

DS1006 Processor Board

RTLib Reference

Release 2021-A – May 2021

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

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

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

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

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

Contents

About This Document	13
Processor Core Modules	15
Data Types and Definitions	17
Elementary Data Types	17
Initialization	18
ds1006_init	18
ds1006_slv_boot_finished	20
ds1006_cache_load	20
Host Service	22
host_service	23
master_cmd_server	25
RTLIB_BACKGROUND_SERVICE	26
rtlib_background_hook	27
rtlib_background_hook_process	28
Time Interval Measurement	30
Data Types for Time Measurement	32
Example of Using Time Measurement Functions	32
ds1006_tic_continue	33
ds1006_tic_count	34
ds1006_tic_delay	35
ds1006_tic_diff	35
ds1006_tic_elapsed	36
ds1006_tic_halt	37
ds1006_tic_read	38
ds1006_tic_start	39
ds1006_tic_total_read	40
ds1006_timebase_fltread	40
ds1006_timebase_low_read	41
ds1006_timebase_read	42
RTLIB_TIC_CONTINUE	42
RTLIB_TIC_COUNT	43
RTLIB_TIC_DELAY	44
RTLIB_TIC_DIFF	45
RTLIB_TIC_ELAPSED	46

RTLIB_TIC_HALT.....	47
RTLIB_TIC_READ.....	48
RTLIB_TIC_READ_TOTAL.....	49
RTLIB_TIC_START.....	49
Time-Stamping.....	51
General Information on Time-Stamping.....	51
Basic Principles of Time-Stamping.....	51
Principles of an Absolute Time in Single-Processor and Multiprocessor Systems.....	52
Implementation of an Absolute Time in Single- and Multiprocessor Systems.....	53
Data Types and Global Variables for Time-Stamping.....	55
Data Types Used for Time-Stamping.....	55
Global Variables Used for Time-Stamping.....	55
Time-Stamping Functions.....	56
ts_init.....	57
ts_reset.....	58
ts_time_read.....	59
ts_timestamp_read.....	59
ts_timestamp_compare.....	60
ts_timestamp_interval.....	61
ts_time_offset.....	62
ts_timestamp_offset.....	63
ts_time_calculate.....	63
ts_timestamp_calculate.....	64
Timer A.....	66
Example of Using Timer A Functions.....	66
RTLIB_SRT_PERIOD.....	67
ds1006_timerA_period_set.....	68
ds1006_timerA_period_reload.....	69
ds1006_timerA_read.....	69
ds1006_timerA_start.....	70
ds1006_timerA_stop.....	70
Timer B.....	72
Example of Using Timer B Functions.....	72
ds1006_timerB_init.....	73
ds1006_timerB_compare_set.....	74
ds1006_timerB_compare_set_periodically.....	75
ds1006_timerB_read.....	76

ds1006_timerB_start.....	76
ds1006_timerB_stop.....	77
Timer D.....	78
Example of Using Timer D functions.....	78
ds1006_timerD_period_set.....	79
ds1006_timerD_period_reload.....	80
ds1006_timerD_read.....	80
ds1006_timerD_start.....	81
ds1006_timerD_stop.....	82
Timer Interrupt Control.....	83
ds1006_begin_isr_timerA.....	85
ds1006_begin_isr_timerB.....	85
ds1006_begin_isr_timerD.....	86
ds1006_end_isr_timerA.....	87
ds1006_end_isr_timerB.....	87
ds1006_end_isr_timerD.....	88
ds1006_start_isr_timerA.....	89
ds1006_start_isr_timerB.....	90
ds1006_start_isr_timerD.....	91
RTLIB_SRT_ISR_BEGIN.....	92
RTLIB_SRT_ISR_END.....	93
RTLIB_SRT_START.....	93
Interrupt Handling.....	95
ds1006_disable_hardware_int.....	97
ds1006_disable_hardware_int_bm.....	98
ds1006_enable_hardware_int.....	99
ds1006_enable_hardware_int_bm.....	101
ds1006_get_interrupt_flag.....	102
ds1006_get_interrupt_flag_bm.....	104
ds1006_get_interrupt_status.....	105
ds1006_get_interrupt_vector.....	106
ds1006_reset_interrupt_flag.....	107
ds1006_reset_interrupt_flag_bm.....	108
ds1006_set_interrupt_status.....	109
ds1006_set_interrupt_vector.....	110
RTLIB_INT_DISABLE.....	112
RTLIB_INT_ENABLE.....	113
RTLIB_INT_RESTORE.....	114
RTLIB_INT_SAVE_AND_DISABLE.....	114
RTLIB_SRT_DISABLE.....	115
RTLIB_SRT_ENABLE.....	116

Subinterrupt Handling.....	117
Basic Principles of Subinterrupt Handling.....	118
Example of Using a Subinterrupt Sender.....	118
Example of Using a Subinterrupt Handler.....	119
Example of Using a Subinterrupt Receiver.....	120
Data Types for Subinterrupt Handling.....	122
dssint_define_int_sender.....	123
dssint_define_int_sender_1.....	125
dssint_define_int_receiver.....	127
dssint_define_int_receiver_1.....	129
dssint_subint_disable.....	131
dssint_subint_enable.....	132
dssint_interrupt.....	133
dssint_decode.....	133
dssint_acknowledge.....	134
dssint_subint_reset.....	135
Exception Handling.....	137
Basics on Exception Handling.....	137
Information Handling.....	139
ds1006_info_board_version_get.....	139
ds1006_info_memory_get.....	140
ds1006_info_clocks_get.....	140
ds1006_info_clocks_khz_get.....	141
ds1006_info_cpu_temperature_get.....	141
Version and Config Section Management.....	143
Basic Principles of VCM.....	145
Data Types for VCM.....	148
vcm_init.....	149
vcm_module_register.....	149
vcm_cfg_malloc.....	151
vcm_memory_ptr_set.....	152
vcm_memory_ptr_get.....	152
vcm_module_find.....	153
vcm_module_status_set.....	154
vcm_module_status_get.....	155
vcm_version_get.....	156
vcm_version_compare.....	157
vcm_module_version_print.....	158
vcm_version_print.....	159

Message Handling.....	160
Basic Principles of Message Handling.....	161
Data Types and Symbols for Message Handling.....	162
msg_error_set.....	164
msg_warning_set.....	165
msg_info_set.....	166
msg_set.....	166
msg_error_printf.....	168
msg_warning_printf.....	170
msg_info_printf.....	171
msg_printf.....	172
msg_default_dialog_set.....	174
msg_mode_set.....	175
msg_reset.....	176
msg_last_error_number.....	177
msg_last_error_submodule.....	178
msg_error_clear.....	179
msg_error_hook_set.....	180
msg_init.....	181
Watchdog Handling.....	183
ds1006_wd_set.....	183
ds1006_wd_init.....	184
ds1006_wd_stop.....	185
ds1006_wd_strobe.....	186
ds1006_wd_system_was_reset.....	186
Serial Interface Communication.....	188
Basic Principles of Serial Communication.....	188
Software FIFO Buffer.....	188
Trigger Levels.....	189
How to Handle Subinterrupts in Serial Communication.....	190
Example of a Serial Interface Communication.....	191
Data Types for Serial Communication.....	192
dsser_ISR.....	193
dsser_LSR.....	195
dsser_MSR.....	196
dsser_subint_handler_t.....	197
dsserChannel.....	198
Generic Serial Interface Communication Functions.....	200
dsser_init.....	201
dsser_free.....	202

ds1006_dsser_config.....	203
ds1006_dsser_transmit.....	206
ds1006_dsser_receive.....	208
ds1006_dsser_receive_term.....	209
ds1006_dsser_fifo_reset.....	211
ds1006_dsser_enable.....	212
ds1006_dsser_disable.....	212
ds1006_dsser_error_read.....	213
ds1006_dsser_transmit_fifo_level.....	214
ds1006_dsser_receive_fifo_level.....	215
ds1006_dsser_status_read.....	216
ds1006_dsser_handle_get.....	217
ds1006_dsser_set.....	218
ds1006_dsser_subint_handler_inst.....	219
ds1006_dsser_subint_enable.....	220
ds1006_dsser_subint_disable.....	221
ds1006_dsser_word2bytes.....	222
ds1006_dsser_bytes2word.....	224
Serial Interface Communication (Low Level).....	226
ds1006_serial_init.....	227
ds1006_serial_config.....	228
ds1006_serial_free.....	230
ds1006_serial_transmit.....	230
ds1006_serial_receive.....	231
ds1006_serial_register_write.....	232
ds1006_serial_register_read.....	233
Special Processor Functions.....	234
RTLIB_FORCE_IN_ORDER.....	234
RTLIB_SYNC.....	235
Conversion Functions.....	236
ds1006_conv32_ieee_to_ti.....	236
ds1006_conv32_ti_to_ieee.....	237
ds1006_conv_float_to_int32.....	237
RTLIB_CONV_FLOAT32_FROM_IEEE32.....	238
RTLIB_CONV_FLOAT32_FROM_TI32.....	239
RTLIB_CONV_FLOAT32_TO_IEEE32.....	239
RTLIB_CONV_FLOAT_TO_SATURATED_INT32.....	240
RTLIB_CONV_FLOAT32_TO_TI32.....	240
Standard Macros.....	242
init().....	244

RTLIB_EXIT.....	245
RTLIB_GET_SERIAL_NUMBER().....	245
RTLIB_MALLOC_PROT.....	246
RTLIB_CALLOC_PROT.....	246
RTLIB_REALLOC_PROT.....	247
RTLIB_FREE_PROT.....	247
Function Execution Times.....	249
Information on the Test Environment.....	249
Measured Execution Times.....	250

I/O Modules 253

I/O Boards.....	254
A/D Conversion.....	254
D/A Conversion.....	255
Automotive Signal Generation and Measurement.....	255
Bit I/O.....	255
Timing I/O.....	256
Interface Boards.....	256
Special I/O.....	257
Integration of FPGA Applications.....	257
PHS-Bus Handling.....	258
get_peripheral_addr.....	260
PHS_BOARD_BASE_GET.....	261
phs_board_type_get.....	261
phs_board_type_from_slot_get.....	262
PHS_REGISTER_READ.....	263
PHS_REGISTER_WRITE.....	264
PHS_REGISTER_PTR.....	265
PHS_IO_ERROR_STATE.....	265
PHS_IO_ERROR_SET.....	266
PHS_SYNCIN_TRIGGER.....	267
PHS_SYNCOUT_TRIGGER.....	267
PHS_SYNC_TRIGGER.....	268
PHS_SYNC_TIMER_SET.....	268
PHS-Bus Interrupt Handling.....	270
General Information on PHS-Bus Interrupts.....	270
Basics of PHS-Bus Interrupts.....	271
Management of the Extended Interrupt System.....	272
Board Identification and PHS-Bus Interrupt Line Programming.....	272
PHS-Bus Interrupt Processing.....	274

Interrupt Priorities.....	275
How to Program PHS-Bus Interrupts.....	276
Example of PHS-Bus Interrupt Handling.....	277
Description of the Demo Application.....	278
PHS-Bus Interrupt Application.....	279
Example with Customer I/O Boards.....	280
PHS-Bus Interrupt Functions.....	282
install_phs_int_vector.....	282
deinstall_phs_int_vector.....	284
declare_phs_int_line.....	285
alloc_phs_int_line.....	286
free_phs_int_line.....	287
enable_phs_int.....	289
disable_phs_int.....	290
init_phs_int.....	291
deinit_phs_int.....	292
set_phs_int_mask.....	293
Troubleshooting for PHS-Bus Interrupt Handling.....	295
Error Codes of PHS-Bus Interrupt Functions.....	295
 Multiprocessing Modules	 297
Initialization.....	298
Data Types for MP System Initialization.....	298
Example of an MP System Initialization.....	299
ds1006_mp_init.....	301
ds1006_mp_synchronize.....	303
ds1006_mp_route_mat.....	305
ds1006_mp_route_syncin.....	305
ds1006_mp_route_syncout.....	306
ds1006_mp_optional_cpu_reduce.....	307
Global Sample Rate Timer in an MP System.....	309
Example of Initializing a Global Sample Rate Timer.....	309
ds1006_mp_global_srt_init.....	310
ds1006_mp_start_isr_global_srt.....	311
ds1006_mp_begin_isr_global_srt.....	313
ds1006_mp_end_isr_global_srt.....	314
Interprocessor Interrupts.....	315
ds1006_ipi_init.....	316
ds1006_ipi_configure.....	316

ds1006_ipi_interrupt.....	317
ds1006_ipi_acknowledge.....	318
ds1006_ipi_enable.....	319
ds1006_ipi_disable.....	320
ds1006_ipi_enable_bm.....	321
ds1006_ipi_disable_bm.....	321
ds1006_ipi_mask_set.....	322
ds1006_ipi_mask_get.....	323
ds1006_ipi_sint_max_snd_set.....	323
ds1006_ipi_sint_max_rcv_set.....	324
ds1006_ipi_install_handler.....	325
Gigalink Communication.....	327
ds1006_gl_init.....	329
ds1006_gl_initialized.....	329
ds1006_gl_synchronized.....	330
ds1006_gl_scanner_init.....	331
ds1006_gl_scan.....	332
ds1006_gl_background_scan.....	333
ds1006_gl_write32.....	333
ds1006_gl_write32_and_switch.....	335
ds1006_gl_write64.....	336
ds1006_gl_block_write.....	338
ds1006_gl_write_buffer_switch.....	339
ds1006_gl_read32.....	341
ds1006_gl_read64.....	342
ds1006_gl_block_read.....	343
ds1006_gl_read_buffer_switch.....	345
ds1006_gl_read_buffer_is_updated.....	346
ds1006_gl_module_present.....	347
ds1006_gl_opto_signal_detect.....	347

Host Programs 349

Host Settings.....	350
Compiler and C Run-Time Libraries.....	350
Environment Variables and Paths.....	350
Folder Structure.....	351
DS1006 Real-Time Library.....	351
File Extensions.....	352
Compiling, Linking and Downloading an Application.....	354
Down1006.exe.....	355

DsBuildApplication.mk.....	358
DsBuildLoad.mk.....	359
DsBuildTemplate.mk.....	359
Ds1006.lk.....	360
Integrating C++ Code.....	361
Debugging an Application.....	362
i686-elf-objdump.....	362
 Index.....	 365

About This Document

Content









This document provides feature-oriented access to the reference information you need to implement the functions provided by the DS1006 Processor Board.

Note

There are examples for some features included in this documentation. You will find the relevant files after the installation of your dSPACE software in the folder <RCP_HIL_InstallationPath>\Demos\DS1006\. Use the dSPACE experiment software to load and start the application on the DS1006.

Symbols

dSPACE user documentation uses the following symbols:

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

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

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

Processor Core Modules

Where to go from here

Information in this section

Data Types and Definitions.....	17
Initialization.....	18
Host Service.....	22
Time Interval Measurement.....	30
Time-Stamping.....	51
Timer A.....	66
Timer B.....	72
Timer D.....	78
Timer Interrupt Control.....	83
Interrupt Handling.....	95
Subinterrupt Handling.....	117
Exception Handling.....	137
Information Handling.....	139
Version and Config Section Management.....	143
Message Handling.....	160
Watchdog Handling.....	183
Serial Interface Communication.....	188
Serial Interface Communication (Low Level).....	226
Special Processor Functions.....	234
Conversion Functions.....	236

Standard Macros.....	242
Function Execution Times.....	249

Data Types and Definitions

Elementary Data Types

Data types

The `dstypes.h` file defines the overall processor-independent data types as follows:

```
typedef char                      Int8;
typedef unsigned char             UInt8;
typedef short                     Int16;
typedef unsigned short            UInt16;
typedef int                       Int32;
typedef unsigned int              UInt32;
typedef struct {UInt32 low; Int32 high;} Int64;
typedef struct {UInt32 low; UInt32 high;} UInt64;
typedef long long                 Long64;
typedef unsigned long long        ULong64;
typedef float                     Float32;
typedef double                    Float64;
typedef double                    dsfloat;
typedef Int8 *                    Int8Ptr;
typedef UInt8 *                   UInt8Ptr;
typedef Int16 *                   Int16Ptr;
typedef UInt16 *                  UInt16Ptr;
typedef Int32 *                   Int32Ptr;
typedef UInt32 *                  UInt32Ptr;
typedef Int64 *                   Int64Ptr;
typedef UInt64 *                  UInt64Ptr;
typedef Long64 *                  Long64Ptr;
typedef ULong64 *                 ULong64Ptr;
typedef Float32 *                  Float32Ptr;
typedef Float64 *                  Float64Ptr;
```

Include file

`dstypes.h`

Initialization

Where to go from here

Information in this section

ds1006_init	18
ds1006_slv_boot_finished	20
ds1006_cache_load	20

ds1006_init

Syntax

```
ds1006_init(void)
```

or

```
init(void)
```

Include file

brtenv.h

Purpose

To initialize all required hard- and software modules for the DS1006.

Note

The initialization function **ds1006_init** must be executed at the beginning of each application. It can only be invoked once. Further calls to this function are ignored.

When you are using RTI, this function is called automatically in the simulation engine. Hence, you do not need to call **ds1006_init** in S-functions. If you need to initialize single components that are not initialized by **ds1006_init** (see below), use the specific initialization functions that are described at the beginning of the function references.

Description

The global variable **ds1006_debug** controls the output of debug messages. Its value is defined by the compiler option `-D[definition]`:

Compiler Option	Meaning
<code>-DDEBUG_INIT</code>	Debug messages are output for initialization functions only.

Compiler Option	Meaning
-DDEBUG_POLL	Debug messages are output for poll functions only.
-DDEBUG_INIT -DDEBUG_POLL	Debug messages are output for initialization and poll functions.

ds1006_init carries out the following initialization steps in this order:

1. Global variables for time measurement purposes
2. Memory management functions
3. Low-level PCI Interrupt handling
4. X86 exception handling
5. RTLib1006 is registered in the VCM module
6. The message module for passing error, warning and info messages to the dSPACE experiment software
7. Registration of some modules at the VCM module
8. The boot firmware is checked. If it is not compatible, the application is terminated and a message is issued
9. The time stamping module in single mode
10. The host service module for data transfer from and to the dSPACE experiment software
11. The exit function **ds1006_exit** is hooked in the program termination routine
12. The PHSBUS base address table is built
13. I/O boards which are connected to the PHS bus
14. The serial interface module
15. The Gigalink module if it exists
16. The interprocessor interrupt module if the Gigalink module is present
17. The status of RTLib1006 is set to initialized

This is the basic configuration of the DS1006 Processor Board.

Like mentioned above, **ds1006_init** installs the function **ds1006_exit** with the function **atexit** from the compiler library. Calling the function **exit()** while a program is running activates **ds1006_exit**. This function disables the interrupts and calls the services for the dSPACE experiment software. So you have the possibility to terminate your application (such as I/O settings).

The DS1006 registers are initialized with the following values:

Register	Value	Meaning
Interrupt Mask Register	0xFFFFFFFF	Disables all interrupts.
Interrupt Register	0xFFE00000	Resets all interrupts.
Timer A Period Register	0xFFFFFFFF	Sets Timer A period to the maximum period.
Timer B Compare Register	0xFFFFFFFF	Sets the Timer B compare register to the maximum value.
Timer D Period Register	0xFFFFFFFF	Sets Timer D period to the maximum period.
Timer Control Register	0	Sets Timer A, Timer B and Timer D to hold mode, and sets Timer B input frequency to 1/4 bus clock.

Related topics**References**

[Standard Macros.....](#) 242

ds1006_slv_boot_finished

Syntax

```
ds1006_slv_boot_finished(void)
```

or

```
void RTLIB_SLAVE_LOAD_ACKNOWLEDGE()
```

Include file

```
init1006.h
```

Purpose

To provide downward compatibility with the DS1005.

Description

Since slave processors are not loaded via the global memory, this function has no effect on the DS1006.

ds1006_cache_load

Syntax

```
void* ds1006_cache_load(
    void* base,
    UInt32 bytes)
```

Include file

```
init1006.h
```

Purpose

To preload a memory block into the cache.

Description

When a portion of code is executed the first time the processor transfers it from local to cache memory. The same applies for data structures, when they are referenced the first time. This causes a significantly longer execution time of the first sample step of a simulation model.

To prevent this, code and data can be preloaded into cache memory by calling `ds1006_cache_load`. This function loads a block of memory into the cache.

The memory is read in amounts of 4-byte words. If the specified area does not end on a 4-byte border, the memory is read up to the last 4-byte border. The number of bytes is implicitly limited to the size of the level 2 cache.

For further information on the cache, refer to [Memory \(DS1006 Features !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)](#)).

Parameters

base Specifies the base address of the memory block to be preloaded.

bytes Specifies the number of bytes to be preloaded. If the size of the piece of data is unknown, you can use the predefined symbol `DS1006_CACHE_LOAD_MAX` to preload as much as possible.

Return value

This function returns a pointer to the first memory location behind the preloaded area.

Example

```
int d_array[1000];
...
void main()
{
    init(); /* or ds1006_init(); */
    /* preload the data array into the cache */
    ds1006_cache_load((void *) d_array, sizeof(d_array));
    ...
}
```

Related topics

Basics

[Memory \(DS1006 Features !\[\]\(0fb13ad0bfa3d86868cdd3883e5665b3_img.jpg\)](#))

Host Service

Introduction

This section describes the functions for exchanging data between the host PC and the real-time hardware.

A host program like ControlDesk reads and writes data from and to the real-time hardware. The host service call `host_service` is the actual point where the data is sampled. The master command server `master_cmd_server` transfers the collected data to the host PC.

One host service call must always be located in the model background, others can be anywhere in the application. The master command server is always located in the model background.

Note

To ensure that both calls are in the background of your application, you should use the macro `RTLIB_BACKGROUND_SERVICE`. It also starts automatically all board-specific functions, that must run in the background loop.

Example

This is the source code for a background loop in an application program:

```
while(1)
{
    RTLIB_BACKGROUND_SERVICE();
}
```

Where to go from here

Information in this section

host_service	23
To make the signals of your application accessible to a host program.	
master_cmd_server	25
To execute a command that is passed from the host PC to the real-time hardware.	
RTLIB_BACKGROUND_SERVICE	26
To execute all relevant background functions with one call.	
rtlib_background_hook	27
To register a specified hook function.	
rtlib_background_hook_process	28
To start the execution of registered hook functions.	

host_service

Syntax

```
host_service(
    UInt16 trace_service_no,
    ts_timestamp_ptr_type ts)
```

Include file

hostsvc.h

Purpose

To service the data exchange between the real-time hardware and host computer.

Description

The host service call performs all variable reads that are requested by host applications like ControlDesk. Hence, when the **host_service** call is missing in your application, the host application issues a relevant error message. The same message is issued when the **host_service** or **master_cmd_server** call is not executed due to an application crash.

To ensure that both the **host_service** and the **master_cmd_server** call are present in the model background loop, the RTLib background macro **RTLIB_BACKGROUND_SERVICE** can be used.

The **host_service** function supports 32 services with different purposes. Service #0 is used for data exchange in the model background. For example, ControlDesk uses this service to refresh the values of instruments like displays or sliders. Hence, every time the model passes its background, display instruments get new data.

Services #1 to #31 are used in the model foreground (e.g., an interrupt service routine). For example, ControlDesk uses these services to acquire data for plotter instruments. For this reason each plotter has a corresponding Capture Settings Window, in which the host service from which the data is received can be selected.

Services #28 to #31 are reserved in RTI generated applications for monitoring features.

Note

If the host wants to read a variable from an interrupt-driven task that has not been started yet, the host application displays the error message "The service function is not called by the real-time application." To avoid this, you can call the corresponding **host_service** function with parameter **ts = 0** within the main application to guarantee the availability of the service.

Parameters

trace_service_no Specifies the trace service number. The values are:

Value	Meaning
0	Background service (host service #0)
1	Base rate service (host service #1)
2 ... 27	Sampling rate service 1 ... 26 (host service #2 ... #27)
28 ... 31	Reserved in RTI generated applications for monitoring features (host service #28 ... #31)

ts Specifies the pointer to a time stamp structure that represents the time of the associated data (for further information, refer to [Time-Stamping](#) on page 51). For example, ControlDesk uses this accurate time measurement for generating the time axis and for setting the samples exactly in a plotter instrument.

Note

The background service does not use the time stamp support. It is always called as `host_service(0,0)`.

An application has to contain one background service. Up to 31 foreground services can be executed. Normally, each host service belongs to one interrupt service routine with its own time stamp structure.

Example

The example shows how to program a foreground host service with time stamp support.

```
...
void isr_func()
{
    ts_timestamp_type ts;
    /* sample step calculation */
    ...
    ts_timestamp_read(&ts);
    host_service(1,&ts);
}

void main(void)
{
    init();
    ...
    /* to make the service #1 available before the task is called */
    host_service(1,0);
    ...

    while(1)
    {
        RTLIB_BACKGROUND_SERVICE();
    }
    ...
}
```


Execution time

The execution times of host service calls #1 to #31 mainly depend on the number of variables which are captured in experiment software by data acquisition instruments. They are affected neither by displays and other instruments nor by the number of variables in the trace file (*.trc).

You can calculate the execution time by using the following formula:

$n_{double} > 0$ or $n_{others} > 0$	$t_{host_service} \approx 900\text{ ns} + n_{double} \cdot 96\text{ ns} + n_{others} \cdot 48\text{ ns}$
$n_{double} = 0$ or $n_{others} = 0$	$t_{host_service} = 30\text{ ns}$

where n_{double} is the number of variables of type **double** and n_{others} is the number of variables of another type (**int**, **float**, **char**,...).

This formula is valid under the following conditions:

- A data capture is in progress. In the time between the completion of a capture and the recognition of the next trigger event, the host services have a constant execution time of 1200 ns. This time is independent of the number of variables in the plot.
- A plain trigger is used for the capture in ControlDesk. Using a pre- or post trigger adds a constant of 80 ns.
- A DS1006 processor board with a CPU clock of 2.2 GHz and a bus clock of 133 MHz is used. If your DS1006 processor board has other clock values, the execution time differs.
- Cache effects are not considered. The values may vary due to cache effects (depending on the memory locations of the captured variables).

Related topics

Basics	
Basic Principles of Time-Stamping.....	51
References	
master_cmd_server.....	25
RTLIB_BACKGROUND_SERVICE.....	26

master_cmd_server

Syntax

```
master_cmd_server()
```

Include file

dscmd.h

Purpose	To call the master command server.				
Description	<p>The master command server executes commands that are passed from the host PC to the real-time hardware. An example of a command is the request for a buffer with plot data sampled by the <code>host_service</code> call. The master command server must be present in each simulation model. Otherwise a relevant error message is issued by the dSPACE experiment software.</p> <p>To ensure that both the <code>host_service</code> and the <code>master_cmd_server</code> call are present in the model background loop, the RTLib background macro <code>RTLIB_BACKGROUND_SERVICE</code> can be used.</p>				
Related topics	<p>References</p> <table border="0"> <tr> <td>host_service.....</td><td>23</td></tr> <tr> <td>RTLIB_BACKGROUND_SERVICE.....</td><td>26</td></tr> </table>	host_service	23	RTLIB_BACKGROUND_SERVICE	26
host_service	23				
RTLIB_BACKGROUND_SERVICE	26				

RTLIB_BACKGROUND_SERVICE

Syntax	<code>RTLIB_BACKGROUND_SERVICE()</code>
Include file	<code>dsstd.h</code>
Purpose	To call the essential functions in the model background loop.
Description	<p>This macro calls the following functions:</p> <ul style="list-style-type: none"> ▪ <code>host_service</code> The background loop is called <code>host_service(0,0)</code>. So, it does not use the time stamp support. ▪ <code>master_cmd_server</code> ▪ <code>elog_service</code> ▪ <code>rtlib_background_hook_process</code> ▪ <code>ds1006_g1_background_scan</code> ▪ <code>ds1006_info_cpu_temperature_get</code> <p>This macro executes all the required background services, for example, for the host communication. It must be continuously called in the background of your application, for example, within a <code>for</code> or a <code>while</code> construct. To constantly maintain its functionality, it must be called at least once per second.</p>

Related topics

References

ds1006_gl_background_scan.....	333
ds1006_info_cpu_temperature_get.....	141
host_service.....	23
master_cmd_server.....	25
rtlib_background_hook_process.....	28

rtlib_background_hook

Syntax

```
int rtlib_background_hook(rtlib_bg_fcn_t *fcnptr)
```

Include file

dsstd.h

Purpose

To register a function to be executed in the background loop.

Description

You can register several functions by calling `rtlib_background_hook` subsequently. The `RTLIB_BACKGROUND_SERVICE` macro starts the execution whereas the last registered function will be executed first.

Note

- The specified function must be of type `rtlib_bg_fcn_t`, which defines a function with no arguments and no return value.
- The background loop waits for the execution of the specified hook functions. Ensure that the hook functions do not completely block the background service.

Parameters

fcnptr Specifies the pointer to the background function.

Return value

This function returns the following values:

Return Value	Meaning
0	The background function has been registered successfully.
1	An error occurred while registering the background function.

Example

This example shows how to implement a simple hook function within the background loop. The variable `bg_count` counts the number of executed background loops.

```
int bg_count=0;
void bg_fcn()
{
    bg_count++;
}
void main(void)
{
    int result;
    init();
    /* setup foreground, for e.g. a timer isr */
    ...
    result = rtlib_background_hook(bg_fcn);
    ...
    /* background loop */
    while(1)
    {
        /* call the background functions */
        RTLIB_BACKGROUND_SERVICE();
    }
}
```

Related topics**References**

rtlib_background_hook_process	28
RTLIB_BACKGROUND_SERVICE	26

rtlib_background_hook_process

Syntax

```
void rtlib_background_hook_process(void)
```

Include file

`dsstd.h`

Purpose

To execute all registered background hook functions.

Description

The background functions which have been registered with the `rtlib_background_hook` function will be executed, beginning with the last registered function.

Note

- The background loop waits on the execution of the specified hook functions. Be sure that the hook functions do not block the background service totally.
- A call to this function is already included in the background service macro `RTLIB_BACKGROUND_SERVICE`. If you call it anyway, the hook function will be executed twice.

Return value	None
---------------------	------

Related topics

References

rtlib_background_hook.....	27
RTLIB_BACKGROUND_SERVICE.....	26

Time Interval Measurement

Introduction

Functions for measuring time intervals are used for profiling application code (execution time measurement) or for implementing time delays. The time is derived from the built-in Time-Stamp Counter, which has a resolution of CPU clock.

Tip

Here you find the descriptions of platform-specific functions and generic `RTLIB_TIC_XXX` macros. It is recommended to use the generic macros.

Where to go from here

Information in this section

Data Types for Time Measurement.....	32
Example of Using Time Measurement Functions.....	32
ds1006_tic_continue.....	33
To resume time measurement after it was paused.	
ds1006_tic_count.....	34
To read the current counter value of the time base.	
ds1006_tic_delay.....	35
To perform the specified time delay.	
ds1006_tic_diff.....	35
To calculate the difference between two time base counter values.	
ds1006_tic_elapsed.....	36
To calculate the difference between a previous time base counter value and the current time base value.	
ds1006_tic_halt.....	37
To pause time measurement.	
ds1006_tic_read.....	38
To read the time period since time measurement was started, minus the breaks.	
ds1006_tic_start.....	39
To start a time measurement.	
ds1006_tic_total_read.....	40
To read the complete time period since the time measurement was started, including all breaks.	
ds1006_timebase_fltread.....	40
To read the Lower and Upper Timebase Registers (TBRL and TBRU) and convert the result to a 64-bit float value (seconds).	
ds1006_timebase_low_read.....	41
To read the lower 32 bits of the time base register.	
ds1006_timebase_read.....	42
To read the 64 bits of the time base register.	
RTLIB_TIC_CONTINUE.....	42
To resume time measurement after it was paused.	
RTLIB_TIC_COUNT.....	43
To read the current counter value of the time base.	
RTLIB_TIC_DELAY.....	44
To perform the specified time delay.	
RTLIB_TIC_DIFF.....	45
To calculate the difference between two time base counter values.	

RTLIB_TIC_ELAPSED.....	46
To calculate the difference between a previous time base counter value and the current time base value.	
RTLIB_TIC_HALT.....	47
To pause time measurement.	
RTLIB_TIC_READ.....	48
To read the time period since time measurement was started minus the breaks made.	
RTLIB_TIC_READ_TOTAL.....	49
To read the complete time period since the time measurement was started, including all breaks made.	
RTLIB_TIC_START.....	49
To start a time measurement.	

Data Types for Time Measurement

Introduction

There is one specific data type used by the **ds1006_tic_count**, **ds1006_tic_elapsed**, **ds1006_tic_diff** functions and their related macros.

rtlib_tic_t

This data type is used to specify the time base counter values. It is defined as ULong64 data type.

Example of Using Time Measurement Functions

Example

The following example shows the source code to measure the execution time of certain actions. Three actions are specified in the program, but only action 1 and action 3 are measured using the board-specific function names:

```
ds1006_tic_start(); /* starts time measurement */
...
time = ds1006_tic_read();
... action 1 ...
ds1006_tic_halt(); /* start of the break */
... action 2 ...
ds1006_tic_continue(); /* end of the break */
... action 3 ...
time = ds1006_tic_read() - time;
/* second read and calculation of the action 1 and 3 period */
```


To measure the execution time of action 1 and action 3 using the standard macros:

```
RTLIB_TIC_START(); /* starts time measurement */
...
time = RTLIB_TIC_READ();
... action 1 ...
RTLIB_TIC_HALT(); /* start of the break */
... action 2 ...
RTLIB_TIC_CONTINUE(); /* end of the break */
... action 3 ...
time = RTLIB_TIC_READ() - time;
/* second read and calculation of the action 1 and 3 period */
```

ds1006_tic_continue

Syntax ds1006_tic_continue()

Include file tic1006.h

Purpose To resume time measurement after it was paused by ds1006_tic_halt.

Description This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 56.

Return value None

Related topics

Examples

Example of Using Time Measurement Functions..... 32

References

ds1006_tic_halt..... 37

RTLIB_TIC_CONTINUE..... 42

ds1006_tic_count

Syntax	<code>rtlib_tic_t ds1006_tic_count(void)</code>
Include file	<code>tic1006.h</code>
Purpose	To read the current counter value of the time base.
Description	Use <code>ds1006_tic_count</code> in conjunction with <code>ds1006_tic_elapsed</code> or <code>ds1006_tic_diff</code> to perform execution time measurement in recursive functions.
Parameters	None
Return value	This function returns the current counter value of the time base as <code>rtlib_tic_t</code> data type.

Example The following example shows how to calculate the time difference between two time base counter values.

```
void main(void)
{
    rtlib_tic_t timer_count1 = 0,
    rtlib_tic_t timer_count2 = 0;
    dsfloat exec_time = 0;

    init();

    timer_count1 = ds1006_tic_count();
    ...
    timer_count2 = ds1006_tic_count();
    exec_time = ds1006_tic_diff(timer_count1, timer_count2);
    ...
}
```

Related topics

References

ds1006_tic_diff.....	35
ds1006_tic_elapsed.....	36

ds1006_tic_delay

Syntax	<code>ds1006_tic_delay(Float64 duration)</code>						
Include file	<code>tic1006.h</code>						
Purpose	To perform the specified time delay.						
Parameters	duration Specifies the time delay in seconds. If you specify a duration that exceeds the maximum range of the timer, the function never stops.						
Return value	None						
Related topics	References <table border="0"> <tr> <td>ds1006_tic_continue.....</td><td>33</td></tr> <tr> <td>ds1006_tic_start.....</td><td>39</td></tr> <tr> <td>RTLIB_TIC_DELAY.....</td><td>44</td></tr> </table>	ds1006_tic_continue.....	33	ds1006_tic_start.....	39	RTLIB_TIC_DELAY.....	44
ds1006_tic_continue.....	33						
ds1006_tic_start.....	39						
RTLIB_TIC_DELAY.....	44						

ds1006_tic_diff

Syntax	<pre>dsfloat ds1006_tic_diff(rtlib_tic_t tmr_cnt1, rtlib_tic_t tmr_cnt2)</pre>
Include file	<code>tic1006.h</code>
Purpose	To calculate the difference between two time base counter values.
Description	Use <code>ds1006_tic_diff</code> in conjunction with <code>ds1006_tic_count</code> or <code>ds1006_tic_elapsed</code> to perform execution time measurement in recursive functions.

Parameters	tmr_cnt1	Specifies the first time base counter value.
	tmr_cnt2	Specifies the second time base counter value.

Return value	This function returns the time difference in seconds.
---------------------	---

Example	The following example shows how to calculate the time difference between two time base counter values.
----------------	--

```
void main(void)
{
    rtlib_tic_t timer_count1 = 0, timer_count2 = 0;
    dsfloat exec_time = 0;

    init();

    timer_count1 = ds1006_tic_count();
    ...
    timer_count2 = ds1006_tic_count();
    exec_time = ds1006_tic_diff(timer_count1, timer_count2);
    ...
}
```

Related topics**References**

ds1006_tic_count	34
ds1006_tic_elapsed	36
RTLIB_TIC_DIFF	45

ds1006_tic_elapsed

Syntax	<code>dsfloat ds1006_tic_elapsed(rtlib_tic_t tmr_cnt)</code>
---------------	--

Include file	<code>tic1006.h</code>
---------------------	------------------------

Purpose	To calculate the difference between a previous time base counter value specified by <code>tmr_cnt</code> and the current time base value in seconds.
----------------	--

Description	Use <code>ds1006_tic_elapsed</code> in conjunction with <code>ds1006_tic_count</code> or <code>ds1006_tic_diff</code> to perform execution time measurement in recursive functions.				
Parameters	tmr_cnt Specifies the previous counter value of the time base.				
Return value	This function returns the elapsed time in seconds.				
Example	<p>The following example shows how to calculate the time difference between a previous time base counter value and the current time base value.</p> <pre> void main(void) { rtlib_tic_t timer_count; dsfloat exec_time = 0; init(); timer_count = ds1006_tic_count(); ... exec_time = ds1006_tic_elapsed(timer_count); ... } </pre>				
Related topics	References <table> <tr> <td>ds1006_tic_count.....</td><td>34</td></tr> <tr> <td>ds1006_tic_diff.....</td><td>35</td></tr> </table>	ds1006_tic_count	34	ds1006_tic_diff	35
ds1006_tic_count	34				
ds1006_tic_diff	35				

ds1006_tic_halt

Syntax	<code>ds1006_tic_halt()</code>
Include file	<code>tic1006.h</code>
Purpose	To pause time measurement.
Description	The break lasts until measurement is resumed by <code>ds1006_tic_continue</code> .

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 56.

Return value None

Related topics

Examples

[Example of Using Time Measurement Functions.....](#) 32

References

[ds1006_tic_continue.....](#) 33
[RTLIB_TIC_HALT.....](#) 47

ds1006_tic_read

Syntax `Float64 ds1006_tic_read()`

Include file `tic1006.h`

Purpose To read the time period since time measurement was started by **ds1006_tic_start**, minus the breaks made from **ds1006_tic_halt** to **ds1006_tic_continue**.

Description Use **ds1006_tic_total_read** to read the complete time period including the breaks.

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 56.

Return value This function returns the time duration in seconds.

Related topics**Examples**

[Example of Using Time Measurement Functions.....](#) 32

References

[ds1006_tic_continue.....](#) 33
[ds1006_tic_halt.....](#) 37
[ds1006_tic_start.....](#) 39
[ds1006_tic_total_read.....](#) 40
[RTLIB_TIC_READ.....](#) 48

ds1006_tic_start

Syntax

```
ds1006_tic_start()
```

Include file

tic1006.h

Purpose

To start a time measurement.

Description

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 56.

Return value

None

Related topics**Examples**

[Example of Using Time Measurement Functions.....](#) 32

References

[RTLIB_TIC_START.....](#) 49

ds1006_tic_total_read

Syntax	Float64 ds1006_tic_total_read()
Include file	tic1006.h
Purpose	To read the complete time period since the time measurement was started by ds1006_tic_start , including all breaks made from ds1006_tic_halt to ds1006_tic_continue .
Description	Use ds1006_tic_read to read the time period minus the breaks made. This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to Time-Stamping Functions on page 56.
Return value	This function returns the time duration in seconds.
Related topics	References <div> ds1006_tic_continue..... 33 ds1006_tic_halt..... 37 ds1006_tic_read..... 38 RTLIB_TIC_READ_TOTAL..... 49 </div>

ds1006_timebase_fltread

Syntax	Float64 ds1006_timebase_fltread(void)
Include file	tmr1006.h
Purpose	To read the Lower and Upper Timebase Registers (TBRL and TBRU) and convert the result to a 64-bit float value (seconds).
Return value	This function returns the current value of the time base registers in seconds.

Related topics**References**

ds1006_timebase_low_read	41
ds1006_timebase_read	42

ds1006_timebase_low_read

Syntax

```
UInt32 ds1006_timebase_low_read(void)
```

Include file

```
tmr1006.h
```

Purpose

To read the lower 32 bits of the time base register.

Description

Use `ds1006_timebase_read` to read the complete time base register.

Note

This function is provided for downward compatibility and should not be used on DS1006

Due to the higher resolution of the DS1006 time base the lower time base register (TBRL) will wrap to zero after less than two seconds ($2^{32} / 2.2\text{GHz} < 2$ seconds). This causes problems if the TBRL is used for interval measurements or delays.

Return value

This function returns the lower 32 bits of the current time base register.

Related topics**References**

ds1006_timebase_fltread	40
ds1006_timebase_read	42

ds1006_timebase_read

Syntax

```
Int64 ds1006_timebase_read(void)
```

Include file

```
tmr1006.h
```

Purpose

To read the 64 bits of the time base register.

Description

For compatibility reasons, a structure of type `Int64` is returned, which consists of an `Int32` (high word) and an `UInt32` (low word) (refer to [Elementary Data Types](#) on page 17). Use `ds1006_timebase_low_read` to read only the lower 32 bits of the Time-Stamp Counter.

The functions `ds1006_tic_start` and `ds1006_tic_count` return the same information in an `ULong64` type.

Return value

This function returns the current value of the time base register.

Related topics

References

```
ds1006_timebase_fltread..... 40
ds1006_timebase_low_read..... 41
```

RTLIB_TIC_CONTINUE

Syntax

```
RTLIB_TIC_CONTINUE()
```

Include file

```
dsstd.h
```

Purpose

To resume time measurement after it was paused by `RTLIB_TIC_HALT`.

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 56.

Return value

None

Related topics**Examples**

[Example of Using Time Measurement Functions.....](#) 32

References

[ds1006_tic_continue.....](#) 33
[RTLIB_TIC_HALT.....](#) 47

RTLIB_TIC_COUNT

Syntax

```
rtlib_tic_t RTLIB_TIC_COUNT(void)
```

Include file

dsstd.h

Purpose

To read the current counter value of the time base.

Description

Use `RTLIB_TIC_COUNT()` in conjunction with `RTLIB_TIC_ELAPSED` or `RTLIB_TIC_DIFF` to perform execution time measurement in recursive functions.

Parameters

None

Return value

This function returns the current counter value of the time base as `rtlib_tic_t` data type.

Example

The following example shows how to calculate the time difference between two time base counter values.

```
void main(void)
{
    rtlib_tic_t timer_count1 = 0,
    rtlib_tic_t timer_count2 = 0;
    dsfloat exec_time = 0;

    init();

    timer_count1 = RTLIB_TIC_COUNT();
    ...
    timer_count2 = RTLIB_TIC_COUNT();
    exec_time = RTLIB_TIC_DIFF(timer_count1, timer_count2);
}
```

Related topics**References**

ds1006_tic_count	34
RTLIB_TIC_DIFF	45
RTLIB_TIC_ELAPSED	46

RTLIB_TIC_DELAY

Syntax

```
RTLIB_TIC_DELAY(Float64 duration)
```

Include file

dsstd.h

Purpose

To perform the specified time delay.

Parameters

duration Specifies the time delay in seconds. If you specify a duration that exceeds the maximum range of the timer, the function never stops.

Return value

None

Related topics

References

ds1006_tic_delay	35
RTLIB_TIC_CONTINUE	42
RTLIB_TIC_START	49

RTLIB_TIC_DIFF

Syntax

```
dsfloat RTLIB_TIC_DIFF(  
    rtlib_tic_t tmr_cnt1,  
    rtlib_tic_t tmr_cnt2)
```

Include file

dsstd.h

Purpose

To calculate the difference between two time base counter values.

Description

Use **RTLIB_TIC_DIFF** in conjunction with **RTLIB_TIC_COUNT** or **RTLIB_TIC_ELAPSED** to perform execution time measurement in recursive functions.

Parameters

- tmr_cnt1** Specifies the first time base counter value.
- tmr_cnt2** Specifies the second time base counter value.

Return value

This function returns the time difference in seconds.

Example

The following example shows how to calculate the time difference between two time base counter values.

```
void main(void)
{
    rtlib_tic_t timer_count1 = 0, timer_count2 = 0;
    dsfloat exec_time = 0;

    init();

    timer_count1 = RTLIB_TIC_COUNT();
    ...
    timer_count2 = RTLIB_TIC_COUNT();
    exec_time = RTLIB_TIC_DIFF(timer_count1, timer_count2);
    ...
}
```

Related topics**References**

ds1006_tic_diff	35
RTLIB_TIC_COUNT	43
RTLIB_TIC_ELAPSED	46

RTLIB_TIC_ELAPSED

Syntax

```
dsfloat RTLIB_TIC_ELAPSED(rtlib_tic_t tmr_cnt)
```

Include file

dsstd.h

Purpose

To calculate the difference between a previous time base counter value specified by **tmr_cnt** and the current time base value in seconds using a generic macro.

Description

Use **RTLIB_TIC_ELAPSED** in conjunction with **RTLIB_TIC_COUNT** or **RTLIB_TIC_DIFF** to perform execution time measurement in recursive functions.

Parameters

tmr_cnt Specifies the previous counter value of the time base.

Return value

This function returns the elapsed time in seconds.

Example

The following example shows how to calculate the time difference between a previous time base counter value and the current time base value.

```
void main(void)
{
    rtlib_tic_t timer_count;
    dsfloat exec_time = 0;

    init();

    timer_count = RTLIB_TIC_COUNT();
    ...
    exec_time = RTLIB_TIC_ELAPSED(timer_count);
    ...
}
```

Related topics**References**

[ds1006_tic_elapsed](#)..... 36

RTLIB_TIC_HALT

Syntax

```
RTLIB_TIC_HALT()
```

Include file

dsstd.h

Purpose

To pause time measurement.

Description

The break lasts until measurement is resumed by **RTLIB_TIC_CONTINUE**.

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 56.

Return value

None

Related topics**Examples**

[Example of Using Time Measurement Functions.....](#) 32

References

[ds1006_tic_halt.....](#) 37
[RTLIB_TIC_CONTINUE.....](#) 42

RTLIB_TIC_READ

Syntax

```
RTLIB_TIC_READ()
```

Include file

```
dsstd.h
```

Purpose

To read the time period since time measurement was started by **RTLIB_TIC_START**, minus the breaks made from **RTLIB_TIC_HALT** to **RTLIB_TIC_CONTINUE**.

Description

Use **RTLIB_TIC_READ_TOTAL** to read the complete time period including the breaks made.

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 56.

Return value

This function returns the time duration in seconds.

Related topics**Examples**

[Example of Using Time Measurement Functions.....](#) 32

References

[ds1006_tic_read.....](#) 38
[RTLIB_TIC_CONTINUE.....](#) 42
[RTLIB_TIC_HALT.....](#) 47
[RTLIB_TIC_START.....](#) 49

RTLIB_TIC_READ_TOTAL

Syntax	<code>RTLIB_TIC_READ_TOTAL()</code>										
Include file	<code>dsstd.h</code>										
Purpose	To read the complete time period since the time measurement was started by RTLIB_TIC_START , including all breaks made from RTLIB_TIC_HALT to RTLIB_TIC_CONTINUE .										
Description	<p>Use RTLIB_TIC_READ to read the time period minus the breaks made.</p> <p>This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to Time-Stamping Functions on page 56.</p>										
Return value	This function returns the time duration in seconds.										
Related topics	References <table border="0"> <tr> <td>ds1006_tic_total_read.....</td><td>40</td></tr> <tr> <td>RTLIB_TIC_CONTINUE.....</td><td>42</td></tr> <tr> <td>RTLIB_TIC_HALT.....</td><td>47</td></tr> <tr> <td>RTLIB_TIC_READ.....</td><td>48</td></tr> <tr> <td>RTLIB_TIC_START.....</td><td>49</td></tr> </table>	ds1006_tic_total_read	40	RTLIB_TIC_CONTINUE	42	RTLIB_TIC_HALT	47	RTLIB_TIC_READ	48	RTLIB_TIC_START	49
ds1006_tic_total_read	40										
RTLIB_TIC_CONTINUE	42										
RTLIB_TIC_HALT	47										
RTLIB_TIC_READ	48										
RTLIB_TIC_START	49										

RTLIB_TIC_START

Syntax	<code>RTLIB_TIC_START()</code>
Include file	<code>dsstd.h</code>
Purpose	To start a time measurement.
Description	<p>This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to Time-Stamping Functions on page 56.</p>

Return value	None
Related topics	<div>Examples</div> <div>Example of Using Time Measurement Functions..... 32</div> <div>References</div> <div>ds1006_tic_start..... 39</div>

Time-Stamping

Introduction The time-stamping module is used to take absolute time stamps from a highly accurate, absolute time base.

Where to go from here

Information in this section

General Information on Time-Stamping.....	51
Data Types and Global Variables for Time-Stamping.....	55
Time-Stamping Functions.....	56

General Information on Time-Stamping

Introduction Gives you information on basic principles and implementation details of the time-stamping feature.

Where to go from here

Information in this section

Basic Principles of Time-Stamping.....	51
Principles of an Absolute Time in Single-Processor and Multiprocessor Systems.....	52
Implementation of an Absolute Time in Single- and Multiprocessor Systems.....	53

Basic Principles of Time-Stamping

Introduction The Time-Stamping module is used to take absolute time stamps from a highly accurate, absolute time base. The time base fulfills the following requirements:

Time stamp accuracy The exact resolution depends on the mode of the Time-Stamping module. See [Modes of the Time-Stamping module](#) on page 53 for the exact resolution.

Time stamp range The time base has a range of 64 bit. Combined with a resolution down to CPU clock, this is enough to measure highly accurate absolute times up to several years.

Synchronization (in multiprocessor systems) The time bases of different processors in a multiprocessor system clocks are synchronized to avoid effects of inaccurate or drifting quartz frequencies. Synchronization is essential when you relate time stamps from different processor boards (see [Multiprocessor systems](#) on page 54).

Principles of an Absolute Time in Single-Processor and Multiprocessor Systems

Introduction

The Time-Stamping module is the fundamental time base for real-time simulations. It provides sufficiently accurate samples of the independent variable time. Therefore, if data and events have been recorded together with the associated time stamps, it is possible to reconstruct their temporal order.

Note

The same information applies to multiprocessor applications running on a multicore system.

Synchronization of local clocks

Each processor has its own local time base (local clock). Due to manufacturing tolerances, which lead to clock drifts, the local clocks in a multiprocessor system have to be synchronized periodically. To keep the communication effort low, synchronization does not take place at every tick of the local clocks (microtick), but at a selected tick of a timing master. This selected tick is called macrotick.

In single-processor systems, no synchronization is required. The macrotick lasts a full cycle of the microtick and the microtick covers the full extent of the time base. The data type of the timestamp structure contains one counter for the microtick and one for the macrotick. Because of this, the timestamp structure meets the requirements of both single- and multiprocessor systems.

When a macrotick occurs, the number of microticks is set to zero and the number of macroticks is increased by 1 at each processor. Starting from this point in time, the absolute time t_{abs} is calculated as follows:

$$t_{abs} = MAT \cdot P_{MAT} + MIT \cdot P_{MIT}$$

In this equation, "MAT" denotes the number of macroticks, which is incremented in the entire system, whereas "MIT" is the number of microticks. " P_{MAT} " is the macrotick period, which is a system-wide constant. " P_{MIT} " denotes the microtick period that can differ from clock to clock. For information on the implementation and accuracy of this approach, refer to the following section

Implementation of an Absolute Time in Single- and Multiprocessor Systems on page 53.

Synchronization by interrupts

The macrotick is dispatched from the timing master to all other processors in the system. This is done by an interrupt line of the DS1006 Gigalinks. The dispatching mechanism and the macrotick event mechanism are implemented in the hardware and are therefore fully transparent for the applications.

Modes of the Time-Stamping module

The Time-Stamping module can operate in three different modes:

single mode This is the mode for single-processor systems (single-core applications).

The microtick (the tick of the local clock) is derived from the processor-internal Time-Stamp Counter, which is driven by the bus clock of the DS1006, scaled by 4. For example, at a DS1006 with 2.2 GHz CPU clock, the resolution of the Microtick Counter is below 0.5 ns. In the dSPACE experiment software, you can find information on the bus clock in the Properties dialog of the DS1006. In the dialog, select the DS1006 Properties page.

multi-master mode This is the mode of the timing master in a multiprocessor system (multicore application).

The microtick is generated by the synchronous time base unit (STBU), and driven by the bus clock of the DS1006, scaled by 2. For example, at a DS1006 with 133 MHz bus clock, the resolution of the Microtick Counter is 15 ns. In the dSPACE experiment software, you can find information on the bus clock in the Properties dialog of the DS1006. In the dialog, select the DS1006 Properties page.

When the Microtick Counter reaches the macrotick period, a system-wide macrotick is generated.

multi-slave mode This is the mode of all other processors in a multiprocessor system (multicore application).

As on the master processor, the microtick is generated by the STBU. Processors in Slave mode receive their macrotick from the timing master.

Implementation of an Absolute Time in Single- and Multiprocessor Systems

Single-processor systems

In single-processor systems the absolute time is identical to the microtick clock time. Microticks are generated locally by a hardware timer.

The following table displays the timer used for this purpose and some of its basic characteristics:

Board	Timer Source	Frequency	Resolution	Condition
DS1006	CPU clock	$f_{\text{CPUCLK}} / 4$	<0.5 ns	$f_{\text{CPUCLK}} = 2.2 \text{ GHz}$

The microtick timers are 64-bit wide and read-only.

Due to the out-of-order execution of the AMD Opteron™ processor, the accuracy cannot be determined easily.

Multiprocessor systems

The time stamping master generates macroticks and dispatches them to other processor boards by a periodic interrupt (macrotick interrupt). The time stamping master can be any processor board. The multiprocessor initialization function `ds1006_mp_init` automatically assigns the time stamping master (this is the CPU with ID 0, see `ds1006_mp_init` on page 301).

The macrotick frequency influences the time accuracy and range:

- When the frequency of the macrotick is too low, the accuracy of the time decreases.
- When the frequency is too high, the time range of the time decreases.

Implementation for the DS1006

The DS1006 has a Synchronous Time Base Unit (STBU), which fulfills all the requirements of an absolute time base in a multiprocessor system. Normally, the STBU of one processor board in a multiprocessor system is programmed as timing master, all others as timing slaves. The master STBU generates the system wide Macrotock, when its Microtick Counter reaches the Macrotock period. Upon a Macrotock all STBUs reset their Microtick Counter and increment the Macrotock Counter. Macroticks are dispatched by the DS1006 Gigalinks.

The following table displays the timer used for this purpose and some of its basic characteristics:

Board	Timer Source	Frequency	Resolution	Condition
DS1006	STBU	$f_{\text{BCLK}}/2$	15 ns	$f_{\text{BCLK}}=133 \text{ MHz}$

Accuracy of the synchronization

The accuracy of the absolute time, t_{abs} , depends on the macrotick and microtick periods, P_{MAT} and P_{MIT} . The latter is determined by the deviation of a crystal oscillator from its nominal period, ϵ_{max} . The oscillators have a nominal deviation of $\epsilon_{\text{max}} = \pm 100 \text{ ppm}$. When the clocks are synchronized every 10 ms the accuracy is approximately to $\pm 1 \mu\text{s}$.

Range

The range of the absolute time also depends on the macrotick period. As the Macrotock counter is a 32-bit counter, a macrotick period of 10 ms leads to a time range of 497 days, when using 100 μs the time range is 4.97 days.

Due to the dependency between the macrotick period and the time range and accuracy we recommend a macrotick period of 1 to 10 ms.

Data Types and Global Variables for Time-Stamping

Introduction

Gives you basic information on data types and global variables used for time-stamping.

Where to go from here

Information in this section

Data Types Used for Time-Stamping.....	55
Global Variables Used for Time-Stamping.....	55

Data Types Used for Time-Stamping

Data types

The following data types are defined for time-stamping:

Data Type	Syntax
ts_timestamp_type	<pre>typedef struct { UInt32 mat; /* 32 bit macrotick counter value */ UInt32 mit; /* 32 bit microtick counter value */ }ts_timestamp_type;</pre>
ts_timestamp_ptr_type	<pre>typedef ts_timestamp_type * ts_timestamp_ptr_type</pre>

Global Variables Used for Time-Stamping

Global variables

The following global variables are defined for time-stamping:

Type	Syntax	Description
dsts_mat_period	dsfloat dsts_mat_period;	Time for one macrotick period (in seconds).
dsts_mit_period	dsfloat dsts_mit_period;	Time for one microtick period (in seconds). This time depends on the frequency of the Time Base Counter.
dsts_mode	int dsts_mode;	<p>Mode of the time-stamping software module. The following symbols are predefined:</p> <ul style="list-style-type: none"> ▪ TS_MODE_SINGLE Used for single-processor systems ▪ TS_MODE_MULTI_MASTER

Type	Syntax	Description
dsts_mit_per_mat	UInt32 dsts_mit_per_mat;	<p>Used for the master board in a multiprocessor system</p> <ul style="list-style-type: none"> ▪ TS_MODE_MULTI_SLAVE <p>Used for slave boards in a multiprocessor system</p> <p>For further information, refer to Modes of the Time-Stamping module on page 53.</p> <p>Nominal number of microticks per macrotick.</p>

Time-Stamping Functions

Introduction

Gives you information on the C functions available for the time-stamping feature.

Where to go from here

Information in this section

ts_init	57
To initialize the Time-Stamping module.	
ts_reset	58
To set the absolute time to 0.	
ts_time_read	59
To read the absolute time in seconds.	
ts_timestamp_read	59
To read the absolute time and return it as time stamp structure.	
ts_timestamp_compare	60
To compare two time stamps.	
ts_timestamp_interval	61
To return the interval between two time stamps.	
ts_time_offset	62
To calculate the difference between two time stamps and add this difference to the reference time.	
ts_timestamp_offset	63
To calculate the difference between two time stamps and add this difference to the reference time stamp.	
ts_time_calculate	63
To convert a time stamp structure to a time value in seconds.	
ts_timestamp_calculate	64
To convert a time value in seconds to a time stamp structure.	

ts_init

Syntax

```
int ts_init(
    int mode,
    float mat_period)
```

Include file

dsts.h

Purpose

To initialize the Time-Stamping module and the hardware, and to reset the Microtick and the Macro-tick Counter.

Description

- The function **ts_init** is called automatically by the board initialization function **init()**, which sets the Time-Stamping module to mode **TS_MODE_SINGLE**.
- The function **ts_init** is also called automatically by the multiprocessor initialization, which sets the Time-Stamping module to the **TS_MODE_MULTI_MASTER** mode at the processor board with ID 0 and to the **TS_MODE_MULTI_SLAVE** mode at all other boards.
- When the Time-Stamping module is initialized with **TS_MODE_MULTI_MASTER** or **TS_MODE_MULTI_SLAVE**, the Synchronous Time Base Unit (STBU) is stopped. It can be started explicitly by calling **ts_reset**.

Parameters

mode Specifies the mode of the Time-Stamping module; the following symbols are predefined:

Predefined Symbol	Meaning
TS_MODE_SINGLE	single mode
TS_MODE_MULTI_MASTER	multi-master mode
TS_MODE_MULTI_SLAVE	multi-slave mode

mat_period Specifies the time in seconds of one macro-tick period. In single-processor systems, this argument is ignored (can be 0.0).

Return value

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

Predefined Symbol	Meaning
TS_INIT_DONE	Module initialization successful
TS_INIT_FAILED	Module initialization failed

Related topics

Basics	
Basic Principles of Time-Stamping.....	51
References	
ts_reset.....	58

ts_reset

Syntax	<code>void ts_reset()</code>
Include file	<code>dsts.h</code>
Purpose	<p>To reset the Time-Stamping module to the absolute time 0.</p> <div> <div>Note</div> <p>The information, that the Time-stamping module performs a reset, is not transferred between processor boards. Hence, if one processor performs a reset, all others must perform it too. To synchronize all processors in a multiprocessor system at a specific location within the code the function <code>ds1006_mp_synchronize</code> can be used before calling <code>ts_reset</code> (see <code>ds1006_mp_synchronize</code> on page 303).</p> </div>
Return value	None

Related topics

Basics	
Basic Principles of Time-Stamping.....	51
References	
ts_init.....	57

ts_time_read

Syntax	<code>double ts_time_read()</code>
Include file	<code>dsts.h</code>
Purpose	To read the absolute time in seconds.
Return value	This function returns the absolute time in seconds since the initialization <code>ts_init</code> or the last reset <code>ts_reset</code> .
Related topics	<div>Basics</div> <div> Basic Principles of Time-Stamping..... 51 </div> <div>References</div> <div> ts_timestamp_read..... 59 </div>

ts_timestamp_read

Syntax	<code>void ts_timestamp_read(ts_timestamp_ptr_type ts)</code>
Include file	<code>dsts.h</code>
Purpose	To read the absolute time and return it as time stamp structure.
Result	The absolute time is read and is written to the time stamp structure <code>ts</code> points to.
Parameters	<code>ts</code> Specifies the pointer to a time stamp structure for the read value.
Return value	None

Related topics**Basics**

[Basic Principles of Time-Stamping..... 51](#)

References

[ts_time_read..... 59](#)

ts_timestamp_compare

Syntax

```
int ts_timestamp_compare(
    ts_timestamp_ptr_type ts1,
    ts_timestamp_ptr_type ts2,
    int operation)
```

Include file

dsts.h

Purpose

To compare two time stamps.

Parameters

ts1 Specifies the pointer to the first time stamp structure.

ts2 Specifies the pointer to the second time stamp structure.

operation Specifies the kind of operation; the following symbols are predefined:

Predefined Symbol	Meaning
TS_COMPARE_LT	less than
TS_COMPARE_LE	less than or equal to
TS_COMPARE_EQ	equal
TS_COMPARE_GE	greater than or equal to
TS_COMPARE_GT	greater than

Return value

This function returns the operation result; the following symbols are predefined:

Value	Meaning
= 0	Result is false
!= 0	Result is true

Related topics**Basics**

[Basic Principles of Time-Stamping..... 51](#)

References

[ts_timestamp_interval..... 61](#)

ts_timestamp_interval

Syntax

```
double ts_timestamp_interval(
    ts_timestamp_ptr_type ts1,
    ts_timestamp_ptr_type ts2)
```

Include file

`dsts.h`

Purpose

To calculate the interval in seconds between time stamps 1 and 2.

Parameters

ts1 Specifies the pointer to the first time stamp structure.
ts2 Specifies the pointer to the second time stamp structure.

Return value

This function returns the interval between time stamps 1 and 2 in seconds.

Related topics**Basics**

[Basic Principles of Time-Stamping..... 51](#)

References

[ts_timestamp_compare..... 60](#)

ts_time_offset

Syntax

```
void ts_time_offset(  
    double reference_time,  
    ts_timestamp_ptr_type ts1,  
    ts_timestamp_ptr_type ts2,  
    ts_timestamp_ptr_type ts_ta)
```

Include file

dsts.h

Purpose

To calculate the time offset.

Result

The interval between time stamps 1 and 2 is calculated and the difference between the time stamps is added to the reference time. The absolute time is returned as a time stamp.

Parameters

reference_time Specifies the reference time in seconds.

ts1 Specifies the pointer to the first time stamp structure.

ts2 Specifies the pointer to the second time stamp structure.

ts_ta Specifies the pointer to the time stamp structure for the calculated value.

Return value

None

Related topics**Basics**

[Basic Principles of Time-Stamping..... 51](#)

References

[ts_timestamp_offset..... 63](#)

ts_timestamp_offset

Syntax	<pre>void ts_timestamp_offset(ts_timestamp_ptr_type ts_reference, ts_timestamp_ptr_type ts1, ts_timestamp_ptr_type ts2, ts_timestamp_ptr_type ts_ta)</pre>
Include file	dsts.h
Purpose	To calculate the time offset.
Result	The interval between time stamps 1 and 2 is calculated and the difference between the time stamps is added to the reference time stamp. The absolute time is returned as a time stamp.
Parameters	<p>ts_reference Specifies the pointer to the time stamp structure holding the reference time.</p> <p>ts1 Specifies the pointer to the first time stamp structure.</p> <p>ts2 Specifies the pointer to the second time stamp structure.</p> <p>ts_ta Specifies the pointer to the time stamp structure holding the absolute time in seconds.</p>
Return value	None
Related topics	<p>Basics</p> <p>Basic Principles of Time-Stamping..... 51</p> <p>References</p> <p>ts_time_offset..... 62</p>

ts_time_calculate

Syntax	<pre>double ts_time_calculate(ts_timestamp_ptr_type ts)</pre>
---------------	---

Include file	<code>dsts.h</code>
---------------------	---------------------

Purpose	To convert a time stamp structure to a time value in seconds.
----------------	---

Parameters	ts Specifies the pointer to a time stamp structure.
-------------------	--

Return value	This function returns the time corresponding to the time stamp.
---------------------	---

Related topics

Basics

[Basic Principles of Time-Stamping..... 51](#)

References

[ts_timestamp_offset..... 63](#)

ts_timestamp_calculate

Syntax

```
void ts_time_calculate(  
    double time,  
    ts_timestamp_ptr_type ts)
```

Include file	<code>dsts.h</code>
---------------------	---------------------

Purpose	To convert a time value in seconds to a time stamp structure.
----------------	---

Parameters	time Specifies the time in seconds. ts Specifies the pointer to a time stamp structure for the calculated value.
-------------------	---

Return value	None
---------------------	------

Related topics**Basics**[Basic Principles of Time-Stamping.....](#) 51**References**[ts_time_calculate.....](#) 63

Timer A

Introduction

Timer A is a down counter generating an interrupt whenever it reaches zero. The period value is then reloaded automatically. Timer A is also used by the **RTLIB_SRT_PERIOD** standard macro as the default sampling rate timer.

For further information on Timer A, refer to [Timer A and Timer D \(DS1006 Features !\[\]\(0f848bbd71cef6b345273b16f905912a_img.jpg\)\)](#).

Where to go from here

Information in this section

Example of Using Timer A Functions.....	66
Gives you an example how to use Timer A.	
RTLIB_SRT_PERIOD.....	67
To set a new period of Timer A and restart it immediately.	
ds1006_timerA_period_set.....	68
To set the period of Timer A.	
ds1006_timerA_period_reload.....	69
To set a new period of Timer A and restart it immediately.	
ds1006_timerA_read.....	69
To read the current value of Timer A.	
ds1006_timerA_start.....	70
To start Timer A.	
ds1006_timerA_stop.....	70
To stop Timer A.	

Information in other sections

For information on handling Timer A interrupts	
Timer Interrupt Control.....	83
Interrupt Handling.....	95

Example of Using Timer A Functions

Example

The following example demonstrates how to use Timer A functions.

```
#include <Brtenv.h>
#define DT 1.0e-4          /* 100 µs simulation step size */
```

```

/* ++ variables for host PC ++++++ */
Float64 exec_time, timeA; /* execution time */
void ad_routine(void)
{
    ts_timestamp_type ts;
    Float64 old_timeA;
    RTLIB_SRT_ISR_BEGIN(); /* overrun check TimerA */
    ds1006_timerA_read(&old_timeA);

    ts_timestamp_read(&ts);
    host_service(1, &ts); /* data acquisition service */
    /* +++ do something +++ */
    ds1006_timerA_read(&timeA);
    exec_time = old_timeA - timeA; /* exec time with Timer A */
    RTLIB_SRT_ISR_END(); /* overrun check TimerA */
}
void main(void)
{
    /* init processor board */
    init();
    /* set period of timerA */
    timeA = DT;
    ds1006_timerA_period_set(timeA);
    /* periodic event in ISR */
    RTLIB_SRT_START(timeA, ad_routine);
    /* Background tasks */
    while(1)
    {
        RTLIB_BACKGROUND_SERVICE(); /* host PC service */
    }
}

```

RTLIB_SRT_PERIOD

Syntax

```
RTLIB_SRT_PERIOD(Float64 time)
```

Include file

dsstd.h

Purpose

To set a new period of Timer A and restart it immediately.

Description

The new value is loaded immediately: Timer A is stopped, the new value is set, and Timer A is started again.

Parameters

time Specifies the period in seconds.

Return value	None
Related topics	<div>References</div> <div> ds1006_timerA_period_set..... 68 Standard Macros..... 242 </div>

ds1006_timerA_period_set

Syntax	<code>void ds1006_timerA_period_set(Float64 time)</code>
Include file	<code>tmr1006.h</code>
Purpose	To set the period of Timer A.
Description	If the timer is not running, the new value is loaded immediately. If the timer is running, the new value is loaded the next time the timer reaches zero.
Parameters	time Specifies the period in seconds.
Return value	None
Related topics	<div>Examples</div> <div> Example of Using Timer A Functions..... 66 </div> <div>References</div> <div> ds1006_timerA_period_reload..... 69 </div>

ds1006_timerA_period_reload

Syntax	<code>void ds1006_timerA_period_reload(Float64 time)</code>
Include file	<code>tmr1006.h</code>
Purpose	To set a new period of Timer A and restart it immediately.
Description	The new value is loaded immediately: Timer A is stopped, the new value is set, and Timer A is started again.
Parameters	time Specifies the period in seconds.
Return value	None
Related topics	References <div> ds1006_timerA_period_set..... 68 RTLIB_SRT_PERIOD..... 67 </div>

ds1006_timerA_read

Syntax	<code>void ds1006_timerA_read(Float64 *time)</code>
Include file	<code>tmr1006.h</code>
Purpose	To read the current value of Timer A.
Parameters	time Specifies the pointer to the current value of Timer A. The value is stated in seconds.
Return value	None

ds1006_timerA_start

Syntax

```
void ds1006_timerA_start(void)
```

Include file

```
tmr1006.h
```

Purpose

To start Timer A.

Description

If no period is set, the counter starts with the highest counter value (0xFFFF FFFF).

Tip

Use `ds1006_timerA_period_set` to set the period.

Return value

None

Related topics**References**

[ds1006_timerA_period_set..... 68](#)

ds1006_timerA_stop

Syntax

```
void ds1006_timerA_stop(void)
```

Include file

```
tmr1006.h
```

Purpose

To stop Timer A.

Tip

Use `ds1006_timerA_start` to resume from the current value.

Return value	None
Related topics	<div>References<div>ds1006_timerA_start..... 70</div></div>

Timer B

Introduction

Timer B is a counter generating an interrupt when it reaches its compare value and continues counting. Thus, Timer B is designed only for single timer events. If your model requires periodic timer events, use Timer A (refer to [Timer A](#) on page 66). If Timer A is already used, use Timer B and set its compare value periodically (function `ds1006_timerB_compare_set_periodically`).

For further information on Timer B, refer to [Timer B \(DS1006 Features !\[\]\(6605b201d6f14d9b3bcb8ab5f274d107_img.jpg\)](#)).

Where to go from here

Information in this section

[Example of Using Timer B Functions..... 72](#)

Gives you an example how to use Timer B.

[ds1006_timerB_init..... 73](#)

To initialize Timer B.

[ds1006_timerB_compare_set..... 74](#)

To set the new compare value.

[ds1006_timerB_compare_set_periodically..... 75](#)

To periodically set a new compare value.

[ds1006_timerB_read..... 76](#)

To read the current value of Timer B.

[ds1006_timerB_start..... 76](#)

To start Timer B.

[ds1006_timerB_stop..... 77](#)

To stop Timer B.

Information in other sections

For information on handling Timer B interrupts

[Timer Interrupt Control..... 83](#)

[Interrupt Handling..... 95](#)

Example of Using Timer B Functions

Example

The following example demonstrates how to use Timer B functions.

```
#include <Brtenv.h>
#define DT 1e-4          /* 100 µs simulation step size */
```



```

/* ++ variables for execution time profiling ++++++ */
Float64 exec_time;          /* execution time */
Float64 timerB;             /* timerB read value */
/* ++ adjust values for timerB ++++++ */
Float64 upc_period = .001;  /* upcounter period in sec */
UInt16 scale_value = 2;     /* set the scaling of timerB */
/* ++ counter for Interrupt service functions ++++++ */
Int32 timerB_counter = 0;
void isr_timerB(void)
{
    ts_timestamp_type ts;
    ds1006_begin_isr_timerB();      /* overrun check */
    RTLIB_TIC_START();
    timerB_counter++;              /* counter for timerB interrupts */
    ds1006_timerB_read(&timerB);    /* read timerB */
    ts_timestamp_read(&ts);
    host_service(1, &ts);          /* data acquisition service */
    exec_time = RTLIB_TIC_READ();
    ds1006_end_isr_timerB();        /* overrun check */
}
void main(void)
{
    /* init processor board */
    init();
    /* periodic event with TimerB */
    ds1006_start_isr_timerB(scale_value,
                           upc_period,
                           isr_timerB);
    /* Background task */
    while(1)
    {
        RTLIB_BACKGROUND_SERVICE(); /* host PC service */
    }
}

```

ds1006_timerB_init

Syntax

```
void ds1006_timerB_init(UInt16 scale)
```

Include file

```
tmr1006.h
```

Purpose

To initialize Timer B.

Parameters

scale Specifies a value within the range 0 ... 7 that defines the prescaler setting of Timer B as a function of the bus clock.

Scale Value	Timer B Clock/ Bus Clock	(if bus clock is 133 MHz)	
		Timer B Clock	Prescaler Period
0	1/4	33.25 MHz	30 ns
1	1/8	16.625 MHz	60 ns
2	1/16	8.3125 MHz	120 ns
3	1/32	4.15625 MHz	241 ns
4	1/64	2.078125 MHz	481 ns
5	1/128	1.0390625 MHz	962 ns
6	1/256	0.51953125 MHz	1925 ns
7	1/512	0.259765625 MHz	3850 ns

Return value

None

Example

To achieve a prescaler period of 120 ns the prescaler must be set to 1/16, assuming a bus clock of 133 MHz:

```
ds1006_timerB_init(2);
```

Related topics**References**

[ds1006_timerB_compare_set..... 74](#)

ds1006_timerB_compare_set

Syntax

```
void ds1006_timerB_compare_set(Float64 delta_time)
```

Include file

tmr1006.h

Purpose

To set the new compare value.

Description

The compare value to be written to the Timer B compare register is calculated by adding the `delta_time` to the current timer value. When the counter value matches the value of the compare register, Timer B generates an interrupt.

To make the Timer B interrupt available, refer to [Timer Interrupt Control](#) on page 83 and [Interrupt Handling](#) on page 95.

If you want to generate a Timer B interrupt periodically, use the function `ds1006_timerB_compare_set_periodically`.

Parameters	delta_time Specifies the period in seconds.
-------------------	--

Return value	None
---------------------	------

Related topics

References

ds1006_timerB_compare_set_periodically	75
ds1006_timerB_init	73

ds1006_timerB_compare_set_periodically

Syntax

```
void ds1006_timerB_compare_set_periodically(
    Float64 delta_time)
```

Include file	tmr1006.h
---------------------	-----------

Purpose	To periodically set a new compare value.
----------------	--

Description

This function is used in the Timer B interrupt service routine to make Timer B a periodic timer. The new compare value to be written to the Timer B compare register is calculated by adding the `delta_time` to the old compare value.

When the counter value matches the value of the compare register, Timer B generates an interrupt.

This function is automatically called in your interrupt service routine when using `ds1006_begin_isr_timerB`.

To make the Timer B interrupt available, refer to [Timer Interrupt Control](#) on page 83 and [Interrupt Handling](#) on page 95.

Parameters	delta_time Specifies the period in seconds.
-------------------	--

Return value	None
---------------------	------

Related topics**References**

ds1006_begin_isr_timerA.....	85
ds1006_timerB_compare_set.....	74

ds1006_timerB_read

Syntax

```
void ds1006_timerB_read(Float64 *time)
```

Include file

```
tmr1006.h
```

Purpose

To read the current value of Timer B.

Parameters

time Specifies the pointer to the current value of Timer B. The value is given in seconds.

Return value

None

Related topics**Examples**

Example of Using Timer B Functions.....	72
---	--------------------

ds1006_timerB_start

Syntax

```
void ds1006_timerB_start(void)
```

Include file

```
tmr1006.h
```

Purpose To start Timer B.

Tip

Use `ds1006_timerB_compare_set` to set the compare value.

Return value None

Related topics

References

[ds1006_timerB_compare_set..... 74](#)
[ds1006_timerB_stop..... 77](#)

ds1006_timerB_stop

Syntax `void ds1006_timerB_stop(void)`

Include file `tmr1006.h`

Purpose To stop Timer B.

Tip

Use `ds1006_timerB_start` to continue.

Return value None

Related topics

References

[ds1006_timerB_start..... 76](#)

Timer D

Introduction

Timer D is functionally identical to Timer A. Timer D is a down counter generating an interrupt whenever it reaches zero. The period value is then reloaded automatically.

For further information on Timer D, refer to [Timer A and Timer D \(DS1006 Features !\[\]\(d3fb9f94af8b26d1c844efa9a98805b0_img.jpg\)\)](#).

Where to go from here

Information in this section

Example of Using Timer D functions.....	78
Gives you an example how to use Timer D.	
ds1006_timerD_period_set.....	79
To define the period of Timer D.	
ds1006_timerD_period_reload.....	80
To set a new period of Timer D and restart it immediately.	
ds1006_timerD_read.....	80
To read the current value of Timer D.	
ds1006_timerD_start.....	81
To start Timer D.	
ds1006_timerD_stop.....	82
To stop Timer D.	

Information in other sections

For information on handling Timer D interrupts.	
Timer Interrupt Control.....	83
Interrupt Handling.....	95

Example of Using Timer D functions

Example

The following example demonstrates how to use Timer D functions.

```
#include <Brtenv.h>
#define DT 1.0e-4          /* 100 µs simulation step size */
/*-- variables for host PC -----*/
Float64 exec_time, timeD;  /* execution time */
```

```

void ad_routine(void)
{
    ts_timestamp_type ts;
    Float64 old_timeD;
    ds1006_begin_isr_timerD(); /* overrun check TimerD */
    ds1006_timerD_read(&old_timeD);
    ts_timestamp_read(&ts);
    host_service(1, &ts); /* data acquisition service */
    /*--- do something ---*/
    ds1006_timerD_read(&timeD);
    exec_time = old_timeD - timeD; /* exec time with Timer D */
    ds1006_end_isr_timerD(); /* overrun check TimerD */
}

void main(void)
{
    /* init processor board */
    init();
    /* set period of timerD */
    timeD = DT;
    ds1006_timerD_period_set(timeD);
    /* periodic event in ISR */
    ds1006_start_isr_timerD(timeD, ad_routine);
    /* Background tasks */
    while(1)
    {
        RTLIB_BACKGROUND_SERVICE(); /* host PC service */
    }
}

```

ds1006_timerD_period_set

Syntax	<code>void ds1006_timerD_period_set(Float64 time)</code>
Include file	<code>tmr1006.h</code>
Purpose	To define the period of Timer D.
Description	If the timer is not running, the new value is loaded immediately. If the timer is running, the new value is loaded the next time the timer reaches zero.
Parameters	time Specifies the period in seconds.
Return value	None

Related topics**Examples**

[Example of Using Timer D functions.....](#) 78

References

[ds1006_timerD_period_reload.....](#) 80

ds1006_timerD_period_reload

Syntax

```
void ds1006_timerD_period_reload(Float64 time)
```

Include file

tmr1006.h

Purpose

To set a new period of Timer D and restart it immediately.

Description

The new value is loaded immediately. Timer D is stopped, the new value is set, and Timer D is started again.

Parameters

time Specifies the period in seconds.

Return value

None

Related topics**References**

[ds1006_timerD_period_set.....](#) 79

ds1006_timerD_read

Syntax

```
void ds1006_timerD_read(Float64 *time)
```

Include file

tmr1006.h

Purpose	To read the current value of Timer D.
Parameters	time Specifies the pointer to the current value of Timer D. The value is stated in seconds.
Return value	None
Related topics	Examples Example of Using Timer D functions..... 78

ds1006_timerD_start

Syntax	<code>void ds1006_timerD_start(void)</code>
Include file	tmr1006.h
Purpose	To start Timer D.
Description	If no period is set, the counter starts with the highest counter value (0xFFFF FFFF). <div> Tip Use <code>ds1006_timerD_period_set</code> to set the period. </div>
Return value	None
Related topics	References ds1006_timerD_period_set..... 79

ds1006_timerD_stop

Syntax

```
void ds1006_timerD_stop(void)
```

Include file

```
tmr1006.h
```

Purpose

To stop Timer D.

Tip

Use `ds1006_timerD_start` to resume from the current value.

Return value

None

Related topics**References**

[ds1006_timerD_start..... 81](#)

Timer Interrupt Control

Introduction

These functions are used to install interrupt service routines for the available timers and to perform overrun checks for the defined interrupt service routines.

Tip

Here you find the descriptions of platform-specific functions and generic `RTLIB_SRT_XXX` macros. It is recommended to use the generic macros if available.

Where to go from here

Information in this section

ds1006_begin_isr_timerA	85
To check for an overrun in the interrupt service routine assigned to Timer A.	
ds1006_begin_isr_timerB	85
To check for an overrun in the interrupt service routine assigned to Timer B.	
ds1006_begin_isr_timerD	86
To check for an overrun in the interrupt service routine assigned to Timer D.	
ds1006_end_isr_timerA	87
To check for an overrun in the interrupt service routine assigned to Timer A.	
ds1006_end_isr_timerB	87
To check for an overrun in the interrupt service routine assigned to Timer B.	
ds1006_end_isr_timerD	88
To check for an overrun in the interrupt service routine assigned to Timer D.	
ds1006_start_isr_timerA	89
To install an interrupt service routine for Timer A.	
ds1006_start_isr_timerB	90
To install an interrupt service routine for Timer B.	
ds1006_start_isr_timerD	91
To install an interrupt service routine for Timer D.	
RTLIB_SRT_ISR_BEGIN	92
To check for an overrun in the interrupt service routine assigned to Timer A.	
RTLIB_SRT_ISR_END	93
To check the overrun in the interrupt service routine assigned to Timer A.	
RTLIB_SRT_START	93
To install an interrupt service routine for Timer A.	

Information in other sections

Interrupt Handling	95
--	----

ds1006_begin_isr_timerA

Syntax	<code>ds1006_begin_isr_timerA()</code>				
Include file	<code>int1006.h</code>				
Purpose	To check for an overrun in the interrupt service routine assigned by ds1006_start_isr_timerA .				
Description	When the execution time of the interrupt service routine exceeds the interrupt period (overrun), the interrupt is stopped, and an appropriate error message is generated.				
Return value	None				
Example	<p>This example shows an interrupt service routine with overrun check:</p> <pre>void timerA_interrupt(void) { ds1006_begin_isr_timerA(); /* interrupt service routine */ ds1006_end_isr_timerA(); }</pre>				
Related topics	<p>References</p> <table> <tr> <td>ds1006_end_isr_timerA.....</td> <td>87</td> </tr> <tr> <td>RTLIB_SRT_ISR_BEGIN.....</td> <td>92</td> </tr> </table>	ds1006_end_isr_timerA.....	87	RTLIB_SRT_ISR_BEGIN.....	92
ds1006_end_isr_timerA.....	87				
RTLIB_SRT_ISR_BEGIN.....	92				

ds1006_begin_isr_timerB

Syntax	<code>ds1006_begin_isr_timerB()</code>
Include file	<code>int1006.h</code>

Purpose	To check for an overrun in the interrupt service routine assigned by ds1006_start_isr_timerB and to reload the compare value.				
Description	When the execution time of the interrupt service routine exceeds the interrupt period (overrun), the interrupt is stopped, and an appropriate error message is generated.				
Return value	None				
Example	<p>This example shows an interrupt service routine with overrun check:</p> <pre>void timerB_interrupt(void) { ds1006_begin_isr_timerB(); /* interrupt service routine */ ds1006_end_isr_timerB(); }</pre>				
Related topics	<p>References</p> <table> <tr> <td>ds1006_end_isr_timerB</td> <td>87</td> </tr> <tr> <td>ds1006_start_isr_timerB</td> <td>90</td> </tr> </table>	ds1006_end_isr_timerB	87	ds1006_start_isr_timerB	90
ds1006_end_isr_timerB	87				
ds1006_start_isr_timerB	90				

ds1006_begin_isr_timerD

Syntax	<code>ds1006_begin_isr_timerD()</code>
Include file	<code>int1006.h</code>
Purpose	To check for an overrun in the interrupt service routine assigned by ds1006_start_isr_timerD .
Description	When the execution time of the interrupt service routine exceeds the interrupt period (overrun), the interrupt is stopped, and an appropriate error message is generated.
Return value	None

Example

This example shows an interrupt service routine with overrun check:

```
void timerD_interrupt(void)
{
    ds1006_begin_isr_timerD();
    /* interrupt service routine */
    ds1006_end_isr_timerD();
}
```

Related topics**References**

[ds1006_end_isr_timerD..... 88](#)

ds1006_end_isr_timerA

Syntax

```
ds1006_end_isr_timerA()
```

Include file

int1006.h

Purpose

To check for an overrun in the interrupt service routine assigned by **ds1006_start_isr_timerA**.

Return value

None

Related topics**References**

[ds1006_begin_isr_timerA..... 85](#)
[ds1006_start_isr_timerA..... 89](#)
[RTLIB_SRT_ISR_END..... 93](#)

ds1006_end_isr_timerB

Syntax

```
ds1006_end_isr_timerB()
```

Include file	<code>int1006.h</code>
---------------------	------------------------

Purpose	To check for an overrun in the interrupt service routine assigned by <code>ds1006_start_isr_timerB</code> .
----------------	---

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

ds1006_begin_isr_timerB.....	85
ds1006_start_isr_timerB.....	90

ds1006_end_isr_timerD

Syntax	<code>ds1006_end_isr_timerD()</code>
---------------	--------------------------------------

Include file	<code>int1006.h</code>
---------------------	------------------------

Purpose	To check for an overrun in the interrupt service routine assigned by <code>ds1006_start_isr_timerD</code> .
----------------	---

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

ds1006_begin_isr_timerD.....	86
ds1006_start_isr_timerD.....	91

ds1006_start_isr_timerA

Syntax

```
ds1006_start_isr_timerA(
    Float64 sampling_period,
    isr_function_name)
```

Include file

int1006.h

Purpose

To install `isr_function_name` as an interrupt service routine for Timer A.

Description

The function sets the period of Timer A, installs the specified routine as interrupt handler, and starts Timer A.

If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use `ds1006_begin_isr_timerA` and `ds1006_end_isr_timerA` in your interrupt service routine to install an overrun check.

Parameters

sampling_period Specifies the period in seconds.

isr_function_name Specifies the name of the function to be assigned to the Timer A interrupt. This function must not have an input parameter or a return value, i.e., `void isr_function_name(void)`.

Return value

None

Example

This example installs the function `timerA_interrupt`, which is called when the Timer A interrupt occurs, namely every 20 μ s:

```
ds1006_start_isr_timerA(20e-6, timerA_interrupt);
```

Related topics

References

ds1006_begin_isr_timerA	85
ds1006_end_isr_timerA	87
RTLIB_SRT_START	93

ds1006_start_isr_timerB

Syntax

```
ds1006_start_isr_timerB(
    UInt16 scale,
    Float64 sampling_period,
    isr_function_name)
```

Include file

int1006.h

Purpose

To install `isr_function_name` as an interrupt service routine for Timer B and initialize Timer B.

Description

The function sets the compare value of Timer B, installs the specified routine as interrupt handler, and starts Timer B. Because Timer B is not a periodic timer, you must use `ds1006_begin_isr_timerB` and `ds1006_end_isr_timerB` in your interrupt service routine to reload the compare value. In addition, you install an overrun check that prevents the execution time of the interrupt service routine from exceeding the interrupt period.

Parameters

scale Specifies a value within the range 0 ... 7 that defines the prescaler setting of Timer B as a function of the bus clock.

Scale Value	Timer B Clock/ Bus Clock	(if bus clock is 133 MHz)	
		Timer B Clock	Prescaler Period
0	1/4	33.25 MHz	30 ns
1	1/8	16.625 MHz	60 ns
2	1/16	8.3125 MHz	120 ns
3	1/32	4.15625 MHz	241 ns
4	1/64	2.078125 MHz	481 ns
5	1/128	1.0390625 MHz	962 ns
6	1/256	0.51953125 MHz	1925 ns
7	1/512	0.259765625 MHz	3850 ns

sampling_period Specifies the period in seconds.

isr_function_name Specifies the name of the function to be assigned to the Timer B interrupt. This function must not have an input parameter or a return value, i.e., `void isr_function_name(void)`.

Return value

None

Example

This example installs the function `timerB_interrupt`, which is called when the Timer B interrupt occurs, namely every 100 μ s:

```
ds1006_start_isr_timerB(0, 100e-6, timerB_interrupt)
```

Related topics**References**

ds1006_begin_isr_timerB	85
ds1006_end_isr_timerB	87

ds1006_start_isr_timerD

Syntax

```
ds1006_start_isr_timerD(
    Float64 sampling_period,
    isr_function_name)
```

Include file

`int1006.h`

Purpose

To install `isr_function_name` as an interrupt service routine for Timer D.

Description

The function sets the period of Timer D, installs the specified routine as interrupt handler, and starts Timer D.

If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use `ds1006_begin_isr_timerD` and `ds1006_end_isr_timerD` in your interrupt service routine to install an overrun check.

Parameters

sampling_period Specifies the period in seconds.

isr_function_name Specifies the name of the function to be assigned to the Timer D interrupt. This function must not have an input parameter or a return value, i.e., `void isr_function_name(void)`.

Return value

None

Example

This example installs the function `timerD_interrupt`, which is called when the Timer D interrupt occurs, namely every 20 μ s:

```
ds1006_start_isr_timerD(20e-6, timerD_interrupt);
```

Related topics**References**

ds1006_begin_isr_timerD.....	86
ds1006_end_isr_timerD.....	88

RTLIB_SRT_ISR_BEGIN

Syntax

```
RTLIB_SRT_ISR_BEGIN()
```

Include file

```
dsstd.h
```

Purpose

To check for an overrun in the interrupt service routine assigned by `RTLIB_SRT_START`.

Description

When the execution time of the interrupt service routine exceeds the interrupt period (overrun), the interrupt is stopped, and an appropriate error message is generated.

Return value

None

Example

This example shows an interrupt service routine with overrun check:

```
void timerA_interrupt(void)
{
    RTLIB_SRT_ISR_BEGIN();
    /* interrupt service routine */
    RTLIB_SRT_ISR_END();
}
```

Related topics**References**

ds1006_begin_isr_timerA.....	85
RTLIB_SRT_START.....	93

RTLIB_SRT_ISR_END

Syntax	<code>RTLIB_SRT_ISR_END()</code>
Include file	<code>dsstd.h</code>
Purpose	To check for an overrun in the interrupt service routine assigned by <code>RTLIB_SRT_START</code> .
Return value	None
Related topics	References <div> ds1006_end_isr_timerA..... 87 RTLIB_SRT_START..... 93 </div>

RTLIB_SRT_START

Syntax	<pre>RTLIB_SRT_START(Float64 sampling_period, isr_function_name)</pre>
Include file	<code>dsstd.h</code>
Purpose	To install <code>isr_function_name</code> as an interrupt service routine for Timer A.
Description	<p>The function sets the period of Timer A, installs the specified routine as interrupt handler, and starts Timer A.</p> <p>If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use <code>RTLIB_SRT_ISR_BEGIN</code> and <code>RTLIB_SRT_ISR_END</code> in your interrupt service routine to install an overrun check.</p>

Parameters	<p>sampling_period Specifies the period in seconds.</p> <p>isr_function_name Specifies the name of the function to be assigned to the Timer A interrupt. This function must not have an input parameter or a return value, i.e., <code>void isr_function_name(void)</code>.</p>
Return value	None
Example	<p>This example installs the function <code>timerA_interrupt</code>, which is called when the Timer A interrupt occurs, namely every 20 μs:</p> <pre>RTLIB_SRT_START(20e-6, timerA_interrupt);</pre>
Related topics	<p>References</p> <div> ds1006_start_isr_timerA.....89 </div>

Interrupt Handling

Introduction

Use the interrupt handling functions to make interrupts available as trigger sources. If you want to use an interrupt, you have to install an appropriate handler and enable interrupt handling. The interrupt handling uses the interrupt identification (IntId) to identify the interrupt handler that has been installed for this interrupt. Whether or not an interrupt has been generated is indicated by the interrupt flag.

Note

For examples of the installation of interrupt service routines for the Timer A, Timer B and Timer D interrupts, refer to [ds1006_set_interrupt_vector](#) on page 110 and [Timer Interrupt Control](#) on page 83.

For further information on the interrupt handling, refer to [Interrupt Controller \(DS1006 Features !\[\]\(8d0f0e0fe25b320c33272c52aec1fbca_img.jpg\)\)](#).

Interrupt service routine type

The interrupt service routine type is defined as follows:

```
typedef void (*DS1006_Int_Handler_Type)(void)
```

Where to go from here

Information in this section

ds1006_disable_hardware_int	97
To disable the specified hardware interrupt when the interrupts are still globally enabled.	
ds1006_disable_hardware_int_bm	98
To disable several hardware interrupts when the interrupts are still globally enabled.	
ds1006_enable_hardware_int	99
To enable the specified hardware interrupt.	
ds1006_enable_hardware_int_bm	101
To enable several hardware interrupts.	
ds1006_get_interrupt_flag	102
To get the interrupt flag for the specified interrupt.	
ds1006_get_interrupt_flag_bm	104
To get the interrupt flag for several interrupts.	
ds1006_get_interrupt_status	105
To get the interrupt status.	
ds1006_get_interrupt_vector	106
To get the address of the interrupt service routine related to the given interrupt.	
ds1006_reset_interrupt_flag	107
To reset the interrupt flag for the specified interrupt.	
ds1006_reset_interrupt_flag_bm	108
To reset the interrupt flag for several interrupts.	
ds1006_set_interrupt_status	109
To set the interrupt status.	
ds1006_set_interrupt_vector	110
To install an interrupt service routine for the selected interrupt.	
RTLIB_INT_DISABLE	112
To globally disable the interrupts.	
RTLIB_INT_ENABLE	113
To globally enable the interrupts.	
RTLIB_INT_RESTORE	114
To restore the previous state.	
RTLIB_INT_SAVE_AND_DISABLE	114
To disable the interrupts globally and save the state.	
RTLIB_SRT_DISABLE	115
To disable the hardware interrupt for the sampling rate timer when the interrupts are still globally enabled.	

[RTLIB_SRT_ENABLE](#)..... 116
To enable the hardware interrupt for the sampling rate timer.

ds1006_disable_hardware_int

Syntax	<code>void ds1006_disable_hardware_int(UInt32 IntID)</code>																																		
Include file	<code>int1006.h</code>																																		
Purpose	To disable the specified hardware interrupt when the interrupts are still globally enabled (see RTLIB_INT_ENABLE).																																		
Description	This function sets the corresponding bit of the Interrupt Mask Register (IMR).																																		
Parameters	<p>IntID Specifies the interrupt that is to be disabled.</p> <p>The following symbols are predefined:</p> <table> <tr> <th>Predefined Symbol</th><th>Meaning</th></tr> <tr> <td>DS1006_INT_IOERR</td><td>PHS-bus I/O error interrupt</td></tr> <tr> <td>DS1006_INT_WD</td><td>Watchdog interrupt</td></tr> <tr> <td>DS1006_INT_PHS_0</td><td>PHS-bus interrupt line 0</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td>DS1006_INT_PHS_7</td><td>PHS-bus interrupt line 7</td></tr> <tr> <td>DS1006_INT_TIMER_A</td><td>Timer A interrupt</td></tr> <tr> <td>DS1006_INT_TIMER_B</td><td>Timer B interrupt</td></tr> <tr> <td>DS1006_INT_GL_0</td><td>Gigalink 0 interrupt</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td>DS1006_INT_GL_3</td><td>Gigalink 3 interrupt</td></tr> <tr> <td>DS1006_INT_SERIAL_UART</td><td>Serial UART interrupt</td></tr> <tr> <td>DS1006_INT_HOST</td><td>Host interrupt</td></tr> <tr> <td>DS1006_INT_MACROTICK</td><td>Macrotick interrupt</td></tr> <tr> <td>DS1006_INT_FWD_WD</td><td>Forwarded watchdog interrupt</td></tr> <tr> <td>DS1006_INT_FWD_TIMER_A</td><td>Forwarded Timer A interrupt</td></tr> <tr> <td>DS1006_INT_FWD_TIMER_B</td><td>Forwarded Timer B interrupt</td></tr> </table>	Predefined Symbol	Meaning	DS1006_INT_IOERR	PHS-bus I/O error interrupt	DS1006_INT_WD	Watchdog interrupt	DS1006_INT_PHS_0	PHS-bus interrupt line 0	DS1006_INT_PHS_7	PHS-bus interrupt line 7	DS1006_INT_TIMER_A	Timer A interrupt	DS1006_INT_TIMER_B	Timer B interrupt	DS1006_INT_GL_0	Gigalink 0 interrupt	DS1006_INT_GL_3	Gigalink 3 interrupt	DS1006_INT_SERIAL_UART	Serial UART interrupt	DS1006_INT_HOST	Host interrupt	DS1006_INT_MACROTICK	Macrotick interrupt	DS1006_INT_FWD_WD	Forwarded watchdog interrupt	DS1006_INT_FWD_TIMER_A	Forwarded Timer A interrupt	DS1006_INT_FWD_TIMER_B	Forwarded Timer B interrupt
Predefined Symbol	Meaning																																		
DS1006_INT_IOERR	PHS-bus I/O error interrupt																																		
DS1006_INT_WD	Watchdog interrupt																																		
DS1006_INT_PHS_0	PHS-bus interrupt line 0																																		
...	...																																		
DS1006_INT_PHS_7	PHS-bus interrupt line 7																																		
DS1006_INT_TIMER_A	Timer A interrupt																																		
DS1006_INT_TIMER_B	Timer B interrupt																																		
DS1006_INT_GL_0	Gigalink 0 interrupt																																		
...	...																																		
DS1006_INT_GL_3	Gigalink 3 interrupt																																		
DS1006_INT_SERIAL_UART	Serial UART interrupt																																		
DS1006_INT_HOST	Host interrupt																																		
DS1006_INT_MACROTICK	Macrotick interrupt																																		
DS1006_INT_FWD_WD	Forwarded watchdog interrupt																																		
DS1006_INT_FWD_TIMER_A	Forwarded Timer A interrupt																																		
DS1006_INT_FWD_TIMER_B	Forwarded Timer B interrupt																																		

Predefined Symbol	Meaning
DS1006_INT_FWD_HOST	Forwarded host interrupt
DS1006_INT_TIMER_D	Timer D interrupt

Note

Level-triggered interrupts (serial UART interrupts) have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value None

Related topics**References**

ds1006_disable_hardware_int_bm	98
ds1006_enable_hardware_int	99
RTLIB_INT_DISABLE	112

ds1006_disable_hardware_int_bm

Syntax

```
void ds1006_disable_hardware_int_bm(UINT32 flag)
```

Include file

int1006.h

Purpose

To disable several hardware interrupts when the interrupts are still globally enabled (see **RTLIB_INT_ENABLE**).

Description

This function sets the corresponding bit of the Interrupt Mask Register (IMR).

Parameters

flag Specifies the interrupts that are to be disabled. To specify more than one interrupt, you can combine the predefined symbols using the logical operator OR.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_INT_MASK_IOERR	PHS-bus I/O error interrupt
DS1006_INT_MASK_WD	Watchdog interrupt

Predefined Symbol	Meaning
DS1006_INT_MASK_PHS_0	PHS-bus interrupt line 0
...	...
DS1006_INT_MASK_PHS_7	PHS-bus interrupt line 7
DS1006_INT_MASK_TIMER_A	Timer A interrupt
DS1006_INT_MASK_TIMER_B	Timer B interrupt
DS1006_INT_MASK_GL_0	Gigalink 0 interrupt
...	...
DS1006_INT_MASK_GL_3	Gigalink 3 interrupt
DS1006_INT_MASK_SERIAL_UART	Serial UART interrupt
DS1006_INT_MASK_HOST	Host interrupt
DS1006_INT_MASK_MACROTICK	Macrotick interrupt
DS1006_INT_MASK_FWD_WD	Forwarded Watchdog interrupt
DS1006_INT_MASK_FWD_TIMER_A	Forwarded Timer A interrupt
DS1006_INT_MASK_FWD_TIMER_B	Forwarded Timer B interrupt
DS1006_INT_MASK_FWD_HOST	Forwarded Host interrupt
DS1006_INT_MASK_TIMER_D	Timer D interrupt

Note

Level-triggered interrupts (serial UART interrupts) have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value None

Related topics**References**

[ds1006_enable_hardware_int_bm.....](#) 101
[RTLIB_INT_ENABLE.....](#) 113

ds1006_enable_hardware_int

Syntax

```
void ds1006_enable_hardware_int(UInt32 IntID)
```

Include file

```
int1006.h
```

Purpose To enable the specified hardware interrupt.

Description This function only clears the corresponding bit of the Interrupt Mask Register (IMR). However, the specified hardware interrupt is available only when the interrupts are globally enabled (see **RTLIB_INT_ENABLE**).

Parameters **IntID** Specifies the interrupt that is to be enabled.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_INT_IOERR	PHS-bus I/O error interrupt
DS1006_INT_WD	Watchdog interrupt
DS1006_INT_PHS_0	PHS-bus interrupt line 0
...	...
DS1006_INT_PHS_7	PHS-bus interrupt line 7
DS1006_INT_TIMER_A	Timer A interrupt
DS1006_INT_TIMER_B	Timer B interrupt
DS1006_INT_GL_0	Gigalink 0 interrupt
...	...
DS1006_INT_GL_3	Gigalink 3 interrupt
DS1006_INT_SERIAL_UART	Serial UART interrupt
DS1006_INT_HOST	Host interrupt
DS1006_INT_MACROTICK	Macrotick interrupt
DS1006_INT_FWD_WD	Forwarded watchdog interrupt
DS1006_INT_FWD_TIMER_A	Forwarded Timer A interrupt
DS1006_INT_FWD_TIMER_B	Forwarded Timer B interrupt
DS1006_INT_FWD_HOST	Forwarded host interrupt
DS1006_INT_TIMER_D	Timer D interrupt

Note

Level-triggered interrupts (serial UART interrupts) have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value None

Related topics

References

ds1006_disable_hardware_int	97
ds1006_enable_hardware_int_bm	101
RTLIB_INT_ENABLE	113

ds1006_enable_hardware_int_bm

Syntax

```
void ds1006_enable_hardware_int_bm(UINT32 flag)
```

Include file

```
int1006.h
```

Purpose

To enable several hardware interrupts.

Description

This function clears the corresponding bits of the Interrupt Mask Register (IMR). However, the specified hardware interrupts are available only when the interrupts are globally enabled (see [RTLIB_INT_ENABLE](#)).

Parameters

flag Specifies the interrupts that are to be enabled. To specify more than one interrupt, you can combine the predefined symbols by using the logical operator OR.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_INT_MASK_IOERR	PHS-bus I/O error interrupt
DS1006_INT_MASK_WD	Watchdog interrupt
DS1006_INT_MASK_PHS_0	PHS-bus interrupt line 0
...	...
DS1006_INT_MASK_PHS_7	PHS-bus interrupt line 7
DS1006_INT_MASK_TIMER_A	Timer A interrupt
DS1006_INT_MASK_TIMER_B	Timer B interrupt
DS1006_INT_MASK_GL_0	Gigalink 0 interrupt
...	...
DS1006_INT_MASK_GL_3	Gigalink 3 interrupt
DS1006_INT_MASK_SERIAL_UART	Serial UART interrupt
DS1006_INT_MASK_HOST	Host interrupt
DS1006_INT_MASK_MACROTICK	Macrotick interrupt

Predefined Symbol	Meaning
DS1006_INT_MASK_FWD_WD	Forwarded Watchdog interrupt
DS1006_INT_MASK_FWD_TIMER_A	Forwarded Timer A interrupt
DS1006_INT_MASK_FWD_TIMER_B	Forwarded Timer B interrupt
DS1006_INT_MASK_FWD_HOST	Forwarded Host interrupt
DS1006_INT_MASK_TIMER_D	Timer D interrupt

Note

Level-triggered interrupts (serial UART interrupts) have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value None

Related topics**References**

[ds1006_disable_hardware_int_bm.....98](#)
[RTLIB_INT_ENABLE.....113](#)

ds1006_get_interrupt_flag

Syntax `int ds1006_get_interrupt_flag(UInt32 IntID)`

Include file `int1006.h`

Purpose To get the interrupt flag for the specified interrupt.

Description The interrupt flag indicates whether or not the specified interrupt has been generated.

Parameters

IntID Specifies the interrupt whose interrupt flag is to be read.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_INT_IOERR	PHS-bus I/O error interrupt
DS1006_INT_WD	Watchdog interrupt
DS1006_INT_PHS_0	PHS-bus interrupt line 0
...	...
DS1006_INT_PHS_7	PHS-bus interrupt line 7
DS1006_INT_TIMER_A	Timer A interrupt
DS1006_INT_TIMER_B	Timer B interrupt
DS1006_INT_GL_0	Gigalink 0 interrupt
...	...
DS1006_INT_GL_3	Gigalink 3 interrupt
DS1006_INT_SERIAL_UART	Serial UART interrupt
DS1006_INT_HOST	Host interrupt
DS1006_INT_MACROTICK	Macrotick interrupt
DS1006_INT_FWD_WD	Forwarded watchdog interrupt
DS1006_INT_FWD_TIMER_A	Forwarded Timer A interrupt
DS1006_INT_FWD_TIMER_B	Forwarded Timer B interrupt
DS1006_INT_FWD_HOST	Forwarded host interrupt
DS1006_INT_TIMER_D	Timer D interrupt

Note

Level-triggered interrupts (serial UART interrupts) have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value

This function returns the value of the interrupt flag:

Value	Meaning
0	Interrupt has not been generated
1	Interrupt has been generated

Related topics**References**

[ds1006_get_interrupt_flag_bm.....](#) 104

ds1006_get_interrupt_flag_bm

Syntax

```
int ds1006_get_interrupt_flag_bm(UINT32 flag)
```

Include file

```
int1006.h
```

Purpose

To get the interrupt flag for several interrupts.

Description

The interrupt flag indicates whether or not one of the specified interrupts has been generated.

Parameters

flag Specifies a bitmask of interrupts that are to be checked. To specify more than one interrupt, you can combine the predefined symbols using the logical operator OR.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_INT_MASK_IOERR	PHS-bus I/O error interrupt
DS1006_INT_MASK_WD	Watchdog interrupt
DS1006_INT_MASK_PHS_0	PHS-bus interrupt line 0
...	...
DS1006_INT_MASK_PHS_7	PHS-bus interrupt line 7
DS1006_INT_MASK_TIMER_A	Timer A interrupt
DS1006_INT_MASK_TIMER_B	Timer B interrupt
DS1006_INT_MASK_GL_0	Gigalink 0 interrupt
...	...
DS1006_INT_MASK_GL_3	Gigalink 3 interrupt
DS1006_INT_MASK_SERIAL_UART	Serial UART interrupt
DS1006_INT_MASK_HOST	Host interrupt
DS1006_INT_MASK_MACROTICK	Macrotick interrupt
DS1006_INT_MASK_FWD_WD	Forwarded Watchdog interrupt
DS1006_INT_MASK_FWD_TIMER_A	Forwarded Timer A interrupt
DS1006_INT_MASK_FWD_TIMER_B	Forwarded Timer B interrupt
DS1006_INT_MASK_FWD_HOST	Forwarded Host interrupt
DS1006_INT_MASK_TIMER_D	Timer D interrupt

Note

Level-triggered interrupts (serial UART interrupts) have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value

This function returns the value of the interrupt flag:

Value	Meaning
0	Interrupt has not been generated
1	At least one interrupt has been generated

Related topics**References**

[ds1006_get_interrupt_flag.....](#) 102

ds1006_get_interrupt_status

Syntax

```
UInt32 ds1006_get_interrupt_status(void)
```

Include file

int1006.h

Purpose

To get the interrupt status.

Description

This function indicates the status of the Interrupt-Enable bit (External Interrupt Enable) of the processor's EFLAGS register. Use this function if you want to disable interrupts during function execution and restore the value of the EE bit afterwards.

Return value

This function returns the value of the Interrupt-Enable bit:

Value	Meaning
0x0	Interrupt-Enable bit = 0; external interrupt disabled
0x1	Interrupt-Enable bit = 1; external interrupt enabled

Related topics**References**

[ds1006_set_interrupt_status..... 109](#)

ds1006_get_interrupt_vector

Syntax

```
DS1006_Int_Handler_Type ds1006_get_interrupt_vector(
    UInt32 IntID)
```

Include file

int1006.h

Purpose

To get the address of the interrupt service routine related to the given interrupt.

Description

Use this function to retrieve the interrupt service routine installed for a given interrupt source. If no user handler has been installed, the default handler will be returned.

Parameters

IntID Specifies the interrupt source for which the installed handler is to be returned.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_INT_IOERR	PHS-bus I/O error interrupt
DS1006_INT_WD	Watchdog interrupt
DS1006_INT_PHS_0	PHS-bus interrupt line 0
...	...
DS1006_INT_PHS_7	PHS-bus interrupt line 7
DS1006_INT_TIMER_A	Timer A interrupt
DS1006_INT_TIMER_B	Timer B interrupt
DS1006_INT_GL_0	Gigalink 0 interrupt
...	...
DS1006_INT_GL_3	Gigalink 3 interrupt
DS1006_INT_SERIAL_UART	Serial UART interrupt
DS1006_INT_HOST	Host interrupt
DS1006_INT_MACROTICK	Macro-tick interrupt

Predefined Symbol	Meaning
DS1006_INT_FWD_WD	Forwarded watchdog interrupt
DS1006_INT_FWD_TIMER_A	Forwarded Timer A interrupt
DS1006_INT_FWD_TIMER_B	Forwarded Timer B interrupt
DS1006_INT_FWD_HOST	Forwarded host interrupt
DS1006_INT_TIMER_D	Timer D interrupt

Return value This function returns the address of the interrupt service routine that is installed for this interrupt.

ds1006_reset_interrupt_flag

Syntax `void ds1006_reset_interrupt_flag(UINT32 IntID)`

Include file `int1006.h`

Purpose To reset the interrupt flag for the specified interrupt.

Parameters **IntID** Specifies the interrupt for which the interrupt flag is to be reset. The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_INT_IOERR	PHS-bus I/O error interrupt
DS1006_INT_WD	Watchdog interrupt
DS1006_INT_PHS_0	PHS-bus interrupt line 0
...	...
DS1006_INT_PHS_7	PHS-bus interrupt line 7
DS1006_INT_TIMER_A	Timer A interrupt
DS1006_INT_TIMER_B	Timer B interrupt
DS1006_INT_GL_0	Gigalink 0 interrupt
...	...
DS1006_INT_GL_3	Gigalink 3 interrupt
DS1006_INT_SERIAL_UART	Serial UART interrupt
DS1006_INT_HOST	Host interrupt
DS1006_INT_MACROTICK	Macrotick interrupt
DS1006_INT_FWD_WD	Forwarded watchdog interrupt

Predefined Symbol	Meaning
DS1006_INT_FWD_TIMER_A	Forwarded Timer A interrupt
DS1006_INT_FWD_TIMER_B	Forwarded Timer B interrupt
DS1006_INT_FWD_HOST	Forwarded host interrupt
DS1006_INT_TIMER_D	Timer D interrupt

Note

Level-triggered interrupts (serial UART interrupts) have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value None

Related topics**References**

[ds1006_reset_interrupt_flag_bm..... 108](#)

ds1006_reset_interrupt_flag_bm

Syntax

```
void ds1006_reset_interrupt_flag_bm(UInt32 flag)
```

Include file

int1006.h

Purpose

To reset the interrupt flag for several interrupts.

Parameters

flag Specifies the bitmask of interrupts whose interrupt flag is to be reset. To specify more than one interrupt, you can combine the predefined symbols using the logical operator OR.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_INT_MASK_IOERR	PHS-bus I/O error interrupt
DS1006_INT_MASK_WD	Watchdog interrupt
DS1006_INT_MASK_PHS_0	PHS-bus interrupt line 0
...	...

Predefined Symbol	Meaning
DS1006_INT_MASK_PHS_7	PHS-bus interrupt line 7
DS1006_INT_MASK_TIMER_A	Timer A interrupt
DS1006_INT_MASK_TIMER_B	Timer B interrupt
DS1006_INT_MASK_GL_0	Gigalink 0 interrupt
...	...
DS1006_INT_MASK_GL_3	Gigalink 3 interrupt
DS1006_INT_MASK_SERIAL_UART	Serial UART interrupt
DS1006_INT_MASK_HOST	Host interrupt
DS1006_INT_MASK_MACROTICK	Macrotick interrupt
DS1006_INT_MASK_FWD_WD	Forwarded Watchdog interrupt
DS1006_INT_MASK_FWD_TIMER_A	Forwarded Timer A interrupt
DS1006_INT_MASK_FWD_TIMER_B	Forwarded Timer B interrupt
DS1006_INT_MASK_FWD_HOST	Forwarded Host interrupt
DS1006_INT_MASK_TIMER_D	Timer D interrupt

Note

Level-triggered interrupts (serial UART interrupts) have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value None

Related topics**References**

[ds1006_reset_interrupt_flag..... 107](#)

ds1006_set_interrupt_status

Syntax

```
ds1006_set_interrupt_status(UInt32 status)
```

Include file

```
int1006.h
```

Purpose

To set the interrupt status.

Description The value of the Interrupt-Enable bit (External Interrupt Enable) of the processor's EFLAGS register is restored.

Parameters **status** Returns the value of the previously executed function `ds1006_get_interrupt_status`.

Return value None

Related topics **References**

[ds1006_get_interrupt_status..... 105](#)

ds1006_set_interrupt_vector

Syntax

```
DS1006_Int_Handler_Type ds1006_set_interrupt_vector(
    UInt32 IntID,
    DS1006_Int_Handler_Type Handler,
    Int SaveRegs)
```

Include file `int1006.h`

Purpose To install an interrupt service routine for the selected interrupt.

Note

- Only `SaveRegs = SAVE_REGS_ON` is supported on the DS1006.
- Use `RTLIB_INT_ENABLE` to enable interrupts.
- The installation of interrupt service routines for the Timer A, Timer B, and Timer D interrupts is different from that of other interrupts. Refer to the example below and to [Timer Interrupt Control](#) on page 83.

Parameters **IntID** Identifies the interrupt that the handler is to be installed for. The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_INT_IOERR	PHS-bus I/O error interrupt
DS1006_INT_WD	Watchdog interrupt

Predefined Symbol	Meaning
DS1006_INT_PHS_0	PHS-bus interrupt line 0
...	...
DS1006_INT_PHS_7	PHS-bus interrupt line 7
DS1006_INT_TIMER_A	Timer A interrupt
DS1006_INT_TIMER_B	Timer B interrupt
DS1006_INT_GL_0	Gigalink 0 interrupt
...	...
DS1006_INT_GL_3	Gigalink 3 interrupt
DS1006_INT_SERIAL_UART	Serial UART interrupt
DS1006_INT_HOST	Host interrupt
DS1006_INT_MACROTICK	Macrotick interrupt
DS1006_INT_FWD_WD	Forwarded watchdog interrupt
DS1006_INT_FWD_TIMER_A	Forwarded Timer A interrupt
DS1006_INT_FWD_TIMER_B	Forwarded Timer B interrupt
DS1006_INT_FWD_HOST	Forwarded host interrupt
DS1006_INT_TIMER_D	Timer D interrupt

Note

Level-triggered interrupts (serial UART interrupts) have to be acknowledged in the interrupt service routine before they are enabled globally again.

Handler Specifies the pointer to the interrupt service routine.

SaveRegs Specifies the save mode for the registers needed for a C-coded interrupt handler. The following symbols are predefined:

Predefined Symbol	Meaning
SAVE_REGS_ON	Saves the relevant registers.
SAVE_REGS_OFF	Does not save the relevant registers – for advanced users only.

Return value

This function returns the address of the interrupt service routine that was previously installed for this interrupt.

Example

The Timer A interrupt is to call the function `timera_interrupt` (see also `ds1006_start_isr_timerA`).

First write the interrupt service routine `timera_interrupt`:

```
void timera_interrupt(void)
{
    ...
}
```

Then install the interrupt vector at the beginning of your application:

```
ds1006_set_interrupt_vector(
    DS1006_INT_TIMER_A,
    (DS1006_Int_Handler_Type) timera_interrupt,
    SAVE_REGS_ON);
```

Related topics

References

ds1006_set_interrupt_status	109
RTLIB_INT_ENABLE	113
Timer Interrupt Control	83

RTLIB_INT_DISABLE

Syntax

```
RTLIB_INT_DISABLE()
```

or

```
global_disable()
```

or

```
DS1006_GLOBAL_INTERRUPT_DISABLE()
```

Include file

`dsstd.h`

Purpose

To globally disable the interrupts.

Note

Use this macro only in conjunction with `RTLIB_INT_ENABLE`.

Tip

The macro `RTLIB_INT_DISABLE` should be used for time interval measurement in an application for different processor boards.

Return value	None
Related topics	References <div> ds1006_disable_hardware_int..... 97 ds1006_disable_hardware_int_bm..... 98 RTLIB_INT_ENABLE..... 113 </div>

RTLIB_INT_ENABLE

Syntax	<pre>RTLIB_INT_ENABLE()</pre> <p>or</p> <pre>global_enable()</pre> <p>or</p> <pre>DS1006_GLOBAL_INTERRUPT_ENABLE()</pre>
Include file	dsstd.h
Purpose	To globally enable the interrupts.
Description	<p>The only hardware interrupts that are available are the ones that are also enabled by ds1006_enable_hardware_int.</p> <div> <p>Note</p> <p>Use this macro only in conjunction with RTLIB_INT_DISABLE.</p> <p>Tip</p> <p>The macro RTLIB_INT_ENABLE should be used for time interval measurement in application for different processor boards.</p> </div>
Return value	None

Related topics**References**

ds1006_enable_hardware_int.....	99
ds1006_enable_hardware_int_bm.....	101
RTLIB_INT_DISABLE.....	112

RTLIB_INT_RESTORE

Syntax

```
void RTLIB_INT_RESTORE(UInt32 var_name)
```

Include file

```
dsstd.h
```

Purpose

To restore the previous interrupt state after calling **RTLIB_INT_SAVE_AND_DISABLE**.

Note

Use this macro only in conjunction with **RTLIB_INT_SAVE_AND_DISABLE**.

Parameters

var_name Returns the value of the previously executed macro **RTLIB_INT_SAVE_AND_DISABLE**.

Return value

None

Related topics**References**

ds1006_set_interrupt_status.....	109
RTLIB_INT_SAVE_AND_DISABLE.....	114

RTLIB_INT_SAVE_AND_DISABLE

Syntax

```
RTLIB_INT_SAVE_AND_DISABLE(UInt32 var_name)
```

Include file `dsstd.h`

Purpose To save the current interrupt status and globally disable the interrupts.

Note

Use this macro only in conjunction with `RTLIB_INT_RESTORE`.

Parameters `var_name` Specifies the variable to store the interrupt status.

Return value None

Example

```
void restore(void)
{
    UInt32 msr_state;

    RTLIB_INT_SAVE_AND_DISABLE(msr_state);
    /* Save the value of the EE bit in MSR and disable interrupts*/
    ...
    RTLIB_INT_RESTORE(msr_state);
    /* Restore the EE bit in MSR at the end of the function*/
}
```

Related topics

References

[ds1006_set_interrupt_status..... 109](#)
[RTLIB_INT_RESTORE..... 114](#)

RTLIB_SRT_DISABLE

Syntax `RTLIB_SRT_DISABLE()`

Include file `dsstd.h`

Purpose To disable the hardware interrupt for the sampling rate timer when the interrupts are still globally enabled (see `RTLIB_INT_ENABLE`).

Description	Timer A is used as the sampling rate timer for a DS1006 board. This function sets the corresponding bit of the Interrupt Mask Register (IMR).
Return value	None
Related topics	References <div> RTLIB_INT_ENABLE..... 113 RTLIB_SRT_ENABLE..... 116 </div>

RTLIB_SRT_ENABLE

Syntax	<code>RTLIB_SRT_ENABLE()</code>
Include file	<code>dsstd.h</code>
Purpose	To enable the hardware interrupt for the sampling rate timer.
Description	<p>Timer A is used as the sampling rate timer for a DS1006 board.</p> <p>This function only clears the corresponding bit of the Interrupt Mask Register (IMR). However, the hardware interrupt for Timer A is available only when the interrupts are globally enabled (see RTLIB_INT_ENABLE).</p>
Return value	None
Related topics	References <div> RTLIB_INT_ENABLE..... 113 RTLIB_SRT_DISABLE..... 115 </div>

Subinterrupt Handling

Introduction

Subinterrupt handling provides functions to extend one hardware interrupt to multiple software subinterrupts.

Where to go from here

Information in this section

Basic Principles of Subinterrupt Handling.....	118
Provides information on the subinterrupt handling principles.	
Example of Using a Subinterrupt Sender.....	118
Gives you instructions on implementing a subinterrupt sender.	
Example of Using a Subinterrupt Handler.....	119
Gives you instructions on implementing a subinterrupt handler.	
Example of Using a Subinterrupt Receiver.....	120
Gives you instructions on implementing a subinterrupt receiver.	
Data Types for Subinterrupt Handling.....	122
Provides the definition of the data types used by the subinterrupt module.	
dssint_define_int_sender.....	123
To define an interrupt sender.	
dssint_define_int_sender_1.....	125
To define an interrupt sender.	
dssint_define_int_receiver.....	127
To define an interrupt receiver.	
dssint_define_int_receiver_1.....	129
To define an interrupt receiver.	
dssint_subint_disable.....	131
To disable subinterrupts.	
dssint_subint_enable.....	132
To enable subinterrupts.	
dssint_interrupt.....	133
To trigger a subinterrupt.	
dssint_decode.....	133
To find out which subinterrupts are pending.	
dssint_acknowledge.....	134
To acknowledge pending subinterrupts.	
dssint_subint_reset.....	135
To clear pending subinterrupts.	

Basic Principles of Subinterrupt Handling

Introduction

In dSPACE multiprocessor systems, interrupts can be dispatched between processors. Typically, there is only one hardware line between processors. To allow multiple different interrupt signals to be sent from a sender to a receiver, a subinterrupt handling is provided which introduces logical interrupt sources. The subinterrupt handling meets the following goals:

- To trigger and handle multiple subinterrupts using a single hardware interrupt line.
- To allow that multiple different subinterrupts are pending at the receiver.
- To transmit and dispatch interrupts between several processors.
- To define interrupt senders/receivers to transmit subinterrupts.
- To use multiple senders and receivers at one processor.
- To get a point-to-point interrupt connection between two processors using a combination of sender and receiver.
- To make priority-based interrupt arbitration available (optional).
- Subinterrupts stay pending if they are disabled at the moment they occur.

Method

The following steps are necessary to program a subinterrupt handling between two applications:

- 1 Install a subinterrupt sender in your application that sends an interrupt.
- 2 Write an interrupt handler in your application that receives the interrupt.
- 3 Install a subinterrupt receiver in your application that receives the interrupt.

Example

See the following examples for more information:

- [Example of Using a Subinterrupt Sender](#) on page 118
- [Example of Using a Subinterrupt Handler](#) on page 119
- [Example of Using a Subinterrupt Receiver](#) on page 120

Example of Using a Subinterrupt Sender

Example

The following example shows the source code for the interrupt sender. It is defined for 16 subinterrupts. Every time the background loop is interrupted by timer 0, the subinterrupt 3 is sent to the receiver. The dual-port memory width is 16 bit and the accesses are direct.

```
#include <Brtenv.h>
#include <Defxxx.h>      /* xxxx stands for the dSPACE */
#include <Mydefs.h>      /* board, e.g., 1401 for DS1401 */
dssint_sender_type *sender;
```

```

void isr_t0()
{
    dssint_interrupt(sender, 3);
}
void main()
{
    sender = dssint_define_int_sender_1(
        16,                /* number of subinterrupts */
        SUBINT_ADDR,       /* start address of int. info */
        ACK_ADDR,          /* start address of ack. info */
        SENDER_ADDR,       /* trigger address */
        DPM_TARGET_DIRECT, /* e.g., PHS bus base address */
        16,                /* dual-port memory width */
        DPM_ACCESS_DIRECT, /* pointer to write function */
        DPM_ACCESS_DIRECT); /* pointer to read function */
    /* ... initialize timer 0 ... */
    global_enable();
    while(1);
}

```

Related topics

Basics

[Basic Principles of Subinterrupt Handling.....](#) 118

Examples

[Example of Using a Subinterrupt Handler.....](#) 119

[Example of Using a Subinterrupt Receiver.....](#) 120

Example of Using a Subinterrupt Handler

Example

The example shows an interrupt handler for the dSPACE real-time kernel.

When the interrupt is triggered, the processor dispatches it to **my_handler**, where it is acknowledged by calling **dssint_acknowledge**. The function **dssint_decode** is called repetitively and returns the according subinterrupt number for every pending subinterrupt. For every subinterrupt, one task is registered by calling **rtk_register_task**.

rtk_register_task sets the task state for the according task to 'ready' when the task priority is not the highest of all registered tasks. The function internally stores the task registered with the highest priority and returns a pointer to it. **rtk_register_task** does not schedule tasks.

Once all tasks are registered, the "task" pointer holds the one with the highest priority. This task can be of a lower, equal or higher priority than the currently running task. Via the "task" pointer the scheduler is called – this is the reason

why the state of the task registered with the highest priority must not be set to 'ready'.

The scheduler clears the stored information about the task registered with the highest priority.

```
void my_handler()
{
    rtk_p_task_control_block task = 0;
    int sub_int;
    dssint_acknowledge(receiver); /* interrupt acknowledge */
    /* Register tasks */
    do {
        if ( (sub_int = dssint_decode(receiver)) >= 0)
            task = rtk_register_task(S_MYSERVICE, sub_int);
    } while(sub_int >= 0);
    /* Call the scheduler */
    if (task)
        rtk_scheduler(task);
}
```

Related topics

Basics

[Basic Principles of Subinterrupt Handling..... 118](#)

Examples

[Example of Using a Subinterrupt Receiver..... 120](#)
[Example of Using a Subinterrupt Sender..... 118](#)

Example of Using a Subinterrupt Receiver

Example

In this example, a receiver with 16 subinterrupts is defined. It is assumed that the kernel installs the function `my_handler` (refer to the [Example of Using a Subinterrupt Handler](#) on page 119) as an interrupt service routine for subinterrupts. The `main` function enables interrupts and enters the background task after creating and binding the tasks to the subinterrupts.

```
#include <Brtenv.h>
#include <Defxxxx.h> /* xxxx stands for the dSPACE */
/* board, e.g. 1401 for DS1401 */

void slave0_task(void)
{
    /*...*/
};
dssint_receiver_type receiver;
```



```

void main()
{
    rtk_p_task_control_block task;
    receiver = dssint_define_int_receiver_1(
        16,                /* number of subinterrupts */
        SUBINT_ADDR,       /* start address of int. info */
        ACK_ADDR,          /* start address of ack. info */
        RECEIVER_ADDR,     /* receiver address */
        DPM_TARGET_DIRECT, /* e.g. PHS bus base address */
        16,                /* dual-port memory width */
        DPM_ACCESS_DIRECT, /* pointer to write function */
        DPM_ACCESS_DIRECT); /* pointer to read function */
    /* ... */
    task = rtk_create_task((rtk_task_fcn_type)slave0_task, 1,
        ovc_queue, rtk_default_overrun_fcn, 10, 0);
    rtk_bind_interrupt(S_SLAVE, 0, task, 0.0, C_LOCAL, 0, 0);
    /*...*/
    global_enable();
    while(1);
}

```

Related topics

Basics

[Basic Principles of Subinterrupt Handling..... 118](#)

Examples

[Example of Using a Subinterrupt Handler..... 119](#)
[Example of Using a Subinterrupt Sender..... 118](#)

Data Types for Subinterrupt Handling

dssint_sender_type

```
typedef struct{
    unsigned int    nr_sint;    /* number of subinterrupts */
    unsigned long   sint_addr; /* start address of the */
                                /* interrupt info */
    unsigned long   ack_addr;  /* start address of the */
                                /* acknowledge info */
    unsigned long   sender_addr; /* writing to this address */
                                /* triggers interrupt */
    unsigned int    nr_words;  /* number of words */
                                /* needed for nr_sint */
    unsigned long*   request;  /* pointer to local copy */
                                /* of sint_addr */
    long            target;    /* e.g. PHS bus base address */
    unsigned int     sint_mem_width;
                                /* width of the */
                                /* dual-port memory */
    dpm_write_fcn_t write_fcn; /* pointer to write function */
    dpm_read_fcn_t  read_fcn;  /* pointer to read function */
    unsigned int     sint_mem_shift;
                                /* internal performance */
                                /* improvement */
}dssint_sender_type;
```

dssint_receiver_type

```
typedef struct{
    unsigned int    nr_sint;    /* number of subinterrupts */
    unsigned long   sint_addr; /* start address of the */
                                /* interrupt info */
    unsigned long   ack_addr;  /* start address of the */
                                /* acknowledge info */
    unsigned long   receiver_addr;
                                /* reading from this address */
                                /* performs hardware ack of */
                                /* interrupt */
    unsigned int    nr_words;  /* number of words */
                                /* needed for nr_sint */
    unsigned long*   acknowledge;
                                /* pointer to local copy */
                                /* of ack_addr */
    unsigned long*   state;    /* pointer to state info */
    long            target;    /* e.g. PHS bus base address */
    unsigned int     sint_mem_width;
                                /* width of the */
                                /* dual-port memory */
    unsigned int     state_position;
                                /* decode position in state */
    dpm_write_fcn_t write_fcn; /* pointer to write function */
    dpm_read_fcn_t  read_fcn;  /* pointer to read function */
    unsigned int     sint_mem_shift;
                                /* internal performance */
                                /* improvement */
    unsigned long*   enable_flag; /* for pending interrupts */
    dssint_ack_fcn_t ack_fcn;    /* pointer to interrupt acknowledge function */
}dssint_receiver_type;
```

Related topics

Basics

Basic Principles of Subinterrupt Handling..... 118

Examples

Example of Using a Subinterrupt Handler..... 119
Example of Using a Subinterrupt Receiver..... 120
Example of Using a Subinterrupt Sender..... 118

dssint_define_int_sender

Syntax

```
dssint_sender_type* dssint_define_int_sender(  
    unsigned int nr_subinterrupts,  
    unsigned long subint_addr,  
    unsigned long ack_addr,  
    unsigned long sender_addr,  
    long target,  
    unsigned int sint_mem_width,  
    dpm_write_fcn_t write_fcn,  
    dpm_read_fcn_t read_fcn)
```

Include file

dssint.h

Purpose

To define the sender of a subinterrupt.

Description

The function defines an interrupt sender and returns a handle to it. A sender processor can have multiple receiver processors to pass interrupts to. The handle identifies where to send an interrupt. The function initializes all memory locations in the dual-port memory used for the subinterrupt handling with 0.

The functions `dssint_define_int_sender` and `dssint_define_int_receiver` define the sender and receiver of a subinterrupt in the following way:

When subinterrupts are sent before the receiver is initialized, these interrupts are stored. After the receiver is initialized these interrupts are passed to the receiver and processed.

Note

- The behavior described above can cause overflows. To avoid this, use the functions `dssint_define_int_sender_1` and `dssint_define_int_receiver_1` instead.
- If you define a sender of a subinterrupt via the function `dssint_define_int_sender`, you must define the receiver via the function `dssint_define_int_receiver`.

Parameters

nr_subinterrupts Specifies the number of different subinterrupts to be transferred. This is necessary to define the width of the memory portion which passes the subinterrupt information. The number of subinterrupts must be equal for sender and receiver.

subint_addr Specifies the memory location the subinterrupt information is passed to.

ack_addr Specifies the memory location the acknowledgment information from the receiver is passed to.

sender_addr Specifies the pointers to the memory location that triggers the interrupt by writing to it (hardware trigger). This address can be the same as `subint_addr`.

target Specifies the address of the target memory, for example, a PHS bus address or COM port number. This parameter is meaningless for direct access.

sint_mem_width Specifies the width of the dual-port memory.

write_fcn Specifies the address of a function that performs a write access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

read_fcn Specifies the address of a function that performs a read access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

Return value

This function returns the handle to an interrupt sender. The function returns 0 if an error occurred.

Example

See [Example of Using a Subinterrupt Sender](#) on page 118.

Related topics

Basics	
Basic Principles of Subinterrupt Handling.....	118
Examples	
Example of Using a Subinterrupt Sender.....	118
References	
dssint_define_int_receiver.....	127
dssint_define_int_receiver_1.....	129
dssint_define_int_sender_1.....	125

dssint_define_int_sender_1

Syntax	<pre>dssint_sender_type* dssint_define_int_sender_1(unsigned int nr_subinterrupts, unsigned long subint_addr, unsigned long ack_addr, unsigned long sender_addr, long target, unsigned int sint_mem_width, dpm_write_fcn_t write_fcn, dpm_read_fcn_t read_fcn)</pre>
--------	---

Include file	dssint.h
--------------	----------

Purpose	To define the sender of a subinterrupt.
---------	---

Description	<p>The function defines an interrupt sender and returns a handle to it. A sender processor can have multiple receiver processors to pass interrupts to. The handle identifies where to send an interrupt. The function initializes all memory locations in the dual-port memory used for the subinterrupt handling with 0.</p> <p>The functions <code>dssint_define_int_sender_1</code> and <code>dssint_define_int_receiver_1</code> define the sender and receiver of a subinterrupt in the following way:</p>
-------------	--

When subinterrupts are sent before the receiver is initialized, these interrupts are not stored to avoid overflows.

Note

If you define a sender of a subinterrupt via the function `dssint_define_int_sender_1`, you have to define the receiver via the function `dssint_define_int_receiver_1`.

Parameters

nr_subinterrupts Specifies the number of different subinterrupts to be transferred. The number of subinterrupts must be equal for sender and receiver. See [dssint_define_int_sender](#) on page 123.

subint_addr Specifies the memory location the subinterrupt information is passed to.

ack_addr Specifies the memory location the acknowledgment information from the receiver is passed to.

sender_addr Specifies the pointers to the memory location that triggers the interrupt by writing to it (hardware trigger). This address can be the same as `subint_addr`.

target Specifies the address of the target memory, for example, a PHS bus address or COM port number. This parameter is meaningless for direct access.

sint_mem_width Specifies the width of the dual-port memory.

write_fcn Specifies the address of a function that performs a write access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

read_fcn Specifies the address of a function that performs a read access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

Return value

This function returns the handle to an interrupt sender. The function returns 0 if an error occurred.

Example

See [Example of Using a Subinterrupt Sender](#) on page 118.

Related topics

Basics	
Basic Principles of Subinterrupt Handling.....	118
Examples	
Example of Using a Subinterrupt Sender.....	118
References	
dssint_define_int_receiver_1.....	129
dssint_define_int_sender.....	123

dssint_define_int_receiver

Syntax

```
dssint_receiver_type *dssint_define_int_receiver(
    unsigned int nr_subinterrupts,
    unsigned long subint_addr,
    unsigned long ack_addr,
    unsigned long receiver_addr,
    long target,
    unsigned int sint_mem_width,
    dpm_write_fcn_t write_fcn,
    dpm_read_fcn_t read_fcn)
```

Include file dssint.h

Purpose To define the receiver of a subinterrupt.

Description

The function reads from the **receiver_addr** to enable interrupt triggering by the sender. It defines an interrupt receiver and returns a handle to it. A receiver processor can have multiple sender processors from which interrupts are retrieved. The handle identifies the appropriate subinterrupt vector and receiving information table for a specific sender.

The functions **dssint_define_int_receiver** and **dssint_define_int_sender** define the receiver and sender of a subinterrupt in the following way:

When subinterrupts are sent before the receiver is initialized, these interrupts are stored. After the receiver is initialized, these interrupts are passed to the receiver and processed.

Note

- The behavior described above can cause overflows. To avoid this, use the functions `dssint_define_int_sender_1` and `dssint_define_int_receiver_1` instead.
- If you define a receiver of a subinterrupt via the function `dssint_define_int_receiver`, you have to define the sender via the function `dssint_define_int_sender`.

Parameters

nr_subinterrupts Specifies the number of different subinterrupts to be transferred. The number of subinterrupts must be equal for sender and receiver. See [dssint_define_int_sender](#) on page 123.

subint_addr Specifies the memory location the subinterrupt information is passed to.

ack_addr Specifies the memory location the acknowledgment information from the receiver is passed to.

receiver_addr Specifies the pointers to the memory location that acknowledges the interrupt by reading it (hardware acknowledge).

target Specifies the address of the target memory, for example, a PHS bus address or COM port number. This parameter is meaningless for direct access.

sint_mem_width Specifies the width of the dual-port memory.

write_fcn Specifies the address of a function that performs a write access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

read_fcn Specifies the address of a function that performs a read access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

Return value

This function returns the address of an interrupt receiver. The function returns 0 if an error occurred.

Example

See [Example of Using a Subinterrupt Receiver](#) on page 120.

Related topics

Basics	
Basic Principles of Subinterrupt Handling.....	118
Examples	
Example of Using a Subinterrupt Receiver.....	120
References	
dssint_define_int_receiver_1.....	129
dssint_define_int_sender.....	123
dssint_define_int_sender_1.....	125

dssint_define_int_receiver_1

Syntax

```
dssint_receiver_type *dssint_define_int_receiver_1(
    unsigned int nr_subinterrupts,
    unsigned long subint_addr,
    unsigned long ack_addr,
    unsigned long receiver_addr,
    long target,
    unsigned int sint_mem_width,
    dpm_write_fcn_t write_fcn,
    dpm_read_fcn_t read_fcn)
```

Include file

dssint.h

Purpose

To define the receiver of a subinterrupt.

Description

The function reads from the **receiver_addr** to enable interrupt triggering by the sender. It defines an interrupt receiver and returns a handle to it. A receiver processor can have multiple sender processors from which interrupts are retrieved. The handle identifies the appropriate subinterrupt vector and receiving information table for a specific sender.

The functions **dssint_define_int_receiver_1** and **dssint_define_int_sender_1** define the receiver and sender of a subinterrupt in the following way:

When subinterrupts are sent before the receiver is initialized, these interrupts will not be stored to avoid overflows.

Note

If you define a receiver of a subinterrupt via the function `dssint_define_int_receiver_1`, you must define the sender via the function `dssint_define_int_sender_1`.

Parameters

nr_subinterrupts Specifies the number of different subinterrupts to be transferred. The number of subinterrupts must be equal for sender and receiver. See [dssint_define_int_sender](#) on page 123.

subint_addr Specifies the memory location the subinterrupt information is passed to.

ack_addr Specifies the memory location the acknowledgment information from the receiver is passed to.

receiver_addr Specifies the pointers to the memory location that acknowledges the interrupt by reading it (hardware acknowledge).

target Specifies the address of the target memory, for example, a PHS bus address or COM port number. This parameter is meaningless for direct access.

sint_mem_width Specifies the width of the dual-port memory.

write_fcn Specifies the address of a function that performs a write access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

read_fcn Specifies the address of a function that performs a read access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

Return value

This function returns the address of an interrupt receiver. The function returns 0 if an error occurred.

Example

See [Example of Using a Subinterrupt Receiver](#) on page 120.

Related topics**Basics**

[Basic Principles of Subinterrupt Handling..... 118](#)

Examples

[Example of Using a Subinterrupt Receiver..... 120](#)

References

[dssint_define_int_receiver..... 127](#)

[dssint_define_int_sender_1..... 125](#)

dssint_subint_disable

Syntax

```
void dssint_subint_disable(
    dssint_receiver_type *receiver,
    unsigned int subinterrupt)
```

Include file

dssint.h

Purpose

To disable a subinterrupt.

Description

After initialization, all subinterrupts are enabled. You must disable the subinterrupt explicitly via this function.

Parameters

receiver Specifies the receiver handler the subinterrupt is located in.

subinterrupt Specifies the subinterrupt to reset.

Example

```
...
dssint_subint_disable(my_receiver, 5);
...
```

Related topics**Basics**

[Basic Principles of Subinterrupt Handling..... 118](#)

References

[dssint_subint_enable..... 132](#)
[dssint_subint_reset..... 135](#)

dssint_subint_enable

Syntax

```
void dssint_subint_enable(
    dssint_receiver_type *receiver,
    unsigned int subinterrupt)
```

Include file

dssint.h

Purpose

To enable a subinterrupt.

Description

After initialization, all subinterrupts are enabled. Use this function if you disabled a subinterrupt via `dssint_subint_disable` before.

Parameters

receiver Specifies the receiver handler the subinterrupt is located in.
subinterrupt Specifies the subinterrupt to reset.

Example

```
...
dssint_subint_enable(my_receiver, 5);
...
```

Related topics**Basics**

[Basic Principles of Subinterrupt Handling..... 118](#)

References

[dssint_subint_disable..... 131](#)
[dssint_subint_reset..... 135](#)

dssint_interrupt

Syntax

```
void dssint_interrupt(  
    dssint_sender_type *sender,  
    unsigned int sub_interrupt)
```

Include file `dssint.h`

Purpose To write the subinterrupt information to the specified memory location and to trigger the interrupt.

Parameters

sender Specifies the handle of the interrupt sender.

sub_interrupt Specifies the subinterrupt to be triggered. Values are within the range 0 ... nr_subinterrupts. Parameter nr_subinterrupts is defined by dssint_define_int_sender (or dssint_define_int_sender_1) and dssint_define_int_receiver (or dssint_define_int_receiver_1).

Example See [Example of Using a Subinterrupt Sender](#) on page 118.

Related topics

Basics	
Basic Principles of Subinterrupt Handling.....	118
Examples	
Example of Using a Subinterrupt Handler.....	119
References	
dssint_define_int_receiver.....	127
dssint_define_int_receiver_1.....	129
dssint_define_int_sender.....	123
dssint_define_int_sender_1.....	125

dssint_decode

Syntax

```
int dssint_decode(dssint_receiver_type *receiver)
```

Include file	<code>dssint.h</code>
Purpose	To identify the pending interrupts.
Description	This function is called repetitively within an interrupt handler. It processes the interrupt information of the receiver data structure that was given by <code>dssint_acknowledge</code> , determines the pending subinterrupt with the highest priority and returns it to the handler. The pending subinterrupt with the highest priority is the one with the smallest subinterrupt number.
Parameters	receiver Specifies the receiver handler the subinterrupt is located in.
Return value	This function returns the number of the pending subinterrupt with highest priority. If there is no pending subinterrupt left, the function returns <code>SINT_NO_SUBINT ("-1")</code> .
Example	See Example of Using a Subinterrupt Handler on page 119.
Related topics	<div>Basics</div> <div>Basic Principles of Subinterrupt Handling..... 118</div> <div>Examples</div> <div>Example of Using a Subinterrupt Handler..... 119</div> <div>References</div> <div><code>dssint_acknowledge</code>..... 134</div>

dssint_acknowledge

Syntax	<code>void dssint_acknowledge(dssint_receiver_type *receiver)</code>
Include file	<code>dssint.h</code>

Purpose	To acknowledge pending subinterrupts.
Description	<p>This function acknowledges the interrupt by reading <code>receiver->receiver_addr</code> (hardware acknowledge), and copies the subinterrupt information to the receiver data structure. Then it performs the software acknowledgment for every pending subinterrupt.</p> <p>For information on the receiver data structure, refer to the type definition given in Data Types for Subinterrupt Handling on page 122.</p>
Parameters	receiver Specifies the receiver handler the subinterrupt is located in.
Example	See Example of Using a Subinterrupt Handler on page 119.
Related topics	<p>Basics</p> <p>Basic Principles of Subinterrupt Handling..... 118</p> <p>Examples</p> <p>Example of Using a Subinterrupt Handler..... 119</p> <p>References</p> <p>Data Types for Subinterrupt Handling..... 122</p> <p>dssint_define_int_receiver..... 127</p> <p>dssint_define_int_receiver_1..... 129</p>

dssint_subint_reset

Syntax	<pre>void dssint_subint_reset(dssint_receiver_type *receiver, unsigned int subinterrupt)</pre>
Include file	<code>dssint.h</code>
Purpose	To clear a pending subinterrupt.

Parameters	<div><div>receiver Specifies the receiver handler the subinterrupt is located in.</div><div>subinterrupt Specifies the subinterrupt to reset.</div></div>
Example	<div><pre>... dssint_subint_reset(my_receiver, 5); ...</pre></div>
Related topics	<div><div>Basics</div><div><div>Basic Principles of Subinterrupt Handling..... 118</div></div></div> <div><div>References</div><div><div>Data Types for Subinterrupt Handling..... 122</div><div>dssint_subint_disable..... 131</div><div>dssint_subint_enable..... 132</div></div></div>

Exception Handling

Basics on Exception Handling

There are some exceptions in the execution of the terminating program, such as program errors, alignment errors, access errors, etc. If one of these exceptions occurs, the exception flag is set, status information is written to the global memory, a message may be generated, and the program terminates.

To get detailed information on program errors, you can debug your application. For further information, refer to [Debugging an Application](#) on page 362.

Exception flags

The following exception flags are predefined:

Predefined Symbol	Meaning
CFG_EXC_NO_EXCEPT	No exception
CFG_EXC_DEVIDE_ERROR	Division by zero exception
CFG_EXC_DEBUG	Data/instruction breakpoint exception
CFG_EXC_MNI	Non-maskable int exception
CFG_EXC_BREAKPOINT	Debugger breakpoint / INT 3
CFG_EXC_OVERFLOW	Overflow exception caused INTO
CFG_EXC_BOUND	Bound exception caused by BOUND
CFG_EXC_INVALID_OPCODE	Invalid opcode exception
CFG_EXC_DEV_NOT_AVAIL	FPU or SSE unit not available exception
CFG_EXC_DOUBLE_FAULT	Exception while handling an exception
CFG_EXC_FPU_OVERRUN	FPU segment overrun (reserved)
CFG_EXC_INVALID_TSS	Invalid Task Switch segment exception
CFG_EXC_SEGMENT_NOT_PRESENT	Segment not present exception
CFG_EXC_STACK_FAULT	Stack fault exception
CFG_EXC_PROTECTION_FAULT	General protection fault exception
CFG_EXC_PAGE_FAULT	Page fault exception
CFG_EXC_FP_ARITHMETICAL	FP arithmetical exception
CFG_EXC_ALIGN_ERR	Alignment error
CFG_EXC_MACHINE_CHECK	Machine check
CFG_EXC_SSE_UNAVAIL	SSE floating-point exception
CFG_EXC_ILLEGAL_INT_ERR	An unexpected interrupt occurred

Note

Arithmetic exceptions are masked by default. The processor does not generate an exception, but continues the execution of the application with possibly wrong values.

Page fault exceptions

The DS1006 implements memory protection for certain address ranges. If a real-time application accesses a protected area, a page fault exception is generated. The program execution is stopped immediately and a message indicating the accessed address range is displayed. The protected memory area are the first 4 MB of the global memory (0x0 to 0x00400000).

Information Handling

Introduction Use these functions to get information on the board version, the memory configuration, the clock frequency and the CPU temperature.

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

ds1006_info_board_version_get.....	139
ds1006_info_memory_get.....	140
ds1006_info_clocks_get.....	140
ds1006_info_clocks_khz_get.....	141
ds1006_info_cpu_temperature_get.....	141

ds1006_info_board_version_get

Syntax

```
void ds1006_info_board_version_get(
    UInt32 *version,
    UInt32 *revision,
    UInt32 *sub_version)
```

Include file `info1006.h`

Purpose To get the board version.

Parameters

version	Specifies the pointer to the variable containing the board version.
revision	Specifies the pointer to the variable containing the board revision.
sub_version	Specifies the pointer to the variable containing the board subversion.

Return value None

ds1006_info_memory_get

Syntax

```
void ds1006_info_memory_get(
    UInt32 *memory_size,
    UInt32 *cached_memory_base,
    UInt32 *noncached_memory_base,
    UInt32 *flash_size,
    UInt32 *flash_base)
```

Include file

info1006.h

Purpose

To get the sizes and the base addresses of the global main memory and flash memory from the config section.

Parameters

memory_size Specifies the pointer to the variable containing the size of the global memory in bytes.

cached_memory_base Returns the pointer to the variable containing the base address of the memory area. This memory area is not cached.

The parameter is only available for compatibility with the obsolete DS1005.

noncached_memory_base Returns the pointer to the variable containing the base address of the memory area.

The parameter is only available for compatibility with the obsolete DS1005.

flash_size Always returns 0 as the CompactFlash cannot be accessed directly on the DS1006.

flash_base Always returns 0 as the CompactFlash cannot be accessed directly on the DS1006.

ds1006_info_clocks_get

Syntax

```
void ds1006_info_clocks_get(
    UInt32 *cpu_clock,
    UInt32 *bus_clock)
```

Include file

info1006.h

Purpose

To get frequency information from the config section.

Note

Do not use this function for the DS1006. It will return 0 for both clock frequencies. Use `ds1006_info_clocks_khz_get` instead.

Parameters

cpu_clock Specifies the pointer to the variable containing the frequency of the CPU clock in Hz.

bus_clock Specifies the pointer to the variable containing the frequency of the bus clock in Hz.

ds1006_info_clocks_khz_get

Syntax

```
void ds1006_info_clocks_khz_get(
    UInt32 *cpu_clock,
    UInt32 *bus_clock)
```

Include file

info1006.h

Purpose

To get frequency information from the config section.

Parameters

cpu_clock Specifies the pointer to the variable containing the frequency of the CPU clock in kHz.

bus_clock Specifies the pointer to the variable containing the frequency of the bus clock in kHz.

ds1006_info_cpu_temperature_get

Syntax

```
Int8 ds1006_info_cpu_temperature_get(void)
```

Include file

info1006.h

Purpose	To get the current CPU temperature.
----------------	-------------------------------------

Description	The corresponding value in the config section is also updated.
--------------------	--

Note

- The time between two calls of this function must be at least 5 ms, otherwise the values are not updated correctly.
- With the multicore DS1006 board, the temperature is correctly measured only on the first CPU core (Core 0). The other cores return 0°C.

Return value	This function returns the temperature of the CPU in °C.
---------------------	---

Version and Config Section Management

Introduction

The Version and Config Section Management (VCM) module is used to manage information required for registering a board and displaying its properties in the experiment software.

Where to go from here

Information in this section

Basic Principles of VCM.....	145
Provides basic information on the Version and Config Section Management module.	
Data Types for VCM.....	148
Provides information on the data types used for version and config section management.	
vcm_init.....	149
To initialize the Version and Config Section Management module.	
vcm_module_register.....	149
To register a software module in the VCM module and to return a pointer to the module descriptor.	
vcm_cfg_malloc.....	151
To allocate a block of the specified size in the config section memory.	
vcm_memory_ptr_set.....	152
To set the pointer and the size of a config section memory block that is associated with the module.	
vcm_memory_ptr_get.....	152
To get the pointer to the config section memory block that is associated with the module.	
vcm_module_find.....	153
To find a pointer to the module descriptor by a given module ID.	
vcm_module_status_set.....	154
To set the status of a software module.	
vcm_module_status_get.....	155
To get the status of a given module.	
vcm_version_get.....	156
To get the version of a module.	
vcm_version_compare.....	157
To compare the version of a module with a given version.	
vcm_module_version_print.....	158
To print the module version into a char buffer.	
vcm_version_print.....	159
To print given version information into a char buffer.	

Basic Principles of VCM

Introduction

The Version and Config Section Management (VCM) module meets the following goals:

- Managing module versions
- Tracking the status of a module
- Managing the config section memory

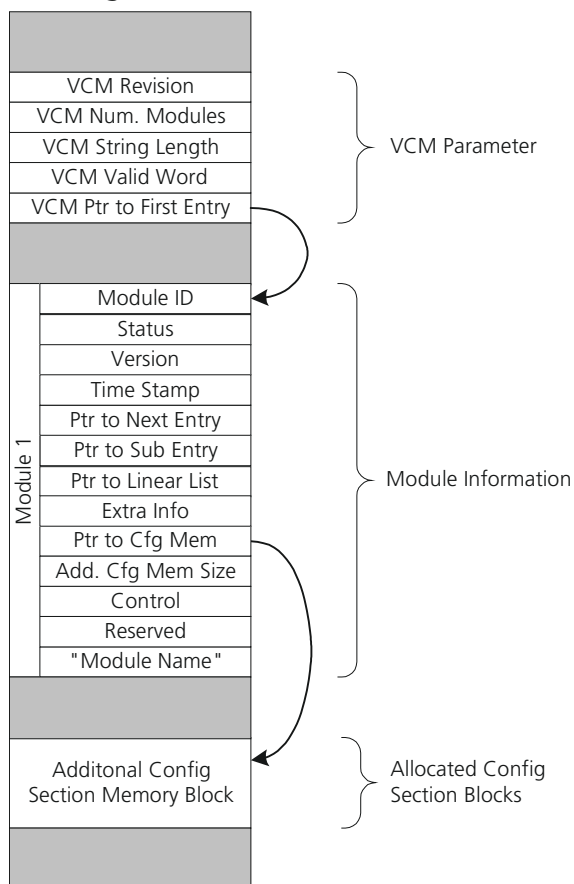
The data structures for version and config section management are located in the global memory of each processor board and can therefore be accessed by the real-time hardware and the host PC. Module version and status information is displayed by the dSPACE experiment software on the property page of processor boards. Right-click at the board, select **Properties...** and click the **Versions** tab. All currently registered modules are shown.

Real-time libraries and applications use the VCM module to register software modules. Upon registering a module a pointer to the module descriptor is returned. This pointer can also be found with the aid of the defined module ID. This pointer is required to read module parameters like the module status or the module version number.

After registering a module, a portion of the config section memory can be requested and associated with the module. Memory can only be taken, the VCM module cannot free memory blocks.

VCM config section data structures

The illustration below shows how the VCM data is structured.

Config Section

There are three different memory areas:

VCM parameters These are the parameters for the VCM module. This portion has a fixed address in the config section. The parameters are:

VCM Parameter	Meaning
VCM Revision	VCM module's revision number
VCM Num. Modules	Number of registered modules
VCM String Length	Maximum length of the "module name" for each module
VCM Valid Word	Keyword that shows that the VCM entries are valid (= VCM_VALID_WORD)
VCM Ptr to First Entry	Pointer to the module information section

Module information The memory for each module descriptor is allocated dynamically. Each module descriptor contains the following fields:

Module Information	Meaning
Module ID	Module ID
Status	Module status (initialized, error code, ...)
Version	Module version (major, minor, maintenance, special build type, special build number, patch level)
Time Stamp	Time stamp (set automatically, currently not implemented)
Ptr to Next Entry	Pointer to next module information block (offset from beginning of config section)
Ptr to Sub Entry	Pointer to a submodule information block, or 0 if there is no submodule (offset from beginning of config section)
Ptr to Linear List	Pointer to a linear module list to avoid recursive search in tree structure on RTP (offset from beginning of config section)
Extra Info	Extra module specific information If module-specific data is ≤ 32 bit no additional config section memory block is required
Ptr to Cfg Mem	Pointer to additional config section memory block (may be 0) (offset from beginning of config section)
Add. Cfg Mem Size	Size of additional config section memory block in sizeof (char)
Control	Special control bits, like VCM_CTRL_HIDDEN
Reserved	32 reserved bits
"Module Name"	Module name as string

Allocated config section blocks Blocks of config section memory. This block can be specific to each module.

Module IDs

Modules are identified by IDs. IDs from 1 to 999 are reserved for user modules. IDs higher than 999 are used for dSPACE modules.

IDs are defined in the header file `dsmodule.h`. This header file also contains the data types for module-specific data like the "extra info" field or the additional config section memory data block.

Modules that are always present are registered by the boot firmware or RTLib. Application modules are registered by using C API functions. Application examples can be:

- RTLib
- Available I/O board scanner
- Comport connection scanner
- I/O board modules
- Message module
- RTI
- S-functions

The VCM data structure is set up in the boot firmware or in the init function. Hence, it is available from the first line of application code after the init function.

Data types

For a definition of data types, refer to the section [Data Types for VCM](#) on page 148.

Data Types for VCM

vcm_version_type

```
typedef union
{
    struct { UInt32 high; UInt32 low; } version;
    struct
    {
        } vs;
    } vcm_version_type;
```

vcm_module_descriptor_type

```
typedef struct vcm_module_descriptor_struct
{
    Int32          vcm_mod_id;
    Int32          vcm_status;
    vcm_version_type vcm_version;
    timestamp_type vcm_timestamp;
    Int32          vcm_next_offs;
    Int32          vcm_sub_offs;
    Int32          vcm_lin_offs;
    Int32          xtra_info;
    Int32          vcm_cfg_mem_offs;
    UInt32         vcm_cfg_mem_size;
    UInt32         vcm_control;
    Int32          vcm_reserved_offs;
    char           vcm_module_name[VCM_MAX_NAME_LENGTH];
} vcm_module_descriptor_type;
```

vcm_size_t

```
typedef UInt32 vcm_size_t;
```

vcm_cfg_mem_ptr_type

```
typedef void* vcm_cfg_mem_ptr_type;
```

Related topics**Basics**

[Basic Principles of VCM.....](#) 145

vcm_init

Syntax

```
void vcm_init(void)
```

Include file

dsvcm.h

Purpose

To initialize the Version and Config Section Management module.

Note

This function is called in the boot firmware or in the `init` function and must not be called by the user.

Related topics

Basics

[Basic Principles of VCM..... 145](#)

References

[ds1006_init..... 18](#)

vcm_module_register

Syntax

```
vcm_module_descriptor_type* vcm_module_register(
    UInt32 mod_id,
    vcm_module_descriptor_type *main_ptr,
    char* module_name,
    UInt8 mar,
    UInt8 mir,
    UInt8 mai,
    UInt8 spb,
    UInt16 spn,
    UInt16 plv,
    UInt32 xtra_info,
    UInt32 control);
```

Include file

dsvcm.h

Purpose To register a software module in the VCM module and to return a pointer to the module descriptor.

Result The module status of newly registered modules is set to 'uninitialized' (VCM_STATUS_UNINITIALIZED).

Parameters **mod_id** Specifies the module ID within the range of 1 ... 999. Module IDs higher than 999 are used for dSPACE modules.

main_ptr If this module is a submodule: this is a pointer to the superior module (superior module must be registered before). If this module is not a submodule, this is 0.

module_name Specifies the module description string.

mar Specifies the major release.

mir Specifies the minor release.

mai Specifies the maintenance number.

spb Specifies the special build type.

spn Specifies the special build number.

plv Specifies the patch level.

xtra_info Specifies the module-specific data.

control Specifies the special control bits. The following symbols are predefined:

Predefined Symbol	Meaning
VCM_CTRL_HIDDEN	Hide the module from the host PC
VCM_CTRL_NO_VS	Hide the version number from the host PC
VCM_CTRL_NO_ST	Hide the status from the host PC
VCM_CTRL_NAME_ONLY	Display only the module name on the host PC

Return value This function returns the pointer to a module descriptor, or 0 if registration failed.

Example

```
#include <dsvcm.h>
#define VCM_MID_MY_USR_SW 0x5801
/* Possible values: 0x5800 ... 0x5FFF */
#define VCM_TXT_MY_USR_SW "My User Software"
vcm_module_descriptor_type* msg_mod_ptr;
```

```
/* Register the message module */
msg_mod_ptr = vcm_module_register(VCM_MID_MY_USR_SW,
                                  (void*)0,
                                  VCM_TXT_MY_USR_SW,
                                  1,
                                  0,
                                  0,
                                  VCM_VERSION_RELEASE,
                                  0,
                                  0,
                                  0,
                                  0);
```

Related topics

Basics	
Basic Principles of VCM.....	145
References	
vcm_memory_ptr_set.....	152
vcm_module_status_set.....	154

vcm_cfg_malloc

Syntax	void *vcm_cfg_malloc(vcm_size_t size)				
Include file	dsvcm.h				
Purpose	To allocate a block of the specified size in the config section memory.				
Parameters	size Specifies the size of the memory block.				
Return value	This function returns the pointer to the allocated config section memory block or 0 if the block could not be allocated.				
Related topics	<table><tr><td colspan="2">Basics</td></tr><tr><td>Basic Principles of VCM.....</td><td>145</td></tr></table>	Basics		Basic Principles of VCM.....	145
Basics					
Basic Principles of VCM.....	145				

vcm_memory_ptr_set

Syntax

```
Int32 vcm_memory_ptr_set(
    vcm_module_descriptor_type* ptr,
    vcm_cfg_mem_ptr_type cfg_mem_ptr,
    Uint32 size)
```

Include file

dsvcm.h

Purpose

To set the pointer and the size of a config section memory block that is associated with the module.

Parameters

ptr Specifies the pointer to a module descriptor.

cfg_mem_ptr Specifies the pointer to allocated config section memory. This pointer is returned by the function `vcm_cfg_malloc`.

size Specifies the size of allocated config section memory.

Return value

Specifies the error code. The following symbols are predefined:

Predefined Symbol	Meaning
VCN_NO_ERROR	Pointer and size set successfully
VCN_INVALID_MODULE	Module does not exist

Related topics

Basics

[Basic Principles of VCM.....](#) 145

References

[vcm_cfg_malloc.....](#) 151
[vcm_memory_ptr_get.....](#) 152

vcm_memory_ptr_get

Syntax

```
vcm_cfg_mem_ptr_type vcm_memory_ptr_get(
    vcm_module_descriptor_type* ptr)
```


Include file	<code>dsvcm.h</code>
Purpose	To get the pointer to the config section memory block that is associated with the module.
Parameters	ptr Specifies the pointer to a module descriptor.
Return value	This function returns the pointer to a config section memory block or 0 if the memory block could not be allocated.
Related topics	<p>Basics</p> <p>Basic Principles of VCM..... 145</p> <p>References</p> <p>vcm_memory_ptr_set..... 152</p>

vcm_module_find

Syntax	<pre>vcm_module_descriptor_type* vcm_module_find(Int32 mod_id, vcm_module_descriptor_type *prev_ptr)</pre>
Include file	<code>dsvcm.h</code>
Purpose	To find a pointer to the module descriptor by a given module ID.
Parameters	<p>mod_id Specifies the module ID within the range of 1 ... 999. Module IDs higher than 999 are used for dSPACE modules.</p> <p>prev_ptr Specifies the pointer to a previously found module, or 0.</p> <p>Note</p> <p>If more than one module with the same module ID are registered, use this parameter to start the search from the previously found pointer.</p>

Return value This function returns the pointer to a module descriptor or 0 if the module was not found.

Related topics

Basics

[Basic Principles of VCM..... 145](#)

References

[vcm_module_register..... 149](#)

vcm_module_status_set

Syntax

```
Int32 vcm_module_status_set(
    vcm_module_descriptor_type* ptr,
    Int32 status)
```

Include file

dsvcm.h

Purpose

To set the status of a software module.

Parameters

ptr Specifies the pointer to a module descriptor.

status Specifies the status value. The following symbols are predefined:

Predefined Symbol	Value	Meaning
VCM_STATUS_UNINITIALIZED	0x00	Module is not initialized
VCM_STATUS_INITIALIZED	0x01	Module is initialized
VCM_STATUS_ERROR	0x02	Error

Tip

You can define other values to be used as error numbers or additional status information.

Return value

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

Predefined Symbol	Meaning
VCM_NO_ERROR	Module status set successfully
VCM_INVALID_MODULE	The requested module does not exist or parameter ptr was 0

Example

```
error=vcm_module_status_set(msg_mod_ptr, VCM_INITIALIZED);
```

Related topics**Basics**

[Basic Principles of VCM..... 145](#)

References

[vcm_module_register..... 149](#)
[vcm_module_status_get..... 155](#)

vcm_module_status_get

Syntax

```
Int32 vcm_module_status_get(vcm_module_descriptor_type* ptr)
```

Include file

dsvcm.h

Purpose

To get the status of a given module.

Parameters

ptr Specifies the pointer to a module descriptor.

Return value

This function returns the module status, or 0 if the module does not exist. The following symbols are predefined:

Predefined Symbol	Value	Meaning
VCM_STATUS_UNINITIALIZED	0x00	Module is not initialized
VCM_STATUS_INITIALIZED	0x01	Module is initialized
VCM_STATUS_ERROR	0x02	Error

Related topics**Basics**[Basic Principles of VCM..... 145](#)**References**[vcm_module_status_set..... 154](#)

vcm_version_get

Syntax

```
vcm_version_type vcm_version_get(  
    vcm_module_descriptor_type* ptr)
```

Include file`dsvcm.h`**Purpose**

To get the version of a module.

Parameters**ptr** Specifies the pointer to a module descriptor.**Return value**This function returns the module version (see [Data Types for VCM](#) on page 148).**Related topics****Basics**[Basic Principles of VCM..... 145](#)**References**[vcm_version_compare..... 157](#)
[vcm_version_print..... 159](#)

vcm_version_compare

Syntax

```
Int32 vcm_version_compare(
    vcm_module_descriptor_type* ptr,
    Int32 operation,
    UInt8 mar,
    UInt8 mir,
    UInt8 mai,
    UInt8 spb,
    UInt16 spn,
    UInt16 plv)
```

Include file

dsvcm.h

Purpose

To compare the version of a module with a given version.

Parameters

ptr Specifies the pointer to a module descriptor.

operation Specifies the constant for operation. The following symbols are predefined:

Predefined Symbol	Meaning
VCM_VERSION_LT	Less than
VCM_VERSION_LE	Less or equal
VCM_VERSION_EQ	Equal
VCM_VERSION_GE	Greater or equal
VCM_VERSION_GT	Greater than

mar Specifies the major release.

mir Specifies the minor release.

mai Specifies the maintenance number.

spb Specifies the special build type.

spn Specifies the special build number.

plv Specifies the patch level.

Return value

This function returns the compare result:

Value	Meaning
0	Result is false
!=0	Result is true

Related topics**Basics**

[Basic Principles of VCM.....](#) 145

References

[vcm_version_get.....](#) 156
[vcm_version_print.....](#) 159

vcm_module_version_print

Syntax

```
Int32 vcm_module_version_print(
    char *buffer,
    vcm_module_descriptor_type* ptr)
```

Include file

dsvcm.h

Purpose

To print the module version into a char buffer.

Parameters

buffer Specifies the pointer to character buffer.
ptr Specifies the pointer to a module descriptor.

Return value

This function returns the number of chars printed into buffer (0: no version printed).

Related topics**Basics**

[Basic Principles of VCM.....](#) 145

References

[vcm_version_print.....](#) 159

vcm_version_print

Syntax

```
Int32 vcm_version_print(
    char *buffer,
    UInt8 mar,
    UInt8 mir,
    UInt8 mai,
    UInt8 spb,
    UInt16 spn,
    UInt16 plv)
```

Include file dsvcm.h

Purpose To print given version information into a char buffer.

Parameters

- buffer** Specifies the pointer to character buffer.
- mar** Specifies the major release.
- mir** Specifies the minor release.
- mai** Specifies the maintenance number.
- spb** Specifies the special build type.
- spn** Specifies the special build number.
- plv** Specifies the patch level.

Return value This function returns the number of chars printed into buffer.

Related topics

Basics	
Basic Principles of VCM.....	145
References	
vcm_module_version_print.....	158

Message Handling

Purpose To configure and generate messages.

Where to go from here

Information in this section

Basic Principles of Message Handling.....	161
Information on the Message module's basic principles.	
Data Types and Symbols for Message Handling.....	162
Information on the data types and symbols defined in the Message module.	
msg_error_set.....	164
To generate an error message.	
msg_warning_set.....	165
To generate a warning message.	
msg_info_set.....	166
To generate an information message.	
msg_set.....	166
To generate a message of the defined message class.	
msg_error_printf.....	168
To generate an error message with arguments using the <code>printf</code> format.	
msg_warning_printf.....	170
To generate a warning message with arguments using the <code>printf</code> format.	
msg_info_printf.....	171
To generate an information message with arguments using the <code>printf</code> format.	
msg_printf.....	172
To generate a message of the specified class with arguments using the <code>printf</code> format.	
msg_default_dialog_set.....	174
To specify the default dialog type for the selected message class.	
msg_mode_set.....	175
To set the mode of the message buffer.	
msg_reset.....	176
To reset the message buffer and clear the values of the last error.	
msg_last_error_number.....	177
To read the number of the last generated error message.	
msg_last_error_submodule.....	178
To read the submodule of the last generated error message.	

msg_error_clear.....	179
To set the number of the last generated error to 0 and the submodule of the last generated error message to MSG_SM_NONE .	
msg_error_hook_set.....	180
To install a hook function.	
msg_init.....	181
To initialize the message handling.	

Basic Principles of Message Handling

Introduction

The Message module provides functions to generate error, warning, and information messages to be displayed by the dSPACE experiment software. Messages are generated by the processor board and written to a message buffer, located in the global memory. Thus, the processor and the host PC have access to the memory section. On the host PC, the dSPACE experiment software displays the messages in the log window and writes them to the log file. Each message consists of a message number and the message string. To use the message module, you have to initialize the board via the initialization function `init()`.

Message characteristics

There are two predefined symbols that define the message buffer. The symbol `MSG_STRING_LENGTH` specifies the maximum length of a generated message. If a message exceeds the given length, it is truncated. The symbol `MSG_BUFFER_LENGTH` specifies the maximum number of messages that can be stored to the reserved memory. The behavior of the message buffer is controlled by the `msg_mode_set` function. The values of the message and buffer lengths are defined in `MsgXXXX.h` (XXXX denotes the relevant dSPACE board) or `StrkMsg.h` when you use DS1007 or MicroLabBox.

For the DS1006 Processor Board, there are the following default values:

Predefined Symbol	Default Value
<code>MSG_STRING_LENGTH</code>	400 characters
<code>MSG_BUFFER_LENGTH</code>	64 messages

Change the values of the standard message length and the message buffer only under the following conditions:

- The time to generate messages is too long.
- The message module needs too much memory.

To make changes work, call `Bldlib.bat` to regenerate the appropriate software environment library.

Message types

There are four message types:

Type	Representation in the dSPACE Experiment Software
ERROR	Dialog box containing the message text and entry in the Log window beginning with ERROR
WARNING	Entry in the Log window beginning with WARNING
INFO	Entry in the Log window
LOG	Entry in the Log file only

The following table gives examples for the three message types ERROR, WARNING, and INFO:

Module	Message Type	Board Name	Submodule	Message Text
Platform:	ERROR			Board is not present or expansion box is off.
DataKernel:	WARNING			Data connection not valid!
Real-Time Processor:		#1 DS1006 -	RTLib:	System started. (0)

Data Types and Symbols for Message Handling

Data types

The following data types are defined:

msg_string_type

```
typedef char msg_string_type;
```

msg_no_type

```
typedef Int32 msg_no_type;
```

msg_class_type

```
typedef enum msg_class_type;
```

msg_dialog_type

```
typedef enum msg_dialog_type;
```

msg_submodule_type

```
typedef UInt32 msg_submodule_type;
```

msg_hookfcn_type

```
typedef int (*msg_hookfcn_type)(msg_submodule_type, msg_no_type);
```

The following symbols are defined:

Predefined Symbol	Message refers to ...
MSG_SM_NONE	No specific module (default)
MSG_SM_USER	User messages
MSG_SM_CAN1401	RTLib: CAN (DS1401)
MSG_SM_CAN2202	RTLib: CAN (DS2202)
MSG_SM_CAN2210	RTLib: CAN (DS2210)
MSG_SM_CAN2211	RTLib: CAN (DS2211)
MSG_SM_CAN4302	RTLib: CAN (DS4302)
MSG_SM_DIO1401	RTLib: Digital I/O (DS1401)
MSG_SM_DS1104SLVLIB	RTLib: Slave DSP (DS1104)
MSG_SM_DS4501	RTLib: DS4501 functions
MSG_SM_DS4502	RTLib: DS4502 functions
MSG_SM_DSBYPASS	RTI: Bypass Blockset
MSG_SM_DSCAN	RTLib: CAN support
MSG_SM_DSETH	RTI: RTI Ethernet Blockset
MSG_SM_DSFR	RTLib: FlexRay support
MSG_SM_DSJ1939	J1939 Support in RTI CAN MultiMessage Blockset
MSG_SM_DSSER	RTLib: Serial interface
MSG_SM_ECU_POD	ECU PODs (DS5xx)
MSG_SM_ECU1401	RTLib: ECU interface (DS1401)
MSG_SM_HOSTSERV	Host services
MSG_SM_LIN	RTLib: LIN support
MSG_SM_REALMOTION	RealMotion / MotionDesk
MSG_SM_RTI	Real-Time Interface
MSG_SM_RTICAN	RTI: CAN Blockset
MSG_SM_RTICAN1401	RTI: CAN Blockset (DS1401)
MSG_SM_RTICAN2202	RTI: CAN Blockset (DS2202)
MSG_SM_RTICAN2210	RTI: CAN Blockset (DS2210)
MSG_SM_RTICAN2211	RTI: CAN Blockset (DS2211)
MSG_SM_RTICAN4302	RTI: CAN Blockset (DS4302)
MSG_SM_RTICANMM	RTI: CAN MultiMessage Blockset
MSG_SM_RTIFLEXRAY	RTI: FlexRay Blockset
MSG_SM_RTIFLEXRAYCONFIG	RTI: FlexRay Configuration Blockset
MSG_SM_RTILINMM	RTI: LIN MultiMessage Blockset
MSG_SM_RTIMP	RTI-MP (Real-Time Interface for multiprocessor systems)
MSG_SM_RTKERNEL	Real-Time Kernel
MSG_SM_RTLIB	Real-Time Board Library
MSG_SM_RTOSAL	RTOS Abstractionlayer

Predefined Symbol	Message refers to ...
MSG_SM_RTPYTHON	RTPythoninterpreter
MSG_SM_SIMENG	RTI: Simulation engine

msg_error_set

Syntax

```
void msg_error_set(
    msg_submodule_type module,
    msg_no_type msg_no,
    msg_string_type *msg)
```

Include file

dsmsg.h

Purpose

To generate an error message.

Note

If there is a hook function installed (see [msg_error_hook_set](#)), the hook function is called before the error message is generated.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 162.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

msg Specifies the message string (for information on the maximum length, see [Message characteristics](#) on page 161).

Return value

None

Related topics

Basics

[Basic Principles of Message Handling](#)..... 161

References

[msg_error_hook_set](#)..... 180
[msg_error_printf](#)..... 168

msg_warning_set

Syntax

```
void msg_warning_set(
    msg_submodule_type module,
    msg_no_type msg_no,
    msg_string_type *msg)
```

Include file

dsmsg.h

Purpose

To generate a warning message.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 162.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

msg Specifies the message string (for information on the maximum length, see [Message characteristics](#) on page 161).

Return value

None

Related topics

Basics

[Basic Principles of Message Handling](#)..... 161

References

[msg_warning_printf](#)..... 170

msg_info_set

Syntax

```
void msg_info_set(
    msg_submodule_type module,
    msg_no_type msg_no,
    msg_string_type *msg)
```

Include file

dsmg.h

Purpose

To generate an information message.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type MSG_SM_USER only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 162.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

msg Specifies the message string (for information on the maximum length, see [Message characteristics](#) on page 161).

Return value

None

Related topics

Basics

[Basic Principles of Message Handling..... 161](#)

References

[msg_info_printf..... 171](#)

msg_set

Syntax

```
void msg_set(
    msg_class_type msg_class,
    msg_dialog_type msg_dialog,
    msg_submodule_type module,
    msg_no_type msg_no,
    msg_string_type *msg)
```

Include file	<code>dsmsg.h</code>																		
Purpose	To generate a message of the defined message class.																		
Description	This function issues an error, information, or warning message that is displayed by the dSPACE experiment software, or a message that only appears in the log file. In addition to the other <code>msg_xxx_set</code> functions, the user can adjust the type of the message dialogs.																		
Parameters	<p>msg_class Specifies the type of the message. The following symbols are predefined:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>MSG_MC_ERROR</code></td><td>Error message</td></tr> <tr> <td><code>MSG_MC_INFO</code></td><td>Information message</td></tr> <tr> <td><code>MSG_MC_WARNING</code></td><td>Warning message</td></tr> <tr> <td><code>MSG_MC_LOG</code></td><td>Message appears only in the log file</td></tr> </tbody> </table> <p>msg_dialog Specifies the type of the dialog. The following types are predefined:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>MSG_DLG_NONE</code></td><td>No dialog, silent mode</td></tr> <tr> <td><code>MSG_DLG_OKCANCEL</code></td><td>OK/Cancel dialog</td></tr> <tr> <td><code>MSG_DLG_DEFAULT</code></td><td>Dialog type specified by <code>msg_default_dialog_set</code></td></tr> </tbody> </table> <p>module Specifies the predefined symbol of the application module generating the message. Use the module type <code>MSG_SM_USER</code> only for handcoded programs. For a list of all predefined symbols, refer to Data Types and Symbols for Message Handling on page 162.</p> <p>msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.</p> <p>msg Specifies the message string (for information on the maximum length, see Message characteristics on page 161).</p>	Predefined Symbol	Meaning	<code>MSG_MC_ERROR</code>	Error message	<code>MSG_MC_INFO</code>	Information message	<code>MSG_MC_WARNING</code>	Warning message	<code>MSG_MC_LOG</code>	Message appears only in the log file	Predefined Symbol	Meaning	<code>MSG_DLG_NONE</code>	No dialog, silent mode	<code>MSG_DLG_OKCANCEL</code>	OK/Cancel dialog	<code>MSG_DLG_DEFAULT</code>	Dialog type specified by <code>msg_default_dialog_set</code>
Predefined Symbol	Meaning																		
<code>MSG_MC_ERROR</code>	Error message																		
<code>MSG_MC_INFO</code>	Information message																		
<code>MSG_MC_WARNING</code>	Warning message																		
<code>MSG_MC_LOG</code>	Message appears only in the log file																		
Predefined Symbol	Meaning																		
<code>MSG_DLG_NONE</code>	No dialog, silent mode																		
<code>MSG_DLG_OKCANCEL</code>	OK/Cancel dialog																		
<code>MSG_DLG_DEFAULT</code>	Dialog type specified by <code>msg_default_dialog_set</code>																		
Return value	None																		

Example

The following example issues an error message without a dialog.

```
msg_set(
    MSG_MC_ERROR,
    MSG_DLG_NONE,
    MSG_SM_USER,
    1,
    "This is an error message.");
```

Related topics**Basics**

[Basic Principles of Message Handling..... 161](#)

References

[msg_printf..... 172](#)

msg_error_printf

Syntax

```
int msg_error_printf(
    msg_submodule_t module,
    msg_no_t msg_no,
    char *format,
    arg1, arg2, etc.)
```

Include file

dsmsg.h

Purpose

To generate an error message with arguments using the `printf` format (see a standard C documentation).

Result

`printf` builds the message string with the standard C command arguments of `printf(char *format, arg1, arg2, etc.)`. The string is then automatically given to `msg_error_set` to generate the message.

Note

If there is a hook function installed (see [msg_error_hook_set](#) on page 180), the hook function is called before the error message is generated.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 162.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

format Specifies the string using the `printf` format.

arg1, arg2, etc. Specifies the optional arguments for the format string (see a standard C documentation).

Note

The length of the format string is not restricted, but the default value of the maximum length is specified by `MSG_STRING_LENGTH`, see [Message characteristics](#) on page 161. Longer messages are truncated.

Return value

This function returns the number of characters which were printed to the message buffer.

Example

This example shows how to generate an error message with the `printf` format:

```
#include <Brtenv.h>
/* An example integer value */
int num = 13;
void main()
{
    /* Initialization of the board */
    init();
    /* Write an error message to the message buffer using the printf format */
    msg_error_printf(MSG_SM_USER, 1, "The value of num is %i", num);
}
```

Related topics

Basics

[Basic Principles of Message Handling](#)..... 161

References

[msg_error_hook_set](#)..... 180
[msg_error_set](#)..... 164

msg_warning_printf

Syntax

```
int msg_warning_printf(
    msg_submodule_type module,
    msg_no_type msg_no,
    char *format,
    arg1, arg2, etc.)
```

Include file

dsmsg.h

Purpose

To generate a warning message with arguments using the `printf` format (see a standard C documentation).

Result

`printf` builds the message string with the standard C command arguments of `printf(char *format, arg1, arg2, etc.)`. The string is then automatically passed to `msg_warning_set` to generate the message.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 162.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

format Specifies the string using the `printf` format.

arg1, arg2, etc. Specifies the optional arguments for the format string (see a standard C documentation).

Note

The length of the format string is not restricted, but the default value of the maximum length is specified by `MSG_STRING_LENGTH`, see [Message characteristics](#) on page 161. Longer messages are truncated.

Return value

This function returns the number of characters which were printed to the message buffer.

Related topics

Basics

[Basic Principles of Message Handling](#)..... 161

References

[msg_warning_set](#)..... 165

msg_info_printf

Syntax

```
int msg_info_printf(
    msg_submodule_t module,
    msg_no_t msg_no,
    char *format,
    arg1, arg2, etc.)
```

Include file

dsmsg.h

Purpose

To generate an information message with arguments using the **printf** format (see a standard C documentation).

Result

printf builds the message string with the standard C command arguments of **printf(char *format, arg1, arg2, etc.)**. The string is then automatically given to **msg_info_set** to generate the message.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type **MSG_SM_USER** only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 162.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

format Specifies the string using the `printf` format.

arg1, arg2, etc. Specifies the optional arguments for the format string (see a standard C documentation).

Note

The length of the format string is not restricted, but the default value of the maximum length is specified by `MSG_STRING_LENGTH`, see [Message characteristics](#) on page 161. Longer messages are truncated.

Return value This function returns the number of characters which were printed to the message buffer.

Related topics

Basics

[Basic Principles of Message Handling..... 161](#)

References

[msg_info_set..... 166](#)

msg_printf

Syntax

```
int msg_printf(
    msg_class_t msg_class,
    msg_dialog_t msg_dialog,
    msg_submodule_t module,
    msg_no_t msg_no,
    char *format,
    arg1, arg2, etc.)
```

Include file

`dsmsg.h`

Purpose

To generate a message of the specified class with arguments using the `printf` format (see a standard C documentation).

Result

`printf` builds the message string with the standard C command arguments of `printf(char *format, arg1, arg2, etc.)`. The string is then automatically given to `msg_set` to generate the message.

Parameters

msg_class Specifies the type of the message. The following symbols are predefined:

Predefined Symbol	Meaning
MSG_MC_ERROR	Error message
MSG_MC_INFO	Information message
MSG_MC_WARNING	Warning message
MSG_MC_LOG	Message appears only in the log file

msg_dialog Specifies the type of the dialog. The following types are predefined:

Predefined Symbol	Meaning
MSG_DLG_NONE	No dialog, silent mode
MSG_DLG_OKCANCEL	OK/Cancel dialog
MSG_DLG_DEFAULT	Dialog type specified by <code>msg_default_dialog_set</code>

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 162.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

format Specifies the string using the `printf` format.

arg1, arg2, etc. Specifies the optional arguments for the format string (see a standard C documentation).

Note

The length of the format string is not restricted, but the default value of the maximum length is specified by `MSG_STRING_LENGTH`, see [Message characteristics](#) on page 161. Longer messages are truncated.

Return value

This function returns the number of characters which were printed to the message buffer.

Example

The following example issues an information message dialog, which can be closed by pressing the **OK** or **Cancel** button.

```
msg_printf(
    MSG_MC_INFO,
    MSG_DLG_OKCANCEL,
    MSG_SM_USER,
    2,
    "The value of f = %f exceeded its critical limit!",
    f);
```

Related topics**Basics**

[Basic Principles of Message Handling..... 161](#)

References

[msg_set..... 166](#)

msg_default_dialog_set

Syntax

```
void msg_default_dialog_set(
    msg_class_type msg_class,
    msg_dialog_type msg_dialog)
```

Include file

dsmg.h

Purpose

To specify the default dialog type for the selected message class.

Result

The message module functions **msg_xxx_set** and **msg_xxx_printf** always use the specified default dialog type. The dialog type of the functions **msg_set** and **msg_printf** is set to the default type when they are calling with the **msg_dialog** argument **MSG_DLG_DEFAULT**.

Parameters

msg_class Specifies the type of the message. The following symbols are predefined:

Predefined Symbol	Meaning
MSG_MC_ERROR	Error message
MSG_MC_INFO	Information message

Predefined Symbol	Meaning
MSG_MC_WARNING	Warning message
MSG_MC_LOG	Message appears only in the log file

msg_dialog Specifies the type of the dialog. The following types are predefined:

Predefined Symbol	Meaning
MSG_DLG_NONE	No dialog, silent mode
MSG_DLG_OKCANCEL	OK/Cancel dialog

Return value None

Example

The following example turns off the dialog for error messages.

```
msg_default_dialog_set(
    MSG_MC_ERROR,
    MSG_DLG_NONE);
```

Related topics

Basics

[Basic Principles of Message Handling..... 161](#)

msg_mode_set

Syntax

```
void msg_mode_set(UINT32 mode)
```

Include file

dsmg.h

Purpose

To set the mode of the message buffer.

Description

This function specifies the behavior of the message buffer if the number of messages exceeds the maximum buffer length. On start-up, the overwrite mode is active.

Parameters

mode Specifies the mode of the message buffer. The following symbols are predefined:

Predefined Symbol	Meaning
MSG_BLOCKING	The message buffer will be filled to the maximum number of entries. Any further messages will be lost.
MSG_OVERWRITE	The message buffer will be filled cyclically. The oldest message will be overwritten when the buffer is full.

Return value

None

Related topics

Basics

[Basic Principles of Message Handling.....](#) 161

msg_reset

Syntax

```
void msg_reset()
```

Include file

dsmsg.h

Purpose

To reset the message buffer and clear the values of the last error (see **msg_error_clear**).

Description

The next message will be the first entry in the message buffer. Nevertheless, the message number will be incremented.

Return value

None

Related topics

Basics

[Basic Principles of Message Handling..... 161](#)

References

[msg_error_clear..... 179](#)

msg_last_error_number

Syntax

```
msg_no_type msg_last_error_number()
```

Include file

dsmsg.h

Purpose

To read the number of the last generated error message.

Description

Independently of the order of the messages in the message buffer, this function returns the number of the last error message. On start-up, the value is set to 0.

Note

Warning and information messages do not change this number.

Return value

This function returns the number of the last generated error message.

Related topics

Basics

[Basic Principles of Message Handling..... 161](#)

References

[msg_error_clear..... 179](#)

[msg_last_error_submodule..... 178](#)

msg_last_error_submodule

Syntax

```
msg_submodule_type msg_last_error_submodule()
```

Include file

dsmsg.h

Purpose

To read the submodule of the last generated error message.

Description

On start-up, the value is set to MSG_SM_NONE (see table below).

Note

Warning and information messages do not change this value.

Return value

This function returns the submodule of the last generated error message. The following symbols are defined:

Predefined Symbol	Message refers to ...
MSG_SM_NONE	No specific module (default)
MSG_SM_USER	User messages
MSG_SM_CAN1401	RTLib: CAN (DS1401)
MSG_SM_CAN2202	RTLib: CAN (DS2202)
MSG_SM_CAN2210	RTLib: CAN (DS2210)
MSG_SM_CAN2211	RTLib: CAN (DS2211)
MSG_SM_CAN4302	RTLib: CAN (DS4302)
MSG_SM_DIO1401	RTLib: Digital I/O (DS1401)
MSG_SM_DS1104SLVLIB	RTLib: Slave DSP (DS1104)
MSG_SM_DS4501	RTLib: DS4501 functions
MSG_SM_DS4502	RTLib: DS4502 functions
MSG_SM_DSBYPASS	RTI: Bypass Blockset
MSG_SM_DSCAN	RTLib: CAN support
MSG_SM_DSETH	RTI: RTI Ethernet Blockset
MSG_SM_DSFR	RTLib: FlexRay support
MSG_SM_DSJ1939	J1939 Support in RTI CAN MultiMessage Blockset
MSG_SM_DSSER	RTLib: Serial interface
MSG_SM_ECU_POD	ECU PODs (DS5xx)
MSG_SM_ECU1401	RTLib: ECU interface (DS1401)
MSG_SM_HOSTSERV	Host services

Predefined Symbol	Message refers to ...
MSG_SM_LIN	RTLib: LIN support
MSG_SM_REALMOTION	RealMotion / MotionDesk
MSG_SM_RTI	Real-Time Interface
MSG_SM_RTICAN	RTI: CAN Blockset
MSG_SM_RTICAN1401	RTI: CAN Blockset (DS1401)
MSG_SM_RTICAN2202	RTI: CAN Blockset (DS2202)
MSG_SM_RTICAN2210	RTI: CAN Blockset (DS2210)
MSG_SM_RTICAN2211	RTI: CAN Blockset (DS2211)
MSG_SM_RTICAN4302	RTI: CAN Blockset (DS4302)
MSG_SM_RTICANMM	RTI: CAN MultiMessage Blockset
MSG_SM_RTIFLEXRAY	RTI: FlexRay Blockset
MSG_SM_RTIFLEXRAYCONFIG	RTI: FlexRay Configuration Blockset
MSG_SM_RTILINMM	RTI: LIN MultiMessage Blockset
MSG_SM_RTIMP	RTI-MP (Real-Time Interface for multiprocessor systems)
MSG_SM_RTKERNEL	Real-Time Kernel
MSG_SM_RTLIB	Real-Time Board Library
MSG_SM_RTOSAL	RTOS Abstractionlayer
MSG_SM_RTPYTHON	RTPythoninterpreter
MSG_SM_SIMENG	RTI: Simulation engine

Related topics

Basics

[Basic Principles of Message Handling.....](#) 161

References

[msg_error_clear.....](#) 179
[msg_last_error_number.....](#) 177

msg_error_clear

Syntax

```
void msg_error_clear()
```

Include file

dsmsg.h

Purpose	To set the number of the last generated error to 0 and the submodule of the last generated error message to <code>MSG_SM_NONE</code> (refer to Data Types and Symbols for Message Handling on page 162).
Return value	None
Related topics	<div>Basics</div> <div>Basic Principles of Message Handling..... 161</div> <div>References</div> <div> msg_last_error_number..... 177 msg_last_error_submodule..... 178 msg_reset..... 176 </div>

msg_error_hook_set

Syntax	<code>void msg_error_hook_set(msg_hookfcn_type hook)</code>
Include file	<code>dsmsg.h</code>
Purpose	To install a hook function.
Description	<p>The hook function is activated when an error message is generated (see <code>msg_error_set</code> and <code>msg_error_printf</code>) and before the message is displayed.</p> <p>Use the hook function to:</p> <ul style="list-style-type: none"> ▪ React to an error (for example, to implement an error correction function) ▪ Suppress the error message <p>The hook function is activated for all errors. To react only for certain submodules or message numbers, you have to manage restrictions within your handcoded function (see example below).</p>
Parameters	hook Specifies the pointer to the hook function.

Return value

This function returns one of the following values:

Value	Meaning
1	The error message is displayed.
0	The error message is not displayed.

Example

This example shows how to use a hook function:

```
#include <Brtenv.h>
int error_hook_function(msg_submodule_type sm, msg_no_type no)
{
    if ((sm == MSG_SM_RTI) && (no == 1))
    {
        /* suppress error message */
        return(0);
    } else
    {
        /* display error message */
        return(1);
    }
}
void main()
{
    /* Initialization of the board */
    init();
    /* Announce the hook function to the message module */
    msg_error_hook_set(error_hook_function);
    /* Write an error message to the message buffer */
    msg_error_set(MSG_SM_USER, 1, "user error message");
    /* This error message will be suppressed by the
       hook function */
    msg_error_set(MSG_SM_RTI, 1, "RTI error message");
}
```

Related topics

Basics

[Basic Principles of Message Handling..... 161](#)

msg_init

Syntax

```
void msg_init(void)
```

Include file

dsmsg.h

Purpose	To initialize the message handling.
----------------	-------------------------------------

Description	This function is called automatically from within the <code>init()</code> function. The mode is set to <code>MSG_OVERWRITE</code> , counter and indices are set to 0. The buffer and string lengths are set according to the values of <code>MSG_BUFFER_LENGTH</code> and <code>MSG_STRING_LENGTH</code> defined in <code>Msgxxxx.h</code> .
--------------------	--

Return value	None
---------------------	------

Related topics

Basics

Basic Principles of Message Handling.....	161
---	---------------------

References

init().....	244
msg_mode_set.....	175

Watchdog Handling

Introduction

Use these functions to supervise a program execution. After you start the watchdog it has to be strobed before the watchdog timer period expires. When the timer period expires, the watchdog timer performs an action. Which action is performed depends on the watchdog mode:

Normal mode A watchdog interrupt (DS1006_INT_WD) is generated and the watchdog is set inactive. The watchdog has to be reinitialized for reuse.

Reset mode At first only a single watchdog interrupt (DS1006_INT_WD) is generated and the timer is reloaded. When it expires a second time a RTP reset pulse is generated. After the reset the DS1006 starts automatically the flash application, if one exists. If no flash application exists, the DS1006 waits as long as an application is loaded into the global memory.

You can install an interrupt handler that is executed when a watchdog interrupt occurs.

Where to go from here

Information in this section

ds1006_wd_set	183
ds1006_wd_init	184
ds1006_wd_stop	185
ds1006_wd_strobe	186
ds1006_wd_system_was_reset	186

ds1006_wd_set

Syntax

```
double ds1006_wd_set(
    int mode,
    double period)
```

Include file

wd1006.h

Purpose

To initialize the watchdog timer and set the watchdog mode.

Note

After initialization the watchdog has to be strobed once (see `ds1006_wd_strobe`) to start operation.

Parameters

mode Specifies the watchdog timer mode. The following symbols are predefined:

Predefined Symbols	Meaning
DS1006_WD_NORMAL	Normal mode on page 183
DS1006_WD_RESET	Reset mode on page 183

period Specifies the watchdog timer period in seconds. The value has to be within the range of 1024/bus clock ... (32768*1024)/bus clock. The timer resolution is 1024/bus clock.

Return value

This function returns the value of the actual watchdog timer period regarding the range and the resolution of the watchdog timer.

Related topics**References**

ds1006_wd_init	184
ds1006_wd_strobe	186

ds1006_wd_init

Syntax

```
double ds1006_wd_init(  
    int mode,  
    double period,  
    void *handler)
```

Include file

`wd1006.h`

Purpose

To initialize the watchdog timer, set the watchdog mode, and install a watchdog interrupt handler.

Note

After initialization the watchdog has to be strobed once (see `ds1006_wd_strobe`) to start operation.

Parameters

mode Specifies the watchdog timer mode. The following symbols are predefined:

Predefined Symbols	Meaning
DS1006_WD_NORMAL	Normal mode on page 183
DS1006_WD_RESET	Reset mode on page 183

period Specifies the watchdog timer period in seconds. The value has to be within the range of 1024/bus clock ... (32768*1024)/bus clock. The timer resolution is 1024/bus clock.

handler Specifies the pointer to the watchdog interrupt handler.

Return value

This function returns the value of the actual watchdog timer period regarding the range and the resolution of the watchdog timer.

Related topics**References**

ds1006_wd_set	183
ds1006_wd_strobe	186

ds1006_wd_stop

Syntax

```
void ds1006_wd_stop(void)
```

Include file

`wd1006.h`

Purpose

To stop the watchdog timer and disable the watchdog interrupt.

Parameters

None

Return value	None
---------------------	------

Related topics**References**

ds1006_wd_init.....	184
-------------------------------------	-----

ds1006_wd_strobe

Syntax

```
void ds1006_wd_strobe()
```

Include file

```
wd1006.h
```

Purpose

To re-trigger the watchdog timer.

Description

If the watchdog timer is not re-triggered within the specified period, an action depending on the watchdog mode is performed. The strobe function resets the counter which detects the occurrence of the watchdog interrupt which is triggered just before the watchdog generates the reset pulse.

Parameters

None

Return value

None

Related topics**References**

ds1006_wd_init.....	184
ds1006_wd_set.....	183
ds1006_wd_system_was_reset.....	186

ds1006_wd_system_was_reset

Syntax

```
int ds1006_wd_system_was_reset(void)
```

Include file	wd1006.h
Purpose	To determine the number of resets since the system was last started normally.
Parameters	None
Return value	The function returns the number of resets which were triggered by the watchdog timer since the system was last started normally.
Related topics	<div>References<div>ds1006_wd_strobe..... 186</div></div>

Serial Interface Communication

Introduction

This section contains the generic functions for communication via a serial interface.

The generic functions use a receive and transmit buffer to buffer the data. Because they do not have direct access to the UART, they are hardware-independent and can be used for different I/O boards. These generic functions are described in this chapter.

Where to go from here

Information in this section

Basic Principles of Serial Communication.....	188
Data Types for Serial Communication.....	192
Generic Serial Interface Communication Functions.....	200

Basic Principles of Serial Communication

Where to go from here

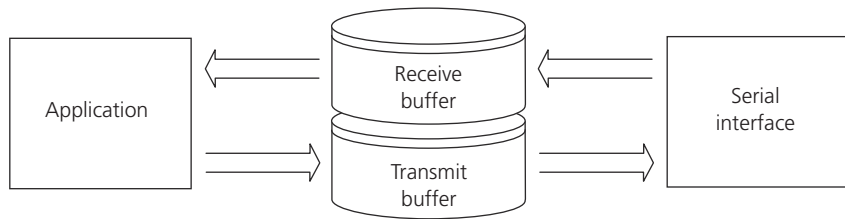
Information in this section

Software FIFO Buffer.....	188
To get information about the receive and transmit buffers.	
Trigger Levels.....	189
To get information about the trigger levels.	
How to Handle Subinterrupts in Serial Communication.....	190
Instructions on handling subinterrupts in serial communication.	
Example of a Serial Interface Communication.....	191
Shows you how to implement serial interface communication.	

Software FIFO Buffer

Introduction

The software FIFO buffer is a memory section that provides the UART with additional space for data storage and ensures that the generic functions are hardware-independent.



The software FIFO buffer stores data that will be written to (transmit buffer) or has been read by (receive buffer) the UART.

Buffer size

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

Transmit buffer

The transmit buffer is filled with data to be sent as long as free space is available. It cannot be overwritten. You can write data to the transmit buffer with the function `dsSER_transmit`.

Receive buffer

The receive buffer is filled with data received by the UART as long as free space is available. If an overflow occurs, old data in the receive buffer is overwritten or new data is rejected. This depends on the mode of the FIFO. You can access the receive buffer by using the functions `dsSER_receive` and `dsSER_receive_term`.

Related topics

Basics

[Trigger Levels.....](#) 189

References

[dsSER_receive.....](#) 208
[dsSER_receive_term.....](#) 209
[dsSER_transmit.....](#) 206

Trigger Levels

Introduction

Two different trigger levels can be configured.

UART trigger level

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

User trigger level

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

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 188

HowTos

[How to Handle Subinterrupts in Serial Communication.....](#) 190

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 189).
DSSER_TX_FIFO_EMPTY_SUBINT	Generated when the transmit buffer has no data.
DSSER_RECEIVER_LINE_SUBINT	Line status interrupt provided by the UART.
DSSER_MODEM_STATE_SUBINT	Modem status interrupt provided by the UART.
DSSER_NO_SUBINT	Generated after the last subinterrupt. This subinterrupt tells your application that no further subinterrupts were generated.

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

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

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

- 2 Initialize your subinterrupt handler:

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

- 3 Enable the required subinterrupts:

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

Related topics**Basics**

Trigger Levels..... 189

References

dsser_subint_enable..... 220
dsser_subint_handler_inst..... 219
dsser_subint_handler_t..... 197
dsserChannel..... 198

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();

    /* allocate a new 1024 byte SW-FIFO */
    serCh = dsser_init(DSSER_ONBOARD, 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.....	193
Provides information about the interrupt identification register.	
dsser_LSR.....	195
Provides information about the status of data transfers.	
dsser_MSR.....	196
Provides information about the state of the control lines.	
dsser_subint_handler_t.....	197
Provides information about the subinterrupt handler.	
dsserChannel.....	198
Provides information about the serial channel.	

dsser_ISR

Syntax

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

Include file

dsserdef.h

Description

The 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..... 216](#)

dsser_LSR

Syntax

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

Include file

dsserdef.h

Description

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

Note

The data type contains the value of the UART's register. The register conforms to a standard 16550 UART such as the TEXAS INSTRUMENTS TL16C550C. For further information, refer to <http://www.ti.com>.

Members

The structure provides the following members.

Member	Description
DSSER_RECEIVE_DATA_RDY	Data ready (DR) indicator
DSSER_OVERRUN_ERR	Overrun error (OE) indicator
DSSER_PARITY_ERR	Parity error (PE) indicator
DSSER_FRAMING_ERR	Framing error (FE) indicator
DSSER_BREAK_STATUS	Break interrupt (BI) indicator
DSSER_THR_STATUS	Transmitter holding register empty (THRE)
DSSER_THR_TSR_STATUS	Transmitter empty (TEMT) indicator
DSSER_FIFO_DATA_ERR	Error in receiver FIFO

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

Related topics

References

[dsser_status_read](#)..... 216

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](#)..... 216

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 190 or [dsser_subint_handler_inst](#) on page 219).

Members

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 201).

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 189).

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..... 189](#)

References

[dsser_init..... 201](#)

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 193.
- lineStatusReg** Line status register. Refer to [dsser_LSR](#) on page 195.
- modemStatusReg** Modem status register. Refer to [dsser_MSR](#) on page 196.

Related topics [References](#)

[dsser_status_read](#)..... 216

Generic Serial Interface Communication Functions

Where to go from here

Information in this section

dsser_init	201
To initialize the serial interface and install the interrupt handler.	
dsser_free	202
To close a serial interface.	
dsser_config	203
To configure and start the serial interface.	
dsser_transmit	206
To transmit data through the serial interface.	
dsser_receive	208
To receive data through the serial interface.	
dsser_receive_term	209
To receive data through the serial interface.	
dsser_fifo_reset	211
To reset the serial interface.	
dsser_enable	212
To enable the serial interface.	
dsser_disable	212
To disable the serial interface.	
dsser_error_read	213
To read an error flag of the serial interface.	
dsser_transmit_fifo_level	214
To get the number of bytes in the transmit buffer.	
dsser_receive_fifo_level	215
To get the number of bytes in the receive buffer.	
dsser_status_read	216
To read the value of one or more status registers and store the values in the appropriate fields of the channel structure.	
dsser_handle_get	217
To check whether the serial interface is in use.	
dsser_set	218
To set a property of the UART.	
dsser_subint_handler_inst	219
To install a subinterrupt handler for the serial interface.	
dsser_subint_enable	220
To enable one or several subinterrupts of the serial interface.	

dsser_subint_disable	221
To disable one or several subinterrupts of the serial interface.	
dsser_word2bytes	222
To convert a word (max. 4 bytes long) into a byte array.	
dsser_bytes2word	224
To convert a byte array with a maximum of 4 elements into a single word.	

dsser_init

Syntax

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

Include file

dsser.h

Purpose

To initialize the serial interface and install the interrupt handler.

Note

Pay attention to the initialization sequence. 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 DSSER_ONBOARD.

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 <code>channel</code> parameter is out of range.
700	Error	x, ch=y, Buffersize: Illegal	The <code>fifo_size</code> parameter is out of range.

Related topics**Basics**

[Basic Principles of Serial Communication..... 188](#)

Examples

[Example of a Serial Interface Communication..... 191](#)

References

[Data Types for Serial Communication..... 192](#)
[dsser_config..... 203](#)
[dsser_free..... 202](#)

dsser_free

Syntax

```
Int32 dsser_free(dsserChannel1*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 201).

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 <code>dsser_init</code> 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 (<code>serCh == NULL</code>).

Related topics

Basics

[Basic Principles of Serial Communication..... 188](#)

References

[dsser_init..... 201](#)

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 201).

fifo_mode Specifies the mode of the receive buffer (see [Software FIFO Buffer](#) on page 188):

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	5 ... 115,200 baud

For further information, refer to [Specifying the Baud Rate of the Serial Interface \(DS1006 Features !\[\]\(ec9132f1d27c8919987d92907322654d_img.jpg\)](#)).

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

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

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

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

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

uart_trigger_level Sets the UART trigger level (see [Trigger Levels](#) on page 189). 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 189):

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_AUTOFLOW_DISABLE	Transfer without HW handshake (RTS/CTS)
DSSER_AUTOFLOW_ENABLE	Transfer with HW handshake (RTS/CTS)

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.

ID	Type	Message	Description
604	Error	x, ch=y, stopbits: Illegal number (1-2 bits allowed)!	The <code>stopbits</code> parameter is out of range.
605	Error	x, ch=y, parity: Illegal parity!	The <code>parity</code> parameter is out of range.
606	Error	x, ch=y, trigger_level: Illegal UART trigger level!	The <code>uart_trigger_level</code> parameter is out of range.
607	Error	x, ch=y, trigger_level: Illegal user trigger level!	The <code>user_trigger_level</code> parameter is out of range.
608	Error	x, ch=y, fifo_mode: Use range 0 ... (fifo_size-1) bytes!	The <code>uart_mode</code> parameter is out of range.
609	Error	x, ch=y, uart_mode: Transceiver not supported!	The selected UART mode does not exist for this serial interface.
611	Error	x, ch=y, uart_mode: Autoflow is not supported!	Autoflow does not exist for this serial interface.

Related topics

Basics

[Basic Principles of Serial Communication.....](#) 188

Examples

[Example of a Serial Interface Communication.....](#) 191

References

[dsrser_init.....](#) 201

dsrser_transmit

Syntax

```
Int32 dsrser_transmit(
    dsrserChannel* serCh,
    UInt32 datalen,
    UInt8* data,
    UInt32* count)
```

Include file

`dsrser.h`

Purpose

To transmit data through the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 201).

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.....](#) 188

Examples

[Example of a Serial Interface Communication.....](#) 191

References

[dsser_init.....](#) 201
[dsser_transmit_fifo_level.....](#) 214

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 201).

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 <code>fifo_mode</code> adjusted with dsser_config : <ul style="list-style-type: none"> ▪ <code>fifo_mode = DSSER_FIFO_MODE_BLOCKED</code> Not all new data could be placed in the FIFO. ▪ <code>fifo_mode = DSSER_FIFO_MODE_OVERWRITE</code> The old data is rejected.
DSSER_COMMUNICATION_FAILED	The function failed with no effect on the input or output data. No data is read from the FIFO. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Example

The following example shows how to receive 4 bytes.

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

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 188

Examples

[Example of a Serial Interface Communication.....](#) 191

References

[dsser_init.....](#) 201

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 201).

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 <code>fifo_mode</code> adjusted with <code>dsser_config</code> : <ul style="list-style-type: none"> ▪ <code>fifo_mode = DSSER_FIFO_MODE_BLOCKED</code> Not all new data could be placed in the FIFO. ▪ <code>fifo_mode = DSSER_FIFO_MODE_OVERWRITE</code> The old data is rejected.
DSSER_COMMUNICATION_FAILED	The function failed with no effect on the input or output data. No data is read from the FIFO. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Example

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

```
UInt8 data[4];
UInt32 count;
Int32 error;
error = dsser_receive_term(serCh, 4, data, &count, '\r');
```

Related topics

Basics

[Basic Principles of Serial Communication..... 188](#)

References

[dsser_init..... 201](#)

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 201).

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..... 188](#)

References

[dsser_enable..... 212](#)
[dsser_init..... 201](#)

dsser_enable

Syntax

```
Int32 dsser_enable(const dsserChannel1* 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 201).

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..... 188](#)

References

[dsser_disable..... 212](#)
[dsser_init..... 201](#)

dsser_disable

Syntax

```
Int32 dsser_disable(const dsserChannel1* serCh)
```

Include file	<code>dsser.h</code>
Purpose	To disable the serial interface.
Description	The serial interface stops transmitting data, incoming data is no longer stored 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 201).
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	<p>Basics</p> <p>Basic Principles of Serial Communication..... 188</p> <p>References</p> <p>dsser_enable..... 212</p> <p>dsser_init..... 201</p>

dsser_error_read

Syntax	<code>Int32 dsser_error_read(const dsserChannel* serCh)</code>
Include file	<code>dsser.h</code>
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 201).

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

Predefined Symbol	Meaning
<code>DSSER_NO_ERROR</code>	No error flag set
<code>DSSER_FIFO_OVERFLOW</code>	Too many bytes for the buffer

Related topics

Basics

[Basic Principles of Serial Communication..... 188](#)

References

[dsser_config..... 203](#)
[dsser_init..... 201](#)

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 201).

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

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 188

References

[dsser_init.....](#) 201
[dsser_receive_fifo_level.....](#) 215

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 201).

Return value

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

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 188

References

[dsser_init.....](#) 201
[dsser_transmit_fifo_level.....](#) 214

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 201).

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

Predefined Symbol	Meaning
DSSER_STATUS_IIR_FCR	Interrupt status register, see dsser_ISR data type.
DSSER_STATUS_LSR	Line status register, see dsser_ISR data type.
DSSER_STATUS_MSR	Modem status register, see dsser_ISR data type.

Return value

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

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

Example

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

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


Related topics**Basics**

[Basic Principles of Serial Communication..... 188](#)

References

[dsser_init..... 201](#)
[dsser_ISR..... 193](#)
[dsser_LSR..... 195](#)
[dsser_MSR..... 196](#)

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 `DSSER_ONBOARD`.

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**

[Basic Principles of Serial Communication..... 188](#)

References

[dsser_init..... 201](#)

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 DS1006 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 201).

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](#)..... 188

References

[dsser_config](#)..... 203
[dsser_init](#)..... 201

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 220).

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 201).

subint_handler Specifies the pointer to the subinterrupt handler.

Return value

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

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 188

Examples

[Example of a Serial Interface Communication.....](#) 191

References

[dsrser_init.....](#) 201
[dsrser_subint_disable.....](#) 221
[dsrser_subint_enable.....](#) 220

dsrser_subint_enable

Syntax

```
Int32 dsrser_subint_enable(
    dsrserChannel* serCh,
    const UInt8 subint)
```

Include file

`dsrser.h`

Purpose

To enable one or several subinterrupts of the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsrser_init](#) on page 201).

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 189)
DSSER_TX_FIFO_EMPTY_SUBINT_MASK	Interrupt triggered when the transmit buffer is empty

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

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

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

Related topics

Basics

[Basic Principles of Serial Communication..... 188](#)

Examples

[Example of a Serial Interface Communication..... 191](#)

References

[dsrser_init..... 201](#)
[dsrser_subint_disable..... 221](#)
[dsrser_subint_handler_inst..... 219](#)

dsrser_subint_disable

Syntax

```
Int32 dsrser_subint_disable(
    dsrserChannel* serCh,
    const UInt8 subint)
```

Include file

dsrser.h

Purpose

To disable one or several subinterrupts of the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsrser_init](#) on page 201).

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 189)
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..... 188](#)

References

[dsser_init..... 201](#)
[dsser_subint_enable..... 220](#)
[dsser_subint_handler_inst..... 219](#)

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	<p>word Specifies the pointer to the input word.</p> <p>bytes Specifies the pointer to the byte array. The byte array must have enough memory for bytesInWord elements.</p> <p>bytesInWord Specifies the number of elements in the byte array. Possible values are 2, 3, 4.</p>
-------------------	---

Return value	This function returns the pointer to a byte array.
---------------------	--

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

```
void word_transmit(dsserChannel* serCh, UInt32* word, UInt32* count)
{
    UInt8    bytes[4];
    UInt8*   data_p;
    if(dsser_transmit_fifo_level(serCh) < serCh->fifo_size - 4)
    {
        data_p = dsser_word2bytes(word, bytes, 4);
        dsser_transmit(serCh, 4, data_p, count);
    }
    else
    {
        *count = 0;
    }
}
```

Use of the function:

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

Related topics

Basics

[Basic Principles of Serial Communication.....](#) 188

References

[dsser_bytes2word.....](#) 224
[dsser_transmit.....](#) 206
[dsser_transmit_fifo_level.....](#) 214

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.....](#) 188

References

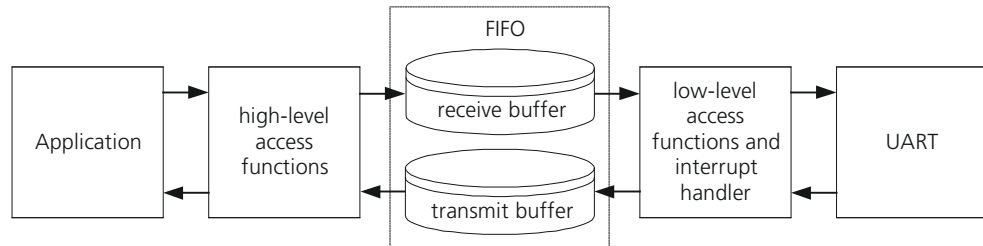
[dsser_receive.....](#) 208
[dsser_receive_fifo_level.....](#) 215
[dsser_word2bytes.....](#) 222

Serial Interface Communication (Low Level)

Introduction

This chapter contains the low-level access functions for communication via a serial interface.

There are two different levels of access functions, which are shown in the following figure.



High-level access functions These functions use a receive and transmit buffer to buffer the data. Because they do not have direct access to the UART they are hardware independent and can be used for different I/O boards. See [Serial Interface Communication](#) on page 188 for a description of the functions.

Low-level access functions These functions use no additional buffer and have direct access to the UART. They are hardware dependent and cannot be used for different I/O boards. The functions are described in this chapter.

Note

- Programming with low-level access functions is much more complex than programming with high-level access functions. You should therefore use low-level access functions only if high-level access functions do not fulfill your requirements, such as when you program your own transfer protocol.
- 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 only to use the generic functions.

Data types

The low-level access functions use the same data types as are defined for the high-level access functions. Refer to [Data Types for Serial Communication](#) on page 192.

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

ds1006_serial_init.....	227
ds1006_serial_config.....	228
ds1006_serial_free.....	230
ds1006_serial_transmit.....	230
ds1006_serial_receive.....	231
ds1006_serial_register_write.....	232
ds1006_serial_register_read.....	233

ds1006_serial_init

Syntax

```
dsserChannel* ds1006_serial_init(UInt32 uart_addr)
```

Include file

```
ser1006.h
```

Purpose

To allocate the serial channel structure.

Parameters

uart_addr Specifies the address of the UART, use DS1006_UART.

Return value

This function returns the pointer to the serial channel structure.

Related topics**References**

ds1006_serial_config.....	228
---	---------------------

ds1006_serial_config

Syntax

```
Int32 ds1006_serial_config(
    dsserChannel* serCh,
    const UInt32 baudrate,
    const UInt32 databits,
    const UInt32 stopbits,
    const UInt32 parity,
    const UInt32 uart_trigger_level,
    const UInt32 autoflow,
    const UInt16 interrupt_mode)
```

Include file

ser1006.h

Purpose

To configure the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see ds1006_serial_init).

baudrate Specifies the baud rate in bits per second. Values are: 300, 600, 1200, 2400, 4800, 9600 19200, 38400, 115000.

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
DS1006_SER_1_STOPBIT	1 stop bit
DS1006_SER_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
DS1006_SER_NO_PARITY	No parity bits
DS1006_SER_ODD_PARITY	Parity bit is set so that there are an odd number of "1" bits in the byte, including the parity bit
DS1006_SER_EVEN_PARITY	Parity bit is set so that there are an even number of "1" bits in the byte, including the parity bit

Predefined Symbol	Meaning
DS1006_SER_FORCED_PARITY_ONE	Parity bit is forced to a logic 1
DS1006_SER_FORCED_PARITY_ZERO	Parity bit is forced to a logic 0

uart_trigger_level Specifies the UART trigger level. The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_SER_1_BYTE_TRIGGER_LEVEL	Trigger level of 1 bytes
DS1006_SER_4_BYTE_TRIGGER_LEVEL	Trigger level of 4 bytes
DS1006_SER_8_BYTE_TRIGGER_LEVEL	Trigger level of 8 bytes
DS1006_SER_14_BYTE_TRIGGER_LEVEL	Trigger level of 14 bytes

Note

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

autoflow Enables or disables the hardware handshaking (autoflow control). The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_SER_AUTOFLOW_DISABLE	Transfer without hardware handshake (RTS/CTS)
DS1006_SER_AUTOFLOW_ENABLE	Transfer with hardware handshake (RTS/CTS)

interrupt_mode Specifies the interrupt mode. The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_SER_RXRDY_INT	Received data interrupt
DS1006_SER_THR_EMPTY_INT	Transmitter holding register empty interrupt
DS1006_SER_LINE_STATE_INT	Receiver line status interrupt
DS1006_SER_MODEM_ST_INT	Modem status interrupt

Return value

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

Predefined Symbol	Meaning
DS1006_SER_NO_ERROR	No error

Related topics

References

[ds1006_serial_init..... 227](#)

ds1006_serial_free

Syntax

```
void ds1006_serial_free(dsserChannel* serCh)
```

Include file

ser1006.h

Purpose

To free the memory of the serial channel structure and disable the serial hardware interrupt.

Parameters

serCh Specifies the pointer to the serial channel structure (see ds1006_serial_init).

Return value

None

Related topics

References

[ds1006_serial_init..... 227](#)

ds1006_serial_transmit

Syntax

```
Int32 ds1006_serial_transmit (
    const dsserChannel* serCh,
    UInt32 len,
    UInt8* data)
```

Include file

ser1006.h

Purpose

To transmit data through the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see ds1006_serial_init).

len Specifies the number of bytes to be transmitted.

data Specifies the start address of the data to be transmitted.

Return value	This function returns the number of bytes that were transmitted.
Related topics	<div>References</div> <div> ds1006_serial_init..... 227 </div>

ds1006_serial_receive

Syntax	<pre>Int32 ds1006_serial_receive(const dsserChannel* serCh, UInt32 len, UInt8* data)</pre>
Include file	ser1006.h
Purpose	To receive data through the serial interface.
Parameters	<p>serCh Specifies the pointer to the serial channel structure (see ds1006_serial_init).</p> <p>len Specifies the length of the data buffer (in bytes).</p> <p>data Specifies the start address of the data buffer.</p>
Return value	This function returns the number of bytes that were received.
Related topics	<div>References</div> <div> ds1006_serial_init..... 227 </div>

ds1006_serial_register_write

Syntax

```
void ds1006_serial_register_write(
    const dsserChannel* serCh,
    const UInt8 register_type,
    const UInt8 register_value)
```

Include file

ser1006.h

Purpose

To set a value in a register of the UART.

Parameters

serCh Specifies the pointer to the serial channel structure (see ds1006_serial_init).

register_type Specifies the register type to be set. The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_SER_RBR_THR	Receiver buffer register (read), transmit holding register (write)
DS1006_SER_IER	Interrupt enable register
DS1006_SER_IIR_FCR	Interrupt identification register (read), FIFO control register (write)
DS1006_SER_LCR	Line control register
DS1006_SER_MCR	Modem control register
DS1006_SER_LSR	Line status register
DS1006_SER_MSR	Modem status register
DS1006_SER_SCR	Scratch register

register_value Specifies the value to be set.

Return value

None

Related topics

References

ds1006_serial_init..... 227
ds1006_serial_register_read..... 233

ds1006_serial_register_read

Syntax

```
UInt8 ds1006_serial_register_read(
    const dsserChannel* serCh,
    const UInt8 register_type)
```

Include file

ser1006.h

Purpose

To read a register of the UART.

Parameters

serCh Specifies the pointer to the serial channel structure (see ds1006_serial_init).

register_type Specifies the register type to be read. The following symbols are predefined:

Predefined Symbol	Meaning
DS1006_SER_RBR_THR	Receiver buffer register (read), transmit holding register (write)
DS1006_SER_IER	Interrupt enable register
DS1006_SER_IIR_FCR	Interrupt identification register (read), FIFO control register (write)
DS1006_SER_LCR	Line control register
DS1006_SER_MCR	Modem control register
DS1006_SER_LSR	Line status register
DS1006_SER_MSR	Modem status register
DS1006_SER_SCR	Scratch register

Return value

This function returns the register value.

Related topics

References

ds1006_serial_init..... 227
ds1006_serial_register_write..... 232

Special Processor Functions

Purpose To ensure proper operation of the AMD Opteron™ processor.

Where to go from here Information in this section

[RTLIB_FORCE_IN_ORDER..... 234](#)
To force the processor to do the last I/O access in order.

[RTLIB_SYNC..... 235](#)
To force the processor to perform all pending memory accesses.

RTLIB_FORCE_IN_ORDER

Syntax `void RTLIB_FORCE_IN_ORDER(void)`

Include file `dsstd.h`

Purpose To force the processor to execute the I/O accesses in order.

Description This macro ensures that the processor executes I/O accesses in the right order. For example, when two I/O accesses are performed sequentially, the processor can change their order. If the `RTLIB_FORCE_IN_ORDER` macro is executed between the two accesses, they are executed in the specified order.

Return value None

Related topics References

[RTLIB_SYNC..... 235](#)
[Standard Macros..... 242](#)

RTLIB_SYNC

Syntax	<code>void RTLIB_SYNC(void)</code>
Include file	<code>dsstd.h</code>
Purpose	To force the processor to perform all pending memory accesses.
Description	This macro ensures that the processor performs all memory accesses that were issued before the macro was called.
Return value	None
Related topics	<div>References<div>RTLIB_FORCE_IN_ORDER..... 234 Standard Macros..... 242</div></div>

Conversion Functions

Introduction

Use these functions to convert floating-point values to other formats. Conversion is necessary because the AMD Opteron™ processor and the TI slave processors on the I/O boards use different floating-point formats. The AMD Opteron™ processor uses the IEEE floating-point format, and the TI slave processor uses the TI floating-point format. TI floating-point values are stored as UInt32 because they are usually transferred to the external hardware through 32-bit I/O registers.

Where to go from here

Information in this section

ds1006_conv32_ieee_to_ti	236
ds1006_conv32_ti_to_ieee	237
ds1006_conv_float_to_int32	237
RTLlib_CONV_FLOAT32_FROM_IEEE32	238
RTLlib_CONV_FLOAT32_FROM_TI32	239
RTLlib_CONV_FLOAT32_TO_IEEE32	239
RTLlib_CONV_FLOAT_TO_SATURATED_INT32	240
RTLlib_CONV_FLOAT32_TO_TI32	240

ds1006_conv32_ieee_to_ti

Syntax

```
UInt32 ds1006_conv32_ieee_to_ti(Float32 ieee_32)
```

or

```
UInt32 RTLlib_CONV_FLOAT32_TO_TI32(Float32 ieee_32)
```

Include file

fc1006.h

Purpose

To convert a value in IEEE floating-point format to TI floating-point format.

Parameters

ieee_32 Specifies the value in IEEE floating-point format.

Return value This function returns the value in TI floating-point format.

Related topics

References

ds1006_conv32_ti_to_ieee	237
RTLIB_CONV_FLOAT32_TO_TI32	240
Standard Macros	242

ds1006_conv32_ti_to_ieee

Syntax

```
Float32 ds1006_conv32_ti_to_ieee(UInt32 ti_32)
```

or

```
Float32 RTLIB_CONV_FLOAT32_FROM_TI32(UInt32 ti_32)
```

Include file `fc1006.h`

Purpose To convert a value in TI floating-point format to IEEE floating-point format.

Parameters **ti_32** Specifies the value in TI floating-point format.

Return value This function returns the value in IEEE floating-point format.

Related topics

References

ds1006_conv32_ieee_to_ti	236
RTLIB_CONV_FLOAT32_FROM_TI32	239
Standard Macros	242

ds1006_conv_float_to_int32

Syntax

```
Int32 ds1006_conv_float_to_int32(double fp_value)
```

or

```
Int32 RTLIB_CONV_FLOAT_TO_SATURATED_INT32(double fp_value)
```

Include file	<code>fc1006.h</code>
Purpose	To convert a value in floating-point format to signed integer format. If <code>fp_value</code> exceeds the limits of a signed integer, the return value is saturated to <code>INT_MAX</code> or <code>INT_MIN</code> respectively.
Parameters	fp_value Specifies the value in floating-point format (float or double).
Return value	This function returns the value in signed integer format, possibly saturated.
Related topics	References RTLIB_CONV_FLOAT_TO_SATURATED_INT32..... 240

RTLIB_CONV_FLOAT32_FROM_IEEE32

Syntax	<pre>Float32 RTLIB_CONV_FLOAT32_FROM_IEEE32(UInt32 ieee_32)</pre>
Include file	<code>dsstd.h</code>
Purpose	To convert a value in IEEE floating-point format to native floating-point format.
Parameters	ieee_32 Specifies the value in IEEE floating-point format.
Return value	This function returns the value in native floating-point format.
Related topics	References RTLIB_CONV_FLOAT32_TO_IEEE32..... 239 Standard Macros..... 242

RTLIB_CONV_FLOAT32_FROM_TI32

Syntax	<code>Float32 RTLIB_CONV_FLOAT32_FROM_TI32(UInt32 ti_32)</code>
Include file	<code>dsstd.h</code>
Purpose	To convert a value in TI floating-point format to IEEE floating-point format.
Parameters	ti_32 Specifies the value in TI floating-point format.
Return value	This function returns the value in IEEE floating-point format.
Related topics	References

ds1006_conv32_ieee_to_ti.....	236
RTLIB_CONV_FLOAT32_TO_TI32.....	240
Standard Macros.....	242

RTLIB_CONV_FLOAT32_TO_IEEE32

Syntax	<code>UInt32 RTLIB_CONV_FLOAT32_TO_IEEE32(Float32 val_32)</code>
Include file	<code>dsstd.h</code>
Purpose	To convert a value in native floating-point format to IEEE floating-point format.
Parameters	val_32 Specifies the value in float32 format.
Return value	This function returns the value in IEEE floating-point format.

Related topics**References**

RTLIB_CONV_FLOAT32_FROM_IEEE32.....	238
Standard Macros.....	242

RTLIB_CONV_FLOAT_TO_SATURATED_INT32

Syntax

```
Int32 RTLIB_CONV_FLOAT_TO_SATURATED_INT32(double fp_value)
```

Include file

`dsstd.h`

Purpose

To convert a value in floating-point format to signed integer format.

Parameters

fp_value Specifies the value in floating-point format (float or double).

Return value

This function returns the value in signed integer format, possibly saturated.

Related topics**References**

RTLIB_CONV_FLOAT32_FROM_IEEE32.....	238
RTLIB_CONV_FLOAT32_TO_IEEE32.....	239
Standard Macros.....	242

RTLIB_CONV_FLOAT32_TO_TI32

Syntax

```
UInt32 RTLIB_CONV_FLOAT32_TO_TI32(Float32 ieee_32)
```

Include file

`dsstd.h`

Purpose

To convert a value in IEEE floating-point format to TI floating-point format.

Parameters	ieee_32 Specifies the value in IEEE floating-point format.
Return value	This function returns the value in TI floating-point format.
Related topics	<div>References<div><div><div><div>ds1006_conv32_ti_to_ieee.....</div><div>237</div></div><div><div>RTLIB_CONV_FLOAT32_FROM_TI32.....</div><div>239</div></div><div><div>Standard Macros.....</div><div>242</div></div></div></div></div>

Standard Macros

Introduction

The include file `dsstd.h` defines several macros that can be used to program board-independent applications. You can find further information about the functionality of a macro either in this topic or at the description of the corresponding function.

Initialization

The board-dependent initialization routine can be replaced by a macro valid for all systems.

Macro	Refer to ...
<code>init</code>	init() on page 244

Application background

The include file `dsstd.h` defines a macro, which can be used to start all board-specific background functions. There are also standard functions for calling hook functions, which shall run in the background of the application.

Macro	Refer to ...
<code>RTLIB_BACKGROUND_SERVICE</code>	RTLIB_BACKGROUND_SERVICE on page 26
<code>rtlib_background_hook</code>	rtlib_background_hook on page 27
<code>rtlib_background_hook_process</code>	rtlib_background_hook_process on page 28

End of application

The include file `dsstd.h` defines a macro, which can be used to release all I/O errors after running the application.

Macro	Refer to ...
<code>RTLIB_EXIT</code>	RTLIB_EXIT on page 245

Reading the board's serial number

The include file `dsstd.h` defines a macro, which you can use to get the serial number of your board.

Macro	Refer to ...
<code>RTLIB_GET_SERIAL_NUMBER</code>	RTLIB_GET_SERIAL_NUMBER() on page 245

Interrupt handling

The include file `dsstd.h` defines macros, which can be used to enable or disable the interrupts globally.

Macro	Refer to ...
<code>RTLIB_INT_ENABLE</code>	RTLIB_INT_ENABLE on page 113
<code>RTLIB_INT_DISABLE</code>	RTLIB_INT_DISABLE on page 112

Sampling rate timer

The DS1006 controller board uses Timer A as the sampling rate timer. The include file `dsstd.h` defines macros to handle this sampling rate timer:

Macro	Refer to ...
<code>RTLIB_SRT_START</code>	ds1006_start_isr_timerA on page 89
<code>RTLIB_SRT_PERIOD</code>	ds1006_timerA_period_reload on page 69
<code>RTLIB_SRT_ISR_BEGIN</code>	ds1006_begin_isr_timerA on page 85
<code>RTLIB_SRT_ISR_END</code>	ds1006_end_isr_timerA on page 87
<code>RTLIB_SRT_ENABLE</code>	RTLIB_SRT_ENABLE on page 116
<code>RTLIB_SRT_DISABLE</code>	RTLIB_SRT_DISABLE on page 115

Time interval measurement

There are macros to be used for time interval measurement.

- [RTLIB_TIC_START](#) on page 49
- [RTLIB_TIC_READ](#) on page 48
- [RTLIB_TIC_READ_TOTAL](#) on page 49
- [RTLIB_TIC_HALT](#) on page 47
- [RTLIB_TIC_CONTINUE](#) on page 42
- [RTLIB_TIC_DELAY](#) on page 44
- [RTLIB_TIC_COUNT](#) on page 43
- [RTLIB_TIC_DIFF](#) on page 45
- [RTLIB_TIC_ELAPSED](#) on page 46

Floating-point conversion

The include file `dsstd.h` defines macros, which should be used when converting floating-point values transferred from or to a TI slave processor of an I/O Board:

Macros	Refer to ...
<code>RTLIB_CONV_FLOAT32_TO_TI32</code>	ds1006_conv32_ieee_to_ti on page 236
<code>RTLIB_CONV_FLOAT32_FROM_TI32</code>	ds1006_conv32_ti_to_ieee on page 237
<code>RTLIB_CONV_FLOAT32_TO_IEEE32</code>	RTLIB_CONV_FLOAT32_TO_IEEE32 on page 239
<code>RTLIB_CONV_FLOAT32_FROM_IEEE32</code>	RTLIB_CONV_FLOAT32_FROM_IEEE32 on page 238
<code>RTLIB_CONV_FLOAT_TO_SATURATED_INT32</code>	RTLIB_CONV_FLOAT_TO_SATURATED_INT32 on page 240

Memory allocation

The include file `dsstd.h` defines macros to handle memory allocation that is protected against interrupt activities.

Macros	Refer to ...
<code>RTLIB_MALLOC_PROT</code>	RTLIB_MALLOC_PROT on page 246
<code>RTLIB_CALLOC_PROT</code>	RTLIB_CALLOC_PROT on page 246
<code>RTLIB_REALLOC_PROT</code>	RTLIB_REALLOC_PROT on page 247
<code>RTLIB_FREE_PROT</code>	RTLIB_FREE_PROT on page 247

Processor functions

The include file `dsstd.h` defines macros to handle the following Assembler commands.

Macros	Meaning
<code>RTLIB_FORCE_IN_ORDER</code>	RTLIB_FORCE_IN_ORDER on page 234
<code>RTLIB_SYNC</code>	RTLIB_SYNC on page 235

init()

Purpose

To initialize the required hardware and software modules for a specific hardware system.

Syntax

```
void init(void)
```

Include file

`dsstd.h`

Description

This macro calls the internal initialization functions of the hardware system.

Note

- I/O boards used within a PHS-bus-based system, like DS1006, or DS1007 are not initialized by calling `init()`.
- The initialization function `init()` must be executed at the beginning of each application. It can only be invoked once. Further calls to `init()` are ignored.
- When you use RTI, this function is called automatically in the simulation engine. Hence, you do not need to call `init()` in S-functions. If you need to initialize single components that are not initialized by `init()`, use the specific initialization functions that are described at the beginning of the function references.

RTLIB_EXIT

Purpose

To exit the application by using the `exit` routine of the standard C library.

Syntax

```
void RTLIB_EXIT(int value)
```

Include file

`dsstd.h`

Parameters

value Specifies the return value (has no effect).

RTLIB_GET_SERIAL_NUMBER()

Purpose

To get the serial number of the processor board.

Syntax

```
RTLIB_GET_SERIAL_NUMBER()
```

Include file

`dsstd.h`

Description

This macro returns the serial number as UInt32 data type.

RTLIB_MALLOC_PROT

Purpose To allocate memory with protection against interrupts by using the `malloc` routine of the standard C library.

Syntax `RTLIB_MALLOC_PROT(void *pointer, UInt32 size)`

Include file `dsstd.h`

Parameters

pointer Specifies the address of the allocated buffer.

size Specifies the memory size to be allocated.

Related topics

References

RTLIB_CALLOC_PROT	246
RTLIB_FREE_PROT	247
RTLIB_REALLOC_PROT	247

RTLIB_CALLOC_PROT

Purpose To allocate memory for an array with protection against interrupts by using the `calloc` routine of the standard C library.

Syntax `RTLIB_CALLOC_PROT(void *pointer, UInt32 nobj, UInt32 size)`

Include file `dsstd.h`

Parameters

pointer Specifies the address of the allocated buffer.

nobj Specifies the number of elements.

size Specifies the size of one element.

Related topics

References

RTLIB_FREE_PROT	247
RTLIB_MALLOC_PROT	246
RTLIB_REALLOC_PROT	247

RTLIB_REALLOC_PROT

Purpose

To change the memory size with protection against interrupts by using the `realloc` routine of the standard C library.

Syntax

```
RTLIB_REALLOC_PROT(void *pointer, UInt32 size)
```

Include file

`dsstd.h`

Parameters

pointer Specifies the address of the allocated buffer.
size Specifies the memory size to be allocated.

Related topics

References

RTLIB_CALLOC_PROT	246
RTLIB_FREE_PROT	247
RTLIB_MALLOC_PROT	246

RTLIB_FREE_PROT

Purpose

To free the allocated memory with protection against interrupts by using the `free` routine of the standard C library.

Syntax

```
RTLIB_FREE_PROT(void *pointer)
```

Include file

`dsstd.h`

Parameters	pointer Specifies the address of the buffer to be freed.
-------------------	---

Related topics

References

RTLIB_CALLOC_PROT.....	246
RTLIB_MALLOC_PROT.....	246
RTLIB_REALLOC_PROT.....	247

Function Execution Times

Where to go from here	Information in this section
	Information on the Test Environment..... 249
	Measured Execution Times..... 250

Information on the Test Environment

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 and the usage of inlining
- Parameters used

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 `downxxx` tool (optimization and inlining). The execution times in the tables below are always the mean measurement values.

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.

The properties of the processor boards used are:

CPU clock	2.6 GHz / 3.0 GHz
Bus clock	133 MHz
Local RAM size	256 MB
Global RAM size	128 MB

Measured Execution Times

Timer A

The following execution times have been measured with the functions for Timer A:

Function	Execution Time (in μ s)	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
ds1006_timerA_period_set	0.043	0.032
ds1006_timerA_read	1.1	1.04
ds1006_timerA_start	0.8	1.03
ds1006_timerA_stop	0.77	1.02

Timer B

The following execution times have been measured with the functions for Timer B:

Function	Execution Time (in μ s)	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
ds1006_timerB_init	1.3	1.64
ds1006_timerB_compare_set	0.76	1.03
ds1006_timerB_compare_set_periodically	0.76	1.03
ds1006_timerB_read	0.6	0.5
ds1006_timerB_start	0.82	1.01
ds1006_timerB_stop	0.76	1.01

Timer D

The following execution times have been measured with the functions for Timer D:

Function	Execution Time (in μ s)	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
ds1006_timerD_period_set	0.043	0.032
ds1006_timerD_read	1.1	0.5
ds1006_timerD_start	0.8	1.01
ds1006_timerD_stop	0.77	1.02

Time stamping

The following execution times have been measured with the functions for time stamping:

Function	Execution Time (in μ s)	
	Single Mode	Multi Mode
ts_init	2.50	4.30
ts_timestamp_read	0.04	0.97
ts_time_read	0.07	1.00
ts_reset	0.04	1.01
ts_timestamp_compare	0.02	0.02
ts_timestamp_interval	0.06	0.06
ts_time_offset	0.17	0.18
ts_timestamp_offset	0.09	0.09
ts_time_calculate	0.04	0.04
ts_timestamp_calculate	0.10	0.10

Interrupt handling

The following execution times have been measured with the functions for interrupt handling:

Function	Execution Time (in μ s)	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
ds1006_set_interrupt_vector	0.2	0.015
ds1006_get_interrupt_vector	0.15	0.015
ds1006_set_interrupt_vector	0.04	0.017
ds1006_get_interrupt_status	0.02	0.016
ds1006_enable_hardware_int	0.1	0.023
ds1006_disable_hardware_int	0.12	0.023
ds1006_reset_interrupt_flag	0.02	0.017
ds1006_get_interrupt_flag	0.6	0.5
RTLIB_INT_ENABLE	0.03	0.023
RTLIB_INT_DISABLE	0.02	0.016
ds1006_enable_hardware_int_bm	0.03	0.023
ds1006_disable_hardware_int_bm	0.03	0.023
ds1006_get_interrupt_flag_bm	1.05	1.1
ds1006_reset_interrupt_flag_bm	0.02	0.017

Information handling

The following execution times have been measured with the functions for information handling:

Function	Execution Time (in μ s)	
	DS1006 with 2.6 GHz	DS1006 with 3.0 GHz
ds1006_info_board_version_get	1.9	1.9
ds1006_info_memory_get	0.7	0.44
ds1006_info_clocks_get	1.0	0.85
ds1006_info_cpu_temperature_get	2114.0	2103

Message handling

The following execution times have been measured with the functions for message handling:

Function	Execution Time (in μ s)
msg_info_set	6.9
msg_warning_set	6.9
msg_error_set	6.9
msg_info_printf	6.9
msg_warning_printf	6.9
msg_error_printf	6.9
msg_last_error_number	0.02
msg_last_error_submodule	0.02
msg_mode_set	2.5
msg_reset	0.02
msg_error_hook_set	0.02
msg_error_clear	0.02

I/O Modules

Introduction	A PHS-bus-based system consists of a processor board and one or more I/O boards. Here you get an overview on the available I/O boards for a DS1006 modular system and the functions to handle the PHS bus that connects the processor board with the installed I/O boards.
--------------	--

Where to go from here

Information in this section

I/O Boards.....	254
PHS-Bus Handling.....	258
PHS-Bus Interrupt Handling.....	270

Information in other sections

Conversion Functions.....	236
Special Processor Functions.....	234

I/O Boards

Where to go from here







Information in this section

A/D Conversion.....	254
D/A Conversion.....	255
Automotive Signal Generation and Measurement.....	255
Bit I/O.....	255
Timing I/O.....	256
Interface Boards.....	256
Special I/O.....	257
Integration of FPGA Applications.....	257

A/D Conversion

Board overview






The following dSPACE boards can be controlled by the DS1006 board to perform A/D conversion. See:

- [DS2001 RTLib Reference](#) 
for the DS2001 High-Speed A/D Converter Board
- [DS2002 RTLib Reference](#) 
for the DS2002 Multi-Channel A/D Converter Board
- [DS2003 RTLib Reference](#) 
for the DS2003 Multi-Channel A/D Converter Board
- [DS2004 RTLib Reference](#) 
for the DS2004 High-Speed A/D Converter Board
- [DS2201 RTLib Reference](#) 
for the DS2201 Multi-I/O Board
- [DS2202 RTLib Reference](#) 
for the DS2202 HIL I/O Board

D/A Conversion

Board overview





The following dSPACE boards can be controlled by the DS1006 board to perform D/A conversion. See:

- [DS2101 RTLib Reference](#) 
for the DS2101 D/A Converter Board
- [DS2102 RTLib Reference](#) 
for the DS2102 High-Resolution D/A Converter Board
- [DS2103 RTLib Reference](#) 
for the DS2103 Multi-Channel D/A Converter Board
- [DS2201 RTLib Reference](#) 
for the DS2201 Multi-I/O Board
- [DS2202 RTLib Reference](#) 
for the DS2202 HIL I/O Board

Automotive Signal Generation and Measurement

Board overview



The following dSPACE boards can be controlled by the DS1006 board for autonomous signal generation. See:






- [DS2202 RTLib Reference](#) 
for the DS2202 HIL I/O Board
- [DS2210 RTLib Reference](#) 
for the DS2210 HIL I/O Board
- [DS2211 RTLib Reference](#) 
for the DS2211 HIL I/O Board
- [DS2302 RTLib Reference](#) 
for the DS2302 Direct Digital Synthesis Board

Bit I/O

Board overview

The following dSPACE boards can be controlled by the DS1006 board to perform bit I/O. See:








- [DS2201 RTLib Reference](#) 
for the DS2201 Multi-I/O Board
- [DS2202 RTLib Reference](#) 
for the DS2202 HIL I/O Board

- [DS2301 RTLib Reference](#) 
for the DS2301 Direct Digital Synthesis Board
- [DS2302 RTLib Reference](#) 
for the DS2302 Direct Digital Synthesis Board
- [DS4001 RTLib Reference](#) 
for the DS4001 Timing and Digital I/O Board
- [DS4002 RTLib Reference](#) 
for the DS4002 Timing and Digital I/O Board
- [DS4003 RTLib Reference](#) 
for the DS4003 Digital I/O Board

Timing I/O

Board overview


The following dSPACE boards can be controlled by the DS1006 board to perform timing I/O, such as the generation of various pulse patterns including PWM or the capture of digital frequency signals. See:






- [DS2201 RTLib Reference](#) 
for the DS2201 Multi-I/O Board
- [DS2301 RTLib Reference](#) 
for the DS2301 Direct Digital Synthesis Board
- [DS2302 RTLib Reference](#) 
for the DS2302 Direct Digital Synthesis Board
- [DS4001 RTLib Reference](#) 
for the DS4001 Timing and Digital I/O Board
- [DS4002 RTLib Reference](#) 
for the DS4002 Timing and Digital I/O Board
- [DS5001 RTLib Reference](#) 
for the DS5001 Digital Waveform Capture Board
- [DS5101 RTLib Reference](#) 
for the DS5101 Digital Waveform Output Board

Interface Boards

Board overview

The following dSPACE boards can be controlled by the DS1006 board to integrate more specialized custom devices into the dSPACE real-time system. See:




- [DS3001 RTLib Reference](#) 
for the DS3001 Incremental Encoder Interface Board

- [DS3002 RTLib Reference](#) 
for the DS3002 Incremental Encoder Interface Board
- [DS4201 RTLib Reference](#) 
for the DS4201 Prototyping Board
- [DS4201-S RTLib Reference](#) 
for the DS4201-S Serial Interface Board
- [DS4302 RTLib Reference](#) 
for the DS4302 CAN Interface Board
- [DS4330 RTLib Reference](#) 
for the DS4330 LIN Interface Board

Special I/O

Board overview

The following dSPACE boards can be controlled by the DS1006 board to perform more specialized I/O. See:

- [DS2301 RTLib Reference](#) 
for the DS2301 Direct Digital Synthesis Board
- [DS2302 RTLib Reference](#) 
for the DS2302 Direct Digital Synthesis Board
- [DS2401 RTLib Reference](#) 
for the DS2401 Resistive Sensor Simulation Board

Integration of FPGA Applications

Board overview

The following dSPACE board can be controlled by the DS1006 board to perform custom FPGA applications. See:

- [DS5202 RTLib Reference](#) 
for the DS5202 FPGA Base Board
- [DS5203 RTLib Reference](#) 
for the DS5203 FPGA Board

PHS-Bus Handling

Introduction

Use these functions to handle the PHS bus, which is used for communication between the DS1006 processor board and the I/O boards.

I/O board base address

When using I/O board functions you always need the board's base address as parameter. This address can simply be obtained by using the `DSXXXX_n_BASE` macros where `DSXXXX` is the board name (e.g., DS2001) and `n` is an index which counts boards of the same type. The board with the lowest base address gets the index 1. The other boards of the same type get the consecutive numbers in order of their base addresses.

The macros refer to an internal data structure, which holds the addresses of all I/O boards in the system. This data structure is created during the initialization phase. Hence, when changing an I/O board base address, it is not necessary to recompile the code of your application.

Note

The `DSXXXX_n_BASE` macros can only be used after the initialization function was called.

Example

This example demonstrates the using of the `DSXXXX_n_BASE` macros. There are two DS2001 boards, two DS2101 boards and one DS2002 board connected to a PHS bus. Their base addresses have been set to distinct addresses. The following table shows the I/O boards, their base addresses and the macros which can be used as base address.

Board	Base address (Hex)	Macro
DS2001	00	DS2001_1_BASE
DS2002	20	DS2002_1_BASE
DS2101	80	DS2101_1_BASE
DS2001	90	DS2001_2_BASE
DS2101	A0	DS2101_2_BASE

Programmable signals

Three of PHS-bus signals are programmable by the user:

I/O error line The DS1006 processor board and the I/O boards can activate the I/O error line if an error occurred. Thus, the other devices are able to react individually.

SYNCIN line A pulse in the SYNCIN line triggers the input channels of all I/O boards to sample their input values.

SYNCOOUT line A pulse in the SYNCOOUT line triggers the output channels of all I/O boards to update their input values.

Where to go from here

Information in this section

To get information about an I/O board

[get_peripheral_addr](#)..... 260

To get the base address of an I/O board specified by board ID and board number.

[PHS_BOARD_BASE_GET](#)..... 261

To get the base address of an I/O board specified by board type and board number.

[phs_board_type_get](#)..... 261

To identify the type of an I/O board at the specified PHS-bus address.

[phs_board_type_from_slot_get](#)..... 262

To identify the type of an I/O board at the specified PHS-bus slot number.

To handle the PHS-bus register

[PHS_REGISTER_READ](#)..... 263

To read from a specified register.

[PHS_REGISTER_WRITE](#)..... 264

To write to a specified register.

[PHS_REGISTER_PTR](#)..... 265

To get the address of a specified register.

To handle the I/O error line

[PHS_IO_ERROR_STATE](#)..... 265

To get the state of the I/O error line.

[PHS_IO_ERROR_SET](#)..... 266

To activate or deactivate the I/O error line.

To handle the SYNCIN and SYNCOUT line

[PHS_SYNCIN_TRIGGER.....267](#)

To generate a pulse on the PHS-bus SYNCIN line.

[PHS_SYNCOUT_TRIGGER.....267](#)

To generate a pulse on the PHS-bus SYNCOUT line.

[PHS_SYNC_TRIGGER.....268](#)

To generate a pulse on the PHS-bus SYNCIN and the SYNCOUT line simultaneously.

[PHS_SYNC_TIMER_SET.....268](#)

To select a trigger source for the SYNCIN or SYNCOUT lines.

get_peripheral_addr

Syntax

```
phs_addr_t get_peripheral_addr(
    UInt32 board_id,
    int board_no)
```

Include file

dsphs.h

Purpose

To get the base address of an I/O board specified by board ID and board number.

Parameters

board_id Specifies the ID of the I/O board. For each I/O board, there is a symbol predefined as **DSxxxx_BOARD_ID**, where DSxxxx stands for the board name. For example, to get the base address of a DS2210 I/O board, you must specify **DS2210_BOARD_ID** for this parameter.

board_no Specifies the board number that distinguishes boards with the same **board_id**.

Return value

This function returns the I/O board base address or 0xFFFFFFFF if the I/O board could not be found in the PHS bus.

PHS_BOARD_BASE_GET

Syntax

```
phs_addr_t PHS_BOARD_BASE_GET(
    int board_type,
    int board_no)
```

Include file

dsphs.h

Purpose

To get the base address of an I/O board specified by board type and board number.

Parameters

board_type Specifies the type number of the I/O board. The following symbols are predefined:

Predefined Symbol	Meaning
PHS_BT_NO_BOARD	No I/O board at the specified PHS-bus address
PHS_BT_UNKNOWN	I/O board type is not known
PHS_BT_INVALID_BASE	Specified PHS-bus address is not valid
PHS_BT_DSxxxx	Specifies the board type, where DSxxxx stands for the board name. For example, you must use PHS_BT_DS2210 to specify the board type of a DS2210 I/O board.

board_no Specifies the board number that distinguishes boards with the same **board_id**.

Return value

This function returns the I/O board base address or **PHS_INVALID_BASE** if the I/O board could not be found in the PHS bus.

Related topics

References

[phs_board_type_get](#)..... 261

phs_board_type_get

Syntax

```
int phs_board_type_get(phs_addr_t base)
```

Include file `dsphs.h`

Purpose To identify the type of an I/O board at the specified PHS-bus address.

Parameters **base** Specifies the I/O board base address, refer to [PHS-Bus Handling](#) on page 258.

Return value This function returns the I/O board type or error code. The following symbols are predefined:

Predefined Symbol	Meaning
PHS_BT_NO_BOARD	No I/O board at the specified PHS-bus address
PHS_BT_UNKNOWN	I/O board type is not known
PHS_BT_INVALID_BASE	Specified PHS-bus address is not valid
PHS_BT_DSxxxx	Returned board type, where DSxxxx stands for the board name. For example, the function returns <code>PHS_BT_DS2210</code> for a DS2210 I/O board.

Related topics

References

[PHS_BOARD_BASE_GET](#)..... 261

phs_board_type_from_slot_get

Syntax `int phs_board_type_from_slot_get(int slot_number)`

Include file `dsphs.h`

Purpose To identify the type of an I/O board at the specified PHS-bus slot number.

Parameters **slot_number** Specifies the PHS-bus slot number in the range of 0 ... 15.

Return value

This function returns the I/O board type or error code. The following symbols are predefined:

Predefined Symbol	Meaning
PHS_BT_NO_BOARD	No I/O board at the specified PHS-bus address
PHS_BT_UNKNOWN	I/O board type is not known
PHS_BT_INVALID_BASE	Specified PHS-bus address is not valid
PHS_BT_DSxxxx	Returned board type, where DSxxxx stands for the board name. For example, the function returns PHS_BT_DS2210 for a DS2210 I/O board.

Related topics**References**

[phs_board_type_get](#)..... 261

PHS_REGISTER_READ

Syntax

```
phs_data_u_t PHS_REGISTER_READ(
    phs_data_u_t base,
    phs_data_u_t offset)
```

Include file

`dsphs.h`

Purpose

To read a value from the register at the specified offset from the specified PHS-bus board base address.

Parameters

base Specifies the I/O board base address, refer to [PHS-Bus Handling](#) on page 258.

offset Specifies the offset for the register address within the range 0x00 ... 0x0F.

Return value

This function returns the contents of the specified PHS-bus register.

Related topics**References**

PHS_REGISTER_PTR.....	265
PHS_REGISTER_WRITE.....	264

PHS_REGISTER_WRITE

Syntax

```
void PHS_REGISTER_WRITE(  
    phs_data_u_t base,  
    phs_data_u_t offset,  
    phs_data_u_t value)
```

Include file

dsphs.h

Purpose

To write a value to the register at the specified offset from the specified PHS-bus board base address.

Parameters

base Specifies the I/O board base address, refer to [PHS-Bus Handling](#) on page 258.

offset Specifies the offset for the register address within the range 0x00 ... 0x0F.

value Specifies the value to be written to the specified PHS-bus register.

Return value

None

Related topics**References**

PHS_REGISTER_PTR.....	265
PHS_REGISTER_READ.....	263

PHS_REGISTER_PTR

Syntax	<pre>phs_data_u_t* PHS_REGISTER_PTR(phs_data_u_t base, phs_data_u_t offset)</pre>				
Include file	dsphs.h				
Purpose	To get the register address at the specified offset from the specified PHS-bus board base address.				
Parameters	<p>base Specifies the I/O board base address, refer to PHS-Bus Handling on page 258.</p> <p>offset Specifies the offset for the register address within the range 0x00 ... 0x0F.</p>				
Return value	This function returns the address of the register.				
Related topics	<p>References</p> <table> <tr> <td>PHS_REGISTER_READ.....</td> <td>263</td> </tr> <tr> <td>PHS_REGISTER_WRITE.....</td> <td>264</td> </tr> </table>	PHS_REGISTER_READ	263	PHS_REGISTER_WRITE	264
PHS_REGISTER_READ	263				
PHS_REGISTER_WRITE	264				

PHS_IO_ERROR_STATE

Syntax	PHS_IO_ERROR_STATE()
Include file	dsphs.h
Purpose	To get the state of the I/O error line.

Return value

This macro returns the state of the I/O error line:

Value	Meaning
TRUE	I/O error line is active
FALSE	I/O error line is not active

Related topics**References**

[PHS_IO_ERROR_SET..... 266](#)

PHS_IO_ERROR_SET

Syntax

`PHS_IO_ERROR_SET(state)`

Include file

`dsphs.h`

Purpose

To activate or deactivate the I/O error line.

Parameters

state Specifies the state of the I/O error line:

Value	Meaning
TRUE	Activate I/O error line
FALSE	Deactivate I/O error line

Return value

None

Related topics**References**

[PHS_IO_ERROR_STATE..... 265](#)

PHS_SYNCIN_TRIGGER

Syntax

```
PHS_SYNCIN_TRIGGER()
```

Include file

```
dsphs.h
```

Purpose

To generate a pulse on the PHS-bus SYNCIN line.

Return value

None

Related topics

References

PHS_SYNC_TRIGGER.....	268
PHS_SYNCOUT_TRIGGER.....	267

PHS_SYNCOUT_TRIGGER

Syntax

```
PHS_SYNCOUT_TRIGGER()
```

Include file

```
dsphs.h
```

Purpose

To generate a pulse on the PHS-bus SYNCOUT line.

Return value

None

Related topics

References

PHS_SYNC_TRIGGER.....	268
PHS_SYNCIN_TRIGGER.....	267

PHS_SYNC_TRIGGER

Syntax

```
PHS_SYNC_TRIGGER()
```

Include file

```
dsphs.h
```

Purpose

To generate a pulse on the PHS-bus SYNCIN and the SYNCOUT line simultaneously.

Return value

None

Related topics

References

PHS_SYNCIN_TRIGGER.....	267
PHS_SYNCOUT_TRIGGER.....	267

PHS_SYNC_TIMER_SET

Syntax

```
PHS_SYNC_TIMER_SET(mode)
```

Include file

```
dsphs.h
```

Purpose

To select Timer A or Timer B as the trigger source for the SYNCIN or SYNCOUT lines.

Parameters

mode Specifies the mode to be set. Combine the following predefined symbols with the logical operator OR:

Predefined Symbol	Meaning
PHS_SYNCIN_DISABLE	SYNCIN line disabled
PHS_SYNCIN_TIMER_A	SYNCIN line triggered by Timer A
PHS_SYNCIN_TIMER_B	SYNCIN line triggered by Timer B
PHS_SYNCOUT_DISABLE	SYNCOUT line disabled

Predefined Symbol	Meaning
PHS_SYNCOUT_TIMER_A	SYNCOUT line triggered by Timer A
PHS_SYNCOUT_TIMER_B	SYNCOUT line triggered by Timer B

Return value None

Related topics

References

PHS_SYNC_TRIGGER.....	268
PHS_SYNCIN_TRIGGER.....	267
PHS_SYNCOUT_TRIGGER.....	267

PHS-Bus Interrupt Handling

PHS bus

The PHS bus (Peripheral High Speed Bus) is used in dSPACE systems to connect I/O and processor boards. The PHS bus supports an extended interrupt system, which adds 64 external interrupts to the standard interrupts recognized by the Real-Time Processor (RTP). The following topics describe the PHS-bus interrupt functions, which provide an easy-to-use, high-level programmer's interface to PHS-bus interrupts. With this software the operation of the extended interrupt system becomes completely transparent to the user.

Note

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

Where to go from here

Information in this section

General Information on PHS-Bus Interrupts.....	270
Example of PHS-Bus Interrupt Handling.....	277
PHS-Bus Interrupt Functions.....	282
Troubleshooting for PHS-Bus Interrupt Handling.....	295

General Information on PHS-Bus Interrupts

Where to go from here

Information in this section

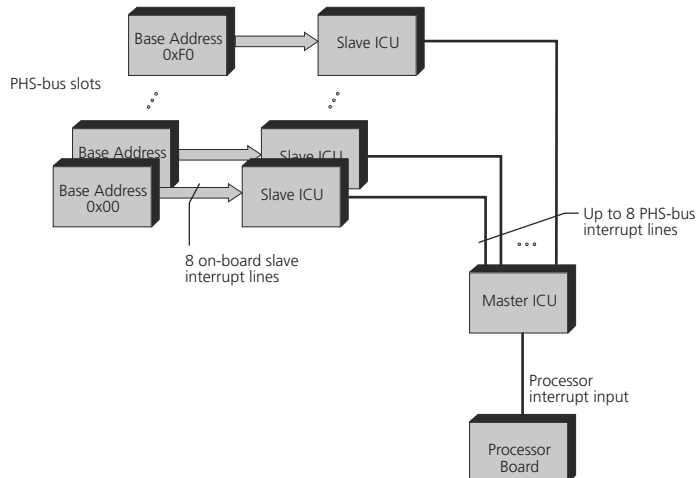
Basics of PHS-Bus Interrupts.....	271
Management of the Extended Interrupt System.....	272
Board Identification and PHS-Bus Interrupt Line Programming.....	272
PHS-Bus Interrupt Processing.....	274
Interrupt Priorities.....	275
How to Program PHS-Bus Interrupts.....	276

Basics of PHS-Bus Interrupts

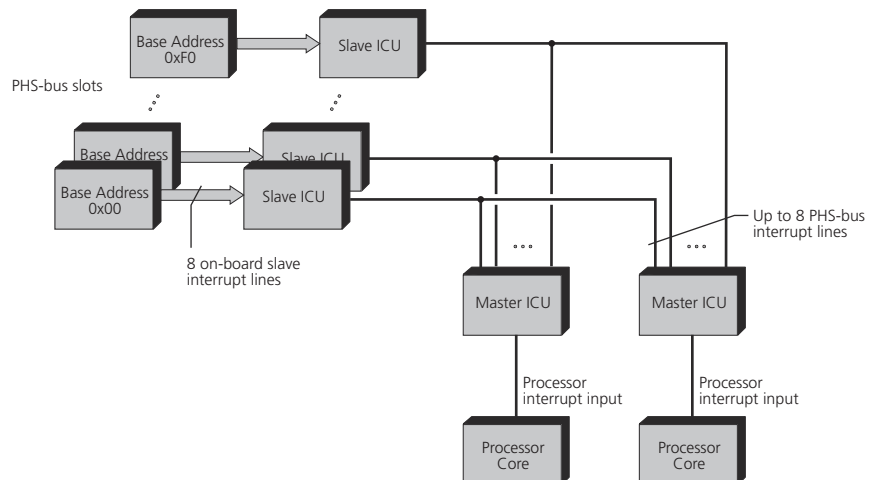
PHS-bus interrupt system

The following illustrations show the extended PHS-bus interrupt system.

Using a single-core processor board.



Using a multicore processor board.



Interrupt control unit

The processor board is equipped with one master interrupt control unit (ICU) per core. The master ICUs are separately configured and can issue interrupts only to the related CPU core. Each master ICU is connected to the eight interrupt lines of the PHS bus (see illustration above). An application always runs on one CPU core only and therefore it uses only the ICU that corresponds to the CPU core. All interrupt requests from these interrupt lines are mapped to an interrupt pin of the processor, one pin per CPU core. I/O boards with interrupt generating devices are provided with on-board slave interrupt controller units. The eight slave interrupt lines supported by each of the interrupt controllers are wired to the various board-specific interrupt sources, such as A/D converters, timers, etc. Each

interrupt controller can be connected to one of the eight PHS-bus interrupt lines by programming three bits in the setup register of the respective I/O board. Thus, a maximum of 64 prioritized PHS-bus interrupts is added to the processor's interrupt system. For more information on the extended PHS-bus interrupt system, refer to the hardware reference manual of your processor board.

The standard initialization procedure provided by the Real-Time Library (RTLib) initializes the slave interrupt controllers for polling mode with all I/O interrupts disabled. The master interrupt controller on the processor board is not initialized. PHS-bus interrupts are disabled in the interrupt enable register of the processor.

Note

Initially, all slave interrupt controllers are connected to interrupt line 0 on the PHS bus. This is the default interrupt line number, which is reserved for I/O boards operated in polling mode.

In order to handle PHS-bus interrupts, the PHS-bus interrupt functions will both initialize the master ICU and reinitialize the slave ICUs that are selected for interrupt mode.

Please note that it is possible to use mixed-mode operation of several peripheral boards. Some boards may be operated in interrupt mode while others may use the standard polling mode.

Management of the Extended Interrupt System

Managing the interrupt system

The slave interrupt controller on a dSPACE I/O board can be connected to one of eight PHS-bus interrupt lines by programming its setup register. For more information, refer to [Board Identification and PHS-Bus Interrupt Line Programming](#) on page 272.

All PHS-bus interrupts are mapped to an interrupt input of the processor. How the PHS-bus interrupts are processed is explained in [PHS-Bus Interrupt Processing](#) on page 274.

The processor provides the means for hardware prioritization of the standard interrupts. For more information, refer to [Interrupt Priorities](#) on page 275.

Board Identification and PHS-Bus Interrupt Line Programming

Introduction

The slave interrupt controller on a dSPACE I/O board can be connected to one of eight PHS-bus interrupt lines by programming three bits in the setup register of the board. The PHS-bus interrupt functions use the dSPACE board identification scheme to find out which I/O boards are connected to the PHS bus and how to

program the interrupt line numbers in the corresponding setup registers. The identification number consists of two 4-bit fields in the board's identification register (ID and SUB-ID field). The following table shows the relationship between board identification numbers and interrupt line programming.

Board ID	Board SUB-ID	Board Type	Interrupt Line Programming
0 ... 13	don't care	Standard dSPACE I/O board	Provided by PHS-bus interrupt functions
14	0 ... 15	Standard dSPACE I/O board	Provided by PHS-bus interrupt functions
15	0 ... 14	Customer I/O board with <ul style="list-style-type: none"> ▪ Static interrupt line ▪ Programmable interrupt line 	None (customer-provided)
15	15	No board	None

The ID and SUB-ID values of the standard I/O boards are defined by dSPACE. For standard I/O boards, the PHS-bus interrupt functions employ the built-in code for assigning and programming the interrupt line numbers. To install an interrupt handler, you only need to call `install_phs_int_vector`. To ensure proper operation, any previous setting of the interrupt lines is overridden. No part of the User-Code should directly modify the interrupt line numbers. Line number 0 is the default interrupt line and reserved for I/O boards that operate in polling mode. Thus, a maximum of seven PHS-bus boards can be used simultaneously in interrupt mode.

Customer-specific I/O boards

In addition to the standard I/O boards recognized by the PHS-bus interrupt functions, a dSPACE system may contain customer-specific I/O boards for which the automatic programming of the PHS-bus interrupt lines does not work. This holds in particular for the DS4201 prototyping board and for boards that are derived from it. (Nevertheless it is assumed that these nonstandard boards are provided with a standard PHS-bus interface including the slave interrupt controller.)

Some boards may use static PHS-bus interrupt lines that are defined by hardware settings. In this case you must call `declare_phs_int_line` for each of them. Other boards may be equipped with nonstandard setup registers, for which the customer has to provide the appropriate initialization code. `alloc_phs_int_line` must be called for these.

Note

Customer I/O boards must use ID 15 in order to be distinguishable from the standard dSPACE I/O boards. The SUB-ID must be in the range 0 ... 14. A customer I/O board with a SUB-ID of 15 cannot be detected.

PHS-Bus Interrupt Processing

PHS-bus interrupt vector table

All PHS-bus interrupts are mapped to an interrupt input of the processor core and thus serviced by a common master interrupt service routine. The address of this master interrupt handler is installed in the processor's interrupt vector table. The vector table is created and maintained by the standard processor run-time environment to handle the standard processor interrupts. We will refer to this vector table as the system interrupt vector table.

To handle peripheral board interrupts, an interrupt dispatcher is needed for switching to the appropriate slave interrupt service routine. As a maximum of 16 PHS-bus boards can be connected to a processor board, each providing a maximum of eight slave interrupt sources, a second interrupt vector table with 128 entries must be maintained. This vector table will be referred to as the PHS-bus interrupt vector table.

Whenever an I/O interrupt is requested, the corresponding slave interrupt controller generates an interrupt vector number by adding the number of the interrupt service to a predefined board interrupt vector offset. The PHS-bus interrupt functions automatically assign each I/O board its own vector offset dependent on the PHS-bus base address of the board. As shown in the table below, a maximum of eight interrupt sources is associated with each PHS-bus base address. Therefore, the board interrupt vector offset is set to 0 for the board at PHS-bus base address 0x00, 8 for the board at base address 0x10, etc. The master interrupt handler reads the resulting vector number as generated by the slave interrupt controller and calls the service routine, whose address is found in the corresponding entry of the PHS-bus interrupt vector table.

In order to install or uninstall interrupt handlers, the PHS-bus library functions need to perform complete initialization sequences of the corresponding interrupt controller chips. As a consequence, each call to `install_phs_int_vector` or `deinstall_phs_int_vector` will clear all pending PHS-bus interrupt requests.

The following table shows the PHS-bus interrupt vector table.

PHS-bus Base Address	Board Interrupt Vector Offset	Slave Interrupt Number	Resulting Interrupt Vector Number
0x00	0	0 ... 7	0 ... 7
0x10	8	0 ... 7	8 ... 15
0x20	16	0 ... 7	16 ... 23
...
0xF0	120	0 ... 7	120 ... 127

Interrupt Priorities

Introduction

The processor provides the means for hardware prioritization of the standard interrupts according to their positions in the system interrupt vector table. The closer an interrupt's vector is to the base address of the vector table, the higher its hardware priority. This prioritization, however, applies only when more than one interrupt request is received in the same clock cycle. In that case, the interrupt with the highest priority is serviced first. On acceptance of an interrupt, the corresponding interrupt service routine is called with all interrupts globally disabled in the processor status register. For more information on the interrupt system, refer to the related documentation of the processor hardware.

Prioritization

The PHS-bus interrupt controllers use a similar hardware prioritization scheme according to interrupt line numbers. After initialization, interrupt requests at interrupt line 0 are of highest priority, those at interrupt line 7 are of lowest priority. The initial prioritization of the interrupt lines can be changed dynamically by sending priority rotation commands to the chip. This feature is currently not used by dSPACE software, leaving interrupt line 0 at the highest priority level all the time. This holds for the master interrupt controller on the processor board as well as for the slave interrupt controllers on the I/O boards. Consequently, the slave interrupt source at interrupt line 0 of the I/O board connected to PHS-bus interrupt line 1 represents the external interrupt with the highest priority, while slave interrupt 7 of the board connected to PHS-bus interrupt line 7 is of lowest priority. Remember that PHS-bus interrupt line 0 is reserved for boards operated in polling mode.

This prioritization again applies when more than one interrupt request is received in the same clock cycle. In that case, the interrupt with the highest priority is serviced first. On acceptance of an interrupt, the corresponding interrupt service routine is called with all interrupts globally disabled in the processor status register. `alloc_phs_int_line` and `install_phs_int_vector` assign interrupt lines to I/O boards in ascending order. If hardware prioritization of the PHS-bus interrupts is of importance, you should carefully adjust the sequence of function calls for allocating PHS-bus interrupt lines and installing interrupt handlers.

Finished-Interrupt command

The PHS-bus interrupt functions operate the interrupt controllers in FI Command mode. A Finished-Interrupt (FI) command is issued by software to acknowledge an interrupt. You can flexibly realize a software prioritization scheme according to the specific needs of your application.

Note

In order to implement preemptable interrupt handlers, you will have to enable interrupts at the beginning of the interrupt service routine and disable them at the end of it. The macros `RTLIB_INT_ENABLE` and `RTLIB_INT_DISABLE` provided by the Real-Time Library can be used for this purpose.

It is necessary to disable interrupts globally before leaving an interrupt handler, because the context switch at the end of the interrupt service is a critical section that must not be interrupted.

Consider reentrancy and overrun problems, which may be encountered if interrupts are enabled within an interrupt handler.

How to Program PHS-Bus Interrupts

Instructions**Note**

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

To use the PHS-bus interrupt functions, the header file `brtenv.h` must be included in your source files. `brtenv.h` includes the header file for the PHS-bus interrupts functions `phsint.h`.

The following table summarizes the required sequences of function calls for different types of I/O boards.

Standard dSPACE I/O Board	Customer I/O Board with	
	Static Interrupt Line	Programmable Interrupt Line
	<code>declare_phs_int_line</code>	<code>alloc_phs_int_line</code>
	↓	↓
		board-specific interrupt line programming
		↓
<code>install_phs_int_vector</code>	<code>install_phs_int_vector</code>	<code>install_phs_int_vector</code>
↓	↓	↓
disable / enable macros (if required)	disable / enable macros (if required)	disable / enable macros (if required)
↓	↓	↓

Standard dSPACE I/O Board	Customer I/O Board with	
	Static Interrupt Line	Programmable Interrupt Line
deinstall_phs_int_vector	deinstall_phs_int_vector	deinstall_phs_int_vector ↓ free_phs_int_line ↓ board-specific interrupt line programming (set to default)

Unless otherwise specified, the functions return 0 on successful completion. As the functions are used to initialize the interrupt system, most of the errors must be considered severe programming faults, preventing the application from executing. An error message is generated and displayed within the dSPACE experiment software by means of the Real-Time Message Module. The error message contains the error code as well as a string describing the error and the name of the function where it occurred (see [Error Codes of PHS-Bus Interrupt Functions](#) on page 295). Program execution is automatically terminated by calling the `exit()` function.

Related topics

Basics	
Management of the Extended Interrupt System.....	272
Examples	
Example of PHS-Bus Interrupt Handling.....	277
References	
Error Codes of PHS-Bus Interrupt Functions.....	295

Example of PHS-Bus Interrupt Handling

Introduction

The PHS-bus interrupt functions are part of the Real-Time Library (RTLib) for your processor board. You can immediately try the demo application coming with the PHS-bus interrupt functions. A dSPACE system with a processor board, a DS2004 ADC board and a DS2101 DAC board are assumed.

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

Description of the Demo Application.....	278
PHS-Bus Interrupt Application.....	279
Example with Customer I/O Boards.....	280

Description of the Demo Application

Where to find the demo application

Change to
`<RCP_HIL_InstallationPath>\Demos\DS1006\IOBoards\Phsint.`

The example consists of a single source file `adint_1006_hc.c`. Use the **Down1006** utility available on your host computer to download the application to the processor board.

Description

A dSPACE system with a processor board, a DS2001 ADC board and a DS2101 DAC board are assumed. The Timer A of the processor board generates interrupts at a specified period. Each time an interrupt occurs, A/D conversion of the input signal provided to channel 1 of the DS2004 board is started. The A/D converter generates an data ready interrupt as soon as the conversion has finished. The digital input value is then read and output to D/A channel 1 of the DS2101 board. The operation can be checked by connecting a function generator to channel 1 of the DS2004 ADC board and an oscilloscope to channel 1 of the DS2101 DAC board.

Interrupt service routines

Two interrupt service routines are needed. The first serves the Timer A interrupts, the other handles the data ready interrupts of the DS2004 ADC board. The `main` function first calls the standard initialization procedure `init` provided by Real-Time Library. The I/O boards are then initialized and an info message is generated. After initialization of the timer, the interrupt handlers are installed by calling `install_phs_int_vector`. To indicate the board base addresses, we use the constants defined in the include file `brtenv.h`, which should reflect your actual hardware setup.

After successful installation of the interrupt service routines, interrupts are globally enabled and the program enters an infinite loop. Program execution is now completely interrupt-driven. The operation can be checked by connecting a function generator to channel 1 of the DS2004 ADC board and an oscilloscope to channel 1 of the DS2101 DAC board.

PHS-Bus Interrupt Application

Example source code

```

/*****
* FILE
*   adint_1006_hc.c
* DESCRIPTION
*   Interrupt driven A/D conversion
*   PHS-bus interrupt demo program using DS2004 and DS2101 interrupts
*   Connect DS2004 ADC channel 1 to a frequency generator, connect DS2101
*   DAC channel 1 to an oscilloscope.
*
* dSPACE GmbH, Technologiepark 25, 33100 Paderborn, Germany
*****/

#include <brtenv.h> /* basic run time environment */
#include <phsint.h> /* PHS-bus interrupt library */
#include <ds2004.h>
#include <ds2101.h>
#define DT 1.0e-4 /* 100 us simulation step size */

/*-----*/
void data_ready_service() /* DS2004 data ready interrupt service routine */
{
    UInt32 dummy;
    UInt32 pending;
    dsfloat value;

    do
    {
        /* read pending DS2004 interrupt */
        pending = ds2004_data_ready_int_pending(DS2004_1_BASE);
        if(pending)
        {
            if(pending == 1) /* interrupt from channel 1 */
            {
                /* read DS2004 ADC value */
                ds2004_single_read(DS2004_1_BASE, DS2004_CH1, &dummy, &value);
                /* write value to DS2101 DAC */
                ds2101_out(DS2101_1_BASE, DS2101_CH1, value);
            }
            else /* interrupt from other channel */
            {
                msg_warning_printf(0, 0, "Unexpected DS2004 INT from channel %d", pending);
            }
            /* acknowledge interrupt */
            ds2004_data_ready_int_ack(DS2004_1_BASE, pending);
        }
    } while(pending != 0);
}
/*-----*/
void timerA_interrupt(void) /* timer A interrupt service routine */
{
    RTLIB_SRT_ISR_BEGIN(); /* overload check */
    /* start conversion of DS2004 ADC 1 */
    ds2004_sw_trigger(DS2004_1_BASE, DS2004_CONV_START_1);
    RTLIB_SRT_ISR_END(); /* overload check */
}

```

```

/*-----*/
int main(void)
{
    init(); /* initialize hardware system */
    ds2004_init(DS2004_1_BASE); /* initialize DS2004 board */
    ds2101_init(DS2101_1_BASE); /* initialize DS2101 board */

    msg_info_set(MSG_SM_RTLIB, 0, "System started.");
    ds2004_single_init(DS2004_1_BASE, DS2004_CH1, DS2004_RNG_10, DS2004_TRIG_SW, 0.0);
    ds2004_data_ready_int_enable(DS2004_1_BASE, DS2004_CH1);
    /* install DS2004 service routine for INT1 (data ready interrupt) */
    install_phs_int_vector(DS2004_1_BASE, 1, data_ready_service);
    RTLIB_SRT_START(DT, timerA_interrupt); /* start sample rate timer */
    RTLIB_INT_ENABLE(); /* enable interrupts globally */

    while(1) /* model background loop */
    {
        RTLIB_BACKGROUND_SERVICE(); /* background service */
    }
}

```

Related topics

Examples

Description of the Demo Application.....	278
Example with Customer I/O Boards.....	280

Example with Customer I/O Boards

Example source code

```

/*****
 *
 * This is an example of using the PHS-bus interrupt functions in
 * combination with customer-specific I/O boards
 *
 *****/
#define CUSTOM_BASE .. /* PHS-bus base of customer I/O board */
#define INT_NUMBER .. /* slave interrupt on customer I/O board */
/* interrupt handler */
void isr()
{
    ..
}
/* main function */
main()
{
    int status, custom_line;
    init(); /* initialize interrupt system to default */
}

```



```

/* get a free interrupt line */
status = alloc_phs_int_line(CUSTOM_BASE);
if (status < 0)
{
    ..                                /* error handling */
}
else
    custom_line = status;
/* set interrupt line of customer I/O board with customer-provided
   initialization function */
custom_initialize(.., custom_line, ..);
/* install interrupt handler */
status = install_phs_int_vector(CUSTOM_BASE, INT_NUMBER, isr);
if (status != PHSINT_NO_ERROR)
{
    ..                                /* error handling */
}
/* background processing */
while (..)
{
    ..
    /* disable slave interrupt in critical sections */
    disable_phs_int(CUSTOM_BASE, INT_NUMBER);
    /* critical section */
    ..
    /* reenale slave interrupt */
    enable_phs_int(CUSTOM_BASE, INT_NUMBER);
    ..
}
/* deinstall interrupt handler on exit */
status = deinstall_phs_int_vector(CUSTOM_BASE, INT_NUMBER);
if (status != PHSINT_NO_ERROR)
{
    ..                                /* error handling */
}
/* release interrupt line */
status = free_phs_int_line(CUSTOM_BASE);
if (status < 0)
{
    ..                                /* error handling */
}
else
    custom_line = status; /* default interrupt line has been returned */
/* reset interrupt line of customer I/O board to default */
custom_initialize(.., custom_line, ..);
}

```

Related topics

Examples

PHS-Bus Interrupt Application..... 279

PHS-Bus Interrupt Functions

Where to go from here

Information in this section

install_phs_int_vector	282
To install a PHS-bus interrupt handler.	
deinstall_phs_int_vector	284
To uninstall a registered interrupt service routine.	
declare_phs_int_line	285
To declare a PHS-bus interrupt line.	
alloc_phs_int_line	286
To allocate PHS-bus interrupt lines for nonstandard I/O boards.	
free_phs_int_line	287
To free PHS-bus interrupt lines for nonstandard I/O boards.	
enable_phs_int	289
To separately enable a single I/O interrupt.	
disable_phs_int	290
To separately disable a single I/O interrupt.	
init_phs_int	291
To initialize a slave interrupt controller without installing an interrupt handler separately.	
deinit_phs_int	292
To deinitialize a slave interrupt controller without uninstalling an interrupt handler separately.	
set_phs_int_mask	293
To set the interrupt mask of a slave interrupt controller.	

install_phs_int_vector

Syntax

```
int install_phs_int_vector (
    phs_addr_t base,
    int n,
    void (*isr)())
```

Include file

phsint.h

Purpose

To install a PHS-bus interrupt handler.

Description


The corresponding I/O interrupt source as well as PHS-bus interrupts in general are enabled. Interrupts are not enabled globally. If the interrupt is the first one to be installed for the indicated I/O board, the board's interrupt controller is connected to a free PHS-bus interrupt line. This does not hold for customer I/O boards (such as the DS4201 Prototyping Board and its derivatives) which use static interrupt lines or customer-specific setup registers. For these boards, `declare_phs_int_line` or `alloc_phs_int_line` must be called before calling `install_phs_int_vector`.

Note

The vector can be uninstalled with `deinstall_phs_int_vector`. `deinit_phs_int` must not be applied to PHS-bus interrupts a vector is installed for.

Parameters

base Specifies the PHS-bus base address of the I/O board. Use the macro as described in [I/O board base address](#) on page 258.

n Specifies the slave interrupt number. Valid numbers are defined by the mapping of interrupt sources to inputs 0 ... 7 of the corresponding slave interrupt controller. Further information can be found in the [PHS Bus System Hardware Reference](#)  and in the *RTLib Reference* of your I/O board.

isr Specifies the entry point address of the interrupt handler to be installed.

Related topics**Basics**

[Management of the Extended Interrupt System..... 272](#)

HowTos

[How to Program PHS-Bus Interrupts..... 276](#)

Examples

[Example of PHS-Bus Interrupt Handling..... 277](#)

References

[alloc_phs_int_line..... 286](#)
[declare_phs_int_line..... 285](#)
[deinit_phs_int..... 292](#)
[deinstall_phs_int_vector..... 284](#)
[Error Codes of PHS-Bus Interrupt Functions..... 295](#)

deinstall_phs_int_vector

Syntax

```
int deinstall_phs_int_vector(  
    phs_addr_t base,  
    int n)
```

Include file

phsint.h

Purpose

To uninstall a registered interrupt service routine.

Description


This is the counterpart of `install_phs_int_vector`. The corresponding I/O interrupt source is disabled. If the interrupt is the last one to be uninstalled for the specified I/O board, the interrupt controller of the board is reconnected to the default PHS-bus interrupt line (0), which is reserved for boards operated in polling mode. If the interrupt was the last enabled PHS-bus interrupt of all, PHS-bus interrupts (not interrupts in general) are disabled.

Note

This function must not be used to deinitialize a PHS-bus interrupt initialized with `init_phs_int`. Use `deinit_phs_int` for this purpose.

Parameters

base Specifies the PHS-bus base address of the I/O board. Use the macro as described in [I/O board base address](#) on page 258.

n Specifies the slave interrupt number. Valid numbers are defined by the mapping of interrupt sources to inputs 0 ... 7 of the corresponding slave interrupt controller. Further information can be found in the [PHS Bus System Hardware Reference](#)  and in the *RTLib Reference* of your I/O board.

Related topics

Basics	
Management of the Extended Interrupt System.....	272
HowTos	
How to Program PHS-Bus Interrupts.....	276
Examples	
Example of PHS-Bus Interrupt Handling.....	277
References	
deinit_phs_int.....	292
Error Codes of PHS-Bus Interrupt Functions.....	295
init_phs_int.....	291
install_phs_int_vector.....	282

declare_phs_int_line

Syntax	<pre>int declare_phs_int_line(phs_addr_t base, int line)</pre>
Include file	phsint.h
Purpose	To declare a PHS-bus interrupt line.

Description

This function provides the means for declaring PHS-bus interrupt lines that are in use by nonstandard I/O boards. Normally, it is called once for each of the boards before any other function of the PHS-bus interrupt functions is called in order to prevent the system from allocating hard-wired interrupt lines to other PHS-bus boards. It is assumed that the interrupt line numbers of these boards have been statically defined by hardware and therefore cannot be modified by software.

Note

This function must be called before installing interrupt handlers for the specified board. Otherwise the function will issue an error message (error code PHSINT_FUNC_NOT_ALLOWED) and terminate the application.

Parameters

base Specifies the PHS-bus base address of the I/O board. Use the macro as described in [I/O board base address](#) on page 258.

line Specifies the statically defined PHS-bus interrupt line number of the board. The value of **line** must match the hardware setting of the PHS-bus interrupt line number of the board.

Related topics**Basics**

[Management of the Extended Interrupt System..... 272](#)

HowTos

[How to Program PHS-Bus Interrupts..... 276](#)

Examples

[Example of PHS-Bus Interrupt Handling..... 277](#)

References

[Error Codes of PHS-Bus Interrupt Functions..... 295](#)

alloc_phs_int_line

Syntax

```
int alloc_phs_int_line(phas_addr_t base)
```

Include file

phasint.h

Purpose

To allocate PHS-bus interrupt lines for nonstandard I/O boards.

Description

In particular, the function must be called for customer I/O boards that need specialized code for programming the PHS-bus interrupt line number. It is assumed that the user provides and executes the initialization code after calling **alloc_phs_int_line**, but before installing interrupt handlers for the board. The initialization will normally include the setting of the interrupt line number in a board-specific setup register. The allocated interrupt line number is therefore passed to the caller as the return value of **alloc_phs_int_line**.

Note

This function must be called before installing interrupt handlers for the specified board. Otherwise the function will issue an error message (error code `PHSINT_FUNC_NOT_ALLOWED`) and terminate the application.

Parameters	base Specifies the PHS-bus base address of the I/O board. Use the macro as described in I/O board base address on page 258.
Return value	Valid PHS-bus interrupt line number if successful.
Example	For an example of using the PHS-bus interrupt functions together with customer-specific I/O boards, refer to Example with Customer I/O Boards on page 280.
Related topics	<p>Basics</p> <p>Management of the Extended Interrupt System..... 272</p> <p>HowTos</p> <p>How to Program PHS-Bus Interrupts..... 276</p> <p>Examples</p> <p>Example of PHS-Bus Interrupt Handling..... 277</p> <p>References</p> <p>Error Codes of PHS-Bus Interrupt Functions..... 295</p> <p>free_phs_int_line..... 287</p>

free_phs_int_line

Syntax	<code>int free_phs_int_line(phas_addr_t base)</code>
Include file	<code>phasint.h</code>
Purpose	To free PHS-bus interrupt lines for nonstandard I/O boards.

Description

This is the counterpart of `alloc_phs_int_line`. It releases a PHS-bus interrupt line that was previously allocated for an I/O board. It is assumed that the user provides and executes the code for resetting the interrupt line number of the board to the default value after calling `free_phs_int_line`. The default interrupt line number is therefore passed to the caller as the return value of `free_phs_int_line`. Remember that the default interrupt line is reserved for boards operated in polling mode.

Note

This function must not be called until all interrupt handlers for the specified board have been uninstalled. Otherwise it will issue an error message (error code `PHSINT_FUNC_NOT_ALLOWED`) and terminate the application.

Parameters

base Specifies the PHS-bus base address of the I/O board. Use the macro as described in [I/O board base address](#) on page 258.

Return value

Default interrupt line number (0).

Example

For an example of using the PHS-bus interrupt functions together with customer-specific I/O boards, refer to [Example with Customer I/O Boards](#) on page 280.

Related topics**Basics**

[Management of the Extended Interrupt System..... 272](#)

HowTos

[How to Program PHS-Bus Interrupts..... 276](#)

Examples

[Example of PHS-Bus Interrupt Handling..... 277](#)

[Example with Customer I/O Boards..... 280](#)

References

[alloc_phs_int_line..... 286](#)

[Error Codes of PHS-Bus Interrupt Functions..... 295](#)

enable_phs_int

Syntax

```
enable_phs_int(  
    phs_addr_t base,  
    int n)
```

Include file `phsint.h`

Purpose To separately enable a single I/O interrupt.

Description The corresponding interrupt handler must have been installed beforehand.

Note

As this macro is assumed to be executed under real-time conditions, the interrupt mask register of the interrupt controller on the specified I/O board is modified without error checking.

Parameters

base Specifies the PHS-bus base address of the I/O board. Use the macro as described in [I/O board base address](#) on page 258.

n Specifies the slave interrupt number. Valid numbers are defined by the mapping of interrupt sources to inputs 0 ... 7 of the corresponding slave interrupt controller. Further information can be found in the [PHS Bus System Hardware Reference](#) and in the *RTLib Reference* of your I/O board.

Related topics

Basics

Management of the Extended Interrupt System..... 272

HowTos

How to Program PHS-Bus Interrupts..... 276

Examples

Example of PHS-Bus Interrupt Handling..... 277

References

disable_phs_int..... 290

Error Codes of PHS-Bus Interrupt Functions..... 295

disable_phs_int

Syntax

```
disable_phs_int(
    phs_addr_t base,
    int n)
```

Include file

phsint.h

Purpose

To separately disable a single I/O interrupt.

Description


The corresponding interrupt handler must have been installed before.

Note

As this macro is assumed to be executed under real-time conditions, the interrupt mask register of the interrupt controller on the specified I/O board is modified without error checking.

Parameters

base Specifies the PHS-bus base address of the I/O board. Use the macro as described in [I/O board base address](#) on page 258.

n Specifies the slave interrupt number. Valid numbers are defined by the mapping of interrupt sources to inputs 0 ... 7 of the corresponding slave interrupt controller. Further information can be found in the [PHS Bus System Hardware Reference](#)  and in the *RTLib Reference* of your I/O board.

Related topics

Basics

[Management of the Extended Interrupt System..... 272](#)

HowTos

[How to Program PHS-Bus Interrupts..... 276](#)

Examples

[Example of PHS-Bus Interrupt Handling..... 277](#)

References

[enable_phs_int..... 289](#)
[Error Codes of PHS-Bus Interrupt Functions..... 295](#)

init_phs_int

Syntax

```
init_phs_int(
    phs_addr_t base,
    int n)
```

Include file

phsint.h

Purpose

To initialize a slave interrupt controller without installing an interrupt handler separately.

Description

The specified I/O interrupt source is enabled at the slave interrupt controller. However, PHS-bus interrupts are not enabled generally. If the interrupt is the first one that is enabled for the indicated I/O board, the board's interrupt controller is connected to a free PHS-bus interrupt line. This does not hold for customer I/O boards (such as the DS4201 Prototyping Board and its derivatives) which use static interrupt lines or customer-specific setup registers. For these boards, `declare_phs_int_line` or `alloc_phs_int_line` must be called before calling `init_phs_int`.

Note

The PHS-bus interrupt can be deinitialized with `deinit_phs_int`. `deinstall_phs_int_vector` must not be used for this purpose.

Parameters

base Specifies the PHS-bus base address of the I/O board. Use the macro as described in [I/O board base address](#) on page 258.

n Specifies the slave interrupt number. Valid numbers are defined by the mapping of interrupt sources to inputs 0 ... 7 of the corresponding slave interrupt controller. Further information can be found in the [PHS Bus System Hardware Reference](#) and in the *RTLib Reference* of your I/O board.

Related topics**Basics**

Management of the Extended Interrupt System..... 272

HowTos

How to Program PHS-Bus Interrupts..... 276

Examples

Example of PHS-Bus Interrupt Handling..... 277

References

alloc_phs_int_line..... 286
 declare_phs_int_line..... 285
 deinit_phs_int..... 292
 deinstall_phs_int_vector..... 284
 Error Codes of PHS-Bus Interrupt Functions..... 295

deinit_phs_int

Syntax

```
deinit_phs_int(
    phs_addr_t base,
    int n)
```

Include file

phsint.h

Purpose

To deinitialize a slave interrupt controller without uninstalling an interrupt handler separately.

Description

It is the counterpart of `init_phs_int`. The specified I/O interrupt source is disabled at the slave interrupt controller. If the interrupt is the last one that is disabled for the specified I/O board, the interrupt controller of the board is reconnected to the default PHS-bus interrupt line (0), which is reserved for boards operated in polling mode. However, under no circumstances are PHS-bus interrupts disabled in general.

Note

This function must not be used to deinitialize a PHS-bus interrupt initialized with `install_phs_int_vector`. Use `deinstall_phs_int_vector` for this purpose.

Parameters

- base** Specifies the PHS-bus base address of the I/O board. Use the macro as described in [I/O board base address](#) on page 258.
- n** Specifies the slave interrupt number. Valid numbers are defined by the mapping of interrupt sources to inputs 0 ... 7 of the corresponding slave interrupt controller. Further information can be found in the [PHS Bus System Hardware Reference](#) and in the *RTLib Reference* of your I/O board.

Related topics

Basics

Management of the Extended Interrupt System..... 272

HowTos

How to Program PHS-Bus Interrupts..... 276

Examples

Example of PHS-Bus Interrupt Handling..... 277

References

deinstall_phs_int_vector..... 284

Error Codes of PHS-Bus Interrupt Functions..... 295

init_phs_int..... 291

install_phs_int_vector..... 282

set_phs_int_mask

Syntax

```
set_phs_int_mask(  
    phs_addr_t base,  
    int mask)
```

Include file

phsint.h

Purpose To set the interrupt mask of a slave interrupt controller.

Description The interrupt controller must have been initialized beforehand. The constants listed in the table below have been defined to simplify the usage of this macro.

Note

As this macro is assumed to be executed under real-time conditions, the mask register of the interrupt controller on the specified I/O board is modified without error checking.

Parameters **base** Specifies the PHS-bus base address of the I/O board. Use the macro as described in [I/O board base address](#) on page 258.

mask Specifies the bit pattern to be programmed into the interrupt mask register. A "1" bit disables the corresponding slave interrupt, while a "0" bit enables it. The following table shows the slave interrupt controller mask bits (defined in `phsint.h`).

Error Code	Value	Meaning
PHS_MASK_SLAVE_INT0	0x01	Bit position of slave interrupt 0
PHS_MASK_SLAVE_INT1	0x02	Bit position of slave interrupt 1
PHS_MASK_SLAVE_INT2	0x04	Bit position of slave interrupt 2
PHS_MASK_SLAVE_INT3	0x08	Bit position of slave interrupt 3
PHS_MASK_SLAVE_INT4	0x10	Bit position of slave interrupt 4
PHS_MASK_SLAVE_INT5	0x20	Bit position of slave interrupt 5

Error Code	Value	Meaning
PHS_MASK_SLAVE_INT6	0x40	Bit position of slave interrupt 6
PHS_MASK_SLAVE_INT7	0x80	Bit position of slave interrupt 7

Related topics

Basics

[Management of the Extended Interrupt System..... 272](#)

HowTos

[How to Program PHS-Bus Interrupts..... 276](#)

Examples

[Example of PHS-Bus Interrupt Handling..... 277](#)

References

[Error Codes of PHS-Bus Interrupt Functions..... 295](#)

Troubleshooting for PHS-Bus Interrupt Handling

Error Codes of PHS-Bus Interrupt Functions

Error codes

There are a few errors where program execution may be continued. In these cases, a warning message is generated and the function returns the appropriate error code. The following table shows the error codes of the PHS-bus interrupt functions (defined in `phsint.h`).

Error Code	Meaning	Message	Action
PHSINT_NO_ERROR	No error	None	Return
PHSINT_NO_BOARD	No board detected at the specified PHS-bus base address	Error	Exit
PHSINT_UNKNOWN_BOARD	Unknown board identification number detected	Error	Exit
PHSINT_BOARD_UNINITIALIZED	The board to be accessed has not been initialized	Error	Exit
PHSINT_NO_INT_LINE	No free PHS-bus interrupt line available	Error	Exit
PHSINT_INVALID_BASE	Invalid board base address specified	Error	Exit
PHSINT_INVALID_SLAVE_NUM	Invalid slave interrupt number specified	Error	Exit

Error Code	Meaning	Message	Action
PHSINT_ICU_NOT_ENABLED	Slave interrupt controller has no interrupts enabled	Warning	Return
PHSINT_SLAVE_NOT_INST	Specified slave interrupt is not installed	Warning	Return
PHSINT_VECTOR_IN_USE	Slave interrupt vector is already in use	Error	Exit
PHSINT_INVALID_LINE_NUM	Invalid PHS-bus interrupt line number specified	Error	Exit
PHSINT_LINE_IN_USE	Specified PHS-bus interrupt line is already in use	Error	Exit
PHSINT_NO_STATIC_INT_LINE	Board at specified base address does not use a static PHS-bus interrupt line	Warning	Return
PHSINT_FUNC_NOT_ALLOWED	Function call is not allowed because slave interrupts are enabled	Error	Exit
PHSINT_NO_PIC	Board at specified base address does not contain a slave interrupt controller	Error	Exit
PHSINT_INCONSISTENT_LINES	Inconsistent line numbers declared for the board at specified base address	Error	Exit
PHSINT_NO_ISR_INSTALLED	No handler is installed for the triggered PHS-bus interrupt.	Warning	Return (from interrupt)

Multiprocessing Modules

Introduction	A DS1006 modular system provides the multiprocessor (MP) feature for the single-core processor, and additionally the multicore (MC) feature for the multicore processor. You use the same functions for MC and MP systems.
Where to go from here	<div>Information in this section<div>Initialization.....298Global Sample Rate Timer in an MP System.....309Interprocessor Interrupts.....315Gigalink Communication.....327</div></div>

Initialization

Introduction

This chapter contains the functions to initialize a multiprocessor system (MP system).

Where to go from here

Information in this section

Data Types for MP System Initialization.....	298
Gives you definitions of the data types used in the MP system initialization.	
Example of an MP System Initialization.....	299
Gives you an example of a three processor system.	
ds1006_mp_init.....	301
To initialize all MP system relevant software modules.	
ds1006_mp_synchronize.....	303
To perform a synchronized start of an MP system.	
ds1006_mp_route_mat.....	305
To route the macrotick interrupt through the system.	
ds1006_mp_route_syncin.....	305
To route the SYNCIN signal through the system.	
ds1006_mp_route_syncout.....	306
To route the SYNCOUT signal through the system.	
ds1006_mp_optional_cpu_reduce.....	307
To clear the <code>mp_topology</code> matrix from not present members in an MP system.	

Data Types for MP System Initialization

Data types

The following data types are defined:

`mp_target_type`

```
typedef struct {
    Int32 cpu_no;    /* CPU number */
    Int32 gl_no;     /* Gigalink number */
}mp_target_type;
```

mp_topology_type

```
typedef struct {
    mp_target_type target[4];
}mp_topology_type;
```

mp_cpu_available_type

```
UInt32 mp_cpu_available_type;
```

Every MP application has a global or local variable of type **mp_topology_type**, which is passed to **ds1006_mp_init**. This variable contains a 4 by *n* matrix, where *n* is the number of processors in the system. Hence, each element of the matrix describes one Gigalink in the MP system.

The elements (source Gigalink) are of type **mp_target_type**, which consists of the CPU and Gigalink number of the target Gigalink, to which the source Gigalink is connected. The matrix is redundant, as two connected Gigalinks form a matching pair in the matrix. The function **ds1006_mp_init** checks the consistency of the matrix. The **#defines CPU_x** (*x* = NONE, 0 ... 15) help to specify the CPU number, the **#defines GL_x** (*x* = NONE, 0 ... 3) help to specify the Gigalink number. An example can be found in [Example of an MP System Initialization](#) on page 299.

The elements of the array that specifies the available CPUs are of type **mp_cpu_available_type**. This array is passed to **ds1006_mp_optional_cpu_reduce** as parameter.

Global variables

The following global variables are defined:

rtlib_cpu_id ID of the local CPU. On default (single processor system) the local CPU is assigned ID 0. The CPU with ID 0 automatically is the master CPU in a MP system (e.g., for the DSTS module).

rtlib_num_cpus Number of CPUs in the system, specified by **ds1006_mp_init**.

Example of an MP System Initialization

Example source code

The example shows an initialization of a DS1006 board in a three processor system. The three boards (CPU 0 to 2) are interconnected by Gigalinks (GL 0 to 3). There are two connections between the boards (physically there may be more connections, but only these two are used by the application):

CPU 0, GL 0 to CPU 1, GL 0

CPU 1, GL 1 to CPU 2, GL 0

The CPU, for which the application is compiled for, is 1 (see **#define CPU_ID**). Please note, that this code frame can be used at all three processors. You only

have to adapt the definition of `CPU_ID` (0 for the master processor, 1 and 2 for the slaves).

```
/* Global sampling rate timer example (for CPU with ID 1) */
/* Connections: CPU0/GL0 <-> CPU1/GL0, CPU1/GL1 <-> CPU2/GL0 */
#include <brtenv.h>
#define CPU_ID 1
#define NUM_CPUS 3
volatile mp_topology_type mp_topology[NUM_CPUS] =
{
  {{ {CPU_1, GL_0}, {CPU_NONE, GL_NONE}, {CPU_NONE, GL_NONE},
    {CPU_NONE, GL_NONE}}},
  {{ {CPU_0, GL_0}, {CPU_2, GL_0}, {CPU_NONE, GL_NONE},
    {CPU_NONE, GL_NONE}}},
  {{ {CPU_1, GL_1}, {CPU_NONE, GL_NONE}, {CPU_NONE, GL_NONE},
    {CPU_NONE, GL_NONE}}}
};
void global_isr()
{
  ds1006_mp_begin_isr_global_srt();
  /* ... body of the isr ... */
  ds1006_mp_end_isr_global_srt();
}
void main(void)
{
  /* initialize single-processor modules */
  ds1006_init on page 18();
  /* initialize the multiprocessor system */
  ds1006_mp_init(NUM_CPUS, CPU_ID, mp_topology,
    0.001, 5.0, 5);
  /* synchronize cpus */
  ds1006_mp_synchronize();
  /* define start time (t=0) for the simulation */
  ts_reset();
  /* start global sampling rate timer */
  ds1006_mp_start_isr_global_srt(0.001, global_isr);
  /* enter background loop */
  while(1)
  {
    RTLIB_BACKGROUND_SERVICE();
  }
}
```

Description

First, the board is initialized by calling `ds1006_init`. This function must be called before any function or variable from RTLlib1006 is used. Then the MP initialization function `ds1006_mp_init` is called. It gives the processor ID 1 and specifies the number of processors in the system as 3. The processor ID must be within the range 0 ... number of CPUs - 1. Every processor ID must be unique in the MP system. The CPU with ID 0 has special importance: It is the master processor in the MP system. It generates the system-wide macrotick (see [Time-Stamping](#) on page 51). When a global sampling rate timer is used, the master processor generates the global timer interrupt.

After assigning the CPU ID the function `ds1006_mp_init` initializes all Gigalinks, which are specified in the topology information. When a specified Gigalink is not connected, the function issues an error message. Furthermore this function initializes the STBU with the given Macrotock period. The STBU of the master

processor is initialized in master mode, all others in slave mode. At this point, the STBU is stopped.

Next, a global sampling rate timer is initialized by calling `ds1006_mp_start_isr_global_srt`. The global timer interrupt is the Timer A interrupt from the master CPU, which is dispatched over Gigalinks to all other processor boards. Hence, at the master CPU this function programs the Timer A interrupt with a period of 0.0001 s and routes it to the Gigalinks. The interrupt service routine is installed at the Timer A interrupt vector. At the slave boards the interrupt service routine is installed at the appropriate Gigalink interrupt.

After the global sampling rate timer is initialized, all CPUs are synchronized by calling `ds1006_mp_synchronize`. All CPUs leave the function at the same time. Now the multiprocessor application is ready for starting real-time. The `ts_reset` function defines the absolute time 0 of the simulation (in MP systems this zeroes the synchronous time base unit) and starts the STBU.

Related topics

References

ds1006_init	18
ds1006_mp_init	301
ds1006_mp_start_isr_global_srt	311
ds1006_mp_synchronize	303
Time-Stamping	51
ts_reset	58

ds1006_mp_init

Syntax

```
void ds1006_mp_init(  
    int num_cpus,  
    int cpu_id,  
    mp_topology_t mp_topology,  
    double mat_period,  
    double timeout,  
    int num_retries)
```

or

```
void ds1006_gl_init(  
    int num_cpus,  
    int cpu_id,  
    mp_topology_t mp_topology,  
    double mat_period,  
    double timeout,  
    int num_retries)
```

Include file

`mp1006.h`

Purpose

To initialize all MP system relevant software modules.

Description

This function has to be carried out after the board initialization function `ds1006_init`. It performs the following setups:

- It assigns the local CPU (`rtlib_cpu_id`) the ID `cpu_id` and sets the number of CPUs (`rtlib_num_cpus`) to `num_cpus`.
- It checks all Gigalink connections that are specified for the local CPU in the topology matrix `mp_topology` and establishes them.
- It sets up the STBU (Synchronous Time Base Unit) by calling the time stamping initialization routine `ts_init`. The STBU is stopped. The processor with ID 0 is defined as time base master, which generates the system macrotick with a period of `mat_period`. The macrotick event is routed through the system. The Gigalinks of the slave processors (IDs from 1 to `num_cpus-1`) are configured for properly receiving the macrotick event.
- It initializes the MP trigger dispatching mechanism, which routes trigger events through the system. These triggers are needed for time stamping and distributed tracing.

The function tries to initialize the MP system until the `timeout` elapses. In this case an info message is issued and the initialization is retried. The maximum number of retries is specified by `num_retries`. When the maximum number of retries is reached and the initialization is still unsuccessful, an error message is issued. The further behavior depends on the sign of `num_retries`. If it is positive, the application is terminated. If it is negative, the application continues.

Note

- If you use a single-core DS1006 board, this function aborts if there is no DS911 Gigalink module installed.
- If you use a multicore DS1006 board, a DS911 Gigalink module is not required, while the Gigalink connections specified in the topology can be realized with the internal Gigalinks. The internal Gigalinks are virtually connected and automatically configured based on the specified topology. For further information, refer to [Using DS1006 With Multicore Processor \(DS1006 Features !\[\]\(661ad2fdbe8fa1392f2b194cfa45d124_img.jpg\)](#)).
- If you use a multicore DS1006 board, the term *CPU* is to be used for one processor core. For example, the `num_cpus` parameter specifies the number of processor cores.

Parameters

num_cpus Specifies the number of CPUs in the multiprocessor system.

cpu_id Specifies the ID of the local CPU within the range 0 ... (`num_cpus-1`).

mp_topology Specifies the topology matrix that stores all Gigalink connections of the multiprocessor system. Each CPU has one row (index: CPU ID) and each Gigalink has one column (index: 0 ... 3). An element at position (m, n)

specifies the target of Gigalink n at CPU m. The target itself is a pair of (k, l), where k is the target CPU and l the target Gigalink.

mat_period Specifies the period of the multiprocessor system macrotick counter in seconds. Due to the dependency between **mat_period** and the time range and accuracy, we recommend a value of 1 ... 10 ms (see [Time-Stamping](#) on page 51).

timeout Specifies the timeout in seconds.

Note

To avoid timeout problems when using handcoded MP applications, set timeout to 10.

num_retries Specifies the maximum number of retries. This value can be positive or negative. When a positive number of retries is used the application is terminated after all retries were unsuccessful. A negative number of retries lets the processor continue after the retries. In both cases an information message is issued after each retry. When the number of retries is set to DS1006_INIT_INF_RETRIES the function retries forever.

Note

To avoid timeout problems when using handcoded MP applications, set num_retries to (2 * num_cpus).

Return value None

Related topics

Examples

Example of an MP System Initialization.....	299
---	---------------------

References

ds1006_init.....	18
ds1006_mp_synchronize.....	303
Time-Stamping.....	51

ds1006_mp_synchronize

Syntax void ds1006_mp_synchronize()

or

```
void dsgl_mp_synchronize()
```

Include file mp1006.h

Purpose To perform a synchronized start of an MP system.

Note

- Use this function only during the initialization of your model or in the model background because the function also resets the Time Stamping module and stops the STBU. Use `ts_reset` to start the STBU again.
- You have to initialize your MP system by calling `ds1006_mp_init` before using this function.

Description The synchronization is performed in three steps:

- In the first step the status word dispatching mechanism is used to send a synchronization request bit from the master CPU (ID = 0) to all slave CPUs.
- The slave receive the request and acknowledge it by setting the appropriate bit in their status words, which are dispatched to the master CPU.
- When the master finds all acknowledge bits in the slave status words set, it signals the system start by sending a MAT interrupt. The slave processors poll the MAT interrupt line and exit this function when they find the interrupt set. So, there is only a small time jitter within all processors exit the synchronization function.

The status word dispatching mechanism and the MAT interrupt is initialized by `ds1006_mp_init`. When this function is not called all processors consider themselves as single processor system and omit synchronization.

Return value None

Related topics

Examples

[Example of an MP System Initialization.....](#) 299

References

[ds1006_mp_init.....](#) 301
[Time-Stamping.....](#) 51

ds1006_mp_route_mat

Syntax

```
void ds1006_mp_route_mat()
```

or

```
void dsgl_mp_route_mat()
```

Include file

mp1006.h

Purpose

To route the macrotick interrupt through the system.

Description

This function is called by `ds1006_mp_init`. For more information on the macrotick interrupt, refer to [Time-Stamping](#) on page 51.

Return value

None

Related topics

References

[ds1006_mp_init](#)..... 301

ds1006_mp_route_syncin

Syntax

```
void ds1006_mp_route_syncin(void)
```

or

```
void dsgl_mp_route_syncin(void)
```

Include file

mp1006.h

Purpose

To route the SYNCIN signal through the system.

Description

This function configures the Gigalink port of one member of a MP system. Depending on the member's position, the port is specified to send, forward or receive the SYNCIN signal. For the dependency of the member's position and the activity, refer to the following table:

Member's position	Activity
Master (CPU ID=0)	Send
Slave directed to master	Receive
Slave directed away from master	Forward

Note

- To route the SYNCIN signal through the entire MP system, each member must call this function.
- Before you can specify the Gigalink port for the SYNCIN signal routing, you must call **ds1006_mp_init**.
- To trigger the SYNCIN signal for the entire MP system, the macro **PHS_SYNCIN_TRIGGER** has to be called by the master CPU.

Return value None

Related topics**References**

ds1006_mp_init	301
ds1006_mp_route_syncout	306
PHS_SYNCIN_TRIGGER	267

ds1006_mp_route_syncout

Syntax

```
void ds1006_mp_route_syncout(void)
```

or

```
void dsgl_mp_route_syncout(void)
```

Include file **mp1006.h**

Purpose To route the SYNCOUT signal through the system.

Description

This function configures the Gigalink port of one member of a MP system. Depending on the member's position, the port is specified to send, forward or receive the SYNCOUT signal. For the dependency of the member's position and the activity, refer to the following table:

Member's position	Activity
Master (CPU ID=0)	Send
Slave directed to master	Receive
Slave directed away from master	Forward

Note

- To route the SYNCOUT signal through the entire MP system, each member must call this function.
- Before you can specify the Gigalink port for the SYNCOUT signal routing, you must call **ds1006_mp_init**.
- To trigger the SYNCOUT signal for the entire MP system, the macro **PHS_SYNCOUT_TRIGGER** has to be called by the master CPU.

Return value None

Related topics	References
	ds1006_mp_init..... 301
	ds1006_mp_route_syncin..... 305
	PHS_SYNCOUT_TRIGGER..... 267

ds1006_mp_optional_cpu_reduce

Syntax

```
void ds1006_mp_optional_cpu_reduce(  
    int num_cpus,  
    mp_topology_type *mp_topology,  
    mp_cpu_available_type *mp_cpu_available)
```

or

```
void ds1006_mp_optional_cpu_reduce(  
    int num_cpus,  
    mp_topology_type *mp_topology,  
    mp_cpu_available_type *mp_cpu_available)
```

Include file mp1006.h

Purpose To clear the **mp_topology** matrix from not present members in a MP system.

Description

This function removes entries of not present CPUs from the `mp_topology` matrix. If a required CPU is not present, the program exits with an error message, if an optional CPU is not present, the entry in the `mp_cpu_available` array is changed to `CPU_ABSENT`.

Parameters

num_cpus Specifies the number of CPUs in the multiprocessor system.

mp_topology Specifies the pointer to the topology matrix that stores all Gigalink connections of the multiprocessor system. Each CPU has one row (index: CPU ID) and each Gigalink has one column (index: 0 to 3). An element at position (m, n) specifies the target of Gigalink n at CPU m. The target itself is a pair of (k, l), where k is the target CPU and l the target Gigalink.

mp_cpu_available Specifies the pointer to the array of required and optional CPUs. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CPU_ABSENT</code>	Optional member is not present
<code>CPU_OPTIONAL</code>	Optional member
<code>CPU_REQUIRED</code>	Required member

Return value

None

Example

```
#define NUM_CPUS 3
mp_topology_type mp_topology[NUM_CPUS] =
{ /* CPU_0 */ {{ {CPU_1, GL_0}, {CPU_2, GL_0},
                 {CPU_NONE, GL_NONE}, {CPU_NONE, GL_NONE} }},
  /* CPU_1 */ {{ {CPU_0, GL_0}, {CPU_2, GL_1},
                 {CPU_NONE, GL_NONE}, {CPU_NONE, GL_NONE} }},
  /* CPU_2 */ {{ {CPU_0, GL_1}, {CPU_1, GL_1},
                 {CPU_NONE, GL_NONE}, {CPU_NONE, GL_NONE} }}
};
mp_cpu_available_type mp_cpus[NUM_CPUS] =
{CPU_REQUIRED, CPU_REQUIRED, CPU_OPTIONAL};
...
init();
ds1006_mp_optional_cpu_reduce(NUM_CPUS, mp_topology, mp_cpus);
ds1006_mp_init(NUM_CPUS, CPU_0, mp_topology, 0.001, 5.0, 3);
...
```

For further information, see also [Example of an MP System Initialization](#) on page 299.

Global Sample Rate Timer in an MP System

Introduction

This chapter contains the functions to initialize a global sampling rate timer in a multiprocessor system (MP system) or a multicore system (MC system).

Where to go from here

Information in this section

Example of Initializing a Global Sample Rate Timer.....	309
ds1006_mp_global_srt_init.....	310
To initialize a global sampling rate timer.	
ds1006_mp_start_isr_global_srt.....	311
To initialize a global sampling rate timer and install an interrupt service routine (ISR) for the corresponding interrupt.	
ds1006_mp_begin_isr_global_srt.....	313
To implement and start an overrun check.	
ds1006_mp_end_isr_global_srt.....	314
To finish an overrun check.	

Example of Initializing a Global Sample Rate Timer

Introduction

This example shows an initialization of a DS1006 processor board in a three multiprocessor system.

Example

```
/* MP initialization of cpu no. 1 of a 3 processor system */
/* Connections: CPU0/GL0 <-> CPU1/GL0, CPU1/GL1 <-> CPU2/GL0 */
#include <brtenv.h>
#define NUM_CPUS 3
#define CPU_ID 0
volatile mp_topology_type mp_topology[NUM_CPUS]=
{
  {{CPU_1, GL_0}, {CPU_NONE, GL_NONE}, {CPU_NONE, GL_NONE},
   {CPU_NONE, GL_NONE}}
  {{CPU_0, GL_0}, {CPU_2, GL_0}, {CPU_NONE, GL_NONE},
   {CPU_NONE, GL_NONE}}
  {{CPU_1, GL_1}, {CPU_NONE, GL_NONE}, {CPU_NONE, GL_NONE},
   {CPU_NONE, GL_NONE}}
};
```

```

void global_isr() /* interrupt service routine */
{
    ds1006_mp_begin_isr_global_srt();
    /* do something */
    ds1006_mp_end_isr_global_srt();
}

void main(void)
{
    /* initialize single-processor modules */
    ds1006_init();
    /* initialize the multiprocessor system */
    ds1006_mp_init(NUM_CPUS, CPU_ID, mp_topology,
        0.001, 5.0, 5);
    /* synchronize cpus */
    ds1006_mp_synchronize();
    /* define start time (t=0) for the simulation */
    ts_init();
    /* start global sampling rate timer */
    ds1006_mp_start_isr_global_srt(0.001, global_isr);
    /* enter background loop */
    while(1)
    {
        RTLIB_BACKGROUND_SERVICE();
    }
}

```

ds1006_mp_global_srt_init

Syntax

```

void ds1006_mp_global_srt_init(
    double period,
    DS1006_Int_Handler_Type isr)

```

or

```

void ds1006_dsgl_global_srt_init(
    double period,
    DS1006_Int_Handler_Type isr)

```

Include file

mp1006.h

Purpose

To initialize a global sampling rate timer by routing the timer A interrupt through the system and to install an interrupt service routine (ISR).

Description

This function initializes a global sampling rate timer by routing the timer A interrupt from the master processor (cpu id = 0) through the system and installs the ISR. The function has to be called on every processor in an MP system to route the interrupt properly.

The function performs the following actions on the master and the slave processors:

When the function is called at the master processor, all the Gigalinks which have slave processors connected to them are configured to send the timer A interrupt (Gigalink interrupt 0). Then the interrupt service routine `isr` is installed at the timer A interrupt vector. The timer is programmed with period `period` and then started. Finally, the interrupt is enabled.

When the function is called at a slave processor, the Gigalink which is directed to the master processor is configured to receive Gigalink interrupt 0. The interrupt service routine `isr` is installed at Gigalink interrupt 0. Gigalinks directed to further slave processors are configured to send interrupt 0. Interrupt 0 is enabled at the master Gigalink.

The function unmask the timer/Gigalink interrupt, but does not enable interrupts globally.

Note

- Initialize the multiprocessor system by calling `ds1006_mp_init` before calling this function. Otherwise the function behaves like `ds1006_start_isr_timerA` and all Gigalink actions are omitted.

Parameters

period Specifies the sampling rate timer period in seconds.

isr Specifies the name of the interrupt service routine.

Return value

None

Related topics

References

[ds1006_mp_begin_isr_global_srt..... 313](#)
[ds1006_mp_start_isr_global_srt..... 311](#)

ds1006_mp_start_isr_global_srt

Syntax

```
void ds1006_mp_start_isr_global_srt(
    Float64 period,
    DS1006_Int_Handler_Type isr_function_name)
```

or

```
void dsgl_mp_start_isr_global_srt(
    Float64 period,
    DS1006_Int_Handler_Type isr_function_name)
```

Include file `int1006.h`

Purpose To initialize a global sampling rate timer and install an interrupt service routine (ISR) for the corresponding interrupt.

Description A global sampling rate timer is used for synchronously triggering ISRs at every processor in an MP system. The interrupt is generated by the master processor and dispatched by Gigalinks to all other processors. As this is implemented in hardware, the interrupt arrives with minimal latency at each DS1006 in the system, even if there are multiple DS1006 on its way between the master and a slave processor.

When the function is called at the master processor, Timer A is initialized with the period `period` and the ISR `isr_function_name` is installed at its interrupt vector. Subsequently the Timer A interrupt is routed to the Gigalink module. The interrupt is dispatched as Gigalink hardware subinterrupt 0.

When the function is called at a slave processor, the ISR `isr_function_name` is installed at the Gigalink subinterrupt vector 0. The interrupt is then routed further to other Gigalinks.

Note

- When a global sampling rate timer is desired, this function must be called on every processor in the multiprocessor system for proper initialization.
- Initialize the multiprocessor system by calling `ds1006_mp_init` before calling this function. Otherwise the function behaves like `ds1006_start_isr_timerA`.

Parameters

period Specifies the sampling rate timer period in seconds.

isr_function_name Specifies the name of the interrupt service routine. This function must not have an input parameter or a return value, meaning, `void isr_function_name(void)`. You can implement an overrun check within your interrupt service routine with the functions `ds1006_mp_begin_isr_global_srt` and `ds1006_mp_end_isr_global_srt`.

Return value None

Related topics**Examples**

[Example of an MP System Initialization.....](#) 299

References

[ds1006_mp_begin_isr_global_srt.....](#) 313
[ds1006_mp_end_isr_global_srt.....](#) 314
[ds1006_mp_global_srt_init.....](#) 310

ds1006_mp_begin_isr_global_srt

Syntax

```
void ds1006_mp_begin_isr_global_srt()
```

or

```
void dsgl_mp_begin_isr_global_srt()
```

Include file

int1006.h

Purpose

To implement an overrun check mechanism for the global sampling rate timer together with `ds1006_mp_end_isr_global_srt`.

Description

Use both functions in the interrupt service routine of the global sampling rate timer. The code enclosed by them is executed with enabled interrupts. When the interrupt reoccurs, an overrun error message is issued and the interrupt is disabled.

Return value

None

Related topics**References**

[ds1006_mp_end_isr_global_srt.....](#) 314
[ds1006_mp_start_isr_global_srt.....](#) 311

ds1006_mp_end_isr_global_srt

Syntax

```
void ds1006_mp_end_isr_global_srt()
```

or

```
void ds1006_mp_end_isr_global_srt()
```

Include file

int1006.h

Purpose

To implement an overrun check mechanism for the global sampling rate timer together with **ds1006_mp_begin_isr_global_srt**.

Description

Use both functions in your interrupt service routine of the global sampling rate timer. The code enclosed by them is executed with enabled interrupts. When the interrupt reoccurs an overrun error message is issued and the interrupt is disabled.

Return value

None

Related topics**References**

```
ds1006_mp_begin_isr_global_srt..... 313
ds1006_mp_start_isr_global_srt..... 311
```

Interprocessor Interrupts

Introduction

This chapter contains the functions that can be used to configure interrupt transmission between multiple processors by using the Gigalink modules and/or multiple cores by using the virtual internal Gigalinks.

Where to go from here

Information in this section

ds1006_ipi_init.....	316
To initialize the interprocessor interrupt module.	
ds1006_ipi_configure.....	316
To configure an interprocessor interrupt line.	
ds1006_ipi_interrupt.....	317
To trigger an interprocessor interrupt.	
ds1006_ipi_acknowledge.....	318
To acknowledge an interprocessor interrupt.	
ds1006_ipi_enable.....	319
To enable a single interrupt line.	
ds1006_ipi_disable.....	320
To disable a single interrupt line.	
ds1006_ipi_enable_bm.....	321
To enable several interrupt lines.	
ds1006_ipi_disable_bm.....	321
To disable several interrupt lines.	
ds1006_ipi_mask_set.....	322
To set an interrupt mask.	
ds1006_ipi_mask_get.....	323
To get an interrupt mask.	
ds1006_ipi_sint_max_snd_set.....	323
To set the maximum number of subinterrupts that can be sent.	
ds1006_ipi_sint_max_rcv_set.....	324
To set the maximum number of subinterrupts that can be received.	
ds1006_ipi_install_handler.....	325
To install an interrupt service routine for an interprocessor interrupt.	

ds1006_ipi_init

Syntax

```
int ds1006_ipi_init(void)
```

or

```
int dsgl_ipi_init(void)
```

Include file

ipi1006.h

Purpose

To register the interprocessor interrupt module at the VCM (version and config section management) module and initialize the interprocessor subinterrupt handling.

Description

This function is called by `ds1006_mp_init`.

Return value

This function returns one of the following values:

Value	Meaning
0	Gigalink module not present
1	Gigalink module present

ds1006_ipi_configure

Syntax

```
int ds1006_ipi_configure(
    int gl_no,
    int line,
    int config)
```

or

```
int dsgl_ipi_configure(
    int gl_no,
    int line,
    int config)
```

Include file

ipi1006.h

Purpose

To set the interrupt source for an outgoing interrupt line at the specified Gigalink.

Description Use the return value to determine whether this outgoing interrupt line is already being used by a different source than the software interrupt.

Parameters

gl_no Specifies the Gigalink number within the range 0 ... 3.

line Specifies the outgoing interrupt line number within the range 0 ... 12.

config Specifies the interrupt line configuration. The following symbols are predefined:

Predefined Symbol	Meaning
IPI_CFG_SOFTWARE	Software interrupt only
IPI_CFG_HARDWARE	Hardware and software interrupts
IPI_CFG_GIGALINK	Gigalink and software interrupts

Return value This function returns the old interrupt configuration. The predefined symbols are shown in the table above.

Example The following example configures Gigalink 0 Line 1 to be driven by Timer B (hardware source). If the line was already being used an error is issued.

```
if (ds1006_ipi_configure(0, 1, IPI_CFG_HARDWARE) !=
    IPI_CFG_SOFTWARE)
{
    msg_error_set(MSG_SM_USER, 0,
        "Gigalink 0 Line 1 was already used.");
}
```

Related topics

References

[ds1006_ipi_sint_max_snd_set..... 323](#)

ds1006_ipi_interrupt

Syntax

```
void ds1006_ipi_interrupt(
    int gl_no,
    int int_no)
```

or

```
void dsgl_ipi_interrupt(
    int gl_no,
    int int_no)
```

Include file `ipi1006.h`

Purpose To trigger an interrupt for an outgoing interrupt line or a software dispatched subinterrupt.

Parameters

gl_no Specifies the Gigalink number within the range 0 ... 3.

int_no Specifies the outgoing interrupt line number within the range 0 ... 12 or software dispatched subinterrupt within the range 16 ... (15+**max_ints**). The variable **max_ints** is specified by the function **ds1006_ipi_sint_max_rcv_set**.

Return value None

Related topics **References**

[ds1006_ipi_sint_max_snd_set..... 323](#)

ds1006_ipi_acknowledge

Syntax

```
void ds1006_ipi_acknowledge(
    int gl_no,
    int line)
```

or

```
void dsgl_ipi_acknowledge(
    int gl_no,
    int line)
```

Include file `ipi1006.h`

Purpose To acknowledge a single incoming Gigalink interrupt.

Description	<p>The real-time software acknowledges an incoming Gigalink interrupt. Call this function only when you want to delete a disabled interrupt before it is enabled.</p> <div> Note This function must not be called for software dispatched subinterrupts. </div>
Parameters	<p>gl_no Specifies the Gigalink number within the range 0 ... 3.</p> <p>line Specifies the input interrupt line number within the range 0 ... 12.</p>
Return value	None

ds1006_ipi_enable

Syntax	<pre>void ds1006_ipi_enable(int gl_no, int line)</pre> <p>or</p> <pre>void ds1006_ipi_enable(int gl_no, int line)</pre>
Include file	ipi1006.h
Purpose	<p>To enable a single incoming interrupt line of the interrupt control unit.</p> <div> Note This function must not be called for software dispatched subinterrupts. </div>
Parameters	<p>gl_no Specifies the Gigalink number within the range 0 ... 3.</p> <p>line Specifies the input interrupt line number within the range 0 ... 12.</p>

Related topics**References**

ds1006_ipi_disable.....	320
ds1006_ipi_enable_bm.....	321

ds1006_ipi_disable

Syntax

```
void ds1006_ipi_disable(
    int gl_no,
    int line)
```

or

```
void ds1006_ipi_disable(
    int gl_no,
    int line)
```

Include file

ipi1006.h

Purpose

To disable a single incoming interrupt line in the interrupt control unit.

Note

This function must not be called for software dispatched subinterrupts.

Parameters**gl_no** Specifies the Gigalink number within the range 0 ... 3.**line** Specifies the input interrupt line number within the range 0 ... 12.**Return value**

None

Related topics**References**

ds1006_ipi_disable_bm.....	321
ds1006_ipi_enable.....	319

ds1006_ipi_enable_bm

Syntax

```
void ds1006_ipi_enable_bm(
    int gl_no,
    UInt32 bitmask)
```

or

```
void dsgl_ipi_enable_bm(
    int gl_no,
    UInt32 bitmask)
```

Include file

ipi1006.h

Purpose

To enable several incoming interrupt lines of the interrupt control unit.

Parameters

gl_no Specifies the Gigalink number within the range 0 ... 3.

bitmask Specifies the interrupt lines to be enabled. Each incoming interrupt line corresponds with one bit in the bitmask (Bit 0 sets interrupt line 0, Bit 1 sets interrupt line 1, etc.). Set for each interrupt line that is to be enabled the corresponding bit to 1.

Return value

None

Related topics

References

ds1006_ipi_disable_bm	321
ds1006_ipi_enable	319

ds1006_ipi_disable_bm

Syntax

```
void ds1006_ipi_disable_bm(
    int gl_no,
    UInt32 bitmask)
```

or

```
void dsgl_ipi_disable_bm(
    int gl_no,
    UInt32 bitmask)
```

Include file	<code>ipi1006.h</code>
Purpose	To disable several incoming interrupt lines in the interrupt control unit.
Parameters	<p>gl_no Specifies the Gigalink number within the range 0 ... 3.</p> <p>bitmask Specifies the interrupt lines to be disabled. Each incoming interrupt line corresponds with one bit in the bitmask (Bit 0 sets interrupt line 0, Bit 1 sets interrupt line 1, etc.). Set for each interrupt line that is to be disabled the corresponding bit to 1.</p>
Return value	None
Related topics	<p>References</p> <div> ds1006_ipi_disable..... 320 ds1006_ipi_enable_bm..... 321 </div>

ds1006_ipi_mask_set

Syntax	<pre>void ds1006_ipi_mask_set(int gl_no, UInt32 mask)</pre> <p>or</p> <pre>void ds1006_ipi_mask_set(int gl_no, UInt32 mask)</pre>
Include file	<code>ipi1006.h</code>
Purpose	To set the interrupt mask of the interrupt control unit.
Parameters	<p>gl_no Specifies the Gigalink number within the range 0 ... 3.</p> <p>mask Specifies the input interrupt mask to mask the incoming interrupts. A set bit disables an incoming interrupt, a clear bit enables it. Bit 0 affects interrupt 0, bit 1 interrupt 1, etc.</p>

Return value	None
Related topics	References <div> ds1006_ipi_mask_get..... 323 </div>

ds1006_ipi_mask_get

Syntax	<pre>UInt32 ds1006_ipi_mask_get(int gl_no)</pre> <p>or</p> <pre>UInt32 ds1006_ipi_mask_get(int gl_no)</pre>
Include file	ipi1006.h
Purpose	To get the interrupt mask of the interrupt control unit.
Parameters	gl_no Specifies the Gigalink number within the range 0 ... 3.
Return value	The current input interrupt mask. The interrupt mask masks the incoming interrupts. If a bit is set, the corresponding interrupt is disabled. If a bit is cleared, the corresponding interrupt is enabled. Bit 0 corresponds to interrupt 0, bit 1 to interrupt 1, etc.

Related topics	References <div> ds1006_ipi_mask_set..... 322 </div>
-----------------------	---

ds1006_ipi_sint_max_snd_set

Syntax	<pre>void ds1006_ipi_sint_max_snd_set(int gl_no, int max_ints)</pre>
---------------	---

or

```
void ds1006_ipi_sint_max_snd_set(
    int gl_no,
    int max_ints)
```

Include file	ipi1006.h
Purpose	To set the maximum number of software dispatched subinterrupts that can be sent from the given Gigalink.
Description	<p>Choose a value as low as possible to decrease the time needed to transfer the subinterrupt information.</p> <div> Note <p>On the receiving Gigalink the number of software dispatched subinterrupts has to be set to the same value with the function <code>ds1006_ipi_sint_max_rcv_set</code>.</p> </div>
Parameters	<p>gl_no Specifies the Gigalink number within the range 0 ... 3.</p> <p>max_ints Specifies the maximal number of software dispatched subinterrupts that can be sent.</p>
Return value	None
Related topics	<p>References</p> <div> ds1006_ipi_sint_max_rcv_set..... 324 </div>

ds1006_ipi_sint_max_rcv_set

Syntax

```
void ds1006_ipi_sint_max_rcv_set(
    int gl_no,
    int max_ints)
```

or

```
void dsgl_ipi_sint_max_rcv_set(
    int gl_no,
    int max_ints)
```

Include file `ipi1006.h`

Purpose To set the maximal number of software dispatched subinterrupts that can be received by the given Gigalink.

Description Choose a value as low as possible to decrease the time needed to transfer the subinterrupt information.

Note

On the sending Gigalink the number of software dispatched subinterrupts has to be set to the same value with the function `ds1006_ipi_sint_max_snd_set`.

Parameters

gl_no Specifies the Gigalink number within the range 0 ... 3.

max_ints Specifies the maximal number of software dispatched subinterrupts that can be sent.

Return value None

Related topics **References**

[ds1006_ipi_sint_max_snd_set](#)..... 323

ds1006_ipi_install_handler

Syntax

```
ipi_handler_type ds1006_ipi_install_handler(
    int gl_no,
    int int_no,
    ipi_handler_type handler)
```

or

```
ipi_handler_type dsgl_ipi_install_handler(  
    int gl_no,  
    int int_no,  
    ipi_handler_type handler)
```

Include file `ipi1006.h`

Purpose To install an interrupt service routine for an outgoing interrupt line or a software dispatched subinterrupt.

Note

This function cannot be used in connection with the RT Kernel or RTI.

Parameters

gl_no Specifies the Gigalink number within the range 0 ... 3.

int_no Specifies the outgoing interrupt line number within the range 0 ... 12 or software dispatched subinterrupt within the range 16 ... (15+**max_ints**). The variable **max_ints** is specified by the function **ds1006_ipi_sint_max_rcv_set**.

handler Specifies the address of the interrupt service routine. This function must not have an input parameter or a return value, meaning, **void ipi_handler_type(void)**.

Return value This function returns the address of the previous interrupt service routine at the given interrupt number.

Gigalink Communication

Introduction

This chapter contains the functions for transmitting data between multiple processor boards when you use DS1006 or DS1007 boards or between multiple processor cores when you use a DS1006 or DS1007 board.

For the Gigalink communication the following data type is used.

gl_scantbl_entry_t

Gigalink scantable entry type

```
typedef struct
{
    UInt32 target_board_snr; /* serial number of the board */
    UInt32 target_gl_no;    /* Gigalink number*/
    Int32 target_cpu_id;    /* CPU ID */
    UInt32 reserved;        /* reserved */
} gl_scantbl_entry_t;
```

Where to go from here

Information in this section

ds1006_gl_init.....	329
To initialize a Gigalink module.	
ds1006_gl_initialized.....	329
To check whether a Gigalink connection has been initialized.	
ds1006_gl_synchronized.....	330
To check whether a Gigalink connection is synchronized with another board.	
ds1006_gl_scanner_init.....	331
To initialize a Gigalink scanner.	
ds1006_gl_scan.....	332
To scan a Gigalink connection of a board.	
ds1006_gl_background_scan.....	333
To scan unused Gigalink connections within the background loop.	
ds1006_gl_write32.....	333
To write a 32-bit data word to the write buffer of a connected receiver.	
ds1006_gl_write32_and_switch.....	335
To write a 32-bit data word to the write buffer of a connected receiver and switch the buffer.	
ds1006_gl_write64.....	336
To write a 64-bit double word to the write buffer of a connected receiver.	
ds1006_gl_block_write.....	338
To write a data block to the write buffer of a connected receiver.	
ds1006_gl_write_buffer_switch.....	339
To switch the write buffer.	
ds1006_gl_read32.....	341
To read a 32-bit data word from the receiver buffer.	
ds1006_gl_read64.....	342
To read a 64-bit double word from the receiver buffer.	
ds1006_gl_block_read.....	343
To read a data block from the receiver buffer.	
ds1006_gl_read_buffer_switch.....	345
To switch the receiver buffer.	
ds1006_gl_read_buffer_is_updated.....	346
To check the status of the receiver write buffer.	
ds1006_gl_module_present.....	347
To check whether a Gigalink module is present or not.	
ds1006_gl_opto_signal_detect.....	347
To check if an optical signal at a Gigalink is detected.	

ds1006_gl_init

Syntax

```
int ds1006_gl_init()
```

or

```
int dsgl_init()
```

Include file

gl1006.h

Purpose

To initialize all four Gigalink modules and clear the receiver buffers.

Description

This function is called by **init()**.

Return value

This function returns one of the following values:

Value	Meaning
0	Gigalink module not present
1	Initialization done

ds1006_gl_initialized

Syntax

```
int ds1006_gl_initialized(UINT32 gl_no)
```

or

```
int dsgl_initialized(UINT32 gl_no)
```

Include file

gl1006.h

Purpose

To check whether a Gigalink connection has been initialized by the **ds1006_mp_init** function.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...

Predefined Symbol	Meaning
GL_3	Gigalink number 3

Return value

This function returns one of the following values:

Value	Meaning
0	Gigalink connection is not initialized
1	Gigalink connection is initialized

ds1006_gl_synchronized

Syntax

```
int ds1006_gl_synchronized(
    UInt32 gl_no,
    gl_scantbl_entry_t *gl_st_ptr)
```

or

```
int ds1006_dsgl_synchronized(
    UInt32 gl_no,
    gl_scantbl_entry_t *gl_st_ptr)
```

Include file

gl1006.h

Purpose

To check whether a Gigalink connection is synchronized with another board.

Description

This function has to be called in a loop for synchronizing the required Gigalink connections. If **gl_st_ptr** is specified, the serial number and the Gigalink number of the target board are stored.

The function is called by the **ds1006_init** function.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

gl_st_ptr Pointer to a structure **gl_scantbl_entry_t**, if the serial number and Gigalink number of target board are to be stored, or 0.

Return value

One of the following values:

Value	Meaning
1	Gigalink module is synchronized
0	Gigalink module is not synchronized

Example

```
gl_scantbl_entry_t gl_0_scan, gl_1_scan;
Int32 gl_0_synchronized = 0, gl_1_synchronized = 0;
init();
...
do
{
...
if (!gl_0_synchronized)
    gl_0_synchronized = ds1006_gl_synchronized(GL_0,&gl_0_scan);
if (!gl_1_synchronized)
    gl_1_synchronized = ds1006_gl_synchronized(GL_1,&gl_1_scan);
...
}
while(!(all_synchronized || timeout))
```

ds1006_gl_scanner_init

Syntax

```
vcm_module_descriptor_type * ds1006_gl_scanner_init(
    UInt32 parent_module)
```

or

```
vcm_module_descriptor_type * ds1006_dsgl_scanner_init(
    UInt32 parent_module)
```

Include file

gl1006.h

Purpose

To initialize a Gigalink scanner.

Description

This function initializes the Gigalink scanner and registers it at the VCM (version and config section management) module.

Parameters

parent_module Module ID of the parent module (see [Version and Config Section Management](#) on page 143)

Return value

The function returns the pointer to the VCM entry of the Gigalink, or NULL pointer, if the Gigalink module is not present.

ds1006_gl_scan

Syntax

```
int ds1006_gl_scan(  
    UInt32 gl_no,  
    gl_scantbl_entry_t *gl_st_ptr)
```

or

```
int dsgl_scan(  
    UInt32 gl_no,  
    gl_scantbl_entry_t *gl_st_ptr)
```

Include file

gl1006.h

Purpose

To scan a Gigalink connection of a board.

Description

This function is used to scan the Gigalink connections of a board.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

gl_st_ptr Pointer to a structure `gl_scantbl_entry_t`, if the serial number and Gigalink number of target board are to be stored, or 0.

Return value

One of the following values:

Value	Meaning
0	No connection detected
1	Connection detected

Related topics**References**

[ds1006_gl_background_scan..... 333](#)

ds1006_gl_background_scan

Syntax

```
void ds1006_gl_background_scan(void)
```

or

```
void dsgl_background_scan(void)
```

Include file

gl1006.h

Purpose

To scan unused Gigalinks for connections with other boards in the background of an application.

Description

The connections are stored in the additional config memory block of the Gigalink scanner module. This function is called by **RTLIB_BACKGROUND_SERVICE**.

Return value

None

Related topics**References**

[RTLIB_BACKGROUND_SERVICE..... 26](#)

ds1006_gl_write32

Syntax

```
void ds1006_gl_write32(
    int gl_no,
    int channel_no,
    int offset,
    Int32 data)
```

or

```
void dsgl_write32(
    int gl_no,
    int channel_no,
    int offset,
    Int32 data)
```

Include file `gl1006.h`

Purpose To write a 32-bit data word to the write buffer of a receiver at a specified buffer offset.

Note

Do not exceed the maximum buffer size of 2KW. For performance reasons it is not checked by this function.

Parameters **gl_no** Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode
SMEM_CH_0	Channel 0 with virtual shared memory mode. This channel is already used as a service channel by RTLib and RTI.
SMEM_CH_1	Channel 1 with virtual shared memory mode
...	...
SMEM_CH_7	Channel 7 with virtual shared memory mode

offset Specifies the word offset (32-bit) within the buffer.

data Specifies the 32-bit data word.

Return value None

Related topics

References

ds1006_gl_block_write.....	338
ds1006_gl_write32_and_switch.....	335
ds1006_gl_write64.....	336

ds1006_gl_write32_and_switch

Syntax

```
void ds1006_gl_write32_and_switch(
    int gl_no,
    int channel_no,
    int offset,
    Int32 data)
```

or

```
void dsgl_write32_and_switch(
    int gl_no,
    int channel_no,
    int offset,
    Int32 data)
```

Include file

gl1006.h

Purpose

To write a 32-bit data word to the write buffer of a receiver at a specified buffer offset and to switch the buffer.

Note

Do not exceed the maximum buffer size of 2KW. For performance reasons it is not checked by this function.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode
SMEM_CH_0	Channel 0 with virtual shared memory mode. This channel is already used as a service channel by RTLib and RTI.
SMEM_CH_1	Channel 1 with virtual shared memory mode
...	...
SMEM_CH_7	Channel 7 with virtual shared memory mode

offset Specifies the word offset (32-bit) within the buffer.

data Specifies the 32-bit data word.

Return value None

Related topics

References

[ds1006_gl_write32..... 333](#)

ds1006_gl_write64

Syntax

```
void ds1006_gl_write64(
    int gl_no,
    int channel_no,
    int offset,
    double data)
```

or

```
void ds1006_dgl_write64(
    int gl_no,
    int channel_no,
    int offset,
    double data)
```

Include file `gl1006.h`

Purpose

To write a 64-bit double value to the write buffer of a receiver at a specified buffer offset.

Note

Do not exceed the maximum buffer size of 2KW. For performance reasons it is not checked by this function.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode
SMEM_CH_0	Channel 0 with virtual shared memory mode. This channel is already used as a service channel by RTLib and RTI.
SMEM_CH_1	Channel 1 with virtual shared memory mode
...	...
SMEM_CH_7	Channel 7 with virtual shared memory mode

offset Specifies the word offset (32-bit) within the buffer.

data Specifies a 64-bit double value.

Return value

None

Related topics**References**

ds1006_gl_block_write	338
ds1006_gl_write32	333

ds1006_gl_block_write

Syntax

```
void ds1006_gl_block_write(
    int gl_no,
    int channel_no,
    int offset,
    int count,
    const void *data,
    int buf_switch)
```

or

```
void dsgl_block_write(
    int gl_no,
    int channel_no,
    int offset,
    int count,
    const void *data,
    int buf_switch)
```

Include file

gl1006.h

Purpose

To write a data block from a source buffer to the write buffer of a receiver.

Note

Do not exceed the maximum buffer size of 2KW. For performance reasons it is not checked by this function.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode
SMEM_CH_0	Channel 0 with virtual shared memory mode. This channel is already used as a service channel by RTLib and RTI.

Predefined Symbol	Meaning
SMEM_CH_1	Channel 1 with virtual shared memory mode
...	...
SMEM_CH_7	Channel 7 with virtual shared memory mode

offset Specifies the word offset (32-bit) within the sender buffer.

count Specifies the number of 32-bit words to be transmitted.

data Specifies the pointer to the data source buffer.

buf_switch Specifies the buffer switch by using one of the following constants:

Constant	Meaning
GL_BUF_SWITCH_ON	Switch buffer after write
GL_BUF_SWITCH_OFF	Do not switch buffer

This parameter is ignored if a channel in the virtual shared memory mode is used.

Return value None

Related topics

References

ds1006_gl_write32.....	333
ds1006_gl_write32_and_switch.....	335

ds1006_gl_write_buffer_switch

Syntax

```
void ds1006_gl_write_buffer_switch(
    int gl_no,
    int channel_no)
```

or

```
void ds1006_dsgl_write_buffer_switch(
    int gl_no,
    int channel_no)
```

Include file gl1006.h

Purpose To send a write buffer switch command to switch the receiver write buffer.

Remarks

The buffer is not switched if a channel in the virtual shared memory mode is used.

Note

Since the last memory location in the receiver buffer is cleared, this location may not be used for any data.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode
SMEM_CH_0	Channel 0 with virtual shared memory mode. This channel is already used as a service channel by RTLib and RTI.
SMEM_CH_1	Channel 1 with virtual shared memory mode
...	...
SMEM_CH_7	Channel 7 with virtual shared memory mode

Return value

None

Related topics**References**

[ds1006_gl_read_buffer_switch..... 345](#)

ds1006_gl_read32

Syntax

```
Int32 ds1006_gl_read32(
    int gl_no,
    int channel_no,
    int offset)
```

or

```
Int32 dsgl_read32(
    int gl_no,
    int channel_no,
    int offset)
```

Include file

gl1006.h

Purpose

To read a 32-bit word from a receiver read buffer.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode
SMEM_CH_0	Channel 0 with virtual shared memory mode. This channel is already used as a service channel by RTLib and RTI.
SMEM_CH_1	Channel 1 with virtual shared memory mode
...	...
SMEM_CH_7	Channel 7 with virtual shared memory mode

offset Specifies the word offset (32-bit) within the buffer.

Return value This function returns the 32-bit value to be read.

Related topics

References

[ds1006_gl_block_read](#)..... 343
[ds1006_gl_read64](#)..... 342

ds1006_gl_read64

Syntax

```
double ds1006_gl_read64(
    int gl_no,
    int channel_no,
    int offset)
```

or

```
double ds1006_dsgl_read64(
    int gl_no,
    int channel_no,
    int offset)
```

Include file

gl1006.h

Purpose

To read a 64-bit double value from a receiver read buffer.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode
SMEM_CH_0	Channel 0 with virtual shared memory mode. This channel is already used as a service channel by RTLib and RTI.

Predefined Symbol	Meaning
SMEM_CH_1	Channel 1 with virtual shared memory mode
...	...
SMEM_CH_7	Channel 7 with virtual shared memory mode

offset Specifies the word offset (32-bit) within the buffer.

Return value This function returns the 64-bit value to be read.

Related topics

References

[ds1006_gl_block_read](#)..... 343
[ds1006_gl_read32](#)..... 341

ds1006_gl_block_read

Syntax

```
void ds1006_gl_block_read(
    int gl_no,
    int channel_no,
    int offset,
    int count,
    void *data,
    int buf_switch)
```

or

```
void dsgl_block_read(
    int gl_no,
    int channel_no,
    int offset,
    int count,
    void *data,
    int buf_switch)
```

Include file `gl1006.h`

Purpose To copy a data block from a receiver read buffer to a destination buffer.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode
SMEM_CH_0	Channel 0 with virtual shared memory mode. This channel is already used as a service channel by RTLib and RTI.
SMEM_CH_1	Channel 1 with virtual shared memory mode
...	...
SMEM_CH_7	Channel 7 with virtual shared memory mode

offset Specifies the word offset (32-bit) within the buffer.

count Specifies the number of 32-bit words to be copied from the receiver buffer.

data Specifies the pointer to the destination buffer.

buf_switch Specifies the buffer switch by using one of the following constants:

Constant	Meaning
GL_BUF_SWITCH_ON	Switch buffer before read
GL_BUF_SWITCH_OFF	Do not switch buffer

This parameter is ignored if a channel in the virtual shared memory mode is used.

Return value

None

Related topics**References**

ds1006_gl_read32.....	341
ds1006_gl_read64.....	342

ds1006_gl_read_buffer_switch

Syntax

```
void ds1006_gl_read_buffer_switch(
    int gl_no,
    int channel_no)
```

or

```
void dsgl_read_buffer_switch(
    int gl_no,
    int channel_no)
```

Include file

gl1006.h

Purpose

To switch a receiver read buffer.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode

Return value

None

Related topics

References

[ds1006_gl_write_buffer_switch..... 339](#)

ds1006_gl_read_buffer_is_updated

Syntax

```
int ds1006_gl_read_buffer_is_updated(
    int gl_no,
    int channel_no)
```

or

```
int ds1006_dsgl_read_buffer_is_updated(
    int gl_no,
    int channel_no)
```

Include file

gl1006.h

Purpose

To check the status of the receiver read buffer.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

channel_no Specifies the channel number. Following symbols are predefined:

Predefined Symbol	Meaning
SBUF_CH_0	Channel 0 with swinging buffer mode
...	...
SBUF_CH_7	Channel 7 with swinging buffer mode
SMEM_CH_0	Channel 0 with virtual shared memory mode. This channel is already used as a service channel by RTLib and RTI.
SMEM_CH_1	Channel 1 with virtual shared memory mode
...	...
SMEM_CH_7	Channel 7 with virtual shared memory mode

Return value

This function returns the status of the receiver read buffer:

Value	Meaning
True	Receiver read buffer has been switched
False	Receiver read buffer has not been switched

In case of a shared memory channel (channel number 9 to 15) the return value is always true.

ds1006_gl_module_present

Syntax

```
int ds1006_gl_module_present(void)
```

or

```
int dsgl_module_present(void)
```

Include file

gl1006.h

Purpose

To check whether a Gigalink module is present or not.

Return value

This function returns one of the following values:

Value	Meaning
0	Gigalink module is not present
1	Gigalink module is present

Note

If you are using a multicore DS1006 board, this function always returns 1, because of its internal virtual gigalinks. For further information, refer to [Using DS1006 With Multicore Processor \(DS1006 Features !\[\]\(2bae76de5ebbd5c4d7d47162f1673734_img.jpg\)\)](#).

ds1006_gl_opto_signal_detect

Syntax

```
int ds1006_gl_opto_signal_detect(int gl_no)
```

or

```
int dsgl_opto_signal_detect(int gl_no)
```

Include file

gl1006.h

Purpose

To check if an optical signal at a Gigalink is detected.

Parameters

gl_no Specifies the Gigalink number. Following symbols are predefined:

Predefined Symbol	Meaning
GL_0	Gigalink number 0
...	...
GL_3	Gigalink number 3

Return value

This function returns one of the following values:

Value	Meaning
0	No optical signal detected
1	Optical signal detected

Host Programs

Introduction	There are some utilities installed on the host PC for building and debugging custom applications.
--------------	---

Where to go from here

Information in this section

Host Settings.....	350
For information about the necessary settings of the software environment and the DS1006 Real-Time Library.	
Compiling, Linking and Downloading an Application.....	354
Information about the batch files, makefiles, and linker command files that support your program development.	
Debugging an Application.....	362
Information about disassembling via <code>i686-elf-objdump</code> .	

Information in other sections

Firmware Manager Manual
Introduces you to the features provided by the Firmware Manager. It provides detailed information on the user interface, its command line options and instructions using the firmware management.

Host Settings

Introduction

This chapter describes the definitions, settings, files and libraries that are necessary to write your own C-coded programs for the AMD Opteron™ processor of DS1006 Processor Board.

Where to go from here

Information in this section

Compiler and C Run-Time Libraries.....	350
Environment Variables and Paths.....	350
Folder Structure.....	351
DS1006 Real-Time Library.....	351
File Extensions.....	352

Compiler and C Run-Time Libraries

Compiler and C run-time libraries

The DS1006 Compiler is automatically installed when you install the dSPACE software. The associated C run-time libraries are also used. The C compiler and the associated run-time libraries are located in the folders during compiler installation: for example, `c:\<name of the compiler>\`. This folder should be located in the root folder. White spaces in paths are permitted. The path is represented by the environment variable `%X86_ROOT%`. For further information, see the documentation provided with the GNU Compiler Collection.

For information on the C++ support, refer to [Integrating C++ Code](#) on page 361.

Environment Variables and Paths

dSPACE command prompt

The dSPACE software installation does not set environment variables and other settings such as enhancements to the search path.

Use the Command Prompt for dSPACE RCP and HIL for the host tools. You find the command prompt as a shortcut in the Windows Start menu. The required paths and environment settings are then automatically set.

Folder Structure

Folder structure

The folder structure of the DS1006 software is as follows:

Folder	Contents
<RCP_HIL_InstallationPath>\DS1006\RTLib	Source and library files of the DS1006 Real-Time Library, makefiles, and linker command file
<RCP_HIL_InstallationPath>\Exe	Batch files for manually programming the DS1006, host programs
<RCP_HIL_InstallationPath>\Demos\DS1006	Demo examples
<RCP_HIL_InstallationPath>\Demos\DS1006mp	Demo examples for multiprocessing

DS1006 Real-Time Library

DS1006.lib

All functions of the DS1006 Real-Time Library were compiled with the highest optimization level and collected in the library **ds1006.lib**. Required objects from this library are automatically linked to any application when **Ds1006.lk** is used for linking. The header files are located in <RCP_HIL_InstallationPath>DS1006\RTLib.

Note

All necessary modules and header files are included by **Brtenv.h**.

The following table shows some modules that are included in the library **ds1006.lib**:

Module (Header File)	Contents
brtenv.h	Basic real-time environment
cfg1006.h	Defines for config section access
dpm1006.h	Defines for dual port memory access
ds1006.h	Addresses and error codes
dsmcom.h, dsmcom_fphs.h	General master-slave communication
dsmg.h, dsmgprn.h	Message module support
dsmem.h	Memory management of global RAM
dsphs.h	Defines for PHS bus I/O access
dserr.h, dsfifo.h	Serial interface
dssint.h, sint1006.h	Subinterrupt handling
dsstd.h	Standard definitions

Module (Header File)	Contents
<code>dsts.h</code>	Time-stamping
<code>dstypes.h</code>	dSPACE type definitions
<code>dsvcm.h</code>	Version and config section management
<code>fc1006.h</code>	Floating point conversion
<code>hsvc1006.h</code>	Host service support
<code>info1006.h</code>	Information from config section
<code>init1006.h</code>	Initialization functions
<code>int1006.h</code>	Interrupt handling
<code>phsint.h</code>	PHS bus interrupt handling
<code>rmserv.h</code>	RealMotion support
<code>tic1006.h</code>	Time measurement and delay
<code>tmr1006.h</code>	Timer access functions (Timer A, Timer B, Timer D, Time Base)
<code>wd1006.h</code>	Watchdog timer

All required libraries, for example, for the supported I/O boards, are dynamically linked to the real-time application.

Multiprocessor system

The following modules for a multiprocessor system are included:

Module (Header File)	Contents
<code>mp1006.h</code>	Multiprocessor functions
<code>gl1006.h</code>	Gigalink communication
<code>ipi1006.h</code>	Interprocessor interrupts

File Extensions

File extensions

The following file extensions are used:

File Extension	Meaning
<code>.c¹⁾</code>	C source files
<code>.lib</code>	Library files
<code>.lk</code>	Linker command file
<code>.mk</code>	Makefiles
<code>.o86</code>	Relocatable object files for the AMD Opteron™ processor
<code>.obj</code>	Object files

File Extension	Meaning
.x86	Executable programs for the AMD Opteron™ processor

¹⁾ For C++ support, refer to [Integrating C++ Code](#) on page 361.

Compiling, Linking and Downloading an Application

Introduction

If you want to build a user application and download it to the target hardware, you can use the **Down** tool for your board.

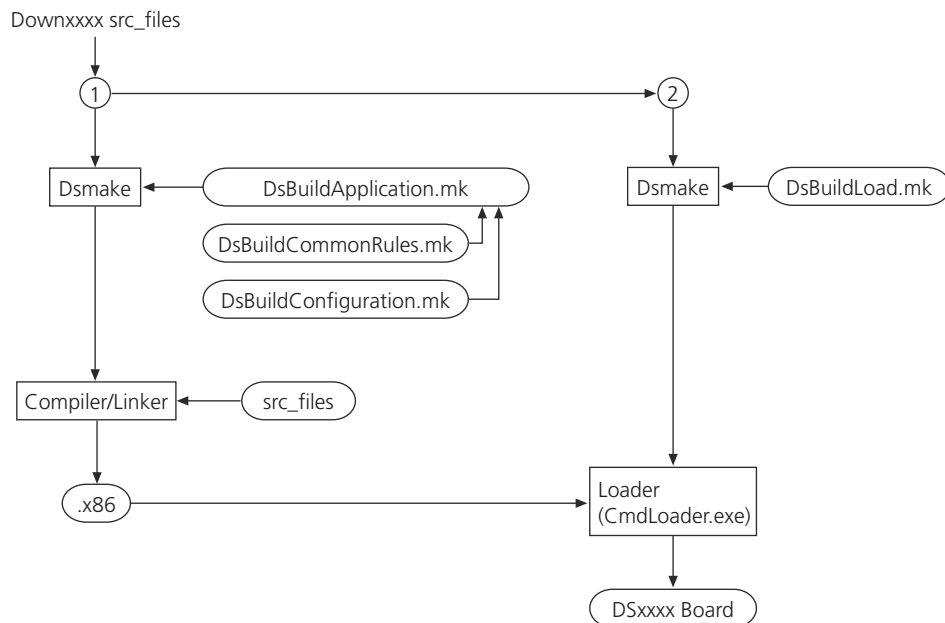
Tip

The executable file Down1006 can be called in a Command Prompt window (DOS window) of your host PC.

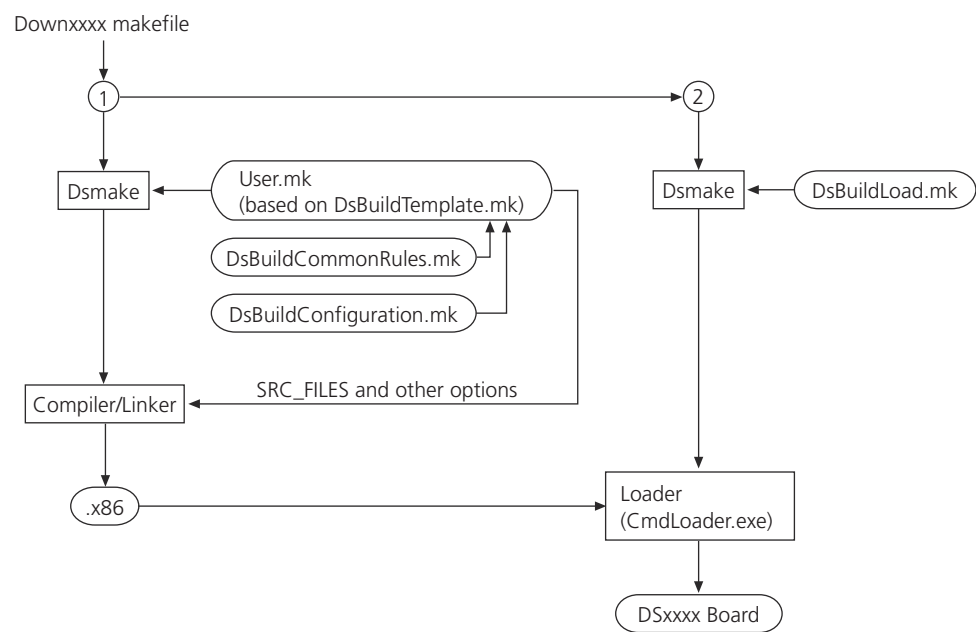
If you use the **Command Prompt for dSPACE RCP and HIL** shortcut in the Windows Start menu, the required paths and environment settings are automatically set.

Process overview

The following schematic shows you the process overview for using Down with the source files as arguments. **DsBuildApplication.mk** is then used for the make process.



The following schematic shows you the process overview for using Down with the custom makefile as an argument. It is recommended to base the makefile on **DsBuildTemplate.mk**.



Where to go from here

Information in this section

Down1006.exe.....	355
To compile, link, and download applications.	
DsBuildApplication.mk.....	358
This is the default makefile if you use Down with source files as arguments.	
DsBuildLoad.mk.....	359
To download an application to the target hardware.	
DsBuildTemplate.mk.....	359
This is a template for a custom makefile.	
Ds1006.lk.....	360
To link an application.	
Integrating C++ Code.....	361
Gives you instructions on enabling the C++ support.	

Down1006.exe

Syntax

```
down1006 file.mk [options] [/?]
```

or

```
down1006 src_file(s) [options] [/?]
```

Purpose

To compile or assemble, link, and download handcoded applications.

Description

The following file types can be handled:

Local makefile (.mk) To compile, link, and download the application using the specified local makefile. Use the makefile DsBuildTemplate.mk as a template to write your own makefile. The resulting program file is named according to the name of the specified makefile.

C-coded source file (.c) To specify the file(s) to be compiled and linked using DsBuildApplication.mk. The resulting program file is named according to the name of the first specified source file.

Note

If you use the Down tool with source files, the relocatable object files are deleted and the object file of the application is overwritten. If you call the Down tool with a user makefile as the argument, the object files remain unchanged until a modified source file requires recompilation.

If the file name extension is omitted, **Down1006** searches for existing files in the above order. If more than one source file is specified at the command line, the first file is treated as the main source file that names the complete application. The remaining source files are compiled or assembled, and linked to the application.

The built application is loaded by default to the DS1006 platform named 'ds1006'. The platform name is set by the dSPACE software, e.g., ControlDesk during platform registration. If you want to access another platform instead, you can specify the platform name using the /p option.

For a graphical process overview, refer to [Compiling, Linking and Downloading an Application](#) on page 354.

For further information on the C++ support, refer to [Integrating C++ Code](#) on page 361.

Options

The following command line options are available:

Option	Meaning
/co <option>	To specify additional compiler options; refer to the GNU Compiler documentation.
/d	To disable downloading the application; only compiling and linking.
/g	To compile for source level debugging; Ds1006dbg.lib is used for linking.
/l	To write all output to down1006.log .
/lib <lib_file>	To specify an additional library to be linked.

Option	Meaning
/lko <option>	To specify a single additional linker option.
/lo <option>	To specify a single additional loader option.
/mo <option>	To specify a single additional DSMAKE option; call dsmake -h to get more information.
/n	To disable beep on error.
/p <PlatformName>	To specify a platform name that differs from the default. The default platform name is ds1006 if you call Down1006.exe .
/pause	To pause execution of Down1006 before exit.
/r	To register the platform specified by the /p option.
/x	To switch off code optimization.
/z	To download an existing object file without building.
/?	To display information.

Messages

The following messages are defined:

Message	Description
ERROR: not enough memory!	The attempt to allocate dynamic memory failed.
ERROR: environment variable PPC_ROOT not found! ERROR: environment variable X86_ROOT not found! ERROR: environment variable DSPACE_ROOT not found!	The respective environment variable is not defined in the DOS environment. The environment variables are set during the dSPACE software installation.
ERROR: can't load DLL '%DSPACE_ROOT %/exe/wbinfo.dll'! [number]	Loading the dynamic link library WBINFO.DLL failed. The number in brackets specifies the internal Windows error.
ERROR: can't read address of function 'GetWorkingBoardName()' ERROR: can't read address of function 'GetWorkingBoardClient()' ERROR: can't read address of function 'GetWorkingBoardConnection()' ERROR: can't read address of function 'GetWorkingBoardType()'	The address of the respective function could not be found in the dynamic link library WBINFO.DLL.
ERROR: can't read working board name! ERROR: can't read working board client! ERROR: can't read working board connection! ERROR: can't read working board type!	The respective working board information could not be read from the dspace.ini file. Register your hardware system using ControlDesk's Platform Manager.
WARNING: The working board type is DS???? instead of DS????! Accessing default board ds????.	The detected working board type is not responding to the DOWN1006 version. For example, if you are using DOWN1006 and the working board is of type DS1104, the board name ds1006 is used.
ERROR: unable to obtain full path of <file name>!	This error occurs if the full path name of the source file contains more than 260 characters, or if an invalid drive letter has been specified, for example, 1:\test.

Message	Description
ERROR: unable to access file <file name>!	The specified file cannot be accessed by Down1006. The file does not exist or another application is accessing it.
ERROR: source files must be available in the same directory!	All source files to be compiled must be available in the same application folder.
ERROR: make file <name> not allowed as additional source file!	Only assembly and C source files are allowed as additional source files.
ERROR: can't redirect stdout to file! ERROR: can't redirect stdout to screen!	The redirection of stdout to a file or to the screen has failed.
ERROR: can't invoke %DSPACE_ROOT %\exe\dsmake.exe: ...	Down1006 was not able to invoke DSMAKE.EXE successfully.
ERROR: making of <file name> failed! ERROR: building of <file name> failed!	An error occurred while executing a makefile, compiling or assembling a source file. See the screen output to get information about the reasons, for example, there can be programming errors in the source file.
ERROR: downloading of <file name> failed!	DOWN1006 was not able to download the application successfully. See dSPACE.log for more information.
ERROR: can't install exit handler!	The available memory space is too small for registering the exit handler.

Related topics

References

DsBuildApplication.mk.....	358
DsBuildLoad.mk.....	359
DsBuildTemplate.mk.....	359

DsBuildApplication.mk

Description

This makefile is used to compile or assemble the application source files. It is called by Down1006.exe if no other makefile is specified. It uses the highest optimization level of the C compiler.

It includes:

- DsBuildCommonRules.mk
- DsBuildConfiguration.mk

Note

Do not edit.
Use the required option with Down1006 or a custom makefile based on DsBuildTemplate.mk instead.

Related topics

References

Down1006.exe.....	355
DsBuildTemplate.mk.....	359

DsBuildLoad.mk

Description

This file is automatically invoked by Down1006.exe to load the application to the target hardware after building, unless you use the /d option. It is also called if you use the /z option for download only.

Note

Do not edit or change this file.

Related topics

References

Down1006.exe.....	355
-----------------------------------	---------------------

DsBuildTemplate.mk

Description

If you want to call Down1006.exe with a custom makefile as an argument, you can use this makefile as a template for it. Copy this file to a local folder, rename it <application_name>.mk, and edit the intended sections before calling it with the Down tool. This makefile uses the highest optimization level of the C compiler.

You can customize this makefile to match your individual requirements:

- CUSTOM_SRC_FILES

You can add additional source files to be compiled by adding the names of the source files.

- **CUSTOM_OBJ_FILES**
You can add additional object files to be linked to the application by adding the names of the object files.
- **CUSTOM_LIB_FILES**
You can add additional libraries to be linked to the application by adding the names of the libraries.
- **CUSTOM_C_OPTS**
You can add additional options for the C compiler.
- **CUSTOM_ASM_OPTS**
You can add additional options for the assembler.
- **CUSTOM_LK_OPTS**
You can add additional options for the linker.
- **USER_BUILD_CPP_APPL**
You can enable the C++ support by setting this make macro to **ON**. For further information, refer to [Integrating C++ Code](#) on page 361.

Related topics**References**

Down1006.exe	355
------------------------------------	-----

Ds1006.lk

Description

This linker command file is automatically used by DsBuildApplication.mk or the custom makefile based on DsBuildTemplate.mk to link DS1006 x86 applications. It does not depend on your application or hardware. Thus you do not have to make any changes to this file. The currently available memory configuration of the board is automatically detected during start-up.

Related topics**References**

Down1006.exe	355
DsBuildApplication.mk	358
DsBuildTemplate.mk	359

Integrating C++ Code

Introduction

To integrate C++ code to your handcoded RTLib application, you have to enable the C++ support.

Adapting the user makefile

For adding C++ code to your application you have to adapt the `DsBuildTemplate.mk` file.

- Enable the C++ support
- Add C++ source files, C++ object files and C++ libraries

Example:

```
# Enable C++ support
USER_BUILD_CPP_APPL = ON
...
# Additional C/C++ source files to be compiled
CUSTOM_SRC_FILES = main.c example.cpp
...
# Additional user object files to be linked
USER_OBJS = MyModule3.o86 MyModule4.cppo86
...
# Additional user libraries to be linked
USER_LIBS = MyCLib.lib MyCppLib.lib
```

For further information on the user makefile, refer to [DsBuildTemplate.mk](#) on page 359.

Debugging an Application

Introduction

Simple application errors can be found by implementing messages in your source code to log measured or calculated values of variables (refer to [Message Handling](#) on page 160).

Run-time errors like exceptions can be investigated by disassembling the application. This identifies the source code that corresponds to a faulty memory location.

For information on relevant RTLib functions for handling exceptions, refer to [Exception Handling](#) on page 137. For detailed information about exceptions, refer to [Handling Exceptions \(RTI and RTI-MP Implementation Guide\)](#).

i686-elf-objdump

Syntax

```
i686-elf-objdump [options] objfile
```

Purpose

To display information about one or more object files.

Description

This utility is mainly used for debugging purposes. For example, it can disassemble an object file and show the machine instructions with their memory locations. The display of particular information is controlled by command line options. At least one option besides `-l` (`--line-numbers`) must be specified.

Note

To make it possible for `i686-elf-objdump` to display correlating source code information, you must build your application with the debug option `-g`.

You can find this utility in
`<RCP_HIL_InstallationPath>\Compiler\x86tools\bin.`

Options

The following command line options are available:

Option	Meaning
-a	Shows object file format and header information from an archive object file.
--archive-headers	
--adjust-vma=<offset>	Adds offset to all the section addresses. This is useful for dumping information, if the section addresses do not correspond to the symbol table.

Option		Meaning
-b <bfdname>	--target=<bfdname>	Specifies the object code format as bfdname . This might not be necessary, because many formats can be recognized automatically. You can list the available formats with -i .
-g	--debugging	Displays debugging information using a C-like syntax.
-C	--demangle	Decodes low-level symbol names into user-level names.
-d	--disassemble	Displays the assembler mnemonics for the machine instructions from sections which are expected to contain instructions.
-D	--disassemble-all	Displays the assembler mnemonics for the machine instructions from all sections.
-z	--disassemble-zeroes	Also disassembles blocks of zeros.
	--prefix-addresses	Prints the complete address of the disassembled code on each line. This is the older disassembly format.
-EB	--endian=big	Specifies the object file as big endian.
-EL	--endian=little	Specifies the object file as little endian.
-f	--file-headers	Displays summary information from the overall header of each file on objfile .
-h	--section-headers --headers	Displays summary information from the section headers of the object file.
-H	--help	Displays the objdump usage.
-i	--info	Displays a list showing all architectures and object formats available for specification with -b or -m .
-j <section>	--section=<section>	Displays information only for the specified section.
-l	--line-numbers	Labels the display with the file name and the source line numbers corresponding to the object code shown. This option is useful only with -d or -D .
-m <machine>	--architecture=<machine>	Specifies the architecture the object file is for. You can list the supported architectures by using -i .
-p	--private-headers	Displays information that is specific to the object file format.
-r	--reloc	Displays the relocation entries of the object file. If used with -d or -D , the relocations are printed interspersed with the disassembly.
-R	--dynamic-reloc	Displays the dynamic relocation entries of the object file. This is only useful for dynamic objects, such as certain types of shared libraries.
-s	--full-contents	Displays the full contents of any sections requested.
-S	--source	Displays source code intermixed with disassembly, if possible. This option implies -d .
	--show-raw-insn	Displays disassembled instructions in HEX as well as in symbolic form.
	--no-show-raw-insn	Does not display the instruction bytes of disassembled instructions.

Option		Meaning
-G	--stabs	Displays the contents of .stab, .stab.index and .stab.excl sections from an ELF file.
	--start-address=<address>	Starts displaying at the specified address. This affects the output of the -d , -r and -s options.
	--stop-address=<address>	Stops displaying at the specified address. This affects the output of the -d , -r and -s options.
-t	--syms	Displays the symbol table entries of the object file.
-T	--dynamic-syms	Displays the dynamic symbol table entries of the object file. This is only useful for dynamic objects, such as certain types of shared libraries.
-w	--wide	Formats some lines for output devices that have more than 80 columns.
-v	--version	Displays the version number.
-x	--all-headers	Displays all available header information, including the symbol table and relocation entries. This option implies -a , -f , -h , -r and -t .

Example

For debugging an application, it is useful to disassemble all sections together with information on the line numbers and corresponding source code of the displayed assembler instructions. `i686-elf-objdump` prints a great amount of data, so it is recommended to redirect the output to a dump file, which you can open with a text editor. The command looks like this:

```
i686-elf-objdump -S -l -D appl.x86 > result.dmp
```

A

A/D conversion 254
 access error 137
 alignment error 137
 alloc_phs_int_line 286
 application
 debugging 362

B

base address 258
 Bit I/O 255
 board version 139
 bus clock
 kHz 141

C

clock information
 kHz 141
 Common Program Data folder 14
 compiler and C run-time libraries 350
 compiling an application 354
 CPU clock
 kHz 141
 CPU temperature 142

D

D/A conversion 255
 data type
 ds1006_ISR 193
 ds1006_LSR 195
 ds1006_MSR 196
 ds1006_subint_handler_t 197
 ds1006_Channel 198
 debug messages 18
 DEBUG_INIT 18
 DEBUG_POLL 18
 declare_phs_int_line 285
 deinit_phs_int 292
 deinstall_phs_int_vector 284
 disable_phs_int 290
 Documents folder 14
 down1006 355
 downloading an application 354
 ds1006.lk 360
 ds1006_begin_isr_timerA 85
 ds1006_begin_isr_timerB 85
 ds1006_begin_isr_timerD 86
 ds1006_cache_load 20
 ds1006_conv_float_to_int32 237
 ds1006_conv32_ieee_to_ti 236
 ds1006_conv32_ti_to_ieee 237
 ds1006_disable_hardware_int 97
 ds1006_disable_hardware_int_bm 98
 ds1006_enable_hardware_int 99
 ds1006_enable_hardware_int_bm 101
 ds1006_end_isr_timerA 87
 ds1006_end_isr_timerB 87
 ds1006_end_isr_timerD 88

ds1006_get_interrupt_flag 102
 ds1006_get_interrupt_flag_bm 104
 ds1006_get_interrupt_status 105
 ds1006_get_interrupt_vector 106
 ds1006_gl_background_scan 333
 ds1006_gl_block_read 343
 ds1006_gl_block_write 338
 ds1006_gl_init 329
 ds1006_gl_initialized 329
 ds1006_gl_module_present 347
 ds1006_gl_opto_signal_detect 347
 ds1006_gl_read_buffer_is_updated 346
 ds1006_gl_read_buffer_switch 345
 ds1006_gl_read32 341
 ds1006_gl_read64 342
 ds1006_gl_scan 332
 ds1006_gl_scanner_init 331
 ds1006_gl_synchronized 330
 ds1006_gl_write_buffer_switch 339
 ds1006_gl_write32 333
 ds1006_gl_write32_and_switch 335
 ds1006_gl_write64 336
 DS1006_GLOBAL_INTERRUPT_DISABLE 112
 DS1006_GLOBAL_INTERRUPT_ENABLE 113
 ds1006_info_board_version_get 139
 ds1006_info_clocks_khz_get 141
 ds1006_info_cpu_temperature_get 141
 ds1006_info_memory_get 140
 ds1006_init 18
 ds1006_ipi_acknowledge 318
 ds1006_ipi_configure 316
 ds1006_ipi_disable 320
 ds1006_ipi_disable_bm 321
 ds1006_ipi_enable 319
 ds1006_ipi_enable_bm 321
 ds1006_ipi_init 316
 ds1006_ipi_install_handler 325
 ds1006_ipi_interrupt 317
 ds1006_ipi_mask_get 323
 ds1006_ipi_mask_set 322
 ds1006_ipi_sint_max_rcv_set 324
 ds1006_ipi_sint_max_snd_set 323
 ds1006_mp_begin_isr_global_srt 313
 ds1006_mp_end_isr_global_srt 314
 ds1006_mp_global_srt_init 310
 ds1006_mp_init 301
 ds1006_mp_optional_cpu_reduce 307
 ds1006_mp_route_mat 305
 ds1006_mp_route_syncin 305
 ds1006_mp_route_syncout 306
 ds1006_mp_start_isr_global_srt 311
 ds1006_mp_synchronize 303
 ds1006_reset_interrupt_flag 107
 ds1006_reset_interrupt_flag_bm 108
 ds1006_serial_config 228
 ds1006_serial_free 230
 ds1006_serial_init 227
 ds1006_serial_receive 231
 ds1006_serial_register_read 233
 ds1006_serial_register_write 232
 ds1006_serial_transmit 230
 ds1006_set_interrupt_status 109
 ds1006_set_interrupt_vector 110
 ds1006_slv_boot_finished 20
 ds1006_start_isr_timerA 89
 ds1006_start_isr_timerB 90
 ds1006_start_isr_timerD 91
 ds1006_tic_continue 33
 ds1006_tic_count 34
 ds1006_tic_delay 35
 ds1006_tic_diff 35
 ds1006_tic_elapsed 36
 ds1006_tic_halt 37
 ds1006_tic_read 38
 ds1006_tic_start 39
 ds1006_tic_total_read 40
 ds1006_timebase_fltread 40
 ds1006_timebase_low_read 41
 ds1006_timebase_read 42
 ds1006_timerA_period_reload 69
 ds1006_timerA_period_set 68
 ds1006_timerA_read 69
 ds1006_timerA_start 70
 ds1006_timerA_stop 70
 ds1006_timerB_compare_set 74
 ds1006_timerB_compare_set_periodically 75
 ds1006_timerB_init 73
 ds1006_timerB_read 76
 ds1006_timerB_start 76
 ds1006_timerB_stop 77
 ds1006_timerD_period_reload 80
 ds1006_timerD_period_set 79
 ds1006_timerD_read 80
 ds1006_timerD_start 81
 ds1006_timerD_stop 82
 ds1006_wd_init 184
 ds1006_wd_set 183
 ds1006_wd_stop 185
 ds1006_wd_strobe 186
 ds1006_wd_system_was_reset 186
 DsBuildApplication.mk 358
 DsBuildLoad.mk 359
 DsBuildTemplate.mk 359
 dsgl_background_scan 333
 dsgl_block_read 343
 dsgl_block_write 338
 dsgl_init 329
 dsgl_initialized 329
 dsgl_ipi_acknowledge 318
 dsgl_ipi_configure 316
 dsgl_ipi_disable 320
 dsgl_ipi_disable_bm 321
 dsgl_ipi_enable 319
 dsgl_ipi_enable_bm 321
 dsgl_ipi_init 316
 dsgl_ipi_install_handler 325
 dsgl_ipi_interrupt 317
 dsgl_ipi_mask_get 323
 dsgl_ipi_mask_set 322
 dsgl_ipi_sint_max_rcv_set 324
 dsgl_ipi_sint_max_snd_set 323
 dsgl_module_present 347

dsgl_mp_begin_isr_global_srt 313
 dsgl_mp_end_isr_global_srt 314
 dsgl_mp_global_srt_init 310
 dsgl_mp_init 301
 dsgl_mp_optional_cpu_reduce 307
 dsgl_mp_route_mat 305
 dsgl_mp_route_syncin 305
 dsgl_mp_route_syncout 306
 dsgl_mp_start_isr_global_srt 311
 dsgl_mp_synchronize 303
 dsgl_opto_signal_detect 347
 dsgl_read_buffer_is_updated 346
 dsgl_read_buffer_switch 345
 dsgl_read32 341
 dsgl_read64 342
 dsgl_scan 332
 dsgl_scanner_init 331
 dsgl_synchronized 330
 dsgl_write_buffer_switch 339
 dsgl_write32 333
 dsgl_write32_and_switch 335
 dsgl_write64 336
 dsr_bytes2word 224
 dsr_config 203
 dsr_disable 212
 dsr_enable 212
 dsr_error_read 213
 dsr_fifo_reset 211
 dsr_free 202
 dsr_handle_get 217
 dsr_init 201
 dsr_ISR 193
 dsr_LSR 195
 dsr_MSR 196
 dsr_receive 208
 dsr_receive_fifo_level 215
 dsr_receive_term 209
 dsr_set 218
 dsr_status_read 216
 dsr_subint_disable 221
 dsr_subint_enable 220
 dsr_subint_handler_inst 219
 dsr_subint_handler_t 197
 dsr_transmit 206
 dsr_transmit_fifo_level 214
 dsr_word2bytes 222
 dsrChannel 198
 dsint_acknowledge 134
 dsint_decode 133
 dsint_define_int_receiver 127
 dsint_define_int_receiver_1 129
 dsint_define_int_sender 123
 dsint_define_int_sender_1 125
 dsint_interrupt 133
 dsint_subint_disable 131
 dsint_subint_enable 132
 dsint_subint_reset 135

E

elementary data types 17
 enable_phs_int 289

environment variables and paths 350
 error messages 161
 example
 using time measurement functions 32
 exception handling 137
 execution times 249
 exit 18

F

folder structure 351
 free_phs_int_line 287

G

get_peripheral_addr 260
 global_disable 112
 global_enable 113

H

host programs 349
 host settings 350
 host_service 23

I

I/O boards overview 254
 I/O error line 258
 i686-elf-objdump 362
 ICU 271
 information
 handling 139
 information messages 161
 init() 244
 init_phs_int 291
 initialization
 of a DS1006 board 18
 of a multiprocessor system 298
 install_phs_int_vector 282
 integration of FPGA applications 257
 interface boards 256
 interrupt
 flag 102
 handling 95
 receiver 117
 sender 117
 status 105
 interrupt control unit 271
 interrupt vector table 274

L

linking an application 354
 Local Program Data folder 14

M

macro definition 242
 master_cmd_server 25
 message
 module 160
 message buffer 161
 message handling 160, 161

message length 161
 message type 162
 msg_default_dialog_set 174
 msg_error_clear 179
 msg_error_hook_set 180
 msg_error_printf 168
 msg_error_set 164
 msg_info_printf 171
 msg_info_set 166
 msg_init 181
 msg_last_error_number 177
 msg_last_error_submodule 178
 msg_mode_set 175
 msg_printf 172
 msg_reset 176
 msg_set 166
 msg_warning_printf 170
 msg_warning_set 165

O

overview
 I/O boards 254

P

PHS bus
 handling 258
 PHS_BOARD_BASE_GET 261
 phs_board_type_from_slot_get 262
 phs_board_type_get 261
 PHS_IO_ERROR_SET 266
 PHS_IO_ERROR_STATE 265
 PHS_REGISTER_PTR 265
 PHS_REGISTER_READ 263
 PHS_REGISTER_WRITE 264
 PHS_SYNC_TIMER_SET 268
 PHS_SYNC_TRIGGER 268
 PHS_SYNCIN_TRIGGER 267
 PHS_SYNCOUT_TRIGGER 267
 PHS-bus
 interrupt functions 270
 interrupt system 271
 interrupt vector table 274
 processor core modules 15
 program error 137

R

receive buffer 189
 receiver type definition 122
 rtlib_background_hook 27
 rtlib_background_hook_process 28
 RTLIB_BACKGROUND_SERVICE 26
 RTLIB_CALLOC_PROT 246
 RTLIB_CONV_FLOAT_TO_SATURATED_INT32 240
 RTLIB_CONV_FLOAT32_FROM_IEEE32 238
 RTLIB_CONV_FLOAT32_FROM_TI32 239
 RTLIB_CONV_FLOAT32_TO_IEEE32 239
 RTLIB_CONV_FLOAT32_TO_TI32 240
 RTLIB_EXIT 245
 RTLIB_FORCE_IN_ORDER 234

RTLIB_FREE_PROT 247
 RTLIB_GET_SERIAL_NUMBER 245
 RTLIB_INT_DISABLE 112
 RTLIB_INT_ENABLE 113
 RTLIB_INT_RESTORE 114
 RTLIB_INT_SAVE_AND_DISABLE 114
 RTLIB_MALLOC_PROT 246
 RTLIB_REALLOC_PROT 247
 RTLIB_SLAVE_LOAD_ACKNOWLEDGE 20
 RTLIB_SRT_DISABLE 115
 RTLIB_SRT_ENABLE 116
 RTLIB_SRT_ISR_BEGIN 92
 RTLIB_SRT_ISR_END 93
 RTLIB_SRT_PERIOD 67
 RTLIB_SRT_START 93
 RTLIB_SYNC 235
 RTLIB_TIC_CONTINUE 42
 RTLIB_TIC_COUNT 43
 RTLIB_TIC_DELAY 44
 RTLIB_TIC_DIFF 45
 RTLIB_TIC_ELAPSED 46
 RTLIB_TIC_HALT 47
 RTLIB_TIC_READ 48
 RTLIB_TIC_READ_TOTAL 49
 RTLIB_TIC_START 49

S

sampling rate timer 243
 sender type definition 122
 serial interface communication 188
 set_phs_int_mask 293
 special I/O 257
 STBU 302
 subinterrupt
 serial communication 190
 subinterrupt handling 117
 Synchronous Time Base Unit 302
 SYNCIN line 258
 SYNCOUT line 259

T

temperature of CPU 142
 Time Base Counter 30
 time interval measurement 30
 time measurement example 32
 time stamping module 51
 timebase counter 30
 Timer A 66
 example 66
 Timer B 72
 example 72
 Timer D 78
 timer interrupt control 83
 timing I/O 256
 transmit buffer 189
 trigger level 189
 ts_init 57
 ts_reset 58
 ts_time_calculate 63
 ts_time_offset 62

ts_time_read 59
 ts_timestamp_calculate 64
 ts_timestamp_compare 60
 ts_timestamp_interval 61
 ts_timestamp_offset 63
 ts_timestamp_read 59

U

UART 188

V

VCM module 143
 vcm_cfg_malloc 151
 vcm_init 149
 vcm_memory_ptr_get 152
 vcm_memory_ptr_set 152
 vcm_module_find 153
 vcm_module_register 149
 vcm_module_status_get 155
 vcm_module_status_set 154
 vcm_module_version_print 158
 vcm_version_compare 157
 vcm_version_get 156
 vcm_version_print 159

W

warning messages 161
 watchdog 183

