DS2210 HIL I/O Board

RTLib Reference

Release 2021-A – May 2021

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

Contents

About This Reference

About this reference

The DS2210 Real-Time Library (RTLib) provides the C functions and macros you need to program the DS2210 HIL I/O Board.

Symbols

dSPACE user documentation uses the following symbols:

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

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

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

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 DS2210 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

ds2210_init

Syntax

```
void ds2210_init(Int32 base)
```

Include file

`Ds2210.h`

Purpose

To perform the basic initialization of the DS2210.

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 ds2210_mode_set) and resets all board functions to their default settings.

Function	Default Setting
Board mode	Master
Digital inputs	Sets 2.5 V threshold voltage
Digital outputs	Disabled
Transformer outputs (APU, slave DSP)	Disabled
Digital waveform outputs	Clearing enabled
Analog outputs (DAC)	Sets 0 V
Resistor outputs (RES)	High-Z (1 MΩ)
Slave DSP C31	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.
- Checks whether the board supports extended functionality. The function evaluates the board and FPGA revision numbers. The information is needed by some functions to identify which mode selections are not allowed. If the board supports extended functionality, the PWM units (inputs and outputs) are initialized to 16-bit resolution else to 14-bit resolution. Extended functionality is supported for the following board versions:
  - DS2210 boards with board revision numbers 4 and higher
  - DS2210 boards with board revision number 3 and FPGA revision numbers 3 and higher (FPGA = field programmable gate array).
- Outputs the board and the FPGA revision numbers.
- Clears the interrupt position flags, the dual-port memories, and clears the waveform data buffer.
- Depending on the global debug status flag DEBUG_INIT, the function outputs an info message signaling the completion of the initialization and the board and FPGA revision numbers. Refer to Initialization (DS1006 RTLib Reference 🕮) or Initialization (DS1007 RTLib Reference 🕮).

`ds2210_init` avoids multiple execution of the initialization code.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

Return value

None

Messages

The following message is defined:

Type	Message	Meaning
Error	ds2210_init(board_offset): Board not found!	There is no DS2210 at the given board index (offset of the PHS-bus address).

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_mode_set

Syntax

```
void ds2210_mode_set(
      Int32 base,
      Int32 mode)
```

Include file

`Ds2210.h`

Purpose

To set the DS2210 board to master or slave mode. The APU of the master board calculates the engine position and supplies the information to slave DS2210 boards via the expansion bus connector (P3).

Note

When cascading several DS2210 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 DS2210 board.

mode The following symbols are predefined to set the mode:

Predefined Symbol	Meaning
`DS2210_SLAVE_MODE`	Sets the board to slave mode.
`DS2210_MASTER_MODE`	Sets the board to master mode.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_digin_threshold_set

Syntax

```
void ds2210_digin_threshold_set(
     Int32 base,
     dsfloat value)
```

Include file

`Ds2210.h`

Purpose

To set the threshold level for all digital inputs of the DS2210 board. 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.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

value Specifies the threshold level within the range of 1 … 7 V. You can set the value in steps of 250 mV (1.000, 1.250, 1.500, …).

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_init.......... 17

ds2210_digout_mode_set

Syntax

```
void ds2210_digout_mode_set(
      Int32 base,
      Int32 mode)
```

Include file

`Ds2210.h`

Purpose

To enable or disable the 12-V digital outputs (for bit I/O unit, PWM generation, and the digital outputs of the crankshaft and the camshaft sensor). For further information, refer to Digital Outputs (PHS Bus System Hardware Reference).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

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

Predefined Symbol	Meaning
`DS2210_DIGOUT_ENABLE`	Enables the digital output circuits.
`DS2210_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 385.

Related topics

References

ds2210_digwform_mode_set

Syntax

```
void ds2210_digwform_mode_set(
      Int32 base,
      Int32 mode)
```

Include file

`Ds2210.h`

Purpose

To enable clearing of the digital waveform outputs of the angular processing unit. If this function is called with the parameter `DS2210_DIGWFORM_CLEAR_ENABLE` the hardware clears the digital waveform 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 DS2210 board.

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

Predefined Symbol	Meaning
DS2210_DIGWFORM_CLEAR_ENABLE	The hardware clears the digital waveform outputs if the APU is stopped or the velocity is 0.
DS2210_DIGWFORM_CLEAR_DISABLE	Digital waveform outputs will not be cleared.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_apu_transformer_mode_set

Syntax

```
void ds2210_apu_transformer_mode_set(
     Int32 base,
     Int32 mode)
```

Include file

`Ds2210.h`

Purpose

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

Description

This function applies to the crankshaft and camshaft signal generation of the APU, as well as 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 DS2210 board.

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

Predefined Symbol	Meaning
`DS2210_APU_TRANSFORMER_ENABLE`	Enables the D/A converters connected to the APU transformers.
`DS2210_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 385.
Related topics	References

Conversion Macros

Introduction

There are some macros to convert the value of an engine position or speed.

Where to go from here

Information in this section

DS2210_DEG

Syntax

```
dsfloat DS2210_DEG(dsfloat position)
```

Include file

`Ds2210.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

DS2210_RAD 26

DS2210_RAD

Syntax	`dsfloat DS2210_RAD(dsfloat position)`
Include file	`Ds2210.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

DS2210_RPM

Syntax	`dsfloat DS2210_RPM(dsfloat speed)`
Include file	`Ds2210.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

DS2210_RAD_S 27

DS2210_RAD_S

Syntax

```
dsfloat DS2210_RAD_S(dsfloat speed)
```

Include file

`Ds2210.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

DS2210_RPM 26

Sensor and Actuator Interface

Introduction

The RTLib provides the functions you need to program the I/O units served by the sensor and actuator interface.

Where to go from here

Information in this section

ADC Unit

Introduction

To program the analog/digital converters.

Where to go from here

Information in this section

Basics of the ADC Unit

Basics

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

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.

I/O mapping

For general information on the ADC and its I/O mapping, refer to ADC Unit (DS2210 Features 📖) and 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 <Ds2210.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);
   RTLIB_SRT_START();         /* start execution time measurement */
   /* start A/D channels 1 … 4 for conversion */
   ds2210_adc_start(DS2210_1_BASE, DS2210_ADC_START4);
   /* read A/D channels 1 … 4 */
   ds2210_adc_block_in_fast(DS2210_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 */
   ds2210_init(DS2210_1_BASE);          /* initialize DS2210 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   RTLIB_TIC_START(DT, isr_t1);       /*initialize sampling clock timer*/
   RTLIB_TIC_INIT();
   while(1)                                   /* background process */
   {
      RTLIB_BACKGROUND_SERVICE();
   } /* while(1) */
```

```
} /* main() */
```

Related topics

References

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 <Ds2210.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 */
   ds2210_adc_start(DS2210_1_BASE, DS2210_ADC_MASK(6));
   /* read A/D channel 6 */
   ds2210_adc_single_in(DS2210_1_BASE, channel, &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 */
   ds2210_init(DS2210_1_BASE);   /* initialize DS2210 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() */
```

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 <Ds2210.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 numbers */
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 */
   ds2210_adc_block_start(DS2210_1_BASE);
   /* read A/D channel 6 */
   ds2210_adc_block_in(DS2210_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 */
   ds2210_init(DS2210_1_BASE);   /* initialize DS2210 board */
   ds2210_adc_block_init(DS2210_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

References

ds2210_adc_block_in 40
ds2210_adc_block_init 37
ds2210_adc_block_start 38
ds2210_init 17
init 212

ds2210_adc_start

Syntax

```
void ds2210_adc_start(
      Int32 base,
      Int32 mask)
```

Include file

`Ds2210.h`

Purpose

To start the A/D conversion for the channels 1 … 4, 1 … 8, 1 … 12 or 1 … 16. The A/D conversion is performed sequentially for the selected channels.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS2210 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

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

Predefined Symbol	Meaning
`DS2210_ADC_START4`	Starts channels 1 … 4.
`DS2210_ADC_START8`	Starts channels 1 … 8.
`DS2210_ADC_START12`	Starts channels 1 … 12.
`DS2210_ADC_START16`	Starts channels 1 … 16.
`DS2210_ADC_MASK(channel)`	Starts the block the given channel belongs to. Use this symbol in connection with ds2210_adc_single_in.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

ds2210_adc_block_in_fast

Syntax

```
void ds2210_adc_block_in_fast(
      Int32 base,
      dsfloat *data)
```

Include file

`Ds2210.h`

Purpose

To read the input values when the A/D conversion is finished.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS2210 Features 📖).

Description

You have to start the conversion with `ds2210_adc_start` first. Then `ds2210_adc_block_in_fast` waits until the A/D conversion is complete and reads the input values of the selected channels. 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 DS2210 board.

data Start address where input data is written. The values are scaled as follows:

Input Voltage Range	Return Value Range
0 … 20 V	0 … 1.0

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

ds2210_adc_single_in

Syntax

```
void ds2210_adc_single_in(
      Int32 base,
      Int32 channel,
      dsfloat *value)
```

Include file

`Ds2210.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 (DS2210 Features 🕮).

Description

You have to start the conversion with `ds2210_adc_start` first. Then `ds2210_adc_single_in` waits until the A/D conversion is complete and reads the input value of the selected channel. The input value is scaled to the range of 0 … 1.0 and is written to memory at the address given by the parameter `value`.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

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

Predefined Symbol	Meaning
`DS2210_ADC_CH1`	A/D channel 1
…	…
`DS2210_ADC_CH16`	A/D channel 16

value Input value. The value is scaled as follows:

Input Voltage Range	Return Value Range
0 … 20 V	0 … 1.0

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

Example of Single Read.. 32

References

ds2210_adc_start.. 34

ds2210_adc_block_init

Syntax

```
void ds2210_adc_block_init(
      Int32 base,
      Int32 count,
      Int32 *channels)
```

Include file

`Ds2210.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 (DS2210 Features 🕮).

Description

`ds2210_adc_block_init` determines the blocks of channels that must be converted in correspondence to the channel numbers specified in the array `channels`.

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 `ds2210_adc_block_init` the conversion will be started for channels 1 … 8.

Parameters

base Specifies the PHS-bus base address of the DS2210 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
`DS2210_ADC_CH1`	A/D channel 1
…	…
`DS2210_ADC_CH16`	A/D channel 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

Example of Multiple Read 33

References

ds2210_adc_block_start 38

ds2210_adc_block_start

Syntax

```
void ds2210_adc_block_start(Int32 base)
```

Include file	`Ds2210.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 (DS2210 Features 📖).
Description	The A/D conversion is started for the channels selected by the `ds2210_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 `ds2210_adc_block_init`, the conversion will be started for channels 1 … 8.

Parameters	**base** Specifies the PHS-bus base address of the DS2210 board.
Return value	None
Execution times	For information, refer to Function Execution Times on page 385.

Related topics

Examples

Example of Multiple Read........33

References

ds2210_adc_block_in........40
ds2210_adc_block_init........37

ds2210_adc_block_in

Syntax

```
void ds2210_adc_block_in(
     Int32 base,
     dsfloat *data)
```

Include file

`Ds2210.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 (DS2210 Features 📖).

Description

You have to start the channels with `ds2210_adc_block_start` first. Then `ds2210_adc_block_in` polls the ADC busy flag until conversion is complete and reads the input values of the selected channels. The input values are scaled to the range of 0 … 1.0 and written to the memory starting at the address given by the `data` parameter.

Parameters

base Specifies the PHS-bus base address of the DS2210 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 … 20 V	0 … 1.0

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

DAC Unit

Purpose

To program the digital/analog converters.

Where to go from here

Information in this section

Information in other sections

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

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 <Ds2210.h>
#define DT 1e-3                         /* 1 ms simulation step size */
/*-------------------------------------------------------------*/
dsfloat val = 0.0;                             /* D/A 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 */
   ds2210_dac_out(DS2210_1_BASE, DS2210_DAC_CH1, value);
   val = (val == 0.0) ? 0.5 : 0.0;           /* toggle D/A value */
   ds2210_dac_out(DS2210_1_BASE, DS2210_DAC_CH2, value);
   exec_time = RTLIB_TIC_READ();        /* calculate execution time */
   RTLIB_SRT_ISR_END();         /* end of interrupt service routine */
}
/*-------------------------------------------------------------*/
void main()
{
   init();                              /* initialize hardware system */
   ds2210_init(DS2210_1_BASE);             /* initialize DS2210 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

ds2210_dac_out 42
ds2210_init 17
init 212

ds2210_dac_out

Syntax

```
void ds2210_dac_out(
     Int32 base,
     Int32 channel,
     dsfloat value)
```

Include file

`Ds2210.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 (DS2210 Features).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Channel number within the range of 1 … 12. You can also use the following predefined symbols:

Predefined Symbol	Meaning
`DS2210_DAC_CH1`	DAC channel 1
…	…
`DS2210_DAC_CH12`	DAC channel 12

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 385.

Related topics

Examples

Example of the DAC Unit..41

References

DAC Unit (DS2210 Features 📖)

Bit I/O Unit

Purpose

To program the digital I/O port.

Where to go from here

Information in this section

Information in other sections

Bit I/O Unit (DS2210 Features)
The bit I/O unit contains one 16-bit port for input that provides 16 discrete digital input lines, and one 16-bit port for output that provides 16 discrete digital outputs.

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 <Ds2210.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;
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   ts_timestamp_read(&ts);
   host_service(1, &ts);          /* data acquisition service*/
   RTLIB_TIC_START();             /* start time measurement */
   /* clears I/O port i and sets I/O port i to "1" */
   mask_clear = i;
   mask_set = i - 1;
   ds2210_bit:io_clear(DS2210_1_BASE, (0x0001 << mask_clear));
   ds2210_bit_io_set(DS2210_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                                  */
   ds2210_bit_io_in(DS2210_1_BASE, &bitmap);
   exec_time = RTLIB_TIC_READ();     /* calculate execution time */
   RTLIB_SRT_ISR_END();      /* end of interrupt service routine */
}
/*---------------------------------------------------------------*/
void main()
{
   init();                           /* initialize hardware system */
   ds2210_init(DS2210_1_BASE);        /* initialize DS2210 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* enable digital outputs */
   ds2210_digout_mode_set(DS2210_1_BASE, DS2210_DIGOUT_ENABLE);
   /* sets the Bit I/O ports 0 … 15 to "1" */
   ds2210_bit_io_out(DS2210_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

ds2210_bit_io_clear 49
ds2210_bit_io_in 46
ds2210_bit_io_out 47
ds2210_bit_io_set 48
ds2210_digout_mode_set 21
ds2210_init 17
init 212

ds2210_bit_io_in

Syntax

```
void ds2210_bit_io_in(
      Int32 base,
      UInt32 *value)
```

Include file

`Ds2210.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 inputs are 12-V compatible. After initialization, the input threshold is set to 2.5 V. Use `ds2210_digin_threshold_set` to set the threshold within the range of 1.0 … 7.0 V.

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

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS2210 Features 📖).

Parameters

base	Specifies the PHS-bus base address of the DS2210 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 385.

Related topics

Examples

References

ds2210_bit_io_out

Syntax

```
void ds2210_bit_io_out(
      Int32 base,
      UInt32 value)
```

Include file

`Ds2210.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 `ds2210_bit_io_set` to set individual output bits without affecting other bits.

By default the outputs are disabled. Use `ds2210_digout_mode_set` to enable the digital outputs.

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 (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 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
DS2210_BITIO_OUT1	0x0001	Sets bit 1
DS2210_BITIO_OUT2	0x0002	Sets bit 2
DS2210_BITIO_OUT3	0x0004	Sets bit 3
...	...	...
DS2210_BITIO_OUT16	0x8000	Sets bit 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

ds2210_bit_io_set

Syntax

```
void ds2210_bit_io_set(
      Int32 base,
      UInt32 mask)
```

Include file

`Ds2210.h`

Purpose

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

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS2210 Features 🕮).

> **Note**
>
> After initialization, the outputs are disabled. Use `ds2210_digout_mode_set` to enable the digital output ports. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2210 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
`DS2210_BITIO_OUT1`	0x0001	Sets bit 1
`DS2210_BITIO_OUT2`	0x0002	Sets bit 2

Predefined Symbol	Value	Meaning
DS2210_BITIO_OUT3	0x0004	Sets bit 3
...	...	...
DS2210_BITIO_OUT16	0x8000	Sets bit 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

ds2210_bit_io_clear

Syntax

```
void ds2210_bit_io_clear(
      Int32 base,
      UInt32 mask)
```

Include file

`Ds2210.h`

Purpose

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

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS2210 Features 📖).

Description

After initialization, the outputs are disabled. Use `ds2210_digout_mode_set` to enable the digital output ports. For further information, refer to Digital Outputs (PHS Bus System Hardware Reference 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 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
`DS2210_BITIO_OUT1`	0x0001	Sets bit 1
`DS2210_BITIO_OUT2`	0x0002	Sets bit 2
`DS2210_BITIO_OUT3`	0x0004	Sets bit 3
…	…	…
`DS2210_BITIO_OUT16`	0x8000	Sets bit 16

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

Example of the Bit I/O Unit.. 44

References

ds2210_bit_io_set.. 48
ds2210_digout_mode_set... 21

D/R Converter

Purpose

To generate resistance outputs.

Where to go from here

Information in this section

Information in other sections

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

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 <Ds2210.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 */
   ds2210_resistance_out(DS2210_1_BASE, DS2210_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 */
   ds2210_init(DS2210_1_BASE);        /* initialize DS2210 board */
```

```
   /* set resistor1 output to 1.5 kOhm */
   ds2210_resistance_init(DS2210_1_BASE, DS2210_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

ds2210_resistance_out

Syntax

```
void ds2210_resistance_out(
      Int32 base,
      Int32 channel,
      dsfloat value)
```

Include file

`Ds2210.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 (DS2210 Features).

Description

`ds2210_resistance_out` writes the resistance value to the specified channel. For values smaller than 15.26 Ω the output saturates at 15.26 Ω. For values higher than 1 MΩ the output resistance becomes infinity.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_RES_CH1`	Resistor channel 1
…	…
`DS2210_RES_CH6`	Resistor channel 6

value Output value within the range of 15.26 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). The following values are possible:

INVRES	Resulting R
0	Infinity
1	1 MΩ
2	500 kΩ
…	…
65,535	15.26 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 385.

Related topics

Examples

Timing Mode

Introduction

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

Where to go from here

Information in this section

ds2210_timing_out_mode_set

Syntax

```
void ds2210_timing_out_mode_set(
      Int32 base,
      Int32 channel,
      Int32 range,
      Int32 mode)
```

Include file

`Ds2210.h`

Purpose

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

Note

- The frequency mode is available only on boards, which support extended functionality. The function checks this restriction and exits with an error message if the extended functionality is not supported by the hardware.
- 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 DS2210 board.

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_PWMOUT_CH1`	PWM channel 1
…	…
`DS2210_PWMOUT_CH6`	PWM channel 6

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
`DS2210_TIMING_RANGE1`	9.54 Hz	20 kHz	50 ns
`DS2210_TIMING_RANGE2`	4.77 Hz	20 kHz	100 ns
`DS2210_TIMING_RANGE3`	2.39 Hz	20 kHz	200 ns
`DS2210_TIMING_RANGE4`	1.20 Hz	20 kHz	400 ns
`DS2210_TIMING_RANGE5`	0.60 Hz	20 kHz	800 ns
`DS2210_TIMING_RANGE6`	0.30 Hz	20 kHz	1.6 µs
`DS2210_TIMING_RANGE7`	0.15 Hz	20 kHz	3.2 µs
`DS2210_TIMING_RANGE8`	75 mHz	20 kHz	6.4 µs
`DS2210_TIMING_RANGE9`	38 mHz	19.53 kHz	12.8 µs
`DS2210_TIMING_RANGE10`	19 mHz	9.76 kHz	25.6 µs
`DS2210_TIMING_RANGE11`	10 mHz	4.88 kHz	51.2 µs
`DS2210_TIMING_RANGE12`	5.0 mHz	2.44 kHz	103 µs
`DS2210_TIMING_RANGE13`	2.5 mHz	1.22 kHz	205 µs
`DS2210_TIMING_RANGE14`	1.2 mHz	610.35 Hz	410 µs
`DS2210_TIMING_RANGE15`	0.6 mHz	305.17 Hz	820 µs
`DS2210_TIMING_RANGE16`	0.3 mHz	152.59 Hz	1.64 ms

The following symbols are predefined in PWM mode (16 bit resolution):

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
`DS2210_TIMING_RANGE1`	200 ns	50 µs	3.27 ms	50 ns
`DS2210_TIMING_RANGE2`	400 ns	50 µs	6.55 ms	100 ns
`DS2210_TIMING_RANGE3`	800 ns	50 µs	13.1 ms	200 ns
`DS2210_TIMING_RANGE4`	1.6 µs	50 µs	26.2 ms	400 ns
`DS2210_TIMING_RANGE5`	3.2 µs	50 µs	52.4 ms	800 ns
`DS2210_TIMING_RANGE6`	6.4 µs	50 µs	104 ms	1.6 µs
`DS2210_TIMING_RANGE7`	12.8 µs	50 µs	209 ms	3.2 µs
`DS2210_TIMING_RANGE8`	25.6 µs	50 µs	419 ms	6.4 µs
`DS2210_TIMING_RANGE9`	51.2 µs	51.2 µs	838 ms	12.8 µs
`DS2210_TIMING_RANGE10`	103 µs	103 µs	1.67 s	25.6 µs

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
DS2210_TIMING_RANGE11	205 µs	205 µs	3.35 s	51.2 µs
DS2210_TIMING_RANGE12	410 µs	410 µs	6.71 s	103 µs
DS2210_TIMING_RANGE13	820 µs	820 µs	13.4 s	205 µs
DS2210_TIMING_RANGE14	1.64 ms	1.64 ms	26.8 s	410 µs
DS2210_TIMING_RANGE15	3.28 ms	3.28 ms	53.6 s	820 µs
DS2210_TIMING_RANGE16	6.55 ms	6.55 ms	107.3 s	1.64 ms

The following symbols are predefined in PWM mode (14 bit resolution):

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
DS2210_TIMING_RANGE1	200 ns	50 µs	820 µs	50 ns
DS2210_TIMING_RANGE2	400 ns	50 µs	1.64 ms	100 ns
DS2210_TIMING_RANGE3	800 ns	50 µs	3.27 ms	200 ns
DS2210_TIMING_RANGE4	1.6 µs	50 µs	6.55 ms	400 ns
DS2210_TIMING_RANGE5	3.2 µs	50 µs	13.1 ms	800 ns
DS2210_TIMING_RANGE6	6.4 µs	50 µs	26.2 ms	1.6 µs
DS2210_TIMING_RANGE7	12.8 µs	50 µs	52.4 ms	3.2 µs
DS2210_TIMING_RANGE8	25.6 µs	50 µs	104 ms	6.4 µs
DS2210_TIMING_RANGE9	51.2 µs	51.2 µs	209 ms	12.8 µs
DS2210_TIMING_RANGE10	103 µs	103 µs	419 ms	25.6 µs
DS2210_TIMING_RANGE11	205 µs	205 µs	838 ms	51.2 µs
DS2210_TIMING_RANGE12	410 µs	410 µs	1.68 s	103 µs
DS2210_TIMING_RANGE13	820 µs	820 µs	3.36 s	205 µs
DS2210_TIMING_RANGE14	1.64 ms	1.64 µs	6.71 s	410 µs
DS2210_TIMING_RANGE15	3.28 ms	3.28 ms	13.4 s	820 µs
DS2210_TIMING_RANGE16	6.55 ms	6.55 ms	26.8 s	1.64 ms

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

Mode	Meaning
DS2210_D2PWM	PWM signal generation
DS2210_D2F_LOW	Square-wave signal generation, the output is set to low level.
DS2210_D2F_HIGH	Square-wave signal generation, the output is set to high level.
DS2210_D2F_HOLD	Square-wave signal generation, the output keeps the current signal level (low or high).

Note

The `mode` parameter for square-wave signal generation defines the output behavior when frequency < f_{min}.

Messages

The following error message is predefined:

Error message	Meaning
ds2210_timing_out_mode_set: DS2210 board revision 4 or higher required!	The frequency mode is supported only for DS2210 boards with board revision 4 and higher and a board revision 3 with a FPGA revision 3 and higher.

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

ds2210_timing_in_mode_set

Syntax

```
void ds2210_timing_in_mode_set(
      Int32 base,
      Int32 channel,
      Int32 range,
      Int32 mode)
```

Include file

`Ds2210.h`

Purpose

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

Note

- The frequency mode is only available on boards, which support extended functionality. The function checks this restriction and exits with an error message if extended functionality is not supported by the hardware.
- 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 DS2210 board.

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_PWMIN_CH1	PWM channel 1
...	...
DS2210_PWMIN_CH8	PWM channel 8

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
DS2210_TIMING_RANGE1	9.54 Hz	20 kHz	50 ns
DS2210_TIMING_RANGE2	4.77 Hz	20 kHz	100 ns
DS2210_TIMING_RANGE3	2.39 Hz	20 kHz	200 ns
DS2210_TIMING_RANGE4	1.20 Hz	20 kHz	400 ns
DS2210_TIMING_RANGE5	0.60 Hz	20 kHz	800 ns
DS2210_TIMING_RANGE6	0.30 Hz	20 kHz	1.6 µs
DS2210_TIMING_RANGE7	0.15 Hz	20 kHz	3.2 µs
DS2210_TIMING_RANGE8	75 mHz	20 kHz	6.4 µs
DS2210_TIMING_RANGE9	38 mHz	19.53 kHz	12.8 µs
DS2210_TIMING_RANGE10	19 mHz	9.76 kHz	25.6 µs
DS2210_TIMING_RANGE11	10 mHz	4.88 kHz	51.2 µs
DS2210_TIMING_RANGE12	5.0 mHz	2.44 kHz	103 µs
DS2210_TIMING_RANGE13	2.5 mHz	1.22 kHz	205 µs
DS2210_TIMING_RANGE14	1.2 mHz	610.35 Hz	410 µs
DS2210_TIMING_RANGE15	0.6 mHz	305.17 Hz	820 µs
DS2210_TIMING_RANGE16	0.3 mHz	152.59 Hz	1.64 ms

The following symbols are predefined in PWM mode (16 bit resolution):

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
DS2210_TIMING_RANGE1	200 ns	50 µs	3.27 ms	50 ns
DS2210_TIMING_RANGE2	400 ns	50 µs	6.55 ms	100 ns
DS2210_TIMING_RANGE3	800 ns	50 µs	13.1 ms	200 ns
DS2210_TIMING_RANGE4	1.6 µs	50 µs	26.2 ms	400 ns
DS2210_TIMING_RANGE5	3.2 µs	50 µs	52.4 ms	800 ns
DS2210_TIMING_RANGE6	6.4 µs	50 µs	104 ms	1.6 µs
DS2210_TIMING_RANGE7	12.8 µs	50 µs	209 ms	3.2 µs
DS2210_TIMING_RANGE8	25.6 µs	50 µs	419 ms	6.4 µs
DS2210_TIMING_RANGE9	51.2 µs	51.2 µs	838 ms	12.8 µs
DS2210_TIMING_RANGE10	103 µs	103 µs	1.67 s	25.6 µs
DS2210_TIMING_RANGE11	205 µs	205 µs	3.35 s	51.2 µs
DS2210_TIMING_RANGE12	410 µs	410 µs	6.71 s	103 µs
DS2210_TIMING_RANGE13	820 µs	820 µs	13.4 s	205 µs
DS2210_TIMING_RANGE14	1.64 ms	1.64 ms	26.8 s	410 µs
DS2210_TIMING_RANGE15	3.28 ms	3.28 ms	53.6 s	820 µs
DS2210_TIMING_RANGE16	6.55 ms	6.55 ms	107.3 s	1.64 ms

The following symbols are predefined in PWM mode (14 bit resolution):

Predefined Symbol	Minimum Period		Maximum Period	Resolution
	Theoretical	Practical		
DS2210_TIMING_RANGE1	200 ns	50 µs	820 µs	50 ns
DS2210_TIMING_RANGE2	400 ns	50 µs	1.64 ms	100 ns
DS2210_TIMING_RANGE3	800 ns	50 µs	3.27 ms	200 ns
DS2210_TIMING_RANGE4	1.6 µs	50 µs	6.55 ms	400 ns
DS2210_TIMING_RANGE5	3.2 µs	50 µs	13.1 ms	800 ns
DS2210_TIMING_RANGE6	6.4 µs	50 µs	26.2 ms	1.6 µs
DS2210_TIMING_RANGE7	12.8 µs	50 µs	52.4 ms	3.2 µs
DS2210_TIMING_RANGE8	25.6 µs	50 µs	104 ms	6.4 µs
DS2210_TIMING_RANGE9	51.2 µs	51.2 µs	209 ms	12.8 µs
DS2210_TIMING_RANGE10	103 µs	103 µs	419 ms	25.6 µs
DS2210_TIMING_RANGE11	205 µs	205 µs	838 ms	51.2 µs
DS2210_TIMING_RANGE12	410 µs	410 µs	1.68 s	103 µs
DS2210_TIMING_RANGE13	820 µs	820 µs	3.36 s	205 µs
DS2210_TIMING_RANGE14	1.64 ms	1.64 µs	6.71 s	410 µs
DS2210_TIMING_RANGE15	3.28 ms	3.28 ms	13.4 s	820 µs
DS2210_TIMING_RANGE16	6.55 ms	6.55 ms	26.8 s	1.64 ms

Note

Signal periods and resolution

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

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

Mode	Meaning
`DS2210_PWM2D`	PWM signal measurement
`DS2210_F2D`	Frequency measurement

Messages

The following error message is predefined:

Error message	Meaning
ds2210_timing_in_mode_set: DS2210 board revision 4 or higher required!	The frequency mode is supported only for DS2210 boards with board revision 4 and higher and a board revision 3 with a FPGA revision 3 and higher.

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

PWM Signal Generation

Purpose

To generate standard PWM signals.

Where to go from here

Information in this section

Information in other sections

PWM Signal Generation (DS2210 Features)
Six independent PWM outputs are available for the generation of nonnegative square-wave signals.

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 <Ds2210.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 */
   ds2210_pwm_out(DS2210_1_BASE, DS2210_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 */
   ds2210_init(DS2210_1_BASE);   /* initialize DS2210 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* enable digital output driver */
   ds2210_digout_mode_set(DS2210_1_BASE,
                        DS2210_DIGOUT_ENABLE);
   /* set PWM range for output channel 1 */
   ds2210_timing_out_mode_set (DS2210_1_BASE,
                            DS2210_PWMOUT_CH1,
                            DS2210_PWM_RANGE5,
                            DS2210_D2PWM);
   /* set values for PWM generation */
   ds2210_pwm_out(DS2210_1_BASE, DS2210_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

ds2210_digout_mode_set 21
ds2210_init 17
ds2210_pwm_out 64
ds2210_timing_out_mode_set 54
init 212

ds2210_pwm_out_range_set (obsolete)

Syntax

```
void ds2210_pwm_out_range_set(
      Int32 base,
      Int32 channel,
      Int32 range)
```

Include file

`Ds2210.h`

Purpose

To select the period range for the specified PWM output channel. The prescaler of the specified PWM output channel is set according to the specified range.

I/O mapping

For information on the I/O mapping, refer to PWM Signal Generation (DS2210 Features 🕮).

Note

This function is obsolete. To ensure compatibility with old software versions, a macro is defined in `Ds2210.h`. The macro replaces a call of this function with the `ds2210_timing_out_mode_set` function. The additional `mode` parameter, needed by the new function, is set to `DS2210_D2PWM` to enable PWM signal generation.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_PWMOUT_CH1`	PWM channel 1
...	...
`DS2210_PWMOUT_CH6`	PWM channel 6

range PWM period range. For a detailed description on the ranges, refer to ds2210_timing_out_mode_set on page 54. If you use this table you can take the timing mode entry (`DS2210_TIMING_RANGEx`) for the corresponding PWM range (`DS2210_PWM_RANGEx`).

For further information on PWM signal generation and the restrictions, refer to PWM Signal Generation (DS2210 Features 🕮).

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_pwm_out

Syntax

```
void ds2210_pwm_out(
      Int32 base,
      Int32 channel,
      dsfloat period,
      dsfloat duty)
```

Include file

`Ds2210.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 (DS2210 Features 📖).

Note

- After initialization, the outputs are disabled. Use `ds2210_digout_mode_set` to enable the digital output ports.
- The function supports 16-bit PWM resolution. The PWM period that can be achieved differs depending on whether the board supports 14-bit or 16-bit resolution.
- 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 (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_PWMOUT_CH1`	PWM channel 1
...	...
`DS2210_PWMOUT_CH6`	PWM channel 6

period PWM period in seconds. Values should remain within the selected period range (refer to ds2210_timing_out_mode_set). For information on PWM signal generation and its restrictions, refer to PWM Signal Generation (DS2210 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 385.

Related topics

Examples

PWM Signal Measurement

Basics

For information on PWM signal measurement and its I/O mapping, refer to PWM Signal Measurement (DS2210 Features 📖).

Where to go from here

Information in this section

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 <Ds2210.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 */
   ds2210_pwm_in(DS2210_1_BASE, DS2210_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 */
   ds2210_init(DS2210_1_BASE);           /* initialize DS2210 board */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   ds2210_digout_mode_set(DS2210_1_BASE, DS2210_DIGOUT_ENABLE);
   /* set PWM range for expected signal periods on input channel 1 */
   ds2210_timing_in_mode_set(DS2210_1_BASE, DS2210_PWMIN_CH1,
                             DS2210_PWM_RANGE5, DS2210_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

ds2210_pwm_in_range_set (obsolete)

Syntax

```
void ds2210_pwm_in_range_set(
     Int32 base,
     Int32 channel,
     Int32 range)
```

Include file

`Ds2210.h`

Purpose

To select the period range for the specified PWM input channel. The prescaler of the specified PWM input channel is set according to the specified range.

Description

Note

This function is obsolete. To ensure compatibility with old software versions, a macro is defined in `Ds2210.h`. The macro replaces a call of this function with the `ds2210_timing_in_mode_set` function. The additional `mode` parameter, needed by the new function, is set to DS2210_PWM2D to enable PWM signal measurement.

I/O mapping

For information on the I/O mapping, refer to PWM Signal Measurement (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Specifies the PWM input channel. The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_PWMIN_CH1	PWM input channel 1
…	…
DS2210_PWMIN_CH8	PWM input channel 8

range Specifies the PWM period range. For a detailed description on the ranges, refer to ds2210_timing_out_mode_set on page 54. If you use this table you can take the timing mode entry (DS2210_TIMING_RANGEx) for the corresponding PWM range (DS2210_PWM_RANGEx).

For further information on PWM signal generation and the restrictions, refer to PWM Signal Measurement (DS2210 Features 📖).

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

ds2210_pwm_in

Syntax

```
void ds2210_pwm_in(
      Int32 base,
      Int32 channel,
      dsfloat *period,
      dsfloat *duty)
```

Include file `Ds2210.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 (DS2210 Features 📖).

Note

- The inputs are 12-V compatible. After initialization, the input threshold is set to 2.5 V. Use `ds2210_digin_threshold_set` to set the threshold within the range of 1.0 … 7.0 V.
- The function supports 16-bit PWM resolution. The PWM period that can be achieved differs depending on whether the board supports 14 bit or 16 bit resolution.
- 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 (DS2210 Features 📖).
- The PWM input channels 7 and 8 are shared with injection capture. This feature is only supported for boards with extended functionality. If you use these channels for injection capture, you cannot use it for PWM signal measurement, refer to ds2210_injection_capture_read on page 121.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Specifies the PWM input channel. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_PWMIN_CH1`	PWM input channel 1
…	…
`DS2210_PWMIN_CH8`	PWM input channel 8

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 385.
Related topics	Examples Example of PWM Signal Measurement........ 66 References ds2210_digin_threshold_set........ 20

Square-Wave Signal Generation

Purpose

To generate square-wave signals.

Where to go from here

Information in this section

Information in other sections

Square-Wave Signal Generation (DS2210 Features)
Six 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 <Ds2210.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 */
   ds2210_d2f(DS2210_1_BASE, DS2210_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 */
   ds2210_init(DS2210_1_BASE);                     /* initialize DS2210 board */
```

```
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
                                          /* enable digital output driver */
   ds2210_digout_mode_set(DS2210_1_BASE,
                          DS2210_DIGOUT_ENABLE);
            /* set frequency range for output channel 1, output(0Hz) = low */
   ds2210_timing_out_mode_set(DS2210_1_BASE,
                              DS2210_PWMOUT_CH1,
                              DS2210_TIMING_RANGE5,
                              DS2210_D2F_LOW);
                                          /* set values for PWM generation */
  ds2210_d2f (DS2210_1_BASE, DS2210_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

ds2210_d2f

Syntax

```
void ds2210_d2f(
      Int32 base,
      Int32 channel,
      dsfloat frequency)
```

Include file

`Ds2210.h`

Purpose

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

I/O mapping

For information on the I/O mapping, refer to Square-Wave Signal Generation (DS2210 Features 📖).

Note

- Do not use this function if the board does not support the extended mode. The function does not check this restriction. You are responsible for ensuring, that the board supports the extended functionality.
- All digital outputs are high-Z out of reset. Outputs are enabled by the software using the `ds2210_timing_out_mode_set` function.
- To minimize the quantization effect on the frequency resolution, you should select the smallest possible frequency range. For detailed information, refer to Square-Wave Signal Generation (DS2210 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 Square-Wave Signal Generation (DS2210 Features 📖).

Note

The frequency ranges can be set using the `ds2210_timing_out_mode_set` function.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel PWM channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_PWMOUT_CH1	PWM channel 1
...	...
DS2210_PWMOUT_CH6	PWM channel 6

frequency Frequency of the generated signal in Hz

Related topics

Examples

Example of Square-Wave Signal Generation........71

References

ds2210_timing_out_mode_set........54

Frequency Measurement

Purpose

To measure the frequency of square-wave signals.

Where to go from here

Information in this section

Information in other sections

Frequency Measurement (DS2210 Features)
Eight independent channels are available to measure the frequency of square-wave signals.

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 <Ds2210.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 */
   ds2210_f2d(DS2210_1_BASE, DS2210_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 */
   ds2210_init(DS2210_1_BASE);                     /* initialize DS2210 board */
```

```
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
                                       /* enable digital output driver */
   ds2210_digout_mode_set(DS2210_1_BASE,
                              DS2210_DIGOUT_ENABLE);
                                /* set frequency range for input channel 1 */
   ds2210_timing_in_mode_set(DS2210_1_BASE,
                                 DS2210_PWMIN_CH1,
                                 DS2210_TIMING_RANGE5,
                                 DS2210_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

ds2210_digout_mode_set....................21
ds2210_f2d....................75
ds2210_init....................17
ds2210_timing_in_mode_set....................57

ds2210_f2d

Syntax

```
void ds2210_f2d(
      Int32 base,
      Int32 channel,
      dsfloat* frequency)
```

Include file

`Ds2210.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 (DS2210 Features 📖).

Note

- Do not use this function if the hardware does not support the extended mode. The function does not check this restriction. You are responsible for ensuring that the board supports the extended mode.
- The input is 12-V compatible. The threshold can be set within a range of 1 … 7 V by using the `ds2210_digin_threshold_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 (DS2210 Features 📖).

Description

The function measures the signal frequency of the specified input channel. The frequency value is scaled to Hz and is written to the memory at the address given 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 (DS2210 Features 📖).

Note

The frequency ranges can be set using the `ds2210_timing_in_mode_set` function.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Specifies the PWM input channel. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_PWMIN_CH1`	PWM input channel 1
…	…
`DS2210_PWMIN_CH8`	PWM input channel 8

frequency Specifies the address where frequency is written in Hz

Related topics

Examples

References

Angular Processing Unit (APU)

Introduction

For general information on the angular processing unit, refer to Angular Processing Unit (DS2210 Features 📖).

Where to go from here

Information in this section

Overall APU Functions

Basics

For general information on the angular processing unit, refer to Angular Processing Unit (DS2210 Features 📖).

Where to go from here

Information in this section

ds2210_apu_position_write

Syntax

```
void ds2210_apu_position_write(
      Int32 base,
      dsfloat pos)
```

Include file

`Ds2210.h`

Purpose

To set the initial engine position.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2210 Features 📖).

Description

`ds2210_apu_position_write` checks the APU status and sets the given engine position only if the APU is stopped. The function has no effect on boards being configured as a slave.

Parameters

base Specifies the PHS-bus base address of the DS2210 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	Meaning
Error	ds2210_apu_position_write(board_offset): No access while APU is running!	You have to stop the APU with `ds2210_apu_stop` before writing the APU position.

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_apu_position_read

Syntax

```
void ds2210_apu_position_read(
      Int32 base,
      dsfloat *pos)
```

Include file

`Ds2210.h`

Purpose

To read the current engine position.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2210 Features 📖).

> **Note**
>
> For cascaded DS2210 boards, this function returns the angle position of the master board.

Parameters

base Specifies the PHS-bus base address of the DS2210 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 385.

Related topics

References

ds2210_apu_velocity_write

Syntax

```
void ds2210_apu_velocity_write(
      Int32 base,
      dsfloat vel)
```

Include file

`Ds2210.h`

Purpose

To update the APU angle velocity. The function has no effect on boards configured as a slave.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

vel Angle velocity in rad/s within the range of –3068 … +3068.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

ds2210_int_position_set

Syntax

```
void ds2210_int_position_set(
      Int32 base,
      Int32 channel,
      Int32 count,
      dsfloat* pos)
```

Include file

`Ds2210.h`

Purpose

To define interrupt positions for the given cylinder.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2210 Features 📖).

Description

`ds2210_int_position_set` checks the APU status and sets the interrupt positions for the given 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.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Cylinder number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_INTPOS_CYL1`	Cylinder 1
…	…
`DS2210_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 0.0015 radians (rad). Two subsequent interrupts have to be set with a distance of 0.006 radians (rad).

Return value

None

Error message

The following error messages are defined:

Type	Error Message	Meaning
Error	ds2210_int_position_set(board_offset): No access while APU is running!	You have to stop the APU with `ds2210_apu_stop` before setting the interrupt positions.
Error	ds2210_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 385.

Example

```
...
dsfloat positions[5] = {DS2210_RAD(0), DS2210_RAD(90),
                        DS2210_RAD(180), DS2210_RAD(270),
                        DS2210_RAD(360)};
...
/* set interrupt positions for cylinder 1 */
ds2210_int_position_set(DS2210_1_BASE,
                        DS2210_INTPOS_CYL1,
                        5,
                        positions);
...
```

Related topics

References

DS2210_RAD 26

Engine Position Phase Accumulator

Purpose

To start or stop the APU and the signal generation.

Where to go from here

Information in this section

Information in other sections

Engine Position Phase Accumulator (DS2210 Features 📖)
Explaining the unit that supplies the engine position.

ds2210_apu_start

Syntax

```
void ds2210_apu_start(Int32 base)
```

Include file

`Ds2210.h`

Purpose

To start the engine phase accumulation and the APU signal generation.

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

I/O mapping

For information on the I/O mapping, refer to APU Reference (DS2210 Features 📖).

Parameters

base	Specifies the PHS-bus base address of the DS2210 board.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

ds2210_apu_stop

Syntax

```
void ds2210_apu_stop(Int32 base)
```

Include file

`Ds2210.h`

Purpose

To stop the APU signal generation.

I/O mapping

For information on the I/O mapping, refer to APU Reference (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

Examples for Crankshaft Signal Generation..89

References

ds2210_apu_start..86

Crankshaft Sensor Signal Generation

Purpose

To generate crankshaft sensor signals.

Where to go from here

Information in this section

Information in other sections

Crankshaft Sensor Signal Generation (DS2210 Features 📖)
The crankshaft signal generator provides one analog crankshaft output and one digital crankshaft output.

Examples for Crankshaft Signal Generation

Loading wavetable data manually

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

Note

Before you can use the crankshaft signal generator you have to initialize the board and enable the output transformers. Refer to ds2210_init on page 17 and ds2210_apu_transformer_mode_set on page 23).

```
#define TBLLEN 8192
Int32 cr_tbl[TBLLEN];
...
/* initialize wavetable data */
for (i=0; i<TBLLEN; i++)
{
   cr_tbl[i] = ...
}
/* load data to crankshaft wavetable 1 */
ds2210_crank_table_load(DS2210_1_BASE, DS2210_CRANK_TBL1, cr_t
/* select crankshaft wavetable 1 for signal generation */
ds2210_crank_table_select(DS2210_1_BASE, DS2210_CRANK_TBL1);
/* set amplitude of crankshaft signal to +/- 10 V */
ds2210_crank_output_ampl_set(DS2210_1_BASE, 20.0);
/* set engine velocity to 1000 rpm */
ds2210_apu_velocity_write(DS2210_1_BASE, DS2210_RAD_S(1000));
/* start signal generation */
ds2210_apu_start(DS2210_1_BASE);
...
```

Using the MATCONV tool

The wavetables used in the APU demo application are generated with MATLAB and the MATCONV tool.

Declaration of the global labels which allow you to access the wavetable data

```
...
/* wav2210.c */
extern UInt32   wav2210_1_crank1;
...
```

Load wavetable data using the global labels

```
...
/* initialize crankshaft signal generation */
ds2210_crank_table_load(DS2210_1_BASE, DS2210_CRANK_TBL1,
                     (Int32*)&wav2210_1_crank1);
...
```

Related topics

References

ds2210_apu_start 86
ds2210_apu_velocity_write 82
ds2210_crank_output_ampl_set 94
ds2210_crank_table_load 91
ds2210_crank_table_select 92

ds2210_crank_table_load

Syntax

```
void ds2210_crank_table_load(
      Int32 base,
      Int32 table,
      Int32* data)
```

Include file

`Ds2210.h`

Purpose

To load wavetable data for the crankshaft sensor signal generation.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2210 Features).

Description

`ds2210_crank_table_load` checks the APU status and loads wavetable data to one of eight crankshaft wavetables in the DS2210 memory, but only if the APU is stopped. You can load up to 8 wavetables and switch from one table to another while the APU is running.

The following table shows the relationship between wavetable data values and output signals. Please note that the internal representation on the hardware uses an inverted sign bit. If you generate a digital wavetable, you have to define negative values for a low output.

Value	Internal	Analog Output	Digital Output
–128 … –1	0x00 … 0x7F	–20 V … –0.16 V	0 (low)
0 … 128	0x80 … 0xFF	0 V … +20 V	1 (high)

Note

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

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

table Crankshaft wavetable number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_CRANK_TBL1`	Crankshaft wavetable 1
...	...
`DS2210_CRANK_TBL8`	Crankshaft wavetable 8

data Source address of the crankshaft wavetable data

Return value

None

Message

The following messages are defined:

Type	Message	Meaning
Error	ds2210_crank_table_load (board_offset): No access while APU is running!	You have to stop the APU with `ds2210_apu_stop`.
Error	ds2210_crank_table_load (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 385.

Related topics

Examples

References

ds2210_crank_table_select

Syntax

```
void ds2210_crank_table_select(
      Int32 base,
      Int32 table)
```

Include file

`Ds2210.h`

Purpose

To select a wavetable for the crankshaft sensor signal generation.

Description

After initialization, the analog transformer outputs are disabled. Use `ds2210_apu_transformer_mode_set` to enable the transformers.

After initialization, the digital crankshaft output is disabled. Use `ds2210_digout_mode_set` to enable the output.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2210 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

table Crankshaft wavetable number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_CRANK_TBL1`	Crankshaft wavetable 1
…	…
`DS2210_CRANK_TBL8`	Crankshaft wavetable 8

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

Examples for Crankshaft Signal Generation..89

References

Digital Outputs (PHS Bus System Hardware Reference 🕮)
ds2210_apu_start..86
ds2210_apu_transformer_mode_set..23
ds2210_crank_table_load..91
ds2210_digout_mode_set..21

ds2210_crank_output_ampl_set

Syntax

```
void ds2210_crank_output_ampl_set(
      Int32 base,
      dsfloat value)
```

Include file

`Ds2210.h`

Purpose

To set the amplitude of the crankshaft output signal.

Description

After initialization, the analog transformer outputs are disabled. Use `ds2210_apu_transformer_mode_set` to enable the transformers.

I/O mapping

For information on the I/O mapping, refer to Crankshaft Sensor Signal Generation (DS2210 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

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 385.

Related topics

Examples

References

Camshaft Sensor Signal Generation

Purpose

To generate camshaft sensor signals.

Where to go from here

Information in this section

Information in other sections

Examples for Camshaft Signal Generation

Loading wavetable data manually

This example shows how to load wavetable 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 and enable the output transformers. Refer to ds2210_init on page 17 and ds2210_apu_transformer_mode_set on page 23).

```
#define TBLLEN 8192
Int32 cam_tbl[TBLLEN];
…
/* initialize wavetable data */
for (i=0; i<TBLLEN; i++)
{
   cam_tbl[i] = ...
}
/* load data to wavetable 1 for both camshafts */
ds2210_cam_table_load(DS2210_1_BASE, DS2210_CAM_CHA,
                      DS2210_CAM_TBL1, cam_tbl);
ds2210_cam_table_load(DS2210_1_BASE, DS2210_CAM_CHB,
                      DS2210_CAM_TBL1, cam_tbl);
/* select camshaft wavetable 1 for both camshafts */
ds2210_cam_table_select(DS2210_1_BASE, DS2210_CAM_CHA,
                        DS2210_CAM_TBL1);
ds2210_cam_table_select(DS2210_1_BASE, DS2210_CAM_CHB,
                        DS2210_CAM_TBL1);
/* set camshaft amplitude of both camshafts to +/- 10 V */
ds2210_cam_output_ampl_set(DS2210_1_BASE, DS2210_CAM_CHA, 20.0);
ds2210_cam_output_ampl_set(DS2210_1_BASE, DS2210_CAM_CHB, 20.0);
/* set camshaft phase offsets */
ds2210_cam_phase_write(DS2210_1_BASE, DS2210_CAM_CHA, 0.15);
ds2210_cam_phase_write(DS2210_1_BASE, DS2210_CAM_CHB, 0.20);
/* set engine velocity to 1000 rpm */
ds2210_apu_velocity_write(DS2210_1_BASE, DS2210_RAD_S(1000));
/* start signal generation */
ds2210_apustart(DS2210_1_BASE);
…
```

Read the phase offset.

```
dsfloat phase1;
…
/* read phase offset of camshaft 1 */
ds2210_cam_phase_read(DS2210_1_BASE, DS2210_CAM_CHA, &phase1);
…
```

Using the MATCONV tool

The wavetables used in the APU demo application are generated with MATLAB and the MATCONV tool.

Declaration of the global labels which allow you to access the wavetable data

```
…
/* wav2210.c */
…
extern UInt32   wav2210_1_camA1;
extern UInt32   wav2210_1_camB1;
…
```

Load wavetable data using the global labels

```
…
/* initialize camshaft signal generation */
ds2210_cam_table_load(DS2210_1_BASE, DS2210_CAM_CHA,
                      DS2210_CAM_TBL1, (Int32*)&wav2210_1_camA1);
```

ds2210_cam_table_load

Syntax

```
void ds2210_cam_table_load(
      Int32 base,
      Int32 channel,
      Int32 table,
      Int32 *data)
```

Include file

`Ds2210.h`

Purpose

To load wavetable data for camshaft sensor signal generation.

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2210 Features 📖).

Description

`ds2210_cam_table_load` checks the APU status and loads wavetable data to one of eight camshaft wavetables in the DS2210 memory, but only if the APU is stopped.

The following table shows the relationship between wavetable data values and output signals. Please note that the internal representation on the hardware uses an inverted sign bit. If you generate a digital wavetable, you have to define negative values for a low output.

Value	Internal	Analog Output	Digital Output
–128 … –1	0x00 … 0x7F	–20 V … –0.16 V	0 (low)
0 … 128	0x80 … 0xFF	0 V … +20 V	1 (high)

Note

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

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_CAM_CHA`	Camshaft 1
`DS2210_CAM_CHB`	Camshaft 2

table Camshaft wavetable number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_CAM_TBL1`	Camshaft wavetable 1
...	...
`DS2210_CAM_TBL8`	Camshaft wavetable 8

data Source address of the wavetable data

Return value

None

Message

The following messages are defined:

Type	Message	Meaning
Error	ds2210_cam_table_load (board_offset): No access while APU is running!	You have to stop the APU with `ds2210_apu_stop` before loading wavetable data.
Error	ds2210_cam_table_load (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 385.

Related topics

Examples

References

ds2210_cam_table_select

Syntax

```
void ds2210_cam_table_select(
      Int32 base,
      Int32 channel,
      Int32 table)
```

Include file

`Ds2210.h`

Purpose

To select a wavetable for camshaft sensor signal generation.

Description

After initialization, the analog transformer outputs are disabled. Use `ds2210_apu_transformer_mode_set` to enable the transformers.

After initialization, the digital camshaft output is disabled. Use `ds2210_digout_mode_set` to enable the output.

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2210 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_CAM_CHA`	Camshaft 1
`DS2210_CAM_CHB`	Camshaft 2

table Camshaft wavetable number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_CAM_TBL1`	Camshaft wavetable 1
…	…
`DS2210_CAM_TBL8`	Camshaft wavetable 8

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

ds2210_cam_output_ampl_set

Syntax

```
void ds2210_cam_output_ampl_set(
      Int32 base,
      Int32 channel,
      dsfloat value)
```

Include file

`Ds2210.h`

Purpose

To set the amplitude of the camshaft output signal.

Description

After initialization, the analog transformer outputs are disabled. Use `ds2210_apu_transformer_mode_set` to enable the transformers.

I/O mapping

For information on the I/O mapping, refer to Camshaft Sensor Signal Generation (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_CAM_CHA	Camshaft 1
DS2210_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 385.

Related topics

References

ds2210_cam_phase_write

Syntax

```
void ds2210_cam_phase_write(
      Int32 base,
      Int32 channel,
      dsfloat phase)
```

Include file

`Ds2210.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 (DS2210 Features).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_CAM_CHA	Camshaft 1
DS2210_CAM_CHB	Camshaft 2

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

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

ds2210_cam_phase_read

Syntax

```
void ds2210_cam_phase_read(
      Int32 base,
      Int32 channel,
      dsfloat *phase)
```

Include file

`Ds2210.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 (DS2210 Features).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Camshaft number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_CAM_CHA`	Camshaft 1
`DS2210_CAM_CHB`	Camshaft 2

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 385.

Related topics

Examples

References

Event Capture Windows

Purpose

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 will be captured.

Where to go from here

Information in this section

Information in other sections

ds2210_event_window_set

Syntax

```
void ds2210_event_window_set(
      Int32 base,
      Int32 channel,
      dsfloat start_pos,
      dsfloat end_pos)
```

Include file

`Ds2210.h`

Purpose

To define ignition capture, auxiliary capture or injection event capture windows. Pulses will be captured only if they occur within the event window.

Description

For basic information on event windows, refer to Event Capture Windows (DS2210 Features 📖).

> **Note**
>
> The following restrictions apply for the event windows:
> - The function is non-reentrant.
> - An event window can cover the whole engine cycle of 0 … 720°. But in this case, no window borders and pseudo edges are detected. Refer to ds2210_ignition_capture_read on page 113.
> - The minimum width of an event window is 0.003 rad (= 0.176°).
> - If you specify the same value for the start and the end position no event window will be set.
> - The minimum distance between two event windows is two APU position steps (2 · 4π/8192). Thus the capture window must not exceed 4π · (1 - 2/8192) or 719.8°.
> - The additional injection capture channels 7 and 8 are supported only with extended functionality. If your board does not support extended functionality, the function exits with an error message.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2210 Features 📖) and Injection Pulse Position and Fuel Amount Measurement (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_EVWIN_IGNCAP1`	Ignition event capture window for cylinder 1
…	…
`DS2210_EVWIN_IGNCAP6`	Ignition event capture window for cylinder 6
`DS2210_EVWIN_AUXCAP1`	Auxiliary event capture window 1
`DS2210_EVWIN_AUXCAP2`	Auxiliary event capture window 2
`DS2210_EVWIN_INJCAP1`	Injection event capture window for cylinder 1
…	…
`DS2210_EVWIN_INJCAP6`	Injection event capture window for cylinder 6
`DS2210_EVWIN_INJCAP7`	Injection event capture window for cylinder 7[1)]
`DS2210_EVWIN_INJCAP8`	Injection event capture window for cylinder 8[1)]

[1)] Only for baords with extended functionality

start_pos Window start position in rad within the range of 0 … 4π. The values can be given with a resolution of 0.0015 radians (rad).

end_pos Window end position in rad within the range of 0 … 4π. The values can be given with a resolution of 0.0015 radians (rad).

Return value

None

Messages

The following messages are defined:

Type	Message	Meaning
Error	ds2210_event_window_set(board_offset): No access while APU is running!	You have to stop the APU with `ds2210_apu_stop` before setting the event windows.
Error	ds2210_event_window_set(board_offset): Memory access error!	The function failed due to a memory access error.
Error	ds2210_event_window_set: DS2210 board rev. 4 or higher required!	Additional injection channels are supported only for DS2210 boards with board revision 4 and higher and a board revision 3 with a FPGA revision 3 and higher. The revision number is printed on the board. You can also read out the number with the experiment software.

Example

This example shows how to define an ignition event capture window for cylinder 1:

```
...
/* define ignition event capture window for cylinder 1 */
ds2210_event_window_set(DS2210_1_BASE, DS2210_EVWIN_IGNCAP1,
                        DS2210_RAD(705), DS2210_RAD(35));
...
```

Related topics

Basics

Event Capture Windows (DS2210 Features 📖)

References

ds2210_apu_stop....87
ds2210_ignition_capture_read....113
ds2210_injection_capture_read....121

Spark Event Capture

Purpose

To measure ignition pulses.

Note

If your board supports extended functionality, the spark event capture unit can also be used for injection event capture. To check whether your board supports extended functionality, refer to ds2210_init on page 17.

Where to go from here

Information in this section

Information in other sections

Spark Event Capture (DS2210 Features 🕮)
The spark event capture unit provides 6 digital ignition inputs and 2 digital auxiliary capture inputs.

ds2210_ign_capture_mode_set

Syntax

```
void ds2210_ign_capture_mode_set(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 count)
```

Include file

`Ds2210.h`

Purpose

To set the ignition capture mode.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2210 Features 🕮).

Note

Some mode settings are only allowed if the hardware supports extended functionality. If the board does not support extended functionality, the function exits with an error message.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

edge Selects high active or low active input pulses. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_IGNCAP_RISE_EDGE`	High active input pulses. Rising edges will be captured as leading edges.
`DS2210_IGNCAP_FALL_EDGE`	Low active input pulses. Falling edges will be captured as leading edges.

mode Capture mode. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_IGNCAP_ALL_EVENT`	All ignition pulses within the event window will be captured.
`DS2210_IGNCAP_FIRST_EVENT`	The position of the leading edge of the first input pulse within the event window will be captured.
`DS2210_IGNCAP_PULSE_POS`	All event positions including pseudo event generation are captured.[1)]

Predefined Symbol	Meaning
`DS2210_IGNCAP_PULSE_DUR`	The start position and duration (4 µs resolution) are captured.[1)]
`DS2210_IGNCAP_PULSE_DUR_HRES`	The start position and duration (1 µs resolution) are captured.[1)]

[1)] Only for boards with extended functionality

count Number of expected events within the range of 1 … 8

Note

If you use `ds2210_ignition_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by the `count` parameter of the function.

Return value

The following error message is defined:

Error message	Meaning
ds2210_ign_capture_mode_set: DS2210 board rev. 4 or higher required!	Additional modes such as DS2210_IGNCAP_PULSE_DUR are supported only for DS2210 boards with board revision 4 and higher and a board revision 3 with a FPGA revision 3 and higher. The revision number is printed on the board. You can also read out the number with the experiment software.

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_aux1_capture_mode_set

Syntax

```
void ds2210_aux1_capture_mode_set(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 count)
```

Include file

`Ds2210.h`

Purpose

To set the capture mode for auxiliary channel 1.

Description

Some mode settings are only allowed if the hardware supports extended functionality. If the board does not support extended functionality, the function exits with an error message.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

edge Selects high active or low active input pulses. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_AUX1CAP_RISE_EDGE`	High active input pulses. Rising edges will be captured as leading edges.
`DS2210_AUX1CAP_FALL_EDGE`	Low active input pulses. Falling edges will be captured as leading edges.

mode Capture mode. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_AUX1CAP_ALL_EVENT`	All pulses within the event window will be captured.
`DS2210_AUX1CAP_FIRST_EVENT`	The position of the leading edge of the first input pulse within the event window will be captured.
`DS2210_AUX1CAP_PULSE_POS`	All event positions including pseudo event generation are captured.[1)]
`DS2210_AUX1CAP_PULSE_DUR`	The start position and duration (4 µs resolution) are captured.[1)]
`DS2210_AUX1CAP_PULSE_DUR_HRES`	The start position and duration (1 µs resolution) are captured.[1)]

[1)] Only for boards with extended functionality

count Number of expected events within the range of 1 … 8

Note

If you use `ds2210_ignition_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by the `count` parameter of the function.

Return value

The following error messages are defined:

Error message	Meaning
ds2210_aux1_capture_mode_set: DS2210 board rev. 4 or higher required!	Additional modes such as DS2210_AUX1CAP_PULSE_DUR are supported only for DS2210 boards with board revision 4 and higher and a board revision 3 with a FPGA revision 3 and higher. The revision number is printed on the board. You can also read out the number with the experiment software.

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_aux2_capture_mode_set

Syntax

```
void ds2210_aux2_capture_mode_set(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 count)
```

Include file

`Ds2210.h`

Purpose

To set the capture mode for the auxiliary channel 2.

Description

Some mode settings are only allowed if the hardware supports extended functionality. If the board does not support extended functionality, the function exits with an error message.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2210 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

edge Selects high active or low active input pulses. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_AUX2CAP_RISE_EDGE`	High active input pulses. Rising edges will be captured as leading edges.
`DS2210_AUX2CAP_FALL_EDGE`	Low active input pulses. Falling edges will be captured as leading edges.

mode Capture mode. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_AUX2CAP_ALL_EVENT`	All pulses within the event window will be captured.
`DS2210_AUX2CAP_FIRST_EVENT`	The position of the leading edge of the first input pulse within the event window will be captured.
`DS2210_AUX2CAP_PULSE_POS`	All event positions including pseudo event generation are captured.[1)]
`DS2210_AUX2CAP_PULSE_DUR`	The start position and duration (4 µs resolution) are captured.[1)]
`DS2210_AUX2CAP_PULSE_DUR_HRES`	The start position and duration (1 µs resolution) are captured.[1)]

[1)] Only for boards with extended functionality

count Number of expected events within the range of 1 … 8

> **Note**
>
> If you use `ds2210_ignition_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by the `count` parameter of the function.

Return value

The following error messages are defined:

Error message	Meaning
ds2210_aux2_capture_mode_set: DS2210 board rev. 4 or higher required!	Additional modes such as DS2210_AUX2CAP_PULSE_DUR are supported only for DS2210 boards with board revision 4 and higher or a board revision 3 with a FPGA revision 3 and higher. The revision number is printed on the board. You can also read out the number with the experiment software

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_aux1_capture_mode_set 110
ds2210_ign_capture_mode_set 108
ds2210_ignition_capture_read 113
ds2210_ignition_fifo_read 115

ds2210_ignition_capture_read

Syntax

```
void ds2210_ignition_capture_read(
      Int32 base,
      Int32 channel,
      Int32 *count,
      dsfloat *start_pos,
      dsfloat *end_pos)
```

Include file

`Ds2210.h`

Purpose

To read the ignition pulse position(s) and durations from the last event window.

> **Note**
>
> This function conflicts with the `ds2210_ignition_fifo_read` function, because both functions access the ignition capture FIFO. Using these functions simultaneously causes unpredictable results.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2210 Features 📖).

Description

The ignition positions and/or durations captured within 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 within the last event window. You can use the ignition channels 1 … 6 and the auxiliary channels 1 and 2 for ignition angle and duration capture.

Note

- If the number of captured events is less than the number of expected events specified by `ds2210_ign_capture_mode_set`, `ds2210_aux1_capture_mode_set`, or `ds2210_aux2_capture_mode_set`, the function returns negative position values for the missing events.
- Events will be captured up to the number of expected events. Additional events will be ignored.
- If the event window covers the range 0 … 719.82°, an input pulse overlapping the event window border is detected as 2 pulses due to the pseudo edges of the start and end positions of the window (only using the modes 2, 3 or 4, refer to ds2210_ign_capture_mode_set). If the whole range (0 … 720°) is used, no pseudo edges of the start and end positions are detected.

Note

The inputs are 12 V compatible. After initialization, the input threshold is set to 2.5 V. Use `ds2210_digin_threshold_set` to set the threshold within the range of 1.0 … 7.0 V. For further information, refer to Digital Inputs (PHS Bus System Hardware Reference 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

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

Predefined Symbol	Meaning
`DS2210_IGNCAP_CH1`	Ignition capture channel 1
…	…
`DS2210_IGNCAP_CH6`	Ignition capture channel 6
`DS2210_AUXCAP_CH1`	Auxiliary capture channel 1
`DS2210_AUXCAP_CH2`	Auxiliary capture channel 2

count Specifies the address where the number of captured events 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 Specifies the address where angle position values of the trailing edge of the pulse are written within the range 0 … 720°.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_ignition_fifo_read

Syntax

```
Int32 ds2210_ignition_fifo_read(
      Int32 base,
      Int32 channel,
      Int32 count,
      Int32 *state,
      dsfloat *pos,
      Int32 *length)
```

Include file

`Ds2210.h`

Purpose

To read the ignition pulse positions/durations captured during run time in the FIFO. The values of the specified channel are returned through the `pos` [deg/sec.] array. You can use the ignition channels 1 … 6 and the auxiliary channels 1 and 2 for ignition angle and duration capture.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2210 Features 🕮).

Description

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 through 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 32 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 ignition capture modes:

Capture Modes (XXX = IGN or AUX)	Meaning
DS2210_XXXCAP_ALL_EVENT	All pulse positions (start and end) are captured in degrees. Regardless of the event window settings *no* dummy events are detected.
DS2210_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.
DS2210_XXXCAP_PULSE_POS	Only the angle positions of the captured events are read in degrees. Under certain circumstances pseudo edges are generated.
DS2210_XXXCAP_PULSE_DUR DS2210_XXXCAP_PULSE_DUR_HRES	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 (HRES). Under certain circumstances pseudo edges are generated.

Note

If the leading or trailing edge of an input pulse lies outside the capture window while 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 DS2210 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_IGNCAP_CH1	Ignition capture channel 1
...	...
DS2210_IGNCAP_CH6	Ignition capture channel 6
DS2210_AUXCAP_CH1	Auxiliary capture channel 1
DS2210_AUXCAP_CH2	Auxiliary capture channel 2

count Specifies the number of events to be read. The maximum number must not exceed the internal FIFO size (32 events).

state Address where the state of the ignition events are written.

Capture State (XXX = IGN or AUX)	State	Meaning
DS2210_XXXCAP_RISE_EDGE	0	Falling edge/low active
	1	Rising edge/high active
DS2210_XXXCAP_FALL_EDGE	0	Rising edge/high active
	1	Falling edge/low active

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 Specifies the 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, an overflow of the FIFO 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:

Fifo level	Meaning
Fifo level = 0	The FIFO is empty, all events were read.
Fifo > 0	The number of events remaining in the FIFO.
Fifo level = –1	A FIFO overflow occurred.

Related topics

References

ds2210_aux1_capture_mode_set 110
ds2210_aux2_capture_mode_set 111
ds2210_event_window_set 104
ds2210_ign_capture_mode_set 108

ds2210_ignition status_read

Syntax

```
void ds2210_ignition_status_read(
      Int32 base,
      Int32 *channels,
      Int32 count,
      Int32 *states)
```

Include file

`Ds2210.h`

Purpose

To read the current status (0/1) of the `count` ignition capture inputs specified by the `channels` array.

I/O mapping

For information on the I/O mapping, refer to Spark Event Capture (DS2210 Features 🕮).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

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

Predefined Symbol	Meaning
`DS2210_IGNCAP_CH1`	Ignition capture channel 1
…	…
`DS2210_IGNCAP_CH6`	Ignition capture channel 6
`DS2210_AUXCAP_CH1`	Auxiliary capture channel 1
`DS2210_AUXCAP_CH2`	Auxiliary capture channel 2

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

Related topics

References

ds2210_ignition_capture_read 113
ds2210_ignition_fifo_read 115

Injection Pulse Position and Fuel Amount Measurement

Purpose

To measure injection pulses and their duration.

Where to go from here

Information in this section

Information in other sections

Injection Pulse Position and Fuel Amount Measurement (DS2210 Features 📖)
The injection event capture unit provides 16 digital injection inputs are split into 2 groups for injection pulse position and fuel amount measurement.

ds2210_inj_capture_mode_set

Syntax

```
void ds2210_inj_capture_mode_set(
      Int32 base,
      Int32 edge,
      Int32 mode,
      Int32 count)
```

Include file

`Ds2210.h`

Purpose

To set the mode for the injection pulse position and fuel amount capture.

Description

For basic information on the injection event capture, refer to Injection Event Capture Unit (DS2210 Features 📖)

Some mode settings are only allowed if the hardware supports extended functionality. If the board does not support extended functionality, the function exits with an error message.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

edge Selects high active or low active input pulses. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_INJCAP_RISE_EDGE`	High active input pulses. Rising edges are captured as leading edges.
`DS2210_INJCAP_FALL_EDGE`	Low active input pulses. Falling edges are captured as leading edges.

mode Specifies the capture mode. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_INJCAP_PULSE_POS`	The positions of leading and trailing edges of up to 8 pulses in the event window will be captured.
`DS2210_INJCAP_PULSE_DUR`	The positions of leading edges and the durations of up to 8 pulses in the event window will be captured with a resolution of 4 µs.
`DS2210_INJCAP_PULSE_DUR_HRES`	The positions of leading edges and the durations of up to 8 pulses in the event window will be captured with a resolution of 1 µs (only with extended functionality).

count Specifies the number of expected events within the range of 1 … 8

Note

If you use `ds2210_injection_fifo_read` to capture values, the `count` parameter is not valid. The number of events to be read is set by the `count` parameter of the function.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Basics

Injection Event Capture Unit (DS2210 Features 📖)

References

ds2210_injection_capture_read

Syntax

```
void ds2210_injection_capture_read(
      Int32 base,
      Int32 channel,
      Int32 *count,
      dsfloat *start_pos,
      dsfloat *end_pos)
```

Include file

`Ds2210.h`

Purpose

To read the injection pulse position(s) and their duration(s) from the last injection event capture window. You can use the injection channels 1 … 6 for injection capture, with extended functionality up to 8 channels (PWM input channels 7 and 8).

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2210 Features 📖).

Description

The fuel injection pulse positions and their durations captured within the last event window on the specified channel are returned by the arrays `start_pos` and `end_pos`. The parameter `count` returns the number of events actually captured within the last event window.

Note

- The input channels 7 and 8 are shared with PWM measurement. If you use these channels for PWM measurement you cannot use them for injection capture, refer to `ds2210_pwm_in` on page 68.
- If the number of captured events is less than the number of expected events specified by the `ds2210_inj_capture_mode_set` function, the function returns negative position or duration values for the missing events.
- Events will be captured up to the number of expected events. Additional events will be ignored.

Note

The inputs are 12 V-compatible. After initialization, the input threshold is set to 2.5 V. Use `ds2210_digin_threshold_set` to set the threshold within the range of 1.0 … 7.0 V. For further information, refer to Digital Inputs (PHS Bus System Hardware Reference).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Channel number. The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_INJCAP_CH1	Injection capture channel 1
…	…
DS2210_INJCAP_CH6	Injection capture channel 6
DS2210_INJCAP_CH7[1)]	Injection capture channel 7
DS2210_INJCAP_CH8[1)]	Injection capture channel 8

[1)] Only for boards with extended functionality

count Address where the current event count is written.

start_pos Address where angle positions at the beginning of the injection pulse are written. The values are given in degrees within the range of 0 … 720.

end_pos Address where angle position(s) or pulse duration(s) at the end of the injection pulse are written. The positions are given in degrees within the range of 0 … 720. The durations are given in seconds within the range of 0 … 1048.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

Digital Inputs (PHS Bus System Hardware Reference 📖)
ds2210_digin_threshold_set 20
ds2210_event_window_set 104
ds2210_inj_capture_mode_set 119
ds2210_pwm_in 68

ds2210_injection_fifo_read

Syntax

```
Int32 ds2210_injection_fifo_read(
      Int32 base,
      Int32 channel,
      Int32 count,
      Int32 *state,
      dsfloat *pos,
      Int32 *length)
```

Include file

`Ds2210.h`

Purpose

To read the injection pulse positions/durations captured during run time in the FIFO. The values of the specified channel are returned through the `pos` [deg/sec] array. You can use the injection channels 1 … 6 and in extended functionality up to 8 injection channels.

I/O mapping

For information on the I/O mapping, refer to Injection Pulse Position and Fuel Amount Measurement (DS2210 Features 📖).

Description

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 through 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 32 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 ignition capture modes:

Capture Modes	Meaning
DS2210_INJCAP_PULSE_POS	Only the angle positions of the captured events are read (in degrees). Under certain circumstances pseudo edges are generated.
DS2210_INJCAP_PULSE_DUR DS2210_INJCAP_PULSE_DUR_HRES	All start positions (in degrees) 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 (HRES). Under certain circumstances pseudo edges are generated.

Note

If the leading or trailing edge of an input pulse lies outside the capture window while 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 DS2210 board.

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

Predefined Symbol	Meaning
DS2210_INJCAP_CH1	Injection capture channel 1
...	...
DS2210_INJCAP_CH6	Injection capture channel 6
DS2210_INJCAP_CH7[1)]	Injection capture channel 7
DS2210_INJCAP_CH8[1)]	Injection capture channel 8

[1)] Only for boards with extended functionality

Note

Injection capture channel 7 and 8 conflict with PWM input channel 7 and 8.

count Specifies the number of events to be read. The maximum number must not exceed the internal FIFO size (32 events).

state Specifies the address where the state of the ignition events is written.

Capture Mode	State	Meaning
DS2210_INJCAP_RISE_EDGE	0	Falling edge/low active
	1	Rising edge/high active
DS2210_INJCAP_FALL_EDGE	0	Rising edge/high active
	1	Falling edge/low active

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 Specifies the 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, an overflow of the FIFO occurs. The overflow can be recognized by the return value of the function.

Return value

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

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

Related topics

References

ds2210_event_window_set..104
ds2210_inj_capture_mode_set..119

ds2210_injection status_read

Syntax

```
void ds2210_ignition_status_read(
      Int32 base,
      Int32 *channels,
      Int32 count,
      Int32 *states)
```

Include file

`Ds2210.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 (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

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

Predefined Symbol	Meaning
`DS2210_INJCAP_CH1`	Injection capture channel 1
…	…
`DS2210_INJCAP_CH6`	Injection capture channel 6
`DS2210_INJCAP_CH7`[1)]	Injection capture channel 7
`DS2210_INJCAP_CH8`[1)]	Injection capture channel 8

[1)] Only for boards with extended functionality

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

Related topics

References

ds2210_ignition_capture_read.......... 113
ds2210_ignition_fifo_read.......... 115

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

Basic Principles of Serial Communication

Where to go from here

Information in this section

Information in other sections

Serial Interface (DS2210 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

HowTos

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 128).
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

References

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();
   ds2210_init(DS2210_1_BASE);

/* allocate a new 1024 byte SW-FIFO */
   serCh = dsser_init(DS2210_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.. 155

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_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........155

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 129 or dsser_subint_handler_inst on page 158).

Members

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 140).

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 128).

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........ 128

References

dsser_init........ 140

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 132.

lineStatusReg Line status register. Refer to `dsser_LSR` on page 134.

modemStatusReg Modem status register. Refer to `dsser_MSR` on page 135.

Related topics

References

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 DS2210_y_BASE, with y as a consecutive number within the range of 1 … 16. For example, if there is only one DS2210 board, use DS2210_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........128

Examples

Example of a Serial Interface Communication........130

References

Data Types for Serial Communication........132
dsser_config........142
dsser_free........141

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 140).

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..128

References

dsser_init..140

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 140).

fifo_mode Specifies the mode of the receive buffer (see Serial Interface (DS2210 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 (DS2210 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 128). 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 128):

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.. 128

Examples

Example of a Serial Interface Communication.. 130

References

dsser_init.. 140

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 140).

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.......... 128

Examples

Example of a Serial Interface Communication.......... 130

References

dsser_init.......... 140
dsser_transmit_fifo_level.......... 153

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 140).

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

Examples

References

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 140).

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

References

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 140).

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

References

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 140).

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.......... 128

References

dsser_disable.......... 151
dsser_init.......... 140

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 140).

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

References

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 140).

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

References

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 140).

Return value

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

Related topics

Basics

References

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 140).

Return value

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

Related topics

Basics

References

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 140).

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

References

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 DS2210_y_BASE, with y as a consecutive number within the range of 1 … 16. For example, if there is only one DS2210 board, use DS2210_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 DS2210 board is delivered with a standard quartz working with the frequency of $1.8432 \cdot 10^6$ 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 140).

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........128

References

dsser_config........142
dsser_init........140

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 159).

> **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 140).

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 140).

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 128)
`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 128

Examples

Example of a Serial Interface Communication 130

References

dsser_init 140
dsser_subint_disable 160
dsser_subint_handler_inst 158

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 140).

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 128)
`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.. 128

References

dsser_init.. 140
dsser_subint_enable.. 159
dsser_subint_handler_inst.. 158

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

References

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..128

References

dsser_receive...147
dsser_receive_fifo_level...154
dsser_word2bytes..162

Slave DSP Access Functions

Introduction

To generate knock sensor signals, wheel speed sensor signals, or access the DSP via the dual-port memory.

Where to go from here

Information in this section

Information in other sections

Basics

Basic Communication Principles

Introduction

The communication between the master processor and the slave DSP is performed via the DPMEM of the DS2210. The access to this DPMEM is not arbitrated by hardware. To avoid conflicts when accessing the DPMEM from both sides – by the TMS320C31 DSP and the master processor – one of the eight semaphores (1 … 8) provided by the DS2210 can be used.

Note

The semaphores do not physically prevent improper access to the DPMEM.

Semaphore handling

A semaphore is requested by the master by writing a `"0"` to it. When you read the semaphore afterwards and get `"0"`, the semaphore has been accessed successfully. Therefore, you should poll the request function until the semaphore is obtained successfully. If the value is not equal to `"0"` the semaphore is obtained by the other side. You have to release the semaphore by writing a `"1"` to 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 **`ds2210_slave_dsp_read_direct`**, **`ds2210_slave_dsp_write_direct`**, **`ds2210_slave_dsp_block_read_di`**, and **`ds2210_slave_dsp_block_write_di`**. For these functions you have to use **`ds2210_slave_dsp_sem_req`** to request and **`ds2210_slave_dsp_sem_rel`** to release a semaphore.

Floating-point conversion

For the master processor of the processor board a different floating-point format is used. The master processor 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`** macros.

Related topics

References

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

To get information on global slave DSP functions.

Where to go from here

Information in this section

Information in other sections

Slave DSP TMS320C31 Basics (DS2210 Features 📖)
Providing general information on the slave DSP.

ds2210_slave_dsp_signal_enable

Syntax

```
Int32 ds2210_slave_dsp_signal_enable(
      Int32 base,
      UInt32 enable_mask)
```

Include file

`Ds2210.h`

Purpose

To start the slave DSP signal generation on the specified channels.

Description

`ds2210_slave_dsp_signal_enable` affects all signals. You specify the signals to be enabled, all other signals will be disabled. Use `ds2210_slave_dsp_channel_enable` to enable or disable only specific signals.

Signal generation on the slave DSP channels is started when the respective bit in `enable_mask` is set, in other case generation is stopped, when bit is not set.

Note

The slave DSP applications for knock and wheel speed signal generation have to be loaded to the slave DSP.

Tip

You can use `ds2210_slave_dsp_signal_enable` also to enable signal generation for your own slave DSP applications. As starting and stopping of the signal generation is an purely software implementation, your slave DSP application must evaluate parameters of this function. For an example, refer to Example of Knock Sensor Simulation on page 175 or Example of Wheel Speed Sensor Simulation on page 184.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

enable_mask Signals to be enabled or disabled; bit 0 = channel 1, bit 1 = channel 2, … The following symbols are predefined. To enable more than one signal, you can combine the symbols using the logical operator OR. If you do not specify a symbol, the signal is disabled.

Predefined Symbol	Meaning
`DS2210_SLVDSP_CH1`	Starts the generation of signal 1.
…	…
`DS2210_SLVDSP_CH8`	Starts the generation of signal 8.

For the knock sensor simulation the symbols `DS2210_SLVDSP_CH1` … `DS2210_SLVDSP_CH8` apply to 8 cylinders. For the wheel speed sensor simulation the symbols `DS2210_SLVDSP_CH1` … `DS2210_SLVDSP_CH4` apply to 4 wheel speed sensors.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_SLVDSP_DPMEM_ACCESS_PASSED`	The function has been performed without error.
`DS2210_SLVDSP_DPMEM_ACCESS_FAILED`	The dual-port memory could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_slave_dsp_channel_enable

Syntax

```
Int32 ds2210_slave_dsp_channel_enable(
      Int32 base,
      Int32 channel,
      UInt8 enable)
```

Include file

`Ds2210.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`.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Signal to be enabled. 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
`DS2210_SLVDSP_CH1`	Starts the generation of signal 1.
...	...
`DS2210_SLVDSP_CH8`	Starts the generation of signal 8.

For the knock sensor simulation the symbols `DS2210_SLVDSP_CH1` … `DS2210_SLVDSP_CH8` apply to 8 cylinders. For the wheel speed sensor simulation the symbols `DS2210_SLVDSP_CH1` … `DS2210_SLVDSP_CH4` apply to 4 wheel speed sensors.

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
`DS2210_SLVDSP_DPMEM_ACCESS_PASSED`	The function has been performed without error.
`DS2210_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 385.

Related topics

References

ds2210_slave_dsp_signal_enable 167

ds2210_slave_dsp_interrupt_set

Syntax

```
Int32 ds2210_slave_dsp_interrupt_set(
      Int32 base,
      void *value)
```

Include file

`Ds2210.h`

Purpose

To generate a slave DSP interrupt.

Description

By writing the specified data value to a defined location of the DPMEM, an interrupt is generated on the slave DSP. The interrupt triggers the external interrupt 1 (INT1) of the slave DSP (refer to Basic Communication Principles on page 166 and Slave DSP Basics on page 205).

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 DS2210 board.

value Data value to be written (the datatype can be float or long). For the master processor the floating-point values have to be converted (refer to Basic Communication Principles on page 166).

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

Basic Communication Principles..166
Slave DSP Basics..205

ds2210_slave_dsp_speedchk

Syntax

```
void ds2210_slave_dsp_speedchk(
      Int32 base,
      dsfloat *exec_min,
      dsfloat *exec_max,
      dsfloat *exec_cur)
```

Include file

`Ds2210.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 will be 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.

Parameters

base Specifies the PHS-bus base address of the DS2210 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 385.

Related topics

References

speedchk..278

ds2210_slave_dsp_error

Syntax

```
Int32 ds2210_slave_dsp_error(
      Int32 base,
      Int32 *state)
```

Include file

`Ds2210.h`

Purpose

To read the error flag of the slave DSP. In your slave DSP application, you can set the error flag with error_set on page 223.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

state Status of the slave DSP error flag

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_SLVDSP_DPMEM_ACCESS_PASSED`	The function has been performed without error.
`DS2210_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 385.

Related topics

References

error_set..223

ds2210_slave_dsp_appl_load

Syntax

```
void ds2210_slave_dsp_appl_load(
      Int32 base,
      Int32 *appl_addr)
```

Include file

`Ds2210.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 DPMEM is completely used for the application, booting the slave DSP takes approximately 25 ms.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

appl_addr Address of the first element of the slave DSP application

Return value

None

Message

Info message will be given only if the status flag DEBUG_INIT was set when compiling the application. The following messages are defined:

Type	Message	Meaning
Error	ds2210_slave_dsp_appl_load(0x??): slave-DSP is not responding!	The application could not be started on the slave DSP.
Error	ds2210_slave_dsp_appl_load(0x??): slave-load already been acknowledged!	The function additionally checks whether the hostmem section is still containing slave DSP application data. If the slave load has been acknowledged by a preceeding call to the `RTLIB_SLAVE_LOAD_ACKNOWLEDGE` function, the function exits with an error message. This error may occur, if you try to load a slave application which have been converted to an *.ASM Data File in a s-function. To avoid this error, generate a c file (*.slc) with slave data instead.
Info	ds2210_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 385.

Example

The following listing shows an example.

```
...
#include "Slv2210_ks.slc"
/**************************************************************
object declarations
**************************************************************/
/* pointer to slave DSP application data */
extern unsigned long ks[];
...
/*----------------------------------------------------------*/
void main()
{
   ...
   init();                              /* init CPU board */
   ds2210_init(DS2210_1_BASE);         /* init DS2210 board */
   /* load DS2210 slave-DSP application 'ks' */
   ds2210_slave_dsp_appl_load(DS2210_1_BASE, (Int32 *) &ks);
...
}
```

Related topics

HowTos

References

Knock Sensor Simulation

Purpose

To generate knock sensor signals.

Where to go from here

Information in this section

Information in other sections

Knock Sensor Simulation (DS2210 Features)
Providing information on the ready-to-use application (knock processor)
implemented on the slave DSP. This application is regarded as part of the
angular processing unit.

Example of Knock Sensor Simulation

Example

The following example shows how to use the knock sensor simulation.

```
#include <brtenv.h>                               /* basic real-time environment */
#include <ds2210.h>
#include "Slv2210_ks.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[];      /* 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);                                     /* call host 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;
  ds2210_apu_velocity_write(DS2210_1_BASE, apu_rad);
  /* read angle position and convert to degree */
  ds2210_apu_position_read(DS2210_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 */
    ds2210_slave_dsp_knock_update(DS2210_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 */
  ds2210_slave_dsp_knock_noise(DS2210_1_BASE, 1, noise[0]);
  exec_noise = RTLIB_TIC_READ() * 1.0e6;               /* read execution time */
  ds2210_slave_dsp_knock_noise(DS2210_1_BASE, 2, noise[1]);
  ds2210_slave_dsp_knock_noise(DS2210_1_BASE, 3, noise[2]);
  ds2210_slave_dsp_knock_noise(DS2210_1_BASE, 4, noise[3]);
  RTLIB_TIC_START();                       /* start execution time mesurement */
  /* write knock signal enable mask */
  ds2210_slave_dsp_signal_enable(DS2210_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 */
  ds2210_init(DS2210_1_BASE);                              /* init DS2210 board */
  /* load DS2210 slave-DSP application 'ks' */
  ds2210_slave_dsp_appl_load(DS2210_1_BASE, (Int32 *) &ks);
  msg_info_set(0, 0, "System started");
  ds2210_apu_transformer_mode_set(DS2210_1_BASE, DS2210_APU_TRANSFORMER_ENABLE);
  /* set APU mode, set APU velocity and start APU */
  ds2210_mode_set(DS2210_1_BASE, DS2210_MASTER_MODE);
  ds2210_apu_velocity_write(DS2210_1_BASE, apu_speed * 0.1047197551197);
  ds2210_apu_start(DS2210_1_BASE);
  /* initialize knock signal generation */
  ds2210_slave_dsp_knock_init(DS2210_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++)
    ds2210_slave_dsp_knock_noise(DS2210_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 */
      ds2210_slave_dsp_speedchk(DS2210_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 */
  }
}
```

ds2210_slave_dsp_knock_init

Syntax

```
void ds2210_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

`Ds2210.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 (DS2210 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 `ds2210_slave_dsp_signal_enable` to start signal generation. Use `ds2210_slave_dsp_knock_noise` to add an additional Gaussian noise to the knock signal.

The slave DSP´s analog outputs used by the knock signal simulator may be enabled or disabled by software (refer to `ds2210_apu_transformer_mode_set`).

After DS2210 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 DS2210 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 ≤ $(2^{31} - 1)$

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 ≤ $(2^{31} - 1)$

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

Example of Knock Sensor Simulation 175

References

ds2210_slave_dsp_knock_noise 182
ds2210_slave_dsp_knock_update 181
ds2210_slave_dsp_signal_enable 167

ds2210_slave_dsp_knock_update

Syntax

```
Int32 ds2210_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

`Ds2210.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 (DS2210 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 `ds2210_slave_dsp_knock_noise` to add an additional Gaussian noise to the knock signal.

Parameters

base Specifies the PHS-bus base address of the DS2210 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
DS2210_SLVDSP_DPMEM_ACCESS_PASSED	The function has been performed without error.
DS2210_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 385.

Related topics

Examples

References

ds2210_slave_dsp_knock_noise

Syntax

```
Int32 ds2210_slave_dsp_knock_noise(
      Int32 base,
      Int32 channel,
      dsfloat noise)
```

Include file

`Ds2210.h`

Purpose

To add an additional Gaussian noise to a knock sensor signal defined by `ds2210_slave_dsp_knock_init`.

Description

Use `ds2210_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 (DS2210 Features 📖).

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Knock sensor channel number within the range of 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
DS2210_SLVDSP_DPMEM_ACCESS_PASSED	The function has been performed without error.
DS2210_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 385.

Related topics

Examples

Example of Knock Sensor Simulation......175

References

ds2210_slave_dsp_knock_init......179
ds2210_slave_dsp_knock_update......181

Wheel Speed Sensor Simulation

Introduction

To generate wheel speed sensor signals.

Where to go from here

Information in this section

Information in other sections

Wheel Speed Sensor Simulation (DS2210 Features)
Providing information on the ready-to-use application implemented on the slave DSP. This application is regarded as part of the sensor and actuator interface.

Example of Wheel Speed Sensor Simulation

Example

The following example shows how to use the wheel speed sensor simulation.

```
#include <brtenv.h>                          /* basic real-time environment */
#include <ds2210.h>
#include "Slv2210_wheel.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 */
    ds2210_slave_dsp_wheel_update(DS2210_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 */
  ds2210_slave_dsp_signal_enable(DS2210_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 */
  ds2210_init(DS2210_1_BASE);                            /* init DS2210 board */
  /* load DS2210 slave-DSP application */
  ds2210_slave_dsp_appl_load(DS2210_1_BASE, (Int32 *) &wheel);
  msg_info_set(0, 0, "System started");
  ds2210_apu_transformer_mode_set(DS2210_1_BASE, DS2210_APU_TRANSFORMER_ENABLE);
  /* initialize wheel speed generation */
  ds2210_slave_dsp_wheel_init(DS2210_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 */
      ds2210_slave_dsp_speedchk(DS2210_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 */
  }
}
```

ds2210_slave_dsp_wheel_init

Syntax

```
void ds2210_slave_dsp_wheel_init(
      Int32 base,
      dsfloat *amplitude,
      dsfloat *speed,
      Int32 *teeth,
      dsfloat *noise)
```

Include file

`Ds2210.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 (DS2210 Features 📖).

Description

This function initializes all four channels at the same time. Use `ds2210_slave_dsp_signal_enable` on page 167 to start signal generation.

The wheel speed signal u(t) is generated according to the formula

$u(t) = a \cdot \sin(2 \cdot \pi \cdot f\_wheel \cdot t) + Noise.$

The wheel speed signal frequency (f_wheel) is calculated as follows:

$f\_wheel = speed \cdot teeth \cdot (1/2\pi)$

The slave DSP´s analog outputs used by the wheel speed signal generator may be enabled or disabled by software (refer to `ds2210_apu_transformer_mode_set` on page 23).

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 DS2210 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 385.

Related topics

Examples

Example of Wheel Speed Sensor Simulation........ 184

References

ds2210_apu_transformer_mode_set........ 23
ds2210_slave_dsp_signal_enable........ 167
ds2210_slave_dsp_wheel_update........ 187

ds2210_slave_dsp_wheel_update

Syntax

```
Int32 ds2210_slave_dsp_wheel_update(
      Int32 base,
      Int32 channel,
      dsfloat amplitude,
      dsfloat speed,
      Int32 teeth,
      dsfloat noise)
```

Include file

`Ds2210.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 (DS2210 Features 📖).

Description

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)

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

channel Channel number of the wheel speed signal within the range of 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
`DS2210_SLVDSP_DPMEM_ACCESS_PASSED`	The function has been performed without error.
`DS2210_SLVDSP_DPMEM_ACCESS_FAILED`	The dual-port memory could not be accessed. The function could not be performed.

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

Examples

References

Slave DSP Memory Access Functions

Where to go from here

Information in this section

Basics of Accessing the Slave DSP Memory

Introduction

You can directly exchange data between the master and the slave DSP accessing the dual-port memory (DPMEM).

Accessing the slave DSP memory

There are two different ways to perform this access:

- Some access functions request and release a semaphore automatically. This means that the semaphore handling will be performed for each call of the access function.
- Request the semaphore, perform several other activities, and release the semaphore afterwards. Therefore, 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 (DPMEM) and handle the semaphore.

```
sem_state = ds2210_slave_dsp_sem_req(DS2210_1_BASE, 2);
/* if the request was successful */
if(!sem_state)
{
   /* read from DS2210 slave DSP */
   ds2210_slave_dsp_read_direct(DS2210_1_BASE, 20,
                               (long *)&respond);
   /*write to DS2210 slave */
   ds2210_slave_dsp_write_direct(DS2210_1_BASE, 0,
                               (long *) &fct_nr);
   /* release semaphore */
   ds2210_slave_dsp_sem_rel(DS2210_1_BASE, 2);
}
```

Related topics

References

ds2210_slave_dsp_read

Syntax

```
Int32 ds2210_slave_dsp_read(
      Int32 base,
      Int32 sem_nr,
      Int32 offset,
      void *value)
```

Include file

`Ds2210.h`

Purpose

To read a data value from the DPMEM.

Description

`ds2210_slave_dsp_read` reads the DPMEM location specified by the `offset` parameter and returns the data through the `value` parameter. DPMEM access is handled by the semaphore specified by the `sem_nr` parameter. For this reason, `ds2210_slave_dsp_read` has to be polled until it returns `DS2210_SLVDSP_DPMEM_ACCESS_PASSED`.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

sem_nr Specifies the number of the semaphore to be used for DPMEM access within the range of 1 … 8

offset DPMEM address offset within the range of 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 207.

value Address in the master processor memory where the data value is written (datatype can be float or long). For the master processor the floating-point values have to be converted (refer to Basic Communication Principles on page 166).

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_SLVDSP_DPMEM_ACCESS_PASSED	The function has been performed without error.
DS2210_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 385.

Example

```
/* read value from DS2210 board */
error = ds2210_slave_dsp_read (DS2210_1_BASE, 1, 0,
                               (UInt32 *)&value);
```

Related topics

References

ds2210_slave_dsp_write

Syntax

```
Int32 ds2210_slave_dsp_write(
      Int32 base,
      Int32 sem_nr,
      Int32 offset,
      void *value)
```

Include file

`Ds2210.h`

Purpose

To write a data value to the DPMEM.

Description

`ds2210_slave_dsp_write` writes the contents of the parameter `value` to the DPMEM location specified by the parameter `offset`. DPMEM access is handled by the semaphore specified by the `sem_nr` variable. For this reason, `ds2210_slave_dsp_write` has to be polled until it returns `DS2210_SLVDSP_DPMEM_ACCESS_PASSED`.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

sem_nr Specifies the number of the semaphore to be used for DPMEM access within the range of 1 … 8

offset Specifies the DPMEM address offset within the range of 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 207.

value Specifies the address in the master processor memory where the data value to be written is stored (can be either float or long). For the master processor the floating-point values have to be converted (refer to Basic Communication Principles on page 166).

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_SLVDSP_DPMEM_ACCESS_PASSED`	The function has been performed without error.
`DS2210_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 385.

Example

```
/* write value to DS2210 board */
error = ds2210_slave_dsp_write(DS2210_1_BASE, 1, 0,
                              (UInt32 *)&value);
```

Related topics

References

Basic Communication Principles........ 166

ds2210_slave_dsp_block_read

Syntax

```
Int32 ds2210_slave_dsp_block_read(
      Int32 base,
      Int32 sem_nr,
      Int32 offset,
      Int32 count,
      void *value)
```

Include file

`Ds2210.h`

Purpose

To read a block of data values from the DPMEM.

Description

`ds2210_slave_dsp_block_read` reads a block of values (specified by the parameter `count`). The function starts with the DPMEM location specified by the parameter `offset` and returns the data through the parameter `value`. DPMEM access is handled by the semaphore specified by the variable `sem_nr`. For this reason, `ds2210_slave_dsp_block_read` has to be polled until it returns DS2210_SLVDSP_DPMEM_ACCESS_PASSED.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

sem_nr Specifies the number of the semaphore to be used for DPMEM access within the range of 1 … 8

offset DPMEM address offset within the range of 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.

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). For the master processor the floating-point values have to be converted (refer to Basic Communication Principles on page 166).

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_SLVDSP_DPMEM_ACCESS_PASSED	The function has been performed without error.
DS2210_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 385.

Related topics

References

Memory Map of the Slave DSP..207

ds2210_slave_dsp_block_write

Syntax

```
Int32 ds2210_slave_dsp_block_write(
      Int32 base,
      Int32 sem_nr,
      Int32 offset,
      Int32 count,
      void *value)
```

Include file

`Ds2210.h`

Purpose

To write a block of data values to the DPMEM.

Description

`ds2210_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.

The access to the DPMEM is handled by a semaphore specified by the `sem_nr` variable. For this reason, `ds2210_slave_dsp_block_write` has to be polled until it returns DS2210_SLVDSP_DPMEM_ACCESS_PASSED.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

sem_nr Specifies the number of the semaphore to be used for DPMEM access within the range of 1 … 8

offset DPMEM address offset within the range of 0x0000 … 0x3FFF. The offset specifies the DPMEM location in the slave DSP's memory map.

count Number of data values to be written

value Address in the master processor memory where the data values to be written are stored (datatype can be either float or long). For the master processor the floating-point values have to be converted (refer to Basic Communication Principles on page 166).

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_SLVDSP_DPMEM_ACCESS_PASSED	The function has been performed without error.
DS2210_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 385.

Related topics

References

ds2210_slave_dsp_sem_req

Syntax

```
Int32 ds2210_slave_dsp_sem_req(
      Int32 base,
      Int32 sem_nr)
```

Include file

`Ds2210.h`

Purpose

To request a semaphore for slave DSP access.

Description

Use this function for `ds2210_slave_dsp_read_direct`, `ds2210_slave_dsp_write_direct`, `ds2210_slave_dsp_block_read_di`, and `ds2210_slave_dsp_block_write_di`.

Use `ds2210_slave_dsp_sem_rel` to release the semaphore.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

sem_nr Number of the semaphore to be requested within the range of 1 … 8.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_SLVDSP_DPMEM_ACCESS_PASSED	The semaphore has been requested successfully.
DS2210_SLVDSP_DPMEM_ACCESS_FAILED	The request failed and the semaphore request is released.

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_slave_dsp_sem_rel

Syntax

```
void ds2210_slave_dsp_sem_rel(
      Int32 base,
      Int32 sem_nr)
```

Include file

`Ds2210.h`

Purpose

To release a requested semaphore.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

sem_nr Number of the semaphore to be released within the range of 1 … 8.

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_slave_dsp_block_read_di 200
ds2210_slave_dsp_block_write_di 201
ds2210_slave_dsp_read_direct 197
ds2210_slave_dsp_sem_req 196
ds2210_slave_dsp_write_direct 198

ds2210_slave_dsp_read_direct

Syntax

```
void ds2210_slave_dsp_read_direct(
      Int32 base,
      Int32 offset,
      void *value)
```

Include file

`Ds2210.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 `ds2210_slave_dsp_sem_req` before invoking `ds2210_slave_dsp_read_direct` and `ds2210_slave_dsp_sem_rel` to release the semaphore after accessing the DPMEM.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

offset DPMEM address offset within the range of 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 207.

value Address in the master processor memory where the data value is written (datatype can be float or long). For the master processor the floating-point values have to be converted (refer to Basic Communication Principles on page 166).

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_slave_dsp_write_direct

Syntax

```
void ds2210_slave_dsp_write_direct(
      Int32 base,
      Int32 offset,
      void *value)
```

Include file

`ds2210.h`

Purpose

To write a data value to the DPMEM.

Description

The contents of the `value` parameter is written to the DS2210 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 `ds2210_slave_dsp_sem_req` before invoking `ds2210_slave_dsp_write_direct` and `ds2210_slave_dsp_sem_rel` to release the semaphore after accessing the DPMEM.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

offset DPMEM address offset within the range of 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 207.

value Specifies the address in the master processor memory where the data value to be written is stored (can be either float or long). For the master processor the floating-point values have to be converted (refer to Basic Communication Principles on page 166).

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_slave_dsp_block_read_di

Syntax

```
void ds2210_slave_dsp_block_read_di(
      Int32 base,
      Int32 offset,
      Int32 count,
      void *value)
```

Include file

`Ds2210.h`

Purpose

To read a block of data values from the DPMEM.

Description

A block of `count` data values is read from the DS2210 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 `ds2210_slave_dsp_sem_req` before invoking `ds2210_slave_dsp_block_read_di` and `ds2210_slave_dsp_sem_rel` to release the semaphore after accessing the DPMEM.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

offset DPMEM address offset within the range of 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 207.

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). For the master processor the floating-point values have to be converted (refer to Basic Communication Principles on page 166).

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

ds2210_slave_dsp_block_write_di

Syntax

```
void ds2210_slave_dsp_block_write_di(
      Int32 base,
      Int32 offset,
      Int32 count,
      void *value)
```

Include file

`Ds2210.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 `offset` parameter. 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 `ds2210_slave_dsp_sem_req` before invoking `ds2210_slave_dsp_block_write_di` and `ds2210_slave_dsp_sem_rel` to release the semaphore after accessing the DPMEM.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

offset DPMEM address offset within the range of 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 207.

count Number of data values to be written

value Address in the master processor memory where the data values to be written are stored (datatype can be either float or long). For the master processor the floating-point values have to be converted (refer to Basic Communication Principles on page 166).

Return value

None

Execution times

For information, refer to Function Execution Times on page 385.

Related topics

References

Slave DSP Functions and Macros

Introduction

You can use a set of macros and functions to write applications running on 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.

Where to go from here

Information in this section

Information in other sections

Slave DSP Basics

Introduction

To write your own applications for the slave DSP TMS320C31 the following information and features are provided.

Where to go from here

Information in this section

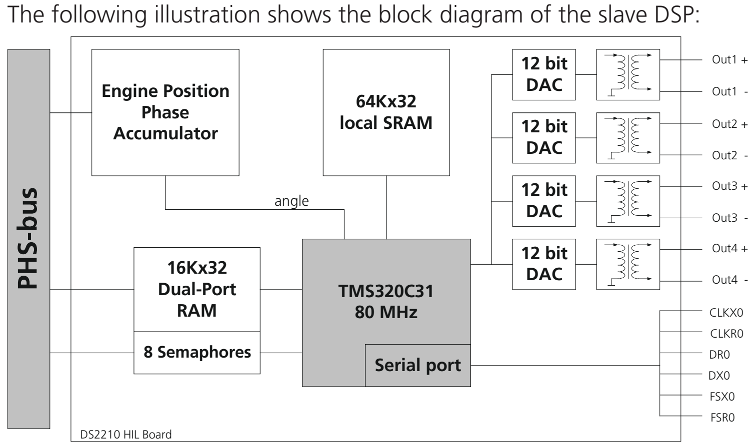
Basics for Programming the Slave DSP

Introduction

You can use the following features to write your own applications for the slave DSP.

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, you do not have to know the memory locations for the predefined functions and macros. Refer to Definitions on page 209.

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. Two external maskable interrupts (INT0 and INT1) are supported. Interrupts are automatically prioritized, allowing interrupts to occur simultaneously and 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 TMS320C31. For the interrupts used on the DS2210 and the alias names, refer to Interrupts on page 214.

Timer 0 and 1

The slave DSP has two 32-bit general-purpose timer modules (timer0 and timer1). The timer interrupt is one of the internal interrupts (TINT0 or TINT1). Refer to timer0, timer1 on page 213

Error flag

To indicate an error state you can use the error flag of the DSP. Refer to Error Handling on page 223.

DSP state

You can indicate the state of the DSP with on-board LEDs. Refer to Status LEDs on page 225.

D/A converters

The slave DSP provides four 16-bit D/A converters. Using these converters you have to scale floating-point values and convert them to the datatype long integer. Refer to D/A Converter on page 227.

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 on page 230. The digital I/O pins are shared with serial interface pins.

Master to slave communication

The communication between the master processor and the slave DSP is performed via the DPMEM of the DS2210. The access to this DPMEM is not arbitrated by hardware. To avoid conflicts when accessing the DPMEM from both sides – by the TMS320C31 DSP and the master processor – one of eight semaphores (1 … 8) provided by the master processor can be used. For the use of the semaphores, refer to DPMEM Access Functions on page 233.

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 238.

Serial interface

The slave DSP provides one serial interface to connect a DS2302 board or another DS2210 board. For information on the initialization and the use of the serial interface, refer to Serial Interface on page 245.

Execution time measurement

The execution times of slave DSP applications can be measured with functions and macros. Refer to Execution Time Measurement on page 260.

Programming environment

The software and the tools needed to write your slave DSP application are installed with the dSPACE installation:

- For information on host PC settings (for example, variables and directory structures), refer to Host PC Settings on page 266.
- Batch files, makefiles and linker command files are available to customize the software environment (refer to Batch Files, Makefiles, Linker Command Files on page 270).
- To calculate the execution times of slave DSP applications, use the speed check utility described in Execution Time Information on page 277.
- There are some ways to optimize the assembly code of your slave DSP application (refer to Assembly Code Optimization on page 280).
- For information on loading applications to the slave DSP, refer to Loading Slave Applications on page 285.

Further information

For more information on the TMS320C31 slave DSP, refer to the Texas Instruments web site at http://www.ti.com and search for "TMS320C31".

Memory Map of the Slave DSP

Introduction

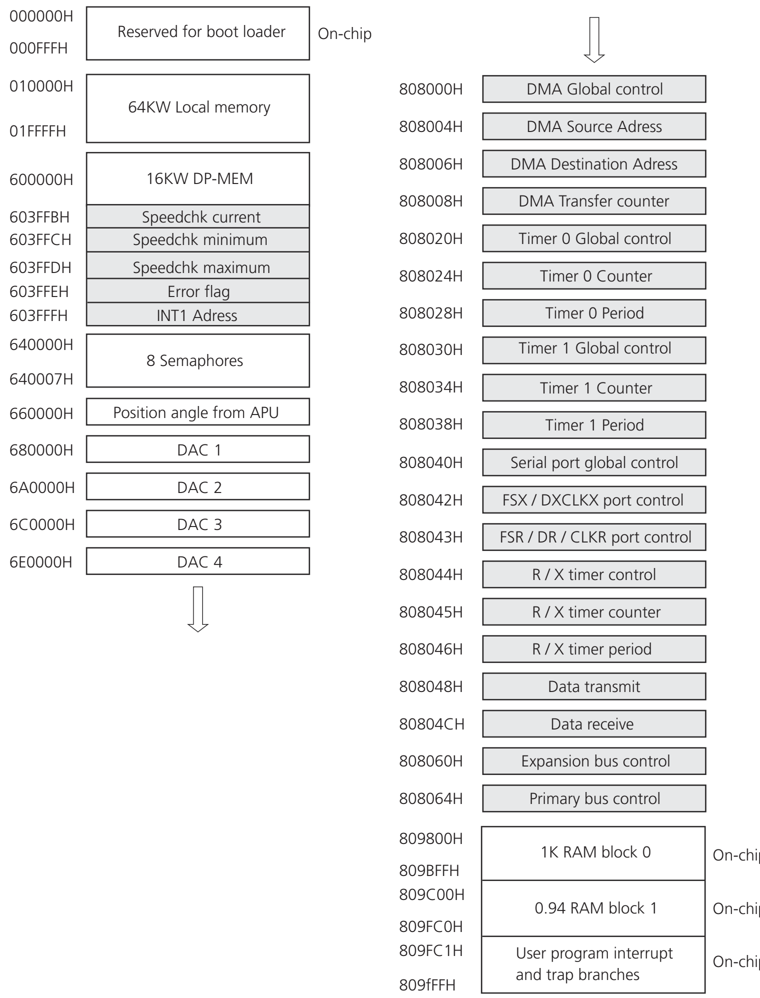
You need to consider the various memory map ranges when writing applications for the slave DSP.

This topic shows the memory map of the slave DSP (digital signal processor, Texas Instruments TMS320C31) on-board the DS2210.

Memory map

You can directly exchange data between the master and the slave DSP accessing the 16-KW DPMEM (dual-port memory), see Slave DSP Access Functions on page 165. 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 TMS320C31 digital signal processor.



Related topics

References

Slave DSP Functions and Macros..203

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 header file `Ds2210.h`. 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.

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 DS2210 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

Identifier	Value	Meaning
SEMAPHORE1	0x640000	Semaphore addresses
SEMAPHORE2	0x640001	
SEMAPHORE3	0x640002	
SEMAPHORE4	0x640003	
SEMAPHORE5	0x640004	
SEMAPHORE6	0x640005	
SEMAPHORE7	0x640006	
SEMAPHORE8	0x640007	
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 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.

Pointers and variables

Note

Each initialized pointer requires 3 words in the .cinit section and an additional word in the .bss section. Refer to Ds2210.lk, DS2210_1.lk, DS2210_2.lk, DS2210_3.lk on page 272.

The following table shows the pointers and variables.

Pointer	Meaning
long *dac1	Pointers to the D/A converter output registers
long *dac2	
long *dac3	
long *dac4	
long *int_tbl	Pointer to the interrupt vector table
float clock_per_sec	Slave DSP clock frequency (40.0 MHz)

Pointer	Meaning
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 reserved for the external interrupt INT1
long *error	Pointer to the error flag

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

Function

```
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

Function

```
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 214.

Parameters

time Required sampling period in seconds at which timer interrupts will be generated

Return value

None

Interrupts

Purpose

To handle interrupts on the slave DSP.

Where to go from here

Information in this section

Basics of Slave DSP Interrupts

Interrupts

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 **`int1_init`** function, refer to int0_init, int1_init on page 217.
XINT0	`c_int05()`	`isr_transmit`	Transmit interrupt of DSP's serial interface, refer to serial_tx_int_init on page 253.
RINT0	`c_int06()`	`isr_receive`	Receive interrupt of the DSP's serial interface, refer to serial_rx_int_init on page 252.
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 213.
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 213.

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.

Example of Slave DSP Interrupts

Introduction

You can use the interrupts INT0 and INT1 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 or int1_pending to poll the interrupt flag. After an interrupt is received, you have to clear the interrupt flags with the appropriate macro int0_ack or int1_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` or `int1_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` or `int1_ack`. Under normal conditions, you should prefer this method because it spends no time polling the interrupt flag.

```
void c_int02()           /* INT1 interrupt service routine */
{
   ...                        /* perform interrupt service */
   int1_ack(value);          /* acknowledge interrupt INT1 */
}
main()
{
   …
   int1_init();       /* initialize INT1 interrupt service */
   …
}
```

Related topics

References

int0_init, int1_init

Function

```
void int0_init()
void int1_init()
```

Include file

`Int0.c`

Purpose

To initialize the interrupts INT0 or INT1 as follows:

- Set the related interrupt vector to point to the corresponding interrupt service routine c_int01() or c_int02(). The alias name isr_int0() or isr_int1() can be used instead of c_int01() or c_int02() for the interrupt service routines.
- Enable the related interrupt.
- Enable interrupts globally.

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 macro `int0_ack` or `int1_ack`.

Return value

None

Related topics

Examples

Example of Slave DSP Interrupts......216

References

int0_ack......220
int1_ack......220
Interrupts......214

enable_int0, enable_int1, enable_tint0, enable_tint1

Macro

```
void enable_int0()
```

```
void enable_int1()
void enable_tint0()
void enable_tint1()
```

Include file

`Ds2210.h`

Purpose

To enable the related interrupt. You have to enable interrupts globally to activate the DSP's interrupts with global_enable.

Return value

None

Related topics

References

disable_int0, disable_int1, disable_tint0, disable_tint1 218
global_disable 219
global_enable 219

disable_int0, disable_int1, disable_tint0, disable_tint1

Macro

```
void disable_int0()
void disable_int1()
void disable_tint0()
void disable_tint1()
```

Include file

`Ds2210.h`

Purpose

To disable the related interrupt.

Return value

None

Related topics

References

enable_int0, enable_int1, enable_tint0, enable_tint1 217
global_disable 219
global_enable 219

global_enable

Macro	`void global_enable()`
Include file	`Ds2210.h`
Purpose	To enable all the interrupts that have been individually enabled with enable_int0, enable_int1, enable_tint0 or enable_tint1.
Return value	None
Related topics	References

enable_int0, enable_int1, enable_tint0, enable_tint1 217
global_disable 219

global_disable

Macro	`void global_disable()`
Include file	`Ds2210.h`
Purpose	To disable all the interrupts that have been enabled with `global_enable`.
Return value	None
Related topics	References

disable_int0, disable_int1, disable_tint0, disable_tint1 218
enable_int0, enable_int1, enable_tint0, enable_tint1 217
global_enable 219

int0_ack

Macro

```
void int0_ack()
```

Include file

`Ds2210.h`

Purpose

To acknowledge an interrupt request for INT0 when the related interrupt service is finished.

Return value

None

Related topics

References

int1_ack 220

int1_ack

Macro

```
void int1_ack(long value)
```

Include file

`Ds2210.h`

Purpose

To acknowledge an interrupt request for INT1 when the related interrupt service is finished.

The parameter **`value`** returns the contents of the DPMEM address reserved for INT1. The master processor writes a value to this address in order 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.

Parameters

value	contents of the INT1 address

Return value None

Related topics

References

int0_ack ... 220

int0_pending

Macro

```
long int0_pending()
```

Include file `Ds2210.h`

Purpose To read the state of the INT0 interrupt flag.

Description Use this macro to serve an interrupt request with no interrupt service routine installed by simply polling the interrupt flag, refer to Method 1 in Example of Slave DSP Interrupts on page 216.

Return value Interrupt flag; the following values are defined:

Value	Meaning
0	Interrupt INT0 is inactive.
1	Interrupt INT0 is pending.

Related topics

Examples

Example of Slave DSP Interrupts ... 216

References

int1_pending ... 222

int1_pending

Macro

```
void int1_pending(long state)
```

Include file

`Ds2210.h`

Purpose

To read the state of the INT1 flag.

Description

Use this macro to serve an interrupt request with no interrupt service routine installed by simply polling the interrupt flag, refer to Method 1 in Example of Slave DSP Interrupts on page 216.

Parameters

state Interrupt flag; the following values are defined:

Value	Meaning
0	Interrupt INT1 is inactive.
1	Interrupt INT1 is pending.

Related topics

Examples

References

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.

Where to go from here

Information in this section

error_set

Macro

```
void error_set(long value)
```

Include file

`Ds2210.h`

Purpose

To set the error flag. Writing different values to this flag allows you to specify different error situations.

Parameters

value	Error state

Return value

None

Related topics

References

ds2210_slave_dsp_error 172
error_read 224

error_read

Macro	`long error_read()`
Include file	`Ds2210.h`
Purpose	To read the error flag. Depending on the contents of the error flag you can handle different error situations.
Return value	Contents of the error flag
Related topics	References

ds2210_slave_dsp_error........172
error_set........223

Status LEDs

Where to go from here

Information in this section

Basics of Status LEDs

Status LEDs

The DS2210 board provides two status LEDs for the slave DSP. The following table shows the meaning of the LEDs:

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 DS2210 Components (PHS Bus System Hardware Reference 🕮).

led_state

Macro

```
void led_state(long value)
```

Include file

`Util2210.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

Introduction

To program the digital/analog converters.

Where to go from here

Information in this section

Basics of the D/A Converter of the Slave DSP

Introduction

The slave DSP provides four 16-bit D/A converters. The value to be written to a D/A converter must be within the 32-bit signed integer range of $-2.14748365 \cdot 10^9 \ldots +2.14748365 \cdot 10^9$. This range corresponds to the full analog output range of ±20 V.

Scale floating-point values

You have to scale floating-point values to the given range and convert them to the datatype 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 header file `Ds2210.h`. Refer to Identifiers for Numerical Constants on page 209.

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 … *dac4, refer to Pointer Declarations and Global Variables on page 210.

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_out4`. The `dac_out` macro allows you to pass the channel number as a parameter, whereas the **dac_out1** … **dac_out4** macros have shorter execution times.

For further information on the DAC and its I/O mapping, refer to DAC Unit (DS2210 Features).

dac_out

Macro

```
void dac_out(long channel, float value)
```

Include file

Ds2210.h

Purpose

To write the value to the given D/A channel.

The output parameter value is scaled by the factor $2.147483648 \cdot 10^9$ and converted to the datatype long integer before it is written to the specified D/A channel.

Parameters

channel	D/A channel within the range of 1 … 4
value	Output value within the range of –1.0 … +1.0

Return value

None

Related topics

References

dac_out1, dac_out2, dac_out3, dac_out4 228

dac_out1, dac_out2, dac_out3, dac_out4

Macro

```
void dac_out1(float value)
void dac_out2(float value)
void dac_out3(float value)
void dac_out4(float value)
```

Include file

Ds2210.h

Purpose

To write the value to the given D/A channel.

The output parameter value is scaled by the factor $2.147483648 \cdot 10^9$ and converted to the datatype long integer before it is written to the related D/A channel.

Parameters

value	Output value within the range of –1.0 … +1.0

Return value

None

Related topics

References

dac_out 228

Digital I/O

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 P5, refer to DS2210 Components (PHS Bus System Hardware Reference 🕮).

Digital I/O lines

The macros are related to the digital output lines as listed in the following table:

Macro	Slave DSP Serial Interface Pin	P5 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 245).

init_dig_out1, ... , init_dig_out6

Macro

```
void init_dig_out1(long value)
...
void init_dig_out6(long value)
```

Include file

`Ds2210.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

References

dig_out1, ... , dig_out6

Macro

```
void dig_out1(long value)
...
void dig_out6(long value)
```

Include file `Ds2210.h`

Purpose To set the respective digital I/O line to "0" or "1".

You have to initialize the pin for output first with the appropriate `init_dig_out<n>` macro.

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 References

dig_in1, ... , dig_in6 ... 232
init_dig_out1, ... , init_dig_out6 ... 231

dig_in1, ... , dig_in6

Macro

```
long dig_in1()
...
long dig_in6()
```

Include file `Ds2210.h`

Purpose To read the state of the respective digital I/O line.

You have to initialize the I/O line for input first with the appropriate `init_dig_out<n>` macro.

Return value State of the digital input line. Valid values are "0" or "1".

Related topics References

dig_out1, ... , dig_out6 ... 231
init_dig_out1, ... , init_dig_out6 ... 231

DPMEM Access Functions

Introduction

To read or write values to the DPMEM.

Where to go from here

Information in this section

Basics of Accessing the Dual-Port Memory

Introduction

The access to the DPMEM is not arbitrated by hardware. To avoid conflicts when accessing the DPMEM from both sides – the slave DSP and the master processor board – one of the eight semaphores should be used.

Note

The semaphores do not physically prevent improper access to the DPMEM.

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 will be 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 will be 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

`Ds2210.h`

Purpose

To request an access to the DPMEM.

Description

An access to the DPMEM is requested by attempting to write a `"0"` to the specified semaphore. 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 have to repeat the request or release the semaphore request by calling the `semaphore_release` macro.

Parameters

nr Number of the semaphore to be requested within the range of 1 … 8

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..235
semaphore1_request, … , semaphore8_request..236

semaphore_release

Macro

```
void semaphore_release(long nr)
```

Include file

`Ds2210.h`

Purpose

To release the specified semaphore by writing a `"1"` to the semaphore.

Parameters

nr Number of the semaphore to be requested within the range of 1 … 8

Return value

None

Related topics

References

semaphore1_request, ... , semaphore8_request

Macro

```
void semaphore1_request(long state)
...
void semaphore8_request(long state)
```

Include file

`Ds2210.h`

Purpose

To request an 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 parameter state does not return `"0"`, the semaphore is used by the opposite port. You have to repeat the request or release the semaphore request by calling the `semaphore_release` macro.

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

semaphore1_release, ... , semaphore8_release

Macro

```
void semaphore1_release()
...
void semaphore8_release()
```

Include file

`Ds2210.h`

Purpose

To release the specified semaphore by writing "1" to the semaphore.

Return value

None

Related topics

References

semaphore_release 235
semaphore1_request, ... , semaphore8_request 236

Direct Memory Access

Introduction

To transfer data to the memory of the slave DSP via a direct memory access (DMA) controller.

Where to go from here

Information in this section

Basics of Direct Memory Access

Basics

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 TMS320C31 slave DSP, refer to the Texas Instruments web site at http://www.ti.com and search for "TMS320C31".

dma_init

Function

```
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

`Ds2210.h`

Purpose

To initialize and start the DMA controller of the slave DSP. 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........243
dma_interrupt_enable........242
dma_stop........240
dma_stop_when_finished........241

dma_stop

Macro

```
void dma_stop()
```

Include file

`Dma31.h`

Purpose	To stop the DMA controller. The current word read or write operation will be completed.
Return value	None
Related topics	References

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

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_reset

Macro	`void dma_reset()`
Include file	`Dma31.h`
Purpose	To reset the DMA controller.
Return value	None
Related topics	References

dma_interrupt_enable

Function	`void dma_interrupt_enable(unsigned long mask)`
Include file	`Dma31.h`

Purpose

To enable the external DMA interrupts.

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_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 243

dma_interrupt_disable

Function

```
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_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.. 242

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, a DS2211, or another DS2210 board. For the location of the serial interface connector P5, refer to DS2210 Components (PHS Bus System Hardware Reference 📖).

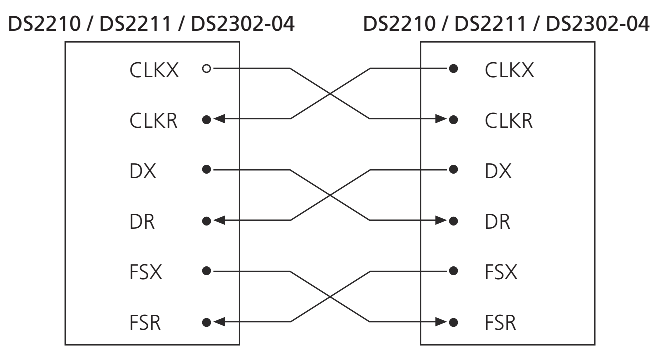
The connection provides a frequency of up to 10 MHz. For the serial transmission the handshake mode is used.

> **Note**
>
> If the serial interface is initialized and used for data transmission, the digital I/O access macros must not be used. Otherwise the serial transmission will fail.

For more information on the TMS320C31 slave DSP, refer to the Texas Instruments web site at http://www.ti.com and search for "TMS320C31".

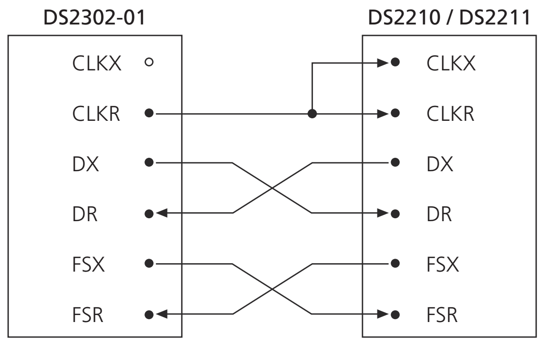
Connection schemes

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.



Related topics

References

Digital I/O........230

Example of Using the Serial Interface of the Slave DSP

Example 1

The following example shows the serial communication in polling mode. The serial interface is initialized for the standard handshake mode. The transmission will be performed with a frequency of 7.5 MHz for a connection to a board with a 60 MHz CPU clock frequency.

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 2

The following example shows the serial transmission in interrupt-driven mode. The serial interface is initialized for the standard handshake mode. The transmission will be performed with a frequency of 7.5 MHz for a connection to a board with 60 MHz.

The transmit and the receive interrupts are initialized. The received data is stored in the `receive_data[]` data array by the `serial_rx_word_int` function. Data to be transmitted is available in the `transmit_data[]` array. To start the 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 sending the 100 data values with the `serial_tx_word_int` function the transmission stops. No data was 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

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 DS2210 – DS2210 connections.

Description

This macro calls **`serial_init`** automatically with the required parameters. For a connection to a DS2302, refer to serial_init_ds2210 on page 249.

> **Note**
>
> The receiving frequency via the DR0 input depends on the setting of the transmitting serial interface.

Parameters

timer_prd Frequency of the serial transmission via the DX0 output. Valid values are 0x0001 … 0xFFFF. The transmission frequency is calculated as follows

f_{trans} = oscillator clock frequency / (8 · timer_prd)

The oscillator clock frequency of DS2210 boards is 80 MHz.

Return value

None

Related topics

References

serial_init_ds2210

Macro

```
void serial_init_ds2210()
```

Include file

`Ser31.h`

Purpose

To initialize the slave DSP's serial interface for data transfer in handshake mode for DS2210 – DS2302 connections.

Descripton

This function calls `serial_init` automatically with the required parameters. For a connection to a DS2210 board, refer to serial_init_std_handshake on page 249.

> **Note**
>
> The transmission frequency depends on the initialization of the DS2302 board's serial interface.

Return value

None

Related topics

References

serial_init

Function

```
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.

Description

The function does the following:

- Clears the global control register and the timer control register to reset the serial interface.
- Initializes the serial interface registers with the specified values.

- Starts the required serial interface timers depending on the control register settings.
- Enables receive and transmit access.

For more information on the TMS320C31 slave DSP, refer to the Texas Instruments web site at http://www.ti.com and search for "TMS320C31".

Note

`serial_init` is called automatically by the board-related initialization macros `serial_init_std_handshake` and `serial_init_ds2210`.

Parameters

g_ctrl Setting of the global control register

timer_prd Frequency of the serial transmission. The value is automatically provided by the board-related initialization macros `serial_init_std_handshake` and `serial_init_ds2210`.

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_ds2210 249
serial_init_std_handshake 249

serial_disable

Macro

```
void serial_disable()
```

Include file

`Ser31.h`

Purpose

To disable and reset the serial interface. All serial interface pins are configured as digital I/O pins and set as inputs (refer to Digital I/O on page 230).

Return value

None

serial_rx_int_init

Function

```
void serial_rx_int_init()
```

Include file

`Ser31ir.h`

Purpose

To initialize the receive interrupt of the serial interface.

Description

The receive interrupt of the serial interface is initialized as follows:

- Set the corresponding interrupt vector RINT0 to point to the receive interrupt routine `c_int06`.
- Enable the receive interrupt and interrupts globally.

The receive interrupt service routine usually contains the receive function and must be programmed by the user.

> **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 247.

Return value

None

Related topics

References

serial_tx_int_init

Function	`void serial_tx_int_init()`
Include file	`Ser31ix.h`
Purpose	To initialize the transmit interrupt of the serial interface.

Description

The transmit interrupt of the serial interface is initialized as follows:

- Set the corresponding interrupt vector XINT0 to point to the receive interrupt routine `c_int05`.
- Enable the receive interrupt and interrupts globally.

The transmit interrupt service routine usually contains the transmit function and must be programmed by the user.

> **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 247.

After the initialization, you have to start the interrupt-driven transmission with `serial_tx_int_start`. This macro will request the first transmit interrupt by setting the respective flag in the DSP's IF register.

Return value

None

Related topics

References

serial_tx_int_start

Macro	`void serial_tx_int_start()`

Include file	`Ser31.h`
Purpose	To request the first transmit interrupt.

Description

Call this macro after the initialization of the transmit interrupt to start the interrupt-driven transmission.

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 start the transmission again each time the transmission has stopped.

Return value

None

Related topics

References

serial_rx_int_init 252
serial_tx_int_init 253

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).

Return value

None

Related topics

References

enable_rx_int, enable_tx_int...255

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. The enable bit for the interrupt RINT0 or XINT0 will be set in the DSP's IE register to enable the corresponding interrupt.

Return value

None

Related topics

References

disable_rx_int, disable_tx_int...254

serial_tx_word_poll

Function

```
int serial_tx_word_poll(void *word)
```

Include file

`Ser31.h`

Purpose

To transmit a 32-bit data word via the serial interface. The value can be either of type float or long. If the transmit buffer of the serial interface is empty, this means the port is ready to transmit, 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..257

serial_tx_word_int

Function

```
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.

Descrption

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

serial_rx_word_poll

Function

```
int serial_rx_word_poll(void *word)
```

Include file

`Ser31.h`

Purpose

To receive a 32-bit data word via the serial interface. If the receive buffer contains new data the buffer will be read and the function returns SP_TRUE. Otherwise, the buffer will not be 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.. 255

serial_rx_word_int

Function

```
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. 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..255
serial_rx_int_init..252
serial_tx_word_int..256

Execution Time Measurement

Purpose

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 262

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 262
tic0_read, tic1_read 263

tic0_halt, tic1_halt

Macro	`void tic0_halt()` `void tic1_halt()`
Include file	`Tic3x.h`
Purpose	To pause the time measurement. The break lasts until the measurement is resumed by **`tic0_continue`** or **`tic1_continue`**.
Return value	None
Related topics	References

tic0_continue, tic1_continue 263
tic0_start, tic1_start 262

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_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. 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_continue, tic1_continue.......263
tic0_halt, tic1_halt.......262
tic0_start, tic1_start.......262

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.

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_continue, tic1_continue.......263
tic0_halt, tic1_halt.......262
tic0_read, tic1_read.......263
tic0_start, tic1_start.......262

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 DS2210 and some board-related utilities.

Where to go from here

Information in this section

Folder Structure

Introduction

The software for the DS2210 slave DSP are in subfolders under the <RCP_HIL_InstallationPath> folder as follows:

Folder	Contents
`\DS2210\SlaveDSP\RTLib`	Source and library files of the DS2210 RTLib, makefiles and linker command file for DS2210 slave applications
`\DS2210\SlaveDSP\Apps`	Standard slave DSP application wheel speed and knock sensor
`\DS2210\Can`	Firmware for the CAN controller
`\DS100x\RTLib`	Source and library files for DS2210 master applications
`\Exe`	Tools for handprogramming the PPC; DS2210 host programs

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 will be copied to the directory <RCP_HIL_InstallationPath>\Ds2210\SlaveDSP or <RCP_HIL_InstallationPath>\DS100x during software installation.

Ds2210.lib

Operations that are not used in time-critical program parts are implemented as functions collected in the real-time library `Ds2210.lib`. All the functions have been 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 `Ds2210.h`, `Util2210.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
`Ds2210.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
`Util2210.h`	Definitions and macros for debugging purposes and turnaround time measurement

Note

You have to link `Ds2210.lib` to each slave DSP application. This is done automatically when you use the standard compile and link utility CL2210.

Related topics

References

Cl2210.exe..270

How to Set the Compiler Path

Objective

Before you can use `Cl2210.exe` to compile and link source code for the slave DSP of your DS2210 board, you have to specify the installation path of your Texas Instruments Compiler (TI Compiler) as an environment variable.

Method

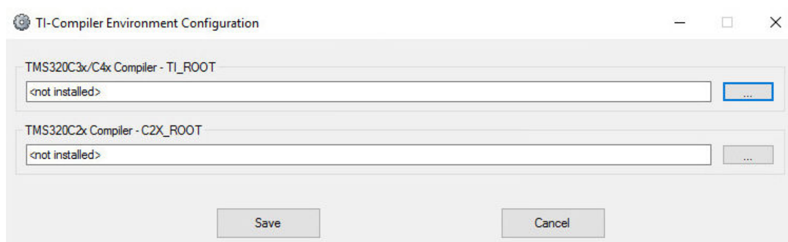
To set a compiler path

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.



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

Cl2210.exe 270

File 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
.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

Cl2210.exe

Syntax

```
cl2210 file[.c] [options] [/?]
```

Purpose

To compile and link a C-coded source file (`File.c`) for the slave DSP with the makefile `Ds2210.mk`. If you do not specify the file extension, the program searches a file of the relevant type in the current directory.

The action depends on the given file type:

- Local makefile (.mk)

  To execute the given local makefile (see `Ds2210.mk` and `Tmpl2210.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 `Ds2210.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 `Ds2210.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.exe` on page 275).

Options

The following command line options are available:

Option	Meaning
/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 instead of an assembly file.
/g	Enables symbolic debugging (using the CL30 options –g –as).
/l	Writes all outputs to the file `Cl2210.log`.
/n	Disables beep on error.
/p	Pauses execution of `Cl2210.exe` after errors. The Command Prompt window will not be closed automatically. This allows you to read error messages.
/s	Optimizes assembly code. Refer to speedy.exe on page 283.
/so <option>	Additional speedup option (can be used several times) to be passed to speedy.exe.
/x	Switches code optimizing off.
/?	Displays a list of the options available.

Error Message

The following error messages are defined for `Cl2210.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 HIL' and enter the following command to configure the compiler path: 'DsConfigTiEnv.exe'	The respective environment variable is not defined in the DOS environment. For more information, refer to How to Set the Compiler Path on page 268.
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.

Message	Meaning
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 directory.
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. Please 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.. 268

References

Ds2210.mk and Tmpl2210.mk.. 274

Ds2210.lk, DS2210_1.lk, DS2210_2.lk, DS2210_3.lk

Description

The linker command file `Ds2210.lk` is located in the directory `<RCP_HIL_InstallationPath>\Ds2210\SlaveDSP`. It is automatically used for linking if you use `Cl2210.exe` and if no local linker command file exists in the directory containing the application source file.

`Ds2210.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 `Ds2210.lk` is listed below to show the standard settings:

```
-stack 0x0100                                      /* 256 byte stack */
-heap  0x0100                                      /* 256 byte 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             */
   PMEM: org = 0x010000 len = 0x010000   /* 64KW primary memory     */
   DMEM: org = 0x600000 len = 0x004000   /* 16KW dual-port memory   */
}
/* section allocation into memory */
SECTIONS
{
   .startup:            > RAM0   /* startup code                    */
   .vectors:            > VECS   /* RESET vector                    */
   .trap:               > TRAP   /* TRAP vectors                    */
   .text:               > RAM0   /* C-code                          */
   .cinit:              > RAM0   /* initialization tables           */
   .const:              > RAM1   /* string literals and switch tables */
   .data:               > RAM1   /* initialized data                */
   .stack:              > RAM1   /* system stack                    */
   .bss:                > RAM1   /* global & static variables       */
   .sysmem:             > RAM1   /* 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 filename of the corresponding application and the suffix `.lk`. If `Cl2210.exe` detects a local linker command file in the directory 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 the built-in memory (RAM1), while the code section .text remains in the primary memory (PMEM). For example:

```
.text:          > PMEM        /* C-code                           */
.bss:          > RAM1         /* global & static variables         */
```

Both sections arranged in the internal memory blocks RAM0 and RAM1, 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 three following command files as your local linker command file:

Ds2210_1.lk If the size of the respective sections exceeds the limited size of RAM0 and RAM1, the linker command file `Ds2210_1.lk` can be used as the local linker command file. It will assign all sections to the internal memory RAM without considering the internal memory block boundaries.

Note

This will slow down the performance of the application.

Ds2210_2.lk and Ds2210_3.lk If the internal memory is not sufficient for the application, the linker command files `Ds2210_2.lk` or `Ds2210_3.lk` can be used as the local linker command file.

`Ds2210_2.lk` assigns the .bss and .stack sections to the internal memory and the other sections to the 64-KW primary memory PMEM. `Ds2210_3.lk` also assigns the .bss section to the PMEM.

Note

Using the PMEM will slow down the performance of the application, especially if the .bss section is placed into it.

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 256 words. For example:

```
-heap    0x0100                        /* 256 byte heap size  */
-stack   0x0100                        /* 256 byte stack size */
```

Related topics

References

Cl2210.exe..270

Ds2210.mk and Tmpl2210.mk

Purpose

Make file to compile or assemble a specified source file(s).

Description

When using `CL2210` it invokes `DSMAKE` with the default makefile `Ds2210.mk`.

Use `Tmpl2210.mk` as a template if you want to generate your own local makefiles. Copy this file to the local directory, rename the file (to the same name as your application to be built), specify the C- or assembler-coded source files, and call `Cl2210.exe`.

Related topics

References

coffconv.exe

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 *Slv2210_* 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 ds2210_slave_dsp_appl_load on page 173.

Note

The data array remains in the memory of the master processor after the application has been loaded to the slave DSP. A C file for the slave DSP application data should be used only if the slave DSP application must be reloaded without reloading the master application, for example, after a watchdog restart.

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 285.

Binary file The coffconv output file contains the application data and can be used by other conversion tools.

Example

```
coffconv demo.obj -c -t DS2210
```

The object file `Demo.obj` will be converted to the assembly file `Slv2210_demo.asm`.

Execution Time Information

Introduction

To calculate the execution time of the timer interrupt service routine (ISR) in DS2210 application programs you can use the `speedchk` macro.

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 DS2210 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, maximum and current execution time of the ISR. The execution time will be measured by speedchk in the unit timer ticks on the DSP and transferred to the master DSP via the DPMEM addresses 0x063FFB … 0x063FFD (as seen by the master DSP).

The `ds2210_slave_dsp_speedchk` function reads the minimum, maximum and current execution time from the DPMEM and supplies the execution times in µs to be displayed in ControlDesk.

Resolution

The resolution is one timer tick (that is, 50 ns for an 80-MHz DS2210 slave DSP) and the maximum error is one timer tick.

Related topics

References

ds2210_slave_dsp_speedchk..171

speedchk

Syntax

```
speedchk(i)
```

Purpose

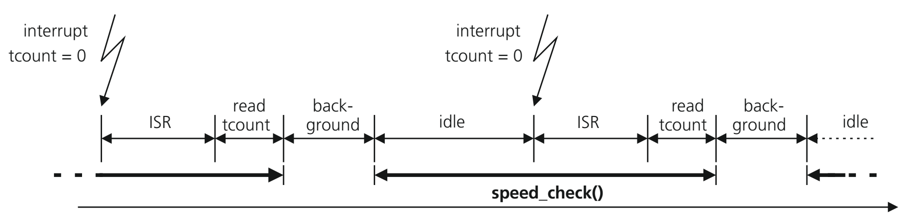
To get execution time information you have to include the `speedchk(i)` macro from the `Util2210.h` header file into the background loop of a DS2210 application program.

Description

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 for timer 0:

```
timer0(TS);                          /* initialize timer0 */
for (;;)
   speedchk(0);           /* include SPEEDCHK code for timer0 */
```

Related topics

References

Assembly Code Optimization

Introduction

To optimize the generated assembly code.

Where to go from here

Information in this section

Saving and Restoring the Context

Introduction

Most DS2210 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..283

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 DS2210 application programs floating-point to integer conversion is needed for the on-board D/A converters (refer to D/A Converter on page 227). 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..283

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 `Ds2210.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` on page 278 to evaluate the actual execution time.

Related topics

References

speedy.exe

Syntax

```
speedy [-v] [-k] [-o outfile] <asmfile>
```

Purpose

To automatically perform code optimization on the assembly level in DS2210 slave applications.

Description

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 `Cl2210.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

CI2210.exe..270

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.

How to Load a Slave Application

Basics

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 `CL2210.exe` compiles the slave application data and converts it to a C array by using the `coffconv` utility. The slave DSP application data of this C array must be loaded to the slave DSP using the `ds2210_slave_dsp_appl_load` function.

For the DS2210 board this is the default loading procedure.

Method

To load a slave application

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. Compile and convert the slave application by entering the following command:

   ```
   CL2210 test_prg.c /f
   ```

   This generates a C file called `Slv2210_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:

   ```
   /* DS2210 slave application data */
   #include Slv2210_test_prg.slc
   extern unsigned long test_prg[];
   void main(void)
   {
      init();        /* initialize master processor system */
      ds2210_init(DS2210_1_BASE);     /* initialize DS2210 */
      ds2210_slave_dsp_appl_load(DS2210_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

coffconv.exe 275
ds2210_slave_dsp_appl_load 173

Migration

Migrating DS2302 Applications

Introduction

If you want to use C programs written for the DS2302 DDS board for the DS2210 board, you have to consider some items and restrictions.

Items and restrictions

The following items and restrictions must be considered:

- Replace the include files `Ds2302.h` and `Util2302.h` with `Ds2210.h` and `Util2210.h`.
- The macros accessing the interrupts INT2 and INT3 cannot be used on the DS2210.
- The macros accessing the interrupts INT0 and INT1 cannot be used on the DS2210 board in the same way as on the DS2302 board.
- The feature of the DS2302 interrupt INT3 is realized on the DS2210 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 DS2210 board because the DS2210 contains 4 D/A converters. Use the `dac_out1`, `dac_out2`, `dac_out3`, and `dac_out4` macros instead. To receive the same output voltages as on the DS2302 board you have to divide the floating-point value for `dac_out` by two because the DS2210 converters have an output voltage range of ±20 V instead of ±10 V on the DS2302.

  Note

  The DS2210 D/A converter analog outputs 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.

- Due to different hardware the following functions and macros are not available on the DS2210 board or have a different functionality than on the DS2302 board:
  - `cvtdsp`
  - `cvtie3`
  - `dac_out`
  - `dig_in7`
  - `dig_out7`
  - `disable_int2`
  - `disable_int3`
  - `enable_int2`

- `enable_int3`
- `init_dig_out7`
- `int0_aux_status`
- `int0_status`
- `int1_ack`
- `int2_ack`
- `int2_init`
- `int3_ack`
- `int3_init`
- `int3_pending`
- `int_xf0`
- `int_xf1`
- `phs_bus_interrupt_request`
- `timer0_sync`

If you use any of these functions and macros in your DS2302 application you have to adapt the application to the capabilities of the DS2210.

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 `ds2210_can_communication_init` on page 302.

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. (ds2210_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 (DS2210 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]: ds2210_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

ds2210_can_channel_init..305

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

ds2210_canChannel

Purpose

The `ds2210_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;
} ds2210_canChannel;
```

Include file Can2210.h

Members

base The PHS-bus base address is provided by the function `ds2210_can_channel_init` or `ds2210_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 `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced`. This parameter is read-only.

btr0 Value of Bit Timing Register 0. This parameter is provided by the function `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced`. This parameter is read-only.

btr1 Value of Bit Timing Register 1. This parameter is provided by the function `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced`. This parameter is read-only.

frequency Frequency of the CAN controller. This parameter is provided by the function `ds2210_can_channel_init` or `ds2210_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
DS2210_CAN_STD	11-bit standard format, CAN 2.0A
DS2210_CAN_EXT	29-bit extended format, CAN 2.0B

This parameter is provided by the function `ds2210_can_channel_init` or `ds2210_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 `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced`. This parameter is read-only.

Related topics

References

ds2210_can_channel_init........305
ds2210_can_channel_init_advanced........307

ds2210_canService

Purpose

The `ds2210_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;
} ds2210_canService;
```

Include file

`Can2210.h`

Members

data0 Contains returned data from the function `ds2210_can_service_read`.

data1 Contains returned data from the function `ds2210_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 `ds2210_can_service_register`. This parameter is read-only.

queue This parameter is provided by the function `ds2210_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 `ds2210_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 `ds2210_can_channel_start` on page 310).

Predefined Symbol	Meaning
`DS2210_CAN_SERVICE_TX_OK`	Number of successfully sent TX/RM/RQTX messages
`DS2210_CAN_SERVICE_RX_OK`	Number of successfully received RX/RQRX messages
`DS2210_CAN_SERVICE_CRC_ERR`	Number of CRC errors
`DS2210_CAN_SERVICE_ACK_ERR`	Number of acknowledge errors
`DS2210_CAN_SERVICE_FORM_ERR`	Number of format errors
`DS2210_CAN_SERVICE_BIT1_ERR`	Number of Bit1 errors
`DS2210_CAN_SERVICE_BIT0_ERR`	Number of Bit0 errors
`DS2210_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 `ds2210_can_channel_start` on page 310).

Predefined Symbol	Meaning
`DS2210_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.
`DS2210_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.
`DS2210_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.
`DS2210_CAN_SERVICE_BUSSTATUS`	Status of the CAN controller. For the predefined values, see below.
`DS2210_CAN_SERVICE_STDMASK`	Status of the global standard mask register

Predefined Symbol	Meaning
DS2210_CAN_SERVICE_EXTMASK	Status of the global extended mask register
DS2210_CAN_SERVICE_MSG_MASK15	Status of the message 15 mask register
DS2210_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 `ds2210_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.
DS2210_CAN_SERVICE_VERSION	Version number of the CAN firmware.

index Table index already allocated by the register function `ds2210_can_service_register`. This parameter is read-only.

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

DS2210_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 `ds2210_can_channel_init`). ▪ Initialize the `mb15_format` parameter with the other format when calling `ds2210_can_channel_init` or `ds2210_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.

DS2210_CAN_SERVICE_BUSSTATUS Provides bus status information; the following symbols are predefined:

Predefined Symbol	Meaning
DS2210_CAN_BUSOFF_STATE	The CAN channel disconnects itself from the CAN bus. Use `ds2210_can_channel_BOff_return` to recover from the bus off state.
DS2210_CAN_WARN_STATE	The CAN controller is still active. The CAN controller recovers from this state automatically.
DS2210_CAN_ACTIVE_STATE	The CAN controller is active.

Note

After calling `ds2210_can_channel_BOff_return`, the service DS2210_CAN_SERVICE_BUSSTATUS will not return DS2210_CAN_BUSOFF_STATE.

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

```
ds2210_canService* service;
UInt16 overflow;
...
service = ds2210_can_service_register(
            txCh, DS2210_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT);
...
ds2210_can_service_request( service );
ds2210_can_service_read( service );
overflow = service->txqueue_overflowcnt_std;
```

Related topics

References

ds2210_canMsg

Purpose

The `ds2210_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;
   ds2210_canChannel*canChannel;
   ds2210_canService *msgService;
    } ds2210_canMsg;
```

Include file

`Can2210.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 `ds2210_can_msg_read` if the message was registered using the `inform` parameter `DS2210_CAN_TIMECOUNT_INFO`.

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

This parameter is updated by the function `ds2210_can_msg_read` if the message was registered with the `inform` parameter `DS2210_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 `ds2210_can_msg_read` if the message was registered with the `inform` parameter `DS2210_CAN_DELAYCOUNT_INFO`.

processed Processed flag of the message. This parameter is updated by the function `ds2210_can_msg_read`. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_CAN_PROCESSED`	The message has been sent/received since the last execution call.
`DS2210_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 `ds2210_can_msg_read` if the message was registered with the `inform` parameter `DS2210_CAN_DATA_INFO`.

data[8] Buffer for CAN message data. This data is updated by the function `ds2210_can_msg_read` if the message was registered with the `inform` parameter `DS2210_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
`DS2210_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2210_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
`DS2210_CAN_TX`	Transmit message registered by `ds2210_can_msg_tx_register`
`DS2210_CAN_RX`	Receive message registered by `ds2210_can_msg_rx_register`
`DS2210_CAN_RM`	Remote message registered by `ds2210_can_msg_rm_register`
`DS2210_CAN_RQTX`	RQTX message registered by `ds2210_can_msg_rqtx_register`
`DS2210_CAN_RQRX`	RQRX message registered by `ds2210_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 `ds2210_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
`DS2210_CAN_NO_INFO`	Returns no information.
`DS2210_CAN_DATA_INFO`	Updates the data and datalen parameters (needed for receive and request (RQRX) messages).
`DS2210_CAN_MSG_INFO`	Updates the message identifier and the message format for RM, RQ, TX, and RX messages.
`DS2210_CAN_TIMECOUNT_INFO`	Updates the timestamp and the deltatime parameters.
`DS2210_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 `ds2210_canChannel` structure where the message object is installed. This parameter is read-only. Refer to `ds2210_canChannel` on page 293.

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`**).

ds2210_can_communication_init

Syntax

```
void ds2210_can_communication_init(
      const UInt32 base,
      const UInt32 bufferwarn)
```

Include file

`Can2210.h`

Purpose

To initialize communication between the master and the slave DS2210.

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 DS2210 board.

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

Predefined Symbol	Meaning
`DS2210_CAN_INT_DISABLE`	The bufferwarn subinterrupt is disabled.
`DS2210_CAN_INT_ENABLE`	The bufferwarn subinterrupt is enabled.

Return value

None

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2210_can_communication_init(x,..) memory: allocation error on master	Memory allocation error. No free memory on the master.
104	Error	ds2210_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	ds2210_can_communication_init(x,..) subint: init failed by master	Master subinterrupt initialization failed. There is not enough memory available.
106	Error	ds2210_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	ds2210_can_communication_init(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` when registering messages or services.
109	Error	ds2210_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	ds2210_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 370
- Example of Handling Request and Remote Messages on page 372
- Example of Using Subinterrupts on page 374

Related topics

References

CAN Channel Handling

Introduction

Provides information on handling CAN interfaces, called *CAN channels*.

Where to go from here

Information in this section

ds2210_can_channel_init

Syntax

```
ds2210_canChannel* ds2210_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

`Can2210.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 `ds2210_can_channel_start` function to complete the CAN channel initialization.

Description

If no error occurs, `ds2210_can_channel_init` returns a pointer to the `ds2210_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 DS2210 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
`DS2210_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2210_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
`DS2210_CAN_NO_SUBINT`	No interrupt for bus off

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

Predefined Symbol	Meaning
`DS2210_CAN_TERMINATION_ON`	Bus termination activated
`DS2210_CAN_TERMINATION_OFF`	Bus termination deactivated

Return value

canChannel Pointer to the `ds2210_canChannel`

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2210_can_channel_init_advanced(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
104	Error	ds2210_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 DS2210_CAN_NO_ERROR.
106	Error	ds2210_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	ds2210_can_channel_init_advanced(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` when registering messages or services.
123	Error	ds2210_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	ds2210_can_channel_init_advanced(x,..) format: wrong format.	Only the symbols `DS2210_CAN_STD` and `DS2210_CAN_EXT` are allowed for the parameter mb15_format.
141	Error	ds2210_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	ds2210_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 370
- Example of Handling Request and Remote Messages on page 372
- Example of Using Subinterrupts on page 374

Related topics

References

ds2210_can_channel_init_advanced

Syntax

```
ds2210_canChannel* ds2210_can_channel_init_advanced(
      const UInt32 mb15_format,
      const Int32 busoff_subinterrupt,
      const UInt32 termination);
```

Include file

`Can2210.h`

Purpose

To perform the initialization of a CAN channel with parameters.

If no error occurs, the function returns a pointer to the `ds2210_canChannel` structure.

> **Note**
>
> You have to call `ds2210_can_channel_start` to complete the CAN channel initialization.

Description

Use the returned handle when calling one of the following functions: `ds2210_can_channel_start`, `ds2210_can_channel_all_sleep`, `ds2210_can_channel_all_wakeup`, `ds2210_can_channel_BOff_go`, `ds2210_can_channel_BOff_return`, `ds2210_can_channel_set`, `ds2210_can_msg_tx_register`, `ds2210_can_msg_rx_register`, `ds2210_can_msg_rqtx_register`, `ds2210_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 `ds2210_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 DS2210 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
`DS2210_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2210_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
`DS2210_CAN_NO_SUBINT`	No interrupt for bus off

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

Predefined Symbol	Meaning
`DS2210_CAN_TERMINATION_ON`	Bus termination activated
`DS2210_CAN_TERMINATION_OFF`	Bus termination deactivated

Return value

canChannel Specifies the pointer to the `ds2210_canChannel` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2210_can_channel_init_advance(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
104	Error	ds2210_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 DS2210_CAN_NO_ERROR.
106	Error	ds2210_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	ds2210_can_channel_init_advanced(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 **`ds2210_can_msg_sleep`** or **`ds2210_can_channel_all_sleep`** when registering messages or services.
123	Error	ds2210_can_channel_init_advanced(x,..) channel: use range 0..1	Use a CAN channel within the range of 0 … 1.
140	Error	ds2210_can_channel_init_advanced(x,..) format: wrong format	Only the `DS2210_CAN_STD` and `DS2210_CAN_EXT` symbols are allowed for the mb15_format parameter.
141	Error	ds2210_can_channel_init_advanced(x,..) subint: use range 0..14	The subinterrupt number must be within the range of 0 … 14.

Example

```
ds2210_canChannel* CH;
CH = ds2210_can_channel_init_advanced(
   DS2210_1_BASE,      /* PHS-bus base address  */
   0,                  /* channel 0 */
   0x80,               /* BTR0      */
   0x6F,               /* BTR1*     */
   DS2210_CAN_STD,     /* use mailbox 15 to receive only    */
                       /* CAN messages with standard format */
   DS2210_CAN_NO_SUBINT, /* generate no subinterrupt when  */
                       /* the CAN controller goes in the    */
                       /* bus off state                     */

   DS2210_CAN_TERMINATION_ON   /* Bus termination activated */
);
```

Related topics

References

ds2210_can_channel_start

Syntax

```
void ds2210_can_channel_start(
      const ds2210_canChannel* canCh,
      const UInt32 status_int);
```

Include file

`Can2210.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 DS2210 slave interrupts will be enabled. Use the returned handle from the function `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2210_canChannel` structure.

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

Predefined Symbol	Meaning
`DS2210_CAN_INT_DISABLE`	No status interrupt will be generated.
`DS2210_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 DS2210 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 DS2210 would be unnecessarily burdened.

Return value

None

Messages

The following messages are defined:

ID	Type	Message	Description
104	Error	ds2210_can_channel_start 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 370
- Example of Handling Request and Remote Messages on page 372
- Example of Using Subinterrupts on page 374

Related topics

References

ds2210_can_channel_init ... 305
ds2210_can_channel_init_advanced ... 307
ds2210_can_msg_sleep ... 351

ds2210_can_channel_all_sleep

Syntax

```
Int32 ds2210_can_channel_all_sleep(
      const ds2210_canChannel* canCh);
```

Include file

`Can2210.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 `ds2210_can_channel_all_wakeup`.

Use the returned handle from the `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced` function to call this function.

Parameters

canCh Specifies the pointer to the `ds2210_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
DS2210_CAN_NO_ERROR	The function was performed without error.
DS2210_CAN_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns DS2210_CAN_NO_ERROR.

Function execution times

For information, refer to Function Execution Times on page 385.

Example

```
ds2210_can_channel_all_sleep(canCh);
```

Related topics

References

ds2210_can_channel_all_wakeup

Syntax

```
Int32 ds2210_can_channel_all_wakeup(
      const ds2210_canChannel* canCh);
```

Include file

`Can2210.h`

Purpose

To reactivate all messages that were deactivated by calling the functions `ds2210_can_channel_all_sleep` and `ds2210_can_msg_sleep`.

Description

Use the returned handle from the function `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2210_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
`DS2210_CAN_NO_ERROR`	The function has been performed without error.
`DS2210_CAN_BUFFER_OVERFLOW`	The communication buffer occurred. Repeat the function until it returns DS2210_CAN_NO_ERROR.

Function execution times

For information, refer to Function Execution Times on page 385.

Example

```
ds2210_can_channel_all_wakeup(canCh);
```

Related topics

References

ds2210_can_channel_BOff_go

Syntax

```
Int32 ds2210_can_channel_BOff_go(
      const ds2210_canChannel* canCh);
```

Include file

`Can2210.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 `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2210_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
DS2210_CAN_NO_ERROR	The function was performed without error.
DS2210_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 DS2210_CAN_NO_ERROR.

Function execution times

For information, refer to Function Execution Times on page 385.

Example

```
ds2210_can_channel_BOff_go(canCh);
```

Related topics

References

ds2210_can_channel_BOff_return

Syntax

```
Int32 ds2210_can_channel_BOff_return(
      const ds2210_canChannel* canCh);
```

Include file

`Can2210.h`

Purpose

To reset the slave DS2210 CAN channel from the bus off state.

Use the returned handle from the function `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2210_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
DS2210_CAN_NO_ERROR	The function was performed without error.
DS2210_CAN_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns DS2210_CAN_NO_ERROR.

Function execution times

For information, refer to Function Execution Times on page 385.

Example

```
ds2210_can_channel_BOff_return(canCh);
```

Related topics

References

ds2210_can_channel_BOff_go.......313
ds2210_can_channel_init.......305
ds2210_can_channel_init_advanced.......307

ds2210_can_channel_set

Syntax

```
Int32 ds2210_can_channel_set(
      const ds2210_canChannel* canCh,
      const UInt32 mask_type,
      const UInt32 mask_value);
```

Include file

`Can2210.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 `ds2210_can_channel_init` or `ds2210_can_channel_init_advanced` to call this function.

Parameters

canCh Specifies the pointer to the `ds2210_canChannel` structure.

mask_type Specifies the mask type. The following symbols are predefined:

Predefined Symbol	Meaning
DS2210_CAN_CHANNEL_SET_MASK15	Sets the Message 15 Mask Register.
DS2210_CAN_CHANNEL_SET_ARBMASK15	Sets the Arbitration Register for mailbox 15.
DS2210_CAN_CHANNEL_SET_TERMINATION	Set the termination resistor for the channel.
DS2210_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
DS2210_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.
DS2210_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.
DS2210_CAN_CHANNEL_SET_TERMINATION	Use one of the following symbols to set the termination resistor: `DS2210_CHANNEL_TERMINATION_ON` or `DS2210_CHANNEL_TERMINATION_OFF`
DS2210_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 361.

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
DS2210_CAN_NO_ERROR	The function was performed without error.
DS2210_CAN_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns DS2210_CAN_NO_ERROR.
DS2210_CAN_BAUDRATE_L_ERROR	The baud rate is too low. The operation is aborted.
DS2210_CAN_BAUDRATE_H_ERROR	The baud rate is too high. The operation is aborted.
DS2210_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	CAN2210 (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

```
ds2210_can_channel_set(
         canCh,
         DS2210_CAN_CHANNEL_SET_MASK15,
         0xFFFFFFFE);
/* Set the lowest bit of the Message 15 Mask Register */
/* to "don't care" */
```

Related topics

References

ds2210_can_channel_txqueue_clear

Syntax

```
Int32 ds2210_can_channel_txqueue_clear(
      const ds2210_canChannel* canCh);
```

Include file

`Can2210.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 `ds2210_canChannel` structure.

Return value

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

Predefined Symbol	Meaning
`DS2210_CAN_NO_ERROR`	The function was performed without error.
`DS2210_CAN_BUFFER_OVERFLOW`	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns DS2210_CAN_NO_ERROR.

CAN Message Handling

Introduction

To handle different kinds of CAN messages.

Where to go from here

Information in this section

ds2210_can_msg_tx_register

Syntax

```
ds2210_canMsg* ds2210_can_msg_tx_register(
      const ds2210_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

`Can2210.h`

Purpose

To register a transmit message on the slave DS2210.

Description

If no error occurs, the function returns a pointer to the `ds2210_canMsg` structure.

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

- `ds2210_can_msg_write` to write new data to the message
- `ds2210_can_msg_read` to read the returned timestamps
- `ds2210_can_msg_send` to send the message with new data
- `ds2210_can_msg_trigger` to send the message
- `ds2210_can_msg_sleep` to deactivate the message
- `ds2210_can_msg_wakeup` to reactivate the message
- `ds2210_can_msg_clear` to clear the message object data
- `ds2210_can_msg_processed_register` to register the processed function
- `ds2210_can_msg_processed_request` to request the processed function
- `ds2210_can_msg_processed_read` to read the returned data

Note

You must call `ds2210_can_msg_write` to make the message valid for the CAN channel.

Parameters

canCh Specifies the pointer to the `ds2210_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
DS2210_CAN_STD	11-bit standard format, CAN 2.0A
DS2210_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
DS2210_CAN_NO_INFO	Returns no information.
DS2210_CAN_MSG_INFO	Updates the message identifier and the message format.

Predefined Symbol	Meaning
`DS2210_CAN_TIMECOUNT_INFO`	Updates the timestamp and the `deltatime` parameters.
`DS2210_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 (DS2210_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
`DS2210_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 `ds2210_can_msg_trigger`:

Predefined Symbol	Meaning
`DS2210_CAN_TRIGGER_MSG`	Calls `ds2210_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
`DS2210_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 `ds2210_canMsg` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2210_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	ds2210_can_msg_tx_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2210_can_msg_tx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2210_CAN_AUTO_INDEX.
104	Error	ds2210_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	ds2210_can_msg_tx_register(x,..) slave: not responding	The slave did not finish the initialization within one second.
107	Error	ds2210_can_msg_tx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` when registering messages or services.
140	Error	ds2210_can_msg_tx_register(x,..) format: wrong format	Only the symbols `DS2210_CAN_STD` and `DS2210_CAN_EXT` are allowed for the parameter `format`.
141	Error	ds2210_can_msg_tx_register(x,..) subint: use range 0..14!	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2210_can_msg_tx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2210_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 `ds2210_canService` on page 295.
144	Error	ds2210_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	ds2210_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	ds2210_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	ds2210_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	ds2210_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	ds2210_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	ds2210_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 `ds2210_can_channel_init` or `ds2210_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 370 and Example of Using Subinterrupts on page 374.

Related topics

References

ds2210_can_msg_rx_register

Syntax

```
ds2210_canMsg* ds2210_can_msg_rx_register(
      const ds2210_canChannel* canCh,
      const Int32 queue,
      const UInt32 identifier,
      const UInt32 format,
      const UInt32 inform,
      const Int32 subinterrupt);
```

Include file

`Can2210.h`

Purpose

To register a receive message on the slave DS2210.

Description

If no error occurs, `ds2210_can_msg_rx_register` returns a pointer to the `ds2210_canMsg` structure.

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

- `ds2210_can_msg_read` to read the returned data and timestamps
- `ds2210_can_msg_sleep` to deactivate the message
- `ds2210_can_msg_wakeup` to reactivate the message
- `ds2210_can_msg_clear` to clear the message data
- `ds2210_can_msg_processed_register` to register the processed function
- `ds2210_can_msg_processed_request` to request the processed function
- `ds2210_can_msg_processed_read` to read the returned data

Parameters

canCh Specifies the pointer to the `ds2210_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
`DS2210_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2210_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
`DS2210_CAN_NO_INFO`	Returns no information.
`DS2210_CAN_DATA_INFO`	Updates the data and datalen parameters (needed for receive and request (RQRX) messages).
`DS2210_CAN_MSG_INFO`	Updates the message identifier and the message format.
`DS2210_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 (DS2210_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
`DS2210_CAN_NO_SUBINT`	No interrupt for the receive message

Return value

canMsg This function returns the pointer to the `ds2210_canMsg` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2210_can_msg_rx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2210_can_msg_rx_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2210_can_msg_rx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2210_CAN_AUTO_INDEX.
104	Error	ds2210_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	ds2210_can_msg_rx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2210_can_msg_rx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` when registering messages or services.
140	Error	ds2210_can_msg_rx_register(x,..) format: wrong format	Only the symbols `DS2210_CAN_STD` and `DS2210_CAN_EXT` are allowed for the parameter `format`.
141	Error	ds2210_can_msg_rx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2210_can_msg_rx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2210_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 `ds2210_canService` on page 295.
144	Error	ds2210_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	ds2210_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 `ds2210_can_channel_init` or `ds2210_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 370 and Example of Using Subinterrupts on page 374.

```
ds2210_canMsg* rxMsg = ds2210_can_msg_rx_register(
         canCh,
         0,
         0x123,
         DS2210_CAN_STD,
         DS2210_CAN_DATA_INFO,
         DS2210_CAN_NO_SUBINT);
```

Related topics

References

ds2210_can_msg_rqtx_register

Syntax

```
ds2210_canMsg* ds2210_can_msg_rqtx_register(
      const ds2210_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

`Can2210.h`

Purpose

To register a request transmission (RQTX) message on the slave DS2210.

Description

Use this function to register a request message. Use the function `ds2210_can_msg_rqtx_register` to register a function that receives the requested data. If no error occurs, `ds2210_can_msg_rqtx_register` returns a pointer to the `ds2210_canMsg` structure.

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

Function	Description
`ds2210_can_msg_rqtx_activate`	to activate the message
`ds2210_can_msg_read`	To read the returned time stamps.
`ds2210_can_msg_sleep`	To deactivate the message.
`ds2210_can_msg_wakeup`	To reactivate the message.
`ds2210_can_msg_trigger`	To send the request message.
`ds2210_can_msg_clear`	To clear the message object data.
`ds2210_can_msg_processed_register`	To register the processed function.
`ds2210_can_msg_processed_request`	To request the processed function.
`ds2210_can_msg_processed_read`	To read the returned data.

Note

You must call `ds2210_can_msg_rqtx_activate` to make the message valid for the CAN channel.

Parameters

canCh Specifies the pointer to the `ds2210_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
`DS2210_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2210_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
DS2210_CAN_NO_INFO	Returns no information.
DS2210_CAN_MSG_INFO	Updates the message identifier and the message format.
DS2210_CAN_TIMECOUNT_INFO	Updates the timestamp and deltatime parameters.
DS2210_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 (DS2210_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
DS2210_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 `ds2210_can_msg_trigger`:

Predefined Symbol	Meaning
DS2210_CAN_TRIGGER_MSG	Calls `ds2210_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
DS2210_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 ds2210_canMsg structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2210_can_msg_rqtx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2210_can_msg_rqtx_register(x,..) queue: Illegal communication queue	There is no communication channel with this queue number.
103	Error	ds2210_can_msg_rqtx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2210_CAN_AUTO_INDEX.
104	Error	ds2210_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	ds2210_can_msg_rqtx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2210_can_msg_rqtx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` when registering messages or services.
140	Error	ds2210_can_msg_rqtx_register(x,..) format: wrong format	Only the symbols DS2210_CAN_STD and DS2210_CAN_EXT are allowed for the parameter format.
141	Error	ds2210_can_msg_rqtx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2210_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_registerds2210_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 DS2210_CAN_SERVICE_MAILBOX_ERR.
144	Error	ds2210_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	ds2210_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 `ds2210_can_channel_init` or `ds2210_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 372.

```
ds2210_canMsg* rqtxMsg = ds2210_can_msg_rqtx_register(
         canCh,
         0,
         0x123,
         DS2210_CAN_STD,
         DS2210_CAN_TIMECOUNT_INFO,
         DS2210_CAN_NO_SUBINT,
         1.5,
         0.3,
         DS2210_CAN_TIMEOUT_NORMAL);
```

Related topics

References

ds2210_can_msg_rqrx_register

Syntax

```
ds2210_canMsg* ds2210_can_msg_rqrx_register(
      const ds2210_canMsg* rqtxMsg,
      const UInt32 inform,
      const Int32 subinterrupt);
```

Include file

`Can2210.h`

Purpose

To register an RQRX message on the slave DS2210.

Description

Use this message to receive the data requested with an RQTX message. If no error occurs, `ds2210_can_msg_rqrx_register` returns a pointer to the `ds2210_canMsg` structure.

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

- `ds2210_can_msg_read` to read the returned data and time stamps
- `ds2210_can_msg_sleep` to deactivate the message
- `ds2210_can_msg_wakeup` to reactivate the message
- `ds2210_can_msg_clear` to clear the message object data
- `ds2210_can_msg_processed_register` to register the processed function
- `ds2210_can_msg_processed_request` to request the processed function
- `ds2210_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
`DS2210_CAN_NO_INFO`	Returns no information.
`DS2210_CAN_DATA_INFO`	Updates the data and datalen parameters (needed for receive and request (RQRX) messages).
`DS2210_CAN_MSG_INFO`	Updates the message identifier and the message format.
`DS2210_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 (DS2210_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
`DS2210_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	ds2210_can_msg_rqrx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2210_can_msg_rqrx_register(x,..) queue: Illegal communication queue	There is no communication channel with this queue number.
103	Error	ds2210_can_msg_rqrx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2210_CAN_AUTO_INDEX.
104	Error	ds2210_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	ds2210_can_msg_rqrx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2210_can_msg_rqrx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` when registering messages or services.
140	Error	ds2210_can_msg_rqrx_register(x,..) format: wrong format	Only the symbols `DS2210_CAN_STD` and `DS2210_CAN_EXT` are allowed for the parameter format.
141	Error	ds2210_can_msg_rqrx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2210_can_msg_rqrx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2210_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 `ds2210_canService` on page 295.
144	Error	ds2210_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	ds2210_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 `ds2210_can_channel_init` or `ds2210_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 372.

```
ds2210_canMsg* rqrxMsg = ds2210_can_msg_rqrx_register(
         rqtxMsg,
         DS2210_CAN_DATA_INFO,
         DS2210_CAN_NO_SUBINT);
```

Related topics

References

ds2210_can_msg_rm_register

Syntax

```
ds2210_canMsg* ds2210_can_msg_rm_register(
      const ds2210_canChannel* canCh,
      const Int32 queue,
      const UInt32 identifier,
      const UInt32 format,
      const UInt32 inform,
      const Int32 subinterrupt);
```

Include file

`Can2210.h`

Purpose

To register a remote message on the slave DS2210.

Description

If no error occurs, the function returns a pointer to the `ds2210_canMsg` structure.

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

- `ds2210_can_msg_write` to support the remote message with data
- `ds2210_can_msg_read` to read the returned time stamps
- `ds2210_can_msg_sleep` to deactivate the message

- `ds2210_can_msg_wakeup` to reactivate the message
- `ds2210_can_msg_clear` to clear the message object data
- `ds2210_can_msg_processed_register` to register the processed function
- `ds2210_can_msg_processed_request` to request the processed function
- `ds2210_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 DS2210 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 `ds2210_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
`DS2210_CAN_STD`	11-bit standard format, CAN 2.0A
`DS2210_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
`DS2210_CAN_NO_INFO`	Returns no information.
`DS2210_CAN_MSG_INFO`	Updates the message identifier and the message format.
`DS2210_CAN_TIMECOUNT_INFO`	Updates the timestamp and the **`deltatime`** parameters.
`DS2210_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 (DS2210_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
`DS2210_CAN_NO_SUBINT`	No interrupt for the RM message

Return value

canMsg This function returns the pointer to the `ds2210_canMsg` structure.

Messages

The following error and warning messages are defined:

ID	Type	Message	Description
101	Error	ds2210_can_msg_rm_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2210_can_msg_rm_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2210_can_msg_rm_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2210_CAN_AUTO_INDEX.
104	Error	ds2210_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	ds2210_can_msg_rm_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2210_can_msg_rm_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` when registering messages or services.
140	Error	ds2210_can_msg_rm_register(x,..) format: wrong format	Only the symbols DS2210_CAN_STD and DS2210_CAN_EXT are allowed for the parameter format.
141	Error	ds2210_can_msg_rm_register(x,..) subint: use range 0...14.	The subinterrupt number must be within the range 0 … 14.
142	Error	ds2210_can_msg_rm_register(x,...) subint: used for busoff.	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	ds2210_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 `ds2210_canService` on page 295.
144	Error	ds2210_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	ds2210_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	ds2210_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 `ds2210_can_channel_init` or `ds2210_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 372.

```
ds2210_canMsg* rmMsg = ds2210_can_msg_rm_register(
         canCh,
         0,
         0x123,
         DS2210_CAN_STD,
         DS2210_CAN_TIMECOUNT_INFO,
         DS2210_CAN_NO_SUBINT);
```

Related topics

References

ds2210_can_msg_set

Syntax

```
Int32 ds2210_can_msg_set(
      ds2210_canMsg* msg,
      const UInt32 type,
      const void* value );
```

Include file

`Can2210.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 = DS2210_CAN_MSG_MASK),
- Set the send period for a TX or RQ message (type = DS2210_CAN_MSG_PERIOD),
- Set the identifier for a TX or RQ message (type = DS2210_CAN_MSG_ID) or
- Set the queue depth for a message (type = DS2210_CAN_MSG_QUEUE).
- Set the length for a message (type = DS2210_CAN_MSG_LEN).

Note

For DS2210_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 `ds2210_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
`DS2210_CAN_MSG_MASK`	To set the arbitrary mask for an RX message
`DS2210_CAN_MSG_PERIOD`	To set the send period for a TX or RQ message
`DS2210_CAN_MSG_ID`	To set the identifier for a TX or RQ message
`DS2210_CAN_MSG_QUEUE`	To set the queue depth for a message
`DS2210_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 DS2210_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
`DS2210_CAN_NO_ERROR`	The function was performed without error.
`DS2210_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 DS2210_CAN_NO_ERROR.
`DS2210_CAN_TYPE_ERROR`	The operation is not allowed for the specified message object.
`DS2210_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 ds2210_can_msg_set.

```
UInt32 mask = 0xFFFFFFF0; // Sets the last four bits to
                          // "Don't Care".
ds2210_can_msg_set( msg, DS2210_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 `ds2210_can_msg_set(msg, DS2210_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 DS2210_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 = ds2210_can_msg_xx_register(...)
  ```

  By default, `myMsg` stores only one message.
- Define a message queue of length *n* for `myMsg`

  ```
  ds2210_can_msg_set(myMsg, DS2210_CAN_MSG_QUEUE, &n)
  ```

- Call `ds2210_can_msg_read(myMsg)` repeatedly until the function returns DS2210_CAN_NO_DATA.

```
UInt32 n;
canMsg = ds2210_can_msg_rx_register( canCh,…
n = 5000;
ds2210_can_msg_set(canMsg, DS2210_CAN_MSG_QUEUE, &n);
...
while(DS2210_CAN_NO_DATA != (error =
             ds2210_can_msg_read(canMsg)))
{
   if(DS2210_CAN_DATA_LOST == error)
   {
      /* error handling */
   }
   else if(DS2210_CAN_NO_ERROR == error)
   {
      /* process the message */
   }
   else /* DS2210_CAN_NO_DATA == error */
   {
      /* no further CAN-messages */
   }
}
```

Related topics

References

ds2210_can_msg_rqtx_activate

Syntax

```
Int32 ds2210_can_msg_rqtx_activate(
     const ds2210_canMsg* canMsg);
```

Include file

`Can2210.h`

Purpose

To activate the request transmission message on the slave DS2210 registered by `ds2210_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 `ds2210_can_msg_trigger` for acyclic sending. Use the returned handle from the function `ds2210_can_msg_rqtx_register` to call this function.

Parameters

canMsg Specifies the pointer to the `ds2210_canMsg` structure.

Return value

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

Predefined Symbol	Meaning
`DS2210_CAN_NO_ERROR`	The function was performed without error.
`DS2210_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 DS2210_CAN_NO_ERROR.
`DS2210_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 372.

Related topics

References

ds2210_can_msg_rqtx_register.. 327
ds2210_can_msg_trigger.. 355

ds2210_can_msg_write

Syntax

```
Int32 ds2210can_msg_write(
      const ds2210_canMsg* canMsg,
      const UInt32 datalen,
      const UInt32* data);
```

Include file

`Can2210.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 DS2210_CAN_TRIGGER_MSG is sent only when calling `ds2210_can_msg_trigger` or `ds2210_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 `ds2210_can_msg_tx_register` or `ds2210_can_msg_rm_register` to call this function.

Parameters

canMsg Specifies the pointer to the `ds2210_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
DS2210_CAN_NO_ERROR	The function has been performed without error.
DS2210_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 DS2210_CAN_NO_ERROR.
DS2210_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 385.

Example

For examples, refer to:

- Example of Handling Transmit and Receive Messages on page 370
- Example of Handling Request and Remote Messages on page 372
- Example of Using Subinterrupts on page 374

Related topics

References

ds2210_can_msg_send

Syntax

```
Int32 ds2210_can_msg_send(
      const ds2210_canMsg* canMsg,
      const UInt32 datalen,
      const UInt32* data,
      const Float32 delay);
```

Include file

`Can2210.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 `ds2210_can_msg_tx_register`. Then `ds2210_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 `ds2210_can_msg_tx_register` to call this function.

Note

Suppose the `ds2210_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 `ds2210_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
DS2210_CAN_NO_ERROR	The function was performed without error.
DS2210_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 DS2210_CAN_NO_ERROR.
DS2210_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 385.

Example

```
UInt32 txData[8] = {1,2,3,4,5,6,7,8};
ds2210_can_msg_send (txMsg, 8, txData, 0.005);
```

Related topics

References

ds2210_can_msg_send_id

Syntax

```
Int32 ds2210_can_msg_send_id (
      ds2210_canMsg* canMsg,
      const UInt32 id,
      const UInt32 datalen,
      const UInt8* data,
      const Float32 delay);
```

Include file

`Can2210.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 `ds2210_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
`DS2210_CAN_NO_ERROR`	The function was performed without error.
`DS2210_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 DS2210_CAN_NO_ERROR.
`DS2210_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 `ds2210_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 `ds2210_can_msg_send_id` function lets you send any message ID with one registered message.

```
ds2210_can_msg_send_id(msg, 0x123, data, 8, 0.001)
```

Related topics

References

ds2210_can_msg_queue_level 346
ds2210_can_msg_send 343
ds2210_can_msg_tx_register 320

ds2210_can_msg_queue_level

Syntax

```
Int32 ds2210_can_msg_queue_level (
      ds2210_canMsg* canMsg);
```

Include file

`Can2210.h`

Purpose

To return the number of messages stored in the message queue allocated on the master with `ds2210_can_msg_set(msg, DS2210_CAN_MSG_QUEUE, &size)`.

Description

Use `ds2210_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 `ds2210_canMsg` structure.

Return value

This function returns the number of messages in the message queue.

Related topics

References

ds2210_can_msg_read 353
ds2210_can_msg_set 337

ds2210_can_msg_txqueue_init

Syntax

```
Int32 ds2210_can_msg_txqueue_init(
      ds2210_canMsg* canMsg,
      const UInt32 overrun_policy,
      Float32 delay);
```

Include file

`Can2210.h`

Purpose

To initialize the transmit queue that is used to queue messages sent by the `ds2210_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 `ds2210_can_msg_send_id_queued` function are stored. The queue stores up to 64 message entries.

Parameter

canMsg Specifies the pointer to the `ds2210_canMsg` structure.

overrun_policy Selects the overrun policy of the transmit queue. The following symbols are predefined:

Predefined Symbol	Meaning
`DS2210_CAN_TXQUEUE_OVERRUN_OVERWRITE`	The oldest message is overwritten.
`DS2210_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 `ds2210_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
`DS2210_CAN_NO_ERROR`	The transmit queue was initialized successfully.
`DS2210_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 DS2210_CAN_NO_ERROR.

Predefined Symbol	Meaning
DS2210_CAN_TXQUEUE_INIT_NOT_REG_ERROR	The message (canMsg) was not registered. The operation is aborted.
DS2210_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	ds2210_can_msg_txqueue_init(): TX message is not registered	The message was not registered successfully.
155	Error	ds2210_can_msg_txqueue_init(): not a TX message	The specified message is not a TX message.
301	Warning	ds2210_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()
{
  ds2210_canMsg* txMsg;
...
  txMsg = ds2210_can_msg_tx_register( txCh,
                    2, 0x1, DS2210_CAN_STD,
                    DS2210_CAN_TIMECOUNT_INFO |
                    DS2210_CAN_MSG_INFO,
                    1, 0.0,
                    DS2210_CAN_TRIGGER_MSG, 0 );
  ds2210_can_msg_txqueue_init (
          txMsg, DS2210_CAN_TXQUEUE_OVERRUN_OVERWRITE, 0.01);
...
}
```

Related topics

References

ds2210_can_msg_send_id_queued...349

ds2210_can_msg_send_id_queued

Syntax

```
Int32 ds2210_can_msg_send_id_queued(
      ds2210_canMsg* canMsg,
      const UInt32 id,
      const UInt32 data_len,
      const UInt32* data);
```

Include file

`Can2210.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 `ds2210_can_msg_txqueue_init` on page 346.

The `DS2210_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 `ds2210_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
`DS2210_CAN_NO_ERROR`	The transmit queue was initialized successfully.
`DS2210_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 DS2210_CAN_NO_ERROR.
`DS2210_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	ds2210_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()
{
  ds2210_canMsg* txMsg;
  UInt32 txMsgData[8];
  ...
  txMsg = ds2210_can_msg_tx_register( txCh,
                  2, 0x1, DS2210_CAN_STD,
                  DS2210_CAN_TIMECOUNT_INFO |
                  DS2210_CAN_MSG_INFO,
                  1, 0.0,
                  DS2210_CAN_TRIGGER_MSG, 0 );
  /* initialize a transmit queue with delay = 0.01 s */
  ds2210_can_msg_txqueue_init (
         txMsg, DS2210_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;
  ds2210_can_msg_send_id_queued(txMsg, 0x12, 1, txMsgData);
  txMsgData[0] = 0x02;
  ds2210_can_msg_send_id_queued(txMsg, 0x13, 1, txMsgData);
  txMsgData[0] = 0x03;
  ds2210_can_msg_send_id_queued(txMsg, 0x14, 1, txMsgData);
...
}
```

Related topics

References

ds2210_can_msg_txqueue_init....................346

ds2210_can_msg_txqueue_level_read

Syntax

```
UInt32 ds2210_can_msg_txqueue_level_read(
      const ds2210_canMsg* canMsg);
```

Include file

`Can2210.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 `ds2210_canMsg` structure.

Return value

Level of TX-queue The number of TX messages in the transmit queue on the CAN slave (0 … 64).

ds2210_can_msg_sleep

Syntax

```
Int32 ds2210_can_msg_sleep(
     const ds2210_canMsg* canMsg);
```

Include file

`Can2210.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 `ds2210_can_msg_wakeup` or `ds2210_can_channel_all_wakeup`.

Parameters

canMsg Specifies the pointer to the `ds2210_canMsg` structure.

Return value

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

Predefined Symbol	Meaning
DS2210_CAN_NO_ERROR	The function was performed without error.
DS2210_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 DS2210_CAN_NO_ERROR.
DS2210_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 385.

Example

```
ds2210_can_msg_sleep(txMsg);
```

Related topics

References

ds2210_can_msg_wakeup

Syntax

```
Int32 ds2210_can_msg_wakeup(
      const ds2210_canMsg* canMsg);
```

Include file

`Can2210.h`

Purpose

To reactivate a message that has been deactivated by calling the `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` function.

Parameters

canMsg Specifies the pointer to the `ds2210_canMsg` structure.

Return value

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

Predefined Symbol	Meaning
DS2210_CAN_NO_ERROR	The function was performed without error.
DS2210_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 DS2210_CAN_NO_ERROR.
DS2210_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 385.

Example

```
ds2210_can_msg_wakeup(txMsg);
```

Related topics

References

ds2210_can_msg_read

Syntax

```
Int32 ds2210_can_msg_rea(
      ds2210_canMsg* canMsg);
```

Include file

`Can2210.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 `ds2210_can_msg_set` on page 337). By default, only one message can be received.

Use the function `ds2210_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 `ds2210_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
`DS2210_CAN_NO_ERROR`	The function was performed without error.
`DS2210_CAN_NO_DATA`	No data was updated.
`DS2210_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 385.

Example

For examples, refer to:
- Example of Handling Transmit and Receive Messages on page 370
- Example of Handling Request and Remote Messages on page 372
- Example of Using Subinterrupts on page 374

Related topics

References

ds2210_can_msg_trigger

Syntax

```
Int32 ds2210_can_msg_trigger(
      const ds2210_canMsg* canMsg,
      const Float32 delay);
```

Include file

`Can2210.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 `ds2210_can_msg_tx_register` or `ds2210_can_msg_rqtx_register` function to call this function.

Parameters

canMsg Specifies the pointer to the `ds2210_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
`DS2210_CAN_NO_ERROR`	The function was performed without error.
`DS2210_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 DS2210_CAN_NO_ERROR.
`DS2210_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 385.

Example

```
ds2210_can_msg_trigger(txMsg, 0.005); /* 5 ms delay */
```

Related topics

References

ds2210_can_msg_clear

Syntax

```
void ds2210_can_msg_clear(
     ds2210_canMsg* canMsg);
```

Include file

`Can2210.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 `ds2210_canMsg` structure.

Return value

None

Function execution times

For information, refer to Function Execution Times on page 385.

Example

```
ds2210_can_msg_clear(rxMsg);
```

Related topics

References

ds2210_can_msg_processed_register

Syntax

```
void ds2210_can_msg_processed_register(
      ds2210_canMsg* canMsg);
```

Include file

`Can2210.h`

Purpose

To register the processed function in the command table.

Use `ds2210_can_msg_processed_read` to read the processed flag and time stamp without registering the message with the inform parameter `DS2210_CAN_TIMECOUNT_INFO`.

Parameters

canMsg Specifies the pointer to the `ds2210_canMsg` structure.

Return value

None

Messages

The following error and warning messages are defined:

ID	Type	Description	Message
101	Error	ds2210_can_msg_processed_registe(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2210_can_msg_processed_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2210_can_msg_processed_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2210_CAN_AUTO_INDEX.
104	Error	ds2210_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	ds2210_can_msg_processed_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2210_can_msg_processed_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` when registering messages or services.

Function execution times

For information, refer to Function Execution Times on page 385.

Example

```
ds2210_can_msg_processed_register(rxMsg);
```

Related topics

References

ds2210_can_msg_processed_request

Syntax

```
Int32 ds2210_can_msg_processed_request(
     const ds2210_canMsg* canMsg);
```

Include file

`Can2210.h`

Purpose

To request the message processed information from the slave DS2210.

Parameters

canMsg Specifies the pointer to the `ds2210_canMsg` structure.

Return value

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

Predefined Symbol	Meaning
DS2210_CAN_NO_ERROR	The function was performed without error.
DS2210_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 DS2210_CAN_NO_ERROR.
DS2210_CAN_NO_DATA	`ds2210_can_msg_processed_request` was called without registering the function with `ds2210_can_msg_processed_register` or an empty canMsg structure was handled.

Function execution times

For information, refer to Function Execution Times on page 385.

Example

```
ds2210_can_msg_processed_request(rxMsg);
```

Related topics

References

ds2210_can_msg_processed_read

Syntax

```
Int32 ds2210_can_msg_processed_read(
      ds2210_canMsg* canMsg,
      double* timestamp,
      UInt32* processed);
```

Include file

`Can2210.h`

Purpose

To read the message processed information from the slave DS2210.

Description

Prior to this, this information must have been requested by the master calling the function `ds2210_can_msg_processed_request` that demands the processed flag and the time stamp from the slave DS2210.

Parameters

canMsg Specifies the pointer to the `ds2210_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
DS2210_CAN_PROCESSED	Message has been sent/received since the last execution call.
DS2210_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
`DS2210_CAN_NO_ERROR`	The function was performed without error.
`DS2210_CAN_NO_DATA`	No data was updated.
`DS2210_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 385.

Related topics

References

CAN Service Functions

Introduction

To get information on errors and status information.

Where to go from here

Information in this section

ds2210_can_service_register

Syntax

```
ds2210_canService* ds2210_can_service_register(
     const ds2210_canChannel* canCh,
     const UInt32 service_type);
```

Include file

`Can2210.h`

Purpose

To register the service function.

Description

Use `ds2210_can_service_read` to read a registered service specified by the `service_type` parameter.

Parameters

canCh Specifies the pointer to the `ds2210_canChannel` structure.

service_type Specifies the service to be installed. For additional information, see the `type` parameter of `ds2210_canService` structure. You can use the bitwise OR operator to combine several services.

Return value

canService This function returns the pointer to the `ds2210_canService` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	ds2210_can_service_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	ds2210_can_service_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	ds2210_can_service_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to DS2210_CAN_AUTO_INDEX.
104	Error	ds2210_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	ds2210_can_service_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	ds2210_can_service_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	ds2210_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 `ds2210_can_msg_sleep` or `ds2210_can_channel_all_sleep` when registering messages or services.
152	Error	ds2210_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 `ds2210_can_channel_start`. Use `ds2210_can_channel_init` or `ds2210_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 376.

```
ds2210_canService* service;
...
  service = ds2210_can_service_register(txCh,
              DS2210_CAN_SERVICE_TX_OK |
              DS2210_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT );
```

Related topics

References

ds2210_can_service_request

Syntax

```
Int32 ds2210_can_service_request(
      const ds2210_canService* service);
```

Include file

`Can2210.h`

Purpose

To request the service information from the slave DS2210. Use `ds2210_can_service_read` to read the registered service.

Description

Use the returned handle from the function `ds2210_can_service_register` on page 361 to call this function.

Parameters

service Specifies the pointer to the `ds2210_canService` structure.

Return value

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

Predefined Symbol	Meaning
DS2210_CAN_NO_ERROR	The function was performed without error.
DS2210_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 DS2210_CAN_NO_ERROR.
DS2210_CAN_NO_DATA	`ds2210_can_service_request` was called without registering the function with `ds2210_can_service_register` or an empty service structure was handled.

Function execution times

For information, refer to Function Execution Times on page 385.

Example

For an example of how to use this function, refer to Example of Using Service Functions on page 376.

Related topics

References

ds2210_can_service_read

Syntax

```
Int32 ds2210_can_service_read(
      ds2210_canService* service);
```

Include file

`Can2210.h`

Purpose

To read the service information from the slave DS2210.

Description

Prior to this, this information must have been requested by the master calling the **`ds2210_can_service_request`** function that asks for the service information from the slave DS2210.

Use the returned handle from the **`ds2210_can_service_register`** function.

Parameters

service Specifies the pointer to the updated **`ds2210_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
`DS2210_CAN_NO_ERROR`	The function was performed without error.
`DS2210_CAN_NO_DATA`	No data was updated.
`DS2210_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 385.

Example

For an example of how to use this function, refer to Example of Using Service Functions on page 376.

```
ds2210_canService* service;
...
service = ds2210_can_service_register(txCh,
            DS2210_CAN_SERVICE_TX_OK |
            DS2210_CAN_SERVICE_TXQUEUE_OVERFLOW_COUNT);
ds2210_can_service_request( service );
ds2210_can_service_read( service );

/* output */
txok = service->tx_ok;
queueoverflow = service->txqueue_overflowcnt_std;
```

Related topics

References

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 `ds2210_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

ds2210_can_subint_handler_install.......367

ds2210_can_subint_handler_install

Syntax

```
ds2210_can_subint_handler_t  ds2210_can_subint_handler_install(
      const UInt32 base,
      const ds2210_can_subint_handler_t handler);
```

Include file

`Can2210.h`

Purpose

To install a subinterrupt handler for all CAN interrupts.

Parameters

base Specifies the PHS-bus base address of the DS2210 board.

handler Specifies the pointer to your callback function.

For information on defining a callback function, refer to Defining a Callback Function on page 366.

Return value

This function returns the following value:

Symbol	Meaning
`ds2210_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 374.

Related topics

Basics

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

ds2210_can_all_data_clear

Syntax

```
void ds2210_can_all_data_clear(const UInt32 base);
```

Include file

`Can2210.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 DS2210 board.

Return value

None

Example

```
ds2210_can_all_data_clear(DS2210_1_BASE)
```

Related topics

References

ds2210_can_msg_clear........356

ds2210_can_error_read

Syntax

```
Int32 ds2210_can_error_read(
      const UInt32 base,
      const Int32 queue);
```

Include file

Can2210.h

Purpose

To read the current error of the slave DS2210 from the dual-port memory.

Parameters

base Specifies the PHS-bus base address of the DS2210 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
DS2210_CAN_NO_ERROR	No error on the slave DS2210.
DS2210_CAN_SLAVE_ALLOC_ERROR	Memory allocation error on the slave DS2210. There are too many functions registered.
DS2210_CAN_SLAVE_BUFFER_OVERFLOW	Not enough memory space between the slave write pointer and the master read pointer.
DS2210_CAN_INIT_ACK	Acknowledge code. This is no error.
DS2210_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 = ds2210_can_error_read(DS2210_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 <Ds2210.h>
#include <Can2210.h>
ds2210_canChannel* txCh;
ds2210_canChannel* rxCh;
ds2210_canMsg* txMsg;
ds2210_canMsg* rxMsg;
UInt32 txMsgData[8] = {1,2,3,4,5,6,7,8};
main()
{
  init(); /* initialize hardware system */
  ds2210_init(DS2210_1_BASE);
  ds2210_can_communication_init(DS2210_1_BASE,
     DS2210_CAN_INT_DISABLE);
  txCh = ds2210_can_channel_init(DS2210_1_BASE, 0,
                 500000,
```

```
                    DS2210_CAN_STD,
                    DS2210_CAN_NO_SUBINT,
                    DS2210_CAN_TERMINATION_ON);
   rxCh = ds2210_can_channel_init(DS2210_1_BASE, 1,
                    500000,
                    DS2210_CAN_STD,
                    DS2210_CAN_NO_SUBINT,
                    DS2210_CAN_TERMINATION_ON);
   txMsg = ds2210_can_msg_tx_register(txCh,
                    2,
                    0x123,
                    DS2210_CAN_STD,
                    DS2210_CAN_TIMECOUNT_INFO,
                    DS2210_CAN_NO_SUBINT,
                    4.0,
                    1.0,
                    DS2210_CAN_TIMEOUT_NORMAL);
   rxMsg = ds2210_can_msg_rx_register(rxCh,
                      3,
                      0x123,
                      DS2210_CAN_STD,
                      DS2210_CAN_DATA_INFO | DS2210_CAN_TIMECOUNT_INFO,
                      DS2210_CAN_NO_SUBINT);
   ds2210_can_msg_write(txMsg, 8, txMsgData);
   ds2210_can_channel_start(rxCh, DS2210_CAN_INT_DISABLE);
   ds2210_can_channel_start(txCh, DS2210_CAN_INT_DISABLE);
   for(;;)
   {
      ds2210_can_msg_read(txMsg);
      if (txMsg->processed == DS2210_CAN_PROCESSED)
      {
         msg_info_printf(MSG_SM_RTLIB, 0,
                      "TX CAN message, time: %f, deltatime: %f ",
                      txMsg->timestamp, txMsg->deltatime);
      }
      ds2210_can_msg_read(rxMsg);
      if (rxMsg->processed == DS2210_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 <Ds2210.h>
#include <Can2210.h>

ds2210_canChannel* rqCh;
ds2210_canChannel* rmCh;
ds2210_canMsg* rqtxMsg;
ds2210_canMsg* rqrxMsg;
ds2210_canMsg* rmMsg;
UInt32 rmMsgData[8] = {1,2,3,4,5,6,7,8};

main()
{
   init(); /* initialize hardware system */

   ds2210_init(DS2210_1_BASE);

   ds2210_can_communication_init(DS2210_1_BASE,
                  DS2210_CAN_INT_DISABLE);

   rqCh = ds2210_can_channel_init(DS2210_1_BASE, 0,
                  500000,
                  DS2210_CAN_STD,
                  DS2210_CAN_NO_SUBINT,
                  DS2210_CAN_TERMINATION_ON);

   rmCh = ds2210_can_channel_init(DS2210_1_BASE, 1,
                  500000,
                  DS2210_CAN_STD,
                  DS2210_CAN_NO_SUBINT,
                  DS2210_CAN_TERMINATION_ON);

   rqtxMsg = ds2210_can_msg_rqtx_register(rqCh,
                        2,
                        0x123,
                        DS2210_CAN_STD,
                        DS2210_CAN_TIMECOUNT_INFO,
                        DS2210_CAN_NO_SUBINT,
                        4.0,
                        2.0,
                        DS2210_CAN_TIMEOUT_NORMAL);

   rqrxMsg = ds2210_can_msg_rqrx_register(rqtxMsg,
                  DS2210_CAN_DATA_INFO | DS2210_CAN_TIMECOUNT_INFO,
                  DS2210_CAN_NO_SUBINT);

   rmMsg = ds2210_can_msg_rm_register(rmCh,
                        3,
                        0x123,
```

```
                        DS2210_CAN_STD,
                        DS2210_CAN_TIMECOUNT_INFO,
                        DS2210_CAN_NO_SUBINT);

   ds2210_can_msg_write(rmMsg, 8, rmMsgData);

   ds2210_can_msg_rqtx_activate(rqtxMsg);

   ds2210_can_channel_start(rqCh, DS2210_CAN_INT_DISABLE);

   ds2210_can_channel_start(rmCh, DS2210_CAN_INT_DISABLE);

   for(;;)
   {

      ds2210_can_msg_read(rqrxMsg);
      ds2210_can_msg_read(rqtxMsg);
      ds2210_can_msg_read(rmMsg);

      if (rqrxMsg->processed == DS2210_CAN_PROCESSED)
      {
         msg_info_printf(MSG_SM_RTLIB, 0,
                  "RQRX CAN message, time: %f,deltatime: %f ",
                  rqrxMsg->timestamp, rqrxMsg->deltatime);
      }

      if (rqtxMsg->processed == DS2210_CAN_PROCESSED)
      {
         msg_info_printf(MSG_SM_RTLIB, 0,
               "RQTX CAN message, time: %f, deltatime: %f ",
               rqtxMsg->timestamp, rqtxMsg->deltatime);
      }

      if (rmMsg->processed == DS2210_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........370
Example of Receiving Different Message IDs........377
Example of Using Service Functions........376
Example of Using Subinterrupts........374

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 <Ds2210.h>
#include <Can2210.h>
#define tx_subint 2
#define rx_subint 3
ds2210_canChannel* txCh;
ds2210_canChannel* rxCh;
ds2210_canMsg* txMsg;
ds2210_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 */
         ds2210_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 */
         ds2210_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 */
   ds2210_init(DS2210_1_BASE);
```

```
    ds2210_can_communication_init(DS2210_1_BASE,
                                  DS2210_CAN_INT_ENABLE);
    ds2210_can_subint_handler_install(DS2210_1_BASE,
                                  can_user_callback);
    txCh = ds2210_can_channel_init (DS2210_1_BASE, 1, 500000,
                                  DS2210_CAN_STD,
                                  DS2210_CAN_NO_SUBINT,
                                  DS2210_CAN_TERMINATION_ON);
    txMsg = ds2210_can_msg_tx_register(txCh,
                                     0,
                                     0x123,
                                     DS2210_CAN_STD,
                                     DS2210_CAN_NO_INFO,
                                     tx_subint,
                                     0.0,
                                     0.0,
                                     DS2210_CAN_TIMEOUT_NORMAL);
    rxCh = ds2210_can_channel_init(DS2210_1_BASE,
                                  0,
                                  500000,
                                  DS2210_CAN_STD,
                                  DS2210_CAN_NO_SUBINT,
                                  DS2210_CAN_TERMINATION_ON);
    rxMsg = ds2210_can_msg_rx_register(rxCh,
                                  0,
                                  0x123,
                                  DS2210_CAN_STD,
                                  DS2210_CAN_DATA_INFO |
                                  DS2210_CAN_TIMECOUNT_INFO,
                                  rx_subint);
    ds2210_can_channel_start(rxCh, DS2210_CAN_INT_DISABLE);
    ds2210_can_channel_start(txCh, DS2210_CAN_INT_DISABLE);
    ds2210_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 DS2210_CAN_SERVICE_TX_OK and DS2210_CAN_SERVICE_RX_OK.

Note

No message is installed on the DS2210 in this example.

```
#include <Brtenv.h>
#include <Ds2210.h>
#include <Can2210.h>
ds2210_canChannel* canCh0;
ds2210_canChannel* canCh1;
ds2210_canService* txokServ;
ds2210_canService* rxokServ;
main()
{
   init();
   ds2210_init(DS2210_1_BASE);
   ds2210_can_communication_init(DS2210_1_BASE,
               DS2210_CAN_INT_DISABLE);
   canCh0 = ds2210_can_channel_init(DS2210_1_BASE, 0,
               500000, DS2210_CAN_STD, DS2210_CAN_NO_SUBINT,
               DS2210_CAN_TERMINATION_ON);
   canCh1 = ds2210_can_channel_init(DS2210_1_BASE, 1,
               500000, DS2210_CAN_STD, DS2210_CAN_NO_SUBINT,
               DS2210_CAN_TERMINATION_ON);
   /* register the tx_ok function which delivers the count */
   /* of the tx-ok counter for CAN channel 0 */
   txokServ = ds2210_can_service_register(canCh0,
               DS2210_CAN_SERVICE_TX_OK);
   /* register the rx_ok function which delivers the count */
   /* of the rx-ok counter for CAN channel 1 */
   rxokServ = ds2210_can_service_register(canCh1,
               DS2210_CAN_SERVICE_RX_OK);
   for(;;)
   {
     /* request the tx-ok counter from the slave DS2210 */
     ds2210_can_service_request(txokServ);
     /* request the rx-ok counter from the slave DS2210 */
     ds2210_can_service_request(rxokServ);
     /* read the tx-ok counter from the slave DS2210 */
     /* the data will be available in txokServ->data0 */
     ds2210_can_service_read(txokServ);
     /* read the rx-ok counter from the slave DS2210 */
     /* the data will be available in rxokServ->data0 */
     ds2210_can_service_read(rxokServ);
     RTLIB_BACKGROUND_SERVICE();
   }
}
```

Related topics

Examples

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 <Ds2210.h>
#include <Can2210.h>
ds2210_canChannel* rxCh;
ds2210_canMsg* canMonitor;
UInt32 data[8];
UInt32 mask = 0x1FFFFF00;
UInt32 queue_size = 64;
main()
{
   init();
   ds2210_init(DS2210_1_BASE);
   ds2210_can_communication_init(DS2210_1_BASE,
                  DS2210_CAN_INT_DISABLE);
   rxCh = ds2210_can_channel_init(DS2210_1_BASE,
                            0,
                            500000,
                            DS2210_CAN_STD,
                            DS2210_CAN_NO_SUBINT,
                            DS2210_CAN_TERMINATION_ON);
   canMonitor = ds2210_can_msg_rx_register (rxCh,
                            1,
                            0x100,
                            DS2210_CAN_STD,
                            DS2210_CAN_TIMECOUNT_INFO |
                                  DS2210_CAN_MSG_INFO,
                            DS2210_CAN_NO_SUBINT);
   ds2210_can_msg_set(canMonitor,
                      DS2210_CAN_MSG_MASK,
                      &mask);
   ds2210_can_msg_set(canMonitor,
                      DS2210_CAN_MSG_QUEUE,
                      &queue_size);
   ds2210_can_channel_start(rxCh, DS2210_CAN_INT_DISABLE);
   for(;;)
   {
      ds2210_can_msg_read(canMonitor);
      if (canMonitor->processed == DS2210_CAN_PROCESSED)
      {
```

```
            msg_info_printf(0,0,"id: %d time: %f",
                canMonitor->identifier, canMonitor->timestamp);
        }
        RTLIB_BACKGROUND_SERVICE();
    }
}
```

Related topics

Examples

Wave Table Generation

Introduction

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

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 Board number of the target DS2210 within the range of 1 … 16

table Target wave table. The following identifier strings are allowed:

Identifier	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 generated wave table can only be used with RTLib functions. It cannot be used with RTI.

data Data array of fixed length 8192

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 DS2210 board number 1.

```
for i = 1:8192
   crank_data(i) = sin(120*4*pi*(i-1)/8192);
end
tbl(1).board = 1;
tbl(1).table = 'crank1';
tbl(1).data = crank_data;
```

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 processor board combined with DS2210 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 DS2210, the allocated memory of the master processor can be used for other purposes. For details, refer to Loading Slave Applications on page 285.

The .slvsect or SlvFwSection section contains one or more data tables. Each table can be identified by a global symbol of the format:

`wav2210<board_no>_<table_id>`

where `board_no` and `table_id` are obtained from the MAT file (refer to Wave Table MAT File Format on page 379).

The generated assembler source file can be used in conjunction with modular systems. 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 `ds2210_crank_table_load` and `ds2210_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: `wav2210<board_no>_<table_id>` where`board_no` and `table_id` are obtained from the corresponding MAT file.

The generated C source file can be used in conjunction with DS1006 or DS1007 systems.

A data table can be loaded by one of the RTLib wave table load functions `ds2210_crank_table_load` and `ds2210_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 following example converts the `wave1.mat` input file to a C source file `wave1.c`. Plain text information about the wave tables being converted is displayed on the screen.

```
matconv wave1.mat /c /v
```

Related topics

References

MAT2C2210.M

Syntax

```
mat2c2210(
   C_fileName,
   MAT_fileNames,
   boardNo,
   tableIdentifier)
```

Description

MAT2C2210.M combines one or more wave table MAT files into a single MAT file (see Wave Table MAT File Format on page 379). It invokes MATCONV.EXE to generate the corresponding C source code. The source MAT files must contain a single array of length 8192 each.

MAT2C2210.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 DS2210 board number

tableIdentifier String cell array to identify the generated wave table, for example, `{'crank1','crank2','camA1'}`

Example

```
mat2c2210('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<xxxx>` 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
CPU clock	2.6 GHz / 3.0 GHz
Bus clock	133 MHz

Measured Execution Times

Execution times

Execution times are are availiable for the following RTLib units:

- Initialization and setup on page 387
- ADC on page 387
- DAC on page 387
- Bit I/O on page 387
- D/R converter on page 388
- Timing mode on page 388
- PWM signal generation on page 388
- PWM measurement on page 388
- Overall APU functions on page 388
- Engine position phase accumulation on page 389
- Crankshaft sensor signal generation on page 389
- Camshaft sensor signal generation on page 389
- Spark event capture on page 389
- Injection pulse position and fuel amount measurement on page 390
- Overall DSP on page 390
- Knock sensor simulation on page 390
- Wheel speed sensor simulation on page 390
- Slave DSP memory access on page 391
- CAN Access on page 391

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 has been measured for initialization and setup functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_init`	41.70 ms	41706 µs
`ds2210_mode_set`	0.69 µs	0.70 µs
`ds2210_digin_threshold_set`	0.10 µs	0.080 µs
`ds2210_digout_mode_set`	0.65 µs	0.74 µs
`ds2210_digwform_mode_set`	0.64 µs	0.72 µs
`ds2210_apu_transformer_mode_set`	0.67 µs	1.13 µs

ADC

The following execution times has been measured for ADC functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_adc_start`	0.039 µs	0.034 µs
`ds2210_adc_block_in_fast` (16 channels)	5.43 µs	5.4 µs
`ds2210_adc_single_in` (channel 16)	11.71 µs	12.77 µs
`ds2210_adc_block_init` (16 channels)	0.35 µs	0.28 µs
`ds2210_adc_block_start`	0.19 µs	0.031 µs
`ds2210_adc_block_in` (16 channels)	21.11 µs	22.08 µs

DAC

The following execution times has been measured for DAC functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_dac_out`	0.065 µs	0.133 µs

Bit I/O

The following execution times has been measured for Bit I/O functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_bit_io_in`	0.62 µs	0.68 µs
`ds2210_bit_io_out`	0.036 µs	0.029 µs
`ds2210_bit_io_set`	0.65 µs	0.63 µs
`ds2210_bit_io_clear`	0.63 µs	0.63 µs

D/R converter

The following execution times has been measured for D/R converter functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_resistance_out`	0.067 µs	0.096 µs

Timing mode

The following execution times has been measured for timing mode functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_timing_out_mode_set`	1.51 µs	0.93 µs
`ds2210_timing_in_mode_set`	1.57 µs	0.74 µs

PWM signal generation

The following execution times has been measured for PWM signal generating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_timing_out_mode_set`	1.55 µs	0.93 µs
`ds2210_pwm_out`	0.37 µs	0.139 µs

PWM measurement

The following execution times has been measured for PWM measuring functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_timing_in_mode_set`	1.57 µs	0.74 µs
`ds2210_pwm_in`	0.74 µs	0.68 µs

Overall APU functions

The following execution times has been measured for overall APU functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_apu_position_write`	0.83 µs	0.81 µs
`ds2210_apu_position_read`	0.67 µs	0.68 µs
`ds2210_apu_velocity_write`	0.049 µs	0.037 µs
`ds2210_int_position_set`	11212 + (1.95 · count) µs	11284.06 + (1.93 · count) µs

Engine position phase accumulation

The following execution times has been measured for engine position phase accumulation functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_apu_start`	2.81 µs	2.69 µs
`ds2210_apu_stop`	0.65 µs	0.62 µs

Crankshaft sensor signal generation

The following execution times has been measured for crankshaft sensor signal generating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_crank_table_load`	5.95 ms	5.97 ms
`ds2210_crank_table_select`	0.66 µs	0.63 µs
`ds2210_crank_output_ampl_set`	0.26 µs	0.28 µs

Camshaft sensor signal generation

The following execution times has been measured for camshaft sensor signal generating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_cam_table_load`	5.95 ms	5.97 ms
`ds2210_cam_table_select`	0.74 µs	0.65 µs
`ds2210_cam_output_ampl_set`	0.19 µs	0.33 µs
`ds2210_cam_phase_write`	0.67 µs	0.81 µs
`ds2210_cam_phase_read`	0.66 µs	0.65 µs

Spark event capture

The following execution times has been measured for spark event capturing functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_event_window_set`	11.20 ms	11.32 ms
`ds2210_ign_capture_mode_set`	3.55 µs	3.56 µs
`ds2210_aux1_capture_mode_set`	3.32 µs	3.28 µs
`ds2210_aux2_capture_mode_set`	3.39 µs	3.35 µs
`ds2210_ignition_capture_read` (8 pulses/720°, 10 ms sampling time, 0 … 30000 rpm)	0.7 … 3.42 µs	0.69 … 3.26 µs

Injection pulse position and fuel amount measurement

The following execution times has been measured for the functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_event_window_set`	11.38 ms	11.32 ms
`ds2210_inj_capture_mode_set`	2.02 µs	1.78 µs
`ds2210_injection_capture_read` (8 pulses/720°, 10 ms sampling time, 0 … 30000 rpm)	0.6 … 3.47 µs	0.59 … 25.04 µs

Overall DSP

The following execution times has been measured for overall DSP functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_slave_dsp_signal_enable`	0.98 µs	1.17 µs
`ds2210_slave_dsp_channel_enable`	1.61 µs	1.56 µs
`ds2210_slave_dsp_interrupt_set`	0.098 µs	0.03 µs
`ds2210_slave_dsp_speedchk`	3.13 µs	3.19 µs
`ds2210_slave_dsp_error`	1.63 µs	1.57 µs
`ds2210_slave_dsp_appl_load`		
knock signal application	1.01 ms	1.031 ms
wheel signal application	0.76 ms	0.765 ms

Knock sensor simulation

The following execution times has been measured for knock sensor simulating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_slave_dsp_knock_init`	18.72 µs	17.74 µs
`ds2210_slave_dsp_knock_update`	2.05 µs	2.0 µs
`ds2210_slave_dsp_knock_noise`	1.83 µs	1.83 µs

Wheel speed sensor simulation

The following execution times has been measured for wheel speed sensor simulating functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_slave_dsp_wheel_init`	6.53 µs	6.57 µs
`ds2210_slave_dsp_wheel_update`	1.26 µs	1.3 µs

Slave DSP memory access

The following execution times has been measured for slave DSP memory accessing functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
`ds2210_slave_dsp_read`	1.53 µs	1.44 µs
`ds2210_slave_dsp_write`	0.94 µs	0.85 µs
`ds2210_slave_dsp_block_read`	1.55 + c · 0.46 µs	1.085 + c · 0.524 µs
`ds2210_slave_dsp_block_write`	0.93 + c · 0.038 µs	0.715 + c · 0.134 µs
`ds2210_slave_dsp_sem_req`	0.94 µs	0.92 µs
`ds2210_slave_dsp_sem_rel`	0.047 µs	0.032 µs
`ds2210_slave_dsp_read_direct`	0.63 µs	0.61 µs
`ds2210_slave_dsp_write_direct`	0.044 µs	0.032 µs
`ds2210_slave_dsp_block_read_di`	0.73 + c · 0.48 µs	0.095 + c · 0.524 µs
`ds2210_slave_dsp_block_write_di`	-0.103 + c · 0.141 µs	-0.10 + c · 0.133 µs

CAN Access

The following execution times has been measured for CAN access functions:

Function	Execution Time	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
CAN channel handling		
`ds2210_can_channel_all_sleep`	1.33 µs	1.55 µs
`ds2210_can_channel_all_wakeup`	1.45 µs	1.52 µs
`ds2210_can_channel_BOff_go`	1.45 µs	1.55 µs
`ds2210_can_channel_BOff_return`	1.48 µs	1.58 µs
CAN message access		
`ds2210_can_msg_write`	2.45 µs	2.40 µs
`ds2210_can_msg_send`	2.57 µs	2.70 µs
`ds2210_can_msg_sleep`	2.06 µs	2.08 µs
`ds2210_can_msg_wakeup`	1.61 µs	1.70 µs
`ds2210_can_msg_read`	7.1 µs	0.781 µs
`ds2210_can_msg_trigger`	2.5 µs	1.782 µs
`ds2210_can_msg_clear` (for each message to be cleared)	0.035 µs	0.04 µs
`ds2210_can_msg_processed_request`	1.3 µs	0.022 µs
`ds2210_can_msg_processed_read`	0.9 µs	0.022 µs
CAN service		
`ds2210_can_service_request`	2.5 µs	1.34 µs
`ds2210_can_service_read`	8.2 µs	4.05 µs