds) of the captured ignition pulses per event window are written. The order of the elements is pos[index for event window][index for position/duration].

real_count Pointer to an array where the current numbers of returned events per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). The numbers written can only be less than or equal to the numbers given by the `expected_count` parameter. If more data is stored in the internal FIFO than read and exceeds the maximum FIFO size, an overflow of the FIFO occurs. The overflow can be recognized by the return value of the function.

fifo_level Pointer to an array where the FIFO level of the temporary FIFO buffer is written. The length of the array must be the number of event windows (see `ds2211_multi_eventwin_set` on page 131). The value represents the level/state of the FIFO after the previous read operations:

Return Value	Description
fifo_level = 0	The FIFO is empty, all events were read.
fifo_level > 0	The number of events remaining in the FIFO.
fifo_level = –1	A FIFO overflow occurred.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_aux1_capture_mode_setup 143
ds2211_aux2_capture_mode_setup 147
ds2211_ign_capture_mode_setup 139
ds2211_multi_eventwin_set 131

ds2211_multiwin_ign_cap_read_var

Syntax

```
void ds2211_multiwin_ign_cap_read_var(
      Int32 base,
      Int32 channel,
      Int32 *real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64],
      Int32 *status,
      Int32 *counter)
```

Include file

`ds2211.h`

Purpose

To read the ignition pulse positions/durations (start and end, or start and duration) from the current event capture window(s).

> **Note**
>
> Do not use the `ds2211_multiwin_ign_cap_read_var` function together with other ignition capture reading functions, for example `ds2211_multiwin_ign_cap_read`.
> There is one exception to this rule: The `ds2211_multiwin_ign_cap_read_var` and `ds2211_mutiwin_ign_fifo_read` functions can be used together.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 📖).

Description

You can use ignition channels 1 … 6 and auxiliary channels 1 and 2 for ignition angle and duration capture.

The values of the specified channel are returned via the `start_pos` and `end_pos` arrays.

The `real_count` parameter returns the number of events actually captured during the last specified event windows.

Note

- If the number of captured events is less than the number of expected events specified by `ds2211_ign_capture_mode_setup,` `ds2211_aux1_capture_mode_setup,` or `ds2211_aux2_capture_mode_setup`, the function returns negative position or duration values for the missing events.
- Events are captured up to the number of expected events. Additional events are ignored.

The `real_count` parameter returns the number of events actually captured during the last event window. If the number of captured events is less than the number of expected events (specified by `ds2211_ign_capture_mode_setup`, `ds2211_aux1_capture_mode_setup`, `ds2211_aux2_capture_mode_setup`), the function returns negative position values for missing pulses. If the number of expected events is reached, further events are ignored.

The function combines the functionality of the event-window-based and continuous event capturing. Although the values are captured on an event-window basis, you get the current status via the status and counter parameters. For an example, refer to Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 🕮).

The function supports the following ignition capture modes:

Capture Mode	Function Behavior
DS2211_IGNCAP_FIRST_EVENT	Only the first event in the last event window is captured. In this case the `count` parameter always returns a value of one. The first event is always the leading edge.
DS2211_IGNCAP_ALL_EVENT	If the leading edge of an input pulse is outside the event window and the trailing edge is inside the window, the start position is invalid (negative) and the end position is valid. The pulse counter is incremented. If the leading edge of an input pulse is inside the event window and the trailing edge is outside the window, the start position is valid and the end position is invalid (negative). The pulse counter is incremented. No dummy events are generated.
DS2211_IGNCAP_PULSE_POS, DS2211_IGNCAP_PULSE_DUR, DS2211_IGNCAP_PULSE_DUR_LOW	If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 deg) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
…	…	…
DS2211_IGNCAP_CH6	Ignition capture channel 6	6
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

real_count Pointer to an array where the current numbers of returned events per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). It returns the total number of pulses captured in the last complete event window. The value can be greater than the specified expected number of pulses. The value is updated at the end of each event window.

start_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the leading edges of the ignition pulses are written. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the trailing edges of the ignition pulses or the pulse durations (in seconds) are written. The order of the elements is end_pos[index for event window][index for end position/duration].

status Pointer to an array where the status of the currently captured event data is written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

Value	Description
0	The update of data in the current event window is still running. The `counter` parameter is still being updated. The `start_pos` and `end_pos` parameters may contain a mix of old and new data.
1	Data in the current event window is no longer being updated. The `counter` parameter is not being updated. The `start_pos` and `end_pos` parameters contain only new data (Exception: In the current event window, either no pulses occurred, or fewer than in the previous event window).

The status is set to 0 when the first event of each event window occurs. The status is set to 1 if the number of expected pulses is reached or at the end of each event window (then the number of expected pulses was not reached in the event window).

After initialization, the status is 0.

counter Pointer to an array where the current event counts are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). The counter counts every edge occurring in the current event window until the number of expected pulses is reached. Additional pulses are ignored.

Negative/positive signs alternate to distinguish between leading and trailing edges.

Sign	Description
Negative	A leading edge is detected. The count value is incremented.
Positive	A trailing edge is detected. The count value is not incremented.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 📖)

References

ds2211_aux1_capture_mode_setup	143
ds2211_aux2_capture_mode_setup	147
ds2211_ign_capture_mode_setup	139
ds2211_multiwin_ign_cap_read_var_abs	172
ds2211_multiwin_ign_cap_read_var_ext	168

ds2211_multiwin_ign_cap_read_var_ext

Syntax

```
void ds2211_multiwin_ign_cap_read_var(
      Int32 base,
      Int32 channel,
      Int32 *real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64],
      dsfloat pulse_dur[][64],
      Int32 *status,
      Int32 *counter)
```

Include file

`ds2211.h`

Purpose

To read the ignition pulse positions (start and end) and the pulse durations from the current event capture window(s).

> **Note**
>
> - This function can be used starting from board revision 3 and FPGA revision 3.
> - Do not use the `ds2211_multiwin_ign_cap_read_var_ext` function together with the `ds2211_multiwin_ign_cap_read_ext` function.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

You can use ignition channels 1 … 6 and auxiliary channels 1 and 2 for ignition angle and duration capture.

The values of the specified channel are returned via the `start_pos` and `end_pos` arrays. The `real_count` parameter returns the number of events actually captured during the last specified event windows.

> **Note**
>
> - If the number of captured events is less than the number of expected events specified by `ds2211_ign_capture_mode_setup`, `ds2211_aux1_capture_mode_setup`, or`ds2211_aux2_capture_mode_setup`, the function returns negative position or duration values for the missing events.
> - Events are captured up to the number of expected events. Additional events are ignored.

The function combines the functionality of the event-window-based and continuous event capturing. Although the values are captured on an event-window basis, you get the current status via the status and counter parameters. For an example, refer to Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 🕮).

The function supports the following ignition capture modes:

Capture Mode	Function Behavior
DS2211_IGNCAP_POS_DUR_EXT	Start position, end position, and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_IGNCAP_POS_DUR_EXT_LOW	Start position, end position, and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 deg) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
…	…	…
DS2211_IGNCAP_CH6	Ignition capture channel 6	6
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

real_count Pointer to an array where the current numbers of returned events per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). It returns the total number of pulses captured in the last complete event window. The value can be greater than the specified expected number of pulses. The value is updated at the end of each event window.

start_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the leading edges of the ignition pulses are written. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the trailing edges of the ignition pulses are written. The order of the elements is end_pos[index for event window][index for end position].

pulse_dur Pointer to a two-dimensional array where the pulse durations (in seconds) are written. The order of the elements is pulse_dur[index for event window][index for end position/duration].

status Pointer to an array where the status of the currently captured event data is written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

Value	Description
0	The update of data in the current event window is still running. The `counter` parameter is still being updated. The `start_pos, end_pos` and `pulse_dur` parameters may contain a mix of old and new data.
1	Data in the current event window is no longer being updated. The `counter` parameter is not being updated. The `start_pos, end_pos` and `pulse_dur` parameters contain only new data (Exception: In the current event window, either no pulses occurred, or fewer than in the previous event window).

The status is set to 0 when the first event of each event window occurs. The status is set to 1 if the number of expected pulses is reached or at the end of each event window (then the number of expected pulses was not reached in the event window).

After initialization, the status is 0.

counter Pointer to an array where the current event counts are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). The counter counts every edge occurring in the current event window until the number of expected pulses is reached. Additional pulses are ignored.

Negative/positive signs alternate to distinguish between leading and trailing edges.

Sign	Description
Negative	A leading edge is detected. The count value is incremented.
Positive	A trailing edge is detected. The count value is not incremented.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

DS2211 Board Revision (DS2211 Features 📖)

Examples

Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 📖)

References

ds2211_aux1_capture_mode_setup	143
ds2211_aux2_capture_mode_setup	147
ds2211_ign_capture_mode_setup	139
ds2211_multiwin_ign_cap_read_var	165
ds2211_multiwin_ign_cap_read_var_abs	172

ds2211_multiwin_ign_cap_read_var_abs

Syntax

```
void ds2211_multiwin_ign_cap_read_var(
      Int32 base,
      Int32 channel,
      Int32 *real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64],
      dsfloat start_time[][64],
      dsfloat end_time[][64],
      Int32 *status,
      Int32 *counter)
```

Include file

`ds2211.h`

Purpose

To read absolute values of the ignition pulse positions (start and end) and the corresponding time stamps from the current event capture window(s).

Note

- This function can be used starting from board revision 3 and FPGA revision 3.
- Do not use the `ds2211_multiwin_ign_cap_read_var_abs` function together with other ignition capture reading functions, for example `ds2211_multiwin_ign_cap_read_abs`.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

You can use ignition channels 1 … 6 and auxiliary channels 1 and 2 for ignition angle and duration capture.

The values of the specified channel are returned via the `start_pos` and `end_pos` arrays. The `real_count` parameter returns the number of events actually captured during the last specified event windows.

Note

- If the number of captured events is less than the number of expected events specified by `ds2211_ign_capture_mode_setup`, `ds2211_aux1_capture_mode_setup`, or `ds2211_aux2_capture_mode_setup`, the function returns negative position or duration values for the missing events.
- Events are captured up to the number of expected events. Additional events are ignored.

The function combines the functionality of the event-window-based and continuous event capturing. Although the values are captured on an event-window basis, you get the current status via the status and counter parameters. For an example, refer to Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 🕮).

The function supports the following injection capture mode:

Capture Mode	Function Behavior
DS2211_IGNCAP_ABSOLUTE	Start position, end position, start time, and end time are captured. All values are absolute values relative to a defined starting point. The default starting point is the start of the angular processing unit. You can define a new starting point at run time using the `DS2211_ABS_COUNTER_RESET` macro.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 deg) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
…	…	…
DS2211_IGNCAP_CH6	Ignition capture channel 6	6
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

real_count Pointer to an array where the current numbers of returned events per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). It returns the total number of pulses captured in the last complete event window.

The value can be greater than the specified expected number of pulses. The value is updated at the end of each event window.

start_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the leading edges of the ignition pulses are written. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the trailing edges of the ignition pulses are written. The order of the elements is end_pos[index for event window][index for end position].

start_time Pointer to a two-dimensional array where the time stamps (in seconds) of the leading edges of the ignition pulses are written. The order of the elements is start_time[index for event window][index for start time].

end_time Pointer to a two-dimensional array where the time stamps (in seconds) of the trailing edges of the ignition pulses are written. The order of the elements is end_time[index for event window][index for end time].

status Pointer to an array where the status of the currently captured event data is written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

Value	Description
0	The update of data in the current event window is still running. The `counter` parameter is still being updated. The `start_pos`, `end_pos`, `start_time` and `end_time` parameters may contain a mix of old and new data.
1	Data in the current event window is no longer being updated. The `counter` parameter is not being updated. The `start_pos`, `end_pos`, `start_time` and `end_time` parameters contain only new data (Exception: In the current event window, either no pulses occurred, or fewer than in the previous event window).

The status is set to 0 when the first event of each event window occurs. The status is set to 1 if the number of expected pulses is reached or at the end of each event window (then the number of expected pulses was not reached in the event window).

After initialization, the status is 0.

counter Pointer to an array where the current event counts are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). The counter counts every edge occurring in the current event window until the number of expected pulses is reached. Additional pulses are ignored.

Negative/positive signs alternate to distinguish between leading and trailing edges.

Sign	Description
Negative	A leading edge is detected. The count value is incremented.

Sign	Description
Positive	A trailing edge is detected. The count value is not incremented.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

DS2211 Board Revision (DS2211 Features 🕮)

References

ds2211_ign_capture_mode_setup..139
ds2211_multiwin_ign_cap_read_var...165

ds2211_ignition_status_read

Syntax

```
void ds2211_ignition_status_read(
      Int32 base,
      Int32 *channels,
      Int32 count,
      Int32 *states)
```

Include file

`ds2211.h`

Purpose

To read the current status (0/1) of the ignition capture inputs.

Note

Use of this function is independent of the current ignition or auxiliary capture mode. The function does not conflict with any other capture read function.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channels Address where the array of length `count` containing the channel numbers (1 … 8) is stored. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_IGNCAP_CH1	Ignition capture channel 1	1
…	…	…
DS2211_IGNCAP_CH6	Ignition capture channel 6	6
DS2211_AUXCAP_CH1	Auxiliary capture channel 1	7
DS2211_AUXCAP_CH2	Auxiliary capture channel 2	8

count Number of channels to be read

states Address where the array of length `count` is stored. The array contains the resulting status values (0 or 1) of the specified capture input channels. The array must be allocated by the calling instance.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_multiwin_ign_cap_read.......... 151
ds2211_multiwin_ign_cap_read_var.......... 165
ds2211_multiwin_ign_fifo_read.......... 162

Injection Pulse Position and Fuel Amount Measurement

Where to go from here

Information in this section

DS2211_ABS_COUNTER_RESET..209
Clears the internal 40-bit time stamp counter (time base=250ns) and the internal 24-bit engine cycle counter.

Information in other sections

Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮)
The injection event capture unit provides 16 digital injection inputs split into 2 groups for injection pulse position and fuel amount measurement.

ds2211_apu_injection_cc_setup

Syntax

```
void ds2211_apu_injection_cc_setup(
      Int32    base,
      dsfloat thresh_a_inj1_6,
      dsfloat thresh_a_inj7,
      dsfloat thresh_a_inj8,
      dsfloat thresh_b,
      dsfloat hyster_b,
      Int32    cc_mode_inj1_8)
```

Include file

`ds2211.h`

Purpose

To set the threshold level, the hysteresis level and the complex comparator capture-mode for INJ1 … 6 and INJ7 … 8 (PWM_IN7 … 8).

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮).

Description

The threshold levels and the hysteresis value for the complex comparator of the injection capture inputs are set. You can also specify the signal capture mode. For details to the complex comparators, refer to Complex Comparators (DS2211 Features 🕮). The function can be called at run time. To reduce execution time, it checks the specified parameters for any changes and performs settings only if changes are detected.

Note

The pins INJ7 and INJ8 are shared with PWM_IN7 and PWM_IN8. Therefore, the settings can also be made via the setup function for the PWM signal measurement (see `ds2211_digin_threshold_set` on page 22).

Parameters

base PHS-bus board base address

thresh_a_inj1_6 Sets the threshold level of comparator A for injection capture inputs INJ1 … INJ6. Range: 1 … 23.8 V.

thresh_a_inj7 Sets the threshold level of comparator A for injection capture input INJ7/PWM_IN7. Range: 1 … 23.8 V.

thresh_a_inj8 Sets the threshold level of comparator A for injection capture inputs INJ8/PWM_IN8. Range: 1 … 23.8 V.

thresh_b Sets the threshold level of comparator B for all injection capture inputs INJ1 … INJ8. Range: 1 … 22.65 V.

hyster_b Sets the hysteresis value of comparator B for all injection capture inputs INJ1 … INJ8. Range: 0.2 … 2.4 V.

cc_mode_inj1_8 Capture mode of complex comparator circuit for all injection capture inputs INJ1 … INJ8. The following modes are supported:

Function	Mode
DS2211_CCM_AL_TO_AT	A leading to A trailing
DS2211_CCM_BL_TO_BT	B leading to B trailing
DS2211_CCM_BL_TO_AT	B leading to A trailing
DS2211_CCM_BT_TO_AT	B trailing to A trailing

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

Complex Comparators (DS2211 Features 🕮)

References

ds2211_digin_threshold_set.. 22

ds2211_inj_capture_mode_set

Syntax

```
void ds2211_inj_capture_mode_set(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 count)
```

Include file

`ds2211.h`

Purpose

To set the injection capture mode.

Note

This function is obsolete. It uses only one event window. Use `ds2211_inj_capture_mode_setup` instead. This uses one or two event windows (see `ds2211_inj_capture_mode_setup` on page 181).

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 📖).

Description

This function sets the injection capture mode. The `edge` parameter defines the measured input signal as either active high or active low. Depending on this setting, `ds2211_injection_capture_read` and `ds2211_injection_fifo_read` define either the rising or the falling edge as leading edge. Overall you have three modes for specifying the behavior of the injection capture unit. The number of expected events per event window is saved in a data structure for use by `ds2211_injection_fifo_read`.

Parameters

base PHS-bus board base address

edge Selects active high or active low input pulses. The following symbols are predefined:

Function	Parameter
DS2211_INJCAP_RISE_EDGE	Active high input pulses. Rising edges are captured as leading edges.
DS2211_INJCAP_FALL_EDGE	Active low input pulses. Falling edges are captured as leading edges.

mode Sets the overall injection capture mode:

Function	Parameter
DS2211_INJCAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_INJCAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_INJCAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

count Specifies the number of expected events (1 … 64). If you use `ds2211_injection_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by `ds2211_injection_fifo_read`.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_injection_capture_read 183
ds2211_injection_fifo_read 193

ds2211_inj_capture_mode_setup

Syntax

```
void ds2211_inj_capture_mode_setup(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 eventwin_count,
      Int32 *count)
```

Include file

`ds2211.h`

Purpose

To set the injection capture mode for the injection capture channels.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 📖).

Description

This function sets the injection capture mode. The `edge` parameter defines the measured input signal as either active high or active low. Each ignition capture function has several modes that you can select from to specify the behavior of the injection capture unit. The number of expected events per event window (`count` parameter) is saved in a data structure for use by the functions for ignition capturing (with the exception of `ds2211_multiwin_ign_fifo_read`).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

edge Selects active high or active low input pulses. The following symbols are predefined:

Function	Parameter
DS2211_INJCAP_RISE_EDGE	Active high input pulses. Rising edges are captured as leading edges.
DS2211_INJCAP_FALL_EDGE	Active low input pulses. Falling edges are captured as leading edges.

mode Sets the overall injection capture mode: What mode you can select depends on the function you want to use.

For the functions

- `ds2211_multiwin_inj_cap_read`
- `ds2211_multiwin_inj_fifo_read`
- `ds2211_multiwin_inj_cap_read_var`

you must use one of the following modes:

Function	Parameter
DS2211_INJCAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_INJCAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_INJCAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

For the functions

- `ds2211_multiwin_inj_cap_read_ext`
- `ds2211_multiwin_inj_cap_read_var_ext`

you must use one of the following modes:

Function	Parameter
DS2211_INJCAP_POS_DUR_EXT	Start position, end position, and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_INJCAP_POS_DUR_EXT_LOW	Start position, end position, and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

For the functions

- `ds2211_multiwin_inj_cap_read_abs`
- `ds2211_multiwin_inj_cap_read_var_abs`

you must use the following mode:

Function	Parameter
DS2211_INJCAP_ABSOLUTE	Start position, end position, start time, and end time are captured. All values are absolute values relative to a defined starting point. The default starting point is the start of the angular processing unit. You can define a new starting point at run time using the `DS2211_ABS_COUNTER_RESET` macro.

eventwin_count Specifies the number of event windows (1 or 2). Specify the same number of event windows within all related functions, otherwise the results are incompatible.

count Pointer to an array containing the numbers of expected events per event window (1 … 64). If you use `ds2211_multiwin_inj_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by `ds2211_multiwin_inj_fifo_read`.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_multi_eventwin_set	131
ds2211_multiwin_ign_fifo_read	162
ds2211_multiwin_inj_cap_read	185
ds2211_multiwin_inj_cap_read_abs	190
ds2211_multiwin_inj_cap_read_ext	188
ds2211_multiwin_inj_cap_read_var	198
ds2211_multiwin_inj_cap_read_var_abs	204
ds2211_multiwin_inj_cap_read_var_ext	201
ds2211_multiwin_inj_fifo_read	195

ds2211_injection_capture_read

Syntax

```
void ds2211_injection_capture_read(
      Int32 base,
      Int32 channel,
      Int32* count,
      dsfloat* start_pos,
      dsfloat* end_pos)
```

Include file

`ds2211.h`

Purpose

To capture fuel injection pulse positions or durations.

> **Note**
>
> This function is obsolete. It uses only one event window. Use `ds2211_multiwin_inj_cap_read` instead. This uses one or two event windows (see `ds2211_multiwin_inj_cap_read` on page 185).

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮).

Description

The fuel injection pulse positions and their durations captured in the last event window on the specified channel are returned by the `start_pos` and `end_pos` arrays. The `count` parameter returns the number of events actually captured in the last event window.

> **Note**
>
> - Input channels 7 and 8 are shared with PWM measurement. If you use these channels for PWM signal measurement you cannot use them for injection capture, refer to `ds2211_pwm_in` on page 85.
> - If the number of captured events is less than the number of expected events specified by `ds2211_inj_capture_mode_set`, the function returns negative position or duration values for the missing events.
> - Events are captured up to the number of expected events. Additional events are ignored.

Input channels INJ … INJ8 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 200mV. Four capture modes are supported by the complex comparator logic connected to INJ1 … INJ8. Thresholds A and B, and hysteresis B of INJ1 … INJ8, and capture mode are set by using `ds2211_apu_injection_cc_setup`. Threshold A of comparator INJ7 … INJ8 can also be set by using `ds2211_digin_threshold_set`.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Sets the channel number:

Predefined Symbol	Description
DS2211_INJCAP_CH1	Injection capture channel 1
…	…
DS2211_INJCAP_CH8	Injection capture channel 8

count Address where the current event count is written.

start_pos Address where the angle positions at the beginning of the injection pulses are written (in degrees).

end_pos Address where the angle positions at the end of the injection pulses (in degrees) or the pulse durations (in seconds) are written.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_apu_injection_cc_setup 178
ds2211_digin_threshold_set 22
ds2211_inj_capture_mode_set 180
ds2211_pwm_in 85

ds2211_multiwin_inj_cap_read

Syntax

```
void ds2211_multiwin_inj_cap_read(
      Int32 base,
      Int32 channel,
      Int32 *real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64])
```

Include file

`ds2211.h`

Purpose

To read the injection pulse positions/durations (start and end, or start and duration) from the last event capture window(s).

> **Note**
>
> Do not use the `ds2211_multiwin_inj_cap_read` function together with other injection capture reading functions, for example `ds2211_multiwin_inj_cap_read_var`. There is one exception to this rule: The `ds2211_multiwin_inj_cap_read` and `ds2211_mutiwin_inj_fifo_read functions` can be used together.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮).

Description

The fuel injection pulse positions captured in the last specified event window(s) on the specified channel are returned by the `start_pos` and `end_pos` arrays. The `count` parameter returns the number of events actually captured in the last specified event windows.

> **Note**
>
> - Input channels 7 and 8 are shared with PWM measurement. If you use these channels for PWM signal measurement you cannot use them for injection capture, refer to `ds2211_pwm_in` on page 85.
> - If the number of captured events is less than the number of expected events specified by `ds2211_inj_capture_mode_setup`, the function returns negative position or duration values for the missing events.
> - Events are captured up to the number of expected events. Additional events are ignored.

The function supports the following injection capture modes:

Function	Parameter
DS2211_INJCAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_INJCAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_INJCAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the

window border (for example, 0 deg) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Inputs channels INJ1 … INJ8 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 200mV. Four capture modes are supported by the complex comparator logic connected to INJ1 … INJ8. Thresholds A and B, and hysteresis B of INJ1 … INJ8, and capture mode are set by using `ds2211_apu_injection_cc_setup`. Threshold A of comparator INJ7 … INJ8 can also be set by using `ds2211_digin_threshold_set`.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Sets the channel number:

Predefined Symbol	Description
DS2211_INJCAP_CH1	Injection capture channel 1
…	…
DS2211_INJCAP_CH8	Injection capture channel 8

real_count Pointer to an array where the current event counts per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

start_pos Pointer to a two-dimensional array where the angle positions at the beginning of the injection pulses per event window are written. The values are specified in degrees in the range 0 … 720°. The order of the elements is start_pos[index for event window][index for start position].

end_pos Address where the angle positions at the end of the injection pulses or the pulse durations per event window are written. The position values are specified in degrees in the range 0 … 720°. The pulse durations are specified in seconds. The order of the elements is end_pos[index for event window][index for end position/duration].

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_apu_injection_cc_setup 178
ds2211_digin_threshold_set 22
ds2211_inj_capture_mode_setup 181
ds2211_multiwin_inj_cap_read_abs 190

ds2211_multiwin_inj_cap_read_ext

Syntax

```
void ds2211_multiwin_inj_cap_read(
      Int32 base,
      Int32 channel,
      Int32 *real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64],
      dsfloat pulse_dur[][64])
```

Include file

`ds2211.h`

Purpose

To read the injection pulse positions (start and end) and the pulse durations from the last event capture window(s).

Note

- This function can be used starting from board revision 3 and FPGA revision 3.
- Do not use the **`ds2211_multiwin_inj_cap_read_ext`** function together with other injection capture reading functions, for example `ds2211_multiwin_inj_cap_read_var_ext`.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮).

Description

The fuel injection pulse positions and their durations captured in the last specified event windows on the specified channel are returned by the **`start_pos`** and **`end_pos`** arrays. The **`count`** parameter returns the number of events actually captured in the last specified event windows.

Note

- Input channels 7 and 8 are shared with PWM measurement. If you use these channels for PWM signal measurement you cannot use them for injection capture, refer to ds2211_pwm_in on page 85.
- If the number of captured events is less than the number of expected events specified by `ds2211_inj_capture_mode_setup`, the function returns negative position or duration values for the missing events.
- Events are captured up to the number of expected events. Additional events are ignored.

The function supports the following injection capture modes:

Function	Parameter
DS2211_INJCAP_POS_DUR_EXT	Start position, end position, and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_INJCAP_POS_DUR_EXT_LOW	Start position, end position, and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 °) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Input channels INJ1 … INJ8 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 200mV. Four capture modes are supported by the complex comparator logic connected to INJ1 … INJ8. Thresholds A and B, and hysteresis B of INJ1 … INJ8, and capture mode are set by using `ds2211_apu_injection_cc_setup`. Threshold A of comparator INJ7 … INJ8 can also be set by using `ds2211_digin_threshold_set`.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Sets the channel number:

Predefined Symbol	Description
DS2211_INJCAP_CH1	Injection capture channel 1
…	…
DS2211_INJCAP_CH8	Injection capture channel 8

real_count Pointer to an array where the current event counts per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

start_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the leading edges of the injection pulses are written. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the trailing edges of the injection pulses are written. The order of the elements is end_pos[index for event window][index for end position/duration].

pulse_dur Pointer to a two-dimensional array where the pulse durations (in seconds) of the injection pulses are written. The order of the elements is pulse_dur[index for event window][index for duration].

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

DS2211 Board Revision (DS2211 Features 📖)

References

ds2211_apu_injection_cc_setup	178
ds2211_digin_threshold_set	22
ds2211_inj_capture_mode_setup	181
ds2211_multiwin_inj_cap_read	185
ds2211_multiwin_inj_cap_read_abs	190
ds2211_pwm_in	85

ds2211_multiwin_inj_cap_read_abs

Syntax

```
void ds2211_multiwin_inj_cap_read(
      Int32 base,
      Int32 channel,
      Int32 *real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64]),
      dsfloat start_time[][64]),
      dsfloat end_time[][64])
```

Include file

`ds2211.h`

Purpose

To read absolute values of the injection pulse positions (start and end) and the corresponding time stamps from the last event capture window(s).

> **Note**
>
> - This function can be used starting from board revision 3 and FPGA revision 3.
> - Do not use the `ds2211_multiwin_inj_cap_read_abs` function together with other injection capture reading functions, for example `ds2211_multiwin_inj_cap_read_abs`.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮).

Description

The fuel injection pulse positions and their durations captured in the last specified event windows on the specified channel are returned by the `start_pos` and `end_pos` arrays. The `count` parameter returns the number of events actually captured in the last specified event windows.

> **Note**
>
> - Input channels 7 and 8 are shared with PWM measurement. If you use these channels for PWM signal measurement you cannot use them for injection capture, refer to ds2211_pwm_in on page 85.
> - If the number of captured events is less than the number of expected events specified by `ds2211_inj_capture_mode_setup`, the function returns negative position or duration values for the missing events.
> - Events are captured up to the number of expected events. Additional events are ignored.

The function supports the following injection capture mode:

Function	Parameter
DS2211_INJCAP_ABSOLUTE	Start position, end position, start time, and end time are captured. All values are absolute values relative to a defined starting point. The default starting point is the start of the angular processing unit. You can define a new starting point at run time using the `DS2211_ABS_COUNTER_RESET` macro.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the

window border (for example, 0°) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Input channels INJ1 … INJ8 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 200mV. Four capture modes are supported by the complex comparator logic connected to INJ1 … INJ8. Thresholds A and B, and hysteresis B of INJ1 … INJ8, and capture mode are set by using `ds2211_apu_injection_cc_setup`. Threshold A of comparator INJ7 … INJ8 can also be set by using `ds2211_digin_threshold_set`.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Sets the channel number:

Predefined Symbol	Description
DS2211_INJCAP_CH1	Injection capture channel 1
…	…
DS2211_INJCAP_CH8	Injection capture channel 8

real_count Pointer to an array where the current event counts per event window are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

start_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the leading edges of the injection pulses are written. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the trailing edges of the injection pulses are written. The order of the elements is end_pos[index for event window][index for end position].

start_time Pointer to a two-dimensional array where the time stamps (in seconds) of the leading edges of the injection pulses are written. The order of the elements is start_time[index for event window][index for start time].

end_time Pointer to a two-dimensional array where the time stamps (in seconds) of the trailing edges of the injection pulses are written. The order of the elements is end_time[index for event window][index for end time].

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

DS2211 Board Revision (DS2211 Features 🕮)

References

DS2211_ABS_COUNTER_RESET	209
ds2211_apu_injection_cc_setup	178
ds2211_digin_threshold_set	22
ds2211_inj_capture_mode_setup	181
ds2211_multiwin_inj_cap_read	185
ds2211_multiwin_inj_cap_read_ext	188
ds2211_pwm_in	85

ds2211_injection_fifo_read

Syntax

```
Int32 ds2211_injection_fifo_read(
      Int32 base,
      Int32 channel,
      Int32 count,
      Int32 *state,
      dsfloat *pos,
      Int32 *length)
```

Include file

`ds2211.h`

Purpose

To read the injection pulse positions/durations captured during run time in the FIFO of the injection channels 1 … 8.

Note

This function is obsolete. It uses only one event window. Use `ds2211_multiwin_inj_fifo_read` instead. This uses one or two event windows (see `ds2211_multiwin_inj_fifo_read` on page 195).

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 🕮).

Description

The values of the specified channel are returned via the `pos` array. The `state` array contains information on whether the corresponding event was the leading or trailing edge of a pulse. The `count` parameter specifies the number of events to be read or expected. The number of events which could be read at all is returned via the `length` parameter. If all expected events are read, the following events are not read within the current function call. You can read them with the next function call. The events are read from the capture FIFO and stored in a temporary internal buffer separated by channel numbers. The temporary buffer can store up to 64 events per channel. A buffer overflow occurs if the events are not read fast enough. A buffer overflow within the last function call can be checked by the return value.

The function supports all possible injection capture modes:

Capture Modes	Description
DS2211_INJCAP_PULSE_POS	Only the angle positions of the captured events are read in degrees. Under certain circumstances pseudo edges are generated.
DS2211_INJCAP_PULSE_DUR, DS2211_INJCAP_PULSE_DUR_LOW	All start positions (in degree) of the pulses and the pulse durations (in seconds) are read. Each start position entry in the `pos` array follows the duration entry of the corresponding pulse. The resolution depends on the mode and can be either 4 µs or 1 µs (…DUR_LOW). Under certain circumstances pseudo edges are generated.

Note

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_INJCAP_CH1	Injection capture channel 1
…	…
DS2211_INJCAP_CH8	Injection capture channel 8

count Number of events to be read. The maximum number must not exceed the internal FIFO size (64 events).

state Address where the state of the injection events are written.

Capture State	State	Description
DS2211_INJCAP_RISE_EDGE	0	Falling edge/active low
	1	Rising edge/active high
DS2211_INJCAP_FALL_EDGE	0	Rising edge/active high
	1	Falling edge/active low

pos Specifies the address where either the angle position (in degrees) or the angle position and pulse durations (in seconds) of the captured injection pulses is written.

length Address where the current number of returned events is written. The `length` parameter can only be less than or equal to the `count` parameter. If more data is stored in the internal FIFO than read and exceeds the maximum FIFO size, a buffer overflow occurs. The overflow can be recognized by the return value of the function.

Return value

fifo_level FIFO level of the temporary FIFO buffer. It represents the level/state of the FIFO after the previous read operations:

Return Value	Description
fifo_level = 0	The FIFO is empty, all events were read.
fifo_level > 0	The number of events remaining in the FIFO.
fifo_level = –1	A buffer overflow occurred.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_event_window_set...129
ds2211_inj_capture_mode_set...180

ds2211_multiwin_inj_fifo_read

Syntax

```
void ds2211_multiwin_inj_fifo_read(
      Int32 base,
      Int32 channel,
      Int32 *expected_count,
      Int32 state[][64],
      dsfloat pos[][64],
      Int32 *real_count,
      Int32 *fifo_level)
```

Include file

`ds2211.h`

Purpose

To read the injection pulse positions/durations captured during run time in the FIFO of the injection channels 1 … 8.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2211 Features 📖).

Description

The values of the specified channel are returned via the `pos` array. The `state` array contains information on whether the corresponding event was the leading or trailing edge of a pulse. The expected_count parameter specifies the number of events to be read or expected per event window. The number of events which could be read at all is returned via the real_count parameter per event window. If all expected events are read, the following events are not read within the current function call. You can read them with the next function call. The events are read from the capture FIFO and stored in a temporary internal buffer separated by channel numbers. The temporary buffer can store up to 64 events per channel. A buffer overflow occurs if the events are not read fast enough. A buffer overflow within the last function call can be checked by the fifo_level parameter.

The function supports the following injection capture modes:

Capture Modes	Description
DS2211_INJCAP_PULSE_POS	Only the angle positions of the captured events are read in degrees. Under certain circumstances pseudo edges are generated.
DS2211_INJCAP_PULSE_DUR, DS2211_INJCAP_PULSE_DUR_LOW	All start positions (in degree) of the pulses and the pulse durations (in seconds) are read. Each start position entry in the `pos` array follows the duration entry of the corresponding pulse. The resolution depends on the mode and can be either 4 µs or 1 µs (…DUR_LOW). Under certain circumstances pseudo edges are generated.

> **Note**
>
> If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description
DS2211_INJCAP_CH1	Injection capture channel 1
…	…
DS2211_INJCAP_CH8	Injection capture channel 8

expected_count Pointer to an array containing the number of events to be read for the specified event window. The maximum number must not exceed the internal FIFO size (64 events). The length of the array must be the number of event windows (see `ds2211_multi_eventwin_set` on page 131).

state Pointer to a two-dimensional array where the state of the injection events per event window are written. The order of the elements is state[index for event window][index for state]. Their meaning depends on the capture mode settings (active low or active high input signal, see `ds2211_inj_capture_mode_setup`).

Capture State	State	Description
DS2211_INJCAP_RISE_EDGE	0	Falling edge/active low
	1	Rising edge/active high
DS2211_INJCAP_FALL_EDGE	0	Rising edge/active high
	1	Falling edge/active low

pos Pointer to a two-dimensional array where either the angle positions (in degrees) or the angle positions and pulse durations (in seconds) of the captured injection pulses per event window are written. The order of the elements is pos[index for event window][index for position/duration].

real_count Address where the current number of returned events per event window is written. The `length` parameter can only be less than or equal to the `expected_count` parameter. If more data is stored in the internal FIFO than read and exceeds the maximum fifo size, a buffer overflow occurs. The buffer overflow can be recognized by the return value of the function.

fifo_level FIFO level of the temporary FIFO buffer. It represents the level/state of the FIFO after the previous read operations:

Return Value	Description
fifo_level = 0	The FIFO is empty, all events were read.
fifo_level > 0	The number of events remaining in the FIFO.
fifo_level = –1	A buffer overflow occurred.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_multiwin_inj_cap_read_var

Syntax

```
void ds2211_multiwin_inj_cap_read_var(
      Int32 base,
      Int32 channel,
      Int32 *real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64],
      Int32 *status,
      Int32 *counter)
```

Include file

`ds2211.h`

Purpose

To read the injection pulse positions/durations (start and end, or start and duration) from the current event capture window(s).

> **Note**
>
> Do not use the `ds2211_multiwin_inj_cap_read_var` function together with other injection capture reading functions, for example `ds2211_multiwin_inj_cap_read`.
>
> There is one exception to this rule: The `ds2211_multiwin_inj_cap_read_var` and `ds2211_mutiwin_inj_fifo_read` functions can be used together.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮).

Description

The fuel injection pulse positions and their durations captured in the last specified event windows on the specified channel are returned by the start_pos and end_pos arrays. The real_count parameter returns the number of events actually captured in the last specified event windows.

> **Note**
>
> - Input channels 7 and 8 are shared with PWM measurement. If you use these channels for PWM signal measurement you cannot use them for injection capture, refer to `ds2211_pwm_in` on page 85.
> - If the number of captured events is less than the number of expected events specified by `ds2211_inj_capture_mode_setup`, the function returns negative position or duration values for the missing events.
> - Events are captured up to the number of expected events. Additional events are ignored.

The function supports the following injection capture modes:

Function	Parameter
DS2211_INJCAP_PULSE_POS	All event positions including pseudo event generation are captured.
DS2211_INJCAP_PULSE_DUR	Start position and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_INJCAP_PULSE_DUR_LOW	Start position and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 deg) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Input channels INJ1 … INJ8 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 200mV. Four capture modes are supported by the complex comparator logic connected to INJ1 … INJ8. Thresholds A and B, and hysteresis B of INJ1 … INJ8, and capture mode are set by using `ds2211_apu_injection_cc_setup`. Threshold A of comparator INJ7 … INJ8 can also be set by using `ds2211_digin_threshold_set`.

The function combines the functionality of the event-window-based and continuous event capturing. Although the values are captured on an event-window basis, you get the current status via the status and counter parameters. For an example, refer to Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_INJCAP_CH1	Injection capture channel 1	1
…	…	…
DS2211_INJCAP_CH8	Injection capture channel 8	8

real_count Pointer to an array where the current numbers of returned events per event window are written. The length of the array must be the number of specified event capture windows (see `ds2211_multi_eventwin_set` on page 131). It returns the total number of pulses captured in the last complete event capture window. The value can be greater than the specified expected number of pulses. The value is updated at the end of each event capture window.

start_pos Pointer to a two-dimensional array where the angle positions (in degrees) at the beginning of the injection pulses per event window are written. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions (in degrees) at the end of the injection pulses or the pulse durations (in seconds) per event window are written. The order of the elements is end_pos[index for event window][index for end position/duration].

status Pointer to an array where the status of the currently captured events data are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

Value	Description
0	The update of data in the current event window is still running. The `counter` parameter is still being updated. The `start_pos` and `end_pos` arrays may contain a mix of old and new data.
1	Data in the current event window is no longer being updated. The `counter` parameter is not being updated. The `start_pos` and `end_pos` arrays contain only new data (Exception: In the current event window, either no pulses occurred, or fewer than in the previous event window).

The status is set to 0 when the first event of each event window occurs. The status is set to 1 if the number of expected pulses is reached or at the end of each event window (then the number of expected pulses was not reached in the event window). After initialization the value is 0.

counter Pointer to an array where the current event counts are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). The counter counts every edge occurring in the current event window until the number of expected pulses is reached. Additional pulses are ignored.

Negative/positive signs alternate to distinguish between leading and trailing edges.

Sign	Description
Negative	A leading edge is detected. The count value is incremented.
Positive	A trailing edge is detected. The count value is not incremented.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_inj_capture_mode_setup 181
ds2211_multiwin_inj_cap_read_var_abs 204
ds2211_multiwin_inj_cap_read_var_ext 201

ds2211_multiwin_inj_cap_read_var_ext

Syntax

```
void ds2211_multiwin_inj_cap_read_var(
      Int32 base,
      Int32 channel,
      Int32 *real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64],
      dsfloat pulse_dur[][64],
      Int32 *status,
      Int32 *counter)
```

Include file

`ds2211.h`

Purpose

To read the injection angle positions and pulse durations captured from the current event capture window.

Note

- This function can be used starting from board revision 3 and FPGA revision 3.
- Do not use the `ds2211_multiwin_inj_cap_read_var_ext` function together with other injection capture reading functions, for example, `ds2211_multiwin_inj_cap_read_ext`.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮).

Description

The fuel injection pulse positions and their durations captured in the last specified event windows on the specified channel are returned by the start_pos and end_pos arrays. The real_count parameter returns the number of events actually captured in the last specified event windows.

Note

- Input channels 7 and 8 are shared with PWM measurement. If you use these channels for PWM signal measurement you cannot use them for injection capture, refer to ds2211_pwm_in on page 85.
- If the number of captured events is less than the number of expected events specified by `ds2211_inj_capture_mode_setup`, the function returns negative position or duration values for the missing events.
- Events are captured up to the number of expected events. Additional events are ignored.

The function supports the following injection capture modes:

Function	Parameter
DS2211_INJCAP_POS_DUR_EXT	Start position, end position, and pulse duration with a duration scale of 250 ns, required if DS2211 is the time-base master.
DS2211_INJCAP_POS_DUR_EXT_LOW	Start position, end position, and pulse duration with a duration scale of 1 µs, required if DS2210 is the time-base master and DS2211 is the time-base slave.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the window border (for example, 0 deg) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Input channels INJ1 … INJ8 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 200 mV. Four capture modes are supported by the complex comparator logic connected to INJ1 … INJ8. Thresholds A and B, and hysteresis B of INJ1 … INJ8, and capture mode are set by using `ds2211_apu_injection_cc_setup`. Threshold A of comparator INJ7 … INJ8 can also be set by using `ds2211_digin_threshold_set`.

The function combines the functionality of the event-window-based and continuous event capturing. Although the values are captured on an event-window basis, you get the current status via the status and counter parameters. For an example, refer to Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_INJCAP_CH1	Injection capture channel 1	1
...	...	...
DS2211_INJCAP_CH8	Injection capture channel 8	8

real_count Pointer to an array where the current numbers of returned events per event window are written. The length of the array must be the number of specified event capture windows (see `ds2211_multi_eventwin_set` on page 131). It returns the total number of pulses captured in the last complete event capture window. The value can be greater than the specified expected number of pulses. The value is updated at the end of each event capture window.

start_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the leading edges of the injection pulses are written. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the trailing edges of the injection pulses are written. The order of the elements is end_pos[index for event window][index for end position].

pulse_dur Pointer to a two-dimensional array where the pulse durations (in seconds) of the injection pulses are written. The order of the elements is pulse_dur[index for event window][index for duration].

status Pointer to an array where the status of the currently captured events data are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

Value	Description
0	The update of data in the current event window is still running. The `counter` parameter is still being updated. The `start_pos`, `end_pos` and `pulse_dur` arrays may contain a mix of old and new data.
1	Data in the current event window is no longer being updated. The `counter` parameter is not being updated. The `start_pos`, `end_pos` and `pulse_dur` arrays contain only new data (Exception: In the current event window, either no pulses occurred, or fewer than in the previous event window).

The status is set to 0 when the first event of each event window occurs. The status is set to 1 if the number of expected pulses is reached or at the end of each event window (then the number of expected pulses was not reached in the event window). After initialization the value is 0.

counter Pointer to an array where the current event counts are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). The counter counts every edge occurring in the current event window until the number of expected pulses is reached. Additional pulses are ignored.

Negative/positive signs alternate to distinguish between leading and trailing edges.

Sign	Description
Negative	A leading edge is detected. The count value is incremented.
Positive	A trailing edge is detected. The count value is not incremented.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

DS2211 Board Revision (DS2211 Features 🕮)

Examples

Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 🕮)

References

ds2211_apu_injection_cc_setup	178
ds2211_digin_threshold_set	22
ds2211_inj_capture_mode_setup	181
ds2211_multiwin_inj_cap_read_var	198
ds2211_multiwin_inj_cap_read_var_abs	204
ds2211_pwm_in	85

ds2211_multiwin_inj_cap_read_var_abs

Syntax

```
void ds2211_multiwin_inj_cap_read_var(
      Int32 base,
      Int32 channel,
      Int32 *real_count,
      dsfloat start_pos[][64],
      dsfloat end_pos[][64],
      dsfloat start_time[][64],
      dsfloat end_time[][64],
      Int32 *status,
      Int32 *counter)
```

Include file

`ds2211.h`

Purpose

To read the injection angle positions and pulse durations captured from the current event capture window.

> **Note**
>
> - This function can be used starting from board revision 3 and FPGA revision 3.
> - Do not use the `ds2211_multiwin_inj_cap_read_var_abs` function together with other injection capture reading functions, for example, `ds2211_multiwin_inj_cap_read_abs`.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮).

Description

The fuel injection pulse positions and their durations captured in the last specified event windows on the specified channel are returned by the start_pos and end_pos arrays. The real_count parameter returns the number of events actually captured in the last specified event windows.

> **Note**
>
> - Input channels 7 and 8 are shared with PWM measurement. If you use these channels for PWM signal measurement you cannot use them for injection capture, refer to ds2211_pwm_in on page 85.
> - If the number of captured events is less than the number of expected events specified by `ds2211_inj_capture_mode_setup`, the function returns negative position or duration values for the missing events.
> - Events are captured up to the number of expected events. Additional events are ignored.

The function supports the following injection capture mode:

Function	Parameter
DS2211_INJCAP_ABSOLUTE	Start position, end position, start time, and end time are captured. All values are absolute values relative to a defined starting point. The default starting point is the start of the angular processing unit. You can define a new starting point at run time using the `DS2211_ABS_COUNTER_RESET` macro.

If the leading or trailing edge of an input pulse is outside the event window and the other edge is inside the window, the capture unit generates a pseudo edge at the respective window border. That means the pulse is assumed to start or end at the window border. The measured pulse corresponds to the part of the pulse that intersects with the window. If the window covers the entire position range of 0 … 720° (continuous injection), an input pulse overlapping the

window border (for example, 0 deg) will be detected as two separate pulses due to the pseudo edges at the window start and end positions.

Input channels INJ1 … INJ8 possess a complex comparator functionality. The complex comparator circuit contains 2 comparators (A and B) with independent setting options for the thresholds (threshold A 1 … 23.8 V, threshold B 1 … 22.65 V) and for the hysteresis of comparator B (0.2 … 2.4 V). The hysteresis of comparator A is fixed at 200mV. Four capture modes are supported by the complex comparator logic connected to INJ1 … INJ8. Thresholds A and B, and hysteresis B of INJ1 … INJ8, and capture mode are set by using `ds2211_apu_injection_cc_setup`. Threshold A of comparator INJ7 … INJ8 can also be set by using `ds2211_digin_threshold_set`.

The function combines the functionality of the event-window-based and continuous event capturing. Although the values are captured on an event-window basis, you get the current status via the status and counter parameters. For an example, refer to Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_INJCAP_CH1	Injection capture channel 1	1
…	…	…
DS2211_INJCAP_CH8	Injection capture channel 8	8

real_count Pointer to an array where the current numbers of returned events per event window are written. The length of the array must be the number of specified event capture windows (see `ds2211_multi_eventwin_set` on page 131). It returns the total number of pulses captured in the last complete event capture window. The value can be greater than the specified expected number of pulses. The value is updated at the end of each event capture window.

start_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the leading edges of the injection pulses are written. The order of the elements is start_pos[index for event window][index for start position].

end_pos Pointer to a two-dimensional array where the angle positions (in degrees) of the trailing edges of the injection pulses are written. The order of the elements is end_pos[index for event window][index for end position].

start_time Pointer to a two-dimensional array where the time stamps (in seconds) of the leading edges of the injection pulses are written. The order of the elements is start_time[index for event window][index for start time].

end_time Pointer to a two-dimensional array where the time stamps (in seconds) of the trailing edges of the injection pulses are written. The order of the elements is end_time[index for event window][index for time stamp].

status Pointer to an array where the status of the currently captured events data are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131).

Value	Description
0	The update of data in the current event window is still running. The `counter` parameter is still being updated. The `start_pos`, `end_pos`, `start_time` and `end_time` arrays may contain a mix of old and new data.
1	Data in the current event window is no longer being updated. The `counter` parameter is not being updated. The `start_pos`, `end_pos`, `start_time` and `end_time` arrays contain only new data (Exception: In the current event window, either no pulses occurred, or fewer than in the previous event window).

The status is set to 0 when the first event of each event window occurs. The status is set to 1 if the number of expected pulses is reached or at the end of each event window (then the number of expected pulses was not reached in the event window). After initialization the value is 0.

counter Pointer to an array where the current event counts are written. The length of the array must be the number of specified event windows (see `ds2211_multi_eventwin_set` on page 131). The counter counts every edge occurring in the current event window until the number of expected pulses is reached. Additional pulses are ignored.

Negative/positive signs alternate to distinguish between leading and trailing edges.

Sign	Description
Negative	A leading edge is detected. The count value is incremented.
Positive	A trailing edge is detected. The count value is not incremented.

Return value None

Execution times For information, refer to Function Execution Times on page 551.

Related topics

Basics

DS2211 Board Revision (DS2211 Features 🕮)

Examples

Example of Capturing a Multiple Event with the VAR APU Blockset (DS2211 Features 🕮)

References

DS2211_ABS_COUNTER_RESET	209
ds2211_apu_injection_cc_setup	178
ds2211_digin_threshold_set	22
ds2211_inj_capture_mode_setup	181
ds2211_multiwin_inj_cap_read_var	198
ds2211_multiwin_inj_cap_read_var_ext	201
ds2211_pwm_in	85

ds2211_injection_status_read

Syntax

```
void ds2211_injection_status_read(
      Int32 base,
      Int32 *channels,
      Int32 count,
      Int32 *states)
```

Include file

`ds2211.h`

Purpose

To read the current status (0/1) of the `count` injection capture inputs specified by the `channels` array.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2211 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channels Address where the array of length `count` containing the channel numbers (1 … 8) is stored. The following symbols are predefined:

Predefined Symbol	Description	Value
DS2211_INJCAP_CH1	Injection capture channel 1	1
…	…	…
DS2211_INJCAP_CH8	Injection capture channel 8	8

> **Note**
>
> Injection capture channel 7 and 8 conflict with PWM input channel 7 and 8.

count Number of channels to be read

states Address where the array of length `count` is stored. The array contains the resulting status values (0 or 1) of the specified capture input channels. The array must be allocated by the calling instance.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

DS2211_ABS_COUNTER_RESET

Macro

```
void DS2211_ABS_COUNTER_RESET(
    Int32 base)
```

Include file

`ds2211_apu.h`

Purpose

Clears the internal 40-bit time stamp counter (time base=250ns) and the internal 24-bit engine cycle counter. This macro lets you restart (reset) the absolute time stamp measurement and absolute engine position/angle measurement at every specified run-time point.

This macro defines a new starting point for measurement, the original angle position of the APU remains unchanged.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

Return value	None
Related topics	References ds2211_multiwin_ign_cap_read_abs.......... 157 ds2211_multiwin_ign_cap_read_var_abs.......... 172 ds2211_multiwin_inj_cap_read_abs.......... 190 ds2211_multiwin_inj_cap_read_var_abs.......... 204

Generating PHS-Bus Interrupts on Capture Events

Where to go from here

Information in this section

ds2211_cap_interrupt_setup

Syntax

```
ds2211_cap_interrupt_setup(
     Int32 base,
     Int32 mask)
```

Include file

`Ds2211.h`

Purpose

To enable leading and trailing edge interrupts of the 16 capture unit inputs (ignition and injection).

Description

Interrupts enabled by this function trigger PHS-bus interrupt line 0.

The leading and trailing edge for each of the 16 capture inputs can be specified to trigger the interrupt.

To use the function, the following preconditions must be fulfilled:

- The APU must be started, refer to `ds2211_apu_start` on page 106.
- An event window must be set for the trigger, refer to `ds2211_multi_eventwin_set` on page 131.
- The capture mode must be configured, refer to `ds2211_ign_capture_mode_setup` on page 139, `ds2211_inj_capture_mode_setup` on page 181, `ds2211_aux1_capture_mode_setup` on page 143, or `ds2211_aux2_capture_mode_setup` on page 147.

- A suitable threshold and mode must be set for the complex comparator, see `ds2211_apu_ignition_cc_setup` on page 135 or `ds2211_apu_injection_cc_setup` on page 178.

Note

Use this function only during initialization and before RTLib interrupts are enabled. Calling this function at run time leads to unpredictable results.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

mask Specifies the events for which an interrupt is triggered. Use the following predefined symbols.

For trailing edges:

- DS2211_IGNCAP1_TRAIL ... IGNCAP6_TRAIL
- DS2211_AUXCAP1_TRAIL, DS2211_AUXCAP2_TRAIL
- DS2211_INJCAP1_TRAIL ... INJCAP8_TRAIL

For leading edges:

- DS2211_IGNCAP1_LEAD ... IGNCAP6_LEAD
- DS2211_AUXCAP1_LEAD, DS2211_AUXCAP2_LEAD
- DS2211_INJCAP1_LEAD ... INJCAP8_LEAD

For trailing and leading edges of all ignition/auxiliary channels:

- DS2211_IGNCAP_ALL

For trailing and leading edges of all injection channels:

- DS2211_INJCAP_ALL

For trailing and leading edges of all ignition/auxiliary and injection channels:

- DS2211_CAP_ALL

You can combine predefined symbols using the logical OR operator.

Return value

None

Example

The following example shows how to set all injection events and the events of both auxiliary channels as interrupt triggers.

```
void main()
{
   ds2211_init(DS2211_1_BASE);
   ds2211_cap_interrupt_setup(DS2211_1_BASE, DS2211_INJCAP_ALL|
                              DS2211_AUXCAP1_TRAIL |
                              DS2211_AUXCAP2_TRAIL |
                              DS2211_AUXCAP1_LEAD  |
                              DS2211_AUXCAP2_LEAD);
   install_phs_int_vector(DS2211_1_BASE, 0, cap_int_function);
...
}
```

Related topics

References

ds2211_cap_interrupt_get

Syntax

```
UInt32 ds2211_cap_interrupt_get(
     Int32 base)
```

Include file

`ds2211.h`

Purpose

To find out which event has triggered capture interrupt INT0.

Description

This function reads the whole content of the interrupt source register (INT0EDG). If several events have occurred simultaneously, the register contains more than one event. The function should be used within the interrupt service routine (ISR) triggered by the interrupts.

Using this function clears the corresponding register.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

Return value

The function returns the content of the interrupt source register. The values are returned as bit masks in the range 0x00000000 … 0x0000FFFF. You can use the appropriate predefined symbols DS2211_IGNCAP1_TRAIL ... INJCAP8_LEAD (see mask on page 212) to test the bits.

Example

The following example shows how to determine the events which trigger the interrupt and store the register information.

```
void cap_int_function(void)
{
   UInt32  IntReg = 0 ;
   IntReg = ds2211_cap_interrupt_get(DS2211_1_BASE);
   if (IntReg & DS2211_IGNCAP1_TRAIL)
   {
        …
   }
}
```

ds2211_cap_interrupt_decode

Syntax

```
UInt32 ds2211_cap_interrupt_decode(
      UInt32 intreg)
```

Include file

`ds2211.h`

Purpose

To read the interrupt source register bit-wise and evaluate each single event that triggers an interrupt.

Description

You can use this function to identify each single event which has possibly triggered the capture interrupt. The function checks if the interrupt register contains any data and evaluates the register bit-wise in a FOR loop. Each time an event is identified its bit mask is returned by the function. When all the events have been read out, the function returns the code DS2211_NO_CAP_INT (0x0) to indicate that no more interrupts are pending.To reduce execution time, the function starts evaluating the interrupt register at the position where the previous interrupt event was found.

Using this function clears the register.

Parameters

intreg Content of the interrupt source register (INT0EDG)

Return value

The function returns each single event of the interrupt source register. The values are returned as bit masks in the range 0x00000000 … 0x80000000. You can use the appropriate predefined symbols DS2211_IGNCAP1_TRAIL ... INJCAP8_LEAD (see mask on page 212) to test the bits.

If no interrupt is pending, the function returns DS2211_NO_CAP_INT.

Example

The following example shows how to define the events which trigger the interrupt and store the register information.

```
void CaptureInterrupts(void)
{
    UInt32 IntReg = 0;
    UInt32 IntFlag   = 0;
    IntReg = ds2211_cap_interrupt_get(DS2211_1_BASE);
    do
    {
        IntFlag = ds2211_cap_interrupt_decode(IntReg);
        switch (IntFlag)
        {
            case DS2211_IGNCAP1_TRAIL:
              break;
            case DS2211_IGNCAP2_TRAIL:
              break;
            .
            .
            .
            default:
               break;
        }
    }while (DS2211_NO_CAP_INT != IntFlag)
}
void isr_t1(void)
{
   …   /* timer interrupt routine */
}
void main()
{   ds2211_init(DS2211_1_BASE);
    ds2211_cap_interrupt_setup(DS2211_1_BASE, DS2211_IGNCAP_ALL);
    install_phs_int_vector(DS2211_1_BASE, 0, CaptureInterrupts);
    RTLIB_SRT_START(TS, isr_t1);
    while(1)
    {
        RTLIB_BACKGROUND_SERVICE();
    }
}
```

Serial Interface Communication

Introduction

This section contains the generic functions for communication via a serial interface.

Where to go from here

Information in this section

Information in other sections

Serial Interface (DS2211 Features 🕮)
The board contains a universal asynchronous receiver and transmitter (UART) to communicate with external devices.

Basic Principles of Serial Communication

Where to go from here

Information in this section

Information in other sections

Serial Interface (DS2211 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 or larger 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

Basics

Basic Principles of Serial Communication......218

HowTos

How to Handle Subinterrupts in Serial Communication......219

How to Handle Subinterrupts in Serial Communication

Introduction

The interrupt functions must be used only in handcoded applications. Using them in Simulink applications (user code or S-functions) conflicts with the internal interrupt handling.

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 218).
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.

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......218

References

dsser_subint_enable......249
dsser_subint_handler_inst......248
dsser_subint_handler_t......226
dsserChannel......227

Example of a Serial Interface Communication

Example

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

```
#include <brtenv.h>
void callback(dsserChannel* serCh, UInt32 subint)
{
   UInt32 count;
   UInt8 data[32];
   switch (subint)
   {
      case DSSER_TRIGGER_LEVEL_SUBINT:
         msg_info_set(0,0,”DSSER_TRIGGER_LEVEL_SUBINT”);
         dsser_receive(serCh,32,data,&count);
         dsser_transmit(serCh,count,data,&count);
         break;
      case DSSER_TX_FIFO_EMPTY_SUBINT:
         msg_info_set(0,0,”DSSER_TX_FIFO_EMPTY_SUBINT”);
         break;
      default:
         break;
   }
}
main()
{
   dsserChannel* serCh;
   init();
   ds2211_init(DS2211_1_BASE);

/* allocate a new 1024 byte SW-FIFO */
   serCh = dsser_init(DS2211_1_BASE, 0, 1024);
   dsser_subint_handler_inst(serCh,
         (dsser_subint_handler_t)callback);
   dsser_subint_enable(serCh,
         DSSER_TRIGGER_LEVEL_SUBINT_MASK |
         DSSER_TX_FIFO_EMPTY_SUBINT_MASK);
/* config and start the UART */
   dsser_config(serCh, DSSER_FIFO_MODE_OVERWRITE,
         9600, 8, DSSER_1_STOPBIT, DSSER_NO_PARITY,
         DSSER_14_BYTE_TRIGGER_LEVEL, 32, DSSER_RS232);
   RTLIB_INT_ENABLE();
   for(;;)
   {
      RTLIB_BACKGROUND_SERVICE();
   }
}
```

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

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

> **Note**
>
> The data type contains the value of the UART's register.
> The register conforms to a standard 16550 UART such as the TEXAS INSTRUMENTS TL16C550C. For further information, refer to http://www.ti.com.

Members

The structure provides the following members:

Member	Description
`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..245

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

> **Note**
>
> The data type contains the value of the UART's register.
> The register conforms to a standard 16550 UART such as the TEXAS INSTRUMENTS TL16C550C. For further information, refer to http://www.ti.com.

Members

The structure provides the following members.

Member	Description
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... 245

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

Note

The data type contains the value of the UART's register.
The register conforms to a standard 16550 UART such as the TEXAS INSTRUMENTS TL16C550C. For further information, refer to http://www.ti.com.

Members

The structure provides the following members.

Member	Description
`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..245

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 on page 219 or dsser_subint_handler_inst on page 248).

Members

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 230).

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 218).

Predefined Symbol	Meaning
DSSER_TX_FIFO_EMPTY_SUBINT	Interrupt triggered when the transmit buffer is empty.
DSSER_RECEIVER_LINE_SUBINT	Line status interrupt of the UART.
DSSER_MODEM_STATE_SUBINT	Modem status interrupt of the UART.
DSSER_NO_SUBINT	Flag that is sent after the last triggered subinterrupt.

Related topics

Basics

Trigger Levels........ 218

References

dsser_init........ 230

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. Refer to `dsser_ISR` on page 222.

lineStatusReg Line status register. Refer to `dsser_LSR` on page 224.

modemStatusReg Modem status register. Refer to `dsser_MSR` on page 225.

Related topics

References

dsser_status_read.. 245

Generic Serial Interface Communication Functions

Where to go from here

Information in this section

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. First, initialize the processor board, then the I/O boards, and then the serial interface.

Parameters

base Specifies the base address of the serial interface. This value has to be set to DS2211_y_BASE, with y as a consecutive number within the range of 1 … 16. For example, if there is only one DS2211 board, use DS2211_1_BASE.

channel Specifies the number of the channel to 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. 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.
101	Warning	x, ch=y, Mixed usage of high and low level API!	It is not allowed to use the generic functions (high-level access functions) and the low-level access functions of the serial interface on the same channel. It is recommended to use only the generic functions.
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 `channel` parameter is out of range.
700	Error	x, ch=y, Buffersize: Illegal	The `fifo_size` parameter is out of range.

Related topics

Basics

Basic Principles of Serial Communication...218

Examples

Example of a Serial Interface Communication...220

References

Data Types for Serial Communication..222
dsser_config...232
dsser_free...231

dsser_free

Syntax

```
Int32 dsser_free(dsserChannel*serCh)
```

Include file

`dsser.h`

Purpose

To close a serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 230).

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. The specified serial interface is closed. Its memory for the buffer is freed and the interrupts are released. A serial interface can be created again using the `dsser_init` function.
DSSER_TX_FIFO_NOT_EMPTY	The serial interface is not closed, because the transmit buffer is not empty.
DSSER_CHANNEL_INIT_ERROR	There is no serial interface to be closed (serCh == NULL).

Related topics

Basics

Basic Principles of Serial Communication...218

References

dsser_init...230

dsser_config

Syntax

```
void dsser_config(
      dsserChannel* 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

`dsser.h`

Purpose

To configure and start the serial interface.

Note

- This function starts the serial interface. Therefore, 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 on page 230).

fifo_mode Specifies the mode of the receive buffer (see Serial Interface (DS2211 Features 🕮)):

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 (DS2211 Features 🕮).

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 218). 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 Trigger Levels on page 218):

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
101	Warning	x, ch=y, Mixed usage of high and low level API!	It is not allowed to use the generic functions (high-level access functions) and the low-level access functions of the serial interface on the same channel. It is recommended to use only the generic functions.
601	Error	x, serCh: The UART channel was not initialized.	The `dsser_config` function was called before the serial interface was initialized with `dsser_init`.
602	Error	x, ch=y, baudrate: Illegal!	The `baudrate` parameter is out of range.
603	Error	x, ch=y, databits: Use range 5 … 8 bits!	The `databits` parameter is out of range.
604	Error	x, ch=y, stopbits: Illegal number (1-2 bits allowed)!	The `stopbits` parameter is out of range.

ID	Type	Message	Description
605	Error	x, ch=y, parity: Illegal parity!	The `parity` parameter is out of range.
606	Error	x, ch=y, trigger_level: Illegal UART trigger level!	The `uart_trigger_level` parameter is out of range.
607	Error	x, ch=y, trigger_level: Illegal user trigger level!	The `user_trigger_level` parameter is out of range.
608	Error	x, ch=y, fifo_mode: Use range 0 … (fifo_size-1) bytes!	The `uart_mode` 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........................218

Examples

Example of a Serial Interface Communication........................220

References

dsser_init........................230

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 on page 230).

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 was able to 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 to 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 might be overloaded. Do not poll this function because it may cause 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..218

Examples

Example of a Serial Interface Communication..220

References

dsser_init..230
dsser_transmit_fifo_level..243

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 on page 230).

datalen Specifies the number of data bytes to 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 `fifo_mode` adjusted with `dsser_config`: ▪ `fifo_mode = DSSER_FIFO_MODE_BLOCKED` Not all new data could be placed in the FIFO. ▪ `fifo_mode = DSSER_FIFO_MODE_OVERWRITE` 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 might be overloaded. Do not poll this function because it might cause 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....................218

Examples

Example of a Serial Interface Communication....................220

References

dsser_init....................230

dsser_receive_term

Syntax

```
Int32 dsser_receive_term(
      dsserChannel* serCh,
      UInt32 datalen,
      UInt8* data,
      UInt32* count,
      const UInt8 term)
```

Include file

`dsser.h`

Purpose

To receive data through the serial interface.

Description

This function 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 completed.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 230).

datalen Specifies the number of data bytes to 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.

term Specifies the character that terminates the reception of bytes.

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 `fifo_mode` adjusted with `dsser_config`: ▪ `fifo_mode = DSSER_FIFO_MODE_BLOCKED` Not all new data could be placed in the FIFO. ▪ `fifo_mode = DSSER_FIFO_MODE_OVERWRITE` 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 might be overloaded. Do not poll this function because it might cause an endless loop.

Example

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

```
UInt8 data[4];
UInt32 count;
Int32 error;
error = dsser_receive_term(serCh, 4, data, &count, '\r');
```

Related topics

Basics

Basic Principles of Serial Communication........218

References

dsser_init........230

dsser_fifo_reset

Syntax

```
Int32 dsser_fifo_reset(dsserChannel* serCh)
```

Include file

`dsser.h`

Purpose

To reset the serial interface.

Description

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

> **Note**
>
> If you want to continue 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 on page 230).

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 might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics

Basics

Basic Principles of Serial Communication...218

References

dsser_enable...241
dsser_init...230

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 dsser_init on page 230).

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 might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics

Basics

Basic Principles of Serial Communication.............218

References

dsser_disable.............241
dsser_init.............230

dsser_disable

Syntax

```
Int32 dsser_disable(const dsserChannel* serCh)
```

Include file

`dsser.h`

Purpose

To disable the serial interface.

Description

The serial interface stops transmitting data, incoming data is no longer stored in the receive buffer and the UART subinterrupts are disabled.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 230).

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 might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics

Basics

Basic Principles of Serial Communication....218

References

dsser_enable....241
dsser_init....230

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 to get all error flags.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 230).

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....................218

References

dsser_config....................232
dsser_init....................230

dsser_transmit_fifo_level

Syntax

```
Int32 dsser_transmit_fifo_level(const dsserChannel* serCh)
```

Include file

`dsser.h`

Purpose

To get the number of bytes in the transmit buffer.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 230).

Return value

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

Related topics

Basics

Basic Principles of Serial Communication..218

References

dsser_init...230
dsser_receive_fifo_level...244

dsser_receive_fifo_level

Syntax

```
Int32 dsser_receive_fifo_level(const dsserChannel* serCh)
```

Include file

`dsser.h`

Purpose

To get the number of bytes in the receive buffer.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 230).

Return value

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

Related topics

Basics

Basic Principles of Serial Communication..218

References

dsser_init...230
dsser_transmit_fifo_level..243

dsser_status_read

Syntax

```
Int32 dsser_status_read(
      dsserChannel*serCh,
      const UInt8 register_type)
```

Include file

dsser.h

Purpose

To read the value of one or more status registers and to store the values in the appropriate fields of the channel structure.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 230).

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 dsser_ISR data type.
DSSER_STATUS_LSR	Line status register, see dsser_ISR data type.
DSSER_STATUS_MSR	Modem status register, see dsser_ISR data type.

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 might be overloaded. Do not poll this function because it might cause 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.......... 218

References

dsser_init.......... 230
dsser_ISR.......... 222
dsser_LSR.......... 224
dsser_MSR.......... 225

dsser_handle_get

Syntax

```
dsserChannel* dsser_handle_get(
   UInt32 base,
   UInt32 channel)
```

Include file

`dsser.h`

Purpose

To check whether the serial interface is in use.

Parameters

base Specifies the base address of the serial interface. This value has to be set to DS2211_y_BASE, with y as a consecutive number within the range of 1 … 16. For example, if there is only one DS2211 board, use DS2211_1_BASE.

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

Return value

This function returns:

- NULL if the specified serial interface is not used.
- A pointer to the serial channel structure of the serial interface that has been created by using the `dsser_init` function.

Related topics

Basics

References

dsser_set

Syntax

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

Include file

`dsser.h`

Purpose

To set a property of the UART.

Description

The DS2211 board is delivered with a standard quartz working with the frequency of 1.8432 · 10^6 Hz. You can replace this quartz with another one with a different frequency. Then you have to set the new quartz frequency using `dsser_set` followed by executing `dsser_config`.

Note

You must execute `dsser_config` after `dsser_set`; otherwise `dsser_set` has no effect.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 230).

type Specifies the property to be changed (`DSSER_SET_UART_FREQUENCY`).

value_p Specifies the pointer to a UInt32-variable with the new value, for example, a variable which contains the quartz frequency.

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 might be overloaded. Do not poll this function because it might cause an endless loop.

Example

This example sets a new value for the frequency.

```
UInt32 freq = 1843200;            /* 1.8432 MHz */
Int32 error;
error = dsser_set(serCh, DSSER_SET_UART_FREQUENCY, &freq);
```

Related topics

Basics

Basic Principles of Serial Communication..218

References

dsser_config..232
dsser_init..230

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 on page 249).

> **Note**
>
> The interrupt functions must 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 on page 230).

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

Examples

References

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 on page 230).

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 the user trigger level is reached (see Trigger Levels on page 218)
`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 might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics

Basics

Basic Principles of Serial Communication.......218

Examples

Example of a Serial Interface Communication.......220

References

dsser_init.......230
dsser_subint_disable.......250
dsser_subint_handler_inst.......248

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 on page 230).

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 218)
`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 might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics

Basics

Basic Principles of Serial Communication.......... 218

References

dsser_init.......... 230
dsser_subint_enable.......... 249
dsser_subint_handler_inst.......... 248

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..218

References

dsser_bytes2word..253
dsser_transmit..235
dsser_transmit_fifo_level..243

dsser_bytes2word

Syntax

```
UInt32* dsser_bytes2word(
      UInt8* bytes_p,
      UInt32* word_p,
      const int bytesInWord)
```

Include file

dsser.h

Purpose

To convert a byte array with a maximum of 4 elements into a single word.

Parameters

bytes_p Specifies the pointer to the input byte array.

word_p Specifies the pointer to the converted word.

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

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..218

References

dsser_receive..237
dsser_receive_fifo_level...244
dsser_word2bytes...252

Single Edge Nibble Transmission (SENT)

Introduction

You can use a DS2211 as SENT receiver or SENT transmitter. SENT is a protocol used between sensors and ECUs to transmit data of high resolution sensors.

Where to go from here

Information in this section

Information in other sections

Single Edge Nibble Transmission (SENT) Support (DS2211 Features 📖)
Provides basic information on the SENT protocol and information how you can use the protocol on a DS2211 board.

Configuring a SENT Transmitter

Introduction

There are five independent SENT transmitters on a DS2211 board. You can use the following functions to configure a SENT transmitter.

Where to go from here

Information in this section

Information in other sections

Implementing SENT Transmitters Using RTLib Functions (DS2211 Features 🕮)
Provides information on the parameters for a SENT transmitter and explains how you can implement it in a handcoded model.

ds2211_sent_tx_init

Syntax

```
void ds2211_sent_tx_init(
   Int32    base,
   UInt32   channel,
   UInt32   nibble_count)
```

Include file

`ds2211.h`

Purpose

To initialize a SENT transmitter of the specified channel.

Description

The function initializes a SENT transmitter and specifies the number of nibbles per SENT message, including the status nibble and the CRC nibble. The configuration parameters are reset and remaining messages in the transmit FIFO are deleted. This function must be executed before using the SENT transmitter of specified channel and before configuring the SENT transmitter by calling the `ds2211_sent_tx_config` or `ds2211_sent_tx_pause_mode` function.

The SENT transmitters shares I/O pins with the bit I/O unit. It is not possible to use bit I/O unit and SENT functionality on the same channel.

SENT TX Channel	I/O Pin
Channel 1	DIG_OUT1
Channel 2	DIG_OUT2
Channel 3	DIG_OUT3
Channel 4	DIG_OUT4
Channel 5	DIG_OUT5

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for transmitting SENT messages (1 ... 5). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3
Channel 5	`DS2211_SENT_CHANNEL5`	4

nibble_count Sets the number of nibbles used in every SENT message including the status nibble and the CRC nibble within the range 1 ... 217 (without pause pulse) or 1 ... 210 (with pause pulse).

Return value

None

Example

```
/* Initialize SENT transmitter channel 1 with 6 nibbles per message       */
ds2211_sent_tx_init(DS2211_1_BASE,        /* DS2211 base address          */
                    DS2211_SENT_CHANNEL1, /* select channel 1             */
                    6);                   /* 6 nibbles per message        */
```

Message

Type	Message	Description
Error	*ds2211_sent_tx_init*(board_offset): Feature is not supported by DS2211 Board/PAL revision. Refer to documentation	This function is not supported by the board/FPGA revision of the DS2211. For details, refer to Using the SENT Protocol on a DS2211 (DS2211 Features 📖).

Related topics

References

ds2211_sent_tx_pause_mode

Syntax

```
void ds2211_sent_tx_pause_mode(
   Int32    base,
   UInt32   channel,
   UInt32   pause_mode)
```

Purpose

To enable or disable pause pulse generation.

Include file

`ds2211.h`

Description

This function sets the pause pulse mode of the transmitter. You can enable and disable the pause pulse.

If the pause pulse is enabled, the transmitter appends a pause pulse at the end of every message. The pause pulse length is defined for each message using the **`ds2211_sent_tx_transmit_pause`** function.

The **`ds2211_sent_tx_pause_mode`** function must be called after initialization of the transmitter using **`ds2211_sent_tx_init`** and before transmitting messages using **`ds2211_sent_tx_transmit_pause`**.

If you enable the pause pulse mode, you can use the **`ds2211_sent_tx_transmit_pause`** function to specify an array of pause pulse length for the **`pause`** parameter. Refer to `ds2211_sent_tx_transmit_pause` on page 263. Do not use the **`ds2211_sent_tx_transmit`** function if the pause pulse mode is enabled.

The pause pulse mode is disabled by default (when this function is not called).

It depends on the board/FPGA revision whether the pause pulse mode is supported. Refer to Using the SENT Protocol on a DS2211 (DS2211 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for transmitting SENT messages (1 ... 5). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3
Channel 5	`DS2211_SENT_CHANNEL5`	4

pause_mode Sets the pause mode of the transmitter. The following modes can be specified:

Predefined Symbol	Value	Description
`DS2211_SENT_DISABLE_PAUSE_PULSE`	0	Disables the pause pulse mode (default).
`DS2211_SENT_ENABLE_PAUSE_PULSE`	1	Enables the pause pulse mode.

Return value

None

Example

```
ds2211_sent_tx_pause_mode(DS2211_1_BASE,              /* base address    */
                          DS2211_SENT_CHANNEL1,           /* Channel 1       */
                          DS2211_SENT_ENABLE_PAUSE_PULSE); /* enable pause
                                                             pulse generation*/
```

Related topics

References

ds2211_sent_tx_init..256
ds2211_sent_tx_transmit_pause..263

ds2211_sent_tx_config

Syntax

```
void ds2211_sent_tx_config(
   Int32    base,
   UInt32   channel,
   UInt32   low_tics,
   UInt32   zero_nibble_high_tics,
   UInt32   sync_high_tics,
   UInt32   autorepeat)
```

Purpose

To configure the SENT transmitter of the specified channel.

Include file

`ds2211.h`

Description

The function configures the SENT transmitter of the specified channel. It sets the number of tick periods used for a SENT pulse and a SENT message. The autorepeat mode can be selected by `autorepeat` parameter. This function can be used during run time to simulate a sensor working out of specification. The configuration must be set before writing the first message to the transmitter buffer by calling the `ds2211_sent_tx_transmit_pause` function, but can be updated during run time. Changes of the number of tick periods per pulse take effect at the beginning of a new pulse.

Note

The resulting pulse length under consideration of the tick period set by `ds2211_sent_set_tx_tic_period` must not exceed the range of allowed pulse length of minimum 2 µs.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for transmitting SENT messages (1 ... 5). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3
Channel 5	`DS2211_SENT_CHANNEL5`	4

low_tics Specifies the number of tick periods used for a SENT low pulse within the range 1 ... 15. The parameter is saturated to its limits.

zero_nibble_high_tics Specifies the number of tick periods used for a SENT nibble high pulse and a pause high pulse with a value of 0 within the range 1 ... 15. The parameter is saturated to its limits.

sync_high_tics Specifies the number of tick periods used for a SENT synchronization high pulse within the range 1 ... 255. The parameter is saturated to its limits.

autorepeat Enables or disables the automatic repeat mode of the SENT transmitter. The autorepeat mode can be enabled to ensure continuos transmission of messages.

Symbol	Description
`DS2211_SENT_DISABLE_AUTO_REPEAT`	Disables the autorepeat mode. Every message is transmitted one time. If the transmit buffer runs empty, the transmission stops and the output stays high until the next message is written to the buffer.
`DS2211_SENT_ENABLE_AUTO_REPEAT`	Enables the autorepeat mode. To avoid intermission of SENT transmission the last written message is repeated when the transmit buffer runs empty. This message is repeated until new messages are written to the buffer.

Return value

None

Example

```
/* Configure SENT transmitter timings and enable auto repeat */
ds2211_sent_tx_config(DS2211_1_BASE,        /* DS2211 base address   */
                      DS2211_SENT_CHANNEL1,  /* select channel 1      */
                      5,                     /* low ticks             */
                      7,                     /* zero nibble high ticks*/
                      51,                    /* sync high ticks       */
                      DS2211_SENT_ENABLE_AUTO_REPEAT);
```

Related topics

References

ds2211_sent_set_tx_tic_period..261
ds2211_sent_tx_transmit_pause...263

ds2211_sent_set_tx_tic_period

Syntax

```
void ds2211_sent_set_tx_tic_period(
   Int32    base,
   UInt32   channel,
   dsfloat  tic_period)
```

Include file

`ds2211.h`

Purpose

To set the tick period for the SENT transmitter of the specified channel.

Description

The function sets the actual used tick period for the SENT transmitter of the specified channel in seconds. This can be used to simulate clock drift of a SENT sensor. The new value takes effect at the beginning of the next tick period.

The tick period must be set before transmitting the first SENT message.

> **Note**
>
> The resulting pulse length under consideration of the tick period set by **`ds2211_sent_set_tx_tic_period`** must not exceed the range of allowed pulse length of minimum 2 µs.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for transmitting SENT messages (1 ... 5). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3
Channel 5	`DS2211_SENT_CHANNEL5`	4

tic_period Specifies the tick period for the SENT transmitter in seconds. The range depends on the board/FPGA revision of the DS2211: 500 ns ... 51.1875 µs or 500 ns ... 204.787 µs. For details of the board/FPGA revision, refer to Using the SENT Protocol on a DS2211 (DS2211 Features 🕮). The resolution is 12.5 ns. The parameter is saturated to its limits.

Return value

None

Example

```
/* Set tick period to 3µs */
ds2211_sent_set_tx_tic_period(DS2211_1_BASE,         /* DS2211 base address  */
                              DS2211_SENT_CHANNEL1,  /* select channel 1     */
                              3e-6);                 /* tick period          */
```

Related topics

Basics

Using the SENT Protocol on a DS2211 (DS2211 Features 🕮)

ds2211_sent_tx_transmit_pause

Syntax

```
UInt32 ds2211_sent_tx_transmit_pause(
   Int32    base,
   UInt32   channel,
   Int8    *data,
   UInt32   len,
   UInt32  *count,
   Int16   *pause)
```

Purpose

To write messages including a pause pulse (optional) to the transmit buffer (transmit FIFO) of the specified channel.

Include file

`ds2211.h`

Description

The function writes messages to the transmit buffer of specified SENT channel. The number of messages to write from `data` to the transmit FIFO is specified by the `len` parameter. The effective successfully written number of messages is returned by the `count` parameter. The values of `len` and `count` differ if not all messages were written to the transmit buffer because of a completely filled write buffer. The return value indicates if not all messages were written. The size of the transmit FIFO depends on the number of nibbles per message specified by `nibble_count` parameter during initialization. If the pause pulse mode is enabled by `ds2211_sent_tx_pause_mode`, the `pause` parameter defines a vector of pause pulses with one pause pulse per message. The length of this vector must match the specified number of messages by the `len` parameter. If the pause pulse mode is disabled, the `pause` parameter is ignored, you can set it to `"NULL"`, for example.

The number of messages that can be stored in the transmit FIFO can be calculated as follows:

If pause pulse mode is disabled:

```
Max_FIFO_Entries = RoundDown (63 / RoundUp(nibble_count / 7))
```

If pause pulse mode is enabled:

`Max_FIFO_Entries = RoundDown(63/(RoundUp(nibble_count/7) + 1))`

Nibbles per Message	FIFO Entries in Messages	
	Without Pause Pulse	With Pause Pulse
1 … 7	63	31
8 … 14	31	21
15 … 21	21	15
22 … 28	15	12
…	…	…

The actual fill level of the transmit FIFO can be read by the `ds2211_sent_tx_fifo_state` function.

The transmitter supports simulation of missing nibbles in a SENT message. To mark a missing nibble, the value is set to `DS2211_SENT_MISSING_NIBBLE` (-128). The transmitter will ignore marked nibbles in a message. It is not possible to ignore all nibbles in a message. If all nibbles are marked as missing, the entire message including the pause pulse is ignored.

A pause pulse can be marked as missing, using `DS2211_SENT_MISSING_PAUSE` (-32768). If marked, the pause pulse is ignored.

The pause pulse value is generated like any other nibble value. The `tic` parameters defined by `ds2211_sent_tx_config` are also relevant for the pause pulse. So, the low pulse length is defined by `low_tics`, the high pulse length is defined by `zero_nibble_high_tics` and the value of `pause`. A resulting pulse length for pause pulse is calculated as:

```
HIGH_LENGTH = (ZERO_NIBBLE_HIGH_TICS + PAUSE_VALUE) *
TIC_PERIOD
```

```
LOW_LENGTH = LOW_TICS * TIC_PERIOD
```

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

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for transmitting SENT messages (1 ... 5). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3
Channel 5	`DS2211_SENT_CHANNEL5`	4

data Specifies a pointer to the buffer holding the messages to be sent. The minimum length of the array must be `nibble_count · len`. The expected format of the data array is as follows (*nibble_count* is the number of nibbles per message):

Position in Data Buffer	Message / Nibble
tx_data[0]	Message 1 / Nibble 1
tx_data[1]	Message 1 / Nibble 2
...	...
tx_data[*nibble_count* - 1]	Message 1 / Nibble *nibble_count*
tx_data[*nibble_count*]	Message 2 / Nibble 1
…	…
tx_data[2 · *nibble_count* - 1]	Message 1 / Nibble *nibble_count*
…	…
tx_data[(*msg* - 1) · *nibble_count* + *nib*-1]	Message *msg* / Nibble *nib*

The range is 0 … 15 and -128

len Specifies the number of SENT messages hold in `data` buffer within the range 1 … 63. These messages are written to the transmit FIFO if not running full.

count Returns the number of messages successfully written to the FIFO. If this value differs from the specified number of messages to sent by `len`, not all messages are written to the FIFO. This is reported by the return value.

pause Specifies a pointer to the buffer holding the pause pulses. One pause pulse per message must be included in the buffer. The minimum length of the array must be `len`. The expected format of the data array is as follows:

Message	Position in Pause Buffer
Pause of message 1	pause[0]
Pause of message 2	pause[1]
…	…
Pause of message m	pause[m-1]

Return value

UInt32 The return value indicates the success of the write operation.

Value	Description
0	Write operation was successful. All messages were written to the FIFO.
1	Not all messages could be written to the FIFO, because the FIFO runs full. The `count` parameter indicates the number of written messages.

Example

```
Int8 tx_msg[] = {1,  2,  3,  4,  5,  6, /* 2 messages with 6 nibbles to send */
                 7,  8,  9, 10, 11, 12};
Int16 tx_pause[] = {100, 200};                     /* 2 pause pulses to send */
UInt32 tx_error;                                 /* recognize transmit error */
UInt32 tx_count;                           /* number of transmitted messages */
/* send 2 messages */
tx_error = ds2211_sent_tx_transmit_pause(DS2211_1_BASE,     /* base address   */
                         DS2211_SENT_CHANNEL1,                /* Channel 1    */
                         tx_msg,                    /* pointer to message buffer */
                         2,                                  /* write 2 messages */
                         &tx_count;                /* number of written messages */
                         tx_pause);                   /* pointer to pause buffer */
```

Message

Type	Message	Description
Error	ds2211_sent_tx_transmit_pause (*board_offset*): Ch *channel*: Function was interrupted by itself	The transmit function was interrupted by another transmit function for the same SENT channel. This can occur if you call SENT transmitter functions for the same channel in different tasks or interrupt service routines.

Related topics

References

ds2211_sent_tx_config.. 260
ds2211_sent_tx_fifo_state.. 269
ds2211_sent_tx_pause_mode.. 258

ds2211_sent_tx_transmit (obsolete)

Syntax

```
UInt32 ds2211_sent_tx_transmit(
   UInt32   base,
   UInt32   channel,
   Int8    *data,
   UInt32   len,
   UInt32  *count)
```

Purpose

To write messages to the transmit buffer (transmit FIFO) of the specified channel.

Note

This function is obsolete, use the `ds2211_sent_tx_transmit_pause` function.

Include file

`ds2211.h`

Description

The function writes messages to the transmit buffer of specified SENT channel. The number of messages to write from `data` to the transmit FIFO is specified by the `len` parameter. The effective successfully written number of messages is returned by the `count` parameter. The values of `len` and `count` differ if not all messages were written to the transmit buffer because of a completely filled write buffer. The return value indicates if not all messages were written. The size of the transmit FIFO depends on the number of nibbles per message specified by `nibble_count` parameter during initialization.

The number of messages that can be stored in the transmit FIFO can be calculated as follows:

```
Max_FIFO_Entries = RoundDown (63 / RoundUp(nibble_number / 7))
```

Nibbles per Message	FIFO Entries in Messages
1 … 7	63
8 … 14	31
15 … 21	21
22 … 28	15
…	…

The actual fill level of the transmit FIFO can be read by the `ds2211_sent_tx_fifo_state` function.

The transmitter supports simulation of missing nibbles in a SENT message. To mark a missing nibble, the value is set to `DS2211_SENT_MISSING_NIBBLE` (-128). The transmitter will ignore marked nibbles in a message. It is not possible to ignore all nibbles in a message. If all nibbles are marked as missing, the entire message is ignored.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for transmitting SENT messages (1 ... 5). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3
Channel 5	`DS2211_SENT_CHANNEL5`	4

data Pointer to the buffer holding the messages to be sent. The minimum length of the array must be `nibble_count · len`. The expected format of the data array is as follows (*nibble_count* is the number of nibbles per message):

Position in Data Buffer	Message / Nibble
tx_data[0]	Message 1 / Nibble 1
tx_data[1]	Message 1 / Nibble 2
…	…
tx_data[*nibble_count* - 1]	Message 1 / Nibble *nibble_count*
tx_data[*nibble_count*]	Message 2 / Nibble 1
…	…
tx_data[2 · *nibble_count* - 1]	Message 1 / Nibble *nibble_count*
…	…
tx_data[(*msg* - 1) · *nibble_count* + *nib*-1]	Message *msg* / Nibble *nib*

The range is 0 … 15 and -128.

len Specifies the number of SENT messages hold in `data` buffer within the range 1 … 63. These messages will be written to the transmit FIFO, if not running full.

count Returns the number of messages successfully written to the FIFO. If this value differs from the specified number of messages to sent by `len`, not all messages could be written to the FIFO. This is reported by the return value.

Return value

UInt32 The return value indicates the success of the write operation.

Value	Description
0	Write operation was successful. All messages were written to the FIFO.
1	Not all messages could be written to the FIFO, because the FIFO runs full. The `count` parameter indicates the number of written messages.

Example

```
Int8 tx_msg[] = {1,  2,  3,  4,  5,  6,   /* 2 messages with 6 nibbles to send */
                 7,  8,  9, 10, 11, 12};
UInt32 tx_error;                            /* recognize transmit error */
UInt32 tx_count;                            /* number of transmitted messages */
/* send 2 messages */
tx_error = ds2211_sent_tx_transmit(DS2211_1_BASE,            /* base address   */
                         DS2211_SENT_CHANNEL1,                 /* Channel 1    */
                         tx_msg,                 /* pointer to message buffer */
                         2,                                /* write 2 messages */
                         &tx_count);            /* number of written messages */
```

Message

Type	Message	Description
Error	ds2211_sent_tx_transmit(*board_offset*): Ch *channel*: Function was interrupted by itself	The transmit function was interrupted by another transmit function for the same SENT channel. This can occur if you call SENT transmitter functions for the same channel in different tasks or interrupt service routines.

Related topics

References

ds2211_sent_tx_fifo_state 269
ds2211_sent_tx_transmit_pause 263

ds2211_sent_tx_fifo_state

Syntax

```
UInt32 ds2211_sent_tx_fifo_state(
   Int32    base,
   UInt32   channel)
```

Include file

`ds2211.h`

Purpose

To return the actual FIFO fill level of the transmit FIFO of the specified channel.

Description

The function returns the current transmitter FIFO fill level of the specified channel. The returned value defines the number of messages, that have been already stored in the FIFO, waiting to be sent. This information is intended to calculate the time required for the next write operation before the transmitter FIFO runs empty.

To calculate the number of messages, that can be written to the FIFO without causing an overflow use the following equation:

```
Free_FIFO_Entries = Max_FIFO_Entries - FIFO_Fill_Level
```

Calculate `Max_FIFO_Entries` as follows:

If the pause pulse mode is disabled:

```
Max_FIFO_Entries = RoundDown (63 / RoundUp(nibble_count / 7))
```

If the pause pulse mode is enabled:

```
Max_FIFO_Entries = RoundDown(63/(RoundUp(nibble_count/7) + 1))
```

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for transmitting SENT messages (1 ... 5). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3
Channel 5	`DS2211_SENT_CHANNEL5`	4

Return value

UInt32 Actual fill level of the transmit FIFO (number of messages).

Example

```
UInt32 tx_fifo_state;
/* read fifo state */
tx_fifo_state = ds2211_sent_tx_fifo_state(DS2211_1_BASE,
                                          DS2211_SENT_CHANNEL1);
```

Configuring a SENT Receiver

Introduction

There are four independent SENT receivers on the DS2211 board. You can use the following RTLib functions to configure a SENT receiver.

Where to go from here

Information in this section

Information in other sections

Implementing SENT Receivers Using RTLib Functions (DS2211 Features 📖)
Provides information on the parameters for a SENT receiver and explains how you can implement it in a handcoded model.

ds2211_sent_rx_init

Syntax

```
void ds2211_sent_rx_init(
   Int32    base,
   UInt32   channel,
   UInt32   nibble_count)
```

Include file

`ds2211.h`

Purpose

To initialize a SENT receiver of the specified channel.

Description

The function initializes the specified SENT receiver and specifies the number of nibbles per SENT message, including the status nibble and the CRC nibble. The internal nibble buffer is allocated if not already done. The receiver FIFO is cleared, so unread messages are deleted. All configuration parameters are reset. This function must be executed before using the SENT receiver of specified channel and before configuring the SENT transmitter by calling the `ds2211_sent_rx_config_pause` function.

The SENT receivers use shared I/O pins with Bit I/O unit and PWMIN. It is not possible to use more than one functionality on one I/O pin.

SENT RX Channel	I/O Pin
Channel 1	DIG_IN1
Channel 2	DIG_IN2
Channel 3	DIG_IN3
Channel 4	DIG_IN4

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for receiving SENT messages (1 … 4). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3

nibble_count Sets the number of nibbles used in every SENT message including the status nibble and the CRC nibble within the range 1 … 217.

Return value

None

Example

```
/* Initialize SENT receiver channel 1 with 6 nibbles per message   */
ds2211_sent_rx_init(DS2211_1_BASE,            /* DS2211 base address    */
                    DS2211_SENT_CHANNEL1,     /* select channel 1       */
                    6);                       /* 6 nibbles per message  */
```

Message

Type	Message	Description
Error	ds2211_sent_rx_init (board_offset): Feature is not supported by DS2211 Board/PAL revision. Refer to documentation	This function is not supported by the board revision. Refer to Using the SENT Protocol on a DS2211 (DS2211 Features 📖).

Related topics

References

ds2211_sent_rx_config_pause..273

ds2211_sent_rx_config_pause

Syntax

```
void ds2211_sent_rx_config_pause(
   Int32    base,
   UInt32   channel,
   dsfloat  tic_period,
   dsfloat  clockdrift,
   UInt32   low_tics,
   UInt32   zero_nibble_high_tics,
   UInt32   sync_high_tics,
   UInt32   timing_range,
   UInt32   pause_mode,
   UInt32   fixed_mes_len_tics)
```

Include file

`ds2211.h`

Purpose

To configure the SENT receiver with a pause pulse (optional) of specified channel.

Description

The function configures the SENT receiver of the specified channel. It sets up the number of ticks used for a SENT pulse and a SENT message. The configuration must be set before reading the first message from the receiver buffer by calling the `ds2211_sent_rx_receive_all_pause` or `ds2211_sent_rx_receive_most_recent_pause` function. `Clockdrift` defines the maximum allowed clock drift of the connected SENT transmitter. This information is used to report synchronization pulses out of specification and to differ between a maximum allowed nibble pulse and a synchronization pulse.

To adapt the receiver clock to different pulse length and transmitter clocks, different timing ranges can be specified by the `timing_range` parameter. This parameter acts as a clock prescaler and reduces the resolution of the receiver clock while increasing the measurable pulse length. For information of the different timing ranges, refer to timing_range on page 275.

The recommended minimum tick period under consideration of the clock drift ensures at least 6 times oversampling to provide correct calculation of nibble values. The minimum pulse length must be at least 2 µs. The maximum measurable pulse length is the maximum high-pulse length and the maximum low-pulse length which can be measured by the receiver.

The `pause_mode` parameter is used to enable or disable the pause pulse mode. If the pause pulse mode is enabled, the receiver accepts a pause pulse at the end of every message, between the last nibble and the sync pulse of the next message. If the pause pulse mode is disabled, pause pulses cannot be handled by the receiver.

If the pause pulse mode is enabled, use the `ds2211_sent_rx_receive_all_pause` or `ds2211_sent_rx_receive_most_recent_pause` function, do not use the `ds2211_sent_rx_receive_all` or `ds2211_sent_rx_receive_most_recent` function.

When the pause pulse is used to get a fixed message length, you can specify the expected length in ticks by the `fixed_mes_len_tics` parameter. Additional diagnostic of fixed message length is then processed. The receiver indicates differences in the received message length via the diagnostic word. If the `fixed_mes_len_tics` parameter is set to 0, the fixed message length diagnostic is disabled and not reported via the diagnostic word. This is recommended, when a transmitter uses pause pulses but does not implement a fixed message length. If the pause pulse mode is disabled, the `fixed_mes_len_tics` parameter is ignored.

Note

The resulting pulse length must not exceed the range of allowed pulse length of the selected timing range. Otherwise, pulses gets lost or are measured with a wrong value.

If a pulse length exceeds the measurable pulse length, a timeout is reported by the read function. The timeout detection depends on the board/FPGA revision. For details of the required board/FPGA revisions, refer to Using the SENT Protocol on a DS2211 (DS2211 Features 🕮).

The maximum resulting nibble pulse under consideration of the specified clock drift must be shorter than the minimum resulting synchronization pulse under consideration of the clock drift.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for receiving SENT messages (1 … 4). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3

tic_period Specifies the tick period used by the connected SENT transmitter in seconds within the range 500 ns … 200 µs. This value is necessary to decode the received messages.

clockdrift Specifies the maximum allowed drift of the transmitter clock. This value is used to report synchronization pulses out of specification and to differ between synchronization and nibble pulses. The value is specified with the range 0 … 0.5 which is a percentage of the tick period, for example,

```
0:     0% clock drift
0.1:  10% clock drift
0.5:  50% clock drift
```

Remark: It is not recommended to use a clock drift of 0. Every transmitter and receiver has a minimum drift, so the measured pulse length can fluctuate around a mean value. The maximum possible nibble pulse is

```
MaxNibPulse = Tic_period · (zero_nibble_high_tics + 15 + low_tics) · (1 +
clockdrift)
```

If a nibble pulse is longer than this maximum, it is recognized as synchronization pulse. So with a clock drift of 0 some fluctuating nibble pulses with a value of 15 could be recognized as synchronization pulses.

low_tics Specifies the number of tick periods used for a SENT low pulse within the range 1 … 15.

zero_nibble_high_tics Specifies the number of tick periods used for a SENT nibble high pulse and a pause pulse with a value of 0 within the range 1 … 15.

sync_high_tics Specifies the number of tick periods used for a SENT synchronization high pulse within the range 17 … 255.

timing_range Specifies a timing range for pulse length measurement. You can use this to adapt the range of measurable pulse length to the transmitter clock and pulse length. The following table shows the predefined symbols and the relevant time information of the different timing ranges.

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

The recommended minimum tick period under consideration of the clock drift ensures at least 6 times oversampling to provide correct calculation of nibble values. The minimum pulse length must be at least 2 µs. The maximum measurable pulse length is the maximum high-pulse length and the maximum low-pulse length which can be measured by the receiver.

pause_mode Specifies the pause mode of the receiver. The following table shows the predefined symbols of the modes which you can specify.

Predefined Symbol	Value	Description
DS2211_SENT_DISABLE_PAUSE_PULSE	0	Disables the pause pulse mode (default).
DS2211_SENT_ENABLE_PAUSE_PULSE	1	Enables the pause pulse mode.

fixed_mes_len_tics Specifies an expected message length in ticks for additional diagnostic evaluation. Use it only if a pause pulse is used to generate a fixed message length. This value must include all low and high ticks of sync pulse, nibble pulses and pause pulse within the range 22 … 10065. If it is 0, no fixed message length diagnostic is processed. If the pause pulse mode is disabled, this value is ignored, you can set it to 0, for example.

Return value None

Example

```
/* Configure SENT receiver */
ds2211_sent_rx_config_pause(DS2211_1_BASE,           /* DS2211 base address   */
                      DS2211_SENT_CHANNEL1,          /* select channel 1      */
                      3e-6,                    /* tick period of transmitter  */
                      0.2,                       /* allowed clock drift 20%   */
                      5,                             /* low ticks             */
                      7,                             /* zero nibble high ticks*/
                      51);                           /* sync high ticks       */
                      DS2211_SENT_TIMING_RANGE1, /* use smallest timing range */
                      DS2211_SENT_ENABLE_PAUSE_PULSE,    /* enable pause pulse
                                                        reception             */
                      333);    /* enable receiver diagnostic for fixed message
                                  length with 333 ticks                       */
```

Message

Type	Message	Description
Error	ds2211_sent_rx_config_pause (board_offset): Ch *channel*: Max nibble pulse *max_nib_pulse_length* longer than min synchronization pulse *min_sync_pulse_length*	The configuration of the SENT receiver with specified channel is invalid. The resulting maximum nibble pulse within the clock drift is longer than the resulting minimum synchronization pulse within clock drift. So the receiver cannot differentiate between synchronization and nibble pulses. Check the configuration of the SENT receiver.

Related topics

Basics

Using the SENT Protocol on a DS2211 (DS2211 Features 📖)

References

ds2211_sent_rx_receive_all_pause 280
ds2211_sent_rx_receive_most_recent_pause 288

ds2211_sent_rx_config (obsolete)

Syntax

```
void ds2211_sent_rx_config(
   Int32     base,
   UInt32    channel,
   dsfloat   tic_period,
   dsfloat   clockdrift,
   UInt32    low_tics,
   UInt32    zero_nibble_high_tics,
   UInt32    sync_high_tics)
```

Include file

`ds2211.h`

Purpose

To configure the SENT receiver of specified channel.

> **Note**
>
> This function is obsolete, use the `ds2211_sent_rx_config_pause` function.

Description

The function configures the SENT receiver of the specified channel. It sets up the number of ticks used for a SENT pulse and a SENT message. The configuration must be set before reading the first message from the receiver buffer by calling `ds2211_sent_rx_receive_all (obsolete)` or `ds2211_sent_rx_receive_most_recent (obsolete)`. `Clockdrift` defines the maximum allowed clock drift of the connected SENT transmitter. This information is used to report synchronization pulses out of specification and to differ between a maximum allowed nibble pulse and a synchronization pulse.

> **Note**
>
> The resulting pulse length of low pulses and high pulses must not exceed the range of allowed pulse length of 2 µs … 819.2 µs.

Parameters

base PHS-bus base address of the DS2211 board.

channel Specifies the used SENT channel (1 … 4). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3

tic_period Specifies the tick period used by the connected SENT transmitter in seconds within the range 500 ns … 51.1875 µs. This value is necessary to decode the received messages.

clockdrift Specifies the maximum allowed drift of the transmitter clock. This value is used to report synchronization pulses out of specification and to differ between synchronization and nibble pulses. The value is specified with the range 0 … 1 which is a percentage of the tick period, for example,

0:	0% clock drift
0.1:	10% clock drift
0.5:	50% clock drift

Remark: It is not recommended to use a clock drift of 0. Every transmitter and receiver has a minimum drift, so the measured pulse length can fluctuate around a mean value. The maximum possible nibble pulse is `Tic_period · (zero_nibble_high_tics + 15 + low_tics) · (1 + clockdrift)`. If a nibble pulse is longer than the maximum, it is recognized as synchronization pulse. So with a clock drift of 0 some fluctuating nibble pulses with a value of 15 could be recognized as synchronization pulses.

low_tics Specifies the number of tick periods used for a SENT low pulse within the range 1 ... 15.

zero_nibble_high_tics Specifies the number of tick periods used for a SENT nibble high pulse with a value of 0 within the range 1 ... 15.

sync_high_tics Specifies the number of tick periods used for a SENT synchronization high pulse within the range 1 ... 255.

Return value

None

Example

```
/* Configure SENT receiver timings */
ds2211_sent_rx_config(DS2211_1_BASE,               /* DS2211 base address   */
                      DS2211_SENT_CHANNEL1,        /* select channel 1      */
                      3e-6,                  /* tick period of transmitter  */
                      0.2,                     /* allowed clock drift 20%   */
                      5,                           /* low ticks             */
                      7,                           /* zero nibble high ticks*/
                      51);                         /* sync high ticks       */
```

Message

Type	Message	Description
Error	ds2211_sent_rx_config(board_offset): Ch *channel*: Max nibble pulse *max_nib_pulse_length* longer than min synchronization pulse *min_sync_pulse_length*	The configuration of the SENT receiver with specified channel is invalid. The resulting maximum nibble pulse within the clock drift is longer than the resulting minimum synchronization pulse within clock drift. So the receiver cannot differentiate between synchronization and nibble pulses. Check the configuration of the SENT receiver.

Related topics

References

ds2211_sent_rx_config_pause 273
ds2211_sent_rx_receive_all (obsolete) 284
ds2211_sent_rx_receive_most_recent (obsolete) 292

ds2211_sent_rx_receive_all_pause

Syntax

```
UInt32 ds2211_sent_rx_receive_all_pause(
   Int32     base,
   UInt32    channel,
   Int8     *data,
   UInt32    count,
   UInt32   *len,
   UInt32   *diagnostic,
   Int16    *pause)
```

Include file

`ds2211.h`

Purpose

To read all new received messages and diagnostic with pause pulses (optional) from the receiver FIFO.

Description

The function reads all complete SENT messages received since the last read operation. The `diagnostic` parameter returns error information for every received message. The number of stored messages is returned by the `len` parameter. The messages are stored to `data` buffer. The buffer must hold at least the number of expected messages defined by the `count` parameter multiplied with the number of nibbles defined by `nibble_count` during initialization.

Read operations have to be executed continuously to avoid an overflow of the message buffer. The number of messages that can be buffered between two read operations can be calculate as follows:

If the pause pulse mode is disabled:

```
FIFO_MESSAGE_DEPTH = RoundDown(128 / (nibble_count + 1))
```

If the pause pulse mode is enabled:

```
FIFO_MESSAGE_DEPTH = RoundDown(128 / (nibble_count + 2))
```

The minimum necessary read frequency to avoid loss of data can be calculated with the knowledge of the timing parameters. The minimum possible message duration multiplied with `FIFO_MESSAGE_DEPTH` is the maximum time between two consecutive read operations. Refer to Implementing SENT Receivers Using RTLib Functions (DS2211 Features 🕮).

When the receive FIFO runs full, all new received pulses get lost until the next read operation is executed. So this will lead to a loss of nibbles or messages.

When recognizing an error in a SENT message (see diagnostic on page 282) the message is stored to the `data` buffer with the defined number of nibbles anyway. So the length of the received data array has always the length indicated by the `len` value in messages. The error is reported by the diagnostic port. The number of written diagnostic values corresponds to the number of received messages `len`. When too few nibbles are received in a message, the remaining nibbles are filled with `DS2211_SENT_MISSING_NIBBLE` (-128). When too many nibbles are received in a messages, the exceeding nibbles are cut off. So the length of a message is always the length defined by `nibble_count` during initialization. When receiving a nibble value out of the allowed range of 0 … 15, the nibble is stored to the message anyway and the diagnostic parameter reports a nibble out of the valid range. Nibble diagnostic information is evaluated for every nibble in a message, even for ignored excess nibbles when exceeding the configured number of nibbles per message (nibble_count).

To avoid writing more messages to the data buffer `data` than memory was allocated, the `count` parameter is used. This is the maximum number of messages that are written to the data buffer. When more messages are received, the remaining messages get lost to avoid an overrun of the message buffer. The return value reports the loss of data.

If you use read functions `ds2211_sent_rx_receive_most_recent_pause` and `ds2211_sent_rx_receive_all_pause` in the same real-time application for the same channel, you have to notice that the message buffer is cleared during every read operation. So if you call `ds2211_sent_rx_receive_all_pause` after calling `ds2211_sent_rx_receive_most_recent_pause` you will only read the messages received since the execution of `ds2211_sent_rx_receive_most_recent_pause`.

If the pause pulse mode is enabled (see `ds2211_sent_rx_config_pause` on page 273), the pause pulse values are returned by the `pause` parameter. One pause value per received message is stored to the pause buffer. The pause pulse buffer must at least have a length of `count`. If the pause pulse mode is disabled, nothing is written to the pause buffer. You can set the `pause` parameter to NULL.

The pause pulse value is processed like any other nibble value. The tic parameters defined by `ds2211_sent_rx_config_pause` are also relevant for the pause pulse. So the low pulse length is specified by `low_tics`, the high pulse length is defined by `zero_nibble_high_tics` and the value of `pause`. A resulting pause value is calculated as follows:

```
PAUSE_VALUE = PULSE_LENGTH_IN_TICS - (ZERO_NIBBLE_HIGH_TICS +
LOW_TICS)
```

If the pause pulse mode with fixed message length is enabled (see `ds2211_sent_rx_config_pause` on page 273), an additional pause pulse diagnostic is processed. The expected message length in tics is compared to the received one. If they differ, a `DS2211_SENT_FIXED_MSG_DIFF` is reported via the diagnostic word. If the ratio of a received sync pulse to the complete received message length differs by more than 1/64 from the nominal ratio, a `DS2211_SYNC_MSG_RATIO_DIFF` is reported via the diagnostic word. The complete message length consists of the low and high tics of the sync pulse, all nibbles and the pause pulse.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for receiving SENT messages (1 … 4). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3

data Specifies a message buffer which is filled with the received messages. The array must hold at least the number of expected messages defined by the `count` parameter.

Position in Data Buffer	Message / Nibble
data[0]	Message 1 / Nibble 1
data[1]	Message 1 / Nibble 2
…	…
data[*nibble_count* - 1]	Message 1 / Nibble *nibble_count*
data[*nibble_count*]	Message 2 / Nibble 1
…	…
data[2**nibble_count* - 1]	Message 2 / Nibble *nibble_count*
…	…
data[(*msg*-1)**nibble_count* + *nib*-1]	Message *msg* Nibble *nib*

When a nibble is out of its valid range, it is nevertheless stored to data buffer and the error is reported by the diagnostic information. Missing nibbles are returned with a value of `DS2211_SENT_MISSING_NIBBLE` (-128).

count Specifies the maximum number of messages that are written to the data buffer. This value is useful, to prevent writing too many messages to the allocated data buffer in the real-time application.

len Returns the number of messages written to the `*data` buffer and, if pause pulse mode is enabled, the number of pause values written to the `*pause` buffer.

diagnostic Returns diagnostic information for each received SENT message. The diagnostic vector must hold at least the number of expected messages defined by the `count` parameter. The format of the diagnostic vector is as follows:

Position in Diagnostic Buffer	Message
diagnostic[0]	Message 1
diagnostic[1]	Message 2
…	…
diagnostic[n-1]	Message n

The SENT receiver generates a diagnostic word for every received message. The diagnostic word consists of flags, indicating different message specific status and diagnostic information. The meaning of the flags is as follows:

Bit	Description
0	Too many nibbles in message.
1	Too few nibbles in message.
2	Nibble value is out of range 0 … 15.
3	Synchronization pulse too long (out of specified allowed clock drift).
4	Synchronization pulse too short (out of specified allowed clock drift).
5	Actual synchronization pulse differs more than factor 1/64 from the last synchronization pulse.
6	The received fixed message length in tics differs from the expected one.
7	The received ratio of sync pulse to message length differs by more than 1/64 from the nominal ratio.

pause Specifies the pause pulse buffer where the received pause pulses are written to. The array must hold minimum the number of expected messages defined by `count`. The format of the vector is as follows:

Message	Position in Pause Buffer
Pause of message 1	pause[0]
Pause of message 2	pause[1]
…	…
Pause of message m	pause[m-1]

Return value

UInt32 The return value provides global error information, like a buffer overflow or a receiver timeout. The timeout detection depends on the board/FPGA revision (see Using the SENT Protocol on a DS2211 (DS2211 Features 📖)).

Value	Predefined Symbol	Description
0	-	No error occurred during this read operation.
0x1	`DS2211_SENT_DATA_LOSS`	The number of received messages exceeded the expected number of messages set by the `count` parameter. At least one message was discarded to prevent an overrun of the allocated message buffer.
0x2	`DS2211_SENT_TIMEOUT`	A receiver timeout occurred. This means that at least one pulse length exceeded the maximum measurable pulse length of the specified timing range set by `ds2211_sent_rx_config_pause`. In this case, the receiver discards the message with a

Value	Predefined Symbol	Description
		timeout and searches for the next sync pulse. The timeout information is active for one read operation only.

Example

```
Int8 rx_msg[22*6];                    /* buffer for 22 messages with 6 nibbles */
Int16 rx_pause[22];                               /* buffer for 22 pause values */
UInt32 rx_len;                    /* store number of messages that were received */
UInt32 rx_diag[22];                          /* buffer for 22 diagnostic values */
UInt32 rx_error;                          /* indicates global error information */
/* receive messages */
rx_error = ds2211_sent_rx_receive_all_pause(DS2211_1_BASE,    /* base address */
                              DS2211_SENT_CHANNEL1,              /* Channel 1 */
                              rx_msg,             /* pointer to message buffer */
                              22,                 /* maximum number of messages */
                              &rx_len,            /* number of received messages */
                              rx_diag,          /* pointer to diagnostic buffer */
                              rx_pause);            /* pointer to pause buffer */
```

Messages

Type	Message	Description
Error	ds2211_sent_rx_receive_all_pause (*board_offset*): Ch *channel*: Function was interrupted by itself.	The receive function was interrupted by another receive function for the same SENT channel. This can occur if you call SENT receiver functions for the same channel in different tasks or interrupt service routines.

Related topics

References

- ds2211_sent_rx_config_pause.............273
- ds2211_sent_rx_receive_most_recent_pause.............288

ds2211_sent_rx_receive_all (obsolete)

Syntax

```
UInt32 ds2211_sent_rx_receive_all(
   Int32     base,
   UInt32    channel,
   Int8     *data,
   UInt32   *count,
   UInt32   *len,
   UInt32   *diagnostic)
```

Include file

`ds2211.h`

Purpose

To read all new received messages and diagnostic from the receiver FIFO.

> **Note**
>
> This function is obsolete, use the `ds2211_sent_rx_receive_all_pause` function.

Description

The function reads all complete SENT messages received since the last read operation. The `diagnostic` parameter returns error information for every received message. The number of stored messages is returned by the `len` parameter. The messages are stored to `data` buffer. The buffer must hold at least the number of expected messages defined by the `count` parameter multiplied with the number of nibbles defined by `nibble_count` during initialization.

Read operations have to be executed continuously to avoid an overflow of the message buffer. The number of messages that can be buffered between two read operations can be calculate as follows: `FIFO_MESSAGE_DEPTH = RoundDown(128 / (nibble_count + 1))` The minimum necessary read frequency to avoid loss of data can be calculated with the knowledge of the timing parameters. The minimum possible message duration multiplied with `FIFO_MESSAGE_DEPTH` is the maximum time between two consecutive read operations. See also Implementing SENT Receivers Using RTLib Functions (DS2211 Features 🕮).

When the receive FIFO runs full, all new received pulses get lost until the next read operation is executed. So this will lead to a loss of nibbles or messages.

When recognizing an error in a SENT message (see diagnostic on page 286) the message is stored to the `data` buffer with the defined number of nibbles anyway. So the length of the received data array has always the length indicated by the `len` value in messages. The error is reported by the diagnostic port. The number of written diagnostic values corresponds to the number of received messages `len`. When too few nibbles are received in a message, the remaining nibbles are filled with `DS2211_SENT_MISSING_NIBBLE` (-128). When too many nibbles are received in a messages, the exceeding nibbles are cut off. So the length of a message is always the length defined by `nibble_count` during initialization. When receiving a nibble value out of the allowed range of 0 .. 15, the nibble is stored to the message anyway and the diagnostic parameter reports a nibble out of the valid range.

To avoid writing more messages to the data buffer `data` than memory was allocated, the `count` parameter is used. This is the maximum number of messages that are written to the data buffer. When more messages are received, the remaining messages get lost to avoid an overrun of the message buffer. The return value reports the loss of data.

If you use read functions `ds2211_sent_rx_receive_most_recent` and `ds2211_sent_rx_receive_all` in the same real-time application for the same channel, you have to notice that the message buffer is cleared during every read operation. So if you call `ds2211_sent_rx_receive_all` after calling ds2211_sent_rx_receive_most_recent you will only read the messages received since the execution of ds2211_sent_rx_receive_most_recent.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for receiving SENT messages (1 … 4). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	DS2211_SENT_CHANNEL1	0
Channel 2	DS2211_SENT_CHANNEL2	1
Channel 3	DS2211_SENT_CHANNEL3	2
Channel 4	DS2211_SENT_CHANNEL4	3

data This message buffer is filled with the received messages. The array must hold at least the number of expected messages defined by the count parameter.

Position in Data Buffer	Message / Nibble
data[0]	Message 1 / Nibble 1
data[1]	Message 1 / Nibble 2
…	…
data[*nibble_count* - 1]	Message 1 / Nibble *nibble_count*
data[*nibble_count*]	Message 2 / Nibble 1
…	…
data[2 · *nibble_count* - 1]	Message 2 / Nibble *nibble_count*
…	…
data[(*msg*-1) · *nibble_count* + *nib*-1]	Message *msg* Nibble *nib*

When a nibble is out of its valid range, it is nevertheless stored to data buffer and the error is reported by the diagnostic information. Missing nibbles are returned with a value of DS2211_SENT_MISSING_NIBBLE (-128).

count To set the maximum number of messages that are written to the data buffer. This value is useful, to prevent writing too many messages to the allocated data buffer in the real-time application.

If more messages are received than specified by the count parameter, the exceeding newest messages are discarded, which is indicated by a return value of DS2211_SENT_DATA_LOSS (1). A return value of DS2211_SENT_NO_DATA_LOSS (0) signals that all messages could be written to the data buffer.

len Returns the number of messages written to the *data buffer.

diagnostic Returns diagnostic information for each received SENT message. The diagnostic vector must hold at least the number of expected messages defined by the count parameter. The format of the diagnostic vector is as follows:

Position in Diagnostic Buffer	Message
diagnostic[0]	Message 1
diagnostic[1]	Message 2
...	...
diagnostic[n-1]	Message n

The SENT receiver generates a diagnostic word for every received message. The diagnostic word consists of flags, indicating different message specific status and diagnostic information. The meaning of the flags is as follows:

Bit	Description
0	Too many nibbles in message.
1	Too few nibbles in message.
2	Nibble value is out of range 0 ... 15.
3	Synchronization pulse too long (out of specified allowed clock drift).
4	Synchronization pulse too short (out of specified allowed clock drift).
5	Actual synchronization pulse differs more than factor 1/64 from the last synchronization pulse.

Return value

UInt32 The return value indicates if all messages could be written to the data buffer, or if messages were discarded, because more messages were received than expected messages were specified by the `count` parameter.

DS2211_SENT_NO_DATA_LOSS (0): The number of received messages did not exceed the expected number of received messages set by `count` parameter.

DS2211_SENT_DATA_LOSS (1): The number of received messages exceeded the expected number of messages set by the `count` parameter. At least one message was ignored to prevent an overrun of the allocated message buffer. The newest messages are discarded.

Example

```
Int8 rx_msg[22][6];             /* buffer for 22 messages with 6 nibbles */
UInt32 rx_len;               /* store number of messages that were received */
UInt32 rx_diag[22];          /* buffer for diagnostic values */
UInt32 rx_error;             /* indicates that not all messages could be read */
/* receive messages */
rx_error = ds2211_sent_rx_receive_all(DS2211_1_BASE,        /* base address  */
                         DS2211_SENT_CHANNEL1,             /* Channel 1      */
                         &rx_msg[0][0],         /* pointer to message buffer */
                         22,                   /* maximum number of messages */
                         &rx_len,             /* number of received messages */
                         rx_diag);           /* pointer to diagnostic buffer */
```

Messages

Type	Message	Description
Error	ds2211_sent_rx_receive_all (*board_offset*): Ch *channel*: Function was interrupted by itself.	The receive function was interrupted by another receive function for the same SENT channel. This could occur if you call SENT receiver functions for the same channel in different tasks or interrupt service routines.

Related topics

References

ds2211_sent_rx_receive_all_pause 280
ds2211_sent_rx_receive_most_recent (obsolete) 292

ds2211_sent_rx_receive_most_recent_pause

Syntax

```
UInt32 ds2211_sent_rx_receive_most_recent_pause(
   Int32     base,
   UInt32    channel,
   Int8     *data,
   UInt32   *msg_count,
   UInt32   *diagnostic,
   Int16    *pause)
```

Include file

`ds2211.h`

Purpose

To read the most recent message and diagnostic with pause pulse (optional) from the receiver FIFO.

Description

The function reads the newest complete received message from the receiver FIFO. The `diagnostic` parameter returns error information for the read message. The message is stored to `data` buffer. The buffer must hold minimum the number of nibbles defined by `nibble_count` during initialization. The number of messages received since the last execution of a receiver function is returned by the `msg_count` parameter. If no complete message was received at all, a message with all nibbles marked as missing nibbles `DS2211_SENT_MISSING_NIBBLE` (-128) and, if enabled, a pause pulse of `DS2211_SENT_MISSING_PAUSE` (-32768) are returned and the `msg_count` parameter is 0.

Read operations have to be executed continuously to avoid an overflow of the internal message buffer. The number of messages that can be buffered between two read operations is calculated as follows:

If the pause pulse mode is disabled:

```
FIFO_MESSAGE_DEPTH = RoundDown(128 / (nibble_count + 1))
```

If the pause pulse mode is enabled:

```
FIFO_MESSAGE_DEPTH = RoundDown(128 / (nibble_count + 2))
```

The minimum necessary read frequency to avoid data loss can be calculated with the knowledge of the timing parameters. The minimum possible message duration multiplied with `FIFO_MESSAGE_DEPTH` is the maximum time between two consecutive read operations. Refer to Implementing SENT Receivers Using RTLib Functions (DS2211 Features 🕮).

If the receive FIFO runs full, all new received pulses get lost until the next read operation is executed. This leads to a loss of nibbles or messages.

If an error in a SENT message is recognized (see diagnostic on page 290), the message is stored to the `data` buffer with the defined number of nibbles anyway. The length of the received data array has always the length of `nibble_count`. The error is reported by the diagnostic port.

If too few nibbles are received in a message, the remaining nibbles are filled with `DS2211_SENT_MISSING_NIBBLE` (-128). If too many nibbles are received in a messages, the exceeding nibbles are cut off. The length of a message is always the length defined by `nibble_count` during initialization. When receiving a nibble value out of the allowed range 0 .. 15, the nibble is stored to the message buffer anyway and the `diagnostic` parameter reports a nibble out of the valid range. Nibble diagnostic information is evaluated for every nibble in a message, even for ignored excess nibbles when exceeding the configured number of nibbles per message (`nibble_count`).

If you use read functions `ds2211_sent_rx_receive_most_recent_pause` and `ds2211_sent_rx_receive_all_pause` in the same real-time application for the same channel, you have to notice that the message buffer is cleared during every read operation. So if you call `ds2211_sent_rx_receive_all_pause` after calling `ds2211_sent_rx_receive_most_recent_pause` you will only read the messages received since the execution of `ds2211_sent_rx_receive_most_recent_pause`.

If the pause mode is enabled (see `ds2211_sent_rx_config_pause` on page 273), the pause pulse value is returned by the `pause` parameter. If the pause pulse mode is disabled, nothing is written to the pause buffer. The pause pulse value is processed like any other nibble value. The tic parameters specified by `ds2211_sent_rx_config_pause` are also relevant for the pause pulse. So the low pulse length is specified by `low_tics`, the high pulse length is defined by `zero_nibble_high_tics` and the value of pause. A resulting pause value is calculated as follows:

```
PAUSE_VALUE = PULSE_LENGTH_IN_TICS - (ZERO_NIBBLE_HIGH_TICS +
LOW_TICS)
```

If the pause pulse mode with fixed message length is enabled (see `ds2211_sent_rx_config_pause` on page 273), an additional pause pulse diagnostic is processed. The expected message length in ticks is compared to the received one. If they differ, a `DS2211_SENT_FIXED_MSG_DIFF` is reported via the diagnostic word. If the ratio of a received sync pulse to the complete received message length differs by more than 1/64 from the nominal ratio, a DS2211_SYNC_MSG_RATIO_DIFF is reported via the diagnostic word. The complete message length consists of the sync pulse, all nibbles and the pause pulse.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for receiving SENT messages (1 … 4). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	DS2211_SENT_CHANNEL1	0
Channel 2	DS2211_SENT_CHANNEL2	1
Channel 3	DS2211_SENT_CHANNEL3	2
Channel 4	DS2211_SENT_CHANNEL4	3

data Specifies a message buffer which is filled with the received message. It must hold at least the number of nibbles per message (nibble_count).

Nibble	Position in Data Buffer
Nibble 1	rx_data[0]
Nibble 2	rx_data[1]
...	...
Nibble *nib*	rx_data[nib - 1]

When a nibble is out of its valid range, it is nevertheless stored to data buffer and the error is reported by the diagnostic information.

Missing nibbles are returned with a value of DS2211_SENT_MISSING_NIBBLE (-128).

msg_count Returns the number of received messages since the last read operation. This information can be used to recognize a too long cycle time of the model, when the number of received messages is nearby the maximum FIFO message depth.

diagnostic A SENT receiver delivers diagnostic information for the received SENT message.

The SENT receiver generates a diagnostic word for the received message. The diagnostic word consists of flags, indicating different message specific status and diagnostic information. The meaning of the flags is as follows:

Bit	Description
0	Too many nibbles in message.
1	Too few nibbles in message.
2	Nibble value is out of range 0 ... 15.
3	Synchronization pulse too long (out of specified allowed clock drift).
4	Synchronization pulse too short (out of specified allowed clock drift).

Bit	Description
5	Actual synchronization pulse differs more than factor 1/64 from the last synchronization pulse.
6	The received fixed message length in ticks differs from the expected one.
7	The received ratio of sync pulse to message length differs by more than 1/64 from the nominal ratio.

pause Specifies the pause pulse buffer where the received pause pulse is written to.

Return value

UInt32 The return value provides global error information, like a receiver timeout. The timeout detection depends on the board/FPGA revision (see Using the SENT Protocol on a DS2211 (DS2211 Features 🕮)).

Value	Predefined Symbol	Description
0	-	No error occurred during this read operation.
0x2	`DS2211_SENT_TIMEOUT`	A receiver timeout occurred. This means that at leasts one pulse length exceeded the maximum measurable pulse length of the specified timing range set by **`ds2211_sent_rx_config_pause`**. In this case, the receiver discards the message with a timeout and searches for the next sync pulse. The timout information is active for one read operation only.

Example

```
Int8 rx_msg[6];                       /* buffer for 1 messages with 6 nibbles */
Int16 rx_pause;                                   /* buffer for 1 pause value */
UInt32 rx_count;               /* store number of messages that were received */
UInt32 rx_diag;                                /* buffer for diagnostic value */
/* receive messages */
ds2211_sent_rx_receive_most_recent_pause(DS2211_1_BASE,       /* base address */
                                   DS2211_SENT_CHANNEL1,         /* Channel 1 */
                                   rx_msg,       /* pointer to message buffer */
                                   &rx_count,  /* number of received messages */
                                   &rx_diag,  /* pointer to diagnostic buffer */
                                   &rx_pause);     /* pointer to pause buffer */
```

Messages

Type	Message	Description
Error	ds2211_sent_rx_receive_most_recent_pause (*board_offset*): Ch *channel*: Function was interrupted by itself.	The receive function was interrupted by another receive function for the same SENT channel. This could occur if you call SENT receiver functions for the same channel in different tasks or interrupt service routines.

Related topics

References

ds2211_sent_rx_config_pause... 273
ds2211_sent_rx_receive_all_pause... 280

ds2211_sent_rx_receive_most_recent (obsolete)

Syntax

```
void ds2211_sent_rx_receive_most_recent(
   Int32      base,
   UInt32     channel,
   Int8 *     data,
   UInt32 *   msg_count,
   UInt32 *   diagnostic)
```

Include file

`ds2211.h`

Purpose

To read the most recent message and diagnostic from the receiver FIFO.

> **Note**
>
> This function is obsolete, use the `ds2211_sent_rx_receive_most_recent_pause` function.

Description

The function reads the newest complete received message from the receiver FIFO. The `diagnostic` parameter returns error information for the read message. The message is stored to `data` buffer. The buffer must hold minimum the number of nibbles defined by `nibble_count` during initialization. The number of messages received since the last execution of a receiver function is returned by the `msg_count` parameter. If no complete message was received at all, a message with all nibbles marked as missing nibbles `DS2211_SENT_MISSING_NIBBLE` (-128) is returned and the `msg_count` parameter is 0.

Read operations have to be executed continuously to avoid an overflow of the internal message buffer. The number of messages that can be buffered between two read operations is calculated as follows:

```
FIFO_MESSAGE_DEPTH = RoundDown(128 / (nibble_count + 1))
```

The minimum necessary read frequency to avoid data loss can be calculated with the knowledge of the timing parameters. The minimum possible message duration multiplied with `FIFO_MESSAGE_DEPTH` is the maximum time between two consecutive read operations, see Implementing SENT Receivers Using RTLib Functions (DS2211 Features 📖).

If the receive FIFO runs full, all new received pulses get lost until the next read operation is executed. This leads to a loss of nibbles or messages.

If an error in a SENT message is recognized (see diagnostic on page 293), the message is stored to the `data` buffer with the defined number of nibbles anyway. The length of the received data array has always the length of `nibble_count`. The error is reported by the diagnostic port.

If too few nibbles are received in a message, the remaining nibbles are filled with `DS2211_SENT_MISSING_NIBBLE` (-128). If too many nibbles are received in a messages, the exceeding nibbles are cut off. The length of a message is always the length defined by `nibble_count` during initialization. When receiving a nibble value out of the allowed range 0 .. 15, the nibble is stored to the message buffer anyway and the `diagnostic` parameter reports a nibble out of the valid range.

If you use read functions `ds2211_sent_rx_receive_most_recent` and `ds2211_sent_rx_receive_all` in the same real-time application for the same channel, you have to notice that the message buffer is cleared during every read operation. So if you call `ds2211_sent_rx_receive_all` after calling `ds2211_sent_rx_receive_most_recent` you will only read the messages received since the execution of `ds2211_sent_rx_receive_most_recent`.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for receiving SENT messages (1 … 4). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3

data Specifies a message buffer which is filled with the received message. It must hold at least the number of nibbles per message (`nibble_count`).

Nibble	Position in Data Buffer
Nibble 1	rx_data[0]
Nibble 2	rx_data[1]
…	…
Nibble *nib*	rx_data[nib - 1]

When a nibble is out of its valid range, it is nevertheless stored to data buffer and the error is reported by the diagnostic information.

Missing nibbles are returned with a value of `DS2211_SENT_MISSING_NIBBLE` (-128).

msg_count Returns the number of received messages since the last read operation. This information can be used to recognize a too long cycle time of the model, when the number of received messages is nearby the maximum FIFO message depth.

diagnostic A SENT receiver delivers diagnostic information for the received SENT message.

The SENT receiver generates a diagnostic word for the received message. The diagnostic word consists of flags, indicating different message specific status and diagnostic information. The meaning of the flags is as follows:

Bit	Description
0	Too many nibbles in message.
1	Too few nibbles in message.
2	Nibble value is out of range 0 … 15.
3	Synchronization pulse too long (out of specified allowed clock drift).
4	Synchronization pulse too short (out of specified allowed clock drift).
5	Actual synchronization pulse differs more than factor 1/64 from the last synchronization pulse.

Return value

None

Example

```
Int8 rx_msg[6];              /* buffer for 1 messages with 6 nibbles */
UInt32 rx_count;             /* store number of messages that were received */
UInt32 rx_diag;              /* buffer for diagnostic value */
/* receive messages */
ds2211_sent_rx_receive_most_recent(DS2211_1_BASE,        /* base address  */
                       DS2211_SENT_CHANNEL1,            /* Channel 1      */
                       rx_msg,              /* pointer to message buffer */
                       &rx_count,          /* number of received messages */
                       &rx_diag);         /* pointer to diagnostic buffer */
```

Messages

Type	Message	Description
Error	ds2211_sent_rx_receive_most_recent (*board_offset*): Ch *channel*: Function was interrupted by itself.	The receive function was interrupted by another receive function for the same SENT channel. This can occur if you call SENT receiver functions for the same channel in different tasks or interrupt service routines.

Related topics

References

ds2211_sent_rx_receive_all (obsolete).. 284
ds2211_sent_rx_receive_most_recent_pause.. 288

ds2211_sent_get_rx_tic_period

Syntax

```
dsfloat ds2211_sent_get_rx_tic_period(
   Int32    base,
   UInt32   channel)
```

Include file

`ds2211.h`

Purpose

To return the actual tick period of the specified channel.

Description

The function returns the actual tick period of the specified SENT receiver channel in seconds. The tick period is extracted from the last received synchronization pulse, when reading SENT messages with `ds2211_sent_rx_receive_most_recent_pause` or `ds2211_sent_rx_receive_all_pause`. The value is updated with every read operation so it is constant between two executions of receive function.

When no message read operation was executed before reading the tick period, a tick period of 0 is returned.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Specifies the channel used for receiving SENT messages (1 … 4). The following symbols are predefined.

Channel	Predefined Symbol	Value
Channel 1	`DS2211_SENT_CHANNEL1`	0
Channel 2	`DS2211_SENT_CHANNEL2`	1
Channel 3	`DS2211_SENT_CHANNEL3`	2
Channel 4	`DS2211_SENT_CHANNEL4`	3

Return value

dsfloat Tick period of last received synchronization pulse in seconds.

Example

```
dsfloat rx_tic_period;          /* stores the actual receiver tick period */
/* get the current receiver tick period */
rx_tic_period = ds2211_sent_get_rx_tic_period(DS2211_1_BASE,
                                               DS2211_SENT_CHANNEL1);
```

Related topics

References

ds2211_sent_rx_receive_all_pause..280
ds2211_sent_rx_receive_most_recent_pause...288

Slave DSP Access Functions

Where to go from here

Information in this section

Information in other sections

Basics

Where to go from here

Information in this section

Basics of Accessing Slave DSP Features

Basics

You need the following master functions to access the slave DSP features:

- Function for communication between the master processor and the slave DSP.
- Global slave DSP functions
- Function for generating knock sensor signals
- Function for generating wheel speed sensor signals
- Function for accessing the DSP via the dual-port memory.

For more information on the TMS320VC33 slave DSP, refer to the Texas Instruments web site at "http://www.ti.com" and search for*TMS320C3x User's Guide* (literature number SPRU031F) and *TMS320VC33 Digital Signal Processor* (literature number SPRS087E).

Basic Communication Principles

Introduction

The communication between the master processor and the slave DSP is performed via the DPMEM of the DS2211. If access to this DPMEM is not arbitrated by hardware, you can avoid conflicts when the DPMEM is accessed from both sides, by the TMS320VC33 DSP and the master processor, by using one of the sixteen semaphores (1 … 16) provided by the DS2211.

Note

The semaphores do not physically prevent improper access to the DPMEM.

Hardware arbitration

DS2211 boards with board revision 3 and FPGA revision 3 or higher have a 32-bit hardware arbitration that avoids conflicts when the DPM is accessed. It is not necessary to use semaphores. However, using a semaphore is helpfull if you transfer several values that must be of integrity.

Tip

The revision number is displayed on the Properties pane in ControlDesk when you select the board in the Platforms/Devices controlbar.

Semaphore handling

The master requests a semaphore by writing a 0 to it. If you read the semaphore afterwards and get 0, the semaphore has been accessed successfully. Continue polling the request function until the semaphore is obtained successfully. If the value is not equal to 0, the semaphore was obtained by the other side. If you obtain the semaphore, you have to write a 1 to it afterwards to release it. If the request was not successful and you do not want to poll, you also have to release the semaphore by writing a 1 to it.

The RTLib functions described in the following topics handle the semaphores automatically where necessary, except for `ds2211_slave_dsp_read_direct`, `ds2211_slave_dsp_write_direct`, `ds2211_slave_dsp_block_read_di`, and `ds2211_slave_dsp_block_write_di`. For these functions you have to use `ds2211_slave_dsp_sem_req` to request and `ds2211_slave_dsp_sem_rel` to release a semaphore.

Floating-point conversion

For the processor board a different floating-point format is used. A processor board uses the IEEE floating-point format whereas the slave DSP uses the TI floating-point format. Therefore, floating-point values have to be converted with the `RTLIB_CONV_FLOAT32_TO_IEEE32` or `RTLIB_CONV_FLOAT32_FROM_IEEE32` conversion macros.

Related topics

References

ds2211_slave_dsp_block_read_di.. 334
ds2211_slave_dsp_block_write_di.. 335
ds2211_slave_dsp_read_direct.. 332
ds2211_slave_dsp_sem_rel.. 331
ds2211_slave_dsp_sem_req.. 330
ds2211_slave_dsp_write_direct.. 333
RTLIB_CONV_FLOAT32_FROM_IEEE32 (DS1006 RTLib Reference 🕮)
RTLIB_CONV_FLOAT32_FROM_IEEE32 (DS1007 RTLib Reference 🕮)
RTLIB_CONV_FLOAT32_TO_IEEE32 (DS1006 RTLib Reference 🕮)
RTLIB_CONV_FLOAT32_TO_IEEE32 (DS1007 RTLib Reference 🕮)

Overall DSP Functions

Introduction

The DS2211 has a slave DSP. The slave DSP is used to generate signals, for example, for simulating wheel speed sensor or knock sensor.

For general information on the slave DSP, refer to Features of the Slave DSP (DS2211 Features 🕮).

Where to go from here

Information in this section

ds2211_slave_dsp_signal_enable

Syntax

```
Int32 ds2211_slave_dsp_signal_enable(
      Int32 base,
      UInt32 enable_mask)
```

Include file

`ds2211.h`

Purpose

To start the slave DSP signal generation on the specified channels.

Description

`ds2211_slave_dsp_signal_enable` affects all signals. You specify the signals to be enabled, all the other signals will be disabled. Use `ds2211_slave_dsp_channel_enable` to enable or disable only specific signals.

Note

This function is used to start the slave DSP signal generation for standard slave applications (knock sensor and wheel speed). The value of the `enable_mask` variable is written to the first dual-port memory address (offset = 0) for evaluation by the applications. If you want to use this function for your own applications, you have to evaluate this value manually.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

enable_mask Signals to be enabled or disabled. The following symbols are predefined. To enable more than one signal, the symbols may be combined by the logical operator OR. If you do not specify a symbol, the signal will be disabled.

Predefined Symbol	Meaning
DS2211_SLVDSP_CH1	Starts the generation of signal 1.
…	…
DS2211_SLVDSP_CH8	Starts the generation of signal 8.

For knock sensor simulation, the symbols DS2211_SLVDSP_CH1 … DS2211_SLVDSP_CH8 apply to 8 cylinders. For wheel speed sensor simulation, the symbols DS2211_SLVDSP_CH1 … DS2211_SLVDSP_CH4 apply to 4 wheel speed sensors. The predefined symbols can be combined with a logical OR.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_SLVDSP_DPMEM_ACCESS_PASSED	The function has been performed without error.
DS2211_SLVDSP_DPMEM_ACCESS_FAILED	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_slave_dsp_channel_enable.. 302

ds2211_slave_dsp_channel_enable

Syntax

```
Int32 ds2211_slave_dsp_channel_enable(
      Int32 base,
      Int32 channel,
      UInt8 enable)
```

Include file

`ds2211.h`

Purpose

To start the slave DSP signal generation on the specified channel.

Description

The signal generation on the specified slave DSP channel is started or stopped depending on the value of `enable`.

Note

This function is used to start the slave DSP signal generation for standard slave applications (knock sensor and wheel speed simulation). This function sets the bit, which corresponds to the channel specified by the `channel` parameter, in the first dual-port memory address (offset = 0). This value is evaluated by the corresponding applications.
If you want to use this function for own applications, you have to evaluate this value manually.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number of the signal to be enabled in the range 1 … 8.

enable Enable value.

Value	Meaning
0	Disables the specified signal.
1	Enables the specified signal.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_SLVDSP_DPMEM_ACCESS_PASSED	The function was performed without error.
DS2211_SLVDSP_DPMEM_ACCESS_FAILED	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

ds2211_slave_dsp_interrupt_set

Syntax

```
Int32 ds2211_slave_dsp_interrupt_set(
      Int32 base,
      void *value)
```

Include file

```
ds2211.h
```

Purpose

To generate a slave DSP interrupt.

Description

The specified data value is written to a defined location of the DPMEM to generate an interrupt on the slave DSP. The interrupt triggers external interrupt 1 (INT1) of the slave DSP (refer to Basic Communication Principles on page 298 and Slave DSP Basics on page 340).

You can write different values to this interrupt address to indicate different subinterrupts and check the value in the corresponding interrupt routine.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

value Data value to be written (the data type can be float or long). If the processor board uses the IEEE floating-point format, the floating-point values have to be converted. Refer to Basic Communication Principles on page 298.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

Basic Communication Principles..298

ds2211_slave_dsp_speedchk

Syntax

```
void ds2211_slave_dsp_speedchk(
      Int32 base,
      dsfloat *exec_min,
      dsfloat *exec_max,
      dsfloat *exec_cur)
```

Include file

`ds2211.h`

Purpose

To get the execution times (minimum, maximum and current) of slave DSP interrupt service routines.

Note

This function must be used in the background and is executed every 1000th call of the background loop to avoid too much traffic on the DPMEM. The slave DSP must execute the **speedchk** macro in the background (see speedchk on page 430). Semaphore 16 is used to access the DPMEM. This function uses the DPMEM address range from 03xFFB to 0x3FFD. For information on semaphore handling, refer to Basic Communication Principles on page 298.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

exec_min Address where the minimum execution time of the slave DSP application (in µs) will be written

exec_max Address where the maximum execution time of the slave DSP application (in µs) will be written

exec_cur Address where the current execution time of the slave DSP application (in µs) will be written

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

speedchk...430

ds2211_slave_dsp_error

Syntax

```
Int32 ds2211_slave_dsp_error(
      Int32 base,
      Int32 *state)
```

Include file

`ds2211.h`

Purpose

To read the error flag of the slave DSP. Semaphore 16 is used to access the DPMEM. In your slave DSP application, you can set the error flag with `error_set` on page 363. This function uses the DPMEM address 03xFFE.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

state Status of the slave DSP error flag

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_SLVDSP_DPMEM_ACCESS_PASSED	The function was performed without error.
DS2211_SLVDSP_DPMEM_ACCESS_FAILED	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

error_set..363

ds2211_slave_dsp_appl_load

Syntax

```
void ds2211_slave_dsp_appl_load(
      Int32 base,
      Int32 *appl_addr)
```

Include file

`ds2211.h`

Purpose

To load a slave DSP application to the DPMEM and start the slave DSP.

Description

After starting the slave DSP, the function waits until the slave has finished its boot sequence. If the whole DPMEM is used for the application, booting the slave DSP takes approximately 25 ms.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

appl_addr Address of the first element of the slave DSP application

Return value

None

Message

A message is output only if the DEBUG_INIT status flag was set when the application was compiled. The following messages are defined:

Type	Message	Meaning
Error	ds2211_slave_dsp_appl_load(0x??): slave-DSP is not responding!	The application could not be started on the slave DSP.
Error	ds2211_slave_dsp_appl_load(0x??): slave-load already been acknowledged!	The function additionally checks whether the hostmem section still contains slave application data. If not and the corresponding flag is already reset, the function exits with an error message.
Info	ds2211_slave_dsp_appl_load(0x??): application loaded successfully!	The application was loaded to the slave DSP successfully.

Execution times

For information, refer to Function Execution Times on page 551.

Example

The following example is taken from the knock sensor simulation demo application `Master.c` (refer to Knock Sensor Simulation Demo on page 443).

```
...
```

```
#include "Slv2211_ks_appl.slc"
/************************************************************
object declarations
************************************************************/
/* pointer to slave DSP application data */
extern unsigned long ks_appl[];
...
/*-----------------------------------------------------*/
void main()
{
   ...
   init();                                /* init CPU board */
   ds2211_init(DS2211_1_BASE);         /* init DS2211 board */
   /* load DS2211 slave-DSP application 'ks_appl' */
   ds2211_slave_dsp_appl_load(DS2211_1_BASE,
                            (Int32 *) &ks_appl);
...
}
```

Related topics

HowTos

How to Load a Slave Application with Permanently Available Slave Data................................. 438
How to Load a Slave Application with Temporarily Available Slave Data................................. 439

References

ds2211_init.. 20

Knock Sensor Simulation

Introduction

You can simulate a knock sensor using a ready-to-use application implemented on the slave DSP.

For general information on knock sensor simulation and its I/O mapping, refer to Knock Sensor Simulation (DS2211 Features 🕮).

Where to go from here

Information in this section

Example of Knock Sensor Simulation

Example

The following example (`Slv_knock_2211_hc.c`) shows how to use the knock sensor simulation.

```
#include <brtenv.h>                            /* basic real-time environment */
#include <ds2211.h>
#include "Slv2211_ks_appl.slc"
/*******************************************************************************
  constant, macro and type definitions
*******************************************************************************/
#define DT  1e-3                                      /* simulation step size */
#define NCYL 8                                        /* number of cylinders */
/*******************************************************************************
  object declarations
*******************************************************************************/
extern unsigned long ks_appl[];      /* pointer to slave-DSP application data */
volatile dsfloat  exec_min;                      /* slave-DSP execution times */
volatile dsfloat  exec_max;
volatile dsfloat  exec_cur;
volatile dsfloat  exec_upd;
volatile dsfloat  exec_enbl;
volatile dsfloat  exec_noise;
```

```
/* cylinder parameter data arrays */
volatile dsfloat kn_freq[NCYL];                            /* knock frequency */
volatile dsfloat kn_ampl[NCYL];                            /* knock amplitude */
volatile dsfloat kn_damp[NCYL];                              /* knock damping */
volatile dsfloat kn_start[NCYL];                               /* knock start */
volatile dsfloat kn_length[NCYL];                             /* knock length */
volatile long    kn_number[NCYL];                             /* knock number */
volatile long    kn_rate[NCYL];                                 /* knock rate */
volatile long    kn_channel[NCYL];                             /* ADC channel */
volatile long    enable[NCYL];
volatile dsfloat apu_speed = 3000;                     /* engine speed in rpm */
volatile dsfloat apu_rad;                            /* engine speed in rad/s */
volatile dsfloat apu_read;                        /* angle position in degree */
volatile int auto_set = 0;                         /* parameter auto set flag */
volatile dsfloat noise[4];                                /* noise ampllitude */
volatile long prg[NCYL];                            /* test sequence number */
/* parameters for test sequence */
int seq_cnt[NCYL];
int seq_sgn[NCYL];
/*******************************************************************************
  function declarations
*******************************************************************************/
/*----------------------------------------------*/
void isr_t1()                             /* timer1 interrupt service routine */
{
  long i;
  long  enbl = 0;                                 /* knock signal enable mask */
  ts_timestamp_type ts;
  RTLIB_SRT_ISR_BEGIN();                                    /* overload check */
  ts_timestamp_read(&ts);
  host_service(1, &ts);                                    /* data acquisition service */
  /* parameter auto set */
  if(auto_set)
  {
    auto_set = 0;
    for(i = 0; i < NCYL; i++)
    {
      kn_freq[i] = 7500.0;
      kn_ampl[i] = 0.5;
      kn_damp[i] = 0.02;
      kn_start[i] = (720.0 / NCYL) * i;
      kn_length[i] = 90.0;
      kn_rate[i] = 1;
      kn_number[i] = 0;
      kn_channel[i] = 1;
    }
    noise[0] = 0.02;
    noise[1] = 0.02;
    noise[2] = 0.02;
    noise[3] = 0.02;
  }
  /* knock signal test sequence */
  for(i = 0; i < NCYL; i++)
  {
    switch (prg[i])
    {
      case 0 : seq_cnt[i] = 500;
               break;
      case 1 : kn_start[i] += 0.1;
               if(kn_start[i] >= 720.0)
                 kn_start[i] = 0.0;
               break;
      case 2 : kn_start[i] -= 0.1;
               if(kn_start[i] < 0.0)
                 kn_start[i] = 719.99;
               break;
      case 3 : if(seq_cnt[i] >= 1000)
               {
                 seq_sgn[i] = -1;
               }
               if(seq_cnt[i] < 0)
               {
                 seq_sgn[i] = 1;
               }
               kn_start[i] += (0.1 * (dsfloat)seq_sgn[i]);
               if(kn_start[i] >= 720.0)
                 kn_start[i] = 0.0;
               if(kn_start[i] < 0.0)
                 kn_start[i] = 719.99;
               seq_cnt[i] = seq_cnt[i] + seq_sgn[i];
               break;
    }
  }
  /* build knock signal enable mask */
  for(i = 0; i < NCYL; i++)
    enbl |= enable[i] << i;
  /* convert engine speed from rpm to rad/s */
  apu_rad = apu_speed * 0.1047197551197;
  ds2211_apu_velocity_write(DS2211_1_BASE, apu_rad);
  /* read angle position and convert to degree */
  ds2211_apu_position_read(DS2211_1_BASE, (dsfloat *)&apu_read);
  apu_read = apu_read * 57.29577951308;
  /* update knock signal parameters for all cylinders */
  for(i = 0; i < NCYL; i++)
  {
    RTLIB_TIC_START();                     /* start execution time mesurement */
    ds2211_slave_dsp_knock_update(DS2211_1_BASE, i+1, kn_channel[i],
        kn_freq[i], kn_ampl[i], kn_damp[i],
        kn_start[i], kn_length[i], kn_number[i], kn_rate[i]);
    exec_upd = RTLIB_TIC_READ() * 1.0e6;               /* read execution time */
  }
  RTLIB_TIC_START();                       /* start execution time mesurement */
  /* update noise amplitudes of knock sensors */
  ds2211_slave_dsp_knock_noise(DS2211_1_BASE, 1, noise[0]);
  exec_noise = RTLIB_TIC_READ() * 1.0e6;               /* read execution time */
  ds2211_slave_dsp_knock_noise(DS2211_1_BASE, 2, noise[1]);
  ds2211_slave_dsp_knock_noise(DS2211_1_BASE, 3, noise[2]);
  ds2211_slave_dsp_knock_noise(DS2211_1_BASE, 4, noise[3]);
  RTLIB_TIC_START();                       /* start execution time mesurement */
  /* write knock signal enable mask */
  ds2211_slave_dsp_signal_enable(DS2211_1_BASE, enbl);
  exec_enbl = RTLIB_TIC_READ() * 1.0e6;                /* read execution time */
  RTLIB_SRT_ISR_END();                    /* end of interrupt service routine */
}
/*-----------------------------------------------------------------------*/
void main()
{
  long  i;
  /* initialize knock parameters */
  for(i = 0; i < NCYL; i++)
  {
    kn_freq[i] = 7500.0;
    kn_ampl[i] = 0.5;
    kn_damp[i] = 0.02;
    kn_start[i] = (720.0 / NCYL) * i;
    kn_length[i] = 90.0;
    kn_rate[i] = 1;
    kn_number[i] = 0;
    kn_channel[i] = 1;
    prg[i] = 0;
    seq_cnt[i] = 500;
    seq_sgn[i] = 1;
  }
  noise[0] = 0.02;
  noise[1] = 0.02;
  noise[2] = 0.02;
  noise[3] = 0.02;
  init();                                                   /* init CPU board */
  ds2211_init(DS2211_1_BASE);                            /* init DS2211 board */
  /* load DS2211 slave-DSP application 'ks' */
  ds2211_slave_dsp_appl_load(DS2211_1_BASE, (Int32 *) &ks_appl);
  msg_info_set(0, 0, "System started");
  ds2211_apu_transformer_mode_set(DS2211_1_BASE, DS2211_APU_TRANSFORMER_ENABLE);
  /* set APU mode, set APU velocity and start APU */
  ds2211_mode_set(DS2211_1_BASE, DS2211_MASTER_MODE);
  ds2211_apu_velocity_write(DS2211_1_BASE, apu_speed * 0.1047197551197);
  ds2211_apu_start(DS2211_1_BASE);
  /* initialize knock signal generation */
  ds2211_slave_dsp_knock_init(DS2211_1_BASE, NCYL, (Int32 *)kn_channel,
        (dsfloat *)kn_freq, (dsfloat *)kn_ampl, (dsfloat *)kn_damp,
        (dsfloat *)kn_start, (dsfloat *)kn_length, (Int32 *)kn_number,
                                                              (Int32 *)kn_rate);
  /* update noise amplitudes of knock sensors */
  for(i = 0; i < 4; i++)
    ds2211_slave_dsp_knock_noise(DS2211_1_BASE, i+1, noise[i]);
  RTLIB_TIC_INIT();                      /* enable execution time measurement */
  RTLIB_SRT_START(DT, isr_t1);             /* initialize sampling clock timer */
  while(1)                                              /* background process */
  {
    while(msg_last_error_number() == MSG_NO_ERROR)
    {
      RTLIB_BACKGROUND_SERVICE();
      /* read execution time of slave-DSP application */
      ds2211_slave_dsp_speedchk(DS2211_1_BASE, (dsfloat *)&exec_min,
                                    (dsfloat *)&exec_max, (dsfloat *)&exec_cur);
    }
    RTLIB_SRT_DISABLE();                      /* disable sampling clock timer */
    while(msg_last_error_number() != MSG_NO_ERROR)
    {
      RTLIB_BACKGROUND_SERVICE();
    }
    msg_info_set(MSG_SM_DEFAULT, 0, "Error released.");
    RTLIB_SRT_ENABLE();                        /* enable sampling clock timer */
  }
}
```

Related topics

Examples

Example of Wheel Speed Sensor Simulation..318

ds2211_slave_dsp_knock_init

Syntax

```
void ds2211_slave_dsp_knock_init(
      Int32 base,
      Int32 cyl_max,
      Int32 *kn_channel,
      dsfloat *kn_freq,
      dsfloat *kn_ampl,
      dsfloat *kn_damp,
      dsfloat *kn_start,
      dsfloat *kn_length,
      Int32 *kn_number,
      Int32 *kn_rate)
```

Include file

`ds2211.h`

Purpose

To initialize all channels for the slave DSP knock sensor simulation for 1 … 8 cylinders.

I/O mapping

For information on the I/O mapping, refer to Knock Sensor Simulation (DS2211 Features 🕮).

Description

The specified parameters are passed to the slave DSP for knock sensor simulation. The knock signal u(t) will be generated according to the formula:

$$u(t) = a \cdot e^{-kn\_damp \cdot 2\pi \cdot kn\_freq \cdot t} \cdot \sin(2\pi \cdot kn\_freq \cdot t)$$

The product of `kn_rate · kn_number` must not exceed (2^{31} - 1). Use **`ds2211_slave_dsp_signal_enable`** to start signal generation. Use **`ds2211_slave_dsp_knock_noise`** to add an additional Gaussian noise to the knock signal.

The analog transformer outputs, also used by the slave DSP, could be enabled or disabled by using the `ds2211_apu_transformer_mode_set` function (see ds2211_apu_transformer_mode_set on page 39). After initialization, the outputs are disabled. For further information, refer to Transformer Outputs (APU and Slave DSP) (PHS Bus System Hardware Reference 🕮).

Note

It is possible to generate up to 8 knock signals on the 4 knock sensor channels.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

cyl_max Maximum number of cylinders within the range 1 … 8.

kn_channel Array (width = `cyl_max`) with the knock sensor channel number for each cylinder within the range 1 … 4.

kn_freq Array (width = `cyl_max`) with the frequency of the knock signals for each cylinder

kn_ampl Array (width = `cyl_max`) with the amplitudes of the knock signals for each cylinder. The values must be given within the range 0.0 … 1.0.

kn_damp Array (width = `cyl_max`) with the damping factors of the knock signals

kn_start Array (width = `cyl_max`) with the start angles of the knock signals in degrees within the range 0 … 719.99

kn_length Array (width = `cyl_max`) with the length of the knock signal in degrees within the range 0 … 359.0

kn_number Array (width = `cyl_max`) with the number of the knock signals to be generated. This parameter defines how often the knock signal will be generated for each cylinder. Take care not to exceed the following restriction: kn_rate · kn_number $\leq (2^{31} - 1)$

kn_rate Array (width = `cyl_max`) with the knock signal rate. The knock signal will be generated every 'kn_rate'th motor cycle. Take care not to exceed the following restriction: kn_rate · kn_number $\leq (2^{31} - 1)$

Return value

None

Messages

The following message is defined:

Type	Message	Meaning
Error	ds2211_slave_dsp_knock_init(0x??): slave-DSP is not responding!	Unable to request a semaphore for communication on the slave-DSP.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

Knock Sensor Simulation (DS2211 Features 🕮)

Examples

Example of Knock Sensor Simulation.. 308

References

ds2211_apu_transformer_mode_set.. 39
ds2211_slave_dsp_knock_noise.. 316
ds2211_slave_dsp_knock_update.. 314
ds2211_slave_dsp_signal_enable.. 300
Transformer Outputs (APU and Slave DSP) (PHS Bus System Hardware Reference 🕮)

ds2211_slave_dsp_knock_update

Syntax

```
Int32 ds2211_slave_dsp_knock_update(
      Int32 base,
      Int32 cylinder,
      Int32 kn_channel,
      dsfloat kn_freq,
      dsfloat kn_ampl,
      dsfloat kn_damp,
      dsfloat kn_start,
      dsfloat kn_length,
      Int32 kn_number,
      Int32 kn_rate)
```

Include file

`ds2211.h`

Purpose

To update the parameters of the slave DSP knock sensor simulation for the specified channel during application's run time.

I/O mapping

For information on the I/O mapping, refer to Knock Sensor Simulation (DS2211 Features 🕮).

Description

The parameters of the specified channel are passed to the slave DSP for knock sensor simulation. The knock signal u(t) will be generated according to the formula:

$u(t) = a \cdot e^{-kn\_damp \cdot 2\pi \cdot kn\_freq \cdot t} \cdot \sin(2\pi \cdot kn\_freq \cdot t)$

Use `ds2211_slave_dsp_knock_noise` to add an additional Gaussian noise to the knock signal.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

cylinder Number of the cylinder to be updated within the range 1 … 8

kn_channel Knock sensor channel number within the range 1 … 4

kn_freq Frequency of the knock signal

kn_ampl Amplitude of the knock signal within the range 0.0 … 1.0

kn_damp Damping factor of the knock signal

kn_start Start angle of the knock signal in degrees within the range 0.0 … 719.99

kn_length Length of the knock signal in degrees within the range 0 … 359.0

kn_number Number of the knock signals to be generated. Take care not to exceed the following restriction: kn_rate · kn_number ≤ $(2^{31} - 1)$

kn_rate Knock signal rate. The knock signal will be generated every 'kn_rate'th motor cycle. Take care not to exceed the following restriction: kn_rate · kn_number ≤ $(2^{31} - 1)$

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_SLVDSP_DPMEM_ACCESS_PASSED	The function has been performed without error.
DS2211_SLVDSP_DPMEM_ACCESS_FAILED	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

Knock Sensor Simulation (DS2211 Features 🕮)

Examples

Example of Knock Sensor Simulation.......... 308

References

ds2211_slave_dsp_knock_init.......... 312
ds2211_slave_dsp_knock_noise.......... 316
ds2211_slave_dsp_signal_enable.......... 300

ds2211_slave_dsp_knock_noise

Syntax

```
Int32 ds2211_slave_dsp_knock_noise(
      Int32 base,
      Int32 channel,
      dsfloat noise)
```

Include file

`ds2211.h`

Purpose

To add an additional Gaussian noise to a knock sensor signal defined by `ds2211_slave_dsp_knock_init`.

> **Note**
>
> Use `ds2211_slave_dsp_knock_update` to modify the other parameters of the knock signal.

I/O mapping

For information on the I/O mapping, refer to Knock Sensor Simulation (DS2211 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Knock sensor channel number within the range 1 … 4

noise Amplitude of the additional Gaussian noise signal within the range 0.0 … 1.0

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
`DS2211_SLVDSP_DPMEM_ACCESS_PASSED`	The function has been performed without error.
`DS2211_SLVDSP_DPMEM_ACCESS_FAILED`	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Basics

Knock Sensor Simulation (DS2211 Features 📖)

Examples

Example of Knock Sensor Simulation..308

References

ds2211_slave_dsp_knock_init..312
ds2211_slave_dsp_knock_update..314

Wheel Speed Sensor Simulation

Introduction

You can simulate a wheel speed sensor using a ready-to-use application implemented on the slave DSP.

For general information on wheel speed sensor simulation and its I/O mapping, refer to Wheel Speed Sensor Simulation (DS2211 Features 📖).

Where to go from here

Information in this section

Example of Wheel Speed Sensor Simulation

Example

The following example (`Slv_wheel_2211_hc.c`) shows how to use the wheel speed sensor simulation.

```
#include <brtenv.h>                            /* basic real-time environment */
#include <ds2211.h>
#include "Slv2211_wheel_appl.slc"
```

```
#define DT  10e-3                                     /* simulation step size */
```

```
/* variables for execution time profiling */
extern unsigned long wheel[];
volatile dsfloat  exec_min;
volatile dsfloat  exec_max;
volatile dsfloat  exec_cur;
```

```
volatile dsfloat  exec_upd;
volatile dsfloat  exec_enbl;
```

```
volatile int enable1 = 0;
volatile int enable2 = 0;
volatile int enable3 = 0;
volatile int enable4 = 0;
volatile int enable = 0;
volatile dsfloat amplitude[4] = {0.5, 0.5, 0.5, 0.5};
volatile dsfloat noise[4] = {0.0, 0.0, 0.0, 0.0};
volatile dsfloat speed[4] = {139.6263401595, 139.6263401595, 139.6263401595, 139.6263401595};
volatile Int32   teeth[4] = {45, 45, 45, 45};
volatile dsfloat freq[4];

/*----------------------------------------------*/
void isr_t1()                             /* timer1 interrupt service routine */
{
  long i;
  ts_timestamp_type ts;

  RTLIB_SRT_ISR_BEGIN();                                    /* overload check */

  ts_timestamp_read(&ts);
  host_service(1, &ts);

  enable = enable1 | (enable2 << 1) | (enable3 << 2) | (enable4 << 3);

  for(i = 0; i < 4; i++)
  {
    /* calculate frequency values */
    freq[i] = speed[i] * teeth[i] * 0.1591549430919;

    RTLIB_TIC_START();                     /* start execution time mesurement */
    /* update wheel speed parameter */
    ds2211_slave_dsp_wheel_update(DS2211_1_BASE, i+1, amplitude[i], speed[i],
                                                            teeth[i], noise[i]);
    exec_upd = RTLIB_TIC_READ() * 1.0e6;               /* read execution time */
  }

  RTLIB_TIC_START();                       /* start execution time mesurement */
  /* update enable mask */
  ds2211_slave_dsp_signal_enable(DS2211_1_BASE, enable);
  exec_enbl = RTLIB_TIC_READ() * 1.0e6;                /* read execution time */

  RTLIB_SRT_ISR_END();                    /* end of interrupt service routine */
}

/*----------------------------------------------*/
void main(void)
{
  init();                                                   /* init CPU board */
  ds2211_init(DS2211_1_BASE);                            /* init DS2211 board */

  /* load DS2211 slave-DSP application */
  ds2211_slave_dsp_appl_load(DS2211_1_BASE, (Int32 *) &wheel_appl);

  msg_info_set(0, 0, "System started");

  ds2211_apu_transformer_mode_set(DS2211_1_BASE, DS2211_APU_TRANSFORMER_ENABLE);

  /* initialize wheel speed generation */
  ds2211_slave_dsp_wheel_init(DS2211_1_BASE, (dsfloat *)amplitude,
                            (dsfloat *)speed, (Int32 *)teeth, (dsfloat *)noise);

  RTLIB_TIC_INIT();                      /* enable execution time measurement */

  RTLIB_SRT_START(DT, isr_t1);             /* initialize sampling clock timer */
  while(1)                                                /* background process */
  {
    while(msg_last_error_number() == MSG_NO_ERROR)
    {
      /* read slave-DSP execution time */
      ds2211_slave_dsp_speedchk(DS2211_1_BASE, (dsfloat *)&exec_min,
                                    (dsfloat *)&exec_max, (dsfloat *)&exec_cur);
      RTLIB_BACKGROUND_SERVICE();
    }
    RTLIB_SRT_DISABLE();                          /* disable sampling clock timer */
    while(msg_last_error_number() != MSG_NO_ERROR)
    {
      RTLIB_BACKGROUND_SERVICE();
    }
    msg_info_set(MSG_SM_DEFAULT, 0, "Error released.");
    RTLIB_SRT_ENABLE();                            /* enable sampling clock timer */
  }
}
```

Related topics

Examples

Example of Knock Sensor Simulation..308

ds2211_slave_dsp_wheel_init

Syntax

```
void ds2211_slave_dsp_wheel_init(
      Int32 base,
      dsfloat *amplitude,
      dsfloat *speed,
      Int32 *teeth,
      dsfloat *noise)
```

Include file

`ds2211.h`

Purpose

To initialize the parameters for slave DSP wheel speed sensor simulation.

I/O mapping

For information on the I/O mapping, refer to Wheel Speed Sensor Simulation (DS2211 Features 📖).

Description

This function initializes all four channels at the same time. Use `ds2211_slave_dsp_signal_enable` to start signal generation.

The wheel speed signal u(t) is generated according to the formula

u(t) = a · sin(2 · π · f_wheel · t) + Noise.

The wheel speed signal frequency (f_wheel) is calculated as follows:

f_wheel = speed · teeth · (1/2π)

The slave DSP´s analog outputs used by the wheel speed signal generator may be enabled or disabled by software (refer to `ds2211_apu_transformer_mode_set` on page 39).

> **Note**
>
> After initialization, the outputs are disabled. For further information, refer to Transformer Outputs (APU and Slave DSP) (PHS Bus System Hardware Reference 📖).

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

amplitude Array with the amplitudes of the 4 wheel speed signals within the range 0.0 … 1.0

speed Array with 4 speed values in rad/s

teeth Array with 4 numbers of sensor teeth

noise Array with the amplitudes of additional Gaussian noise signals for the four channels within the range 0.0 … 1.0

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of Wheel Speed Sensor Simulation	318

References

ds2211_apu_transformer_mode_set	39
ds2211_slave_dsp_signal_enable	300
ds2211_slave_dsp_wheel_update	322
Transformer Outputs (APU and Slave DSP) (PHS Bus System Hardware Reference 📖)	

ds2211_slave_dsp_wheel_update

Syntax

```
Int32 ds2211_slave_dsp_wheel_update(
      Int32 base,
      Int32 channel,
      dsfloat amplitude,
      dsfloat speed,
      Int32 teeth,
      dsfloat noise)
```

Include file

`ds2211.h`

Purpose

To set new parameters for one channel of the wheel speed sensor simulation during the application’s run time.

I/O mapping

For information on the I/O mapping, refer to Wheel Speed Sensor Simulation (DS2211 Features 📖).

Description

The wheel speed signal u(t) is generated according to the formula

$u(t) = a \cdot \sin(2 \cdot \pi \cdot \text{f\_wheel} \cdot t) + \text{Noise}$.

The wheel speed signal frequency (f_wheel) is calculated as follows:

$\text{f\_wheel} = \text{speed} \cdot \text{teeth} \cdot (1/2\pi)$

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

channel Channel number of the wheel speed signal within the range 1 … 4

amplitude Amplitude of the wheel speed signal within the range of 0.0 … 1.0

speed Speed value in rad/s

teeth Number of sensor teeth

noise Amplitudes of an additional Gaussian noise signal within the range 0.0 … 1.0.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_SLVDSP_DPMEM_ACCESS_PASSED	The function has been performed without error.
DS2211_SLVDSP_DPMEM_ACCESS_FAILED	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

Examples

Example of Wheel Speed Sensor Simulation..318

References

ds2211_slave_dsp_signal_enable..300
ds2211_slave_dsp_wheel_init...320

Slave DSP Memory Access Functions

Where to go from here

Information in this section

Basics of Accessing the Slave DSP Memory

Basics

You can exchange data between the master and the slave DSP directly by accessing the dual-port memory. There are two different ways to perform this access:

- Some access functions request and release a semaphore automatically. This means that semaphore handling is performed for each call of the access function.

- Request the semaphore, perform several other activities, and release the semaphore afterwards. To do so, you have to handle the semaphores with special functions.

Example of Slave DSP Memory Access Functions

Example

This example shows how to explicitly access the dual-port memory and handle the semaphore.

```
sem_state = ds2211_slave_dsp_sem_req(DS2211_1_BASE, 2);
/* if the request was successful */
if(!sem_state)
{
   /* read from DS2211 slave DSP */
   ds2211_slave_dsp_read_direct(DS2211_1_BASE, 20,
                               (long *)&respond);
   /*write to DS2211 slave */
   ds2211_slave_dsp_write_direct(DS2211_1_BASE, 0,
                               (long *) &fct_nr);
   /* release semaphore */
   ds2211_slave_dsp_sem_rel(DS2211_1_BASE, 2);
}
```

Related topics

References

ds2211_slave_dsp_read_direct........ 332
ds2211_slave_dsp_sem_rel........ 331
ds2211_slave_dsp_sem_req........ 330
ds2211_slave_dsp_write_direct........ 333

ds2211_slave_dsp_read

Syntax

```
Int32 ds2211_slave_dsp_read(
      Int32 base,
      Int32 sem_nr,
      Int32 offset,
      void *value)
```

Include file

`ds2211.h`

Purpose

To read a data value from the DPMEM.

Description

`ds2211_slave_dsp_read` reads the DPMEM location specified by the `offset` parameter and returns the data via the `value` parameter. DPMEM access is handled by the semaphore specified by the `sem_nr` parameter. For this reason, `ds2211_slave_dsp_read` has to be polled until it returns DS2211_SLVDSP_DPMEM_ACCESS_PASSED.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

sem_nr Number of the semaphore to be used for DPMEM access within the range 1 … 16

offset DPMEM address offset within the range 0x0000 … 0x3FFF. Data will be read from this location. The offset specifies the DPMEM location in the slave DSP's memory map. Refer to Memory Map of the Slave DSP on page 343.

value Address in the master processor memory where the data value is written (the data type can be float or long). If the processor board uses the IEEE floating-point format, the format of floating-point values has to be converted. Refer to Basic Communication Principles on page 298.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
`DS2211_SLVDSP_DPMEM_ACCESS_PASSED`	The function was performed without error.
`DS2211_SLVDSP_DPMEM_ACCESS_FAILED`	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

Example

```
/* read value from DS2211 board */
error = ds2211_slave_dsp_read (DS2211_1_BASE, 1, 0,
                               (UInt32 *)&value);
```

ds2211_slave_dsp_write

Syntax

```
Int32 ds2211_slave_dsp_write(
      Int32 base,
      Int32 sem_nr,
      Int32 offset,
      void *value)
```

Include file

`ds2211.h`

Purpose

To write a data value to the DPMEM.

Description

`ds2211_slave_dsp_write` writes the contents of the `value` parameter to the DPMEM location specified by the `offset` parameter. DPMEM access is handled by the semaphore specified by the `sem_nr` variable. For this reason, `ds2211_slave_dsp_write` has to be polled until it returns DS2211_SLVDSP_DPMEM_ACCESS_PASSED.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

sem_nr Number of the semaphore to be used for DPMEM access within the range 1 … 16

offset DPMEM address offset within the range 0x0000 … 0x3FFF. Data will be written to this location. The offset specifies the DPMEM location in the slave DSP's memory map. Refer to Memory Map of the Slave DSP on page 343.

value Address in the master processor memory where the data value to be written is stored (can be either float or long). If the processor board uses the IEEE floating-point format, the format of floating-point values has to be converted. Refer to Basic Communication Principles on page 298.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_SLVDSP_DPMEM_ACCESS_PASSED	The function was performed without error.
DS2211_SLVDSP_DPMEM_ACCESS_FAILED	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

Example

```
/* write value to DS2211 board */
error = ds2211_slave_dsp_write(DS2211_1_BASE, 1, 0,
                               (UInt32 *)&value);
```

ds2211_slave_dsp_block_read

Syntax

```
Int32 ds2211_slave_dsp_block_read(
      Int32 base,
      Int32 sem_nr,
      Int32 offset,
      Int32 count,
      void *value)
```

Include file

`ds2211.h`

Purpose

To read a block of data values from the DPMEM.

Description

`ds2211_slave_dsp_block_read` reads a block of values (specified by the `count` parameter). The function starts with the DPMEM location specified by the `offset` parameter and returns the data via the `value` parameter. DPMEM access is handled by the semaphore specified by the variable `sem_nr`. For this reason, `ds2211_slave_dsp_block_read` has to be polled until it returns DS2211_SLVDSP_DPMEM_ACCESS_PASSED.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

sem_nr Number of the semaphore to be used for DPMEM access within the range 1 … 16

offset DPMEM address offset within the range 0x0000 … 0x3FFF. The data block to be read starts with this memory location. The offset specifies the DPMEM location in the slave DSP's memory map. Refer to Memory Map of the Slave DSP on page 343.

count Number of data values to be read

value Address in the master processor memory where the data values are written (can be either float or long). If the processor board uses the IEEE floating-point format, the format of floating-point values has to be converted. Refer to Basic Communication Principles on page 298.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
`DS2211_SLVDSP_DPMEM_ACCESS_PASSED`	The function was performed without error.
`DS2211_SLVDSP_DPMEM_ACCESS_FAILED`	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

ds2211_slave_dsp_block_write

Syntax

```
Int32 ds2211_slave_dsp_block_write(
      Int32 base,
      Int32 sem_nr,
      Int32 offset,
      Int32 count,
      void *value)
```

Include file

`ds2211.h`

Purpose

To write a block of data values to the DPMEM.

Description

`ds2211_slave_dsp_block_write` writes a number of values (specified by the `count` parameter) to the DPMEM starting with the address specified by the `offset` parameter. The `value` parameter points to the memory location of the master processor where the data values are stored.

Access to the DPMEM is handled by a semaphore specified by the `sem_nr` variable. For this reason, `ds2211_slave_dsp_block_write` has to be polled until it returns DS2211_SLVDSP_DPMEM_ACCESS_PASSED.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

sem_nr Number of the semaphore to be used for DPMEM access within the range 1 … 16

offset DPMEM address offset within the range 0x0000 … 0x3FFF. The offset specifies the DPMEM location in the slave DSP's memory map. Refer to Memory Map of the Slave DSP on page 343.

count Number of data values to be written

value Address in the master processor memory where the data values to be written are stored (the data type can be float or long). If the processor board uses the IEEE floating-point format, the format of floating-point values has to be converted. Refer to Basic Communication Principles on page 298.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_SLVDSP_DPMEM_ACCESS_PASSED	The function was performed without error.
DS2211_SLVDSP_DPMEM_ACCESS_FAILED	The DPMEM could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 551.

ds2211_slave_dsp_sem_req

Syntax

```
Int32 ds2211_slave_dsp_sem_req(
      Int32 base,
      Int32 sem_nr)
```

Include file

`ds2211.h`

Purpose

To request a semaphore for slave DSP access.

Description

Use this function for `ds2211_slave_dsp_read_direct`, `ds2211_slave_dsp_write_direct`, `ds2211_slave_dsp_block_read_di`, and `ds2211_slave_dsp_block_write_di`. Use `ds2211_slave_dsp_sem_rel` to release the semaphore. A request for the semaphores is sent. If the request fails, the semaphore is released automatically and DS2211_SLVDSP_DPMEM_ACCESS_FAILED is returned. In this case `ds2211_slave_dsp_sem_rel` does not have to be called.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

sem_nr Number of the semaphore to be requested within the range 1 … 16

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_SLVDSP_DPMEM_ACCESS_PASSED	The semaphore was requested successfully.
DS2211_SLVDSP_DPMEM_ACCESS_FAILED	The request failed and the semaphore request is released.

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_slave_dsp_block_read_di	334
ds2211_slave_dsp_block_write_di	335
ds2211_slave_dsp_read_direct	332
ds2211_slave_dsp_sem_rel	331
ds2211_slave_dsp_write_direct	333

ds2211_slave_dsp_sem_rel

Syntax

```
void ds2211_slave_dsp_sem_rel(
     Int32 base,
     Int32 sem_nr)
```

Include file

`ds2211.h`

Purpose

To release a requested semaphore.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

sem_nr Number of the semaphore to be released within the range 1 … 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_slave_dsp_block_read_di	334
ds2211_slave_dsp_block_write_di	335
ds2211_slave_dsp_read_direct	332
ds2211_slave_dsp_sem_req	330
ds2211_slave_dsp_write_direct	333

ds2211_slave_dsp_read_direct

Syntax

```
void ds2211_slave_dsp_read_direct(
      Int32 base,
      Int32 offset,
      void *value)
```

Include file

ds2211.h

Purpose

To read a data value from the DPMEM.

Description

The DPMEM location specified by the `offset` parameter is read and returned by the `value` parameter.

> **Note**
>
> In this function the access to the DPMEM is not handled by a semaphore. For this reason, you have to call `ds2211_slave_dsp_sem_req` before invoking `ds2211_slave_dsp_read_direct` and `ds2211_slave_dsp_sem_rel` to release the semaphore after accessing the DPMEM.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

offset DPMEM address offset within the range 0x0000 … 0x3FFF. The offset specifies the DPMEM location in the slave DSP's memory map. Refer to Memory Map of the Slave DSP on page 343.

value Address in the master processor memory where the data value is written (the data type can be float or long). If the processor board uses the IEEE floating-point format, the format of floating-point values has to be converted. Refer to Basic Communication Principles on page 298.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_slave_dsp_sem_rel..331
ds2211_slave_dsp_sem_req...330

ds2211_slave_dsp_write_direct

Syntax

```
void ds2211_slave_dsp_write_direct(
      Int32 base,
      Int32 offset,
      void *value)
```

Include file

`ds2211.h`

Purpose

To write a data value to the DPMEM.

Description

The contents of the `value` parameter are written to the DS2211 DPMEM location specified by the `offset` parameter.

Note

In this function the access to the DPMEM is *not* handled by a semaphore. For this reason, you have to call `ds2211_slave_dsp_sem_req` before invoking `ds2211_slave_dsp_write_direct` and `ds2211_slave_dsp_sem_rel` to release the semaphore after accessing the DPMEM.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

offset DPMEM address offset within the range 0x0000 … 0x3FFF. The offset specifies the DPMEM location in the slave DSP's memory map. Refer to Memory Map of the Slave DSP on page 343.

value Address in the master processor memory where the data value to be written is stored (can be either float or long). If the processor board uses the IEEE floating-point format, the format of floating-point values has to be converted. Refer to Basic Communication Principles on page 298.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_slave_dsp_block_read_di

Syntax

```
void ds2211_slave_dsp_block_read_di(
      Int32 base,
      Int32 offset,
      Int32 count,
      void *value)
```

Include file

`ds2211.h`

Purpose

To read a block of data values from the DPMEM.

Description

A block of `count` data values is read from the DS2211 slave DSP's DPMEM starting at the address specified by the `offset` parameter. The data values are stored in a data block pointed to by the `data` parameter.

Note

In this function the access to the DPMEM is *not* handled by the semaphore. For this reason, you have to call `ds2211_slave_dsp_sem_req` before invoking `ds2211_slave_dsp_block_read_di` and `ds2211_slave_dsp_sem_rel` to release the semaphore after accessing the DPMEM.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

offset DPMEM address offset within the range 0x0000 … 0x3FFF. The offset specifies the DPMEM location in the slave DSP's memory map. Refer to Memory Map of the Slave DSP on page 343.

count Number of data values to be read

value Address in the master processor memory where the data values are written (can be either float or long). If the processor board uses the IEEE floating-point format, the format of floating-point values has to be converted. Refer to Basic Communication Principles on page 298.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_slave_dsp_block_write_di

Syntax

```
void ds2211_slave_dsp_block_write_di(
      Int32 base,
      Int32 offset,
      Int32 count,
      void *value)
```

Include file

`ds2211.h`

Purpose

To write a block of data values to the DPMEM.

Description

A block of `count` data values is written to the DPMEM starting at the address specified by the parameter `offset`. The `value` parameter points to the memory location where the data values are stored.

Note

In this function the access to the DPMEM is *not* handled by the semaphore. For this reason, you have to call `ds2211_slave_dsp_sem_req` before invoking `ds2211_slave_dsp_block_write_di` and `ds2211_slave_dsp_sem_rel` to release the semaphore after accessing the DPMEM.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

offset DPMEM address offset within the range 0x0000 … 0x3FFF. The offset specifies the DPMEM location in the slave DSP's memory map. Refer to Memory Map of the Slave DSP on page 343.

count Number of data values to be written

value Address in the master processor memory where the data values to be written are stored (the data type can be float or long). If the processor board uses the IEEE floating-point format, the format of floating-point values has to be converted. Refer to Basic Communication Principles on page 298.

Return value

None

Execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_slave_dsp_block_read_di........334
ds2211_slave_dsp_sem_rel........331
ds2211_slave_dsp_sem_req........330

Slave DSP Functions and Macros

Introduction

In addition to the standard functions for the master processor board accessing the slave DSP, you can use a set of macros and functions to write applications running on the slave DSP TMS320VC33.

Where to go from here

Information in this section

Information in other sections

Slave DSP Access Functions..297
To access the slave DSP and generate knock sensor signals or wheel speed sensor signals.

Slave DSP Basics

Introduction

The following information and features are provided for you to write your own applications for the slave DSP TMS320VC33.

Where to go from here

Information in this section

Basics for Programming the Slave DSP

Introduction

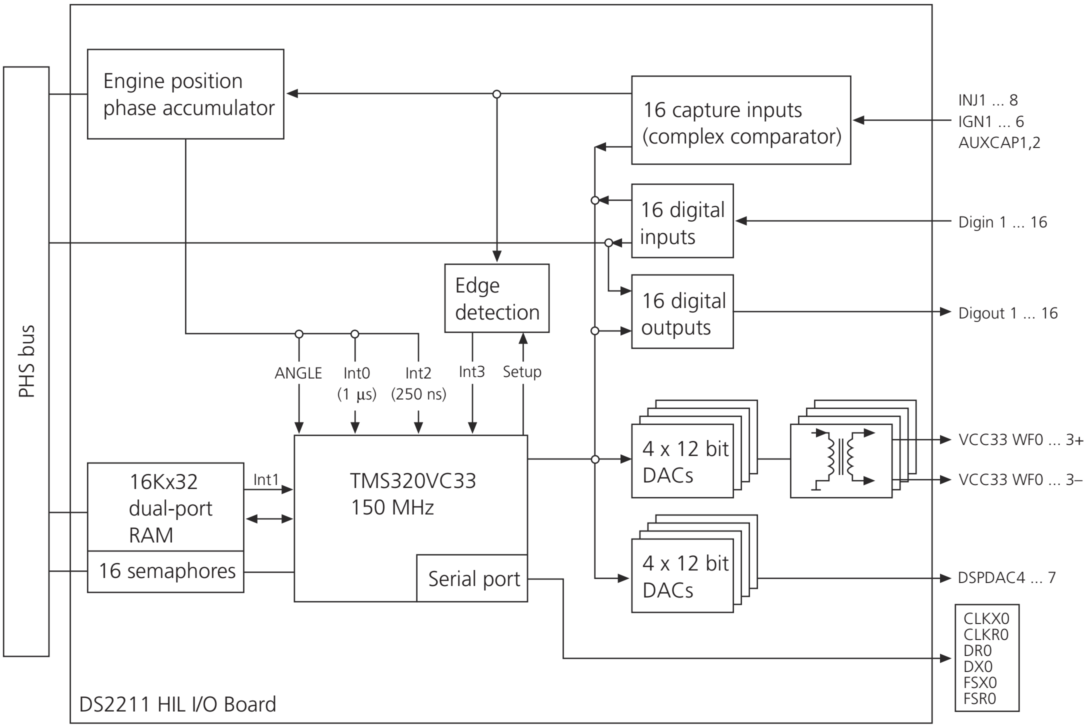
You can use the following features to write your own applications for the slave DSP.

Note

You cannot use your own applications and the standard applications (knock sensor and wheel speed sensor simulation) at the same time.

Block diagram

The following illustration shows the block diagram of the slave DSP.



Identifiers and constants

There are predefined identifiers and constants to make programming easier. Using these symbols means you do not have to know the memory locations for the predefined functions and macros, refer to Definitions on page 345.

Interrupts

The slave DSP supports multiple internal and external interrupts, which can be used for a variety of applications. Internal interrupts are generated by the DMA controller, the timers and the serial interface. Four external maskable interrupts (INT0, INT1, INT2 and INT3) are supported. Interrupts are automatically prioritized, allowing them to occur simultaneously and be serviced in a predefined order.

Note

Interrupt service routines must use the naming conventions (c_int*nn*) as described in the *TMS320 Floating-Point DSP Optimizing C Compiler User's Guide* from Texas Instruments. Using one of these function names defines an interrupt routine. When the compiler encounters one of these function names, it generates the necessary code automatically.

There are functions to enable the interrupts in the interrupt enable register (IE) and to enable interrupts globally in the status register (ST) of the TMS320VC33.

For the interrupts used on the DS2211 and the alias names, refer to Interrupts on page 349.

Timer 0 and 1

The slave DSP has two 32-bit general-purpose timer modules (timer0 and timer1). The timer interrupts are part of the internal interrupts (TINT0 and TINT1), refer to timer0, timer1 on page 348.

Error flag

To indicate an error state, you can use the error flag of the DSP, refer to Error Handling on page 363.

DSP state

You can indicate the state of the DSP with on-board LEDs, refer to Status LEDs on page 365.

D/A converters

The slave DSP provides eight 12-bit D/A converters. To use these converters, you have to scale floating-point values and convert them to the long integer data type, refer to D/A Converter (Slave DSP) on page 367.

Digital I/O

You can use six digital I/O lines. Each line can be configured as input or output (refer to Digital I/O via Serial Port on page 371). The digital I/O pins are shared with serial interface pins. You can also access the 16 bit digital I/O units and the 16 capture inputs (complex comparator).

Master to slave communication

Communication between the master processor and the slave DSP is via the DPMEM of the DS2211. If access to this DPMEM is not arbitrated by hardware, you can avoid conflicts when the DPMEM is accessed from both sides, by the TMS320VC33 DSP and the master processor, by using one of 16 semaphores (1 … 16) provided by the master processor. For instructions on using the semaphores, refer to DPMEM Access Functions on page 384.

DMA access

The on-chip DMA controller can read from or write to any location in the slave DSP's memory without interfering with the CPU operation. The slave DSP can interface to slow external memories and peripherals without reducing throughput to the CPU.

A DMA operation consists of a block or single-word transfer to or from memory, refer to Direct Memory Access on page 389.

Serial interface

The slave DSP provides one serial interface to connect a DS2302, a DS2210 or another DS2211 board. For information on initializing and using the serial interface, refer to Serial Interface on page 396.

Execution time measurement

The execution times of slave DSP applications can be measured with the functions and macros described in Execution Time Measurement on page 412.

Programming environment

The software and tools needed to write your slave DSP application are provided on the dSPACE DVD:

- For information on host PC settings (for example, variables and folder structures), refer to Host PC Settings on page 418.
- Batch files, makefiles and linker command files are available to customize the software environment (refer to Batch Files, Makefiles, Linker Command Files on page 423).
- To calculate the execution times of slave DSP applications, use the speed check utility described in Execution Time Information on page 429.
- There are some ways to optimize the assembly code of your slave DSP application, refer to Assembly Code Optimization on page 432.
- For information on loading applications to the slave DSP, refer to Loading Slave Applications on page 437.

Further information

For more information on the TMS320VC33 slave DSP, refer to the Texas Instruments web site at "http://www.ti.com" and search for the *TMS320C3x User's Guide* (literature number SPRU031F) and *TMS320VC33 Digital Signal Processor* (literature number SPRS087E).

Related topics

References

D/A Converter (Slave DSP)	367
Definitions	345
Digital I/O via Serial Port	371
Direct Memory Access	389
DPMEM Access Functions	384
Error Handling	363
Interrupts	349
Serial Interface	396
Status LEDs	365
timer0, timer1	348

Memory Map of the Slave DSP

Introduction

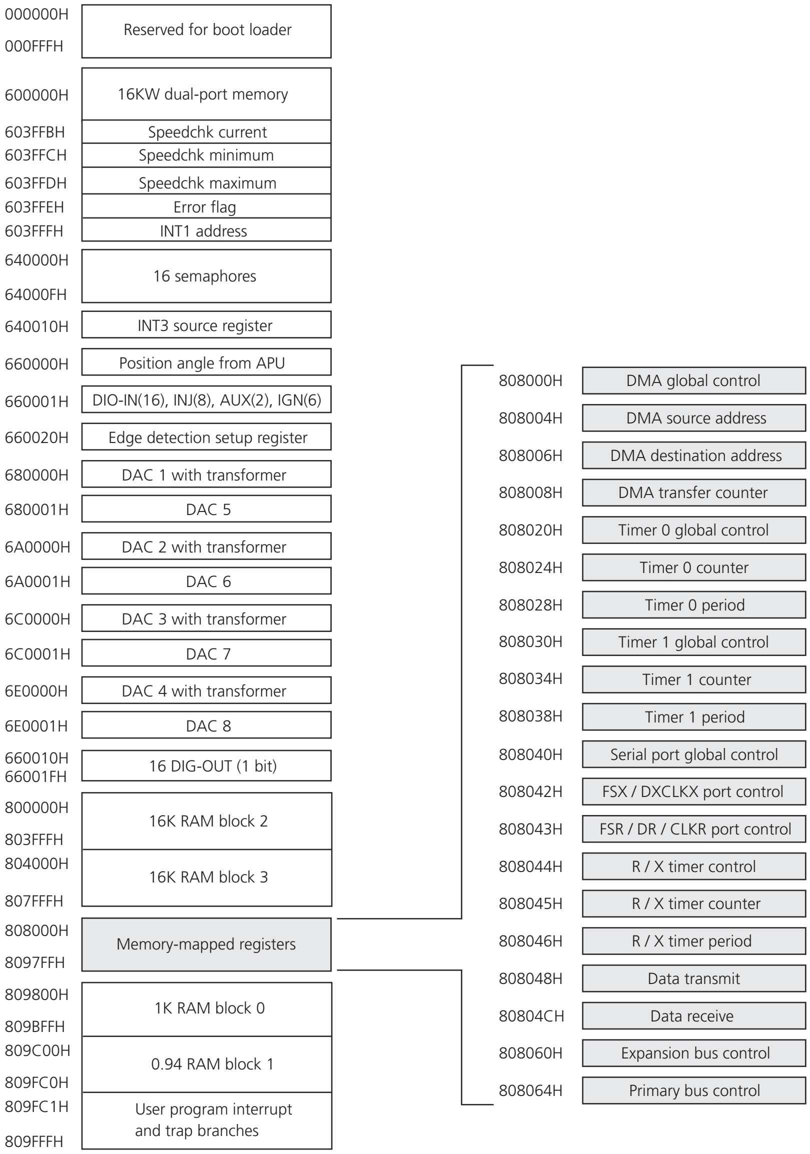
This topic shows the memory map of the slave DSP (digital signal processor, Texas Instruments TMS320VC33) on the DS2211.

You need to consider the various memory map ranges when writing applications for the slave DSP.

Memory map

Communication between the master processor and the slave DSP is via the 16-KW DPMEM of the DS2211, see Slave DSP Access Functions on page 297. DPMEM address is specified by the offset in relation to the start address (600000H) of the DPMEM.

The following illustration shows the memory map of the Texas Instruments TMS320VC33 digital signal processor.



000000H 000FFFH	Reserved for boot loader
600000H	16KW dual-port memory
603FFBH	Speedchk current
603FFCH	Speedchk minimum
603FFDH	Speedchk maximum
603FFEH	Error flag
603FFFH	INT1 address
640000H 64000FH	16 semaphores
640010H	INT3 source register
660000H	Position angle from APU
660001H	DIO-IN(16), INJ(8), AUX(2), IGN(6)
660020H	Edge detection setup register
680000H	DAC 1 with transformer
680001H	DAC 5
6A0000H	DAC 2 with transformer
6A0001H	DAC 6
6C0000H	DAC 3 with transformer
6C0001H	DAC 7
6E0000H	DAC 4 with transformer
6E0001H	DAC 8
660010H 66001FH	16 DIG-OUT (1 bit)
800000H 803FFFH	16K RAM block 2
804000H 807FFFH	16K RAM block 3
808000H 8097FFH	Memory-mapped registers
809800H 809BFFH	1K RAM block 0
809C00H 809FC0H	0.94 RAM block 1
809FC1H 809FFFH	User program interrupt and trap branches

808000H	DMA global control
808004H	DMA source address
808006H	DMA destination address
808008H	DMA transfer counter
808020H	Timer 0 global control
808024H	Timer 0 counter
808028H	Timer 0 period
808030H	Timer 1 global control
808034H	Timer 1 counter
808038H	Timer 1 period
808040H	Serial port global control
808042H	FSX / DXCLKX port control
808043H	FSR / DR / CLKR port control
808044H	R / X timer control
808045H	R / X timer counter
808046H	R / X timer period
808048H	Data transmit
80804CH	Data receive
808060H	Expansion bus control
808064H	Primary bus control

Related topics

References

Definitions

Introduction

To make programming easier, there are predefined identifiers and variables.

Where to go from here

Information in this section

Identifiers for Numerical Constants

Introduction

The identifiers listed in the table below are defined in the `Ds2211.h` header file. These definitions are used by the standard functions and macros described in the following sections. When writing your own application you should use these identifiers as well.

Identifiers

Identifier	Value	Meaning
SCAL	2147483648.0	Scaling factor for D/A output values
TIMER_CLOCK	(clock_per_sec/2.0) = 20 MHz	Timer clock rate of DS2211 board
ERROR_FLAG	0x603FFE	Address of the error flag
SPEEDCHK_MAX	0x603FFD	Address of the maximum execution time for the speed_check macro
SPEEDCHK_MIN	0x603FFC	Address of the minimum execution time for the speed_check macro
SPEEDCHK_CUR	0x603FFB	Address of the current execution time for the speed_check macro
INT_TBL_ADDR	0x809FC0	Start address of the interrupt vector table
INT1_ADDR	0x603FFF	Reserved DPMEM address for host interrupt INT1
SEMAPHORE1	0x640000	Semaphore addresses
…	…	
SEMAPHORE16	0x64000F	
dp_mem	((volatile float_or_int *) 0x00600000)	Pointer to the local DPMEM. The pointer is of a union type in order to transfer floating-point values as well as integer values. For example, the memory

Identifier	Value	Meaning
		location j is accessed by dp_mem[j].f for a value of type float or dp_mem[j].i for an integer value.
DP_MEM_BASE	0x600000	Base address of the DPMEM
DP_MEM_SIZE	0x4000	Size of the DPMEM

Pointer Declarations and Global Variables

Introduction

The following declarations of global pointers and variables are defined in the file `Init.c`. These variables are used by the standard functions and macros described in the following sections. When writing your own application you can use these identifiers as well.

Note

Each initialized pointer requires 3 words in the .cinit section and an additional word in the .bss section (refer to DS2211.lk, DS2211_1.lk, DS2211_2.lk on page 425).

Pointer	Meaning
long *dac1	Pointers to the D/A converter output registers
…	
long *dac8	
long *int_tbl	Pointer to the interrupt vector table
float clock_per_sec	Slave DSP clock frequency (40.0 MHz)
float time_per_tick	Slave DSP timer period (2.0/clock_per_sec = 50 ns)
long *angle_pos	Pointer to the crankshaft position angle value of the angular processing unit
long *int1	Pointer to the DPMEM location 0x00603FFF reserved for the external interrupt INT1
long *error	Pointer to the error flag at the DPMEM location 0x00603FFE

Initialization

Introduction

Before you can use the DSP and the built-in timers you have to perform an initialization process.

Where to go from here

Information in this section

init

Syntax

```
void init()
```

Include file

`Init.h`

Purpose

To initialize the slave DSP as follows:

- Reset the hardware system
- Clear pending interrupts
- Initialize global pointers and variables
- Activate the status LED connected to the slave DSP's XF0 I/O line

Note

You have to call this function in each user application at the beginning of the `main()` routine.

Return value

None

timer0, timer1

Syntax

```
void timer0(float time)
void timer1(float time)
```

Include file

`Timer0.h`, `Timer1.h`

Purpose

To initialize the built-in timer0 or timer1 of the DSP as follows:

- Generate timer interrupts at the sampling rate specified by the parameter time.
- Set the appropriate interrupt vector to point to the corresponding timer interrupt service routine c_int09 for timer0 or c_int10 for timer1.
- Enable the corresponding timer interrupt TINT0 or TINT1 in the interrupt enable register (IE).
- Enable interrupts globally in the status register (ST).

When you call timer0 or timer1 the corresponding timer starts immediately to generate timer interrupts at the specified sampling rate.

> **Tip**
>
> You can use the alias name isr_t0 instead of c_int09 for the timer0 interrupt service routine and the alias name isr_t1 instead of c_int10 for the timer1 interrupt service routine (refer to Interrupts on page 349).

Parameters

time Lets you specify the required sampling period in seconds at which timer interrupts will be generated.

Return value

None

Interrupts

Where to go from here

Information in this section

Basic of Slave DSP Interrupts

Overview

The slave DSP supports the following interrupts:

Interrupt	Service Routine Name		Description
	Internal Name	Alias	
INT0	`c_int01()`	`isr_int0`	Interrupt INT0 is triggered by the angle processing unit (APU) when the crankshaft angle value has been updated (every 1 µs).
INT1	`c_int02()`	`isr_int1`	Master to DSP interrupt triggered by writing to the DPMEM address 0x63FFF (as seen from the master). The value being written to the DPMEM can be used for interrupt-driven data transfer, for example. This interrupt is initialized by the function int1_init, refer to int0_init, ... , int3_init on page 352.
INT2	`c_int03()`	`isr_int2`	Interrupt INT2 is triggered by the Angle Processing Unit when the crankshaft angle value has been updated every 250 ns.
INT3	`c_int04()`	`isr_int3`	Edge detection interrupt triggered by the complex comparators. To enable the INT3 request on leading or trailing edges of the complex comparator outputs, refer to edge_int_enable on page 380.
XINT0	`c_int05()`	`isr_transmit`	Transmit interrupt of DSP's serial interface (refer to serial_tx_int_init on page 405).
RINT0	`c_int06()`	`isr_receive`	Receive interrupt of the DSP's serial interface, refer to serial_rx_int_init on page 404.
TINT0	`c_int09()`	`isr_t0`	Interrupt for the built-in timer0. It can be used to generate sampling clock interrupts and is initialized by timer0, refer to timer0, timer1 on page 348.
TINT1	`c_int10()`	`isr_t1`	Interrupt for the built-in timer1. It can be used to generate sampling clock interrupts and is initialized by the function timer1 (refer to timer0, timer1 on page 348).

Note

Interrupt service routines must use the naming conventions as written in the *TMS320 Floating-Point DSP Optimizing C Compiler User's Guide* from Texas Instruments. As an alternative you may use the alias names listed in the table above.

Related topics

References

edge_int_enable	380
int0_init, ... , int3_init	352
serial_rx_int_init	404
serial_tx_int_init	405
timer0, timer1	348

Example of Slave DSP Interrupts

Introduction

You can use the slave DSP interrupts INT0, … ,INT3 in two different ways.

Method 1

The most simple method is to poll the corresponding interrupt flag in the DSP's interrupt flag register (IF). In this case no interrupt service routine must be implemented and no initialization of the respective interrupt is required. Use the appropriate macro `int0_pending`, … ,`int3_pending` to poll the interrupt flag. After an interrupt is received, you have to clear the interrupt flags with the appropriate macro `int0_ack`, … ,`int3_ack`.

```
if (int1_pending(state)) /* test for pending interrupt INT1 */
{
   ...                         /* perform interrupt service */
   int1_ack(value);           /* acknowledge interrupt INT1 */
}
```

Method 2

The second method uses an interrupt service routine to serve an requested interrupt. In this case, the appropriate initialization function `int0_init`, … ,`int3_init` must be called in the initialization part of the application. You have to supply an interrupt service routine for the respective interrupt. The interrupt service routine must clear the interrupt flags with the appropriate macro `int0_ack`, … ,`int3_ack`. Under normal conditions, you should prefer this method because it spends no time polling the interrupt flag.

```
void isr_int1()          /* INT1 interrupt service routine */
{
   ...                         /* perform interrupt service */
   int1_ack(value);           /* acknowledge interrupt INT1 */
}
main()
{
   ...
   int1_init();        /* initialize INT1 interrupt service */
   ...
}
```

Related topics

References

int1_ack......356
int1_pending......358

int0_init, ... , int3_init

Syntax

```
void int0_init()
void int1_init()
void int2_init()
void int3_init()
```

Include file

`Int0.c`

Purpose

To initialize the interrupts INT0, INT1, INT2, or INT3.

Description

The function performs the following steps for the initialization:

1. Sets the related interrupt vector to point to the corresponding interrupt service routine `c_int01()`, `c_int02()`, `c_int03()`, or `c_int04()`. The alias name `isr_int0()`, `isr_int1()`, `isr_int2()`, or `isr_int3()` can be used instead of `c_int01()`, `c_int02()`, `c_int03()`, or `c_int04()` for the interrupt service routines.
2. Enables the related interrupt in the interrupt enable register (IE) of the VC33.
3. Enables interrupts globally in the status register (ST) of the VC33.

This function must be called before the corresponding interrupt can be used with an interrupt service routine.

> **Note**
>
> You have to provide the interrupt service routine yourself. Otherwise, the linker will detect an unresolved external reference. You have to clear the interrupt flags at the end of the interrupt service routine with the `int0_ack`, ... ,`int3_ack` macros.

Return value

None

Related topics

Examples

Example of Slave DSP Interrupts.......351

References

int0_ack.......355
int1_ack.......356
int2_ack.......356

int3_ack 357
Interrupts 349

enable_int0, ... ,enable_int3, enable_tint0, enable_tint1

Macro

```
void enable_int0(void)
void enable_int1(void)
void enable_int2(void)
void enable_int3(void)
void enable_tint0(void)
void enable_tint1(void)
```

Include file

`ds2211.h`

Purpose

To set interrupt enable bit for an individual interrupt INT0, INT1, INT2, INT3, TINT0, or TINT1 in the IE register of the TMS320VC33 to enable the corresponding interrupt.

Return value

None

Related topics

References

disable_int0, ... ,disable_int3, disable_tint0, disable_tint1 353
edge_int_enable 380
global_disable 355
global_enable 354

disable_int0, ... ,disable_int3, disable_tint0, disable_tint1

Macro

```
void disable_int0(void)
void disable_int1(void)
void disable_int2(void)
void disable_int3(void)
void disable_tint0(void)
void disable_tint1(void)
```

Include file

`ds2211.h`

Purpose

To clear the interrupt enable bit for an individual interrupt INT0, INT1, INT2, INT3, TINT0 or TINT1 in the IE register of the TMS320VC33 to disable the corresponding interrupt.

Return value

None

Related topics

References

enable_int0, ... ,enable_int3, enable_tint0, enable_tint1 353
global_disable 355
global_enable 354

global_enable

Macro

```
void global_enable()
```

Include file

`ds2211.h`

Purpose

To enable all the interrupts that have been individually enabled with `enable_int0`, `enable_int1`, `enable_tint0` or `enable_tint1`.

Description

The function sets the global interrupt enable bit (GIE) in the status register (ST) to enable interrupts globally.

Return value

None

Related topics

References

enable_int0, ... ,enable_int3, enable_tint0, enable_tint1 353
global_disable 355

global_disable

Macro	`void global_disable()`
Include file	`ds2211.h`
Purpose	To disable all the interrupts that have been enabled with `global_enable`.
Description	The function clears the global interrupt enable bit (GIE) in the status register (ST) to disable interrupts globally.
Return value	None
Related topics	References disable_int0, ... ,disable_int3, disable_tint0, disable_tint1 .... 353 enable_int0, ... ,enable_int3, enable_tint0, enable_tint1 .... 353 global_enable 354

int0_ack

Macro	`void int0_ack()`
Include file	`ds2211.h`
Purpose	To acknowledge an interrupt request for INT0 when the related interrupt service is finished.
Description	The macro clears the INT0 interrupt flag in the interrupt flag register (IF).
Return value	None

int1_ack

Macro

```
void int1_ack(long value)
```

Include file

`ds2211.h`

Purpose

To acknowledge an interrupt request for INT1 when the related interrupt service is finished.

Description

The macro clears the INT1 interrupt flag in the interrupt flag register (IF). The `value` parameter returns the contents of the DPMEM address reserved for INT1. The master processor writes a value to this address to request an INT1 interrupt.

> **Note**
>
> The DPMEM address 0x603FFF as seen by the slave DSP corresponds to the address 0x63FFF as seen by the master (offset = 0x3FFF).

Return value

None

int2_ack

Macro

```
void int2_ack(void)
```

Include file

`ds2211.h`

Purpose

To clear the INT2 interrupt flag in the IF register of the TMS320VC33.

Description

This macro can be used to acknowledge an interrupt request after an INT2 interrupt service has finished.

Return value

None

int3_ack

Macro	`void int3_ack(void)`
Include file	`ds2211.h`
Purpose	To clear the INT3 interrupt flag in the IF register of the TMS320VC33.
Description	This macro can be used to acknowledge an interrupt request after an INT3 interrupt service has finished.
Return value	None

int0_pending

Macro

```
void int0_pending(long state)
```

Include file

`ds2211.h`

Purpose

To read the state of the INT0 interrupt flag in the interrupt flag register (IF).

Description

Use this macro to serve an interrupt request with no interrupt service routine installed by simply polling the interrupt flag in the IF register (refer to Method 1 in Example of Slave DSP Interrupts on page 351).

Parameters

state Interrupt flag; the following values are defined:

Value	Meaning
0	Interrupt INT0 is inactive.
1	Interrupt INT0 is pending.

Return value

None

int1_pending

Macro

```
void int1_pending(long state)
```

Include file

`ds2211.h`

Purpose

To read the state of the INT1 flag in the interrupt flag register (IF).

Description

Use this macro to serve an interrupt request with no interrupt service routine installed by simply polling the interrupt flag in the IF register (refer to Method 1 in Example of Slave DSP Interrupts on page 351).

Parameters

state Interrupt flag; the following values are defined:

Value	Meaning
0	Interrupt INT1 is inactive.
1	Interrupt INT1 is pending.

Return value

None

Related topics

Examples

Example of Slave DSP Interrupts..351

int2_pending

Macro

```
void int2_pending(long state)
```

Include file

`ds2211.h`

Purpose

To return the state of the INT2 interrupt flag in the IF register of the VC33 to indicate whether an interrupt is pending.

Description

This macro can be used to serve an interrupt request with no interrupt service routine installed, by simply polling the interrupt flag in the VC33's IF register.

Parameters

state Interrupt flag; the following values are defined:

Value	Meaning
0	Interrupt INT2 is inactive.
1	Interrupt INT2 is pending.

Return value

None

int3_pending

Macro

```
void int3_pending(long state)
```

Include file

`ds2211.h`

Purpose

To return the state of the INT3 interrupt flag in the IF register of the VC33 to indicate whether an interrupt is pending.

Description

This macro can be used to serve an interrupt request with no interrupt service routine installed, by simply polling the interrupt flag in the VC33's IF register.

Parameters

state Interrupt flag; the following values are defined:

Value	Meaning
0	Interrupt INT3 is inactive.
1	Interrupt INT3 is pending.

Return value

None

Triggering Interrupts on the DS2211

Introduction

The TMS320VC33 slave DSP has access to DS2211 PHS-bus interrupts DRQ0...DRQ5. You can trigger the interrupts via macros.

Where to go from here

Information in this section

phs_irq0_trigger, ... ,phs_irq5_trigger

Macro

```
void irq0_trigger(void)
...
void irq5_trigger(void)
```

Include file

`ds2211.h`

Purpose

To trigger the corresponding interrupt on the DS2211 board.

Description

DS2211 interrupts IRQ0...IRQ5 are shared with capture and APU angle interrupts.

	Capture Interrupt	*Angle Interrupt*	*VC33 Interrupt*
IRQ0	✓	✓	✓
IRQ1	–	✓	✓
IRQ2	–	✓	✓
IRQ3	–	✓	✓
IRQ4	–	✓	✓
IRQ5	–	✓	✓
IRQ6: CAN interrupt			
IRQ7: UART interrupt			

Parameters

None

Return value

None

Related topics

References

phs_irq_trigger(long irq)..361

phs_irq_trigger(long irq)

Macro

```
void irq_trigger(long irq)
```

Include file

`ds2211.h`

Purpose

To trigger the specified interrupt on the DS2211 board.

Description

DS2211 interrupts IRQ0...IRQ5 are shared with capture and APU angle interrupts.

	Capture Interrupt	*Angle Interrupt*	*VC33 Interrupt*
IRQ0	✓	✓	✓
IRQ1	–	✓	✓
IRQ2	–	✓	✓
IRQ3	–	✓	✓
IRQ4	–	✓	✓
IRQ5	–	✓	✓
IRQ6: CAN interrupt			
IRQ7: UART interrupt			

Parameters

irq The number of the interrupt to be triggered.

Return value

None

Related topics

References

phs_irq0_trigger, ... ,phs_irq5_trigger...360

Error Handling

Where to go from here

Information in this section

Basics on Error Handling

Introduction

To indicate an error state you can use the error flag of the DSP. Writing different values to this location allows you to specify different error situations.

error_set

Macro

```
void error_set(long value)
```

Include file

`ds2211.h`

Purpose

To set the error flag (DPMEM address 0x603FFE).

Description

Writing different values to this flag allows you to specify different error situations.

Parameters

value	Error state

Return value

None

Related topics

References

ds2211_slave_dsp_error...305
error_read...364

error_read

Macro

```
long error_read()
```

Include file

`ds2211.h`

Purpose

To read the error flag (DPMEM address 0x603FFE).

Description

Depending on the contents of the error flag you can handle different error situations.

Return value

Contents of the error flag

Related topics

References

ds2211_slave_dsp_error...305
error_set...363

Status LEDs

Where to go from here

Information in this section

Status LEDs for the Slave DSP

Introduction

The DS2211 board provides two status LEDs for the slave DSP:

LED on	Meaning
XF0	The slave DSP is running an application. The LED is active after the init function and inactive after reset.
XF1	You can use this LED to indicate a special state of your application.

For the location of the LEDs on the board, refer to DS2211 Components (PHS Bus System Hardware Reference 🕮).

led_state

Macro

```
void led_state(long value)
```

Include file

`Util2211.h`

Purpose

To set the status LED XF1 to the specified state.

Parameters

value Enables or disables the LED. Use the following values:

Value	Meaning
0	LED is inactive.
1	LED is active.

Return value	None

D/A Converter (Slave DSP)

Where to go from here

Information in this section

Basics of the D/A Converter of the Slave DSP

Basics

The slave DSP provides eight 12-bit D/A converters. The value to be written to a D/A converter must be within the 32-bit signed integer range -2.14748365 · 10^9 … +2.14748365 · 10^9. DACs 1 … 4 are connected to a transformer, DACs 5 … 8 not. The signed integer range corresponds to different output:

DACs	Voltage Range	Description
1 … 4	±20 V	Connected to transformer, cannot transfer DC voltages
5 … 8	±10 V	Not connected to transformer, can transfer DC voltage

Note

After initialization, the transformer outputs are at 0 V and DC mode outputs are at high impedance. Use `ds2211_apu_transformer_mode_set` to enable the analog output ports.

Scale floating-point values

You have to scale floating-point values to the given range and convert them to the data type long integer before you can write the value to the D/A output. For example:

```
float y;
y = ...
*dac1 = (long) (SCAL * y);
```

The scaling factor SCAL is defined in the `Ds2211.h` header file, refer to Identifiers for Numerical Constants on page 345.

Using the D/A converters

You can use the D/A converters in the following two ways:

Write to predefined addresses To access a D/A converter you only have to write a scaled value to one of the predefined addresses *dac1 … *dac8, refer to Pointer Declarations and Global Variables on page 346.

Use predefined macros To scale floating-point values to the full D/A converter range and write them to the D/A channels you can use the predefined macros `dac_out`, `dac_out1`, …, `dac_out8`. `dac_out` allows you to pass the channel number as a parameter, whereas `dac_out1` … `dac_out8` have shorter execution times.

For further information on the DAC and its I/O mapping, refer to DAC Unit (DS2211 Features 📖).

I/O mapping

The analog output signals are available at the following DS2211's connector pins (P2 / P3).

Connector	Pin	Signal	Macro
P2	63	VC33-Waveform 0+	dac_out1
P2	65	VC33-Waveform 0–	
P2	67	VC33-Waveform 1+	dac_out2
P2	69	VC33-Waveform 1–	
P2	71	VC33-Waveform 2+	dac_out3
P2	73	VC33-Waveform 2–	
P2	75	VC33-Waveform 3+	dac_out4
P2	77	VC33-Waveform 3–	
P3	30	DSP_DAC4	dac_out5
P3	14	DSP_DAC5	dac_out6
P3	47	DSP_DAC6	dac_out7
P3	31	DSP_DAC7	dac_out8

Related topics

References

ds2211_apu_transformer_mode_set..39

dac_out

Macro

```
void dac_out(long channel, float value)
```

Include file

`ds2211.h`

Purpose

To write the value to the given D/A channel.

Description

The output parameter value is scaled by the factor $2.147483648 \cdot 10^9$ and converted to the data type long integer before it is written to the specified D/A channel.

> **Note**
>
> After initialization, the transformer outputs are at 0 V and DC mode outputs are at high impedance. Use `ds2211_apu_transformer_mode_set` to enable the analog output ports.

Parameters

channel D/A channel in the range 1 … 8

value Output value in the range –1.0 … +1.0, which corresponds to an output voltage of –20.0 V … +20.0 V (DAC1 … DAC4) or –10.0 V … +10.0 V (DAC5 … DAC8)

Return value

None

Related topics

Basics

Basics of the D/A Converter of the Slave DSP..367

References

dac_out1, ..., dac_out8..369
ds2211_apu_transformer_mode_set..39

dac_out1, ..., dac_out8

Macro

```
void dac_out1(float value)
void dac_out2(float value)
...
void dac_out8(float value)
```

Include file

`ds2211.h`

Purpose

To write the value to the specified D/A channel.

Description

The output parameter value is scaled by the factor 2.147483648 · 10^9 and converted to the long integer data type before it is written to the related D/A channel.

> **Note**
>
> After initialization, the transformer outputs are at 0 V and DC mode outputs are at high impedance. Use `ds2211_apu_transformer_mode_set` to enable the analog output ports.

Parameters

value Output value in the range –1.0 … +1.0, which corresponds to an output voltage of –20.0 V … +20.0 V (DAC1 … DAC4) or –10.0 V … +10.0 V (DAC5 … DAC8)

Return value

None

Related topics

Basics

Basics of the D/A Converter of the Slave DSP.. 367

References

dac_out.. 368
ds2211_apu_transformer_mode_set.. 39

Digital I/O via Serial Port

Where to go from here

Information in this section

Basics of Digital I/O via Serial Port

Introduction

The slave DSP supports up to 6 digital I/O lines via the serial interface pins. The digital I/O pins are shared with the serial interface pins. Each line can be configured as input or output. For the location of the serial interface connector P6, refer to DS2211 Components (PHS Bus System Hardware Reference 🕮).

I/O mapping

The macros are related to the digital output lines as listed in the following table:

Macro	Slave DSP Serial Interface Pin	P6 Connector Pin
`init_dig_out1`, `dig_out1`, `dig_in1`	DX0	6
`init_dig_out2`, `dig_out2`, `dig_in2`	FSX0	10
`init_dig_out3`, `dig_out3`, `dig_in3`	CLKR0	4
`init_dig_out4`, `dig_out4`, `dig_in4`	DR0	8
`init_dig_out5`, `dig_out5`, `dig_in5`	FSR0	12
`init_dig_out6`, `dig_out6`, `dig_in6`	CLKX0	2

Note

If the serial interface is initialized and used for data transmission, the digital I/O access macros must not be used. Otherwise the serial transmission will fail (refer to Serial Interface on page 396).

init_dig_out1, ..., init_dig_out6

Macro

```
void init_dig_out1(long value)
...
void init_dig_out6(long value)
```

Include file

`ds2211.h`

Purpose

To initialize the slave DSP's pins for input or output.

Parameters

value Initializes the I/O line. Use the following values:

Value	Meaning
0	Initializes the I/O line for input.
1	Initializes the I/O line for output.

Return value

None

Related topics

Basics

Basics of Digital I/O via Serial Port.......371

References

dig_in1, ..., dig_in6.......373
dig_out1, ..., dig_out6.......372

dig_out1, ..., dig_out6

Macro

```
void dig_out1(long value)
...
void dig_out6(long value)
```

Include file

`ds2211.h`

Purpose

To set the respective digital I/O line to 0 or 1.

Description

You have to initialize the pin for output first with the appropriate `init_dig_out<n>` macro (<*n*> = 1 … 6).

Parameters

value Specifies the state of the digital output line. The valid values are 0 for low level or 1 for high level.

Return value

None

Related topics

Basics

Basics of Digital I/O via Serial Port........371

References

dig_in1, ..., dig_in6........373
init_dig_out1, ..., init_dig_out6........372

dig_in1, ..., dig_in6

Macro

```
long dig_in1()
...
long dig_in6()
```

Include file

`ds2211.h`

Purpose

To read the state of the respective digital I/O line.

Description

You have to initialize the I/O line for input first with the appropriate `init_dig_out<n>` macro (<*n*> = 1 … 6).

Return value

State of the digital input line. The valid values are 0 for low level or 1 for high level.

Related topics

Basics

Basics of Digital I/O via Serial Port.. 371

References

dig_out1, ..., dig_out6.. 372
init_dig_out1, ..., init_dig_out6... 372

Digital I/O via DS2211 I/O Unit

Where to go from here

Information in this section

Basics of Digital I/O via DS2211 I/O Unit

Introduction

The slave DSP has access to the 16 digital I/O input lines and to the 16 digital I/O output lines of the DS2211 board.

Input lines

The input lines can be read from the slave DSP and the master processor board at the same time.

`dig_io_in1` ... `dig_io_in16` macros can be used to input data from the respective serial port I/O lines. All 16 input lines can be read out simultaneously with the `dig_io_in` macro.

Output lines

The state of the digital output lines is the result of a logical OR operation between the output values set by the slave DSP and the output values set by the master processor board.

`dig_io_out1` … `dig_out16` macros can be used to output data from the respective serial port I/O lines.

Digital I/O lines

The macros are related to the digital I/O lines as listed in the following table:

Macro	Connector	PIN	Signal
`dig_io_in1()`	P1	3	DIG_IN1
`dig_io_in2()`	P1	4	DIG_IN2

Macro	Connector	PIN	Signal
`dig_io_in3()`	P1	5	DIG_IN3
`dig_io_in4()`	P1	6	DIG_IN4
`dig_io_in5()`	P1	7	DIG_IN5
`dig_io_in6()`	P1	8	DIG_IN6
`dig_io_in7()`	P1	9	DIG_IN7
`dig_io_in8()`	P1	10	DIG_IN8
`dig_io_in9()`	P1	11	DIG_IN9
`dig_io_in10()`	P1	12	DIG_IN10
`dig_io_in11()`	P1	13	DIG_IN11
`dig_io_in12()`	P1	14	DIG_IN12
`dig_io_in13()`	P1	15	DIG_IN13
`dig_io_in14()`	P1	16	DIG_IN14
`dig_io_in15()`	P1	17	DIG_IN15
`dig_io_in16()`	P1	18	DIG_IN16

Macro	Connector	PIN	Signal
`dig_io_out1()`	P2	32	DIG_OUT1
`dig_io_out2()`	P2	34	DIG_OUT2
`dig_io_out3()`	P2	36	DIG_OUT3
`dig_io_out4()`	P2	38	DIG_OUT4
`dig_io_out5()`	P2	40	DIG_OUT5
`dig_io_out6()`	P2	42	DIG_OUT6
`dig_io_out7()`	P2	44	DIG_OUT7
`dig_io_out8()`	P2	46	DIG_OUT8
`dig_io_out9()`	P2	50	DIG_OUT9
`dig_io_out10()`	P2	52	DIG_OUT10
`dig_io_out11()`	P2	54	DIG_OUT11
`dig_io_out12()`	P2	56	DIG_OUT12
`dig_io_out13()`	P2	58	DIG_OUT13
`dig_io_out14()`	P2	60	DIG_OUT14
`dig_io_out15()`	P2	62	DIG_OUT15
`dig_io_out16()`	P2	64	DIG_OUT16

dig_io_out1, ... , dig_io_out16

Macro

```
void dig_io_out1(long value)

void dig_io_out2(long value)

...

void dig_io_out16(long value)
```

Include file

`ds2211.h`

Purpose

To set the corresponding digital I/O line to 0 (low level) or 1 (high level) depending on the parameter value.

Parameters

value The parameter value specifies the state of the digital output line. The value must be set to 0 (low level) or 1 (high level).

The state of the digital output lines is the result of a logical OR operation between the output values set by the slave DSP and the output values set by the master processor board.

Return value

None

dig_io_in

Macro

```
long dig_io_in(void)
```

Include file

`ds2211.h`

Purpose

To return the state of all 16 digital I/O input lines.

Return value

The state of the 16 digital I/O input lines (0x00000000 ... 0x0000FFFF).

Bit 0 represents the state of I/O channel 1 and bit 15 the state of I/O channel 16.

Related topics

References

dig_io_in1, ... , dig_io_in16 ... 378

dig_io_in1, ... , dig_io_in16

Macro

```
long dig_io_in1(void)
…
long dig_io_in16(void)
```

Include file

`ds2211.h`

Purpose

To return the state of the specified digital I/O input line.

Return value

State of the digital I/O input line (0 or 1).

Related topics

References

dig_io_in ... 377

Capture Input Access Functions

Where to go from here

Information in this section

Basics of Capture Input Access Functions

Basics

You can access the outputs of the 16 DS2211 complex comparators via the slave DSP. The settings of the capture windows and the capture modes take effect in the same way as on the DS2211 master I/O.

Note

The settings of the capture windows and capture modes also affect the outputs on the slave.

The state of the comparator output lines can be observed using the `inj_ign_state` macro.

An edge on one or more comparator output line may request the INT3 interrupt on the slave DSP. Using the `edge_int_enable` macro you can specify the comparator output(s) and the edge direction(s) to request the interrupt with.

After receiving an INT3 interrupt, you can examine the comparator output line(s) which requested it by using the `int3_source` macro.

> **Note**
>
> The INT3 interrupt depends on the APU and the capture window. It is only generated, when the APU is running and the triggering pulse is within the current capture window.

inj_ign_state

Macro

```
long inj_ign_state(void)
```

Include file

`ds2211.h`

Purpose

To return the state of the 16 complex comparator output lines.

Description

You can use the defines IGNCAP … IGNCAP6, AUXCAP1, AUXCAP2 and INJCAP1 … INJCAP8 to test if the corresponding bit are set.

For information on the complex comparators, refer to Complex Comparators (DS2211 Features 📖).

Return value

State of the complex comparator outputs (0x00000000 ... 0x0000FFFF)

Related topics

References

dig_out1, ..., dig_out6..372
init_dig_out1, ..., init_dig_out6..372

edge_int_enable

Macro

```
void edge_int_enable(long mask)
```

Include file

`ds2211.h`

Purpose

To enable the INT3 request on leading or trailing edges of the complex comparator outputs.

Parameters

mask Lest you specify a mask with channels and edges specified to generate INT3 interrupts.

Use the following defines for trailing edges:

- `IGNCAP1_TRAIL` … `IGNCAP6_TRAIL`
- `AUXCAP1_TRAIL`, `AUXCAP2_TRAIL`
- `INJCAP1_TRAIL` … `INJCAP8_TRAIL`

Use the following defines for leading edges:

- `IGNCAP1_LEAD` … `IGNCAP6_LEAD`
- `AUXCAP1_LEAD`, `AUXCAP2_LEAD`
- `INJCAP1_LEAD` … `INJCAP8_LEAD`

> **Note**
>
> All the defines can be combined using the logical OR operation.

Return value

None

Related topics

References

dig_out1, ..., dig_out6 372
init_dig_out1, ..., init_dig_out6 372

int3_source

Macro

```
long int3_source(void)
```

Include file

`ds2211.h`

Purpose

To return a bit mask that indicates which channel(s) has triggered the INT3 interrupt.

Description

When a channel triggers an interrupt, the corresponding bit is set in the INT3 source register. The macro reads the INT3 source register, which is cleared after the read access. The `int3_source` macro can therefore be invoked once in an INT3 interrupt service function.

You can use the predefined defines `IGNCAP` … `IGNCAP6`, `AUXCAP1`, `AUXCAP2` and `INJCAP1` … `INJCAP8` to test the corresponding bits.

> **Note**
>
> You can only use either the `int3_source` macro or the `int3_source_edge` macro once in an INT3 service function, because each call clears the information on which channel(s) or edge(s) caused the INT3 interrupt.
> It might therefore be necessary to store the return value in a variable.

Return value

A bit mask indicating which channel(s) has triggered the interrupt in the range 0x00000000 … 0x0000FFFF.

Related topics

References

dig_out1, ..., dig_out6..372
init_dig_out1, ..., init_dig_out6..372

int3_source_edge

Macro

```
long int3_source_edge(void)
```

Include file

`ds2211.h`

Purpose

To return a bit mask that indicates which channel(s) and which edge(s) has triggered the INT3 interrupt.

Description

When a channel triggers an interrupt, the bit indicating the channel-edge combination is set in the INT3 source register. The macro reads the INT3 source register, which is cleared after the read access. The `int3_source_edge` macro can therefore be invoked once in an INT3 interrupt service function.

You can use the following predefined symbols to test the corresponding bits.

For trailing edges:

- `IGNCAP1_TRAIL` ... `IGNCAP6_TRAIL`
- `AUXCAP1_TRAIL`, `AUXCAP2_TRAIL`
- `INJCAP1_TRAIL` ... `INJCAP8_TRAIL`

For leading edges:

- `IGNCAP1_LEAD` ... `IGNCAP6_LEAD`
- `AUXCAP1_LEAD`, `AUXCAP2_LEAD`
- `INJCAP1_LEAD` ... `INJCAP8_LEAD`

Note

You can only use either the `int3_source` macro or the `int3_source_edge` macro once in an INT3 service function, because each call clears the information on which channel(s) or edge(s) caused the INT3 interrupt.
It might therefore be necessary to store the return value in a variable.

Return value

A bit mask indicating which channel(s) and edge(s) has triggered the interrupt in the range 0x00000000 ... 0xFFFFFFFF.

Related topics

References

dig_out1, ..., dig_out6 372
init_dig_out1, ..., init_dig_out6 372

DPMEM Access Functions

Where to go from here

Information in this section

Basics of Accessing the DPMEM

Introduction

If access to the DPMEM is not arbitrated by hardware, you can avoid conflicts when the DPMEM is accessed from both sides, by the TMS320VC33 DSP and the master processor, by using one of the sixteen (1 … 16) semaphores provided by the DS2211.

Note

The semaphores do not physically prevent improper access to the DPMEM.

Hardware arbitration

DS2211 boards with board revision 3 and FPGA revision 3 or higher have a 32-bit hardware arbitration that avoids conflicts when the DPM is accessed. It is not necessary to use semaphores. However, using a semaphore is helpfull if you transfer several values that must be of integrity.

> **Tip**
>
> The revision number is displayed on the **Properties** pane in ControlDesk when you select the board in the **Platforms/Devices** controlbar.

Using semaphores

The semaphore is requested by writing a 0 to it. If you read the semaphore afterwards and get the value 0 the semaphore has been accessed successfully. If the value is unequal to 0, the semaphore is obtained by the other side. The semaphore must be released by writing a 1 to it. If you do not release the semaphore, it will remain blocked.

Example of DPMEM Access Functions

Example 1

This example performs the following operations:

- Requests semaphore 1.
- If the request fails, the semaphore request is released.
- If the request was successful, the DPMEM is accessed.
- The semaphore is released afterwards.

```
semaphore1_request(state);
if(state) /* semaphore request failed */
   semaphore1_release();
else
{
   /* accessing the DPMEM */
   ...
   /* release the semaphore */
   semaphore1_release();
}
```

Example 2

This example performs the following operations:

- Requests semaphore 1 until the request is successful.
- If the request was successful, the DPMEM is accessed.
- The semaphore is released afterwards.

```
do
{
   semaphore1_request(state);
}while(state);
   /* accessing the DPMEM */
   ...
   /* release the semaphore */
   semaphore1_release();
}
```

semaphore_request

Macro

```
void semaphore_request(long nr, long state)
```

Include file

`ds2211.h`

Purpose

To request access to the DPMEM by writing a 0 to the specified semaphore.

Description

The `state` parameter returns 0 if the request was successful.

> **Note**
>
> If the parameter value does not return 0, the semaphore is used by the opposite port. You have to repeat the request or release the semaphore by calling the `semaphore_release` macro.

Parameters

nr Number of the semaphore to be requested in the range 1 ... 16

state State of the request. The following values are possible:

Value	Meaning
0	The semaphore has been requested successfully.
1	The request has failed.

Return value

None

Related topics

References

semaphore_release....................386
semaphore1_release, ... semaphore16_release....................388
semaphore1_request, ... semaphore16_request....................387

semaphore_release

Macro

```
void semaphore_release(long nr)
```

Include file

`ds2211.h`

Purpose

To release the specified semaphore by writing a 1 to it.

Parameters

nr Number of the semaphore to be requested in the range 1 … 16

Return value

None

Related topics

References

semaphore_request..386
semaphore1_release, ... semaphore16_release..388
semaphore1_request, ... semaphore16_request..387

semaphore1_request, ... semaphore16_request

Macro

```
void semaphore1_request(long state)
…
void semaphore16_request(long state)
```

Include file

`ds2211.h`

Purpose

To request access to the DPMEM by writing a 0 to the related semaphore.

Description

The `state` parameter returns 0 if the request was successful.

If the `state` parameter does not return 0, the semaphore is used by the opposite port. You must repeat the request or release the semaphore request by calling the `semaphore<n>_release` macro (<*n*> = 1 … 16).

Parameters

state State of the request. The following values are possible:

Value	Meaning
0	The semaphore has been requested successfully.
1	The request has failed.

Return value

None

Related topics

References

semaphore_release.......386
semaphore_request.......386
semaphore1_release, ... semaphore16_release.......388

semaphore1_release, ... semaphore16_release

Macro

```
void semaphore1_release(void)
…
void semaphore16_release(void)
```

Include file

`ds2211.h`

Purpose

To release the related semaphore by writing 1 to it.

Return value

None

Related topics

References

semaphore_release.......386
semaphore_request.......386
semaphore1_request, ... semaphore16_request.......387

Direct Memory Access

Where to go from here

Information in this section

Basics of Direct Memory Access

Introduction

The slave DSP comprises a direct memory access (DMA) controller supporting one DMA channel. The DMA controller transfers blocks of data to any location in the memory without interfering with CPU operation. Therefore, it is possible to interface the DSP to slow external memories and peripherals (A/D converters, serial interfaces, for example) without reducing the computational throughput of the CPU. The result is improved system performance and decreased system cost.

Further information

For more information on the DMA controller, refer to the Texas Instruments web site at "http://www.ti.com" and search for the *TMS320C3x User's Guide* (literature number SPRU031F) and *TMS320VC33 Digital Signal Processor* (literature number SPRS087E).

dma_init

Syntax

```
void dma_init(
      unsigned long src_addr,
      unsigned long dst_addr,
      unsigned long count,
      unsigned int src_mode,
      unsigned int dst_mode,
      unsigned int int_sync,
      unsigned int tc,
      unsigned int tcint)
```

Include file

`Dma31.h`

Purpose

To initialize and start the DMA controller of the slave DSP.

Description

Use `dma_stop` or `dma_stop_when_finished` to stop the DMA controller.

Parameters

src_addr Address of the source data to be transferred by the DMA controller

dst_addr Destination address to which the data will be transferred

count Number of words to be transferred in the range 1 … 16,777,215

src_mode Mode of source address modification. The following symbols are predefined:

Predefined Symbol	Meaning
DMA_NO_MODIFY	The source address is not modified.
DMA_INCREMENT	The source address is incremented after each DMA read access.
DMA_DECREMENT	The source address is decremented after each DMA read access.

dst_mode Mode of destination address modification. The following symbols are predefined:

Predefined Symbol	Meaning
DMA_NO_MODIFY	The destination address is not modified.
DMA_INCREMENT	The destination address is incremented after each DMA write access.
DMA_DECREMENT	The destination address is decremented after each DMA write access.

int_sync DMA synchronization mode. The following symbols are predefined:

Predefined Symbol	Meaning
`DMA_NO_SYNC`	No synchronization
`DMA_SRC_SYNC`	Source synchronization. This means that a read access is performed when a DMA interrupt occurs.
`DMA_DST_SYNC`	Destination synchronization. This means that a write access is performed when a DMA interrupt occurs.

tc DMA transfer mode. The following symbols are predefined:

Predefined Symbol	Meaning
`DMA_CONTINUOUS`	Transfer restarts when the specified number of words has been transferred.
`DMA_TERMINATE`	Transfer is terminated when the specified number of words has been transferred.

tcint Sets the mode for the DMA to CPU interrupt. The following symbols are predefined:

Predefined Symbol	Meaning
`DMA_TCINT_DISABLE`	No DMA interrupt is generated when the transfer has finished.
`DMA_TCINT_ENABLE`	A DMA interrupt is generated when the transfer has finished.

Return value

None

Related topics

References

dma_interrupt_disable........394
dma_interrupt_enable........393
dma_stop........391
dma_stop_when_finished........392

dma_stop

Macro

```
void dma_stop
```

Include file

`Dma31.h`

Purpose

To stop the DMA controller.

Description

The current word read or write operation is completed.

Return value

None

Related topics

References

dma_restart 392
dma_stop_when_finished 392

dma_stop_when_finished

Macro

```
void dma_stop_when_finished
```

Include file

`Dma31.h`

Purpose

To stop the DMA controller when the entire transfer has been completed.

Return value

None

Related topics

References

dma_restart 392
dma_stop 391

dma_restart

Macro

```
void dma_restart
```

Include file

`Dma31.h`

Purpose

To restart the DMA controller from reset or a previous state.

Return value

None

Related topics

References

dma_stop 391
dma_stop_when_finished 392

dma_reset

Macro

```
void dma_reset
```

Include file

`Dma31.h`

Purpose

To reset the DMA controller.

Return value

None

Related topics

References

dma_init 390

dma_interrupt_enable

Syntax

```
void dma_interrupt_enable(unsigned long mask)
```

Include file

`Dma31.h`

Purpose

To enable the external DMA interrupts.

Description

The interrupt sources of the slave DSP are connected to the CPU and to the DMA controller. To enable a DMA interrupt, the respective interrupt enable flag is set in the IE register of the slave DSP.

Parameters

mask Interrupt(s) to be enabled. The following symbols are predefined:

Predefined Symbol	Meaning
`DMA_EINT0`	External interrupt INT0
`DMA_EINT1`	External interrupt INT1
`DMA_EINT2`	External interrupt INT2
`DMA_EINT3`	External interrupt INT3
`DMA_EXINT0`	Serial interface transmit interrupt
`DMA_ERINT0`	Serial interface receive interrupt
`DMA_ETINT0`	Timer0 interrupt
`DMA_ETINT1`	Timer1 interrupt
`DMA_EDINT`	DMA controller interrupt

Return value

None

Related topics

References

dma_interrupt_disable..........394

dma_interrupt_disable

Syntax

```
void dma_interrupt_disable(unsigned long mask)
```

Include file

`Dma31.h`

Purpose

To disable the external DMA interrupts.

Description

The interrupt sources of the slave DSP are connected to the CPU and to the DMA controller. To disable a DMA interrupt, the respective interrupt enable flag is cleared in the IE register of the slave DSP.

Parameters

mask Interrupt(s) to be disabled. The following symbols are predefined:

Predefined Symbol	Meaning
`DMA_EINT0`	External interrupt INT0
`DMA_EINT1`	External interrupt INT1
`DMA_EINT2`	External interrupt INT2
`DMA_EINT3`	External interrupt INT3
`DMA_EXINT0`	Serial interface transmit interrupt
`DMA_ERINT0`	Serial interface receive interrupt
`DMA_ETINT0`	Timer0 interrupt
`DMA_ETINT1`	Timer1 interrupt
`DMA_EDINT`	DMA controller interrupt

Return value

None

Related topics

References

dma_interrupt_enable..393

Serial Interface

Where to go from here

Information in this section

Basics of Using the Slave DSP's Serial Interface

Description

The bi-directional serial interface of the slave DSP allows you to connect a DS2302 board, a DS2210 board or another DS2211 board. For the location of the serial interface connector P6, refer to DS2211 Components (PHS Bus System Hardware Reference 🕮).

The connection provides a frequency of up to 10 MHz. For the serial transmission the handshake mode is used.

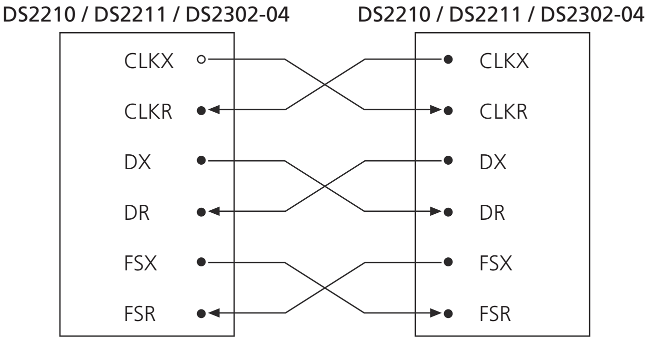
Note

If the serial interface is initialized and used for data transmission, the digital I/O access macros (refer to Digital I/O via Serial Port on page 371) must not be used. Otherwise the serial transmission will fail.

For more information on the serial interface, refer to the Texas Instruments web site at "http://www.ti.com" and search for the *TMS320C3x User's Guide* (literature number SPRU031F) and *TMS320VC33 Digital Signal Processor* (literature number SPRS087E).

Connection schemes

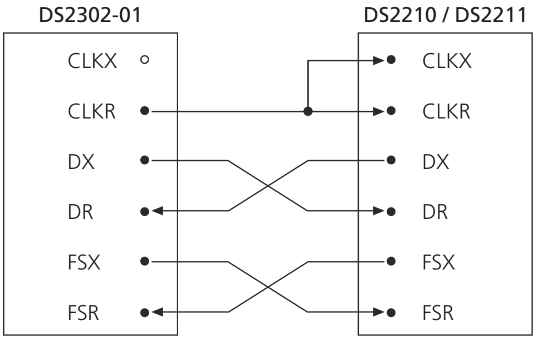
The following illustration shows the scheme for DS2210 / DS2302-04 / DS2211 connections in handshake mode.



The following illustration shows the scheme for the old DS2302-01 / DS2211 connection scheme connections in handshake mode.

Note

On the DS2302-01, the serial interface pin CLKX0 is not available for serial transmission.



Example of Using the Serial Interface of the Slave DSP

Introduction

The examples show serial communication in polling mode and interrupt-driven mode.

Example (polling mode)

The following example shows serial communication in polling mode. The serial interface is initialized for the standard handshake mode. Transmission is performed at a frequency of 10.0 MHz for a connection to a DS2210 board running at 80 MHz.

The `serial_rx_word_poll` receive function is invoked until one data word is received. After that, the `serial_tx_word_poll` function is invoked until the data word is transmitted successfully.

```
void main(void)
{
   long data;
   /* initialize hardware system */
   init();
   /* initialize the serial interface */
   serial_init_std_handshake(1);
   /* receive data */
   while(serial_rx_word_poll((long *)&data) != SP_TRUE );
   /* transmit data */
   while(serial_tx_word_poll((long *)&data) != SP_TRUE );
}
```

Example (interrupt-driven mode)

The following example shows serial transmission in interrupt-driven mode. The serial interface is initialized for the standard handshake mode. Transmission is performed at a frequency of 10.0 MHz for a connection to a DS2210 board running at 80 MHz.

The transmit and the receive interrupts are initialized. The received data is stored in the data array `receive_data[]` by the `serial_rx_word_int` function. The data to be transmitted is available in the `transmit_data[]` array. To start interrupt-driven transmission, the **serial_tx_int_start** macro must be invoked. Each time the serial interface has sent a data word and is ready to send the next data word, a new transmit interrupt is requested. After the 100 data values are sent with the **serial_tx_word_int** function, transmission stops. No data is sent in the last execution of the transmit interrupt service routine due to x ≤ 100. To restart sending, the index x must be set to 0 and the **serial_tx_int_start** macro must be called again.

```
long transmit_data[100];
long receive_data[100];
void isr_receive() /* receive interrupt service routine */
{
   if(i >= 100)
      i = 0;
   serial_rx_word_int((long *)&receive_data[i++]);
}
void isr_transmit() /* transmit interrupt service routine */
{
   if(x < 100)
      serial_tx_word_int((long *)&transmit_data[x++]);
}
void main(void)
{
   /* initialize hardware system */
   init();
   /* initialize the serial interface */
   serial_init_std_handshake(1);
   /* initialize receive interrupt */
   serial_rx_int_init();
   /* initialize transmit interrupt */
   serial_tx_int_init();
   /* start transmission */
   serial_tx_int_start();
...
}
```

Related topics

References

init	347
serial_init_std_handshake	400
serial_rx_int_init	404
serial_rx_word_int	410
serial_rx_word_poll	410
serial_tx_int_init	405
serial_tx_int_start	406
serial_tx_word_int	409
serial_tx_word_poll	408

serial_init_std_handshake

Macro

```
void serial_init_std_handshake(unsigned long timer_prd)
```

Include file

`Ser31.h`

Purpose

To initialize the slave DSP's serial interface for data transfer in handshake mode for the following connections:

- DS2211 – DS2210
- DS2211 – DS2211
- DS2302-04 – DS2210
- DS2302-04 – DS2211
- DS2302-04 – DS2302-04

Description

This macro automatically calls **`serial_init`** with the required parameters. For a connection to a DS2302-01, refer to serial_init_ds2211 on page 401.

> **Note**
>
> The receiving frequency via the DR0 input depends on the setting of the opposite transmitting serial interface.

Parameters

timer_prd Frequency of the serial transmission via the DX0 output. The valid values are 0x0001 … 0xFFFF. The transmission frequency is calculated as follows:

$$f_{trans} = \frac{osc_{clk}}{8 \cdot timer\_prd}$$

Where

osc_{clk} is the oscillator clock frequency of the DSP.

To ensure successful operation the transmission frequency must not exceed the value calculated below:

$$f_{trans} \leq \frac{osc_{clk,min}}{5.2}$$

Where

$osc_{clk,min}$ is the slowest oscillator clock frequency of both involved boards.

The following table shows the oscillator clock frequencies of the boards:

Board	Oscillator Clock Frequency [MHz]
DS2210	80
DS2211	150

Board	Oscillator Clock Frequency [MHz]
DS2302-01	60
DS2302-04	150

Return value

None

Related topics

References

serial_init.. 402

serial_init_ds2211

Macro

```
void serial_init_ds2211(void)
```

Note

Only valid for DS2302-01 connections.

Include file

`Ser31.h`

Purpose

To initialize the slave DSP's serial interface for data transfer in handshake mode for DS2211 – DS2302-01 connections.

Description

This function calls `serial_init` automatically with the required parameters.

Note

This macro must only be used with a DS2211 board with its serial port being connected to a DS2302-01 board. For a connection to a DS2210 or DS2211 board, refer to `serial_init_std_handshake` on page 400.

Return value

None

Related topics

References

serial_init 402
serial_init_std_handshake 400

serial_init

Syntax

```
serial_init(
      unsigned long g_ctrl,
      unsigned long timer_prd,
      unsigned long tx_ctrl,
      unsigned long rx_ctrl,
      unsigned long timer_ctrl);
```

Include file

`Ser31.h`

Purpose

To initialize the serial interface of the slave DSP as follows:

- Clear the global control register and the timer control register to reset the serial interface.
- Initialize the serial interface registers with the specified values.
- Start the required serial interface timers according to the control register settings.
- Enable receive and transmit access.

For more information on the serial interface, refer to the Texas Instruments web site at http://www.ti.com and search for the *TMS320C3x User’s Guide* (literature number SPRU031F) and *TMS320VC33 Digital Signal Processor* (literature number SPRS087E).

> **Note**
>
> `serial_init` is called automatically by the board-related initialization macros `serial_init_std_handshake` and `serial_init_ds2211`.

Parameters

g_ctrl Setting of the global control register

timer_prd Frequency of the serial transmission via the DX0 output. The valid values are 0x0001 … 0xFFFF. The transmission frequency is calculated as follows:

$$f_{trans} = \frac{osc_{clk}}{8 \cdot timer\_prd}$$

Where

osc_{clk} is the oscillator clock frequency of the DSP.

To ensure successful operation the transmission frequency must not exceed the value calculated below:

$$f_{trans} \leq \frac{osc_{clk,min}}{5.2}$$

Where

$osc_{clk,min}$ is the slowest oscillator clock frequency of both involved boards.

The following table shows the oscillator clock frequencies of the boards:

Board	Oscillator Clock Frequency [MHz]
DS2210	80
DS2211	150
DS2302-01	60
DS2302-04	150

tx_ctrl Setting of the transmit control register

rx_ctrl Setting of the receive control register

timer_ctrl Setting of the timer control register

Return value None

Related topics References

serial_init_ds2211 401
serial_init_std_handshake 400

serial_disable

Macro

```
void serial_disable
```

Include file `Ser31.h`

Purpose To disable and reset the serial interface.

Description

All serial interface pins are configured as digital I/O pins and set as inputs (refer to Digital I/O via Serial Port on page 371).

Return value

None

Related topics

References

Digital I/O via Serial Port..371

serial_rx_int_init

Syntax

```
void serial_rx_int_init()
```

Include file

`Ser31ir.h`

Purpose

To initialize the receive interrupt of the serial interface as follows:

- Set the corresponding interrupt vector RINT0 to point to the receive interrupt routine c_int06.
- Enable the receive interrupt and interrupts globally.

Description

The receive interrupt service routine usually contains the receive function. You must program the routine yourself.

> **Tip**
>
> You can use the alias name `isr_receive` instead of `c_int06` for the receive interrupt service routine.

For an example, refer to example 2 in Example of Using the Serial Interface of the Slave DSP on page 398.

Return value

None

Related topics

Examples

Example of Using the Serial Interface of the Slave DSP..398

References

serial_tx_int_init..405
serial_tx_int_start..406

serial_tx_int_init

Syntax

```
void serial_tx_int_init()
```

Include file

`Ser31ix.h`

Purpose

To initialize the transmit interrupt of the serial interface as follows:

- Set the interrupt vector XINT0 to point to the receive interrupt routine c_int05.
- Enable the receive interrupt and interrupts globally.

Description

The transmit interrupt service routine usually contains the transmit function. You must program the routine yourself.

> **Tip**
>
> You can use the alias name `isr_transmit` instead of `c_int05` for the transmit interrupt service routine.

For an example, refer to example 2 in Example of Using the Serial Interface of the Slave DSP on page 398.

After initialization, you have to start the interrupt-driven transmission with `serial_tx_int_start` on page 406. This macro requests the first transmit interrupt by setting the respective flag in the DSP's IF register.

Return value

None

Related topics

Examples

Example of Using the Serial Interface of the Slave DSP.. 398

References

serial_rx_int_init.. 404
serial_tx_int_start.. 406

serial_tx_int_start

Macro

```
void serial_tx_int_start()
```

Include file

`Ser31.h`

Purpose

To request the first transmit interrupt by setting the respective interrupt flag in the DSP's IF register. Call this macro after the initialization of the transmit interrupt to start interrupt-driven transmission.

Description

The transmit interrupt is requested by the serial interface when the port is ready to transmit a new word after a preceding transmission.

Note

Use this macro to restart transmission each time it stops.

Return value

None

Related topics

References

serial_rx_int_init.. 404
serial_tx_int_init... 405

disable_rx_int, disable_tx_int

Macro

```
void disable_rx_int()
void disable_tx_int()
```

Include file

`Ser31.h`

Purpose

To disable the serial receive or transmit interrupt (RINT0 or XINT0). The enable bit for the interrupt RINT0 or XINT0 is cleared in the DSP's IE register to disable the corresponding interrupt.

Return value

None

Related topics

References

enable_rx_int, enable_tx_int..407

enable_rx_int, enable_tx_int

Macro

```
void enable_rx_int()
void enable_tx_int()
```

Include file

`Ser31.h`

Purpose

To enable the serial receive or transmit interrupt.

Description

The enable bit for the interrupt RINT0 or XINT0 is set in the DSP's IE register to enable the corresponding interrupt.

Return value

None

Related topics

References

serial_tx_word_poll

Syntax

```
int serial_tx_word_poll(void *word)
```

Include file

`Ser31.h`

Purpose

To transmit a 32-bit data word via the serial interface.

Description

The value can be of either float or long type. If the transmit buffer of the serial interface is empty, this means the port is ready to transmit, and the function writes the value to the buffer.

Note

You have to initialize the receiving serial interface before starting a transmission. Otherwise, you have to initialize the transmitting port again after the initialization of the receiving port.

Parameters

word 32-bit word to be transmitted (datatype float or long)

Return value

Transmission state; the following symbols are predefined:

Predefined Symbol	Value	Meaning
SP_TRUE	0	The transmission has been performed successfully.
SP_FALSE	1	The serial interface was not ready to transmit data.

Related topics

References

serial_rx_word_poll..410

serial_tx_word_int

Syntax

```
void serial_tx_word_int(void *word)
```

Include file

`Ser31.h`

Purpose

To transmit a 32-bit data word via the serial interface in a transmit interrupt service routine.

Description

The data word is written to the transmit buffer of the serial interface.

> **Note**
>
> You have to initialize and enable the transmit interrupt with `serial_tx_int_init` and enable_tx_int before using `serial_tx_word_int`.
> You have to initialize the receiving serial interface before starting a transmission. Otherwise, you have to initialize the transmitting port again after the initialization of the receiving port.

Parameters

word	32-bit word to be transmitted (datatype float or long)

Return value

None

Related topics

References

enable_rx_int, enable_tx_int...407
serial_tx_int_init..405

serial_rx_word_poll

Syntax

```
int serial_rx_word_poll(void *word)
```

Include file

`Ser31.h`

Purpose

To receive a 32-bit data word via the serial interface.

Description

If the receive buffer contains new data the buffer is read and the function returns SP_TRUE. Otherwise, the buffer is not read and the function returns SP_FALSE.

Parameters

word 32-bit word to be received (datatype can be float or long)

Return value

Transmission state; the following symbols are predefined:

Predefined Symbol	Value	Meaning
`SP_TRUE`	0	The transmission has been performed successfully.
`SP_FALSE`	1	The serial interface was not ready to transmit data.

Related topics

References

serial_tx_word_poll........408

serial_rx_word_int

Syntax

```
void serial_rx_word_int(void *word)
```

Include file

`Ser31.h`

Purpose

To receive a 32-bit data word via the serial interface in a receive interrupt service routine.

Description

The data word is read from the receive buffer of the serial interface directly.

Note

You have to initialize and enable the receive interrupt with `serial_rx_int_init` and `enable_rx_int` before using `serial_rx_word_int`.

Parameters

word	32-bit word to be received (datatype can be float or long)

Return value

None

Related topics

References

enable_rx_int, enable_tx_int.. 407
serial_rx_int_init.. 404
serial_tx_word_int... 409

Execution Time Measurement

Introduction

To measure the execution times of parts of your slave DSP application.

Where to go from here

Information in this section

Example of Execution Time Measurement

Example

To measure the execution time of function 1 and 3:

```
void isr_t0()
{
   ...
   /* start execution time measurement     */
   tic1_start();
   function1(arg);
   /* halt execution time measurement      */
   tic1_halt()
   function2(arg);
   /* continue execution time measurement  */
   tic1_continue();
   function3(arg);
   /* read execution time of function 1 and 3 */
   exec_time = tic1_read();
   ...
}
void main()
{
   ...
   /* initialize timer 1 */
   tic1_init();
   ...
}
```

tic0_init, tic1_init

Macro

```
void tic0_init()
void tic1_init()
```

Include file

`Tic3x.h`

Purpose

To initialize and start timer 0 or 1 for execution time measurement.

> **Note**
>
> Do not call this macro if the respective timer is already in use, for example, for timer interrupt generation.

Return value

None

Related topics

References

tic0_start, tic1_start..414

tic0_start, tic1_start

Macro

```
void tic0_start()
void tic1_start()
```

Include file

`Tic3x.h`

Purpose

To start execution time measurement.

Return value

None

Related topics

References

tic0_halt, tic1_halt 414
tic0_read, tic1_read 415

tic0_halt, tic1_halt

Macro

```
void tic0_halt()
void tic1_halt()
```

Include file

`Tic3x.h`

Purpose

To pause the time measurement.

Description

The break lasts until the measurement is resumed by `tic0_continue` or `tic1_continue`.

Return value

None

Related topics

References

tic0_continue, tic1_continue......415
tic0_start, tic1_start......414

tic0_continue, tic1_continue

Macro

```
void tic0_continue()
void tic1_continue()
```

Include file

`Tic3x.h`

Purpose

To resume time measurement after it has been paused by `tic0_halt` or `tic1_halt`.

Return value

None

Related topics

References

tic0_halt, tic1_halt......414
tic0_read, tic1_read......415

tic0_read, tic1_read

Macro

```
float tic0_read()
float tic1_read()
```

Include file

`Tic3x.h`

Purpose

To read the time period since the time measurement was started by `tic0_start` or `tic1_start` minus the breaks (from `tic0_halt` to `tic0_continue` or from `tic1_halt` to `tic1_continue`) that were made.

Note

Use `tic0_read_total` or `tic1_read_total` to read the complete time period including the breaks that were made.

Return value

Time duration in seconds

Related topics

References

tic0_read_total, tic1_read_total

Macro

```
float tic0_read_total()
float tic1_read_total()
```

Include file

`Tic3x.h`

Purpose

To read the complete time period since the time measurement was started by `tic0_start` or `tic1_start`, including all breaks (from `tic0_halt` to `tic0_continue` or from `tic1_halt` to `tic1_continue`) that were made.

Note

Use `tic0_read` or `tic1_read` to read the time period minus the breaks that were made.

Return value

Time duration in seconds

Related topics

References

tic0_delay, tic1_delay

Macro

```
void tic0_delay(float duration)
void tic1_delay(float duration)
```

Include file

`Tic3x.h`

Purpose

To hold the program execution for a specified time.

Parameters

duration delay time in seconds. The minimum delay time is 1.4 µs. The delay time can be adjusted in steps of 0.4 µs with a maximum error of +0.5 µs (optimization at level -o2 assumed).

Return value

None

Host PC Settings

Introduction

The following topics deal with the slave DSP software environment of the DS2211 and some board-related utilities.

Where to go from here

Information in this section

Environment Variables and Paths

Envirnment variables

The following environment variables are defined:

Environment Variable	Meaning
`PPC_ROOT`	Root folder of the PPC compiler.
`TI_ROOT`	Root folder of the Slave DSP Compiler. You must specify the compiler path after installation. For more information, refer to How to Set the Compiler Path on page 419.
`DSPACE_ROOT`	Root folder of the dSPACE software.

When you use the Command Prompt for dSPACE RCP and HIL, these definitions are set automatically.

Search path

The following subdirectories are included in the search path of your host system:

Subdirectory	Meaning
`%PPC_ROOT%\Bin`	Executable programs of the PPC compiler
`%TI_ROOT%`	Executable programs of the Slave DSP Compiler
`%DSPACE_ROOT%\Exe`	Executable programs of the dSPACE software

Related topics

HowTos

How to Set the Compiler Path..419

How to Set the Compiler Path

Objective

Before you can use `Cl2211.exe` to compile and link source code for the slave DSP of your DS2211 board, you have to specify the installation path of your Texas Instruments Compiler (TI Compiler) as an environment variable.

Method

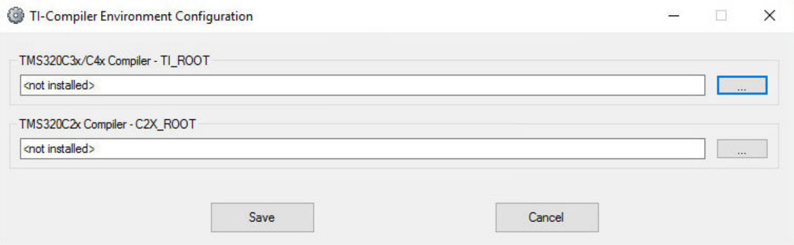
To set a compiler path

1 From the Windows Start menu, select dSPACE RCP and HIL <ReleaseVersion> **-** Command Prompt for dSPACE RCP and HIL <ReleaseVersion>.

A command prompt with required default settings is started.

2 Type `DsConfigTiEnv` and click **Enter**.

The TI-Compiler Environment Configuration dialog opens.



TMS320C3x/C4x Compiler - TI_ROOT

<not installed>

TMS320C2x Compiler - C2X_ROOT

<not installed>

3 Click the Browse button in the TMS320C3x/C4x Compiler - TI_ROOT setting to open a file explorer.

4 Navigate to the *main path* of the installed TI Compiler and click OK.

The main path to be specified depends on the installed compiler.

Compiler	Main Path
C3x/C4x TI Compiler Version 4.70 C3x/C4x TI Compiler Version 5.11	<InstallationPath>\c3xtools
C3x/C4x Code Composer Tools	<InstallationPath>\tic3x4x

5 Close the dialog by clicking **Save**.

Result

The compiler path of your TI compiler is set. The required paths for compiling and linking the source code of the slave DSP are now available in the **Command Prompt for dSPACE RCP and HIL**.

Related topics

References

Cl2211.exe..423

Folder Structure

Introduction

The folder structure of the software for the DS2211 slave DSP is as follows:

Folder	Contents
`<RCP_HIL_InstallationPath>\DS2211\SlaveDSP\RTLib`	Source and library files of the DS2211 RTLib, makefiles and linker command file for DS2211 slave applications
`<RCP_HIL_InstallationPath>\DS2211\SlaveDSP\Apps`	Standard slave DSP application wheel speed and knock sensor
`<RCP_HIL_InstallationPath>\DS2211\Can`	Firmware for the CAN controller
`<RCP_HIL_InstallationPath>\DS100x\RTLib`	Source and library files for DS2211 master applications
`<RCP_HIL_InstallationPath>\Exe`	Batch files for handprogramming the PPC; DS2211 host programs
`<RCP_HIL_InstallationPath>\Demos\DS100x\IOBoards\Ds2211\SlaveDSP`	Demo examples for the slave DSP
`<RCP_HIL_InstallationPath>\Demos\DS100x\IOBoards\Ds2211\APU`	Demo examples for the APU
`<RCP_HIL_InstallationPath>\Demos\DS100x\IOBoards\Ds2211\DSSER`	Demo examples for the serial interface

Software Environment

Introduction

The basic software environment of the slave DSP comprises macros and functions to perform the system initialization, to access the built-in I/O features and control interrupt operations. All necessary files are copied to `<RCP_HIL_InstallationPath>\Ds2211\SlaveDSP` or `<RCP_HIL_InstallationPath>\DS100x` during software installation.

Ds2211.lib

Operations that are not used in time-critical program parts are implemented as functions collected in the real-time library `Ds2211.lib`. All the functions were compiled with the highest optimization level.

Header files

Time-critical operations, such as I/O access, are implemented as macros collected in the header files `Ds2211.h`, `Util2211.h` and `Tic3x.h`.

The following modules are included:

Module (Headerfile)	Contents
`Brtenv.h`	Basic real-time environment
`Dma31.h`	Access functions for the slave DSP’s DMA controller
`Ds2211.h`	All setup and I/O access functions for the slave DSP (unless otherwise noted)
`Ser31.h`	Access functions for the slave DSP’s serial interface
`Tic3x.h`	Definitions and macros for user-specific execution time measurement
`Util2211.h`	Definitions and macros for debugging purposes and turnaround time measurement

Note

You have to link `Ds2211.lib` to each slave DSP application. This is done automatically when you use the standard compile and link utility CL2211 (refer to `Cl2211.exe` on page 423).

File Name Extensions

Introduction

The following naming conventions for file names are used:

File Extension	Meaning
.asm	Assembly source files for the slave DSP
.c	C source files
.lib	Library files

File Extension	Meaning
.mk	Makefiles
.lk	Linker command file

Batch Files, Makefiles, Linker Command Files

Introduction

The following batch files, makefiles and linker command files are available to customize the software environment and to implement user slave DSP applications.

Where to go from here

Information in this section

Cl2211.exe

Syntax

```
cl2211 file[.c] [options] [/?]
```

Purpose

To compile and link a C-coded source file (`File.c`) for the slave DSP with the makefile `Ds2211.mk`.

Description

If you do not specify the file extension, the program searches a file of the relevant type in the current folder.

The action depends on the given file type:

- Local makefile (.mk)

  To execute the given local makefile (see `Ds2211.mk` and `Tmpl2211.mk`). To compile the application using different options, you first have to delete the object files manually. The resulting program file is named according to the given makefile.
- C-coded sourcefile (.c)

  To compile and link the C-coded sourcefile with `Ds2211.mk`. Object files are deleted automatically. The resulting program file is named according to the given file.

- Assembly-coded sourcefile (.asm)

  To assemble and link the assembly-coded sourcefile with `Ds2211.mk`. Object files are deleted automatically. The resulting program file is named according to the given file.

After translating and building an object file, the COFF file conversion utility COFFCONV will be started (refer to `coffconv` on page 427).

Options

The following command line options are available:

Option	Meaning
/a	Object file will be converted to an assembly file instead of a C file.
/ao <option>	Additional assembler options; refer to the Texas Instruments Assembler documentation.
/co <option>	Additional compiler options; refer to the C compiler documentation.
/f	Object file will be converted to a C file. (default)
/g	Enables symbolic debugging (using the CL30 options –g –as).
/l	Writes all outputs to the file `Cl2211.log`.
/n	Disables beep on error.
/p	Pauses execution of `Cl2211.exe` after errors. The Command Prompt window is not closed automatically. This allows you to read error messages.
/s	Optimizes assembly code (refer to `speedy.exe` on page 435).
/so <option>	Additional speedup option (can be used several times) to be passed to `speedy.exe` (refer to `speedy.exe` on page 435)
/x	Switches code optimizing off.
/?	Displays a list of the options available.

Error Message

The following error messages are defined for `Cl2211.exe`:

Message	Meaning
ERROR: not enough memory!	The attempt to allocate dynamic memory failed.
ERROR: environment variable TI_ROOT not found! Please open 'Command Prompt for RCP and HL' and enter the following command to configure the compiler path: 'DsConfigTiEnv.exe'	The environment variable TI_Root could not be found. For more information, refer to How to Set the Compiler Path on page 419.
ERROR: unable to access file <file_name>! ...	The specified file could not be accessed. Either another application has locked the file or the file does not exist.
ERROR: file <file_name> not found!	The specified file was not found.
ERROR: can't redirect stdout to file! ERROR: can't redirect stdout to screen!	The redirection of the standard output to a file or to the screen has failed.
ERROR: can’t invoke ..\DSPACE\DSMAKE!	Starting `Dsmake.exe` failed. Check if `Dsmake.exe` is located in the given folder.

Message	Meaning
ERROR: making of <file_name> failed ERROR: assembling of <file_name> failed ERROR: compiling of <file_name> failed	An error occurred while executing a makefile, compiling, or assembling a source file. Refer to the standard output to get information on the error reason, for example, programming errors in the source file.

Related topics

HowTos

How to Set the Compiler Path..419

References

Ds2211.mk and Tmpl2211.mk...427

DS2211.lk, DS2211_1.lk, DS2211_2.lk

Description

The linker command file `DS2211.lk` is located in `<RCP_HIL_InstallationPath>\Ds2211\SlaveDSP\RTLib`. It is automatically used for linking if you use `Cl2211.exe` and if no local linker command file exists in the folder containing the application source file.

`DS2211.lk` defines where to place the STARTUP code and the different sections created by the C compiler in the slave DSP's memory and instructs the linker which object modules and libraries have to be linked.

Standard linker command file

The standard linker command file `DS2211.lk` is listed below to show the standard settings:

```
/* DS2211.LK ************************************************************
  System memory map for the DS2211 board VC33 DSP
  comment : used as linker command file
  (C) 2003 dSPACE GmbH
  $RCSfile: ds2211.lk $ $Revision: 1.1 $ $Date: 2003/09/08 11:18:18GMT+01:00 $
************************************************************************/
-stack 0x03c1                                        /* 961 word stack */
-heap  0x03fe                                        /* 1022 word heap */
MEMORY
{
  VECS: org = 0x809fc1 len = 0x00000b      /* INT branches             */
  TRAP: org = 0x809fe0 len = 0x000020      /* TRAP branches            */
  BOOT: org = 0x809800 len = 0x000002      /* reserved for boot loader */
  RAM0: org = 0x809802 len = 0x0003fe      /* RAM block 0              */
  RAM1: org = 0x809c00 len = 0x0003c1      /* RAM block 1              */
  RAM2: org = 0x800000 len = 0x004000      /* RAM block 2, 16KW        */
  RAM3: org = 0x804000 len = 0x004000      /* RAM block 3, 16KW        */
  DMEM: org = 0x600000 len = 0x004000      /* 16KW dual-port memory    */
}
```

```
/* section allocation into memory */
SECTIONS
{
  .startup:           > RAM2    /* startup code                  */
  .vectors:           > VECS    /* RESET vector                  */
  .trap:              > TRAP    /* TRAP vectors                  */
  .text:              > RAM2    /* C-code                        */
  .cinit:             > RAM2    /* initialization tables         */
  .const:             > RAM3    /* string literals and switch tables */
  .data:              > RAM3    /* initialized data              */
  .stack:             > RAM1    /* system stack                  */
  .bss:               > RAM3    /* global & static variables     */
  .sysmem:            > RAM0    /* dynamic memory                */
}
/* modules which are always linked */
-u startup
```

Local linker command file

If you need an individual memory layout for an application, you can use a local linker command file. Local linker command files must use the file name of the corresponding application C-coded source file and the suffix .lk. If `Cl2211.exe` detects a local linker command file in the folder containing the application, this file will be used for linking instead of the standard linker command file.

Individual sections in the DSP memory

There are several possible ways to place the individual sections in the DSP memory. The .bss section comprising global and static data could also reside in on-chip 16 KW memory (RAM2), while the code section .text remains in the on-chip 16 KW memory (RAM3). For example:

```
.text:          > PMEM       /* C-code                    */
.bss:          > RAM1       /* global & static variables  */
```

Both sections arranged in the internal memory blocks RAM2 and RAM3, as specified in the default linker command file, avoid the lack of performance, since OP code and operands are accessed via a separate data bus.

You can use one of the two following command files as your local linker command file:

Ds2211_1.lk If the size of the respective sections exceeds the limited size of RAM2 and RAM3, the linker command file `Ds2211_1.lk` can be used as the local linker command file. It will assign all sections to the internal 2x16 KW memory RAM without considering the internal memory block boundaries.

Note

This will slow down the performance of the application, because OP code and operands may access via a single data bus and stack and heap are assigned to the small 2 KW on-chip memory block.

Ds2211_2.lk If the 2 KW on-chip memory is not sufficient for the stack and the dynamic memory of the application, the linker command file `Ds2211_2.lk` can be used as the local linker command file.

`Ds2211_2.lk` assigns the all sections to the 2x16 KW on-chip memory without considering the memory boundaries.

Increasing heap and stack size

Additional options to increase the sizes of the heap and the stack may be defined in the linker command file. The heap is located in the .sysmem section and the stack is located in the .stack section. The stack is used for context save, local variables and to pass parameters to functions. The heap is used for memory allocated with `malloc()`, that is, for dynamic data. The default sizes of the heap and the stack are:

```
-stack 0x03c1                           /* 961 word stack */
-heap  0x03fe                           /* 1022 word heap */
```

Related topics

References

Cl2211.exe..423

Ds2211.mk and Tmpl2211.mk

Purpose

Makefile to compile or assemble a specified source file(s).

Description

- When using `CL2211` it invokes `DSMAKE` with the default makefile `Ds2211.mk`.
- Use `Tmpl2211.mk` as a template if you want to generate your own local makefiles. Copy this file to the local folder, rename the file (to the same name as your application to be built), specify the C- or assembler-coded source files, and call `Cl2211.exe`.

Related topics

References

Cl2211.exe..423

coffconv

Syntax

```
coffconv obj_file [options] [/?]
```

Purpose

To convert a COFF (common object file format) object file to an assembly file that can be included into a master application. `coffconv` adds the prefix *Slv2211_* to the name of the given object file.

Options

The following command line options are available:

Option	Meaning
/a	Generates an assembly file with the default extension `asm`.
/b	Generates a binary file with the default extension `bin`.
/slc	Generates a C-source file with the default extension `slc`.
/n	Disables beep on error.
/o <output_file>	Name of the file to be generated
/q	Quiet mode
/t <board_type>	Specifies the target board type for the object file to be converted. By default: DS2210
/?	Displays a list of the options available.

Generated files and loading mechanism

C-source file The coffconv output file contains a data array named according to the converted object file. The data array is needed for the master processor loader function (refer to `ds2211_slave_dsp_appl_load` on page 305).

> **Note**
>
> The data array remains in the memory of the master processor after the application has been loaded to the slave DSP.

Assembler file The coffconv output file contains the data section `SlvSect` with the application data. This section will be loaded by the host loader to the master's memory only temporarily. When the application has been loaded to the slave DSP, the data section will be cleared from the memory. For more information on the slave loading procedure, refer to Loading Slave Applications on page 437.

Binary file The coffconv output file contains the application data and can be used by other conversion tools.

Example

```
coffconv demo.obj -c -t DS2211
```

The object file `Demo.obj` will be converted to the C file `Slv2211_demo.slc`.

Execution Time Information

Where to go from here

Information in this section

Basics of Using speedchk

Introduction

To calculate the execution time of the timer interrupt service routine (ISR) in DS2211 application programs you can use the `speedchk` macro.

Evaluated times

Since many application programs comprise various program paths of different length, `speedchk` evaluates the minimum and maximum execution time. The minimum, maximum and current number of timer ticks (tcount) will be computed by `speedchk` on the DSP and transferred to the master DSP via the DPMEM addresses 0x063FFB … 0x063FFD (as seen by the master DSP).

The `ds2211_slave_dsp_speedchk` function reads the minimum, maximum and current execution time from the DPMEM and supplies the parameters to be displayed by ControlDesk.

Resolution

The resolution is one timer tick (that is, 50 ns for an 150 MHz DS2211 slave DSP) and the maximum error is one timer tick.

Function overview

To calculate execution time information, use `speedchk`.

To read execution time information in a DS2211 application, use `ds2211_slave_dsp_speedchk`.

Related topics

References

ds2211_slave_dsp_speedchk..304

speedchk

Syntax

```
speedchk(i)
```

Purpose

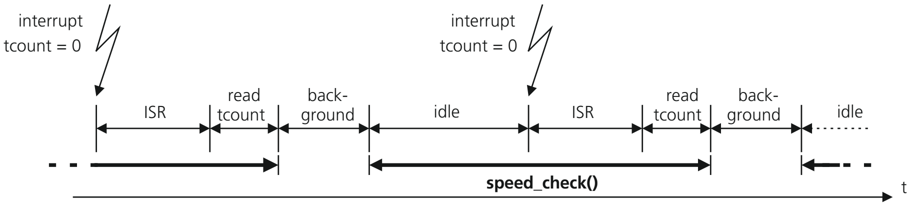
To get execution time information you have to include the macro `speedchk(i)` from the header file `Util2211.h` into the background loop of a DS2211 application program.

Note

`speedchk` contains the assembly instruction `idle` that waits for an interrupt. Thus, any additional code in the background loop will be executed only once each time an interrupt is received. The `idle` instruction also sets the GIE bit in the ST register of the slave DSP, which enables interrupts globally.

`speedchk` will not work properly if any other interrupt except for a single timer interrupt is used in the application.

The following illustration shows how speedchk works:



Parameters

i slave DSP timer to be used. The following values are allowed:

Value	Meaning
0	Timer 0
1	Timer 1

The specified timer must match the timer that is actually used to generate the sampling clock interrupts.

Example

The following shows an example of timer 0:

```
timer0(TS);                            /* initialize timer0 */
for (;;)
...
```

```
#include "Slv2211_ks_appl.slc"
/*************************************************************
object declarations
*************************************************************/
/* pointer to slave DSP application data */
extern unsigned long ks_appl[];
...
/*-----------------------------------*/
void main()
{
   ...
   init();                                /* init CPU board */
   ds2211_initds2211_init(DS2211_1_BASE);         /* init DS2211 boar
d */
   /* load DS2211 slave-DSP application 'ks_appl' */
   ds2211_slave_dsp_appl_loadds2211_slave_dsp_appl_load(DS2211_1_BAS
E,
                           (Int32 *) &ks_appl);
...
}
speedchk(0);          /* include SPEEDCHK code for timer0 */
```

Related topics

References

Assembly Code Optimization

Introduction

To automatically perform code optimization on the assembly level in DS2211 slave applications, you can use an assembly code optimization utility.

Where to go from here

Information in this section

Saving and Restoring the Context

Introduction

Most DS2211 application programs consist only of a single interrupt service routine and contain only little or no code at all in the background loop.

PUSH and POP instructions

For this reason, most of the context save and restore instructions (PUSH/POP) performed at the beginning and at the end of interrupt service routines are not necessary and can be removed in order to save execution time.

`speedy.exe` removes the PUSH and POP instructions from interrupt service routines (`c_int09()` or `c_int10()`, for example). If you use the command line option -k the removed instructions are kept as comments, as shown in the following example.

Example

```
_c_int09:
           PUSH     ST
*o*        PUSH     R0
*o*        PUSHF    R0
*o*        PUSH     R1
*o*        PUSHF    R1
*o*        PUSH     R2
*o*        PUSHF    R2
*o*        PUSH     AR0
*o*        PUSH     AR1
*o*        PUSH     AR2
           ...
*o*        POP      AR2
*o*        POP      AR1
*o*        POP      AR0
*o*        POPF     R2
*o*        POP      R2
*o*        POPF     R1
*o*        POP      R1
*o*        POPF     R0
*o*        POP      R0
           POP      ST
           RETI
```

Register substitution

`speedy.exe` searches interrupt service routines for registers that are already used by the main() routine or by another interrupt service routine. If any register conflicts are detected, registers will be substituted by other unused registers. If no unused register is available, the context save/restore of particular registers remains unaffected. Register substitution within interrupt service routines will be performed in the order of their occurrence in the assembly source code. Thus, the first interrupt service routine gets the highest optimization priority.

Status register ST

Saving the status register (ST) is not affected by `speedy.exe`.

Auxiliary register AR3

Any instructions that use the auxiliary register AR3 will not be changed by `speedy.exe`. This register is used as the frame pointer in Texas Instruments C compiler generated programs.

Use `speedy.exe` with the parameter -v to receive detailed information on the register substitution that is actually performed.

Related topics

References

speedy.exe.. 435

Floating-Point to Integer Conversion

Introduction

The C compiler uses FIX instructions to convert floating-point values to integer values. The FIX instruction rounds towards negative infinity, followed by a 4-instruction sequence to correct negative values.

In DS2211 application programs floating-point to integer conversion is needed for the on-board D/A converters (refer to D/A Converter (Slave DSP) on page 367). In this case, the correction of negative values is not necessary, due to the limited precision of the D/A converter.

When `speedy.exe` detects an appropriate code sequence in conjunction with the keyword @_dac in the following load instruction, the extra instructions will be removed:

```
               FIX      R1,R3
    *o*        NEGF     R1
    *o*        FIX      R1
    *o*        NEGI     R1
    *o*        LDILE    R1,R3
               LDI      @_dac1,AR2
               STI      R3,*AR2
```

Note

- `speedy.exe` only detects the correction sequence if a TI compiler version 4.7 or lower is used.
- For faster execution, the correction sequence for floating-point to integer conversion can also be suppressed with the CL30 option `-mc`. However, this will affect each floating-point to integer conversion.

Related topics

References

speedy.exe..435

Optimization Limitations

Introduction

When using `speedy.exe` you have to consider the following limitations:

- If an application program contains function calls, `speedy.exe` uses the exact register usage information for local functions and for the run-time support arithmetic routines from the run-time library `rts30.lib` (div, mod, etc.). All other functions are assumed to use all registers.
- Register usage of local functions is evaluated in order of their occurrence; that is, if a function is called prior to its declaration, use of all registers is assumed.

- `speedy.exe` was designed to be applied to assembly source code generated with the Texas Instruments C compiler. If you use handcoded assembly programs or inline assembly statements, macro definitions and substitution symbols must not be used together with `speedy.exe`.
- You should use `speedy.exe` only for application programs that consist of a single assembly source file, except for the standard object modules from the object library `Ds2211.lib`. In case of modular programs, special care must be taken that no externally linked object code can be interrupted by interrupt service routines that were optimized with `speedy.exe`. Otherwise register conflicts may cause unpredictable system behavior.
- Interrupt service routines optimized by `speedy.exe` are no longer reentrant. So, whenever interrupts are enabled in an interrupt service routine, you have to ensure that the interrupt service routine is never interrupted by itself. Otherwise, registers will be corrupted, which will cause unpredictable results. If you use timer interrupt service routines, the sampling rate must be chosen appropriately to make sure that an interrupt service is already finished before the next interrupt will be received. Select a sufficiently large sampling period first and use `speedchk` to evaluate the actual execution time.

Related topics

References

speedchk.. 430
speedy.exe.. 435

speedy.exe

Syntax

```
speedy [-v] [-k] [-o outfile] <asmfile>
```

Purpose

To perform assembly code optimization.

Description

To automatically perform code optimization on the assembly level in DS2211 slave applications you can use the `speedy.exe` assembly code optimization utility. The optimization removes unnecessary context saving and restoring instructions from the timer interrupt service routine and the unnecessary code for floating-point to integer type conversion in conjunction with data output to the D/A converter.

Use `Cl2211.exe` with the command line option -s to perform the optimization when compiling and linking the application. If you need a different behavior you can invoke `speedy.exe` directly.

Parameters

-v Generates verbose information about register use, subroutine calls, and register replacements

-k Keeps removed assembly instructions as comments

-o outfile Output file for the resulting optimizer output. By default, the output is written to `Speedup.out`.

<asmfile> ASM file to be optimized

Note

The assembly source file must be specified including the suffix asm.

Related topics

References

CI2211.exe...423

Loading Slave Applications

Where to go from here

Information in this section

Basics of Loading Slave Applications

Introduction

The host PC cannot access a slave DSP directly for loading a slave application. To load a slave application to the slave DSP, it must be included in the real-time application of the master processor board in an intermediate format. When the real-time application is executed on the processor board, the slave application is loaded to the slave DSP via the PHS bus.

You can load a slave application with permanently or temporarily available slave data. For the DS2211 board loading with permanently available slave data is the default loading procedure.

Tip

If you load the slave application via a slave application data file with permanently available slave data, a lot of memory is used, because each generated file contains a global data array with slave application data. To avoid this disadvantage, you can use the slave loading via a generated slave application data file with temporary available slave data.

How to Load a Slave Application with Permanently Available Slave Data

Objective

This loader concept allows to load the slave applications by the master processor using permanently available slave application data. The slave application data is stored as a C array in the .bss section of the master processor memory. The .bbs section is used for global variables and the slave application data is therefore permanently available.

The compile and link utility `CL2211.exe` compiles the slave application data and converts it to a C array by using the `coffconv` utility (see `coffconv` on page 427). The slave DSP application data of this C array must be loaded to the slave DSP using the `ds2211_slave_dsp_appl_load` function.

Method

To load a slave application with permanently available slave data

1. On the Windows Start menu, select dSPACE RCP and HIL 20xx-x – Command Prompt for dSPACE RCP and HIL 20xx-x to open a Command Prompt window in which the required paths and environment settings are preset.
2. Change to the folder of the slave application.
3. To compile and convert the slave application, enter the following command:

   ```
   CL2211 test_prg.c
   ```

   This generates a C file called `Slv2211_test_prg.slc` containing the slave application data. Copy this file to the folder of your master processor application.
4. Add the following marked lines to your master processor application to load the slave application via the master processor board:

   ```
   /* DS2211 slave application data */
   #include Slv2211_test_prg.slc
   extern unsigned long test_prg[];
   void main(void)
   {
      init();        /* initialize master processor system */
      ds2211_init(DS2211_1_BASE);     /* initialize DS2211 */
      ds2211_slave_dsp_appl_load(DS2211_1_BASE, (Int32 *) &test_prg);
      ...
   }
   ```
5. To compile and load your master processor application, enter the following command:

   ```
   down<xxxx> master.c
   ```

`down<xxxx>` must correspond to the processor board type, for example, `down1006` for a DS1006.

Related topics

References

How to Load a Slave Application with Temporarily Available Slave Data

Objective

This loader concept allows slave applications to be loaded by the master processor. The slave application is not stored in the master processor memory permanently.

> **Note**
>
> Do not use this loader concept within an S-function. You can use the loader concept with permanently available slave data instead (see How to Load a Slave Application with Permanently Available Slave Data on page 438).

Assembly file

The slave application object file must be converted into an assembly file containing the application data to be loaded into the slave DSP memory. To do this, the compile and link utility `CL2211.exe` invokes the `coffconv` utility (refer to `coffconv` on page 427). The assembly file assigns the slave DSP application data to the `SlvFwSection` section on PowerPC-based systems.

After that, the object file of the master processor application contains the `SlvFwSection` section with the slave DSP application data. When the host loader detects this section, it loads its contents into the `.hostmem` section of the master memory. You can link more than one slave application to the master application by using several DS2211 boards. The data of the additional applications will also be stored in the `SlvFwSection` section.

.hostmem section

The slave application is temporarily stored in the `.hostmem` section of the master processor, which is used for the trace buffer after the master application has been started. Use the `ds2211_slave_dsp_appl_load` function to load the application data from the `.hostmem` section into the memory.

Start addresses

The start address of each slave application can be accessed by a global symbol that has the same name as the slave application.

Example

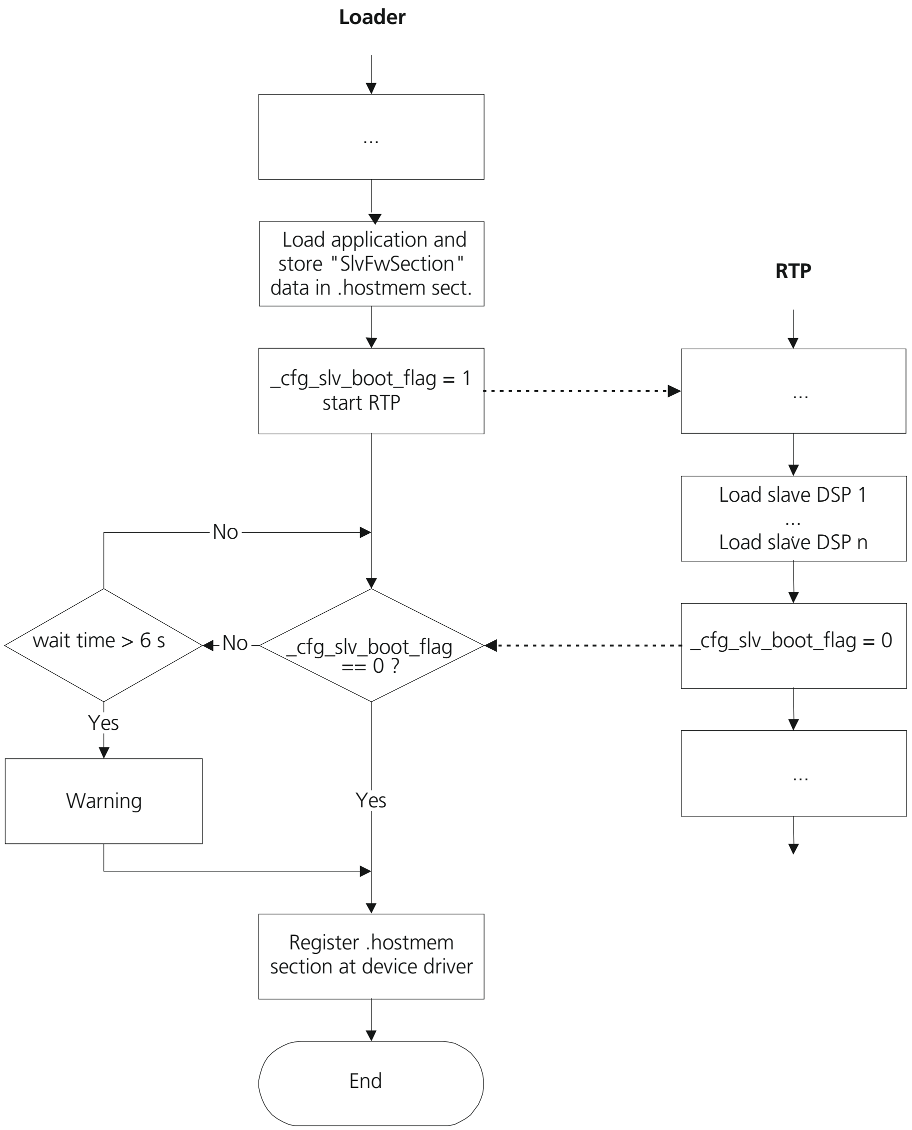
If the slave application is named `mytest`, the global symbol `mytest` will contain the address of the first data element of the `mytest` application in the `SlvFwSection` section.

Clearing .hostmem section

When the master processor has loaded all slave DSPs, the slave application data is no longer needed and the memory of the `.hostmem` section can be used by the trace buffer. The `.hostmem` section memory is registered at the device driver for use as a trace buffer. The host loader therefore needs to know when the slave loading procedure has finished and the `.hostmem` section can be registered. This requires communication between the real-time processor and the host PC. This communication is done by a flag variable in the `.config` section called _cfg_slv_boot_flag. The host loader sets the flag after downloading the application and before starting the master processor. Then it waits for the flag to be cleared, when slave loading has completely finished.

> **Note**
>
> You must clear the flag by calling the `RTLIB_SLAVE_LOAD_ACKNOWLEDGE` macro after all the slave loader functions have been executed. The host loader waits approximately six seconds for the flag to be cleared, after which it registers the `.hostmem` section and displays a warning message.

The following illustration shows the slave loader concept.



Method

To load a slave application with temporarily available slave data

1 On the Windows Start menu, select dSPACE RCP and HIL 20xx-x – Command Prompt for dSPACE RCP and HIL 20xx-x to open a Command Prompt window in which the required paths and environment settings are preset.

2 Change to the folder of the slave application.

3 To compile and convert the slave application, enter the following command (The switch “-a” forces CL2211 to generate an assembly file instead of an C file):

```
CL2211 test_prg.c -a
```

You will receive a generated assembly file `Slv2211_test_prg.asm` containing the slave application data. Copy this file to the folder of your master processor application.

4. Add the following marked lines to your master processor application to load the slave application via the master processor board:

```
/* DS2211 slave application data */
extern unsigned long test_prg;  /* slave appl. data block*/
void main(void)
{
   init();         /* initialize master processor system */
   DS2211_init(DS2211_1_BASE);      /* initialize DS2211 */
   DS2211_slave_dsp_load_application(DS2211_1_BASE,
            (Int32 *) &test_prg);
   ...
}
```

5. To compile and load your master processor application and the assembly file, enter the following command:

   `down<xxxx> master.c Slv2211_test_prg.asm`

   `down<xxxx>` must correspond to the processor board type, for example, `down1006` for a DS1006.

Related topics

References

coffconv..427
ds2211_slave_dsp_appl_load..............................305

Slave DSP Demo Applications

Introduction

The demo applications demonstrate the features of the slave DSP.

To view the applications and modify the parameters you have to use ControlDesk and load the experiments defined for each demo application.

Where to go from here

Information in this section

Knock Sensor Simulation Demo

Introduction

This demo application shows how to use the standard knock sensor simulation application on the slave DSP. Knock sensor signals are often needed in automotive applications, for example, for hardware-in-the-loop simulations in conjunction with an electronic engine control unit (ECU).

The knock application consists of the master processor application `Slv_knock_2211_hc.c` and the slave DSP application `Ks.c`. The demo application is installed in `<RCP_HIL_InstallationPath>\Demos\DS100x\IOBoards\Ds2211\SlaveDSP\Knock`.

Slave DSP application

The slave DSP application generates the knock signals for up to 8 cylinders. The knock signal of each cylinder can be assigned to one of four D/A output channels. The engine angle is read from the angular processing unit (APU).

Master processor application

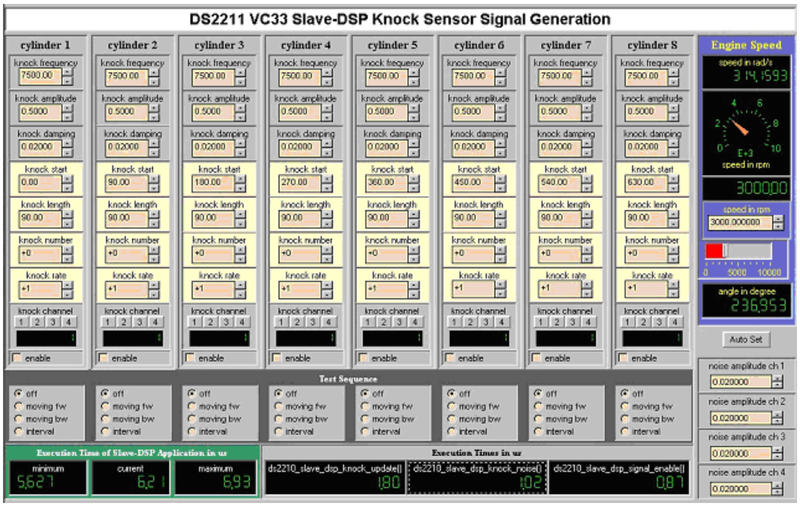
The master processor application allows you to modify the knock signal variables and the engine speed.

Viewing knock signals

To view the knock signals, connect the wave form outputs of channel 0 and 1 to an oscilloscope. To receive a signal synchronized to the engine angle, use the PHI15 pin of the P5 connector as external trigger signal.

Modifying parameters

Use ControlDesk to modify the knock signal parameters. The layout of the knock application is shown in the following illustration.



DS2211 VC33 Slave-DSP Knock Sensor Signal Generation

	cylinder 1	cylinder 2	cylinder 3	cylinder 4	cylinder 5	cylinder 6	cylinder 7	cylinder 8
knock frequency	7500.00	7500.00	7500.00	7500.00	7500.00	7500.00	7500.00	7500.00
knock amplitude	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
knock damping	0.02000	0.02000	0.02000	0.02000	0.02000	0.02000	0.02000	0.02000
knock start	0.00	90.00	180.00	270.00	360.00	450.00	540.00	630.00
knock length	90.00	90.00	90.00	90.00	90.00	90.00	90.00	90.00
knock number	+0	+0	+0	+0	+0	+0	+0	+0
knock rate	+1	+1	+1	+1	+1	+1	+1	+1
knock channel	1	1	1	1	1	1	1	1
☐ enable	☐	☐	☐	☐	☐	☐	☐	☐
Test Sequence	● off	● off	● off	● off	● off	● off	● off	● off

Engine Speed

speed in rad/s 314.1593

speed in rpm 3000.00

speed in rpm 3000.000000

angle in degree 236.953

noise amplitude ch 1 0.020000

noise amplitude ch 2 0.020000

noise amplitude ch 3 0.020000

noise amplitude ch 4 0.020000

Execution Time of Slave-DSP Application in us

minimum 5.627

current 6.21

maximum 6.93

Execution Times in us

ds2210_slave_dsp_knock_update() 1.80

ds2210_slave_dsp_knock_noise() 1.02

ds2210_slave_dsp_signal_enable() 0.87

On the right side of the layout you can specify the engine speed (0 … 10000 rpm) and the number of cylinders to be used for knock sensor simulation (1 … 8). The Auto Set button will set the knock parameters to the default values for the specified number of cylinders. The execution times of the slave DSP application and of the DS2211 board access function are displayed.

On the left side of the layout the knock signal parameters can be specified. For a description of the knock signal parameters, refer to `ds2211_slave_dsp_knock_init` on page 312.

In the test sequence box some test cycles can be enabled. The *move fw* mode will move the start angle of the knock signal forward continuously. The *move bw* mode will move the start angle of the knock signal backward continuously. The *interval* mode moves the start angle forward and backward in intervals.

Related topics

References

ds2211_slave_dsp_knock_init..312
Knock Sensor Simulation..308

Wheel Speed Sensor Signal Generation Demo

Introduction

This demo application shows how to use the standard wheel speed sensor signal generation application on the slave DSP. Wheel speed sensor signals are needed in automotive applications, for example, for hardware-in-the-loop simulations in conjunction with an electronic engine control unit (ECU).

For general information on wheel speed sensor simulation and its I/O mapping, refer to Wheel Speed Sensor Simulation (DS2211 Features 🕮).

The wheel speed application consists of the master processor application `Slv_wheel_2211_hc.c` and the slave DSP application `Wheel.c`. The demo application is installed in `<RCP_HIL_InstallationPath>\Demos\DS100x\IOBoards\Ds2211\SlaveDSP\Wheel_speed`.

Slave DSP application

The slave DSP application generates the wheel speed signals for up to 4 channels.

Master processor application

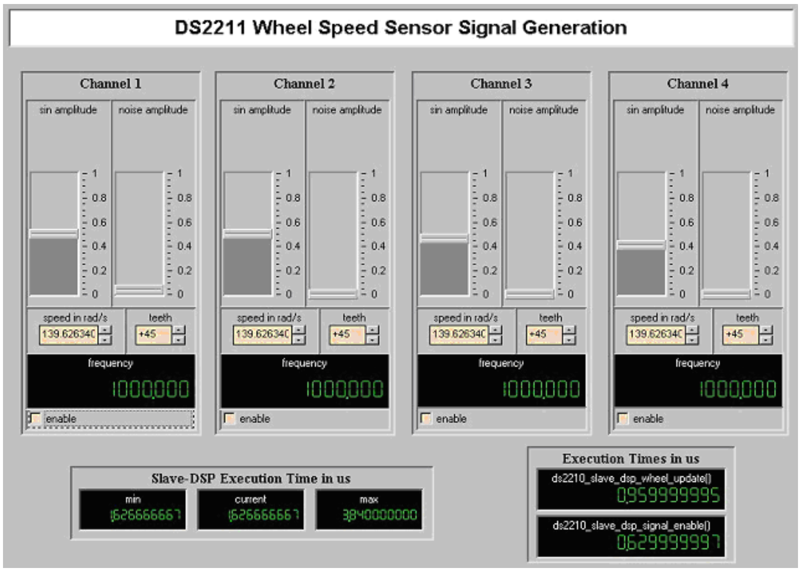
The master processor application is used to modify the wheel speed signal variables via the DPMEM of the DS2211 board.

Viewing wheel speed signals

To view the wheel speed signals connect the wave form outputs of channel 0 … 3 to an oscilloscope.

Modifying parameters

Use ControlDesk to modify the wheel speed signal parameters. The layout of the wheel speed application is shown in the following illustration.



DS2211 Wheel Speed Sensor Signal Generation

Channel 1: speed in rad/s 139.62634(; teeth +45; frequency 1000.000; □ enable

Channel 2: speed in rad/s 139.62634(; teeth +45; frequency 1000.000; □ enable

Channel 3: speed in rad/s 139.62634(; teeth +45; frequency 1000.000; □ enable

Channel 4: speed in rad/s 139.62634(; teeth +45; frequency 1000.000; □ enable

Slave-DSP Execution Time in us: min 1.626666667; current 1.626666667; max 3.840000000

Execution Times in us: ds2210_slave_dsp_wheel_update() 0.959999995; ds2210_slave_dsp_signal_enable() 0.629999997

The speed value must be entered in rad/s, the parameter teeth specifies the teeth number of the simulated wheel speed sensor. For a description of the wheel speed signal parameters, refer to `ds2211_slave_dsp_wheel_init` on page 320.

Related topics

References

Serial Transmission Test Demo

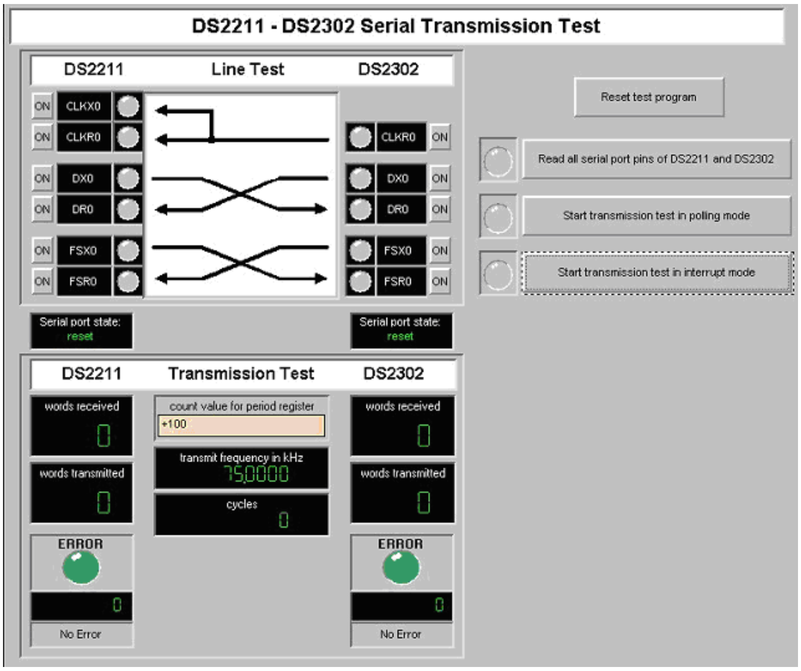
Introduction

This application allows you to test the serial connection between the serial interface of the slave DSP and a DS2302 board.

Serial transmission

The serial transmission test application consists of the master processor application `Slv_test_serial_2211_hc.c`, the slave DSP application `Ser2211.c` and the DS2302 board application `Ser2302.c`. The demo application is installed in `<RCP_HIL_InstallationPath>\Demos\DS100x\IOBoards\Ds2211\SlaveDSP\Test_serial`.

To run this application connect the serial interfaces as described in Serial Interface on page 396.


DS2211 - DS2302 Serial Transmission Test
DS2211
Line Test
DS2302
ON
CLKX0
CLKR0
DX0
DR0
FSX0
FSR0
Reset test program
Read all serial port pins of DS2211 and DS2302
Start transmission test in polling mode
Start transmission test in interrupt mode
Transmission Test
ERROR
No Error

DS2211: Serial port state: reset; words received 0; words transmitted 0; ERROR 0; No Error

count value for period register: +100

transmit frequency in kHz: 75.0000

cycles: 0

DS2302: Serial port state: reset; words received 0; words transmitted 0; ERROR 0; No Error

Click the **read all serial port pins of DS2211 and DS2302** button to configure all serial interface pins as inputs. The input signal level of the pins is displayed by the LEDs in the **Line Test** frame. The two serial interfaces should not be connected. You have to connect a signal to the serial interface pin in this mode. This mode is used to test the hardware of the slave DSP's serial interface.

The **ON** buttons beside the LEDs in the **Line Test** frame can be used to check the connection between the two serial interfaces. The LEDs at the begin and the end of the arrow will flash if the connection is active.

Click the **Start transmission test in interrupt mode** button to test the interrupt-driven serial transmission between the DS2211 and the DS2302 board. The `cycles` variable show how many times a data package, containing 1000 data words, has been transmitted from and to the serial interface.

Click the **Start transmission test in polling mode** button to test the serial transmission in polling mode.

The LEDs on the right and the left side of the layout display the current state of the serial interface. The error LEDs display transmission errors. The error number specifies the number of the data word causing the transmit error.

Click the **Reset test program** button to stop the transfer and set the application to its initial state.

Transferring APU Values to a DS2302 Demo

Introduction

This application demonstrates the APU value transfer from the slave DSP to a DS2302 board via the slave DSP's serial interface.

The APU export application consists of the master processor application `Slv_apu_export_2211_hc.c`, the slave DSP application `Tx2211.c` and the DS2302 board application `Rx2302.c`. The demo applications is installed in `<RCP_HIL_InstallationPath>\Demos\DS100x\IOBoards\Ds2211\SlaveDSP\APU_export`.

Slave DSP application

The DS2211 slave DSP application `Tx2211.c` sends the APU value via the serial interface.

To read the APU value on the DS2211 slave DSP, the DMA controller must be used. The APU value is updated every 1 µs. If the value is read during the APU update, the value may be invalid. After the APU value has been updated, the slave DSP's interrupt INT0 will be requested.

DS2211 DMA initialization

The INT0 interrupt triggers the DMA controller to read the APU value immediately after it is valid and write it into the transmit register of the serial interface. Reading the APU value and writing it to the serial interface using the DMA controller does not consume any execution time of the CPU.

The DMA controller is initialized as shown below:

```
/* Initialize DMA controller for reading APU angle and writing   */
/* it to the serial interface transmit register. DMA transfer    */
/* is triggered by the INT0 interrupt, requested by the APU.     */
dma_init((unsigned long)angle_pos, /* source address (APU value) */
           0x808048,   /* destination address (serial interface  */
                       /* transmit register)                     */
           1,                   /* words to be transmitted       */
           DMA_NO_MODIFY,       /* do not increment/decrement    */
                                /*  source address               */
           DMA_NO_MODIFY,       /* do not increment/decrement    */
                                /* destination address           */
           DMA_SRC_SYNC,        /* DMA read is performed when    */
                                /*  interrupt occurs             */
           DMA_CONTINUOUS,      /* continuous DMA transfers      */
           DMA_TCINT_DISABLE);  /* no DMA->CPU interrupts        */
/* Enable DMA interrupt EINT0. EINT0 is requested by the APU if  */
/* the angle has been updated.                                   */
dma_interrupt_enable(DMA_EINT0);
```

The serial interface is initialized with the following command:

```
/* Initialize serial interface for connection to DS2302.     */
/* The frequency of serial transmission depends on the       */
/* initialization of the DS2302 board's serial interface.    */
serial_init_ds2211();
```

The serial transmission of a 32-bit word takes 4.5 µs, if the fastest transmission frequency of 7.5 MHz is chosen for the DS2302's serial interface. So the serial interface transmit register will be updated 5 times before a new APU value is sent because the APU value is updated every 1 µs.

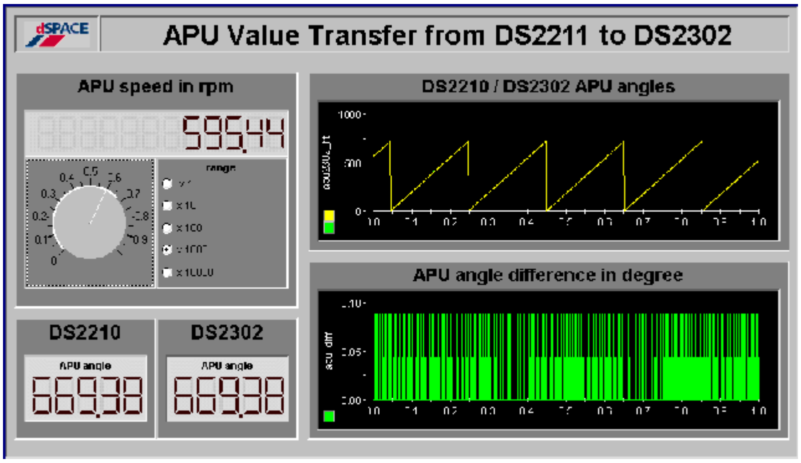
DS2302 DMA initialization

The DS2302 board application `Rx2302.c` also uses the DMA controller to read the received APU value from the serial interface. In this case, the DMA controller is triggered by the serial interface receive interrupt. If the serial interface requests the receive interrupt ERINT0, the APU value is read from the serial interfaces receive register and written to a global variable. The DMA controller and the serial interface are initialized as shown below:

```
/* Initialize serial interface for connection to DS2211. The  */
/* frequency of serial transmission is 7.5 MHz (transmission */
/* of 1 word takes 4.4e-6 seconds).                          */
serial_init_ds2302(1);
/* Initialize DMA controller for reading data from serial    */
/* port receive register and writing it to APU variable.     */
/* The DMA transfer is triggered by the serial interface     */
/* receive interrupt. */
dma_init((unsigned long)0x80804C, /* source address (serial   */
                            /*  interface receive register)  */
         (unsigned long)apu,    /* destination address       */
                                /* (global variable)         */
         1,                     /* words to be transmitted   */
          DMA_NO_MODIFY,        /* do not increment or       */
                                /* decrement source address  */
         DMA_NO_MODIFY,         /* do not increment or       */
                                /* decrement destination     */
                                /* address                   */
         DMA_SRC_SYNC,          /* DMA read is performed     */
                                /* when interrupt occurs     */
         DMA_CONTINUOUS,        /* continuous DMA transfer   */
         DMA_TCINT_DISABLE);    /* no DMA->CPU interrupts    */
/* Enable DMA interrupt ERINT0. ERINT0 is requested by the   */
/* serial interface if it has received new data.             */
dma_interrupt_enable(DMA_ERINT0);
```

Master application

The master application `Slv_apu_export_2211_hc.c` loads the applications to the DS2211 and DS2302 board and reads the APU values from both boards. They are displayed in the layout of ControlDesk shown in the following illustration. The values may be not exactly synchronous because they are not read at the same time.



To run this application, connect the serial interfaces of the boards as shown in the table below. Refer also to Serial Interface on page 396.

DS2302 Board			Direction	DS2211 Board	
Serial Interface	P4 Connector	P7 Connector		P6 Connector	Serial Interface
CLKR (DIGA3)	–	4	—>	2	CLKX
			—>	4	CLKR
DR (DIGA4)	–	5	<—	6	DX
DX (DIGA1)	9	–	—>	8	DR
FSR (DIGA5)	–	6	<—	10	FSX
FSX (DIGA2)	7	–	—>	12	FSR
GND	11	1	–	1, 3, 5, 7, 9, 11	GND

Migration from Other Boards

Migrating DS2302 Applications

Introduction

If you want to use C programs written for the DS2302 DDS board for the DS2211 board you have to consider the following items and restrictions:

- Replace the include files `Ds2302.h` and `Util2302.h` with `Ds2211.h` and `Util2211.h`.
- The macros accessing the interrupts INT0 … INT3 cannot be used on the DS2211 board in the same way as on the DS2302 board.
- The feature of the DS2302 interrupt INT3 is realized on the DS2211 with the interrupt INT1. You have to modify INT3 macro calls of the DS2302 application to macro calls for INT1.
- The D/A conversion output macro dac_out is not available on the DS2211 board because the DS2211 contains 4 D/A converters. Use the macros dac_out1, … dac_out8 instead. To receive the same output voltages as on the DS2302 board you have to divide the floating-point value for DAC0 … DAC3 by two because these DS2211 converters have an output voltage range of ±20 V in the transformer mode instead of ±10 V on the DS2302.

Note

The DS2211 D/A converter analog outputs of the DACs 1 … 4 are connected to audio transformers and it is not possible to output a constant voltage. The minimum frequency for an output signal with the maximum amplitude of ±20 V is 500 Hz. If a constant voltage is needed, you can use the DACs 5 … 8.

- Due to different hardware the following functions and macros are not available on the DS2211 board or have a different functionality than on the DS2302 board:

Macro	RTLib2211
timer0_sync()	not available
dac_out()	different
init_dig_out7()	not available
dig_out7()	not available
dig_in7()	not available
int0_status()	not available
int0_aux_status()	not available
int1_ack()	different
phs_bus_interrupt_request()	not available
int_xf0()	not available

Macro	RTLib2211
int_xf1()	not available
cvtie3()	not available
cvtdsp()	not available

If you use any of these functions and macros in your DS2302 application, you have to adapt the application to the capabilities of the DS2211.

Porting DS2210 Applications to the DS2211 Board

Porting DS2210 Applications to the DS2211 Board

Introduction

It is possible to use C code written for the DS2210 board with the DS2211 board.

Porting DS2210 application

The include files `ds2210.h` and `util2210.h` must be replaced by `DS2211.h` and `util2211.h`.

Although the TMS320C31 code is object compatible to the code of the TMS320VC33 a DS2210 application must be recompiled because the DS2211 needs a different startup-code.

All features available on the DS2210 like interrupts, ADCs and so on are also available on the DS2211.

If you are using custom local linker command files, you must replace it by new ones from the RTLib2211. This is necessary, because the VC33 features 32 KW additional internal on-chip memory. The 64 KW local memory available on the DS2210 was removed on the DS2211 board.

Slave CAN Access Functions

Where to go from here

Information in this section

Basics on Slave CAN Access Functions

Introduction

Provides basics on the communication principles between the master processor board and the slave CAN subsystem, and on the CAN error message types.

Where to go from here

Information in this section

Basic Principles of Master-Slave Communication

Introduction

The master processor board uses slave access functions to control the slave CAN subsystem and exchange data with it.

> **Note**
>
> You have to initialize the communication between the master and the slaves. Refer to `ds2211_can_communication_init` on page 468.

Communication process

- The master application initializes the required slave functions based on the CAN controller.
- The message register functions write all required values to the appropriate handle, e.g. (ds2211_canMsg). The appropriate request and read functions get the information from this handle later on.
- To perform a read operation, the master processor board requests that the previously registered slave function be carried out. The slave then performs the required functions independently and writes the results back to 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

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 copy data results from the buffer into the processor board application variables.

Function classes

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 be 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.

Error handling

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 handle the error code.

Communication channels and priorities

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). Like 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

Basics

Basics on the RTI CAN Blockset (RTI CAN Blockset Reference 🕮)
CAN Support (DS2211 Features 🕮)

CAN Error Message Types

Introduction

The functions of the CAN environment report error, warning, and information messages if a problem occurs. These messages are displayed by the **Message Viewer** of the experiment software. The message consists of an error number, the function name, the board index (offset of the PHS-bus address) and the message text. For example:

```
Error[121]: ds2211_can_channel_init (6,..) baudrate: too low
(min. 10 kBaud)!
```

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

Related topics

References

ds2211_can_channel_init.. 471

Data Structures for CAN

Introduction

The data structures provide information on channels, services, and messages to be used by other functions. Using CAN RTLib functions, you access the structures *automatically*. You do not have to access them explicitly in your application.

Where to go from here

Information in this section

Information in other sections

ds2211_canChannel

Purpose

The **ds2211_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;
} ds2211_canChannel;
```

Include file

`can2211.h`

Members

base The PHS-bus base address is provided by the function `ds2211_can_channel_init` or `ds2211_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 `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced`. This parameter is read-only.

btr0 Value of Bit Timing Register 0. This parameter is provided by the function `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced`. This parameter is read-only.

btr1 Value of Bit Timing Register 1. This parameter is provided by the function `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced`. This parameter is read-only.

frequency Frequency of the CAN controller. This parameter is provided by the function `ds2211_can_channel_init` or `ds2211_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
`DS2211_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2211_CAN_EXT`	29-bit extended format, CAN 2.0B

This parameter is provided by the function `ds2211_can_channel_init` or `ds2211_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 `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced`. This parameter is read-only.

Related topics

References

ds2211_can_channel_init...471
ds2211_can_channel_init_advanced...473

ds2211_canService

Purpose

The `ds2211_canService` structure contains information on the CAN service. The CAN service provides information on errors and status information (see the `type` parameter).

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;
} ds2211_canService;
```

Include file

`can2211.h`

Members

data0 Contains returned data from the function `ds2211_can_service_read`.

data1 Contains returned data from the function `ds2211_can_service_read`.

Note

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

module The CAN module is provided by the function `ds2211_can_service_register`. This parameter is read-only.

queue This parameter is provided by the function `ds2211_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 `ds2211_can_service_register` function. This parameter is read-only.

Note

Start the CAN channel with the enabled status interrupt to use the following predefined services (see `ds2211_can_channel_start` on page 476).

Predefined Symbol	Meaning
`DS2211_CAN_SERVICE_TX_OK`	Number of successfully sent TX/RM/RQTX messages
`DS2211_CAN_SERVICE_RX_OK`	Number of successfully received RX/RQRX messages
`DS2211_CAN_SERVICE_CRC_ERR`	Number of CRC errors
`DS2211_CAN_SERVICE_ACK_ERR`	Number of acknowledge errors
`DS2211_CAN_SERVICE_FORM_ERR`	Number of format errors
`DS2211_CAN_SERVICE_BIT1_ERR`	Number of Bit1 errors
`DS2211_CAN_SERVICE_BIT0_ERR`	Number of Bit0 errors
`DS2211_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 `ds2211_can_channel_start` on page 476).

Predefined Symbol	Meaning
`DS2211_CAN_SERVICE_RX_LOST`	Number of lost RX messages. The RX lost counter is incremented when a received message is overwritten in the receive mailbox before the message has been read.
`DS2211_CAN_SERVICE_DATA_LOST`	Number of data lost errors. The data lost counter is incremented when the data of a message is overwritten before the data has been written to the communication queue.
`DS2211_CAN_SERVICE_MAILBOX_ERR`	Number of mailbox errors. If a message to be sent cannot be assigned to a mailbox, the mailbox error counter is increased by one. For possible error reasons, see below.
`DS2211_CAN_SERVICE_BUSSTATUS`	Status of the CAN controller. For the predefined values, see below.
`DS2211_CAN_SERVICE_STDMASK`	Status of the global standard mask register

Predefined Symbol	Meaning
DS2211_CAN_SERVICE_EXTMASK	Status of the global extended mask register
DS2211_CAN_SERVICE_MSG_MASK15	Status of the message 15 mask register
DS2211_CAN_SERVICE_TXQUEUE_ OVERFLOW_COUNT	Overflow counter of the transmit queue. The overflow counter (STD or XTD message format) is incremented when the queue is filled (64 messages) and a new message arrives. Depending on the `overrun_policy` parameter set with `ds2211_can_msg_txqueue_init`, the new message overwrites the oldest message entry or is ignored. The overflow counters are 16-bit counters. The wraparound occurs after 65535 overflows.
DS2211_CAN_SERVICE_VERSION	Version number of the CAN firmware.

index Table index already allocated by the register function `ds2211_can_service_register`. This parameter is read-only.

Parameter type

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

DS2211_CAN_SERVICE_MAILBOX_ERR Provides the number of mailbox errors. The following table describes possible error reasons and how to you can avoid these errors:

Error reason	Description	Workaround
All mailboxes are filled.	The messages are not removed from a mailbox fast enough.	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.	Try the first element of the following list. If the error counter still increases, try the next one: ▪ Decrease the timeout value for messages with the same format as mailbox 14 – i.e., with the opposite format of mailbox 15 (refer to `ds2211_can_channel_init`). ▪ Initialize the `mb15_format` parameter with the other format when calling `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced`. ▪ Choose different message IDs for messages of mailbox 14 format. ▪ Do not use standard and extended messages on one CAN channel simultaneously.

DS2211_CAN_SERVICE_BUSSTATUS Provides bus status information; the following symbols are predefined:

Predefined Symbol	Meaning
DS2211_CAN_BUSOFF_STATE	The CAN channel disconnects itself from the CAN bus. Use `ds2211_can_channel_BOff_return` to recover from the bus off state.
DS2211_CAN_WARN_STATE	The CAN controller is still active. The CAN controller recovers from this state automatically.
DS2211_CAN_ACTIVE_STATE	The CAN controller is active.

Note

After calling **`ds2211_can_channel_BOff_return`**, the service DS2211_CAN_SERVICE_BUSSTATUS will not return DS2211_CAN_BUSOFF_STATE.

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

```
ds2211_canService* service;
UInt16 overflow;
...
service = ds2211_can_service_register(
            txCh, DS2211_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT);
...
ds2211_can_service_request( service );
ds2211_can_service_read( service );
overflow = service->txqueue_overflowcnt_std;
```

Related topics

References

ds2211_can_channel_BOff_go	479
ds2211_can_channel_BOff_return	480
ds2211_can_channel_init	471
ds2211_can_channel_start	476
ds2211_can_msg_txqueue_init	512
ds2211_can_service_read	530
ds2211_can_service_register	527

ds2211_canMsg

Purpose

The **`ds2211_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;
   ds2211_canChannel*canChannel;
   ds2211_canService *msgService;
    } ds2211_canMsg;
```

Include file

`can2211.h`

Members

timestamp This parameter contains the following values:

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

This parameter is updated by the function `ds2211_can_msg_read` if the message was registered using the `inform` parameter `DS2211_CAN_TIMECOUNT_INFO`.

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

This parameter is updated by the function `ds2211_can_msg_read` if the message was registered with the `inform` parameter `DS2211_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 delay time between the update and the sending of a message is used. For acyclic sending, the delay time between the trigger and the successful sending of a message is used. The valid range is 0.0 ... 100.0 seconds.

This parameter is updated by the function `ds2211_can_msg_read` if the message was registered with the `inform` parameter `DS2211_CAN_DELAYCOUNT_INFO`.

processed Processed flag of the message. This parameter is updated by the function `ds2211_can_msg_read`. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2211_CAN_PROCESSED`	The message has been sent/received since the last execution call.
`DS2211_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 bytes. This parameter is updated by the function `ds2211_can_msg_read` if the message was registered with the `inform` parameter `DS2211_CAN_DATA_INFO`.

data[8] Buffer for CAN message data. This data is updated by the function `ds2211_can_msg_read` if the message was registered with the `inform` parameter `DS2211_CAN_DATA_INFO`.

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

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

Predefined Symbol	Meaning
`DS2211_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2211_CAN_EXT`	29-bit extended format, CAN 2.0B

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
`DS2211_CAN_TX`	Transmit message registered by `ds2211_can_msg_tx_register`
`DS2211_CAN_RX`	Receive message registered by `ds2211_can_msg_rx_register`
`DS2211_CAN_RM`	Remote message registered by `ds2211_can_msg_rm_register`
`DS2211_CAN_RQTX`	RQTX message registered by `ds2211_can_msg_rqtx_register`
`DS2211_CAN_RQRX`	RQRX message registered by `ds2211_can_msg_rqrx_register`

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

inform Specifies the kind of information returned by the function `ds2211_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
`DS2211_CAN_NO_INFO`	Returns no information.
`DS2211_CAN_DATA_INFO`	Updates the data and datalen parameters (needed for receive and request (RQRX) messages).
`DS2211_CAN_MSG_INFO`	Updates the message identifier and the message format for RM, RQ, TX, and RX messages.
`DS2211_CAN_TIMECOUNT_INFO`	Updates the timestamp and the deltatime parameters.
`DS2211_CAN_DELAYCOUNT_INFO`	Updates the delaytime parameter.

Note

If you modify the `inform` parameter after the message was registered, 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 `ds2211_canChannel` structure where the message object is installed. This parameter is read-only. Refer to `ds2211_canChannel` on page 459.

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

Initialization

Introduction

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

ds2211_can_communication_init

Syntax

```
void ds2211_can_communication_init(
      const UInt32 base,
      const UInt32 bufferwarn)
```

Include file

`can2211.h`

Purpose

To initialize communication between the master and the slave DS2211.

Description

This function also initializes seven communication channels with fixed queues (0 … 6) for the master-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 is aborted.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

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

Predefined Symbol	Meaning
`DS2211_CAN_INT_DISABLE`	The bufferwarn subinterrupt is disabled.
`DS2211_CAN_INT_ENABLE`	The bufferwarn subinterrupt is enabled.

Return value

None

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2211_can_communication_init(x,..) memory: allocation error on master	Memory allocation error. No free memory on the master.
104	Error	ds2211_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	ds2211_can_communication_init(x,..) subint: init failed by master	Master subinterrupt initialization failed. There is not enough memory available.
106	Error	ds2211_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	ds2211_can_communication_init(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2211_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 `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` when registering messages or services.
109	Error	ds2211_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	ds2211_can_communication_init(x,..) slave: not connected to HwInterrupt	There may be a hardware failure or the initialization process is not correct.

Example

For examples, refer to:

- Example of Handling Transmit and Receive Messages on page 536
- Example of Handling Request and Remote Messages on page 538
- Example of Using Subinterrupts on page 540

Related topics

References

ds2211_can_channel_all_sleep 477
ds2211_can_channel_init 471
ds2211_can_msg_sleep 517

CAN Channel Handling

Introduction

Provides information on handling CAN interfaces, called *CAN channels*.

Where to go from here

Information in this section

ds2211_can_channel_init

Syntax

```
ds2211_canChannel* ds2211_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

`can2211.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**
>
> You have to call the `ds2211_can_channel_start` function to complete the CAN channel initialization.

Description

If no error occurs, `ds2211_can_channel_init` returns a pointer to the `ds2211_canChannel` structure.

If an interrupt is to 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 DS2211 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
`DS2211_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2211_CAN_EXT`	29-bit extended format, CAN 2.0B

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_SUBINT`	No interrupt for bus off

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

Predefined Symbol	Meaning
`DS2211_CAN_TERMINATION_ON`	Bus termination activated
`DS2211_CAN_TERMINATION_OFF`	Bus termination deactivated

Return value

canChannel Pointer to the `ds2211_canChannel`

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2211_can_channel_init_advanced(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
104	Error	ds2211_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 DS2211_CAN_NO_ERROR.
106	Error	ds2211_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	ds2211_can_channel_init_advanced(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2211_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 to a filled queue. To prevent this error, deactivate all messages with `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` when registering messages or services.
123	Error	ds2211_can_channel_init_advanced(x,..) channel: use range 0..1!	Use a CAN channel within the range of 0 … 1.
124	Error		The clock frequency of the CAN clock generator limited by is too low.
140	Error	ds2211_can_channel_init_advanced(x,..) format: wrong format.	Only the symbols `DS2211_CAN_STD` and `DS2211_CAN_EXT` are allowed for the parameter mb15_format.
141	Error	ds2211_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	250	ds2211_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.

Example

For examples, refer to:

- Example of Handling Transmit and Receive Messages on page 536
- Example of Handling Request and Remote Messages on page 538
- Example of Using Subinterrupts on page 540

Related topics

References

ds2211_can_channel_all_sleep	477
ds2211_can_channel_all_wakeup	478
ds2211_can_channel_BOff_go	479
ds2211_can_channel_BOff_return	480
ds2211_can_channel_init_advanced	473
ds2211_can_channel_set	481
ds2211_can_channel_start	476
ds2211_can_msg_rm_register	500
ds2211_can_msg_rqrx_register	497
ds2211_can_msg_rqtx_register	493
ds2211_can_msg_rx_register	490
ds2211_can_msg_sleep	517
ds2211_can_msg_tx_register	486
ds2211_can_service_register	527
ds2211_canChannel	459

ds2211_can_channel_init_advanced

Syntax

```
ds2211_canChannel* ds2211_can_channel_init_advanced(
      const UInt32 mb15_format,
      const Int32 busoff_subinterrupt,
      const UInt32 termination);
```

Include file

`can2211.h`

Purpose

To perform the initialization of a CAN channel with parameters.

If no error occurs, the function returns a pointer to the `ds2211_canChannel` structure.

Note

You have to call `ds2211_can_channel_start` to complete the CAN channel initialization.

Description

Use the returned handle when calling one of the following functions: `ds2211_can_channel_start`, `ds2211_can_channel_all_sleep`, `ds2211_can_channel_all_wakeup`, `ds2211_can_channel_BOff_go`, `ds2211_can_channel_BOff_return`, `ds2211_can_channel_set`, `ds2211_can_msg_tx_register`, `ds2211_can_msg_rx_register`, `ds2211_can_msg_rqtx_register`, `ds2211_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 `ds2211_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 DS2211 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
`DS2211_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2211_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
`DS2211_CAN_NO_SUBINT`	No interrupt for bus off

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

Predefined Symbol	Meaning
`DS2211_CAN_TERMINATION_ON`	Bus termination activated
`DS2211_CAN_TERMINATION_OFF`	Bus termination deactivated

Return value

canChannel Specifies the pointer to the `ds2211_canChannel` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2211_can_channel_init_advanced(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
104	Error	ds2211_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 DS2211_CAN_NO_ERROR.
106	Error	ds2211_can_channel_init_advanced(x,..) slave: not responding	The slave did not finish the initialization of the communication in the time DSCOMDEF_TIMEOUT (in seconds).
107	Error	ds2211_can_channel_init_advanced(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2211_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 **`ds2211_can_msg_sleep`** or **`ds2211_can_channel_all_sleep`** when registering messages or services.
123	Error	ds2211_can_channel_init_advanced(x,..) channel: use range 0..1	Use a CAN channel within the range of 0 … 1.
140	Error	ds2211_can_channel_init_advanced(x,..) format: wrong format	Only the `DS2211_CAN_STD` and `DS2211_CAN_EXT` symbols are allowed for the mb15_format parameter.
141	Error	ds2211_can_channel_init_advanced(x,..) subint: use range 0..14	The subinterrupt number must be within the range of 0 … 14.

Example

```
ds2211_canChannel* CH;
CH = ds2211_can_channel_init_advanced(
   DS2211_1_BASE,      /* PHS-bus base address  */
   0,                  /* channel 0 */
   0x80,               /* BTR0      */
   0x6F,               /* BTR1*     */
   DS2211_CAN_STD,     /* use mailbox 15 to receive only    */
                       /* CAN messages with standard format */
   DS2211_CAN_NO_SUBINT, /* generate no subinterrupt when  */
                       /* the CAN controller goes in the    */
                       /* bus off state                     */

   DS2211_CAN_TERMINATION_ON   /* Bus termination activated */
);
```

Related topics

References

ds2211_can_channel_all_sleep....477
ds2211_can_channel_init....471
ds2211_can_channel_start....476
ds2211_can_communication_init....468
ds2211_can_msg_sleep....517

ds2211_can_channel_start

Syntax

```
void ds2211_can_channel_start(
      const ds2211_canChannel* canCh,
      const UInt32 status_int);
```

Include file

`can2211.h`

Purpose

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

Description

The CAN channel will change to the bus on state and the DS2211 slave interrupts will be enabled. Use the returned handle from the function `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

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

Predefined Symbol	Meaning
`DS2211_CAN_INT_DISABLE`	No status interrupt will be generated.
`DS2211_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 occurs on each successful reception or transmission on the CAN bus, regardless of whether the DS2211 slave has configured a message object to receive that particular message identifier. This interrupt is useful to detect bus errors caused by physical layer issues, such as noise. In most applications, it is recommended to not set this bit. Because this interrupt occurs for each message, the DS2211 would be unnecessarily burdened.

Return value

None

Messages

The following messages are defined:

ID	Type	Message	Description
104	Error	ds2211_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.

Example

For examples, refer to:

- Example of Handling Transmit and Receive Messages on page 536
- Example of Handling Request and Remote Messages on page 538
- Example of Using Subinterrupts on page 540

Related topics

References

ds2211_can_channel_init..471
ds2211_can_channel_init_advanced...473

ds2211_can_channel_all_sleep

Syntax

```
Int32 ds2211_can_channel_all_sleep(
      const ds2211_canChannel* canCh);
```

Include file

`can2211.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 to sleep mode until they are reactivated by `ds2211_can_channel_all_wakeup`.

Use the returned handle from the `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` function to call this function.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
DS2211_CAN_NO_ERROR	The function was performed without error.
DS2211_CAN_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns DS2211_CAN_NO_ERROR.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

```
ds2211_can_channel_all_sleep(canCh);
```

Related topics

References

ds2211_can_channel_all_wakeup

Syntax

```
Int32 ds2211_can_channel_all_wakeup(
      const ds2211_canChannel* canCh);
```

Include file

`can2211.h`

Purpose

To reactivate all messages that were deactivated by calling the functions `ds2211_can_channel_all_sleep` and `ds2211_can_msg_sleep`.

Description

Use the returned handle from the function `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
DS2211_CAN_NO_ERROR	The function has been performed without error.
DS2211_CAN_BUFFER_OVERFLOW	The communication buffer occurred. Repeat the function until it returns DS2211_CAN_NO_ERROR.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

```
ds2211_can_channel_all_wakeup(canCh);
```

Related topics

References

ds2211_can_channel_all_sleep..477
ds2211_can_msg_sleep..517

ds2211_can_channel_BOff_go

Syntax

```
Int32 ds2211_can_channel_BOff_go(
      const ds2211_canChannel* canCh);
```

Include file

`can2211.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 `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
DS2211_CAN_NO_ERROR	The function was performed without error.
DS2211_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 DS2211_CAN_NO_ERROR.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

```
ds2211_can_channel_BOff_go(canCh);
```

Related topics

References

ds2211_can_channel_BOff_return 480
ds2211_can_channel_init 471
ds2211_can_channel_init_advanced 473
ds2211_can_communication_init 468

ds2211_can_channel_BOff_return

Syntax

```
Int32 ds2211_can_channel_BOff_return(
      const ds2211_canChannel* canCh);
```

Include file

`can2211.h`

Purpose

To reset the slave DS2211 CAN channel from the bus off state.

Use the returned handle from the function `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
DS2211_CAN_NO_ERROR	The function was performed without error.
DS2211_CAN_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns DS2211_CAN_NO_ERROR.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

```
ds2211_can_channel_BOff_return(canCh);
```

Related topics

References

ds2211_can_channel_BOff_go....................479
ds2211_can_channel_init....................471
ds2211_can_channel_init_advanced....................473

ds2211_can_channel_set

Syntax

```
Int32 ds2211_can_channel_set(
      const ds2211_canChannel* canCh,
      const UInt32 mask_type,
      const UInt32 mask_value);
```

Include file

`can2211.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 `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

mask_type Specifies the mask type. The following symbols are predefined:

Predefined Symbol	Meaning
DS2211_CAN_CHANNEL_SET_MASK15	Sets the Message 15 Mask Register.
DS2211_CAN_CHANNEL_SET_ARBMASK15	Sets the Arbitration Register for mailbox 15.
DS2211_CAN_CHANNEL_SET_TERMINATION	Set the termination resistor for the channel.
DS2211_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
DS2211_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.
DS2211_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.
DS2211_CAN_CHANNEL_SET_TERMINATION	Use one of the following symbols to set the termination resistor: `DS2211_CHANNEL_TERMINATION_ON` or `DS2211_CHANNEL_TERMINATION_OFF`
DS2211_CAN_CHANNEL_SET_BAUDRATE	Sets the baud rate (in baud). Valid range: 10,000 … 1,000,000. Some baud rates in the allowed range cannot be met. If the actual baud rate differs from the one you specify by more than 1%, 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 527.

For further information on the registers, refer to the manual of the CAN controller.

Return value

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

Predefined Symbol	Meaning
DS2211_CAN_NO_ERROR	The function was performed without error.
DS2211_CAN_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns DS2211_CAN_NO_ERROR.
DS2211_CAN_BAUDRATE_L_ERROR	The baud rate is too low. The operation is aborted.
DS2211_CAN_BAUDRATE_H_ERROR	The baud rate is too high. The operation is aborted.
DS2211_CAN_BAUDRATE_SET_BAUDR_ERROR	Error during the calculation of the new bit timing parameters. The operation is aborted.

Messages

The following messages are defined:

Type	Message	Description
Warning	CAN2211 (0x y,...): baudrate on channel ... doesn't match the desired baudrate. New baudrate = ... bit/s (y: board index)	The actual baud rate differs from the one you specified by more than 1%.

Example

```
ds2211_can_channel_set(
        canCh,
        DS2211_CAN_CHANNEL_SET_MASK15,
        0xFFFFFFFE);
/* Set the lowest bit of the Message 15 Mask Register */
/* to “don't care" */
```

Related topics

References

ds2211_can_channel_txqueue_clear

Syntax

```
Int32 ds2211_can_channel_txqueue_clear(
     const ds2211_canChannel*  canCh);
```

Include file

`can2211.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 `ds2211_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_CAN_BUFFER_OVERFLOW`	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns DS2211_CAN_NO_ERROR.

CAN Message Handling

Introduction

To handle different kinds of CAN messages.

Where to go from here

Information in this section

ds2211_can_msg_tx_register

Syntax

```
ds2211_canMsg* ds2211_can_msg_tx_register(
      const ds2211_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

`can2211.h`

Purpose

To register a transmit message on the slave DS2211.

Description

If no error occurs, the function returns a pointer to the `ds2211_canMsg` structure.

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

- `ds2211_can_msg_write` to write new data to the message
- `ds2211_can_msg_read` to read the returned timestamps
- `ds2211_can_msg_send` to send the message with new data
- `ds2211_can_msg_trigger` to send the message
- `ds2211_can_msg_sleep` to deactivate the message
- `ds2211_can_msg_wakeup` to reactivate the message
- `ds2211_can_msg_clear` to clear the message object data
- `ds2211_can_msg_processed_register` to register the processed function
- `ds2211_can_msg_processed_request` to request the processed function
- `ds2211_can_msg_processed_read` to read the returned data

Note

You must call `ds2211_can_msg_write` to make the message valid for the CAN channel.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

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
`DS2211_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2211_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 OR operator. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2211_CAN_NO_INFO`	Returns no information.
DS2211_CAN_MSG_INFO	Updates the message identifier and the message format.

Predefined Symbol	Meaning
`DS2211_CAN_TIMECOUNT_INFO`	Updates the timestamp and the `deltatime` parameters.
`DS2211_CAN_DELAYCOUNT_INFO`	Updates the `delaytime` parameter.

subinterrupt Specifies the subinterrupt number for a received message. The valid range is 0 … 14.

> **Note**
>
> The interrupt number 15 is occupied by the buffer overflow warning interrupt (DS2211_CAN_SUBINT_BUFFERWARN). Do not use this number for any other interrupt.

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_SUBINT`	No interrupt for the TX message

start_time Specifies the point in time of the first sending after timer start. Enter the value in seconds within the range 0 … 420.

repetition_time Specifies the time interval for repeating the message automatically. Enter the value in seconds within the range 0 ... 100.

Use the following predefined symbol to define a message sent only once with `ds2211_can_msg_trigger`:

Predefined Symbol	Meaning
`DS2211_CAN_TRIGGER_MSG`	Calls `ds2211_can_msg_trigger` to send the message.

timeout The message will occupy the mailbox only up to this point in time. When the threshold is exceeded, the message is released from the mailbox. Enter the value in seconds within the range 0 … 100.

Use the following predefined symbol to calculate the timeout value internally:

Predefined Symbol	Meaning
`DS2211_CAN_TIMEOUT_NORMAL`	The timeout value is calculated internally when registering the message. This timeout value works in most cases.

Return value

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2211_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	ds2211_can_msg_tx_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2211_can_msg_tx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2211_CAN_AUTO_INDEX.
104	Error	ds2211_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	ds2211_can_msg_tx_register(x,..) slave: not responding	The slave did not finish the initialization within one second.
107	Error	ds2211_can_msg_tx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2211_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 to a filled queue. To prevent this error, deactivate all messages with `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` when registering messages or services.
140	Error	ds2211_can_msg_tx_register(x,..) format: wrong format	Only the symbols `DS2211_CAN_STD` and `DS2211_CAN_EXT` are allowed for the parameter `format`.
141	Error	ds2211_can_msg_tx_register(x,..) subint: use range 0..14!	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2211_can_msg_tx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2211_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, refer to `ds2211_canService` on page 461.
144	Error	ds2211_can_msg_tx_register(x,..) Too much messages (max. 100)!	The total number of registered messages is limited to 100. The program is aborted.
145	Error	ds2211_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 is aborted.
146	Error	ds2211_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 is aborted.
147	Error	ds2211_can_msg_tx_register(x,..) rep. time: too low !	Must be at least CAN_FRAME_TIME. A lower value causes an error and the program is aborted. Note that CAN_FRAME_TIME = (136 / Baud rate).
148	Error	ds2211_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 is aborted.
149	Error	ds2211_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 is aborted. Note that CAN_FRAME_TIME = (136 / Baud rate).

ID	Type	Message	Description
152	Error	ds2211_can_msg_tx_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: ▪ You try to register a CAN message on an uninitialized CAN channel. ▪ You try to register a CAN service on an uninitialized CAN channel. Use `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` to initialize the CAN channel.

Example

For examples of how to use this function, refer to Example of Handling Transmit and Receive Messages on page 536 and Example of Using Subinterrupts on page 540.

Related topics

References

ds2211_can_channel_all_sleep	477
ds2211_can_channel_init	471
ds2211_can_channel_init_advanced	473
ds2211_can_msg_clear	522
ds2211_can_msg_processed_read	525
ds2211_can_msg_processed_register	523
ds2211_can_msg_processed_request	524
ds2211_can_msg_read	519
ds2211_can_msg_send	509
ds2211_can_msg_sleep	517
ds2211_can_msg_trigger	521
ds2211_can_msg_wakeup	518
ds2211_can_msg_write	507
ds2211_canMsg	464

ds2211_can_msg_rx_register

Syntax

```
ds2211_canMsg* ds2211_can_msg_rx_register(
      const ds2211_canChannel* canCh,
      const Int32 queue,
      const UInt32 identifier,
      const UInt32 format,
      const UInt32 inform,
      const Int32 subinterrupt);
```

Include file

`can2211.h`

Purpose

To register a receive message on the slave DS2211.

Description

If no error occurs, `ds2211_can_msg_rx_register` returns a pointer to the `ds2211_canMsg` structure.

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

- `ds2211_can_msg_read` to read the returned data and timestamps
- `ds2211_can_msg_sleep` to deactivate the message
- `ds2211_can_msg_wakeup` to reactivate the message
- `ds2211_can_msg_clear` to clear the message data
- `ds2211_can_msg_processed_register` to register the processed function
- `ds2211_can_msg_processed_request` to request the processed function
- `ds2211_can_msg_processed_read` to read the returned data

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

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
`DS2211_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2211_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 OR operator. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2211_CAN_NO_INFO`	Returns no information.
`DS2211_CAN_DATA_INFO`	Updates the data and datalen parameters (needed for receive and request (RQRX) messages).
`DS2211_CAN_MSG_INFO`	Updates the message identifier and the message format.
`DS2211_CAN_TIMECOUNT_INFO`	Updates the timestamp and deltatime parameters.

subinterrupt Specifies the subinterrupt number for a received message. The valid range is 0 … 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (DS2211_CAN_SUBINT_BUFFERWARN). Do not use this number for any other interrupt.

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_SUBINT`	No interrupt for the receive message

Return value

canMsg This function returns the pointer to the `ds2211_canMsg` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2211_can_msg_rx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2211_can_msg_rx_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2211_can_msg_rx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2211_CAN_AUTO_INDEX.
104	Error	ds2211_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	ds2211_can_msg_rx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2211_can_msg_rx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2211_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 to a filled queue. To prevent this error, deactivate all messages with `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` when registering messages or services.
140	Error	ds2211_can_msg_rx_register(x,..) format: wrong format	Only the symbols `DS2211_CAN_STD` and `DS2211_CAN_EXT` are allowed for the parameter `format`.
141	Error	ds2211_can_msg_rx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2211_can_msg_rx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2211_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 `ds2211_canService` on page 461.
144	Error	ds2211_can_msg_rx_register(x,..): Too much messages (max. 100)!	The total number of registered messages is limited to 100. The program is aborted.
152	Error	ds2211_can_msg_rx_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: ▪ You try to register a CAN message on an uninitialized CAN channel.

ID	Type	Message	Description
			▪ You try to register a CAN service on an uninitialized CAN channel. Use `ds2211_can_channel_init` to initialize the CAN channel.

Example

For examples of how to use this function, refer to Example of Handling Transmit and Receive Messages on page 536 and Example of Using Subinterrupts on page 540.

```
ds2211_canMsg* rxMsg = ds2211_can_msg_rx_register(
         canCh,
         0,
         0x123,
         DS2211_CAN_STD,
         DS2211_CAN_DATA_INFO,
         DS2211_CAN_NO_SUBINT);
```

Related topics

References

ds2211_can_channel_init	471
ds2211_can_channel_init_advanced	473
ds2211_can_msg_clear	522
ds2211_can_msg_processed_read	525
ds2211_can_msg_processed_register	523
ds2211_can_msg_processed_request	524
ds2211_can_msg_read	519
ds2211_can_msg_send	509
ds2211_can_msg_sleep	517
ds2211_can_msg_trigger	521
ds2211_can_msg_wakeup	518
ds2211_can_msg_write	507

ds2211_can_msg_rqtx_register

Syntax

```
ds2211_canMsg* ds2211_can_msg_rqtx_register(
      const ds2211_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

`can2211.h`

Purpose

To register a request transmission (RQTX) message on the slave DS2211.

Description

Use this function to register a request message. Use the function `ds2211_can_msg_rqtx_register` to register a function that receives the requested data. If no error occurs, `ds2211_can_msg_rqtx_register` returns a pointer to the `ds2211_canMsg` structure.

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

Function	Description
`ds2211_can_msg_rqtx_activate`	to activate the message
`ds2211_can_msg_read`	To read the returned time stamps.
`ds2211_can_msg_sleep`	To deactivate the message.
`ds2211_can_msg_wakeup`	To reactivate the message.
`ds2211_can_msg_trigger`	To send the request message.
`ds2211_can_msg_clear`	To clear the message object data.
`ds2211_can_msg_processed_register`	To register the processed function.
`ds2211_can_msg_processed_request`	To request the processed function.
`ds2211_can_msg_processed_read`	To read the returned data.

Note

You must call `ds2211_can_msg_rqtx_activate` to make the message valid for the CAN channel.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

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
`DS2211_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2211_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 OR operator; the following symbols are predefined:

Predefined Symbol	Meaning
`DS2211_CAN_NO_INFO`	Returns no information.
`DS2211_CAN_MSG_INFO`	Updates the message identifier and the message format.
`DS2211_CAN_TIMECOUNT_INFO`	Updates the timestamp and deltatime parameters.
`DS2211_CAN_DELAYCOUNT_INFO`	Updates the delaytime parameter.

subinterrupt Specifies the subinterrupt number for a received message. The valid range is 0 … 14.

> **Note**
>
> The interrupt number 15 is occupied by the buffer overflow warning interrupt (DS2211_CAN_SUBINT_BUFFERWARN). Do not use this number for any other interrupt.

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_SUBINT`	No interrupt for the RQTX messages.

start_time Specifies the point in time of the first sending after timer start. Enter the value in seconds within the range 0 … 420.

repetition_time Specifies the time interval for repeating the message automatically. Enter the value in seconds within the range 0 ... 100.

Use the following predefined symbol to define a message sent only once with `ds2211_can_msg_trigger`:

Predefined Symbol	Meaning
`DS2211_CAN_TRIGGER_MSG`	Calls `ds2211_can_msg_trigger` to send the message.

timeout The message will occupy the mailbox only up to this point in time. When the threshold is exceeded, the message is released from the mailbox. Enter the value in seconds within the range 0 … 100.

Use the following predefined symbol to calculate the timeout value internally:

Predefined Symbol	Meaning
`DS2211_CAN_TIMEOUT_NORMAL`	The timeout value is calculated internally when registering the message. This timeout value works in most cases.

Return value

canMsg This function returns the pointer to the ds2211_canMsg structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2211_can_msg_rqtx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2211_can_msg_rqtx_register(x,..) queue: Illegal communication queue	There is no communication channel with this queue number.
103	Error	ds2211_can_msg_rqtx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2211_CAN_AUTO_INDEX.
104	Error	ds2211_can_msg_rqtx_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	ds2211_can_msg_rqtx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2211_can_msg_rqtx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2211_can_msg_rqtx_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 to a filled queue. To prevent this error, deactivate all messages with `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` when registering messages or services.
140	Error	ds2211_can_msg_rqtx_register(x,..) format: wrong format	Only the symbols DS2211_CAN_STD and DS2211_CAN_EXT are allowed for the parameter format.
141	Error	ds2211_can_msg_rqtx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2211_can_msg_rqtx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	can_tp1_msg_rqtx_registerds2211_can_msg_rqtx_register(x,..) id: Illegal id or id conflict	The CAN controller does not install the identifier specified in the program. For further information, refer to DS2211_CAN_SERVICE_MAILBOX_ERR.
144	Error	ds2211_can_msg_rqtx_register(x,..): Too much messages (max.100)!	The total number of registered messages is limited to 100. The program is aborted.
152	Error	ds2211_can_msg_rqtx_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: ▪ You try to register a CAN message on an uninitialized CAN channel. ▪ You try to register a CAN service on an uninitialized CAN channel.

ID	Type	Message	Description
			Use ds2211_can_channel_init or ds2211_can_channel_init_advanced to initialize the CAN channel.

Example

For examples of how to use this function, refer to Example of Handling Request and Remote Messages on page 538.

```
ds2211_canMsg* rqtxMsg = ds2211_can_msg_rqtx_register(
          canCh,
          0,
          0x123,
          DS2211_CAN_STD,
          DS2211_CAN_TIMECOUNT_INFO,
          DS2211_CAN_NO_SUBINT,
          1.5,
          0.3,
          DS2211_CAN_TIMEOUT_NORMAL);
```

Related topics

References

ds2211_can_msg_rqrx_register

Syntax

```
ds2211_canMsg* ds2211_can_msg_rqrx_register(
      const ds2211_canMsg* rqtxMsg,
      const UInt32 inform,
      const Int32 subinterrupt);
```

Include file

can2211.h

Purpose

To register an RQRX message on the slave DS2211.

Description

Use this message to receive the data requested with an RQTX message. If no error occurs, `ds2211_can_msg_rqrx_register` returns a pointer to the `ds2211_canMsg` structure.

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

- `ds2211_can_msg_read` to read the returned data and time stamps
- `ds2211_can_msg_sleep` to deactivate the message
- `ds2211_can_msg_wakeup` to reactivate the message
- `ds2211_can_msg_clear` to clear the message object data
- `ds2211_can_msg_processed_register` to register the processed function
- `ds2211_can_msg_processed_request` to request the processed function
- `ds2211_can_msg_processed_read` to read the returned data

Parameters

rqtxMsg Specifies the pointer to the related RQTX message.

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_INFO`	Returns no information.
`DS2211_CAN_DATA_INFO`	Updates the data and datalen parameters (needed for receive and request (RQRX) messages).
`DS2211_CAN_MSG_INFO`	Updates the message identifier and the message format.
`DS2211_CAN_TIMECOUNT_INFO`	Updates the timestamp and deltatime parameters.

subinterrupt Specifies the subinterrupt number for a received message. The valid range is 0 … 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (DS2211_CAN_SUBINT_BUFFERWARN). Do not use this number for any other interrupt.

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_SUBINT`	No interrupt for the RQRX message.

Return value

canMsg Specifies the pointer to the `DSxyz_canMsg` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2211_can_msg_rqrx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2211_can_msg_rqrx_register(x,..) queue: Illegal communication queue	There is no communication channel with this queue number.
103	Error	ds2211_can_msg_rqrx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2211_CAN_AUTO_INDEX.
104	Error	ds2211_can_msg_rqrx_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	ds2211_can_msg_rqrx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2211_can_msg_rqrx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2211_can_msg_rqrx_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 to a filled queue. To prevent this error, deactivate all messages with `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` when registering messages or services.
140	Error	ds2211_can_msg_rqrx_register(x,..) format: wrong format	Only the symbols `DS2211_CAN_STD` and `DS2211_CAN_EXT` are allowed for the parameter format.
141	Error	ds2211_can_msg_rqrx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2211_can_msg_rqrx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2211_can_msg_rqrx_register(x,..) id: Illegal id or id conflict	The CAN controller does not install the identifier specified in the program. For further information, see `ds2211_canService` on page 461.
144	Error	ds2211_can_msg_rqrx_register(x,..): Too much messages (max.100)!	The total number of registered messages is limited to 100. The program is aborted.
152	Error	ds2211_can_msg_rqrx_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: ▪ You try to register a CAN message on an uninitialized CAN channel. ▪ You try to register a CAN service on an uninitialized CAN channel. Use `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` to initialize the CAN channel.

Example

For examples of how to use this function, refer to Example of Handling Request and Remote Messages on page 538.

```
ds2211_canMsg* rqrxMsg = ds2211_can_msg_rqrx_register(
         rqtxMsg,
         DS2211_CAN_DATA_INFO,
         DS2211_CAN_NO_SUBINT);
```

Related topics

References

ds2211_can_channel_all_sleep....477
ds2211_can_channel_init....471
ds2211_can_channel_init_advanced....473
ds2211_can_msg_clear....522
ds2211_can_msg_processed_read....525
ds2211_can_msg_processed_register....523
ds2211_can_msg_processed_request....524
ds2211_can_msg_read....519
ds2211_can_msg_sleep....517
ds2211_can_msg_wakeup....518
ds2211_canMsg....464

ds2211_can_msg_rm_register

Syntax

```
ds2211_canMsg* ds2211_can_msg_rm_register(
      const ds2211_canChannel* canCh,
      const Int32 queue,
      const UInt32 identifier,
      const UInt32 format,
      const UInt32 inform,
      const Int32 subinterrupt);
```

Include file

`can2211.h`

Purpose

To register a remote message on the slave DS2211.

Description

If no error occurs, the function returns a pointer to the `ds2211_canMsg` structure.

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

- `ds2211_can_msg_write` to support the remote message with data
- `ds2211_can_msg_read` to read the returned time stamps
- `ds2211_can_msg_sleep` to deactivate the message

- `ds2211_can_msg_wakeup` to reactivate the message
- `ds2211_can_msg_clear` to clear the message object data
- `ds2211_can_msg_processed_register` to register the processed function
- `ds2211_can_msg_processed_request` to request the processed function
- `ds2211_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 an associated request message and carries the requested data.

Note

A remote message permanently occupies a mailbox on the slave DS2211 CAN channel. Therefore, only 10 remote messages are allowed within the same model for each CAN channel to ensure secure CAN operation. If this is not done, the function outputs an error and aborts the program.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

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
`DS2211_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2211_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 OR operator. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2211_CAN_NO_INFO`	Returns no information.
DS2211_CAN_MSG_INFO	Updates the message identifier and the message format.
`DS2211_CAN_TIMECOUNT_INFO`	Updates the timestamp and the `deltatime` parameters.
`DS2211_CAN_DELAYCOUNT_INFO`	Updates the `delaytime` parameter.

subinterrupt Specifies the subinterrupt number for a received message. The valid range is 0 … 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (DS2211_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
`DS2211_CAN_NO_SUBINT`	No interrupt for the RM message

Return value

canMsg This function returns the pointer to the `ds2211_canMsg` structure.

Messages

The following error and warning messages are defined:

ID	Type	Message	Description
101	Error	ds2211_can_msg_rm_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2211_can_msg_rm_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2211_can_msg_rm_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2211_CAN_AUTO_INDEX.
104	Error	ds2211_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	ds2211_can_msg_rm_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2211_can_msg_rm_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2211_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 to a filled queue. To prevent this error, deactivate all messages with `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` when registering messages or services.
140	Error	ds2211_can_msg_rm_register(x,..) format: wrong format	Only the symbols DS2211_CAN_STD and DS2211_CAN_EXT are allowed for the parameter format.
141	Error	ds2211_can_msg_rm_register(x,..) subint: use range 0...14.	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2211_can_msg_rm_register(x,...) subint: used for busoff.	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2211_can_msg_rm_register(x,..) id: illegal id or id conflict	The CAN controller does not install the identifier specified in the program. For further information, see `ds2211_canService` on page 461.
144	Error	ds2211_can_msg_rm_register(x,..) Too much messages (max. 100).	The total number of registered messages is limited to 100. The program is aborted.
150	Error	ds2211_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 ensure secure CAN operation. If this is not done, the function outputs an error and the program is aborted.

ID	Type	Message	Description
152	Error	ds2211_can_msg_rm_register(x,...) canCh: the CAN channel wasn't initialized	This message is displayed if: ▪ You try to register a CAN message on an uninitialized CAN channel. ▪ You try to register a CAN service on an uninitialized CAN channel. Use `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` to initialize the CAN channel.

Example

For examples of how to use this function, refer to Example of Handling Request and Remote Messages on page 538.

```
ds2211_canMsg* rmMsg = ds2211_can_msg_rm_register(
          canCh,
          0,
          0x123,
          DS2211_CAN_STD,
          DS2211_CAN_TIMECOUNT_INFO,
          DS2211_CAN_NO_SUBINT);
```

Related topics

References

ds2211_can_msg_set

Syntax

```
Int32 ds2211_can_msg_set(
      ds2211_canMsg* msg,
      const UInt32 type,
      const void* value );
```

Include file

`can2211.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 = DS2211_CAN_MSG_MASK),
- Set the send period for a TX or RQ message (type = DS2211_CAN_MSG_PERIOD),
- Set the identifier for a TX or RQ message (type = DS2211_CAN_MSG_ID) or
- Set the queue depth for a message (type = DS2211_CAN_MSG_QUEUE).
- Set the length for a message (type = DS2211_CAN_MSG_LEN).

Note

For DS2211_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 `ds2211_can_msg_set` for another message, the bitmask is removed from the first message.
- Using the bitmask might cause conflicts with messages installed for one message ID. In this case, message data is received via the message installed for this ID.
- You can skip the bitmask by 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
`DS2211_CAN_MSG_MASK`	To set the arbitrary mask for an RX message
`DS2211_CAN_MSG_PERIOD`	To set the send period for a TX or RQ message
`DS2211_CAN_MSG_ID`	To set the identifier for a TX or RQ message
`DS2211_CAN_MSG_QUEUE`	To set the queue depth for a message
`DS2211_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 DS2211_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
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_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 DS2211_CAN_NO_ERROR.
`DS2211_CAN_TYPE_ERROR`	The operation is not allowed for the specified message object.
`DS2211_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

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 ds2211_can_msg_set.

```
UInt32 mask = 0xFFFFFFF0; // Sets the last four bits to
                          // "Don’t Care".
ds2211_can_msg_set( msg, DS2211_CAN_MSG_MASK, &mask );
```

Example

This example shows how to receive different message IDs with one message via a bitmask:

- A message with ID 0x120 was registered. Now, you set the bitmask via `ds2211_can_msg_set(msg, DS2211_CAN_MSG_MASK,&mask);` with `mask = 0xFFFFFFF0`.

  This lets you receive the message IDs 0x120, 0x121, …, 0x12F.
- A message with ID 0x120 was registered. Now, you set the bitmask to 0x1FFFFFEF. This lets you receive the message IDs 0x120 and 0x130.

Example

This example shows how to apply the DS2211_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 = ds2211_can_msg_xx_register(...)
  ```

  By default, `myMsg` stores only one message.
- Define a message queue of length *n* for `myMsg`

  ```
  ds2211_can_msg_set(myMsg, DS2211_CAN_MSG_QUEUE, &n)
  ```

- Call `ds2211_can_msg_read(myMsg)` repeatedly until the function returns DS2211_CAN_NO_DATA.

```
UInt32 n;
canMsg = ds2211_can_msg_rx_register( canCh,…
n = 5000;
ds2211_can_msg_set(canMsg, DS2211_CAN_MSG_QUEUE, &n);
...
while(DS2211_CAN_NO_DATA != (error =
             ds2211_can_msg_read(canMsg)))
{
   if(DS2211_CAN_DATA_LOST == error)
   {
      /* error handling */
   }
   else if(DS2211_CAN_NO_ERROR == error)
   {
      /* process the message */
   }
   else /* DS2211_CAN_NO_DATA == error */
   {
      /* no further CAN-messages */
   }
}
```

Related topics

References

ds2211_can_msg_read.. 519

ds2211_can_msg_rqtx_activate

Syntax

```
Int32 ds2211_can_msg_rqtx_activate(
     const ds2211_canMsg* canMsg);
```

Include file

`can2211.h`

Purpose

To activate the request transmission message on the slave DS2211 registered by `ds2211_can_msg_rqtx_register`.

Description

This function does not send the message. Sending the message is done by the timer for cyclic sending or by calling `ds2211_can_msg_trigger` for acyclic sending. Use the returned handle from the function `ds2211_can_msg_rqtx_register` to call this function.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

Return value

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_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 DS2211_CAN_NO_ERROR.
`DS2211_CAN_TYPE_ERROR`	The operation is not allowed for the given message object.

Example

For examples of how to use this function, refer to Example of Handling Request and Remote Messages on page 538.

Related topics

References

ds2211_can_msg_rqtx_register 493
ds2211_can_msg_trigger 521

ds2211_can_msg_write

Syntax

```
Int32 ds2211_can_msg_write(
      const ds2211_canMsg* canMsg,
      const UInt32 datalen,
      const UInt32* data);
```

Include file

`can2211.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 is sent automatically with the specified value. A TX message registered by DS2211_CAN_TRIGGER_MSG is sent only when calling `ds2211_can_msg_trigger` or `ds2211_can_msg_send`.

Calling this function again updates 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 specified data and data length. The remote message is sent when a corresponding request message is received.

  Calling this function again updates CAN message data and data length.

Use the returned handle from the function `ds2211_can_msg_tx_register` or `ds2211_can_msg_rm_register` to call this function.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

datalen Specifies the length of the CAN message data. The 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
`DS2211_CAN_NO_ERROR`	The function has been performed without error.
`DS2211_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 DS2211_CAN_NO_ERROR.
`DS2211_CAN_TYPE_ERROR`	The operation is not allowed for the specified message object.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

For examples, refer to:

- Example of Handling Transmit and Receive Messages on page 536
- Example of Handling Request and Remote Messages on page 538
- Example of Using Subinterrupts on page 540

Related topics

References

ds2211_can_msg_rm_register	500
ds2211_can_msg_send	509
ds2211_can_msg_trigger	521
ds2211_can_msg_tx_register	486

ds2211_can_msg_send

Syntax

```
Int32 ds2211_can_msg_send(
      const ds2211_canMsg* canMsg,
      const UInt32 datalen,
      const UInt32* data,
      const Float32 delay);
```

Include file

`can2211.h`

Purpose

To write CAN message data and send the data immediately after the delay time. To send the transmit message with new data.

Description

The transmit message must have been registered by calling `ds2211_can_msg_tx_register`. Then `ds2211_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 `ds2211_can_msg_tx_register` to call this function.

Note

Suppose the `ds2211_can_msg_send` function is called twice. If the interval between the function calls is short, the second function call might 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 `ds2211_canMsg` structure.

datalen Specifies the length of the CAN message data. The valid range is 0 … 8 bytes.

data Specifies the buffer for CAN message data.

delay Sends the message after the delay time. The valid range is 0.0 … 100.0 seconds.

Return value

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

Predefined Symbol	Meaning
DS2211_CAN_NO_ERROR	The function was performed without error.
DS2211_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 DS2211_CAN_NO_ERROR.
DS2211_CAN_TYPE_ERROR	The operation is not allowed for the specified message object.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

```
UInt32 txData[8] = {1,2,3,4,5,6,7,8};
ds2211_can_msg_send (txMsg, 8, txData, 0.005);
```

Related topics

References

ds2211_can_msg_send_id...510
ds2211_can_msg_tx_register..486

ds2211_can_msg_send_id

Syntax

```
Int32 ds2211_can_msg_send_id (
      ds2211_canMsg* canMsg,
      const UInt32 id,
      const UInt32 datalen,
      const UInt8* data,
      const Float32 delay);
```

Include file

`can2211.h`

Purpose

To send a message with a modified identifier. This lets you send any message ID with one registered message.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

id Specifies the ID of the message to be modified.

datalen Specifies the length of the CAN message data. The valid range is 0 … 8 bytes.

data Specifies the buffer for CAN message data.

delay Sends the message after the delay time. The valid range is 0.0 … 100.0 seconds.

Return value

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_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 DS2211_CAN_NO_ERROR.
`DS2211_CAN_TYPE_ERROR`	The operation is not allowed for the specified message object.

Note

- The message format is determined by the format in which the message was installed when it was used for the first time.
- You have to use a handshake mechanism to send a message via `ds2211_can_msg_send_id` to make sure that a message installed for the message object has been sent already.
  Each message object is buffered only once on the slave. This might cause conflicts when you try to send several message objects with different IDs.

Example

The `ds2211_can_msg_send_id` function lets you send any message ID with one registered message.

```
ds2211_can_msg_send_id(msg, 0x123, data, 8, 0.001)
```

Related topics

References

ds2211_can_msg_queue_level.......................................512
ds2211_can_msg_send.......................................509
ds2211_can_msg_tx_register.......................................486

ds2211_can_msg_queue_level

Syntax

```
Int32 ds2211_can_msg_queue_level (
      ds2211_canMsg* canMsg);
```

Include file

`can2211.h`

Purpose

To return the number of messages stored in the message queue allocated on the master with `ds2211_can_msg_set(msg, DS2211_CAN_MSG_QUEUE, &size)`.

Description

Use `ds2211_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 `ds2211_canMsg` structure.

Return value

This function returns the number of messages in the message queue.

Related topics

References

ds2211_can_msg_read..519
ds2211_can_msg_set..503

ds2211_can_msg_txqueue_init

Syntax

```
Int32 ds2211_can_msg_txqueue_init(
      ds2211_canMsg* canMsg,
      const UInt32 overrun_policy,
      Float32 delay);
```

Include file

`can2211.h`

Purpose

To initialize the transmit queue that is used to queue messages sent by the `ds2211_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 `ds2211_can_msg_send_id_queued` function are stored. The queue stores up to 64 message entries.

Parameter

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

overrun_policy Selects the overrun policy of the transmit queue. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2211_CAN_TXQUEUE_OVERRUN_OVERWRITE`	The oldest message is overwritten.
`DS2211_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 might 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) can be used for queuing for every channel. Nevertheless `ds2211_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
`DS2211_CAN_NO_ERROR`	The transmit queue was initialized successfully.
`DS2211_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 DS2211_CAN_NO_ERROR.

Predefined Symbol	Meaning
DS2211_CAN_TXQUEUE_INIT_NOT_REG_ERROR	The message (canMsg) was not registered. The operation is aborted.
DS2211_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	ds2211_can_msg_txqueue_init(): TX message is not registered	The message was not registered successfully.
155	Error	ds2211_can_msg_txqueue_init(): not a TX message	The specified message is not a TX message.
301	Warning	ds2211_can_msg_txqueue_init(): delay time: too high (max. 10 s). Set to maximum.	The delay time must be within the range 0 … 10 s.

Example

The following example shows you how to initialize a TX queue.

```
void main()
{
  ds2211_canMsg* txMsg;
...
  txMsg = ds2211_can_msg_tx_register( txCh,
                    2, 0x1, DS2211_CAN_STD,
                    DS2211_CAN_TIMECOUNT_INFO |
                    DS2211_CAN_MSG_INFO,
                    1, 0.0,
                    DS2211_CAN_TRIGGER_MSG, 0 );
  ds2211_can_msg_txqueue_init (
          txMsg, DS2211_CAN_TXQUEUE_OVERRUN_OVERWRITE, 0.01);
...
}
```

Related topics

References

ds2211_can_msg_send_id_queued..515

ds2211_can_msg_send_id_queued

Syntax

```
Int32 ds2211_can_msg_send_id_queued(
      ds2211_canMsg* canMsg,
      const UInt32 id,
      const UInt32 data_len,
      const UInt32* data);
```

Include file

can2211.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 ds2211_can_msg_txqueue_init on page 512.

The DS2211_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT service allows the overflow counter of the transmit queue to be requested to check whether an overflow occurred.

Parameter

canMsg Specifies the pointer to the ds2211_canMsg structure.

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.

data_len Specifies the length of data within the range 0 … 8.

data Specifies the message data.

Return value

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

Predefined Symbol	Meaning
DS2211_CAN_NO_ERROR	The transmit queue was initialized successfully.
DS2211_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 DS2211_CAN_NO_ERROR.
DS2211_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	ds2211_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()
{
  ds2211_canMsg* txMsg;
  UInt32 txMsgData[8];
  ...
  txMsg = ds2211_can_msg_tx_register( txCh,
                    2, 0x1, DS2211_CAN_STD,
                    DS2211_CAN_TIMECOUNT_INFO |
                    DS2211_CAN_MSG_INFO,
                    1, 0.0,
                    DS2211_CAN_TRIGGER_MSG, 0 );
  /* initialize a transmit queue with delay = 0.01 s */
  ds2211_can_msg_txqueue_init (
           txMsg, DS2211_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;
  ds2211_can_msg_send_id_queued(txMsg, 0x12, 1, txMsgData);
  txMsgData[0] = 0x02;
  ds2211_can_msg_send_id_queued(txMsg, 0x13, 1, txMsgData);
  txMsgData[0] = 0x03;
  ds2211_can_msg_send_id_queued(txMsg, 0x14, 1, txMsgData);
...
}
```

Related topics

References

ds2211_can_msg_txqueue_init 512

ds2211_can_msg_txqueue_level_read

Syntax

```
UInt32 ds2211_can_msg_txqueue_level_read(
      const ds2211_canMsg* canMsg);
```

Include file

`can2211.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 pointer to the `ds2211_canMsg` structure.

Return value

Level of TX-queue The number of TX messages in the transmit queue on the CAN slave (0 … 64).

ds2211_can_msg_sleep

Syntax

```
Int32 ds2211_can_msg_sleep(
        const ds2211_canMsg* canMsg);
```

Include file

`can2211.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 sleep mode until it is reactivated by calling `ds2211_can_msg_wakeup` or `ds2211_can_channel_all_wakeup`.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

Return value

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_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 DS2211_CAN_NO_ERROR.
`DS2211_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 551.

Example

```
ds2211_can_msg_sleep(txMsg);
```

Related topics

References

ds2211_can_channel_all_sleep.......477
ds2211_can_channel_all_wakeup.......478
ds2211_can_msg_wakeup.......518

ds2211_can_msg_wakeup

Syntax

```
Int32 ds2211_can_msg_wakeup(
      const ds2211_canMsg* canMsg);
```

Include file

`can2211.h`

Purpose

To reactivate a message that has been deactivated by calling the `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` function.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

Return value

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_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 DS2211_CAN_NO_ERROR.
`DS2211_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 551.

Example

```
ds2211_can_msg_wakeup(txMsg);
```

Related topics

References

ds2211_can_channel_all_sleep........477
ds2211_can_channel_all_wakeup........478
ds2211_can_msg_sleep........517

ds2211_can_msg_read

Syntax

```
Int32 ds2211_can_msg_read(
      ds2211_canMsg* canMsg);
```

Include file

`can2211.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 `ds2211_can_msg_set` on page 503). By default, only one message can be received.

Use the function `ds2211_can_msg_clear` to clear the message data and time stamps. This is useful for simulation start/stop transitions.

Note

The status information that is returned depends on the previously specified inform parameter in the register function that corresponds to the message.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

Parameter	Meaning
data	Buffer with the updated data
datalen	Data length of the message
deltatime	Delta time of the message
timestamp	Time stamp 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
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_CAN_NO_DATA`	No data was updated.
`DS2211_CAN_DATA_LOST`	The input data of a previous request for the specified function was overwritten.

Example

For examples, refer to:

- Example of Handling Transmit and Receive Messages on page 536
- Example of Handling Request and Remote Messages on page 538
- Example of Using Subinterrupts on page 540

Related topics

References

ds2211_can_msg_clear.. 522
ds2211_can_msg_set.. 503

ds2211_can_msg_trigger

Syntax

```
Int32 ds2211_can_msg_trigger(
      const ds2211_canMsg* canMsg,
      const Float32 delay);
```

Include file

`can2211.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 `ds2211_can_msg_tx_register` or `ds2211_can_msg_rqtx_register` function to call this function.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

delay Sends the message after the delay time. The valid range is 0.0 … 100.0 seconds.

Return value

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

Predefined Symbol	Meaning
DS2211_CAN_NO_ERROR	The function was performed without error.
DS2211_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 DS2211_CAN_NO_ERROR.
DS2211_CAN_TYPE_ERROR	The operation is not allowed for the specified message object.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

```
ds2211_can_msg_trigger(txMsg, 0.005); /* 5 ms delay */
```

Related topics

References

ds2211_can_msg_rqtx_register..493
ds2211_can_msg_tx_register..486

ds2211_can_msg_clear

Syntax

```
void ds2211_can_msg_clear(
      ds2211_canMsg* canMsg);
```

Include file

`can2211.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, because any manipulation of these structure members would corrupt the message object.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

Return value

None

Function execution times

For information, refer to Function Execution Times on page 551.

Example

```
ds2211_can_msg_clear(rxMsg);
```

Related topics

References

ds2211_can_all_data_clear	534
ds2211_can_msg_rm_register	500
ds2211_can_msg_rqrx_register	497
ds2211_can_msg_rqtx_register	493
ds2211_can_msg_rx_register	490
ds2211_can_msg_tx_register	486

ds2211_can_msg_processed_register

Syntax

```
void ds2211_can_msg_processed_register(
        ds2211_canMsg* canMsg);
```

Include file

`can2211.h`

Purpose

To register the processed function in the command table.

Use `ds2211_can_msg_processed_read` to read the processed flag and time stamp without registering the message with the inform parameter `DS2211_CAN_TIMECOUNT_INFO`.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

Return value

None

Messages

The following error and warning messages are defined:

ID	Type	Description	Message
101	Error	ds2211_can_msg_processed_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2211_can_msg_processed_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2211_can_msg_processed_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2211_CAN_AUTO_INDEX.
104	Error	ds2211_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	ds2211_can_msg_processed_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2211_can_msg_processed_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	DS2211_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 to a filled queue. To prevent this error, deactivate all messages with `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` when registering messages or services.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

```
ds2211_can_msg_processed_register(rxMsg);
```

Related topics

References

ds2211_can_msg_processed_request

Syntax

```
Int32 ds2211_can_msg_processed_request(
      const ds2211_canMsg* canMsg);
```

Include file

`can2211.h`

Purpose

To request the message processed information from the slave DS2211.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

Return value

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

Predefined Symbol	Meaning
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_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 DS2211_CAN_NO_ERROR.
`DS2211_CAN_NO_DATA`	`ds2211_can_msg_processed_request` was called without registering the function with `ds2211_can_msg_processed_register` or an empty canMsg structure was handled.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

```
ds2211_can_msg_processed_request(rxMsg);
```

Related topics

References

ds2211_can_msg_processed_read

Syntax

```
Int32 ds2211_can_msg_processed_read(
      ds2211_canMsg* canMsg,
      double* timestamp,
      UInt32* processed);
```

Include file

`can2211.h`

Purpose

To read the message processed information from the slave DS2211.

Description

Prior to this, this information must have been requested by the master calling the function `ds2211_can_msg_processed_request` that demands the processed flag and the time stamp from the slave DS2211.

Parameters

canMsg Specifies the pointer to the `ds2211_canMsg` structure.

timestamp Specifies the time stamp when the message was last sent or received.

processed Specifies the processed flag of the message. The following symbols are predefined:

Symbols	Meaning
DS2211_CAN_PROCESSED	Message has been sent/received since the last execution call.
DS2211_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
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_CAN_NO_DATA`	No data was updated.
`DS2211_CAN_DATA_LOST`	The input data of a previous request for the specified function was overwritten.

Function execution times

For information, refer to Function Execution Times on page 551.

Related topics

References

ds2211_can_msg_processed_register.. 523
ds2211_can_msg_processed_request.. 524

CAN Service Functions

Introduction

To get information on errors and status information.

Where to go from here

Information in this section

ds2211_can_service_register

Syntax

```
ds2211_canService* ds2211_can_service_register(
     const ds2211_canChannel* canCh,
     const UInt32 service_type);
```

Include file

`can2211.h`

Purpose

To register the service function.

Description

Use `ds2211_can_service_read` to read a registered service specified by the `service_type` parameter.

Parameters

canCh Specifies the pointer to the `ds2211_canChannel` structure.

service_type Specifies the service to be installed. For additional information, see the `type` parameter of `ds2211_canService` structure. You can use the bitwise OR operator to combine several services.

Return value

canService This function returns the pointer to the `ds2211_canService` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2211_can_service_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2211_can_service_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2211_can_service_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2211_CAN_AUTO_INDEX.
104	Error	ds2211_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	ds2211_can_service_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2211_can_service_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2211_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 to a filled queue. To prevent this error deactivate all messages with `ds2211_can_msg_sleep` or `ds2211_can_channel_all_sleep` when registering messages or services.
152	Error	ds2211_can_service_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: ▪ 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 `ds2211_can_channel_start`. Use `ds2211_can_channel_init` or `ds2211_can_channel_init_advanced` to initialize the CAN channel.

Example

For a detailed example of how to use this function, refer to Example of Using Service Functions on page 542.

```
ds2211_canService* service;
...
  service = ds2211_can_service_register(txCh,
               DS2211_CAN_SERVICE_TX_OK |
               DS2211_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT );
```

Related topics

References

ds2211_can_channel_all_sleep 477
ds2211_can_channel_init 471
ds2211_can_msg_sleep 517

ds2211_can_service_request

Syntax

```
Int32 ds2211_can_service_request(
      const ds2211_canService* service);
```

Include file

can2211.h

Purpose

To request the service information from the slave DS2211. Use ds2211_can_service_read to read the registered service.

Description

Use the returned handle from the function ds2211_can_service_register on page 527 to call this function.

Parameters

service Specifies the pointer to the ds2211_canService structure.

Return value

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

Predefined Symbol	Meaning
DS2211_CAN_NO_ERROR	The function was performed without error.
DS2211_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 DS2211_CAN_NO_ERROR.
DS2211_CAN_NO_DATA	ds2211_can_service_request was called without registering the function with ds2211_can_service_register or an empty service structure was handled.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

For an example of how to use this function, refer to Example of Using Service Functions on page 542.

Related topics

References

ds2211_can_service_read

Syntax

```
Int32 ds2211_can_service_read(
         ds2211_canService* service);
```

Include file

`can2211.h`

Purpose

To read the service information from the slave DS2211.

Description

Prior to this, this information must have been requested by the master calling the `ds2211_can_service_request` function that asks for the service information from the slave DS2211.

Use the returned handle from the `ds2211_can_service_register` function.

Parameters

service Specifies the pointer to the updated `ds2211_canService` structure. 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.

Return value

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

Symbols	Meaning
`DS2211_CAN_NO_ERROR`	The function was performed without error.
`DS2211_CAN_NO_DATA`	No data was updated.
`DS2211_CAN_DATA_LOST`	The input data of a previous request for the specified function was overwritten.

Function execution times

For information, refer to Function Execution Times on page 551.

Example

For an example of how to use this function, refer to Example of Using Service Functions on page 542.

```
ds2211_canService* service;
...
service = ds2211_can_service_register(txCh,
            DS2211_CAN_SERVICE_TX_OK |
            DS2211_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT);
ds2211_can_service_request( service );
ds2211_can_service_read( service );

/* output */
txok = service->tx_ok;
queueoverflow = service->txqueue_overflowcnt_std;
```

Related topics

References

ds2211_can_service_register 527
ds2211_can_service_request 529
ds2211_canService 461

CAN Subinterrupt Handling

Where to go from here

Information in this section

Defining a Callback Function

Callback function

The callback function is a function that performs the action(s) that you define for a given subinterrupt. The callback function must be installed with the `ds2211_can_subint_handler_install` function.

Each time a CAN subinterrupt occurs, the subinterrupt handling then passes the information to the callback function.

Defining a callback function

Define your callback function as follows:

```
void can_callback_fcn(void* subint_data, Int32 subint);
```

with the parameters

subint_data Pointer to the board index of the related board within the range 0 … 15

subint Subinterrupt number within the range 0 … 14

Note

The last subinterrupt number to be generated is always "-1". This value indicates that there are no more pending subinterrupts.

Related topics

References

ds2211_can_subint_handler_install 533

ds2211_can_subint_handler_install

Syntax

```
ds2211_can_subint_handler_t  ds2211_can_subint_handler_install(
      const UInt32 base,
      const ds2211_can_subint_handler_t handler);
```

Include file

`can2211.h`

Purpose

To install a subinterrupt handler for all CAN interrupts.

Parameters

base Specifies the PHS-bus base address of the DS2211 board.

handler Specifies the pointer to your callback function.
For information on defining a callback function, refer to Defining a Callback Function on page 532.

Return value

This function returns the following value:

Symbol	Meaning
`ds2211_can_subint_handler_t`	Pointer to the previously installed callback function

Example

For an example of how to use this function, refer to Example of Using Subinterrupts on page 540.

Related topics

Basics

Defining a Callback Function... 532

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

ds2211_can_all_data_clear

Syntax

```
void ds2211_can_all_data_clear(const UInt32 base);
```

Include file

`can2211.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 DS2211 board.

Return value

None

Example

```
ds2211_can_all_data_clear(DS2211_1_BASE)
```

Related topics

References

ds2211_can_msg_clear......522

ds2211_can_error_read

Syntax

```
Int32 ds2211_can_error_read(
      const UInt32 base,
      const Int32 queue);
```

Include file

`can2211.h`

Purpose

To read the current error of the slave DS2211 from the dual-port memory.

Parameters

base Specifies the PHS-bus base address of the DS2211 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
DS2211_CAN_NO_ERROR	No error on the slave DS2211.
DS2211_CAN_SLAVE_ALLOC_ERROR	Memory allocation error on the slave DS2211. There are too many functions registered.
DS2211_CAN_SLAVE_BUFFER_OVERFLOW	Not enough memory space between the slave write pointer and the master read pointer.
DS2211_CAN_INIT_ACK	Acknowledge code. This is no error.
DS2211_CAN_SLAVE_UNDEF_ERROR	Undefined error. An error that cannot be assigned to one of the previous errors.

Example

```
#define QUEUE0   0
Int32 slave_error;
slave_error = ds2211_can_error_read(DS2211_1_BASE, QUEUE0);
/*     */
/* error handling */
/*     */
```

Examples of Using CAN

Introduction

Examples of how to use the CAN functions.

Where to go from here

Information in this section

Example of Handling Transmit and Receive Messages

Example

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 seconds. 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 sent to the message module.

```
#include <Brtenv.h>
#include <Ds2211.h>
#include <Can2211.h>
ds2211_canChannel* txCh;
ds2211_canChannel* rxCh;
ds2211_canMsg* txMsg;
ds2211_canMsg* rxMsg;
UInt32 txMsgData[8] = {1,2,3,4,5,6,7,8};
main()
{
  init(); /* initialize hardware system */
  ds2211_init(DS2211_1_BASE);
  (DS2211_1_BASE,
     DS2211_CAN_INT_DISABLE);
  txCh = ds2211_can_channel_init(DS2211_1_BASE, 0,
                  500000,
                  DS2211_CAN_STD,
                  DS2211_CAN_NO_SUBINT,
                  DS2211_CAN_TERMINATION_ON);
  rxCh = ds2211_can_channel_init(DS2211_1_BASE, 1,
                  500000,
                  DS2211_CAN_STD,
                  DS2211_CAN_NO_SUBINT,
                  DS2211_CAN_TERMINATION_ON);
  txMsg = ds2211_can_msg_tx_register(txCh,
                  2,
                  0x123,
                  DS2211_CAN_STD,
                  DS2211_CAN_TIMECOUNT_INFO,
                  DS2211_CAN_NO_SUBINT,
                  4.0,
                  1.0,
                  DS2211_CAN_TIMEOUT_NORMAL);
  rxMsg = ds2211_can_msg_rx_register(rxCh,
                    3,
                    0x123,
                    DS2211_CAN_STD,
                    DS2211_CAN_DATA_INFO | DS2211_CAN_TIMECOUNT_INFO,
                    DS2211_CAN_NO_SUBINT);
  ds2211_can_msg_write(txMsg, 8, txMsgData);
  ds2211_can_channel_start(rxCh, DS2211_CAN_INT_DISABLE);
  ds2211_can_channel_start(txCh, DS2211_CAN_INT_DISABLE);
  for(;;)
  {
     ds2211_can_msg_read(txMsg);
     if (txMsg->processed == DS2211_CAN_PROCESSED)
     {
        msg_info_printf(MSG_SM_RTLIB, 0,
                    "TX CAN message, time: %f, deltatime: %f ",
                    txMsg->timestamp, txMsg->deltatime);
     }
     ds2211_can_msg_read(rxMsg);
     if (rxMsg->processed == DS2211_CAN_PROCESSED)
     {
        msg_info_printf(MSG_SM_RTLIB, 0,
                    "RX CAN message, time: %f,deltatime: %f ",
                     rxMsg->timestamp, rxMsg->deltatime);
     }
     RTLIB_BACKGROUND_SERVICE();
  }
}
```

Related topics

Examples

Example of Handling Request and Remote Messages

Example

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 seconds. 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 sent to the message module.

```
#include <Brtenv.h>
#include <Ds2211.h>
#include <Can2211.h>

ds2211_canChannel* rqCh;
ds2211_canChannel* rmCh;
ds2211_canMsg* rqtxMsg;
ds2211_canMsg* rqrxMsg;
ds2211_canMsg* rmMsg;
UInt32 rmMsgData[8] = {1,2,3,4,5,6,7,8};

main()
{
   init(); /* initialize hardware system */

   ds2211_init(DS2211_1_BASE);

   ds2211_can_communication_init(DS2211_1_BASE,
                  DS2211_CAN_INT_DISABLE);

   rqCh = ds2211_can_channel_init(DS2211_1_BASE, 0,
                  500000,
                  DS2211_CAN_STD,
                  DS2211_CAN_NO_SUBINT,
                  DS2211_CAN_TERMINATION_ON);

   rmCh = ds2211_can_channel_init(DS2211_1_BASE, 1,
                  500000,
                  DS2211_CAN_STD,
                  DS2211_CAN_NO_SUBINT,
                  DS2211_CAN_TERMINATION_ON);

   rqtxMsg = ds2211_can_msg_rqtx_register(rqCh,
                        2,
                        0x123,
                        DS2211_CAN_STD,
                        DS2211_CAN_TIMECOUNT_INFO,
                        DS2211_CAN_NO_SUBINT,
                        4.0,
                        2.0,
                        DS2211_CAN_TIMEOUT_NORMAL);

   rqrxMsg = ds2211_can_msg_rqrx_register(rqtxMsg,
                  DS2211_CAN_DATA_INFO | DS2211_CAN_TIMECOUNT_INFO,
                  DS2211_CAN_NO_SUBINT);

   rmMsg = ds2211_can_msg_rm_register(rmCh,
                        3,
                        0x123,
                        DS2211_CAN_STD,
                        DS2211_CAN_TIMECOUNT_INFO,
                        DS2211_CAN_NO_SUBINT);

   ds2211_can_msg_write(rmMsg, 8, rmMsgData);

   ds2211_can_msg_rqtx_activate(rqtxMsg);

   ds2211_can_channel_start(rqCh, DS2211_CAN_INT_DISABLE);

   ds2211_can_channel_start(rmCh, DS2211_CAN_INT_DISABLE);

   for(;;)
   {

      ds2211_can_msg_read(rqrxMsg);
      ds2211_can_msg_read(rqtxMsg);
      ds2211_can_msg_read(rmMsg);

      if (rqrxMsg->processed == DS2211_CAN_PROCESSED)
      {
         msg_info_printf(MSG_SM_RTLIB, 0,
                  "RQRX CAN message, time: %f,deltatime: %f ",
                  rqrxMsg->timestamp, rqrxMsg->deltatime);
      }

      if (rqtxMsg->processed == DS2211_CAN_PROCESSED)
      {
         msg_info_printf(MSG_SM_RTLIB, 0,
               "RQTX CAN message, time: %f, deltatime: %f ",
               rqtxMsg->timestamp, rqtxMsg->deltatime);
      }

      if (rmMsg->processed == DS2211_CAN_PROCESSED)
      {
         msg_info_printf(MSG_SM_RTLIB, 0,
               "RM CAN message, time: %f, deltatime: %f ",
               rmMsg->timestamp, rmMsg->deltatime);
      }

      RTLIB_BACKGROUND_SERVICE();
   }
}
```

Related topics

Examples

Example of Handling Transmit and Receive Messages........ 536
Example of Receiving Different Message IDs........ 543
Example of Using Service Functions........ 542
Example of Using Subinterrupts........ 540

Example of Using Subinterrupts

Example

This example shows how to register messages that can generate a subinterrupt.

The CAN controller is started and a CAN message is 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 specified 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.

```c
#include <Brtenv.h>
#include <Ds2211.h>
#include <Can2211.h>
#define tx_subint 2
#define rx_subint 3
ds2211_canChannel* txCh;
ds2211_canChannel* rxCh;
ds2211_canMsg* txMsg;
ds2211_canMsg* rxMsg;
UInt32 txMsgData[8] = { 1,2,3,4,5,6,7,8 };
void can_user_callback(void* subint_data, Int32 subint)
{
   switch(subint)
   {
      case tx_subint:
         txMsgData[0] = (txMsgData[0]+1) & 0xFF;
         /* send the message delayed */
         ds2211_can_msg_send( txMsg, 8, txMsgData, 0.1);
         msg_info_printf(MSG_SM_RTLIB, 0, "TX Subint:%d", subint);
      break;
      case rx_subint:
         /* read the message from the communication buffer */
         ds2211_can_msg_read(rxMsg);
         msg_info_printf(MSG_SM_RTLIB,
            0,
            "RX Subint:%d, time: %fs, deltatime: %fs data[0]: %x",
            subint,
            rxMsg->timestamp,
            rxMsg->deltatime,
            rxMsg->data[0]);
      break;
      default:
      break;
   }
}
main()
{
   init(); /* initialize hardware system */
   ds2211_init(DS2211_1_BASE);
   ds2211_can_communication_init(DS2211_1_BASE,
                                 DS2211_CAN_INT_ENABLE);
   ds2211_can_subint_handler_install(DS2211_1_BASE,
                                 can_user_callback);
   txCh = ds2211_can_channel_init (DS2211_1_BASE, 1, 500000,
                                 DS2211_CAN_STD,
                                 DS2211_CAN_NO_SUBINT,
                                 DS2211_CAN_TERMINATION_ON);
   txMsg = ds2211_can_msg_tx_register(txCh,
                                    0,
                                    0x123,
                                    DS2211_CAN_STD,
                                    DS2211_CAN_NO_INFO,
                                    tx_subint,
                                    0.0,
                                    0.0,
                                    DS2211_CAN_TIMEOUT_NORMAL);
   rxCh = ds2211_can_channel_init(DS2211_1_BASE,
                                 0,
                                 500000,
                                 DS2211_CAN_STD,
                                 DS2211_CAN_NO_SUBINT,
                                 DS2211_CAN_TERMINATION_ON);
   rxMsg = ds2211_can_msg_rx_register(rxCh,
                                 0,
                                 0x123,
                                 DS2211_CAN_STD,
                                 DS2211_CAN_DATA_INFO |
                                 DS2211_CAN_TIMECOUNT_INFO,
                                 rx_subint);
   ds2211_can_channel_start(rxCh, DS2211_CAN_INT_DISABLE);
   ds2211_can_channel_start(txCh, DS2211_CAN_INT_DISABLE);
   ds2211_can_msg_send( txMsg, 8, txMsgData, 0.0);
   RTLIB_INT_ENABLE();
   for(;;)
   {
      RTLIB_BACKGROUND_SERVICE();
   }
}
```

Related topics

Examples

Example of Using Service Functions

Example

This example shows how to use the service functions `DS2211_CAN_SERVICE_TX_OK` and `DS2211_CAN_SERVICE_RX_OK`.

Note

No message is installed on the DS2211 in this example.

```
#include <Brtenv.h>
#include <Ds2211.h>
#include <Can2211.h>
ds2211_canChannel* canCh0;
ds2211_canChannel* canCh1;
ds2211_canService* txokServ;
ds2211_canService* rxokServ;
main()
{
   init();
   ds2211_init(DS2211_1_BASE);
   ds2211_can_communication_init(DS2211_1_BASE,
               DS2211_CAN_INT_DISABLE);
   canCh0 = ds2211_can_channel_init(DS2211_1_BASE, 0,
               500000, DS2211_CAN_STD, DS2211_CAN_NO_SUBINT,
               DS2211_CAN_TERMINATION_ON);
   canCh1 = ds2211_can_channel_init(DS2211_1_BASE, 1,
               500000, DS2211_CAN_STD, DS2211_CAN_NO_SUBINT,
               DS2211_CAN_TERMINATION_ON);
   /* register the tx_ok function which delivers the count */
   /* of the tx-ok counter for CAN channel 0 */
   txokServ = ds2211_can_service_register(canCh0,
               DS2211_CAN_SERVICE_TX_OK);
   /* register the rx_ok function which delivers the count */
   /* of the rx-ok counter for CAN channel 1 */
   rxokServ = ds2211_can_service_register(canCh1,
               DS2211_CAN_SERVICE_RX_OK);
   for(;;)
   {
     /* request the tx-ok counter from the slave DS2211 */
     ds2211_can_service_request(txokServ);
     /* request the rx-ok counter from the slave DS2211 */
     ds2211_can_service_request(rxokServ);
     /* read the tx-ok counter from the slave DS2211 */
     /* the data will be available in txokServ->data0 */
     ds2211_can_service_read(txokServ);
     /* read the rx-ok counter from the slave DS2211 */
     /* the data will be available in rxokServ->data0 */
     ds2211_can_service_read(rxokServ);
     RTLIB_BACKGROUND_SERVICE();
   }
}
```

Related topics

Examples

Example of Handling Request and Remote Messages.......... 538
Example of Handling Transmit and Receive Messages.......... 536
Example of Receiving Different Message IDs.......... 543
Example of Using Subinterrupts.......... 540

Example of Receiving Different Message IDs

Example

This example shows you how to set up a CAN controller to receive the message IDs 0x100 … 0x1FF via one message queue.

```
#include <Brtenv.h>
#include <Ds2211.h>
#include <Can2211.h>
ds2211_canChannel* rxCh;
ds2211_canMsg* canMonitor;
UInt32 data[8];
UInt32 mask = 0x1FFFFF00;
UInt32 queue_size = 64;
main()
{
   init();
   ds2211_init(DS2211_1_BASE);
   ds2211_can_communication_init(DS2211_1_BASE,
                 DS2211_CAN_INT_DISABLE);
   rxCh = ds2211_can_channel_init(DS2211_1_BASE,
                           0,
                           500000,
                           DS2211_CAN_STD,
                           DS2211_CAN_NO_SUBINT,
                           DS2211_CAN_TERMINATION_ON);
   canMonitor = ds2211_can_msg_rx_register (rxCh,
                           1,
                           0x100,
                           DS2211_CAN_STD,
                           DS2211_CAN_TIMECOUNT_INFO |
                                 DS2211_CAN_MSG_INFO,
                           DS2211_CAN_NO_SUBINT);
   ds2211_can_msg_set(canMonitor,
                     DS2211_CAN_MSG_MASK,
                     &mask);
   ds2211_can_msg_set(canMonitor,
                     DS2211_CAN_MSG_QUEUE,
                     &queue_size);
   ds2211_can_channel_start(rxCh, DS2211_CAN_INT_DISABLE);
   for(;;)
   {
      ds2211_can_msg_read(canMonitor);
      if (canMonitor->processed == DS2211_CAN_PROCESSED)
      {
         msg_info_printf(0,0,"id: %d time: %f",
            canMonitor->identifier, canMonitor->timestamp);
      }
      RTLIB_BACKGROUND_SERVICE();
   }
}
```

Related topics

Examples

Example of Handling Request and Remote Messages.......... 538
Example of Handling Transmit and Receive Messages.......... 536
Example of Using Service Functions.......... 542
Example of Using Subinterrupts.......... 540

Wave Table Generation

Introduction

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

Where to go from here

Information in this section

Wave Table MAT File Format

Introduction

Wave tables must be specified as MAT file. The MAT files must contain an array of defined data structures.

Syntax

Wave table MAT files must contain an array of data structures that uses the following syntax:

```
tbl(i).board
tbl(i).table
tbl(i).data
```

Naming conventions

The array can be given any required name. The structure elements `board`, `table` and `data` must be named exactly as specified above.

Structure elements

board Specifies the board number of the target DS2211 within the range 1 … 16

table Specifies the target wave table. The following identifier strings are allowed:

Predefined Symbol	Meaning
'crank1' … 'crank8'	Wave table 1 for crankshaft signal generation. … Wave table 8 for crankshaft signal generation.
'camA1' … 'camA8'	Wave table 1 for camshaft signal generation on channel 1. … Wave table 8 for camshaft signal generation on channel 1.
'camB1' … 'camB8'	Wave table 1 for camshaft signal generation on channel 2. … Wave table 8 for camshaft signal generation on channel 2.

> **Note**
>
> - A different table identifier has to be specified for each wave table of the same board. Otherwise, a compiler error may be generated due to multiply defined labels. The target wave table is the first crankshaft wave table on DS2211 board number 1.
> - The generated wave table can only be used with RTLib functions. It cannot be used with RTI.

data Specifies the data array of fixed length 65536.

Example

The following M-script generates a sine wave with amplitude ±1 and 120 periods per 720°. The target wave table is the first crankshaft wave table on DS2211 board number 1.

```
for i = 1:65536
   crank_data(i) = sin(120*4*pi*(i-1)/8192);
end
tbl(1).board = 1;
tbl(1).table = 'crank1';
tbl(1).data = crank_data;
```

Tip

You can find further example wave tables in `<RCP_HIL_InstallationPath>\Demos\DS100<x>\IOBoards\Ds2211\APU\Wavetables`.

MATCONV.EXE

Introduction

The MATCONV conversion utility converts a DS2210 or DS2211 wave table MAT file to assembler or C source code. The object file resulting from a generated assembler or C file can be linked to a master RTP application to load DS2210 or DS2211 wave tables.

Assembly source code

The generated assembler source file contains the .slvsect or SlvFwSection data section for systems combined with DS2211 boards. This section is loaded to the master processor memory together with the master application. When the wave table data has been loaded to the DS2211, the allocated memory of the master processor can be used for other purposes (see the description of the generic slave loading concept for further details).

The .slvsect or SlvFwSection section contains one or more data tables. Each table can be identified by a global symbol of the format:

```
wav2211<board_no>_<table_id>
```

where `board_no` and `table_id` are obtained from the MAT file (see Wave Table MAT File Format on page 545).

The generated assembler source file can be used in conjunction with the processor board. The assembler must be called with the command line option `-D DS_BOARD_TYPE=1006` to specify the target system. This is automatically performed if the corresponding Down utility is used.

A data table can be loaded by one of the RTLib wave table load functions `ds2211_crank_table_load` and `ds2211_cam_table_load`.

C source code

The C source code is compiled, linked and loaded together with the master application. It is handled like any other additional C application.

The C source code contains one or more data tables. Each table can be identified by a global symbol of the format:

```
wav2211<board_no>_<table_id>
```

`board_no` and `table_id` are obtained from the corresponding MAT file.

The generated C source file can be used in conjunction with DS1006 and DS1007 systems.

A data table can be loaded by one of the RTLib wave table load functions `ds2211_crank_table_load` and `ds2211_cam_table_load`.

The conversion utility is invoked by using the following syntax:

```
matconv input_file [options]
```

input_file MAT input file (default extension `.mat`)

Options The following command line options are available:

Option	Description
`/a`	Generate an *.asm assembly-file with data for loader
`/c`	Generate a *.c c-module with data array (default)
`/n`	No beep on error
`/o`	output_file output file name (default: input_file.<asm/c>)
`/t board_type`	Target board type: DS2210 or DS2211 (default)

Example The example converts the `wave1.mat` input file to an assembler source file `wave1.c`. Plain text information about the wave tables being converted is displayed on the screen.

```
matconv wave1.mat /c /v
```

MAT2C2211.M

Syntax

```
mat2c2211(
   C_fileName,
   MAT_fileNames,
   boardNo,
   tableIdentifier)
```

Description

MAT2C2211.M combines one or more wavetable MAT files into a single MAT file (see Wave Table MAT File Format on page 545) and invokes MATCONV.EXE to generate the corresponding assembly source code. The source MAT files must contain a single array of length 65536 each.

MAT2C2211.M is installed in `<RCP_HIL_InstallationPath>\matlab\rtlib100x\tools`.

Parameters

C_fileName String to identify the C file, for example, `'wavetabledata'`

MAT_fileNames String cell array to identify the source MAT files, for example, `{'crank1','c:\temp\crank2','camA1'}`

boardNo Double array to specify the DS2211 board number

tableIdentifier String cell array to identify the generated wave table, for example, `{'crank1','crank2','camA1'}`

Example

```
mat2c2211('wavetabledata',
          {'crank1','c:\temp\crank2','camA1'},
          [1,1,2],
          {'crank1','crank2','camA1'})
```

Related topics

References

Function Execution Times

Introduction

To give you the mean function execution times and basic information on the test environment used.

Where to go from here

Information in this section

Information on the Test Environment

Introduction

The execution times of the C functions can vary, since they depend on different factors. The measured execution times are influenced by the test environment used.

Test environment

The execution time of a function can vary, since it depends on different factors, for example:

- CPU clock and bus clock frequency of the processor board used
- Optimization level of the compiler
- Use of inlining parameters

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 down<xxxxx> tool (optimization and inlining). The execution times in the tables below are always the mean measurement values.

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 with 2.6 GHz	DS1006 with 3.0 GHz
ds2211_init	268.70 ms	266.949 ms
ds2211_mode_set	0.76 µs	0.71 µs
ds2211_digin_threshold_set	0.93 µs	1.67 µs
ds2211_digout_mode_set	0.69 µs	0.71 µs
ds2211_digwform_mode_set	0.68 µs	0.64 µs
ds2211_apu_transformer_mode_set	0.69 µs	0.72 µs
ds2211_digout_hs_vbat1_clear	0.65 µs	0.63 µs
ds2211_digout_hs_vbat1_set	0.65 µs	0.63 µs
ds2211_digout_hs_vbat1_write	0.04 µs	0.11 µs
ds2211_digout_hs_vbat2_clear	0.58 µs	0.55 µs
ds2211_digout_hs_vbat2_set	0.58 µs	0.62 µs
ds2211_digout_hs_vbat2_write	0.04 µs	0.035 µs
ds2211_digout_ls_write	0.04 µs	0.033 µs
ds2211_digout_ls_set	0.66 µs	0.63 µs
ds2211_digout_ls_clear	0.79 µs	0.64 µs

ADC

The following execution times were measured for ADC functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_adc_start`	0.29 µs	0.05 µs
`ds2211_adc_block_in_fast` (16 channels)	20.41 µs	24.03 µs
`ds2211_adc_single_in` (channel 16)	17.75 µs	16.99 µs
`ds2211_adc_block_init` (16 channels)	0.37 µs	0.36 µs
`ds2211_adc_block_start`	0.11 µs	0.031 µs
`ds2211_adc_block_in` (16 channels)	29.09 µs	29.17 µs

DAC

The following execution time were measured for DAC functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_dac_out`	0.07 µs	0.10 µs

Bit I/O

The following execution times were measured for Bit I/O functions:

Function	Execution Time		
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz	DS1006 Multicore with 2.8 GHz
`ds2211_bit_io_in`	0.71 µs	0.62 µs	
`ds2211_bit_io_in_group`	0.78 µs	0.78 µs	0.67 µs
`ds2211_bit_io_out`	0.16 µs	0.029 µs	
`ds2211_bit_io_set`	0.75 µs	0.64 µs	
`ds2211_bit_io_clear`	0.69 µs	0.63 µs	

D/R converter

The following execution times were measured for D/R converter functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_resistance_out`	0.07 µs	0.058 µs

Timing mode

The following execution times were measured for timing mode functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_timing_out_mode_set`	0.88 µs	0.89 µs
`ds2211_timing_in_mode_set`	0.94 µs	0.96 µs

PWM signal generation

The following execution times were measured for PWM generating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	**DS1006 with 3.0 GHz**
`ds2211_timing_out_mode_set`	0.88 µs	0.89 µs
`ds2211_pwm_out`	0.28 µs	0.10 µs

PWM signal measurement

The following execution times were measured for PWM measuring functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	**DS1006 with 2.6 GHz**
`ds2211_timing_in_mode_set`	0.94 µs	0.96 µs
`ds2211_pwm_in`	0.83 µs	0.67 µs

Square-wave signal measurement and generation

The following execution times were measured for square-wave signal measurement and generation functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	**DS1006 with 3.0 GHz**
`ds2211_d2f`	0.14 µs	0.29 µs
`ds2211_f2d`	0.87 µs	0.66 µs

SENT functions

The following execution times were measured for the SENT functions.

Function	Execution Times[1], [2]		
	DS1006 with 2.6 GHz	**DS1006 with 3.0 GHz**	**DS1006 Multicore with 2.8 GHz**
`ds2211_sent_tx_init`	0.18 µs	0.18 µs	0.14 µs
`ds2211_sent_tx_config`	0.10 µs	0.09 µs	0.13 µs
`ds2211_sent_set_tx_tic_period`	0.12 µs	0.10 µs	0.15 µs
`ds2211_sent_tx_transmit_pause` (pause pulse disabled)	m · RoundUp[n / 7] · 0.17 µs + 0.70 µs	m · RoundUp[n / 7] · 0.17 µs + 0.69 µs	m · RoundUp[n / 7] · 0.17 µs + 0.80 µs
`ds2211_sent_tx_transmit_pause` (pause pulse enabled)	m · RoundUp[n / 7 + 1] · 0.17 µs + 0.70 µs	m · RoundUp[n / 7 + 1] · 0.17 µs + 0.69 µs	m · RoundUp[n / 7 + 1] · 0.17 µs + 0.80 µs
`ds2211_sent_tx_fifo_state`	0.67 µs	0.66 µs	0.65 µs
`ds2211_sent_tx_pause_mode`	0.06 µs	0.05 µs	0.05 µs
`ds2211_sent_rx_init`	n · 0.005 µs + 2.02 µs	n · 0.004 µs + 1.82 µs	n · 0.004 µs + 1.98 µs
`ds2211_sent_rx_config_pause`	1.03 µs	1.03 µs	0.98 µs
`ds2211_sent_get_rx_tic_period`	0.02 µs	0.016 µs	0.037 µs

Function	Execution Times[1], [2]		
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz	DS1006 Multicore with 2.8 GHz
`ds2211_sent_rx_receive_all_pause` (pause pulse disabled)	m · (n · 0.66 µs + 0.75 µ s) + 0.73 µs	m · (n · 0.66 µs + 0.75 µ s) + 0.73 µs	m · (n · 0.66 µs + 0.72 µ s) + 0.76 µs
`ds2211_sent_rx_receive_most_recent_pause` (pause pulse disabled)	m · (n · 0.67 µs + 0.75 µ s) + 0.78 µs	m · (n · 0.65 µs + 0.72 µ s) + 0.75 µs	m · (n · 0.66 µs + 0.71 µ s) + 0.80 µs
`ds2211_sent_rx_receive_all_pause` (pause pulse enabled)	m · (n · 0.67 µs + 1.26 µ s) + 0.71 µs	m · (n · 0.66 µs + 1.22 µ s) + 0.73 µs	m · (n · 0.66 µs + 1.24 µ s) + 0.75 µs
`ds2211_sent_rx_receive_most_recent_pause` (pause pulse enabled)	m · (n · 0.67 µs + 1.26 µ s) + 0.78 µs	m · (n · 0.65 µs + 1.22 µ s) + 0.75 µs	m · (n · 0.66 µs + 1.23 µ s) + 0.80 µs

[1] n is the number of nibbles per message (nibble_count).

[2] m is the number of received messages or messages to transmit.

Overall APU functions

The following execution times were measured for overall APU functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_apu_position_write`	0.83 µs	0.84 µs
`ds2211_apu_position_read`	0.71 µs	0.95 µs
`ds2211_apu_velocity_write`	0.13 µs	0.12 µs
`ds2211_int_position_set`	91254 + (4.9 · count) µs	92681 + (14.3 · count) µs

Engine position phase accumulation

The following execution times were measured for engine position phase accumulation functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_apu_start`	0.81 µs	0.71 µs
`ds2211_apu_stop`	1.40 µs	1.25 µs

Crankshaft sensor signal generation

The following execution times were measured for crankshaft sensor signal generating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_crank_table_load`	48.90 ms	48.93 ms
`ds2211_crank_table_select`	0.68 µs	0.64 µs

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_crank_output_ampl_set`	0.47 µs	0.29 µs
`ds2211_reverse_crank_setup`	0.95 µs	0.88 µs
`ds2211_crank_mode`	0.29 µs	0.25 µs

Camshaft sensor signal generation

The following execution times were measured for camshaft sensor signal generating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_cam_table_load`	92.04 ms	92.19 ms
`ds2211_cam_table_select`	0.70 µs	0.80 µs
`ds2211_cam_output_ampl_set`	0.43 µs	0.30 µs
`ds2211_cam_phase_write`	0.79 µs	0.82 µs
`ds2211_cam_phase_read`	0.78 µs	0.70 µs
`ds2211_cam_phase_offset_update_mode`	0.76 µs	0.81 µs

APU capture units

The following execution times were measured for APU capturing functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_apu_ignition_cc_setup`	11.35 µs	1.53 µs
`ds2211_apu_injection_cc_setup`	9.80 µs	1.07 µs
`ds2211_event_window_set` (capture window = 720°)	93.44 ms	91859 µs
`ds2211_ign_capture_mode_set`	3.11 µs	16.12 µs
`ds2211_aux1_capture_mode_set`	3.11 µs	8.02 µs
`ds2211_aux2_capture_mode_set`	3.11 µs	8.10 µs
`ds2211_inj_capture_mode_set`	1.48 µs	4.08 µs
`ds2211_ignition_status_read` (8 channels)	0.65 µs	0.67 µs
`ds2211_injection_status_read` (8 channels)	0.67 µs	0.62 µs

Spark event capture

The following execution times were measured for spark event capturing functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_event_window_set`	93.44 ms	91.86 ms
`ds2211_ign_capture_mode_set`	3.11 µs	15.96 µs
`ds2211_aux1_capture_mode_set`	3.11 µs	8.02 µs

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_aux2_capture_mode_set`	3.11 µs	8.10 µs
`ds2211_ignition_fifo_read` (8 pulses/720°, 10 ms sampling time, 0 … 30000 rpm)	0.67 … 1.32 µs	0.66 … 0.68 µs
`ds2211_ignition_capture_read` (8 pulses/720°, 10 ms sampling time, 0 … 30000 rpm)	0.83 … 1.73 µs	0.76 … 38.85 µs

Injection pulse position and fuel amount measurement

The following execution times were measured for the functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_event_window_set`	93.44 ms	91.859 ms
`ds2211_inj_capture_mode_set`	19.69 µs	17.10 µs
`ds2211_injection_fifo_read` (8 pulses/720°, 10 ms sampling time, 0 … 30000 rpm)	0.67 … 1.3 µs	0.66 … 0.70 µs
`ds2211_injection_capture_read` (8 pulses/720°, 10 ms sampling time, 0 … 30000 rpm)	0.77 … 1.67 µs	0.74 … 151.69 µs

Overall DSP

The following execution times were measured for overall DSP functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_slave_dsp_signal_enable`	1.07 µs	0.87 µs
`ds2211_slave_dsp_channel_enable`	1.56 µs	1.54 µs
`ds2211_slave_dsp_interrupt_set`	0.29 µs	0.032 µs
`ds2211_slave_dsp_speedchk`	3.23 µs	3.30 µs
`ds2211_slave_dsp_error`	1.63 µs	1.57 µs
`ds2211_slave_dsp_appl_load` (loading application for knock sensor simulation)	0.92 ms	0.894 ms
`ds2211_slave_dsp_appl_load` (loading application for wheel speed sensor simulation)	0.73 ms	0.894 ms

Knock sensor simulation

The following execution times were measured for knock sensor simulating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_slave_dsp_knock_init`	18.09 µs	17.67 µs
`ds2211_slave_dsp_knock_update`	1.73 µs	2.05 µs
`ds2211_slave_dsp_knock_noise`	0.99 µs	1.82 µs

Wheel speed sensor simulation

The following execution times were measured for wheel speed sensor simulating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_slave_dsp_wheel_init`	6.69 µs	6.64 µs
`ds2211_slave_dsp_wheel_update`	1.16 µs	1.32 µs

Slave DSP memory access

The following execution times were measured for slave DSP memory accessing functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2211_slave_dsp_read`	1.49 µs	1.43 µs
`ds2211_slave_dsp_write`	0.87 µs	0.86 µs
`ds2211_slave_dsp_block_read`	1.53 + count · 0.496 µs	0.925 + count · 0.528 µs
`ds2211_slave_dsp_block_write`	1.08 + count · 0.15 µs	1.0 + count · 0.142 µs
`ds2211_slave_dsp_sem_req`	1.02 µs	0.855 µs
`ds2211_slave_dsp_sem_rel`	0.04 µs	0.035 µs
`ds2211_slave_dsp_read_direct`	0.67 µs	0.62 µs
`ds2211_slave_dsp_write_direct`	0.04 µs	0.032 µs
`ds2211_slave_dsp_block_read_di`	0.65 + count · 0.49 µs	0.088 + count · 0.528 µs
`ds2211_slave_dsp_block_write_di`	-0.105 + count · 0.142 µs	-0.10 + count · 0.133 µs

CAN Access

The following execution times were measured for CAN access functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
CAN channel handling		
`ds2211_can_channel_all_sleep`	1.33 µs	1.44 µs
`ds2211_can_channel_all_wakeup`	1.33 µs	1.44 µs

Function	Execution Time	
	DS1006 with 2.6 GHz	**DS1006 with 3.0 GHz**
`ds2211_can_channel_BOff_go`	1.51 µs	1.49 µs
`ds2211_can_channel_BOff_return`	1.32 µs	1.52 µs
CAN message access		
`ds2211_can_msg_write`	2.0 µs	2.00 µs
`ds2211_can_msg_send`	2.27 µs	2.68 µs
`ds2211_can_msg_sleep`	1.97 µs	2.03 µs
`ds2211_can_msg_wakeup`	1.58 µs	1.69 µs
`ds2211_can_msg_read`	7.1 µs	1.72 µs
`ds2211_can_msg_trigger`	2.5 µs	2.03 µs
`ds2211_can_msg_clear` (for each message to be cleared)	0.05 µs	0.04 µs
`ds2211_can_msg_processed_register`	1.3 µs	1.368 µs
`ds2211_can_msg_processed_read`	0.9 µs	0.73 µs
CAN service		
`ds2211_can_service_request`	2.5 µs	1.38 µs
`ds2211_can_service_read`	8.2 µs	4.11 µs

Related topics

Basics

Information on the Test Environment.. 551