DS1104 R&D Controller Board

RTLib Reference

Release 2021-A – May 2021

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

Contents

Host Programs 389

Index 405

About This Reference

Content

This RTLib Reference (Real-Time Library) gives detailed descriptions of the C functions needed to program a DS1104 R&D Controller Board. The C functions can be used to program RTI-specific Simulink S-functions, or to implement your control models manually using C programs.

Symbols

dSPACE user documentation uses the following symbols:

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

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.

Master PPC

Introduction

This section describes the elementary data types, the overall PowerPC functions (arranged according to the tool they are associated with) and the functions you need to program the I/O units directly served by the master PowerPC.

Where to go from here

Information in this section

Data Types and Definitions

Where to go from here

Information in this section

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 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 Float32 *                             Float32Ptr
typedef Float64 *                             Float64Ptr
```

Include file

`dstypes.h`

Standard Definitions

Following, there are some board-specific definitions. For macros, that are independent from the board used, refer to Standard Macros on page 296.

Constants, macros, and type definitions for the DS1104

```
#define global_enable() DS1104_GLOBAL_INTERRUPT_ENABLE()
#define global_disable() DS1104_GLOBAL_INTERRUPT_DISABLE()
```

Refer to DS1104_GLOBAL_INTERRUPT_ENABLE() on page 109 and DS1104_GLOBAL_INTERRUPT_DISABLE() on page 109.

Definitions for debugging via msg-module

```
#define DS1104_DEBUG_INIT 1
#define DS1104_DEBUG_POLL 2
```

Refer to ds1104_init on page 19.

Object declarations

```
extern volatile UInt16 ds1104_debug
```

Include file `Init1104.h`

Initialization

Where to go from here

Information in this section

Information in other sections

ds1104_init

Syntax

```
ds1104_init(void)
```

or

```
init(void)
```

Include file

`Brtenv.h`

Purpose

To initialize all required hard- and software modules for the DS1104.

> **Note**
>
> The initialization function **`ds1104_init`** must be executed at the beginning of each application. It can be invoked only once. Further calls to **`ds1104_init`** are ignored.
> When you are using RTI this function is called automatically in the simulation engine. Hence, in S-functions you don't need to call **`ds1104_init`**. If you need to initialize single components, which are not initialized by **`ds1104_init`** (see below), use the specific initialization functions that are described at the beginning of the function references.

Description

The global variable `ds1104_debug` controls the output of debug messages. Its value is defined by the compiler option –D[definition]:

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

`ds1104_init` carries out the following initialization steps:

1. the global variables for time measurement purposes
2. the memory management functions
3. RTLib1104 is registered in the VCM module
4. the message module for passing error, warning and info messages to the Platform Manager
5. registration of some modules at the VCM module
6. boot firmware is checked. If it is not compatible, the application is terminated and a message is issued
7. the time stamping module in single mode
8. the subinterrupt module to handle interrupts between PowerPC and slave DSPs
9. the host service module for the data transfer from and to the host PC
10. the exit function `ds1104_exit` is hooked in the program termination routine
11. In addition, the I/O components are initialized using the specific initialization functions:
    - The external trigger for the A/D converter is disabled.
    - The input multiplexer of the A/D converter (ADCMUX) is set to channel 1.
    - The D/A converters are set to transparent mode, the external trigger is disabled. The `init()` function resets the D/A converters.
    - The incremental encoder is set to TTL mode with no reset on index. The external trigger is disabled.
    - The I/O pins of the bit I/O unit are set to input.
    - The SYNCIN/SYNCOUT triggers on the rising edge.
    - The slave DSP is set to flash boot. The `init()` function resets the slave DSP.
    - The serial I/O interface (UART) is set to 192000 bit/s, no parity, RS232 mode. The FIFOs are activated with 14-Byte trigger level, autoflow and interrupts are disabled.

This is the basic configuration of the DS1104 R&D Controller Board.

Like mentioned above, `ds1104_init` installs the function `ds1104_exit` with the function `atexit` from the compiler library. Calling the function `exit()` while a program is running activates `ds1104_exit`. This function disables the interrupts and calls the services for ControlDesk. So, you have the possibility to terminate your application (such as I/O settings). After calling `init()`, it is also possible to install a user hook function with `atexit` to start a further termination routine when you call `exit()`. However, an infinite loop is not allowed in the hook function because the `ds1104_exit` routine must terminate the application completely to ensure a predefined state of the board.

Related topics

References

Standard Macros..296

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

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 48). 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();
   }
   ...
}
```

Related topics

Basics

Basic Principles of Time-Stamping.. 48

References

master_cmd_server.. 25
RTLIB_BACKGROUND_SERVICE.. 25

master_cmd_server

Syntax

```
master_cmd_server()
```

Include file

`dscmd.h`

Purpose

To call the master command server.

Description

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 **`host_service`** call. The master command server must be present in each simulation model. Otherwise a relevant error message is issued by the dSPACE experiment software.

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.

Related topics

References

host_service.. 23
RTLIB_BACKGROUND_SERVICE.. 25

RTLIB_BACKGROUND_SERVICE

Syntax

```
RTLIB_BACKGROUND_SERVICE()
```

Include file	`dsstd.h`
Purpose	To call the essential functions in the model background loop.

Description

This macro calls the following functions:

- `host_service`

  The background loop is called `host_service(0,0)`. So, it does not use the time stamp support.
- `master_cmd_server`
- `elog_service`
- `rtlib_background_hook_process`

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 `for` or a `while` construct. To constantly maintain its functionality, it must be called at least once per second.

Related topics

References

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........25

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........26
RTLIB_BACKGROUND_SERVICE........25

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 PowerPC time base, which has a resolution of 4/bus 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

Introduction

There is one specific data type used by the **ds1104_tic_count**, **ds1104_tic_elapsed**, **ds1104_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 UInt32 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:

```
ds1104_tic_start();   /* starts time measurement */
...
time = ds1104_tic_read();
... action 1 ...
ds1104_tic_halt(); /* start of the break */
... action 2 ...
ds1104_tic_continue(); /* end of the break */
... action 3 ...
time = ds1104_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 */
```

ds1104_tic_continue

Syntax

```
ds1104_tic_continue()
```

Include file

`tic1104.h`

Purpose

To resume time measurement after it was paused by **`ds1104_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 51.

Return value

None

Related topics

Examples

Example of Using Time Measurement Functions........31

References

ds1104_tic_halt........36
RTLIB_TIC_CONTINUE........40

ds1104_tic_count

Syntax

```
rtlib_tic_t ds1104_tic_count(void)
```

Include file

`tic1104.h`

Purpose

To read the current counter value of the time base.

Description

Use **`ds1104_tic_count`** in conjunction with **`ds1104_tic_elapsed`** or **`ds1104_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 = ds1104_tic_count();
   …
   timer_count2 = ds1104_tic_count();
   exec_time = ds1104_tic_diff(timer_count1, timer_count2);
   …
}
```

Related topics

References

ds1104_tic_delay

Syntax

```
ds1104_tic_delay(Float64 duration)
```

Include file

`tic1104.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

ds1104_tic_diff

Syntax

```
dsfloat ds1104_tic_diff(
     rtlib_tic_t tmr_cnt1,
     rtlib_tic_t tmr_cnt2)
```

Include file

`tic1104.h`

Purpose

To calculate the difference between two time base counter values.

Description

Use **`ds1104_tic_diff`** in conjunction with **`ds1104_tic_count`** or **`ds1104_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 = ds1104_tic_count();
   …
   timer_count2 = ds1104_tic_count();
   exec_time = ds1104_tic_diff(timer_count1, timer_count2);
   …
}
```

Related topics

References

ds1104_tic_count 33
ds1104_tic_elapsed 35

ds1104_tic_elapsed

Syntax

```
dsfloat ds1104_tic_elapsed(rtlib_tic_t tmr_cnt)
```

Include file

`tic1104.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.

Description

Use **`ds1104_tic_elapsed`** in conjunction with **`ds1104_tic_count`** or **`ds1104_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 = ds1104_tic_count();
   …
   exec_time = ds1104_tic_elapsed(timer_count);
   …
}
```

Related topics

References

ds1104_tic_halt

Syntax

```
ds1104_tic_halt()
```

Include file

`tic1104.h`

Purpose

To pause time measurement.

Description

The break lasts until measurement is resumed by `ds1104_tic_continue`.

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to Time-Stamping Functions on page 51.

Return value

None

Related topics

Examples

Example of Using Time Measurement Functions..31

References

ds1104_tic_continue..32
RTLIB_TIC_HALT..44

ds1104_tic_read

Syntax

```
Float64 ds1104_tic_read()
```

Include file

`tic1104.h`

Purpose

To read the time period since time measurement was started by **`ds1104_tic_start`**, minus the breaks made from **`ds1104_tic_halt`** to **`ds1104_tic_continue`**.

Description

Use **`ds1104_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 51.

Return value

This function returns the time duration in seconds.

Related topics

Examples

Example of Using Time Measurement Functions..31

References

ds1104_tic_continue..32
ds1104_tic_halt..36
ds1104_tic_start..38
ds1104_tic_total_read..38
RTLIB_TIC_READ..45

ds1104_tic_start

Syntax	`ds1104_tic_start()`
Include file	`tic1104.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 51.
Return value	None
Related topics	Examples Example of Using Time Measurement Functions .... 31 References RTLIB_TIC_START 47

ds1104_tic_total_read

Syntax	`Float64 ds1104_tic_total_read()`
Include file	`tic1104.h`
Purpose	To read the complete time period since the time measurement was started by **`ds1104_tic_start`**, including all breaks made from **`ds1104_tic_halt`** to **`ds1104_tic_continue`**.
Description	Use **`ds1104_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 51.

Return value	This function returns the time duration in seconds.
Related topics	References

ds1104_timebase_low_read

Syntax	`UInt32 ds1104_timebase_low_read(void)`
Include file	`tmr1104.h`
Purpose	To read the Lower Timebase Register (TBRL).
Description	Use **`ds1104_timebase_read`** to read both registers TBRL and TBRU.
Return value	This function returns the current value of the TBRL.
Related topics	References

ds1104_timebase_read

Syntax	`Int64 ds1104_timebase_read(void)`
Include file	`tmr1104.h`

Purpose	To read the Lower and Upper Timebase Registers (TBRL and TBRU).
Description	Since the PowerPC cannot work with 64-bit integer values, a structure Int64, which consists of an Int32 (high word) and an UInt32 (low word), is defined (refer to Elementary Data Types on page 17). Use `ds1104_timebase_low_read` to read the TBRL only.
Return value	This function returns the current value of TBRL and TBRU.
Related topics	References ds1104_timebase_low_read.......... 39

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 51.
Return value	None
Related topics	Examples Example of Using Time Measurement Functions.......... 31 References ds1104_tic_continue.......... 32 RTLIB_TIC_HALT.......... 44

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

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

ds1104_tic_delay 34
RTLIB_TIC_CONTINUE 40
RTLIB_TIC_START 47

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

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

ds1104_tic_elapsed 35

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 51.

Return value

None

Related topics

Examples

References

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 51.

Return value

This function returns the time duration in seconds.

Related topics

Examples

References

RTLIB_TIC_READ_TOTAL

Syntax

```
RTLIB_TIC_READ_TOTAL()
```

Include file

`dsstd.h`

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

Use **`RTLIB_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 51.

Return value

This function returns the time duration in seconds.

Related topics

References

RTLIB_TIC_START

Syntax	`RTLIB_TIC_START()`
Include file	`dsstd.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 51.
Return value	None
Related topics	Examples Example of Using Time Measurement Functions....................... 31 References ds1104_tic_start....................... 38

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

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

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 bus clock of the board.

Time stamp range The time base has a range of 64 bit. Combined with a resolution of 40 ns, this is enough to measure highly accurate absolute times up to several years.

Principles of an Absolute Time in Single-Processor 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.

Microticks and macroticks

In multiprocessor systems 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 there is no synchronization 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.

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 time base, which is driven by the bus clock of the DS1104, scaled by 4. For example, at a DS1104 with 100 MHz bus clock, the resolution of the Microtick Counter is 40 ns. In the dSPACE experiment software, you can find information on the bus clock in the Properties dialog of the DS1104. In the dialog, select the DS1104 Properties page.

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

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

Note

Multi master mode and multi slave mode are not available on single-processor systems, even if you installed more than one single-processor board in your system.

Implementation of an Absolute Time in Single-Processor Systems

Timer characteristics

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 (BCLK means bus clock):

Board	Timer Source	Frequency	Resolution	Condition
DS1104	Time Base Counter	f_{BCLK} / 4	40 ns	f_{BCLK}=100 MHz

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

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

Data types

The following data types are defined for time-stamping:

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

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;	Mode of the time-stamping software module. The following symbols are predefined: ▪ `TS_MODE_SINGLE` Used for single-processor systems ▪ `TS_MODE_MULTI_MASTER` Used for the master board in a multiprocessor system ▪ `TS_MODE_MULTI_SLAVE` Used for slave boards in a multiprocessor system For further information, refer to Modes of the Time-Stamping module on page 49.
dsts_mit_per_mat	`UInt32 dsts_mit_per_mat;`	Nominal number of microticks per macrotick (synchronization tick at a timing master in a multiprocessor system).

Note

Multi master mode and multi slave mode are not available on single-processor systems, even if you installed more than one single-processor board in your system.

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

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.

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`.

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

Note

Multi master mode and multi slave mode are not available on single-processor systems, even if you installed more than one single-processor board in your system.

mat_period Specifies the time in seconds of one macrotick 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.......... 48

References

ts_reset.......... 53

ts_reset

Syntax

```
void ts_reset()
```

Include file	`dsts.h`
Purpose	To reset the Time-Stamping module to the absolute time 0.
Return value	None
Related topics	Basics

References

ts_time_read

Syntax	`double ts_time_read()`
Include file	`dsts.h`
Purpose	To read the absolute time in seconds.
Return value	This function returns the absolute time in seconds since the initialization `ts_init` or the last reset `ts_reset`.
Related topics	Basics

References

ts_timestamp_read

Syntax

```
void ts_timestamp_read(ts_timestamp_ptr_type ts)
```

Include file

`dsts.h`

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 `ts` points to.

Parameters

ts Specifies the pointer to a time stamp structure for the read value.

Return value

None

Related topics

Basics

Basic Principles of Time-Stamping.. 48

References

ts_time_read.. 54

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........ 48

References

ts_timestamp_interval........ 56

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.. 48

References

ts_timestamp_compare.. 55

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.. 48

References

ts_timestamp_offset... 58

ts_timestamp_offset

Syntax

```
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)
```

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

ts_reference Specifies the pointer to the time stamp structure holding the reference time.

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 holding the absolute time in seconds.

Return value

None

Related topics

Basics

Basic Principles of Time-Stamping........ 48

References

ts_time_offset........ 57

ts_time_calculate

Syntax

```
double ts_time_calculate(ts_timestamp_ptr_type ts)
```

Include file

`dsts.h`

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........ 48

References

ts_timestamp_offset........ 58

ts_timestamp_calculate

Syntax

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

Include file	`dsts.h`
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

References

Timer 0

Introduction

Timer 0 is a down counter generating an interrupt whenever it reaches zero. Then the period value is reloaded automatically. Timer 0 is also used by the standard macros as default sample rate timer, see Standard Macros on page 296.

Where to go from here

Information in this section

Information in other sections

Example of Using Timer 0 Functions

Example source code

The following example demonstrates how to use Timer 0 functions to generate periodic events with the interrupt routine `isr_timer0`. You can start and stop the Timer, and specify the timer period. This example can also be used with the functions of Timer 1, Timer 2 and Timer 3. You can find the relevant files in the directory `<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\Tmr0_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <Brtenv.h>
#define DT 100e-3          /* 100 ms simulation step size */
/* variables for execution time profiling */
Float64 exec_time;         /* execution time */
Float64 timer0;            /* timer0 read value */
```

```
/* adjust values for timer0 */
Float64 timer0_period = DT;         /* period in sec */
Float64 timer0_actual_period = DT;  /* actual period in sec */
/* adjust timer0 period */
Int32 timer0_period_set = 0;
Int32 timer0_period_set_lock = 0;
/* start/stop timer0 */
Int32 timer0_mode = 1;
Int32 timer0_mode_old = 1;
/* counter for Interrupt service routines */
Int32 timer0_counter = 0;
void isr_timer0(void)
{
  ts_timestamp_type ts;
  /* overrun check, enable interrupts globally */
  ds1104_begin_isr_timer0();
  RTLIB_TIC_START();
  timer0_counter++;       /* counter for timer0 interrupts */
  ts_timestamp_read(&ts);
  host_service(1, &ts);     /* data acquisition service */
  exec_time = RTLIB_TIC_READ();
  /* overrun check end, disable interrupts globally */
  ds1104_end_isr_timer0();
}
void main(void)
{
  /* DS1104 and RTLib1104 initialization */
  init();
  msg_info_set(MSG_SM_RTLIB, 0, "System started.");
  /* periodic event with timer0 */
  ds1104_start_isr_timer0(timer0_period, isr_timer0);
  /* Background tasks */
  while(1)
  {
    RTLIB_BACKGROUND_SERVICE();      /* background service */
    /* read timer0 count and convert it to a time */
    ds1104_timer0_read(&timer0);
    /* start/stop timer0 */
    if (timer0_mode != timer0_mode_old)
    {
      if (timer0_mode)
        ds1104_timer0_start();
      else
        ds1104_timer0_stop();
      timer0_mode_old = timer0_mode;
    }
    /* adjust timer0 period */
    if (timer0_period_set && !timer0_period_set_lock)
    {
      ds1104_timer0_period_set(timer0_period);
      timer0_actual_period = timer0_period;
      timer0_period_set_lock = 1;
    }
    else if (!timer0_period_set && timer0_period_set_lock)
    {
      timer0_period_set_lock = 0;
    }
  }
}
```

Related topics

Basics

Timer Features (DS1104 Features 📖)

References

Timer 0.. 61

ds1104_timer0_period_set

Syntax

```
void ds1104_timer0_period_set(Float64 time)
```

Include file

`tmr1104.h`

Purpose

To define the period of Timer 0.

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 is down to zero.

Parameters

time	period in seconds

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions.. 61

References

Elementary Data Types.. 17

ds1104_timer0_read

Syntax

```
void ds1104_timer0_read(Float64 *time)
```

Include file

`tmr1104.h`

Purpose

To read the current value of Timer 0.

Parameters

time address where the current value of Timer 0 is written. The value is given in seconds.

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions.. 61

References

Elementary Data Types... 17

ds1104_timer0_start

Syntax

```
void ds1104_timer0_start(void)
```

Include file

`tmr1104.h`

Purpose

To start Timer 0.

Description

If no period is set, the counter starts with the highest counter value (0xFFFF FFFF). Use `ds1104_timer0_period_set` to set the period.

Return value	None
Related topics	References

ds1104_timer0_period_set 63

ds1104_timer0_stop

Syntax	`void ds1104_timer0_stop(void)`
Include file	`tmr1104.h`
Purpose	To stop Timer 0.
Description	Use `ds1104_timer0_start` to resume from the current value.
Return value	None
Related topics	References

ds1104_timer0_start 64

Timer 1

Introduction

Timer 1 is a down counter generating an interrupt whenever it reaches zero. Then the period value is reloaded automatically.

Where to go from here

Information in this section

Information in other sections

ds1104_timer1_period_set

Syntax

```
void ds1104_timer1_period_set(Float64 time)
```

Include file

`tmr1104.h`

Purpose

To define the period of Timer 1.

Result

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 is down to zero.

Parameters

time period in seconds

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions.......... 61

References

Elementary Data Types.......... 17

ds1104_timer1_read

Syntax

```
void ds1104_timer1_read(Float64 *time)
```

Include file

`tmr1104.h`

Purpose

To read the current value of Timer 1.

Parameters

time address where the current value of Timer 1 is written. The value is given in seconds.

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions.......... 61

References

Elementary Data Types.......... 17

ds1104_timer1_start

Syntax	`void ds1104_timer1_start(void)`
Include file	`tmr1104.h`
Purpose	To start Timer 1.
Description	If no period is set, the counter starts with the highest counter value (0xFFFF FFFF). Use `ds1104_timer1_period_set` to set the period.
Return value	None
Related topics	References ds1104_timer1_period_set 66

ds1104_timer1_stop

Syntax	`void ds1104_timer1_stop(void)`
Include file	`tmr1104.h`
Purpose	To stop Timer 1.
Description	Use `ds1104_timer1_start` to resume from the current value.

Return value	None
Related topics	References ds1104_timer1_start ... 68

Timer 2

Introduction

Timer 2 is a down counter generating an interrupt whenever it reaches zero. Then the period value is reloaded automatically.

Where to go from here

Information in this section

Information in other sections

ds1104_timer2_period_set

Syntax

```
void ds1104_timer2_period_set(Float64 time)
```

Include file

`tmr1104.h`

Purpose

To define the period of Timer 2.

Result

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 is down to zero.

Parameters

time period in seconds

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions.. 61

References

Elementary Data Types.. 17

ds1104_timer2_read

Syntax

```
void ds1104_timer2_read(Float64 *time)
```

Include file

`tmr1104.h`

Purpose

To read the current value of Timer 2.

Parameters

time address where the current value of Timer 2 is written. The value is given in seconds.

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions.. 61

References

Elementary Data Types.. 17

ds1104_timer2_start

Syntax	`void ds1104_timer2_start(void)`
Include file	`tmr1104.h`
Purpose	To start Timer 2.
Description	If no period is set, the counter starts with the highest counter value (0xFFFF FFFF). Use **`ds1104_timer2_period_set`** to set the period.
Return value	None
Related topics	References ds1104_timer2_period_set 70

ds1104_timer2_stop

Syntax	`void ds1104_timer2_stop(void)`
Include file	`tmr1104.h`
Purpose	To stop Timer 2.
Description	Use **`ds1104_timer2_start`** to resume from the current value.

Return value None

Related topics

References

Timer 3

Introduction

Timer 3 is a down counter generating an interrupt whenever it reaches zero. Then the period value is reloaded automatically.

Where to go from here

Information in this section

Information in other sections

ds1104_timer3_period_set

Syntax

```
void ds1104_timer3_period_set(Float64 time)
```

Include file

`tmr1104.h`

Purpose

To define the period of Timer 3.

Result

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 is down to zero.

Parameters

time period in seconds

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions..61

References

Elementary Data Types..17

ds1104_timer3_read

Syntax

```
void ds1104_timer3_read(Float64 *time)
```

Include file

`tmr1104.h`

Purpose

To read the current value of Timer 3.

Parameters

time address where the current value of Timer 3 is written. The value is given in seconds.

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions..61

References

Elementary Data Types..17

ds1104_timer3_start

Syntax	`void ds1104_timer3_start(void)`
Include file	`tmr1104.h`
Purpose	To start Timer 3.
Description	If no period is set, the counter starts with the highest counter value (0xFFFF FFFF). Use **`ds1104_timer3_period_set`** to set the period.
Return value	None
Related topics	References ds1104_timer3_period_set........ 74

ds1104_timer3_stop

Syntax	`void ds1104_timer3_stop(void)`
Include file	`tmr1104.h`
Purpose	To stop Timer 3.
Description	Use **`ds1104_timer3_start`** to resume from the current value.

Return value

None

Related topics

References

Decrementer

Introduction

The Decrementer is the PowerPC built-in Decrementer.

Where to go from here

Information in this section

Information in other sections

ds1104_decrementer_set

Syntax

```
void ds1104_decrementer_set(UInt32 decrementer_value)
```

Include file

`tmr1104.h`

Purpose

To set the counter value of the free running Decrementer.

When the Decrementer is down to 0, an interrupt occurs and the Decrementer is reloaded by software with the value specified with `ds1104_decrementer_period_set`.

Parameters

decrementer_value Specifies the counter value.

Return value

None

Related topics

References

ds1104_decrementer_period_set 79
Elementary Data Types 17

ds1104_decrementer_period_set

Syntax

```
void ds1104_decrementer_period_set(Float64 time)
```

Include file

`tmr1104.`

Purpose

To convert the period given in seconds to a counter value and set the counter value.

Result

When the decrementer is down to 0, an interrupt occurs and the Decrementer is reloaded by software with the value specified with `ds1104_decrementer_period_set`.

Parameters

time	period in seconds

Return value

None

Related topics

References

ds1104_decrementer_set 78
Elementary Data Types 17

ds1104_decrementer_read

Syntax

```
void ds1104_decrementer_read(Float64 *time)
```

Include file

`tmr1104.h`

Purpose	To read the current decrementer value and convert it to seconds.
Parameters	**time** Specifies the address where the value is written. The time is given in seconds.
Return value	None
Related topics	References

Timer Interrupt Control

Purpose

Use these functions to install interrupt service functions – for Timer 0, Timer 1, Timer 2, Timer 3, and the Decrementer – and to perform overrun checks for the defined interrupt service routines.

Where to go from here

Information in this section

Information in other sections

ds1104_start_isr_timer0

Syntax

```
ds1104_start_isr_timer0(
      Float64 sampling_period,
      isr_function_name)
```

or

```
RTLIB_SRT_START(
      Float64 sampling_period,
      isr_function_name)
```

Include file

`int1104.h`

Purpose

To install the `isr_function_name` as an interrupt service routine for Timer 0.

Description

The function sets the period of the Timer 0, installs the interrupt service routine in the interrupt vector and starts the Timer 0.

Result

If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use `ds1104_begin_isr_timer0` and `ds1104_end_isr_timer0` 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 0 interrupt. This function must not have an input parameter or a return value, for example, `void isr_function_name(void)`.

Return value

None

Example

This example installs the function `timer0_interrupt` that is called when the Timer 0 interrupt occurs, namely, every 20 µs:

```
ds1104_start_isr_timer0(20e-6, timer0_interrupt)
```

Related topics

Examples

References

ds1104_start_isr_timer1

Syntax

```
ds1104_start_isr_timer1(
     Float64 sampling_period,
     isr_function_name)
```

Include file

`int1104.h`

Purpose

To install the `isr_function_name` as an interrupt service routine for Timer 1.

Description

The function sets the period of the Timer 1, installs the interrupt service routine in the interrupt vector and starts the Timer 1.

Result

If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use `ds1104_begin_isr_timer1` and `ds1104_end_isr_timer1` 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 1 interrupt. This function must not have an input parameter or a return value, for example, `void isr_function_name(void)`.

Return value

None

Example

This example installs the function `timer1_interrupt` that is called when the Timer 1 interrupt occurs, namely, every 20 µs:

```
ds1104_start_isr_timer1(20e-6, timer1_interrupt)
```

Related topics

Examples

References

ds1104_start_isr_timer2

Syntax

```
ds1104_start_isr_timer2(
      Float64 sampling_period,
      isr_function_name)
```

Include file

`int1104.h`

Purpose

To install the `isr_function_name` as an interrupt service routine for Timer 2.

Description

The function sets the period of the Timer 2, installs the interrupt service routine in the interrupt vector and starts the Timer 2.

Result

If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use `ds1104_begin_isr_timer2` and `ds1104_end_isr_timer2` 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 2 interrupt. This function must not have an input parameter or a return value, for example, `void isr_function_name(void)`.

Return value

None

Example

This example installs the function `timer2_interrupt` that is called when the Timer 2 interrupt occurs, namely, every 20 µs:

```
ds1104_start_isr_timer2(20e-6, timer2_interrupt)
```

Related topics

Examples

Example of Using Timer 0 Functions........ 61

References

ds1104_begin_isr_timer2........ 91
ds1104_end_isr_timer2........ 92
Elementary Data Types........ 17

ds1104_start_isr_timer3

Syntax

```
ds1104_start_isr_timer3(
      Float64 sampling_period,
      isr_function_name)
```

Include file

`int1104.h`

Purpose

To install the `isr_function_name` as an interrupt service routine for Timer 3.

Description

The function sets the period of the Timer 3, installs the interrupt service routine in the interrupt vector and starts the Timer 3.

Result

If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use `ds1104_begin_isr_timer3` and `ds1104_end_isr_timer3` 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 3 interrupt. This function must not have an input parameter or a return value, for example, `void isr_function_name(void)`.

Return value

None

Example

This example installs the function `timer3_interrupt` that is called when the Timer 3 interrupt occurs, namely, every 20 µs:

```
ds1104_start_isr_timer3(20e-6, timer3_interrupt)
```

Related topics

Examples

Example of Using Timer 0 Functions..61

References

ds1104_begin_isr_timer3..93
ds1104_end_isr_timer3..94
Elementary Data Types..17

ds1104_start_isr_decrementer

Syntax

```
ds1104_start_isr_decrementer(
      Float64 sampling_period,
      isr_function_name)
```

Include file

`int1104.h`

Purpose

To install `isr_function_name` as an interrupt service routine for the Decrementer.

Description

The function sets the period of the Decrementer, installs the interrupt service routine in the interrupt vector, and starts the Decrementer.

Result

If the execution time of the interrupt service routine exceeds the interrupt period an overrun occurs. Use `ds1104_begin_isr_decrementer` and `ds1104_end_isr_decrementer` 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 Decrementer interrupt. This function must not have an input parameter or a return value, for example, `void isr_function_name(void)`.

Return value

None

Example

This example installs the function `decr_interrupt` that is called when the Decrementer interrupt occurs, namely, every 20 µs:

```
ds1104_start_isr_decrementer(20e-6, decr_interrupt)
```

Related topics

References

ds1104_begin_isr_decrementer........94
ds1104_end_isr_decrementer........95
Elementary Data Types........17

ds1104_begin_isr_timer0

Syntax

```
ds1104_begin_isr_timer0()
```

or

```
RTLIB_SRT_ISR_BEGIN()
```

Include file

`int1104.h`

Purpose

To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_timer0`.

Result

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 timer0_interrupt(void)
{
   ds1104_begin_isr_timer0();
   /* interrupt service routine */
   ds1104_end_isr_timer0();
}
```

Following, there is the same example written with the standard macros.

```
void timer0_interrupt(void)
{
   RTLIB_SRT_ISR_BEGIN();
   RTLIB_SRT_ISR_END();
}
```

Related topics

Examples

Example of Using Timer 0 Functions........ 61

References

ds1104_end_isr_timer0........ 89
Standard Macros........ 296

ds1104_end_isr_timer0

Syntax

```
ds1104_end_isr_timer0()
```

or

```
RTLIB_SRT_ISR_END()
```

Include file

`int1104.h`

Purpose

To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_timer0`.

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions........ 61

References

ds1104_begin_isr_timer0........ 88
ds1104_start_isr_timer0........ 83
Standard Macros........ 296

ds1104_begin_isr_timer1

Syntax

```
ds1104_begin_isr_timer1()
```

Include file

`int1104.h`

Purpose

To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_timer1`.

Result

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 timer1_interrupt(void)
{
   ds1104_begin_isr_timer1();
   /* interrupt service routine */
   ds1104_end_isr_timer1();
}
```

Related topics

Examples

References

ds1104_end_isr_timer1

Syntax

```
ds1104_end_isr_timer1()
```

Include file

`int1104.h`

Purpose

To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_timer1`.

Return value

None

Related topics

Examples

References

ds1104_begin_isr_timer2

Syntax

```
ds1104_begin_isr_timer2()
```

Include file

`int1104.h`

Purpose

To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_timer2`.

Result

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 timer2_interrupt(void)
{
   ds1104_begin_isr_timer2();
   /* interrupt service routine */
   ds1104_end_isr_timer2();
}
```

Related topics

Examples

Example of Using Timer 0 Functions.. 61

References

ds1104_end_isr_timer2.. 92

ds1104_end_isr_timer2

Syntax

```
ds1104_end_isr_timer2()
```

Include file

`int1104.h`

Purpose

To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_timer2`.

Return value

None

Related topics

Examples

References

ds1104_begin_isr_timer3

Syntax

```
ds1104_begin_isr_timer3()
```

Include file

`int1104.h`

Purpose

To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_timer3`.

Result

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 timer3_interrupt(void)
{
   ds1104_begin_isr_timer3();
   /* interrupt service routine */
   ds1104_end_isr_timer3();
}
```

Related topics

Examples

Example of Using Timer 0 Functions.. 61

References

ds1104_end_isr_timer3.. 94

ds1104_end_isr_timer3

Syntax

```
ds1104_end_isr_timer3()
```

Include file

`int1104.h`

Purpose

To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_timer3`.

Return value

None

Related topics

Examples

Example of Using Timer 0 Functions.. 61

References

ds1104_begin_isr_timer3.. 93
ds1104_start_isr_timer3.. 86

ds1104_begin_isr_decrementer

Syntax

```
ds1104_begin_isr_decrementer()
```

Include file

`int1104.h`

Purpose	To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_decrementer`.
Result	When the execution time of the interrupt service routine exceeds the interrupt period (overrun), the interrupt is stopped, and an error message is generated.
Return value	None

Example

This example shows an interrupt service routine with overrun check:

```
void decr_interrupt(void){
   ds1104_begin_isr_decrementer();
   /* interrupt service routine */
   ds1104_end_isr_decrementer();}
```

Related topics

References

ds1104_end_isr_decrementer....95

ds1104_end_isr_decrementer

Syntax

```
ds1104_end_isr_decrementer()
```

Include file	`int1104.h`
Purpose	To check for an overrun in the interrupt service routine assigned by `ds1104_start_isr_decrementer`.
Return value	None

Related topics

References

ds1104_begin_isr_decrementer....94

Interrupt Handling

Purpose

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.

Note

The installing of interrupt service routines for the Timer 0, Timer 1, Timer 2, Timer 3 and Decrementer interrupts is exceptional. Refer to the example in ds1104_set_interrupt_vector on page 97 and to Timer Interrupt Control on page 81.

Interrupt service routine type

The interrupt service routine type is defined as follows:

```
typedef void (*DS1104_Int_Handler_Type)(void)
```

Where to go from here

Information in this section

ds1104_set_interrupt_vector

Syntax

```
DS1104_Int_Handler_Type ds1104_set_interrupt_vector(
      UInt32 IntID,
      DS1104_Int_Handler_Type Handler,
      Int SaveRegs)
```

Include file

`int1104.h`

Purpose

To install an interrupt service routine for the selected interrupt.

Description

Use `DS1104_GLOBAL_INTERRUPT_ENABLE()` to enable interrupts.

Note

- Set the parameter **SaveRegs** on page 99 to `SAVE_REGS_ON` to save and restore the registers. This is absolutely essential for C-coded interrupt service routines. `SAVE_REGS_OFF` is only allowed for assembler-coded interrupt service routines, which store the used registers themselves.
- Do not choose `SaveRegs = SAVE_REGS_OFF` if you want to globally enable the interrupts in the interrupt service routine. Without saving the registers the interrupt service routine may be called before the hardware has acknowledged a triggered interrupt. This would result in another trigger interrupt of the same interrupt.
- The installation of interrupt service routines for the Timer 0, Timer 1, Timer 2, Timer 3, and Decrementer interrupts is exceptional. Refer to the example below and to Timer Interrupt Control on page 81.

Parameters

IntID Specifies the interrupt that the handler is to be installed for.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INT_DECREMENTER	Decrementer interrupt
DS1104_INT_TIMER_0	Timer 0 interrupt
DS1104_INT_TIMER_1	Timer 1 interrupt
DS1104_INT_TIMER_2	Timer 2 interrupt
DS1104_INT_TIMER_3	Timer 3 interrupt
DS1104_INT_EXTERNAL_0	External interrupt 0
DS1104_INT_EXTERNAL_1	External interrupt 1
DS1104_INT_EXTERNAL_2	External interrupt 2
DS1104_INT_EXTERNAL_3	External interrupt 3
DS1104_INT_HOST	Host interrupt
DS1104_INT_SLAVE_DSP	Slave DSP interrupt
DS1104_INT_SLAVE_DSP_PWM	Slave DSP interrupt PWM generation
DS1104_INT_SERIAL_UART	Serial UART interrupt
DS1104_INT_INC_ENC_CH1	Encoder index channel 1
DS1104_INT_INC_ENC_CH2	Encoder index channel 2
DS1104_INT_ADC_CONVERSION_1	ADC 1 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_2	ADC 2 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_3	ADC 3 end-of-conversion interrupt

Predefined Symbol	Meaning
DS1104_INT_ADC_CONVERSION_4	ADC 4 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_5	ADC 5 end-of-conversion interrupt

Note

- The level triggered interrupt (serial UART interrupt) has to be acknowledged in the interrupt service routine before the interrupt is enabled globally again.
- The interrupt module must prohibit, that a digital I/O pin (IO16 … 19), which is programmed as output, is simultaneously used as external interrupt input (INT1 … 4).

Handler Specifies the address of the interrupt service routine.

SaveRegs Saves 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.

Note

If you do not save the relevant register (`SaveRegs = SAVE_REGS_OFF`), you have to save and restore the registers in your program code by yourself. However, the registers r3, r4, r5, cr, ctr, and xer will be saved automatically even if SaveRegs is set to SAVE_REGS_OFF.

Return value

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

Example

The Timer 0 interrupt is supposed to call the function `timer0_interrupt` (see also `ds1104_start_isr_timer0`).

First write the function `timer0_interrupt`:

```
void timer0_interrupt(void)
{
...
}
```

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

```
ds1104_set_interrupt_vector(DS1104_INT_TIMER_0, (DS1104_Int_Handler_Type) timer0_interrupt, SAVE_REGS_ON);
```

Related topics

References

ds1104_get_interrupt_vector

Syntax

```
DS1104_Int_Handler_Type ds1104_get_interrupt_vector(UInt32 IntID)
```

Include file

`int1104.h`

Purpose

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

Description

Use this function to check if the handler is really installed.

Parameters

IntID Specifies the interrupt for which the address is to be read.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INT_DECREMENTER	Decrementer interrupt
DS1104_INT_TIMER_0	Timer 0 interrupt
DS1104_INT_TIMER_1	Timer 1 interrupt
DS1104_INT_TIMER_2	Timer 2 interrupt
DS1104_INT_TIMER_3	Timer 3 interrupt
DS1104_INT_EXTERNAL_0	External interrupt 0
DS1104_INT_EXTERNAL_1	External interrupt 1
DS1104_INT_EXTERNAL_2	External interrupt 2
DS1104_INT_EXTERNAL_3	External interrupt 3
DS1104_INT_HOST	Host interrupt
DS1104_INT_SLAVE_DSP	Slave DSP interrupt
DS1104_INT_SLAVE_DSP_PWM	Slave DSP interrupt PWM generation
DS1104_INT_SERIAL_UART	Serial UART interrupt
DS1104_INT_INC_ENC_CH1	Encoder index channel 1
DS1104_INT_INC_ENC_CH2	Encoder index channel 2

Predefined Symbol	Meaning
DS1104_INT_ADC_CONVERSION_1	ADC 1 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_2	ADC 2 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_3	ADC 3 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_4	ADC 4 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_5	ADC 5 end-of-conversion interrupt

Note

The level triggered interrupt (serial UART interrupt) has to be acknowledged in the interrupt service routine before the interrupt is enabled globally again.

Return value

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

Related topics

References

ds1104_get_interrupt_status

Syntax

```
UInt32 ds1104_get_interrupt_status(void)
```

Include file

`int1104.h`

Purpose

To get the interrupt status.

Description

This function indicates the status of the EE bit (External Interrupt Enable) of the Machine Status Register (MSR). Use this function if you want to disable interrupts during function execution and restore the value of the EE bit afterwards (see the example in ds1104_set_interrupt_status on page 102).

Return value

This function returns the value of the EE bit

Value	Meaning
0x0000	EE bit = 0; external interrupt disabled
0x8000	EE bit = 1; external interrupt enabled

Related topics

References

ds1104_set_interrupt_status 102
Elementary Data Types 17

ds1104_set_interrupt_status

Syntax

```
void ds1104_set_interrupt_status(UInt32 status)
```

Include file

`int1104.h`

Purpose

To set the interrupt status.

Description

The value of the EE bit of the Machine Status Register (MSR) is restored.

Parameters

status Returns the value of the previously executed function `ds1104_get_interrupt_status`.

Return value

None

Example

This example shows how to save and restore the value of the EE bit:

```
void restore(void)
{
UInt32 msr_state;
msr_state = ds1104_get_interrupt_status();
/* Saves the value of the EE bit in MSR */
   RTLIB_INT_DISABLE();   /* Disables interrupts */
```

```
...
ds1104_set_interrupt_status(msr_state);
/* Restores the EE bit in MSR at the end of the function*/
}
```

Related topics

References

ds1104_enable_hardware_int

Syntax

```
void ds1104_enable_hardware_int(UInt32 IntID)
```

or

```
RTLIB_SRT_ENABLE()
```

Include file

`int1104.h`

Purpose

To enable the specified hardware interrupt.

Description

This function only clears the corresponding mask bit. However, the specified hardware interrupt is available only when the interrupts are globally enabled (see `DS1104_GLOBAL_INTERRUPT_ENABLE()`).

Parameters

IntID Specifies the interrupt that is to be enabled.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INT_DECREMENTER	Decrementer interrupt
DS1104_INT_TIMER_0	Timer 0 interrupt
DS1104_INT_TIMER_1	Timer 1 interrupt
DS1104_INT_TIMER_2	Timer 2 interrupt
DS1104_INT_TIMER_3	Timer 3 interrupt
DS1104_INT_EXTERNAL_0	External interrupt 0
DS1104_INT_EXTERNAL_1	External interrupt 1
DS1104_INT_EXTERNAL_2	External interrupt 2
DS1104_INT_EXTERNAL_3	External interrupt 3

Predefined Symbol	Meaning
DS1104_INT_HOST	Host interrupt
DS1104_INT_SLAVE_DSP	Slave DSP interrupt
DS1104_INT_SLAVE_DSP_PWM	Slave DSP interrupt PWM generation
DS1104_INT_SERIAL_UART	Serial UART interrupt
DS1104_INT_INC_ENC_CH1	Encoder index channel 1
DS1104_INT_INC_ENC_CH2	Encoder index channel 2
DS1104_INT_ADC_CONVERSION_1	ADC 1 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_2	ADC 2 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_3	ADC 3 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_4	ADC 4 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_5	ADC 5 end-of-conversion interrupt

Note

The level triggered interrupt (serial UART interrupt) has to be acknowledged in the interrupt service routine before the interrupt is enabled globally again.

Return value

None

Related topics

References

ds1104_disable_hardware_int 104
DS1104_GLOBAL_INTERRUPT_DISABLE() 109
DS1104_GLOBAL_INTERRUPT_ENABLE() 109
Elementary Data Types 17
Standard Macros 296

ds1104_disable_hardware_int

Syntax

```
void ds1104_disable_hardware_int(UInt32 IntID)
```

or

```
RTLIB_SRT_DISABLE()
```

Include file

`int1104.h`

Purpose

To disable the specified hardware interrupt when the interrupts are still globally enabled (see `DS1104_GLOBAL_INTERRUPT_ENABLE()`).

Description

This function sets the corresponding mask bit.

Parameters

IntID Specifies the interrupt that is to be disabled.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INT_DECREMENTER	Decrementer interrupt
DS1104_INT_TIMER_0	Timer 0 interrupt
DS1104_INT_TIMER_1	Timer 1 interrupt
DS1104_INT_TIMER_2	Timer 2 interrupt
DS1104_INT_TIMER_3	Timer 3 interrupt
DS1104_INT_EXTERNAL_0	External interrupt 0
DS1104_INT_EXTERNAL_1	External interrupt 1
DS1104_INT_EXTERNAL_2	External interrupt 2
DS1104_INT_EXTERNAL_3	External interrupt 3
DS1104_INT_HOST	Host interrupt
DS1104_INT_SLAVE_DSP	Slave DSP interrupt
DS1104_INT_SLAVE_DSP_PWM	Slave DSP interrupt PWM generation
DS1104_INT_SERIAL_UART	Serial UART interrupt
DS1104_INT_INC_ENC_CH1	Encoder index channel 1
DS1104_INT_INC_ENC_CH2	Encoder index channel 2
DS1104_INT_ADC_CONVERSION_1	ADC 1 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_2	ADC 2 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_3	ADC 3 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_4	ADC 4 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_5	ADC 5 end-of-conversion interrupt

Note

The level triggered interrupt (serial UART interrupt) has to be acknowledged in the interrupt service routine before the interrupt is enabled globally again.

Return value

None

Related topics

References

ds1104_get_interrupt_flag

Syntax

```
int ds1104_get_interrupt_flag(UInt32 IntID)
```

Include file

`int1104.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
DS1104_INT_DECREMENTER	Decrementer interrupt
DS1104_INT_TIMER_0	Timer 0 interrupt
DS1104_INT_TIMER_1	Timer 1 interrupt
DS1104_INT_TIMER_2	Timer 2 interrupt
DS1104_INT_TIMER_3	Timer 3 interrupt
DS1104_INT_EXTERNAL_0	External interrupt 0
DS1104_INT_EXTERNAL_1	External interrupt 1
DS1104_INT_EXTERNAL_2	External interrupt 2
DS1104_INT_EXTERNAL_3	External interrupt 3
DS1104_INT_HOST	Host interrupt
DS1104_INT_SLAVE_DSP	Slave DSP interrupt
DS1104_INT_SLAVE_DSP_PWM	Slave DSP interrupt PWM generation
DS1104_INT_SERIAL_UART	Serial UART interrupt
DS1104_INT_INC_ENC_CH1	Encoder index channel 1

Predefined Symbol	Meaning
DS1104_INT_INC_ENC_CH2	Encoder index channel 2
DS1104_INT_ADC_CONVERSION_1	ADC 1 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_2	ADC 2 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_3	ADC 3 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_4	ADC 4 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_5	ADC 5 end-of-conversion interrupt

Note

The level triggered interrupt (serial UART interrupt) has to be acknowledged in the interrupt service routine before the interrupt is 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

Elementary Data Types... 17

ds1104_reset_interrupt_flag

Syntax

`void ds1104_reset_interrupt_flag(UInt32 IntID)`

Include file

`int1104.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
DS1104_INT_DECREMENTER	Decrementer interrupt
DS1104_INT_TIMER_0	Timer 0 interrupt
DS1104_INT_TIMER_1	Timer 1 interrupt
DS1104_INT_TIMER_2	Timer 2 interrupt
DS1104_INT_TIMER_3	Timer 3 interrupt
DS1104_INT_EXTERNAL_0	External interrupt 0
DS1104_INT_EXTERNAL_1	External interrupt 1
DS1104_INT_EXTERNAL_2	External interrupt 2
DS1104_INT_EXTERNAL_3	External interrupt 3
DS1104_INT_HOST	Host interrupt
DS1104_INT_SLAVE_DSP	Slave DSP interrupt
DS1104_INT_SLAVE_DSP_PWM	Slave DSP interrupt PWM generation
DS1104_INT_SERIAL_UART	Serial UART interrupt
DS1104_INT_INC_ENC_CH1	Encoder index channel 1
DS1104_INT_INC_ENC_CH2	Encoder index channel 2
DS1104_INT_ADC_CONVERSION_1	ADC 1 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_2	ADC 2 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_3	ADC 3 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_4	ADC 4 end-of-conversion interrupt
DS1104_INT_ADC_CONVERSION_5	ADC 5 end-of-conversion interrupt

Note

The level triggered interrupt (serial UART interrupt) has to be acknowledged in the interrupt service routine before the interrupt is enabled globally again.

Return value

None

Related topics

References

Elementary Data Types........ 17

DS1104_GLOBAL_INTERRUPT_ENABLE()

Syntax

```
DS1104_GLOBAL_INTERRUPT_ENABLE
```

or

```
RTLIB_INT_ENABLE
```

Include file

`int1104.h`

Purpose

To globally enable the interrupts.

Description

However, only those hardware interrupts are available that are additionally enabled by `ds1104_enable_hardware_int`.

Return value

None

Related topics

References

ds1104_disable_hardware_int 104
ds1104_enable_hardware_int 103
DS1104_GLOBAL_INTERRUPT_DISABLE() 109
Standard Macros 296

DS1104_GLOBAL_INTERRUPT_DISABLE()

Syntax

```
DS1104_GLOBAL_INTERRUPT_DISABLE
```

or

```
RTLIB_INT_DISABLE
```

Include file

`int1104.h`

Purpose

To globally disable the interrupts.

Return value

None

Related topics

References

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

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

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

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 114
- Example of Using a Subinterrupt Handler on page 114
- Example of Using a Subinterrupt Receiver on page 115
- Example of Using Subinterrupts with Slave Communication on page 116

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 <Defxxxx.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

Examples

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(subint >= 0);
  /* Call the scheduler */
   if (task)
      rtk_scheduler(task);
}
```

Related topics

Basics

Examples

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 114) 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........113

Examples

Example of Using a Subinterrupt Handler........114
Example of Using a Subinterrupt Sender........114

Example of Using Subinterrupts with Slave Communication

Example

The following example demonstrates how to use subinterrupt functions with Slave communication. You find the relevant files in the directory `<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\SlaveDSP\Slv_SubInt_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <Brtenv.h>
#define DT 1.0e-3                    /* 1 ms sample period */
#define NINPUTS 2                    /* number of INPUTS */
/* variables for communication with Slave DSP */
Int16 index_ch1 = -1;            /* command table index for ch. 1 */
Int16 index_ch2 = -1;            /* command table index for ch. 2 */
Int16 task_id = 0;               /* communication channel */
/* variables for ControlDesk */
Float64 freq[NINPUTS];
UInt16 status[NINPUTS];
Int32 slave_err1;    /* function return error code for ch. 1 */
Int32 slave_err2;          /* function return error code for ch. 2 */
Int32 sint_type;           /* type of subinterrupt */
Float64 exec_time;         /* execution time */
void slave_interrupt(void)
{
   RTLIB_TIC_START();                    /* start time measurement */
   dssint_acknowledge(dssint_slvdsp_fw_receiver);
   while ((sint_type = dssint_decode(dssint_slvdsp_fw_receiver)) !=
         SINT_NO_SUBINT)
   {
      switch (sint_type)
      {
         case (SINT1104_DSP2PPC_FW_F2D_CH1):
           /* request read frequency from slave DSP */
           slave_err1 = ds1104_slave_dsp_f2d_read(
              task_id, index_ch1, &freq[0], &status[0]);
           break;
         case (SINT1104_DSP2PPC_FW_F2D_CH2):
           /* request read frequency from slave DSP */
           slave_err2 = ds1104_slave_dsp_f2d_read(
              task_id, index_ch2, &freq[1], &status[1]);
           break;
         default:
           break;
      }
   }
   exec_time = RTLIB_TIC_READ();
}
void srt_isr(void)
{
   ts_timestamp_type ts;
   RTLIB_SRT_ISR_BEGIN();                /* overload check */
   /* request AD read for subinterrupt mode */
   ds1104_slave_dsp_f2d_read_request(task_id, index_ch1);
   ds1104_slave_dsp_f2d_read_request(task_id, index_ch2);
   ts_timestamp_read(&ts);
   host_service(1,&ts);              /* data acquisition service */
   RTLIB_SRT_ISR_END();                        /* overload check */
}
```

```
void main(void)
{
   /* DS1104 and RTLib1104 initialization */
   init();
   /* init communication with slave_dsp */
   ds1104_slave_dsp_communication_init();
   /* init of F2D frequency measurement on slave DSP */
   ds1104_slave_dsp_f2d_init(task_id, 1.0, 1.0, 1.0, 1.0);
   /* set interrupt for slave DSP and enable */
   ds1104_set_interrupt_vector(
      DS1104_INT_SLAVE_DSP,
      (DS1104_Int_Handler_Type) &slave_interrupt,
      SAVE_REGS_ON);
   ds1104_enable_hardware_int(DS1104_INT_SLAVE_DSP);
   RTLIB_INT_ENABLE();
   /* registration of F2D read commands */
   ds1104_slave_dsp_f2d_read_register(
      task_id, &index_ch1, 1, SLVDSP1104_INT_ENABLE);
   ds1104_slave_dsp_f2d_read_register(
      task_id, &index_ch2, 2, SLVDSP1104_INT_ENABLE);
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   RTLIB_SRT_START(DT, srt_isr);      /* start sample rate timer */
   /* Background task */
   while(1)
   {
     RTLIB_BACKGROUND_SERVICE();           /* background service */
   }
}
```

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..... 113

Examples

Example of Using a Subinterrupt Handler..... 114
Example of Using a Subinterrupt Receiver..... 115
Example of Using a Subinterrupt Sender..... 114

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 114.

Related topics

Basics

Examples

References

dssint_define_int_sender_1

Syntax

```
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)
```

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_1` and `dssint_define_int_receiver_1` define the sender and receiver of a subinterrupt in the following way:

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 120.

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 114.

Related topics

Basics

Examples

References

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 120.

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 115.

Related topics

Basics

Examples

References

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 120.

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 115.

Related topics

Basics

Examples

References

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

References

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........113

References

dssint_subint_disable........127
dssint_subint_reset........132

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 114.

Related topics

Basics

Examples

References

dssint_decode

Syntax

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

Include file

`dssint.h`

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 `dssint_acknowledge`, 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 SINT_NO_SUBINT ("–1").

Example

See Example of Using a Subinterrupt Handler on page 114.

Related topics

Basics

Examples

References

dssint_acknowledge

Syntax

```
void dssint_acknowledge(dssint_receiver_type *receiver)
```

Include file

`dssint.h`

Purpose

To acknowledge pending subinterrupts.

Description

This function acknowledges the interrupt by reading `receiver->receiver_addr` (hardware acknowledge), and copies the subinterrupt information to the receiver data structure. Then it performs the software acknowledgment for every pending subinterrupt.

For information on the receiver data structure, refer to the type definition given in Data Types for Subinterrupt Handling on page 118.

Parameters

receiver	Specifies the receiver handler the subinterrupt is located in.

Example

See Example of Using a Subinterrupt Handler on page 114.

Related topics

Basics

Examples

References

dssint_subint_reset

Syntax

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

Include file

`dssint.h`

Purpose

To clear a pending subinterrupt.

Parameters

receiver Specifies the receiver handler the subinterrupt is located in.

subinterrupt Specifies the subinterrupt to reset.

Example

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

Related topics

Basics

References

DMA Function Interface

Introduction

This section contains basic information about the RTLib functions that you can use to program a memory transfer via the DMA (Direct Memory Access) controller of your PowerPC. Technical details can be found in the *MPC8240 Integrated Processor User's Manual* by MOTOROLA.

Where to go from here

Information in this section

Basic Principles of DMA Memory Transfer

Introduction

The DMA controller transfers blocks of data independently of the processor or PCI hosts. The DS1104 board has two DMA channels, each with a 64-byte queue to facilitate the gathering and sending of data within the local memory.

The DMA channels can be used in direct mode using function parameters, and in chaining mode using a descriptor table with specific transfer data. While transfers in direct mode can only be started once, transfers in chaining mode can also be started by a timer for periodic data movements.

Related topics

Examples

Example of Using DMA Controller Functions........ 135

References

Data Types for DMA Memory Transfer........ 136

Example of Using DMA Controller Functions

Example source code

This example shows how to program a single DMA transfer on channel 0 in chaining mode using a descriptor table.

```
dma_descr_ptr descriptor;
int i, entries = 100;
// allocate memory for descriptor data
descriptor = (dma_descr_ptr) malloc((entries+1)*32);
if (descriptor == NULL)
   exit(1);
// align descriptor start address to an 8-word boundary
descriptor = (dma_descr_ptr) (((UInt32) &descriptor[1]) &
               0xFFFFFFE0);
// build descriptor table
for (i = 0; i < entries; i++)
{
   if (i == (entries-1) )
      ds1104_dma_add_descr((UInt32) descriptor, i,
         (UInt32) &source[i],
         (UInt32) &destination[i], 4,
         end_flag);
   else
      ds1104_dma_add_descr((UInt32) descriptor, i,
         (UInt32) &source[i],
         (UInt32) &destination[i], 4,
         end_flag);
}
ds1104_dma_init_chaining_transfer(0, (UInt32) descriptor,
         periodic);
ds1104_dma_transfer_start(0);
```

Data Types for DMA Memory Transfer

dma_descr

```
typedef struct{
   UInt32   src_addr;    /* source address */
   UInt32   dest_addr;   /* destination address */
   UInt32   next;        /* next descriptor */
   UInt32   count;       /* byte count */
} dma_descr;
```

dma_descr_ptr

```
typedef dma_descr * dma_descr_ptr;
```

Related topics

References

Elementary Data Types.. 17

ds1104_dma_init_direct_transfer

Syntax

```
void ds1104_dma_init_direct_transfer(
   int channel,
   UInt32 src_addr,
   UInt32 dst_addr,
   UInt32 count)
```

Include file

`dma1104.h`

Purpose

To initialize a direct mode transfer of the specified DMA controller channel.

Result

If another periodic DMA transfer is active, it is disabled and the function waits for 2 seconds. If the DMA controller is not idle after this time, an error message appears.

Parameters

channel DMA channel number (0, 1)

src_addr address of the source memory

dst_addr address of the destination memory

count no. of bytes to be transferred

Related topics

References

ds1104_dma_init_chaining_transfer

Syntax

```
void ds1104_dma_init_chaining_transfer(
   int channel,
   UInt32 descr_addr,
   int periodic)
```

Include file

`dma1104.h`

Purpose

To initialize a chaining mode transfer of the specified DMA controller.

Result

If another periodic DMA transfer is active, it is disabled and the function waits for 2 seconds. If the DMA controller is not idle after this time, an error message appears.

Description

This function needs a user-specified descriptor table, which provides the parameters for the DMA transfer. An entry in the descriptor table can be generated with the function `ds1104_dma_add_descr`. It must have the following format:

Offset	Local Memory
0x00	Source address
0x04	Reserved
0x08	Destination address
0x0C	Reserved
0x10	Address of the next descriptor entry
0x14	Reserved
0x18	Byte count
0x1C	Reserved

If the chaining mode transfer is initialized, you can start the DMA transfer by using `ds1104_dma_periodic_transfer_start`.

Parameters

channel DMA channel number (0, 1)

descr_addr address of the descriptor table - must be aligned to an 8-word boundary

periodic enables or disables periodic DMA transfer. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_DMA_PERIODIC_ENABLE	To enable the periodic DMA transfer mode
DS1104_DMA_PERIODIC_DISABLE	To disable the periodic DMA transfer mode

Note

If you use the periodic DMA mode, the function `ds1104_dma_periodic_transfer_start` sets Timer 2 for DMA channel 0 and Timer 3 for DMA channel 1. The timer is reconfigured and further interrupt generation is disabled.

Related topics

Examples

Example of Using DMA Controller Functions 135

References

ds1104_dma_add_descr 138
ds1104_dma_init_direct_transfer 136
ds1104_dma_periodic_transfer_start 140

ds1104_dma_add_descr

Syntax

```
void ds1104_dma_add_descr(
   UInt32 descr_addr,
   UInt32 index,
   UInt32 src_addr,
   UInt32 dst_addr,
   UInt32 count,
   int end_flag)
```

Include file

`dma1104.h`

Purpose

To add an entry in the descriptor table.

> **Note**
>
> The memory for the descriptor table must be allocated beforehand. Because the descriptor table address must be aligned to an 8-word memory boundary, the total size of the descriptor memory must be 32 * (number_of_entries + 1). If the specified descriptor table address does not fulfill the 8-word boundary alignment, an error message appears and the application stops.

Parameters

descr_addr address of the descriptor table

index no. of the descriptor table entry

src_addr address of the source memory

dst_addr address of the destination memory

count no. of bytes to be transferred

end_flag flag indicating the last descriptor entry
The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_DMA_DESCR_CONTINUE	Further entry in descriptor table follows
DS1104_DMA_DESCR_END	Last entry in descriptor table

Related topics

Examples

Example of Using DMA Controller Functions..135

References

ds1104_dma_init_chaining_transfer..137

ds1104_dma_transfer_start

Syntax

```
void ds1104_dma_transfer_start(int channel)
```

Include file

`dma1104.h`

Purpose	To start a single DMA transfer in either direct or chaining mode.
Result	The transfer initialized by `ds1104_dma_init_direct_transfer` or `ds1104_dma_init_chaining_transfer` starts and the specified number of bytes is copied from the source address to the destination address.
Parameters	**channel** DMA channel number (0, 1)

Related topics

Examples

Example of Using DMA Controller Functions........135

References

ds1104_dma_init_chaining_transfer........137
ds1104_dma_init_direct_transfer........136

ds1104_dma_periodic_transfer_start

Syntax

```
void ds1104_dma_periodic_transfer_start(
    int channel,
    Float64 period)
```

Include file

`dma1104.h`

Purpose

To start a periodic DMA transfer in chaining mode.

Description

A periodic DMA transfer must be initialized beforehand by using `ds1104_dma_init_chaining_transfer` with periodic mode enabled.

> **Note**
>
> If you use the periodic DMA mode, this function sets Timer 2 for DMA channel 0 and Timer 3 for DMA channel 1. The respective timer is stopped and reinitialized with the specified sampling (transfer) period. Further interrupt generation for that timer is disabled. Each time the timer counter expires a DMA transfer is triggered.

Note

The specified DMA transfer period must be longer than the time for a complete transfer required by the DMA controller. Otherwise the operation can lead to an unpredictable result.

Parameters

channel DMA channel number (0, 1)

period DMA transfer period in seconds

Related topics

References

ds1104_dma_init_chaining_transfer 137
Timer 2 70
Timer 3 74

ds1104_dma_status_read

Syntax

```
UInt32 ds1104_dma_status_read(int channel)
```

Include file

`dma1104.h`

Purpose

To read the status register of the specified DMA channel.

Parameters

channel DMA channel number (0, 1)

Return value

contents of the status register. The following predefined symbols specify certain DMA errors:

Predefined Symbol	Meaning
DS1104_DMA_SR_LME	Local memory error
DS1104_DMA_SR_PE	PCI error (abort condition or parity error)
DS1104_DMA_SR_CB	DMA channel busy
DS1104_DMA_SR_EOSI	DMA transfer finished (CDAR[EOSIE] = 1)

Predefined Symbol	Meaning
DS1104_DMA_SR_EOCAI	DMA transfer finished (MDR[EOTIE] = 1)

Stack Overflow Detection

Introduction

This chapter contains basic information about the Stack Overflow Detection module for a PowerPC.

Where to go from here

Information in this section

Basic Principles of Stack Overflow Detection

Introduction

The Stack Overflow Detection module provides functions to get information about the size of the PowerPC stack and to activate or deactivate the stack size monitoring.

Program abortions due to incorrect stack manipulation are difficult to debug. The point at which the exception leads to a program exit need not be the place on which the error occurred. To get an error message as soon as a stack overflow or a stack underflow occurs, each activity of the stack must be monitored. This can be done by the Stack Overflow Detection, which realizes a program exit with detailed information about the stack malfunction.

Normally, the default size of the stack meets the requirements of most of the applications. If your application needs a larger stack size, you can change the value of the symbol STACK_SIZE that is specified in the Linker command file `DSxxxx.lk` (`xxxx` denotes the relevant dSPACE board).

ppc_stack_control_enable

Syntax

```
void ppc_stack_control_enable(void)
```

Include file	`ppcstack.h`
Purpose	To activate stack monitoring.

Description

This function monitors each stack activity. If a stack overflow or a stack underflow occurs, the Stack Overflow Detection generates a detailed error message and stops the program in a controlled way.

> **Note**
>
> - If you install an interrupt service routine in an application with activated stack monitoring, you must set the `SaveRegs` parameter from the interrupt service to SAVE_REGS_ON to save and restore the registers.
> - Active stack monitoring increases the execution times of the application.

Return value

None

Example

```
#include <Brtenv.h>
...
void main(void)
{
   init();
   ...
   ppc_stack_control_enable();
   ...
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();
   }
}
```

Related topics

Basics

Basic Principles of Stack Overflow Detection 143

References

ppc_stack_control_disable 145

ppc_stack_control_disable

Syntax	`void ppc_stack_control_disable(void)`
Include file	`ppcstack.h`
Purpose	To deactivate stack monitoring.
Description	This function stops the stack monitoring of the Stack Overflow Detection module.
Return value	None
Related topics	Basics

References

ppc_stack_size_get

Syntax	`UInt32 ppc_stack_size_get(void)`
Include file	`ppcstack.h`
Purpose	To get the size of the total stack.
Description	This function reads the value of the parameter `STACK_SIZE` defined in the linker command file `Dsxxxx.lk`. It can be used without active stack monitoring.
Return value	This function returns the size of the total stack in bytes.

Related topics

Basics

Basic Principles of Stack Overflow Detection.. 143

References

ppc_available_relative_stack_size_get.. 146
ppc_available_stack_size_get.. 146

ppc_available_stack_size_get

Syntax

```
UInt32 ppc_available_stack_size_get(void)
```

Include file

`ppcstack.h`

Purpose

To get the size of the free stack.

Description

This function can be used without active stack monitoring.

Return value

This function returns the size of the free stack in bytes.

Related topics

Basics

Basic Principles of Stack Overflow Detection.. 143

References

ppc_available_relative_stack_size_get.. 146
ppc_stack_size_get.. 145

ppc_available_relative_stack_size_get

Syntax

```
Float64 ppc_available_relative_stack_size_get(void)
```

Include file	`ppcstack.h`
Purpose	To get the relation between the currently free and the total stack size.
Description	This function can be used without active stack monitoring.
Return value	This function returns the free stack size divided by total stack size.

Related topics

Basics

Basic Principles of Stack Overflow Detection.. 143

References

ppc_available_stack_size_get.. 146
ppc_stack_size_get.. 145

Exception Handling

Introduction

There are some exceptions in the execution of the PowerPC 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, and the program terminates. The following descriptions only relate to the handling of arithmetical floating point exceptions. If you want to use one of the exceptions, you have to install an appropriate handler.

To get detailed information on program errors, you can debug your application. For further information, refer to Debugging an Application on page 402.

Where to go from here

Information in this section

Definition of the Exception Handler Function Type

Exception handler function type

The exception handler function type is defined as follows.

Syntax

```
typedef void (*DS1104_Exc_Handler_Type)(
      UInt ExcID,
      UInt32 *ExcAddr,
      UInt32 Counter,
      struct SaveRegs *Regs)
```

Parameters

ExcID Specifies the identification of the exception that is handled by this function.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EXC_DZE	Division by zero
DS1104_EXC_FOV	Overflow
DS1104_EXC_UNF	Underflow
DS1104_EXC_INE	Inexact result
DS1104_EXC_SNAN	Not a number
DS1104_EXC_ISI	Infinity subtract infinity
DS1104_EXC_IDI	Infinity add infinity
DS1104_EXC_ZDZ	Zero divide zero
DS1104_EXC_IMZ	Infinity multiply zero
DS1104_EXC_VC	Invalid compare
DS1104_EXC_SQRT	Invalid square root
DS1104_EXC_ICON	Invalid integer conversion

ExcAddr Specifies the program address subsequent to the instruction that caused the exception.

Counter Specifies the current counter for this type of exception.

Regs Specifies the pointer to a data structure used to save the processor registers. The registers are restored from this data structure after returning from the handler function.

Note

If the underflow exception is enabled, an underflow exception occurs for values less than 2.2e-308. If the underflow exception is disabled, a calculation with such a value will return 0. To avoid the misrepresentation of 0 with enabled underflow exception, you can specify a limit (i.e. ±10e-200) in your application at which the result is changed exactly to 0, or you can disable the underflow exception.

The same problem can appear for overflow exceptions.

For more information about the floating point model of the processor, refer to the PowerPC documentation available at http://www.nxp.com/.

ds1104_exception_handler_set

Syntax

```
DS1104_Exc_Handler_Type ds1104_exception_handler_set(
     UInt32 ExcID,
     DS1104_Exc_Handler_Type Handler,
     UInt32 ExcMode)
```

Include file

`Exc1104.h`

Purpose

To install and uninstall an exception handler for the specified exception.

Description

To uninstall an exception handler, enter "0" as `DS1104_Exc_Handler_Type`.

Parameters

ExcID Specifies the identification of the exception that is handled by this function.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EXC_DZE	Division by zero
DS1104_EXC_FOV	Overflow
DS1104_EXC_UNF	Underflow
DS1104_EXC_INE	Inexact result
DS1104_EXC_SNAN	Not a number
DS1104_EXC_ISI	Infinity subtract infinity
DS1104_EXC_IDI	Infinity add infinity
DS1104_EXC_ZDZ	Zero divide zero

Predefined Symbol	Meaning
DS1104_EXC_IMZ	Infinity multiply zero
DS1104_EXC_VC	Invalid compare
DS1104_EXC_SQRT	Invalid square root
DS1104_EXC_ICON	Invalid integer conversion

Handler Specifies the address of the exception handler function (pointer) or 0 to deinstall an exception handler.

ExcMode Specifies the additional information to be given. You can combine the predefined symbols using the logical operator (OR). The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EM_STOP	Stops execution of the program after exception handling
DS1104_EM_CHKERR	Writes information for the dSPACE experiment to the global memory
DS1104_EM_DUMP	Writes a full register dump to the global memory
DS1104_EM_LAST_EXC	In combination with DS1104_EM_CHKERR and DS1104_EM_DUMP: reports the last occurred exception. On default the first exception is reported.

Return value

This function returns the address of the handler function that was previously installed for this exception.

Related topics

References

ds1104_all_exception_handlers_set 152

ds1104_all_exception_handlers_set

Syntax

```
void ds1104_all_exception_handlers_set(
      DS1104_Exc_Handler_Type Handler,
      UInt32 ExcMode)
```

Include file

`Exc1104.h`

Purpose

To install a common exception handler for all types of exceptions.

Parameters

Handler Specifies the address of the common exception handler function.

ExcMode Specifies the additional information to be given. You can combine the predefined symbols using the logical operator (OR). The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EM_STOP	Stops execution of the program after exception handling
DS1104_EM_CHKERR	Writes information for the dSPACE experiment to the global memory
DS1104_EM_DUMP	Writes a full register dump to the global memory
DS1104_EM_LAST_EXC	In combination with DS1104_EM_CHKERR and DS1104_EM_DUMP: reports the last occurred exception. On default the first exception is reported.

Related topics

References

ds1104_exception_handler_set..151

ds1104_exception_enable

Syntax

```
void ds1104_exception_enable(UInt32 ExcID)
```

Include file

`Exc1104.h`

Purpose

To enable the specified exception.

Description

However, the exception is available only when the exceptions are globally enabled (see `ds1104_global_exception_disable`).

Parameters

ExcID Specifies the identification of the exception that is handled by this function.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EXC_DZE	Division by zero
DS1104_EXC_FOV	Overflow
DS1104_EXC_UNF	Underflow
DS1104_EXC_INE	Inexact result
DS1104_EXC_SNAN	Not a number
DS1104_EXC_ISI	Infinity subtract infinity
DS1104_EXC_IDI	Infinity add infinity
DS1104_EXC_ZDZ	Zero divide zero
DS1104_EXC_IMZ	Infinity multiply zero
DS1104_EXC_VC	Invalid compare
DS1104_EXC_SQRT	Invalid square root
DS1104_EXC_ICON	Invalid integer conversion

Related topics

References

ds1104_exception_disable

Syntax

```
void ds1104_exception_disable(UInt32 ExcID)
```

Include file

`Exc1104.h`

Purpose

To disable the specified exception when the exceptions are still globally enabled (see **`ds1104_global_exception_enable`**).

Parameters

ExcID Specifies the identification of the exception that is handled by this function.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EXC_DZE	Division by zero
DS1104_EXC_FOV	Overflow

Predefined Symbol	Meaning
DS1104_EXC_UNF	Underflow
DS1104_EXC_INE	Inexact result
DS1104_EXC_SNAN	Not a number
DS1104_EXC_ISI	Infinity subtract infinity
DS1104_EXC_IDI	Infinity add infinity
DS1104_EXC_ZDZ	Zero divide zero
DS1104_EXC_IMZ	Infinity multiply zero
DS1104_EXC_VC	Invalid compare
DS1104_EXC_SQRT	Invalid square root
DS1104_EXC_ICON	Invalid integer conversion

Related topics

References

ds1104_exception_enable... 153
ds1104_global_exception_disable.. 156

ds1104_global_exception_enable

Syntax

```
void ds1104_global_exception_enable(void)
```

Include file

`Exc1104.h`

Purpose

To enable all exceptions that were enabled before using `ds1104_exception_enable`.

Related topics

References

ds1104_exception_enable... 153
ds1104_global_exception_disable.. 156

ds1104_global_exception_disable

Syntax

```
void ds1104_global_exception_disable(void)
```

Include file

`Exc1104.h`

Purpose

To disable all exceptions.

Related topics

References

ds1104_exception_disable........ 154
ds1104_global_exception_enable........ 155

ds1104_exception_mode_get

Syntax

```
UInt32 ds1104_exception_mode_get(UInt32 ExcID)
```

Include file

`Exc1104.h`

Purpose

To get the exception mode for the specified exception.

Parameters

ExcID Specifies the identification of the exception that is handled by this function.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EXC_DZE	Division by zero
DS1104_EXC_FOV	Overflow
DS1104_EXC_UNF	Underflow
DS1104_EXC_INE	Inexact result
DS1104_EXC_SNAN	Not a number
DS1104_EXC_ISI	Infinity subtract infinity
DS1104_EXC_IDI	Infinity add infinity
DS1104_EXC_ZDZ	Zero divide zero
DS1104_EXC_IMZ	Infinity multiply zero

Predefined Symbol	Meaning
DS1104_EXC_VC	Invalid compare
DS1104_EXC_SQRT	Invalid square root
DS1104_EXC_ICON	Invalid integer conversion

Return value

This function returns the current exception mode. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EM_STOP	Stops execution of the program after exception handling
DS1104_EM_CHKERR	Writes information for the dSPACE experiment to the global memory
DS1104_EM_DUMP	Writes a full register dump to the global memory
DS1104_EM_LAST_EXC	In combination with DS1104_EM_CHKERR and DS1104_EM_DUMP: reports the last occurred exception. On default the first exception is reported.

Related topics

References

ds1104_exception_mode_set..157

ds1104_exception_mode_set

Syntax

```
UInt32 ds1104_exception_mode_set(
      UInt32 ExcID,
      UInt32 ExcMode)
```

Include file

`Exc1104.h`

Purpose

To set the exception mode for the specified handler.

Parameters

ExcID Specifies the identification of the exception that is handled by this function.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EXC_DZE	Division by zero
DS1104_EXC_FOV	Overflow
DS1104_EXC_UNF	Underflow
DS1104_EXC_INE	Inexact result
DS1104_EXC_SNAN	Not a number
DS1104_EXC_ISI	Infinity subtract infinity
DS1104_EXC_IDI	Infinity add infinity
DS1104_EXC_ZDZ	Zero divide zero
DS1104_EXC_IMZ	Infinity multiply zero
DS1104_EXC_VC	Invalid compare
DS1104_EXC_SQRT	Invalid square root
DS1104_EXC_ICON	Invalid integer conversion

ExcMode Specifies the additional information to be given. You can combine the predefined symbols using the logical operator (OR). The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EM_STOP	Stops execution of the program after exception handling
DS1104_EM_CHKERR	Writes information for the dSPACE experiment to the global memory
DS1104_EM_DUMP	Writes a full register dump to the global memory
DS1104_EM_LAST_EXC	In combination with DS1104_EM_CHKERR and DS1104_EM_DUMP: reports the last occurred exception. On default the first exception is reported.

Return value

This function returns the exception mode previously assigned to this exception. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EM_STOP	Stops execution of the program after exception handling
DS1104_EM_CHKERR	Writes information for the dSPACE experiment to the global memory
DS1104_EM_DUMP	Writes a full register dump to the global memory
DS1104_EM_LAST_EXC	In combination with DS1104_EM_CHKERR and DS1104_EM_DUMP: reports the last occurred exception. On default the first exception is reported.

Related topics

References

ds1104_exception_mode_get.. 156

ds1104_exception_counter_get

Syntax

```
UInt32 ds1104_exception_counter_get(UInt32 ExcID)
```

Include file

`Exc1104.h`

Purpose

To get the exception counter of the specified exception.

Parameters

ExcID Specifies the identification of the exception that is handled by this function.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EXC_DZE	Division by zero
DS1104_EXC_FOV	Overflow
DS1104_EXC_UNF	Underflow
DS1104_EXC_INE	Inexact result
DS1104_EXC_SNAN	Not a number
DS1104_EXC_ISI	Infinity subtract infinity
DS1104_EXC_IDI	Infinity add infinity
DS1104_EXC_ZDZ	Zero divide zero
DS1104_EXC_IMZ	Infinity multiply zero
DS1104_EXC_VC	Invalid compare
DS1104_EXC_SQRT	Invalid square root
DS1104_EXC_ICON	Invalid integer conversion

Return value

This function returns the counter value for the specified exception.

Related topics

References

ds1104_exception_counter_reset

Syntax

```
UInt32 ds1104_exception_counter_reset(UInt32 ExcID)
```

Include file

`Exc1104.h`

Purpose

To reset the counter of the specified exception.

> **Note**
>
> Resetting of one counter influences the total amount of exceptions (see `ds1104_total_exception_count_get`).

Parameters

ExcID Specifies the identification of the exception that is handled by this function.

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_EXC_DZE	Division by zero
DS1104_EXC_FOV	Overflow
DS1104_EXC_UNF	Underflow
DS1104_EXC_INE	Inexact result
DS1104_EXC_SNAN	Not a number
DS1104_EXC_ISI	Infinity subtract infinity
DS1104_EXC_IDI	Infinity add infinity
DS1104_EXC_ZDZ	Zero divide zero
DS1104_EXC_IMZ	Infinity multiply zero
DS1104_EXC_VC	Invalid compare
DS1104_EXC_SQRT	Invalid square root
DS1104_EXC_ICON	Invalid integer conversion

Return value

This function returns the counter value for the specified exception before resetting.

Related topics

References

ds1104_exception_counter_get ... 159
ds1104_total_exception_count_get ... 161

ds1104_total_exception_count_get

Syntax

```
UInt32 ds1104_total_exception_count_get(void)
```

Include file

`Exc1104.h`

Purpose

To summarize the counter values of all exceptions.

> **Note**
>
> Resetting (see `ds1104_exception_counter_reset`) of one counter influences the total amount of exceptions.

Return value

This function returns the total amount of all exceptions encountered so far.

Related topics

References

ds1104_exception_counter_reset ... 160

ds1104_exception_flag_get

Syntax

```
UInt32 ds1104_exception_flag_get(void)
```

Include file

`Exc1104.h`

Purpose

To get the exception flag indicating whether or not an exception has occurred.

Description

The exception flag is set by each exception.

Return value

This function returns the value of the flag. The following symbols are predefined:

Predefined Symbol	Meaning
CFG_EXC_NO_EXCEPT	No exception
CFG_EXC_FP_ARITHMETICAL	Floating-point arithmetic exception

Related topics

References

ds1104_exception_flag_reset 162

ds1104_exception_flag_reset

Syntax

```
void ds1104_exception_flag_reset(void)
```

Include file

`Exc1104.h`

Purpose

To reset the exception flags.

Return value

This function returns the exception flag value before reset. The following symbols are predefined:

Predefined Symbol	Meaning
CFG_EXC_NO_EXCEPT	No exception
CFG_EXC_FP_ARITHMETICAL	Floating-point arithmetic exception

Related topics

References

ds1104_exception_flag_get 161

Information Handling

Purpose

Use the functions of the information handling to get information on the board version, the memory configuration and the clock frequency.

Where to go from here

Information in this section

ds1104_info_version_board_get

Syntax

```
void ds1104_info_version_board_get(
      UInt32 *version,
      UInt32 *revision,
      UInt32 *sub_version)
```

Include file

`info1104.h`

Purpose

To get the board version.

Parameters

version Specifies the address of the variable containing the board version.

revision Specifies the address of the variable containing the board revision.

sub_version Specifies the address of the variable containing the board subversion.

Related topics

References

Elementary Data Types.. 17

ds1104_info_memory_get

Syntax

```
void ds1104_info_memory_ get(
      UInt32 *memory_size,
      UInt32 *cached_memory_base,
      UInt32 *flash_size,
      UInt32 *flash_base)
```

Include file

`info1104.h`

Purpose

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

Parameters

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

cached_memory_base Specifies the address of the variable containing the global memory base address.

flash_size Specifies the address of the variable containing the flash memory size in bytes.

flash_base Specifies the address of the variable containing the flash memory base address.

Related topics

References

Elementary Data Types.. 17

ds1104_info_clocks_get

Syntax

```
void ds1104_info_clocks_get(
      UInt32 *cpu_clock,
      UInt32 *bus_clock)
```

Include file

`info1104.h`

Purpose

To get frequency information from the config section.

Parameters

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

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

Related topics

References

Elementary Data Types.. 17

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

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

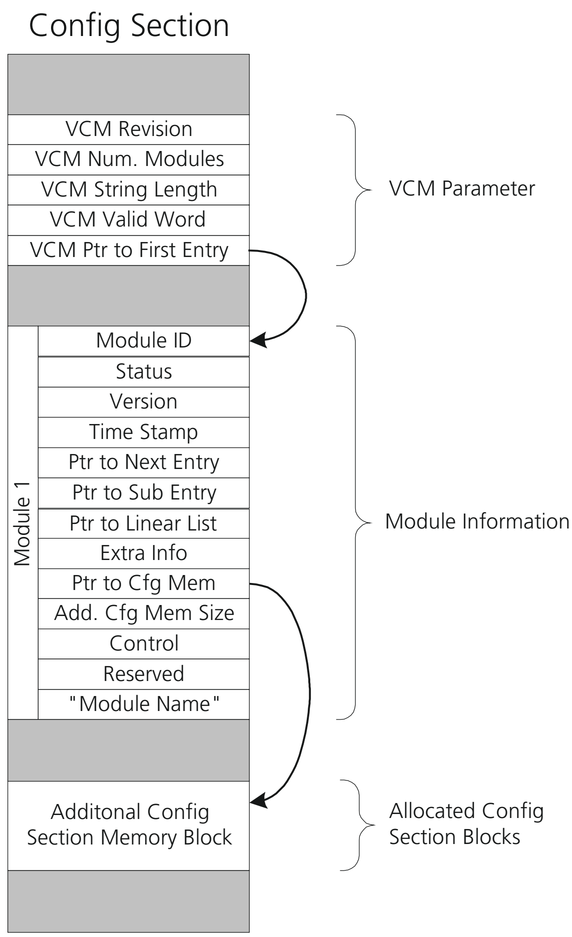
The data structures for version and config section management are located in the global memory of each processor board and can therefore be accessed by the real-time hardware and the host PC. Module version and status information is displayed by the dSPACE experiment software on the property page of 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.



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 171.

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

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.. 168

References

ds1104_init.. 19

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.. 168

References

vcm_memory_ptr_set.. 175
vcm_module_status_set.. 177

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

Basics

Basic Principles of VCM.. 168

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
VCM_NO_ERROR	Pointer and size set successfully
VCM_INVALID_MODULE	Module does not exist

Related topics

Basics

Basic Principles of VCM........168

References

vcm_cfg_malloc........174
vcm_memory_ptr_get........175

vcm_memory_ptr_get

Syntax

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

Include file `dsvcm.h`

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

Basics

References

vcm_module_find

Syntax

```
vcm_module_descriptor_type* vcm_module_find(
   Int32 mod_id,
   vcm_module_descriptor_type *prev_ptr)
```

Include file `dsvcm.h`

Purpose To find a pointer to the module descriptor by a given module ID.

Parameters

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

prev_ptr Specifies the pointer to a previously found module, or 0.

Note

If more than one module with the same module ID are registered, use this parameter to start the search from the previously found pointer.

Return value

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

Related topics

Basics

References

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........168

References

vcm_module_register........172
vcm_module_status_get........178

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

References

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 171).

Related topics

Basics

References

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........168

References

vcm_version_get........179
vcm_version_print........182

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........168

References

vcm_version_print........182

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.. 168

References

vcm_module_version_print.. 181

Message Handling

Purpose

To configure and generate messages.

Where to go from here

Information in this section

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 DS1104 R&D Controller Board, there are the following default values:

Predefined Symbol	Default Value
`MSG_STRING_LENGTH`	80 characters
`MSG_BUFFER_LENGTH`	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 DS1104 -`	`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 185.

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

msg Specifies the message string (for information on the maximum length, see Message characteristics on page 184).

Return value

None

Related topics

Basics

References

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 185.

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

msg Specifies the message string (for information on the maximum length, see Message characteristics on page 184).

Return value

None

Related topics

Basics

References

msg_info_set

Syntax

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

Include file

`dsmsg.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 185.

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

msg Specifies the message string (for information on the maximum length, see Message characteristics on page 184).

Return value

None

Related topics

Basics

References

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	`dsmsg.h`
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 `msg_xxx_set` functions, the user can adjust the type of the message dialogs.

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 **`msg_default_dialog_set`**

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 185.

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 184).

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

References

msg_error_printf

Syntax

```
int msg_error_printf(
   msg_submodule_typemodule,
   msg_no_typemsg_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 203), 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 185.

msg_no Specifies the number of the message within the range of $-2^{31} \ldots 2^{31}-1$ 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 184. 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....184

References

msg_error_hook_set....203
msg_error_set....187

msg_warning_printf

Syntax

```
int msg_warning_printf(
   msg_submodule_typemodule,
   msg_no_typemsg_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 185.

msg_no Specifies the number of the message within the range of $-2^{31} \ldots 2^{31}-1$ 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 184. Longer messages are truncated.

Return value

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

Related topics

Basics

References

msg_info_printf

Syntax

```
int msg_info_printf(
   msg_submodule_typemodule,
   msg_no_typemsg_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 185.

msg_no Specifies the number of the message within the range of -2^{31} … 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 184. Longer messages are truncated.

Return value

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

Related topics

Basics

References

msg_printf

Syntax

```
int msg_printf(
   msg_class_typemsg_class,
   msg_dialog_typemsg_dialog,
   msg_submodule_typemodule,
   msg_no_typemsg_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 **`msg_default_dialog_set`**

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 185.

msg_no Specifies the number of the message within the range of -2^{31} … 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 184. 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

References

msg_default_dialog_set

Syntax

```
void msg_default_dialog_set(
   msg_class_type msg_class,
   msg_dialog_type msg_dialog)
```

Include file

`dsmsg.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

msg_mode_set

Syntax

```
void msg_mode_set(UInt32 mode)
```

Include file

dsmsg.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.. 184

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

References

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

References

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.. 184

References

msg_error_clear.. 202
msg_last_error_number.. 200

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 `MSG_SM_NONE` (refer to Data Types and Symbols for Message Handling on page 185).

Return value

None

Related topics

Basics

References

msg_error_hook_set

Syntax

```
void msg_error_hook_set(msg_hookfcn_type hook)
```

Include file

`dsmsg.h`

Purpose

To install a hook function.

Description

The hook function is activated when an error message is generated (see **`msg_error_set`** and **`msg_error_printf`**) and before the message is displayed.

Use the hook function to:

- React to an error (for example, to implement an error correction function)
- Suppress the error message

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).

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........184

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 init() function. The mode is set to MSG_OVERWRITE, counter and indices are set to 0. The buffer and string lengths are set according to the values of `MSG_BUFFER_LENGTH` and `MSG_STRING_LENGTH` defined in `Msgxxxx.h.`
Return value	None

Related topics

Basics

Basic Principles of Message Handling 184

References

init() 298
msg_mode_set 198

Synchronous I/O Trigger

Introduction

DS1104 provides a feature for the triggering of I/O components.

Where to go from here

Information in this section

Basic Information on the Synchronous I/O Trigger

Introduction

Some applications (e.g., in drives control) require an exact timing for the controller analog inputs, outputs or incremental encoder position reads. Additionally, you might want to have the triggering of I/O components, like conversion start or position read, performed synchronously to a PWM or external hardware signal. For further information, refer to Synchronizing I/O Features of the Master PPC (DS1104 Features 🕮).

When using the standard ADC, DAC or incremental encoder interface functions, the I/O triggering is done via software (e.g., by using the `ds1104_adc_start` function). To get a more accurate I/O timing, the DS1104 provides a synchronous I/O trigger hardware feature:

- *SyncIn* triggers all input components (ADC, Incremental Encoder interface) synchronously
- *SyncOut* triggers all output components (DAC) synchronously

The SyncIn and SyncOut triggers can be activated in three different ways:

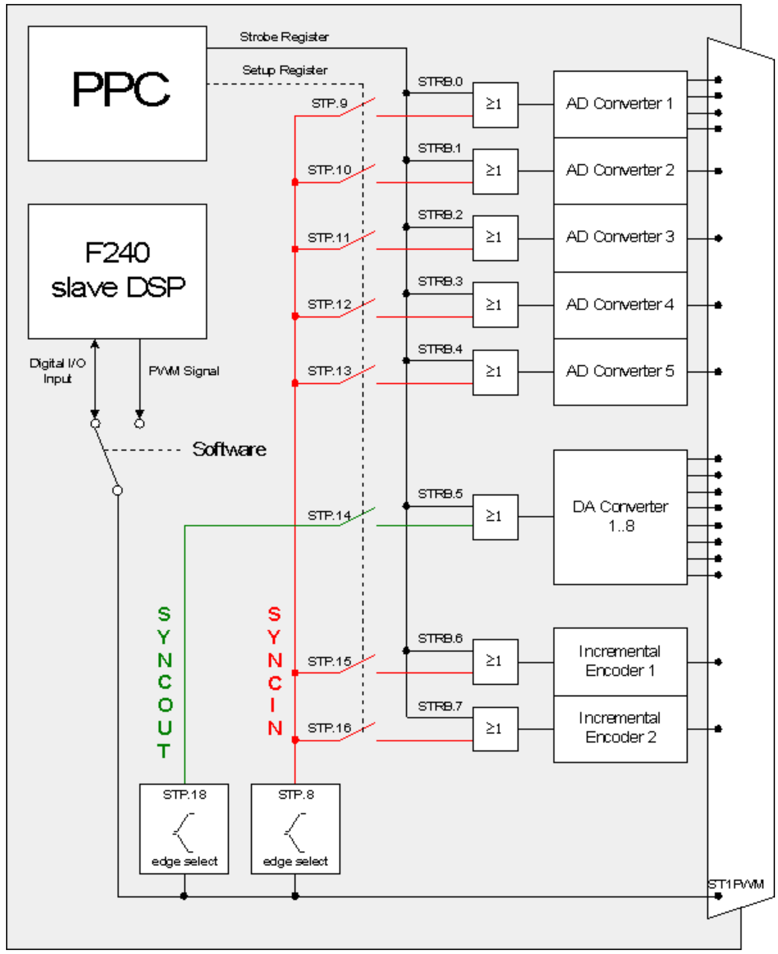
1. Via slave DSP PWM signal

   Using PWM signal for triggering, you must enable triggering of the I/O components and specify the signal edge.
2. Via external trigger

   If you want to use the external trigger, the ST1PWM pin on the DS1104 bracket must be configured as input pin by `ds1104_external_trigger_enable`. Furthermore, you must enable the I/O components for triggering and specify the signal edge.
3. Via software

   You can use `ds1104_syncin_trigger` and `ds1104_syncout_trigger` for the I/O components that are enabled for triggering.

The signal edge, on which the I/O is triggered, can be specified individually for the input and output components. As default, all I/O components are disabled for triggering.



Note

If you enable synchronous triggering of an ADC channel, you cannot use software triggering for the other ADC channels. You cannot mix the trigger modes.

ds1104_syncin_edge_setup

Syntax

```
void ds1104_syncin_edge_setup(UInt16 edge)
```

Include file

`io1104.h`

Purpose

To specify the signal edge of an external trigger for an input component (ADC, Incremental Encoder).

Description

If the input components are triggered by PWM or external input, this function sets up the edge for the SYNCIN event via the ST1PWM pin.

Parameters

edge Specifies the signal edge triggering the input component. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_SYNC_TRIGGER_RISING	Trigger on rising signal edge
DS1104_SYNC_TRIGGER_FALLING	Trigger on falling signal edge

Return value

None

Related topics

References

ds1104_syncin_trigger.......... 210
ds1104_syncout_edge_setup.......... 209

ds1104_syncout_edge_setup

Syntax

```
void ds1104_syncout_edge_setup(UInt16 edge)
```

Include file

`io1104.h`

Purpose

To specify the signal edge of an external trigger for an output component (DAC).

Description

If the output components are triggered by PWM or external input, this function sets up the edge for the SYNCOUT event via the ST1PWM pin.

Parameters

edge Specifies the signal edge triggering the output component. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_SYNC_TRIGGER_RISING	Trigger on rising signal edge
DS1104_SYNC_TRIGGER_FALLING	Trigger on falling signal edge

Return value

None

Related topics

References

ds1104_syncin_edge_setup..208
ds1104_syncout_trigger..211

ds1104_syncin_trigger

Syntax

```
void ds1104_syncin_trigger(void)
```

Include file

`io1104.h`

Purpose

To trigger the input components that are enabled for triggering (ADC, Incremental Encoder).

Description

With this function you can trigger the input components by software, without using the ST1PWM pin.

Return value

None

Related topics

References

ds1104_syncin_edge_setup..208

ds1104_syncout_trigger

Syntax	`void ds1104_syncout_trigger(void)`
Include file	`io1104.h`
Purpose	To trigger the output components that are enabled for external triggering (DAC).
Description	With this function you can trigger the output components by software, without using the ST1PWM pin.
Return value	None
Related topics	References ds1104_syncout_edge_setup 209

ds1104_external_trigger_enable

Syntax	`void ds1104_external_trigger_enable(void)`
Include file	`io1104.h`
Purpose	To enable the external trigger via the ST1PWM pin on the bracket.
Description	The master/slave communication is initialized with the ST1PWM as digital I/O input pin. **Note** Enabling the external trigger conflicts with the slave DSP bit I/O unit. It is not possible to use bit group 2 for digital I/O purposes.

Return value None

ADC Unit

Where to go from here

Information in this section

Information in other sections

Example of Using the ADC Functions

Example source code

The following example demonstrates how to use ADC functions. You find the relevant files in the directory `<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\Adc_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <Brtenv.h>
#define DT 1.0e-4            /* 100 us simulation step size */
#define NINPUTS 4                /* number of INPUTS */
/* scantable array */
UInt16 scantable[4] = {1, 2, 3, 4};
/* switch between channels 1-4 and 5-8 */
int channels = 0;
Float64 dummy;
/* variables for ControlDesk */
Float64 u[NINPUTS];
Float64 exec_time;                          /* execution time */
void isr_srt(void)
{
   ts_timestamp_type ts;
```

```
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);   /* data acquisition service*/
```

```
   if (!channels)
   {
      /* start and read 4 multiplexed 16-bit ADC channels
         subsequently */
      ds1104_adc_read_mux(scantable, 4, u);
   }
   else
   {
      /* start 4 12-bit ADC's simultaneously */
      ds1104_adc_start(DS1104_ADC2 | DS1104_ADC3 |
                       DS1104_ADC4| DS1104_ADC5);
      /* read out the ADC's subsequently */
      ds1104_adc_read_all(&dummy,
         &u[0], &u[1], &u[2], &u[3]);
   }
   exec_time = RTLIB_TIC_READ();
   RTLIB_SRT_ISR_END();   /* overload check */
}
void main(void)
{
   init();   /* DS1104 and RTLib1104 initialization */
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* start sample rate timer */
   RTLIB_SRT_START(DT, isr_srt);
   /* Background task */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();   /* background service */
   }
}
```

ds1104_adc_start

Syntax

```
void ds1104_adc_start(UInt16 mask)
```

Include file

`io1104.h`

Purpose

To start one or more A/D converters.

Description

To start one or more A/D converters with a delay time of 1 µs, use `ds1104_adc_delayed_start`.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS1104 Features).

Parameters

mask Specifies the converters to be started. To start conversion for more than one converter, you can combine the predefined symbols by using the logical operator OR. The following symbols are predefined:

Predefined Symbol	A/D Converter	Channel(s)
DS1104_ADC1	ADC 1	1 … 4
DS1104_ADC2	ADC 2	5
DS1104_ADC3	ADC 3	6
DS1104_ADC4	ADC 4	7
DS1104_ADC5	ADC 5	8

Related topics

Examples

Example of Using the ADC Functions..214

References

ds1104_adc_delayed_start..215

ds1104_adc_delayed_start

Syntax

```
void ds1104_adc_delayed_start(UInt16 mask)
```

Include file	`io1104.h`
Purpose	To start one or more A/D converters with a delay time.
Description	Using this function in combination with `ds1104_adc_mux`, the correct adjusting of a multiplexer is guaranteed. During the delay time the PowerPC is idle.
I/O mapping	For information on the I/O mapping, refer to ADC Unit (DS1104 Features).

Parameters

mask Specifies the converters to be started with a delay time of 1 µs. To start conversion for more than one converter, you can combine the predefined symbols by using the logical operator OR. The following symbols are predefined:

Predefined Symbol	A/D Converter	Channel(s)
DS1104_ADC1	ADC 1	1 … 4
DS1104_ADC2	ADC 2	5
DS1104_ADC3	ADC 3	6
DS1104_ADC4	ADC 4	7
DS1104_ADC5	ADC 5	8

Related topics

References

ds1104_adc_mux...216
ds1104_adc_start...215

ds1104_adc_mux

Syntax

```
void ds1104_adc_mux(UInt16 channel)
```

Include file `io1104.h`

Purpose To set the input multiplexer of the multiplexed A/D converter for the specified channel.

Description

As this setting takes 1 µs, use `ds1104_adc_delayed_start` to start the converter.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 4. The A/D converter ADC 1 is automatically assigned.

Related topics

References

ds1104_adc_read_ch

Syntax

```
Float64 ds1104_adc_read_ch(UInt16 channel)
```

Include file

`io1104.h`

Purpose

To read one A/D channel in polling mode.

Description

Note

- Before using this function, the converter must be started by means of `ds1104_adc_start`.
- The multiplexed converter (channels 1 … 4) must be set with `ds1104_adc_mux` and started by using `ds1104_adc_delayed_start`.
- If you specify a channel of the multiplexed converter, this function reads the value according to the multiplexer settings.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 8.

Return value

This function returns the scanned value scaled as follows:

Input Voltage Range	Return Value Range
–10 V … +10 V	–1.0 … +1.0

Related topics

Basics

ADC Unit (DS1104 Features 🕮)

References

ds1104_adc_read_ch_immediately........218
ds1104_adc_read_conv........219

ds1104_adc_read_ch_immediately

Syntax

```
Float64 ds1104_adc_read_ch_immediately(UInt16 channel)
```

Include file

`io1104.h`

Purpose

To read one A/D channel without polling the end-of-conversion flag.

Description

This function can be used in an ADC end-of-conversion interrupt service routine.

Note

Before using this function, the converter must be started by means of `ds1104_adc_start`. The multiplexed converter (channels 1 … 4) must be set with `ds1104_adc_mux` and started by using `ds1104_adc_delayed_start`.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS1104 Features 🕮).

Parameters

channel Specifies the channel number within the range 1 … 8.

Return value

This function returns the scanned value scaled as follows:

Input Voltage Range	Return Value Range
–10 V … +10 V	–1.0 … +1.0

Related topics

Basics

ADC Unit (DS1104 Features 📖)

References

ds1104_adc_read_conv

Syntax

```
Float64 ds1104_adc_read_conv(UInt16 converter)
```

Include file

`io1104.h`

Purpose

To read from an A/D converter in polling mode.

Description

To read one after another from up to 4 channels of the multiplexed A/D converter, use the function **`ds1104_adc_read_mux`**.

> **Note**
>
> Before using this function, the converters must be started by means of **`ds1104_adc_start`**. The multiplexed converter (ADC 1) must be set with **`ds1104_adc_mux`** and started by using **`ds1104_adc_delayed_start`**.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS1104 Features 📖).

Parameters

converter	Specifies the converter number within the range 1 … 5.

Return value

This function returns the scanned value scaled as follows:

Input Voltage Range	Return Value Range
–10 V … +10 V	–1.0 … +1.0

Related topics

Basics

ADC Unit (DS1104 Features 📖)

References

ds1104_adc_read_conv_immediately

Syntax

```
Float64 ds1104_adc_read_conv_immediately(UInt16 converter)
```

Include file

`io1104.h`

Purpose

To read from an A/D converter without polling the end-of-conversion flag.

Description

This function can be used in an ADC end-of-conversion interrupt service routine.

Note

Before using this function, the converters must be started by means of `ds1104_adc_start`. The multiplexed converter (ADC 1) must be set with `ds1104_adc_mux` and started by using `ds1104_adc_delayed_start`.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS1104 Features 📖).

Parameters

converter Specifies the converter number within the range 1 … 5.

Return value

This function returns the scanned value scaled as follows:

Input Voltage Range	Return Value Range
–10 V … +10 V	–1.0 … +1.0

Related topics

Basics

ADC Unit (DS1104 Features)

References

ds1104_adc_read_mux

Syntax

```
void ds1104_adc_read_mux(
      UInt16 *adc_scantable,
      UInt16 scantable_size,
      Float64 *pvalues)
```

Include file

`io1104.h`

Purpose

To read one after another from up to 4 channels of the multiplexed A/D converter in polling mode.

> **Note**
>
> This function comprises the setting of the multiplexer, the start of the converters, and the read function.

Description

The scanned values are scaled as follows:

Input Voltage Range	Scanned Value Range
–10 V … +10 V	–1.0 … +1.0

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS1104 Features 🕮).

Parameters

adc_scantable Specifies the array address that contains the channel numbers.

scantable_size Specifies the number of channels.

pvalues Specifies the array address where the scanned values are written.

Related topics

Basics

ADC Unit (DS1104 Features 🕮)

References

ds1104_adc_read_all

Syntax

```
void ds1104_adc_read_all(
      Float64 *value1,
      Float64 *value2,
      Float64 *value3,
      Float64 *value4,
      Float64 *value5)
```

Include file

`io1104.h`

Purpose

To read from all A/D converters at the same time in polling mode.

Note

Before using this function, the converters must be started by means of `ds1104_adc_start`. The multiplexed converter (converter 1) must be set with `ds1104_adc_mux` and started by using `ds1104_adc_delayed_start`.

Description

The scanned values are scaled as follows:

Input Voltage Range	Scanned Value Range
–10 V … +10 V	–1.0 … +1.0

Parameters

value1 Specifies the address where the scanned value of the first A/D converter is written.

value2 Specifies the address where the scanned value of the second A/D converter is written.

value3 Specifies the address where the scanned value of the third A/D converter is written.

value4 Specifies the address where the scanned value of the fourth A/D converter is written.

value5 Specifies the address where the scanned value of the fifth A/D converter is written.

Related topics

Basics

ADC Unit (DS1104 Features 🕮)

ds1104_adc_trigger_setup

Syntax

```
void ds1104_adc_trigger_setup(
     UInt16 converter,
     UInt16 state)
```

Include file

`io1104.h`

Purpose

To enable or disable triggering of an A/D converter via synchronous trigger (SyncIn).

Description

A/D converters that are enabled for triggering can be triggered via software, by a slave PWM signal or an external trigger. For further information, refer to Synchronous I/O Trigger on page 206.

Note

If you enable synchronous triggering of an ADC channel, you cannot use software triggering for the other ADC channels. You cannot mix the trigger modes.

I/O mapping

For information on the I/O mapping, refer to ADC Unit (DS1104 Features 🕮).

Parameters

converter Specifies the converter number within the range 1 … 5.

state Specifies the state of the external trigger. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_TRIGGER_DISABLE	Disables the external trigger
DS1104_TRIGGER_ENABLE	Enables the external trigger

Related topics

References

ds1104_syncin_edge_setup.......... 208
Elementary Data Types.......... 17
Synchronous I/O Trigger.......... 206

Bit I/O Unit

Where to go from here

Information in this section

Information in other sections

Example of Using the Bit I/O Functions

Example source code

The following example demonstrates how to use Bit I/O functions. You find the relevant files in the directory `<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\BitIo_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <Brtenv.h>
#include <math.h>
#define DT 1.0e-1            /* 100 ms simulation step size */
```

```
/* variables for ControlDesk */
volatile UInt32 bitmap = 0;
volatile UInt32 mask_clear = 0;
volatile UInt32 mask_set = 0;
volatile UInt32 bitstate[20];
Float64 exec_time;
void isr_srt(void)
{
   ts_timestamp_type ts;
   static UInt32 bitpos = 0;
   UInt32 n;
```

```
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);   /* data acquisition service*/
```

```
   /* sets I/O port i to “1” */
   mask_set = 0x1 << bitpos;
   ds1104_bit_io_set(mask_set);
   /* increments bitpos until channel 10 is reached */
   bitpos++;
   if (bitpos == 10)
      bitpos = 0;
   /* sets I/O port i to “0” */
   mask_clear = 0x1 << bitpos;
   ds1104_bit_io_clear(mask_clear);
   /* reads all 20 Bit I/O ports and writes in UInt32-value */
   bitmap = ds1104_bit_io_read();
   /* updates the bitstate array */
   for (n = 0; n < 20; n++)
   {
      if (bitmap & (0x1 << n))
         bitstate[n] = 1;
      else
         bitstate[n] = 0;
   }
   exec_time = RTLIB_TIC_READ();
   RTLIB_SRT_ISR_END();
}
void main(void)
{
   init();   /* DS1104 and RTLib1104 initialization */
  /* sets IO0 to IO9 to output and IO10 to IO19 to input */
   ds1104_bit_io_init(DS1104_DIO0_OUT | DS1104_DIO1_OUT |
                      DS1104_DIO2_OUT | DS1104_DIO3_OUT |
                      DS1104_DIO4_OUT | DS1104_DIO5_OUT |
                      DS1104_DIO6_OUT | DS1104_DIO7_OUT |
                      DS1104_DIO8_OUT | DS1104_DIO9_OUT |
                      DS1104_DIO10_IN | DS1104_DIO11_IN |
                      DS1104_DIO12_IN | DS1104_DIO13_IN |
                      DS1104_DIO14_IN | DS1104_DIO15_IN |
                      DS1104_DIO16_IN | DS1104_DIO17_IN |
                      DS1104_DIO18_IN | DS1104_DIO19_IN);
```

```
   /* sets the Bit I/O ports 0..9 to "1" */
   /* writing on pins configured as input has no effect */
   ds1104_bit_io_write(0x000003FF);
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* start sample rate timer */
   RTLIB_SRT_START(DT, isr_srt);
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();   /* background service */
   }
}
```

ds1104_bit_io_init

Syntax

```
void ds1104_bit_io_init(UInt32mask)
```

Include file

`io1104.h`

Purpose

To initialize the bit I/O unit and clearing the output pins.

Description

You can configure each bit in the range 0 … 19 for input or output. The pins initialized for output are cleared.

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS1104 Features).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features).

Parameters

mask Configures the I/O pins for input or output. You can use the predefined symbols DS1104_DIOx_IN and DS1104_DIOx_OUT, where x specifies the bit within the range 0 … 19. To set more than one bit at once, you can combine the predefined symbols by using the logical operator OR.:

Predefined Symbol	Meaning
DS1104_DIOx_OUT	Sets I/O pin x to output (x = 0 … 19)
DS1104_DIOx_IN	Sets I/O pin x to input (x = 0 … 19)

Example

To configure the I/O pins 1 and 3 for output, and the I/O pins 2 and 4 for input:

```
ds1104_bit_io_init(DS1104_DIO1_OUT | DS1104_DIO2_IN | DS1104_DIO3_OUT | DS1104_DIO4_IN)
```

Related topics

Examples

Example of Using the Bit I/O Functions........ 225

ds1104_bit_io_init_with_preset

Syntax

```
void ds1104_bit_io_init_with_preset(
      UInt32 mask,
      UInt32 preset)
```

Include file

`io1104.h`

Purpose

To initialize the bit I/O unit and specifying the output pins with a preset value.

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS1104 Features 📖).

Note

The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

mask Configures the I/O pins for input or output. You can use the predefined symbols DS1104_DIOx_IN and DS1104_DIOx_OUT, where x specifies the bit within the range of 0 … 19. To set more than one bit at once, you can combine the predefined symbols by using the logical operator OR.

Predefined Symbol	Meaning
DS1104_DIOx_OUT	Sets I/O pin x to output (x = 0 … 19)
DS1104_DIOx_IN	Sets I/O pin x to input (x = 0 … 19)

preset Specifies the bits to be set. You can use the predefined symbols DS1104_DIOx, where x specifies the bit within the range of 0 … 19. To set more than one bit at once, you can combine the predefined symbols by using the logical operator OR.

Predefined Symbol	Meaning
DS1104_DIOx	Sets bit x to 1 (x = 0 … 19)

Related topics

Examples

References

ds1104_bit_io_write

Syntax

```
void ds1104_bit_io_write(UInt32 value)
```

Include file

`io1104.h`

Purpose

To write a 20-bit value to the digital I/O port.

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS1104 Features 📖).

Note

The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

value Specifies the 20-bit value, which is written to the output pins 0 … 19.

Related topics

Examples

ds1104_bit_io_read

Syntax

```
UInt32 ds1104_bit_io_read(void)
```

Include file

`io1104.h`

Purpose

To read the 20-bit value from the digital I/O port.

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS1104 Features 📖).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Return value

This function returns the 20-bit value, which represents the states of the input pins 0 … 19.

Related topics

Examples

Example of Using the Bit I/O Functions.. 225

ds1104_bit_io_set

Syntax

```
void ds1104_bit_io_set(UInt32 mask)
```

Include file

`io1104.h`

Purpose

To set particular bits of the digital I/O port to 1.

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS1104 Features 📖).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

mask Specifies the bits to be set. You can use the predefined symbols DS1104_DIOx, where x specifies the bit within the range 0 … 19. To set more than one bit at once, you can combine the predefined symbols by using the logical operator OR. Unspecified bits do not change.

Predefined Symbol	Meaning
DS1104_DIOx	Sets bit x to 1 (x = 0 … 19)

Example

To set bit 0 and bit 14:

```
ds1104_bit_io_set(DS1104_DIO0 | DS1104_DIO14)
```

Related topics

Examples

Example of Using the Bit I/O Functions........225

References

ds1104_bit_io_clear........231
ds1104_bit_io_init........227

ds1104_bit_io_clear

Syntax

```
void ds1104_bit_io_clear(UInt32 mask)
```

Include file

`io1104.h`

Purpose

To set particular bits of the digital I/O port to 0.

I/O mapping

For information on the I/O mapping, refer to Bit I/O Unit (DS1104 Features 📖).

Note

The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

mask Specifies the bits to be cleared. You can use the predefined symbols DS1104_DIOx, where x specifies the bit within the range 0 … 19. To clear more than one bit at once, you can combine the predefined symbols by using the logical operator OR. Unspecified bits do not change.

Predefined Symbol	Meaning
DS1104_DIOx	Sets bit x to 0 (x = 0 … 19)

Example

To clear the bits 4 and 18:

```
ds1104_bit_io_clear(DS1104_DIO4 | DS1104_DIO18)
```

Related topics

Examples

References

DAC Unit

Where to go from here

Information in this section

Information in other sections

Example of Using the DAC Functions

Example source code

The following example demonstrates how to use DAC functions. You find the relevant files in the directory <RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\Dac_1104_hc. You can use ControlDesk to load and start the application on the DS1104.

```
#include <Brtenv.h>
#include <math.h>
#define DT 100e-6            /* 100 us simulation step size */
#define PI 3.141592654
/* variables for ControlDesk */
Float64 u;
Float64 exec_time;            /* execution time */
volatile Int32 T = 100;       /* period time in DT seconds */
static Int16 i = 0;
```

```
void isr_srt(void)
{
   ts_timestamp_type ts;

   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);   /* data acquisition service*/

   /* generate a sine function u with period T */
   u = sin(2*PI*i/T);
   i++;
   if (i >= T)
      i = 0;
   /* set DACH1 to u /
   ds1104_dac_write(1, u);
   /* set DACH3 to u /
   ds1104_dac_write(3, u);
   /* set DACH4 to -u /
   ds1104_dac_write(4, -u);
   /* activate the previously written DAC values
      synchronously */
   ds1104_dac_strobe();
   exec_time = RTLIB_TIC_READ();
   RTLIB_SRT_ISR_END();
}
void main(void)
{
   init(); /* DS1104 and RTLib1104 */;
   /* init D/A converter in latched mode */
   ds1104_dac_init(DS1104_DACMODE_LATCHED);
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* start sample rate timer */
   RTLIB_SRT_START(DT, isr_srt);
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();   /* background service */
   }
}
```

ds1104_dac_init

Syntax

```
void ds1104_dac_init(UInt16 dac_mode)
```

Include file

`io1104.h`

Purpose

To initialize the D/A converters and set the DAC mode (transparent or latched).

Description

When using the transparent mode, the written value is output immediately. When using the latched mode, the written value is output only when `ds1104_dac_strobe` is executed, that is, you can write one after another to more than one channel, and output the values simultaneously. Use `ds1104_dac_write` to write to the D/A channels. The DAC output voltage is set to 0 V.

Parameters

dac_mode The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_DACMODE_LATCHED	when latched
DS1104_DACMODE_TRANSPARENT	when transparent

Related topics

Examples

Example of Using the DAC Functions....................233

References

ds1104_dac_reset....................235
ds1104_dac_strobe....................237
ds1104_dac_write....................237

ds1104_dac_reset

Syntax

```
void ds1104_dac_reset(void)
```

Include file

`io1104.h`

Purpose

To reset the D/A converters.

Description

The DAC output voltage is set to 0 V. After reset, you must again initialize with the function `ds1104_dac_init`.

Related topics

References

ds1104_dac_trigger_setup

Syntax

```
void ds1104_dac_trigger_setup(UInt16 state)
```

Include file

`io1104.h`

Purpose

To enable or disable triggering of all D/A converters via synchronous trigger (SyncOut).

Description

D/A converters that are enabled for triggering can be triggered via software, by a slave PWM signal or an external trigger. For further information, refer to Synchronous I/O Trigger on page 206.

Parameters

state Specifies the state of the external trigger. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_TRIGGER_DISABLE	Disables the external trigger
DS1104_TRIGGER_ENABLE	Enables the external trigger

Return value

None

Related topics

References

ds1104_dac_write

Syntax

```
void ds1104_dac_write(
      UInt16 converter,
      Float64 value)
```

Include file

`io1104.h`

Purpose

To write a value to the specified D/A converter.

Description

If the converter is in latched mode, the written value is output only when `ds1104_dac_strobe` is executed.

I/O mapping

For information on the I/O mapping, refer to DAC Unit (DS1104 Features).

Parameters

converter Specifies the converter number within the range 1 … 8.

value Specifies the value to be written within the range from –1.0 … +1.0. The value is scaled as follows:

Value Range	Output Voltage Range
–1.0 … +1.0	–10 V … +10 V

Related topics

Examples

References

ds1104_dac_strobe

Syntax

```
void ds1104_dac_strobe(void)
```

Include file	`io1104.h`
Purpose	To strobe all D/A converters (in latched mode).
Description	Use `ds1104_dac_init` to set the latched mode.

Note

The converter values must be loaded with `ds1104_dac_write`, before you can use `ds1104_dac_strobe`.

Related topics

References

ds1104_dac_init 234
ds1104_dac_write 237

Incremental Encoder Interface

Where to go from here

Information in this section

Information in other sections

Basic Information on the Incremental Encoder Interface

Calculating the counter values

When you use the RTLib functions for the incremental encoder interface, you can measure or calculate the number of increments by using the line counter.

Line counter The line counter contains the number of increments. For a 4-fold subdivision, one encoder line corresponds to 4 hardware lines. The line counter can be calculated by the position value, and vice versa. This counter cannot be accessed by RTI blocks, but by RTLib functions.

The following table shows some examples for the relationship between the position values and the line counter for 4-fold subdivision:

Line Counter	Position Value
0	0/4 = 0.00
1	1/4 = 0.25
2	2/4 = 0.5
3	3/4 = 0.75
4	4/4 = 1.00
...	...
41	41/4 = 10.25
...	...

Interrupt handling

For information on how to make the encoder channel interrupts available, refer to Interrupt Handling on page 96.

Example of Using the Incremental Encoder Interface Functions

Example source code

The following example demonstrates how to use functions of the incremental encoder interface. You find the relevant files in the directory

`<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\Enc_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <Brtenv.h>
#define DT 10e-3            /* 10 ms simulation step size */
/* variables for ControlDesk */
/* position and velocity of Encoder channel 1 */
Float64 inc_pos, inc_vel;
/* flag for index found */
/* Use Int32 instead of Int16 for .trc !!*/
Int32 ind_found = 0;
Int32 ind_reset = 1;
Float64 exec_time;
void isr_srt(void)
{
   Int16 temp;
   ts_timestamp_type ts;
```

```
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);   /* data acquisition service*/
```

```
   /* set flag for index found */
   if (ind_reset)
   {
      temp = ds1104_inc_index_read(1, DS1104_INC_IDXMODE_ON);
   }
   else
   {
      temp = ds1104_inc_index_read(1, DS1104_INC_IDXMODE_OFF);
   }

   if (temp == DS1104_INC_IDX_SET)
      ind_found = temp;
   /* encoder interrupt functions for digital channel 1 */
   /* read with highest resolution, 1/4 line */
   inc_pos = ds1104_inc_position_read(1,
               DS1104_INC_LINE_SUBDIV_4);
   /* calculate the velocity in lines per second */
   inc_vel = ds1104_inc_delta_position_read(1,
               DS1104_INC_LINE_SUBDIV_4) / DT;
   exec_time = RTLIB_TIC_READ();
   RTLIB_SRT_ISR_END();   /* overload check */
}
void main(void)
{
   init();      /* DS1104 and RTLib1104 initialization*/
   /* init incremental encoder channel 1 */
   /* input signal for channel 1 via RS422 */
   ds1104_inc_init(1, DS1104_INC_MODE_RS422);
   /* set reset on index for channel 1 and latch to 0 */
   ds1104_inc_set_idxmode(1, DS1104_INC_IDXMODE_ON);
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* start sample rate timer */
   RTLIB_SRT_START(DT, isr_srt);
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();   /* background service */
   }
}
```

ds1104_inc_init

Syntax

```
void ds1104_inc_init(
      UInt16 channel,
      UInt16 inc_mode)
```

Include file `io1104.h`

Purpose

To initialize an incremental encoder channel.

Description

After initialization, you should call ds1104_inc_set_idxmode on page 243 to set the reset-on-index mode.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

inc_mode Sets the channel characteristics. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INC_MODE_TTL	Input signal TTL (channels 1, 2)
DS1104_INC_MODE_RS422	Input signal RS422 (channels 1, 2)

Example

To initialize channel 1 for TTL input signal:

```
ds1104_inc_init(1, DS1104_INC_MODE_TTL)
```

Initialization sequence for the encoder channels 1 and 2:

```
ds1104_inc_init(1, DS1104_INC_MODE_TTL);
ds1104_inc_init(2, DS1104_INC_MODE_TTL);
ds1104_inc_set_idxmode(1, DS1104_INC_IDXMODE_ON);
ds1104_inc_set_idxmode(2, DS1104_INC_IDXMODE_ON);
```

Related topics

References

ds1104_inc_set_idxmode..243

ds1104_inc_set_idxmode

Syntax

```
void ds1104_inc_set_idxmode(
     UInt16 channel,
     UInt16 idx_mode)
```

Include file

`io1104.h`

Purpose

To activate the reset-on-index mode for the specified encoder channel.

Description

If the reset-on-index mode is set, the counter of the specified channel is reset to 0 when an index signal occurs. Usually, the function is called after initializing an encoder channel with `ds1104_inc_init`. You can use the function `ds1104_inc_index_read` to activate the reset-on-index mode again.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 🕮).

Parameters

channel Specifies the encoder channel to be reset on index found.

idx_mode Specifies the reset-on-index mode. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INC_IDXMODE_ON	Reset on index
DS1104_INC_IDXMODE_OFF	No reset on index

Related topics

References

ds1104_inc_position_read

Syntax

```
Float64 ds1104_inc_position_read(
      UInt16 channel,
      Int32 line_subdiv)
```

Include file

`io1104.h`

Purpose

To get the current position of an encoder channel.

Result

The counter value is read and scaled back as lines with the resolution you set.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

line_subdiv Specifies the resolution by masking out particular bits of the counter value. The bits 0 and 1 of the resulting counter value determine the line subdivision. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INC_LINE_SUBDIV_4	No bit masking (complete resolution; 4-fold)
DS1104_INC_LINE_SUBDIV_2	Clears bit 0 (2-fold)
DS1104_INC_LINE_SUBDIV_1	Clears bits 0 and 1 (1-fold)
DS1104_INC_LINE_SUBDIV_1_2	Clears bits 0 … 2 (1/2-fold)
DS1104_INC_LINE_SUBDIV_1_4	Clears bits 0 … 3 (1/4-fold)
DS1104_INC_LINE_SUBDIV_1_8	Clears bits 0 … 4 (1/8-fold)

Return value

This function returns the position difference to the previous channel position given in lines. The line subdivision is given as a decimal place. The digital channels 1 and 2 use a 4-fold line subdivision, which is specified in multiples of 0.25.

Related topics

References

ds1104_inc_delta_position_read 247
ds1104_inc_position_write 249

ds1104_inc_position_read_immediately

Syntax

```
Float64 ds1104_inc_position_read_immediately(
     UInt16 channel,
     Int32 line_subdiv)
```

Include file

`io1104.h`

Purpose

To get the current position of an encoder channel without a preceding strobe of the output register.

Description

The counter value is read and scaled back as lines with the resolution you set. This function can be used in an interrupt service routine for the ST1PWM slave-master interrupt. The SYNCIN trigger of the incremental encoder must be enabled (see ds1104_inc_trigger_setup on page 255). During function execution, all interrupts are disabled.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

line_subdiv Specifies the resolution by masking out particular bits of the counter value. The bits 0 and 1 of the resulting counter value determine the line subdivision. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INC_LINE_SUBDIV_4	No bit masking (complete resolution; 4-fold)
DS1104_INC_LINE_SUBDIV_2	Clears bit 0 (2-fold)
DS1104_INC_LINE_SUBDIV_1	Clears bits 0 and 1 (1-fold)
DS1104_INC_LINE_SUBDIV_1_2	Clears bits 0 … 2 (1/2-fold)
DS1104_INC_LINE_SUBDIV_1_4	Clears bits 0 … 3 (1/4-fold)
DS1104_INC_LINE_SUBDIV_1_8	Clears bits 0 … 4 (1/8-fold)

Return value

This function returns the position difference to the previous channel position given in lines. The line subdivision is given as a decimal place. The digital channels 1 and 2 use a 4-fold line subdivision, which is specified in multiples of 0.25.

Related topics

References

ds1104_inc_delta_position_read

Syntax

```
Float64 ds1104_inc_delta_position_read(
      UInt16 channel,
      Int32 line_subdiv)
```

Include file

`io1104.h`

Purpose

To read the position difference of the encoder channel.

Description

The difference is calculated by subtracting the previously read position from the current position. If reset-on-index is set for the specified encoder channel (refer to ds1104_inc_set_idxmode on page 243), you have to regard the following situation: When an index has occurred before `ds1104_inc_delta_position_read` is executed, the previously read position is set to 0. This causes a deviation between the real and the calculated delta position.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 🕮).

Parameters

channel Specifies the channel number within the range 1 … 2.

line_subdiv Specifies the resolution by masking out particular bits of the counter value. The bits 0 and 1 of the resulting counter value determine the line subdivision. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INC_LINE_SUBDIV_4	No bit masking (complete resolution; 4-fold)
DS1104_INC_LINE_SUBDIV_2	Clears bit 0 (2-fold)
DS1104_INC_LINE_SUBDIV_1	Clears bits 0 and 1 (1-fold)
DS1104_INC_LINE_SUBDIV_1_2	Clears bits 0 … 2 (1/2-fold)
DS1104_INC_LINE_SUBDIV_1_4	Clears bits 0 … 3 (1/4-fold)
DS1104_INC_LINE_SUBDIV_1_8	Clears bits 0 … 4 (1/8-fold)

Return value

This function returns the position difference to the previous channel position given in lines. The line subdivision is given as a decimal place. The digital channels 1 and 2 use a 4-fold line subdivision, which is specified in multiples of 0.25.

Related topics

References

ds1104_inc_delta_position_read_immediately

Syntax

```
Float64 ds1104_inc_delta_position_read_immediately(
      UInt16 channel,
      Int32 line_subdiv)
```

Include file

`io1104.h`

Purpose

To read the position difference of the encoder channel without a preceding strobe of the output register.

Description

The difference is calculated by subtracting the previously read position from the current position. If reset-on-index is set for the specified encoder channel (refer to ds1104_inc_set_idxmode on page 243), you have to regard the following situation: When an index has occurred before `ds1104_inc_delta_position_read_immediately` is executed, the previously read position is set to 0. This causes a deviation between the real and the calculated delta position.

This function can be used in an interrupt service routine for the ST1PWM slave-master interrupt. The SYNCIN trigger of the incremental encoder must be enabled. During function execution, all interrupts are disabled.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

line_subdiv Specifies the resolution by masking out particular bits of the counter value. The bits 0 and 1 of the resulting counter value determine the line subdivision. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INC_LINE_SUBDIV_4	No bit masking (complete resolution; 4-fold)

Predefined Symbol	Meaning
DS1104_INC_LINE_SUBDIV_2	Clears bit 0 (2-fold)
DS1104_INC_LINE_SUBDIV_1	Clears bits 0 and 1 (1-fold)
DS1104_INC_LINE_SUBDIV_1_2	Clears bits 0 … 2 (1/2-fold)
DS1104_INC_LINE_SUBDIV_1_4	Clears bits 0 … 3 (1/4-fold)
DS1104_INC_LINE_SUBDIV_1_8	Clears bits 0 … 4 (1/8-fold)

Return value

This function returns the position difference to the previous channel position given in lines. The line subdivision is given as a decimal place. The digital channels 1 and 2 use a 4-fold line subdivision, which is specified in multiples of 0.25.

Related topics

References

ds1104_inc_delta_position_read........247
ds1104_inc_position_read........244
ds1104_inc_set_idxmode........243

ds1104_inc_position_write

Syntax

```
void ds1104_inc_position_write(
      UInt16 channel,
      Float64 position,
      Int32 line_subdiv)
```

Include file

`io1104.h`

Purpose

To set the position of an encoder channel.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

position Specifies the position given in lines including the line subdivision as decimal place. The digital channels 1 and 2 use a 4-fold line subdivision, which is specified in multiples of 0.25.

line_subdiv Specifies the resolution by masking out particular bits of the counter value. The bits 0 and 1 of the resulting counter value determine the line subdivision. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INC_LINE_SUBDIV_4	No bit masking (complete resolution; 4-fold)
DS1104_INC_LINE_SUBDIV_2	Clears bit 0 (2-fold)
DS1104_INC_LINE_SUBDIV_1	Clears bits 0 and 1 (1-fold)
DS1104_INC_LINE_SUBDIV_1_2	Clears bits 0 … 2 (1/2-fold)
DS1104_INC_LINE_SUBDIV_1_4	Clears bits 0 … 3 (1/4-fold)
DS1104_INC_LINE_SUBDIV_1_8	Clears bits 0 … 4 (1/8-fold)

Example

- To set channel 1 to position 10.25 with complete resolution (no bit masking):
  ```
  ds1104_inc_position_write(1, 10.25, DS1104_INC_LINE_SUBDIV_4);
  ```
  The function calculates the counter value as follows: 10.25 x 4 = 41.
- To set channel 1 to position 10.25 and masks out the bits 0 … 4:
  ```
  ds1104_inc_position_write(1, 10.25, DS1104_INC_LINE_SUBDIV_1_8);
  ```
  The function calculates the counter value as follows: 10.25 x 4 = 41, which is equivalent to binary 101001. Masking the bits 0 … 4 results in binary 100000, that is, the resulting counter value is 32.

Related topics

References

ds1104_inc_position_read 244

ds1104_inc_counter_read

Syntax

```
Int32 ds1104_inc_counter_read(UInt16 channel)
```

Include file

`io1104.h`

Purpose

To read the counter value of an encoder channel.

Description

For information how to calculate the counter values, refer to Incremental Encoder Interface on page 239.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

Return value

This function returns the position of the counter value in 0.25 lines.

Related topics

References

ds1104_inc_counter_read_immediately

Syntax

```
Int32 ds1104_inc_counter_read_immediately(UInt16 channel)
```

Include file

`io1104.h`

Purpose

To read the counter value of an encoder channel without a preceding strobe of the output register.

Description

For information how to calculate the counter values, refer to Incremental Encoder Interface on page 239.

This function can be used in an interrupt service routine for the ST1PWM slave-master interrupt. The SYNCIN trigger of the incremental encoder must be enabled. During function execution, all interrupts are disabled.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

Return value

This function returns the position of the counter value in 0.25 lines.

Related topics

References

ds1104_inc_counter_clear

Syntax

```
void ds1104_inc_counter_clear(UInt16 channel)
```

Include file

`io1104.h`

Purpose

To clear an encoder channel counter.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

Related topics

References

ds1104_inc_counter_write

Syntax

```
void ds1104_inc_counter_write(
      UInt16 channel,
      Int32 count)
```

Include file

`io1104.h`

Purpose

To set an encoder channel counter.

Description

Calculating the counter values on page 240 explains the dependencies between counter values and position values.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

count Specifies the counter value.

Example

To set the channel 1 counter to 41:

```
ds1104_inc_counter_write(1, 41);
```

Setting the counter value to 41 is equivalent to setting the position to 41 / 4 = 10.25 without bit masking. See the example in ds1104_inc_position_write on page 249.

Related topics

References

ds1104_inc_index_read

Syntax

```
Int16 ds1104_inc_index_read(
      UInt16 channel,
      UInt16 reset_enable)
```

Include file

`io1104.h`

Purpose

To check for an index.

Description

If an index signal occurs, the corresponding channel bit in the setup register is set. The function determines whether an index input of the specified encoder channel occurred. After generating the return value, the index bit is reset.

Note

You can use the encoder channel index interrupts to handle the index without time delay. To make the interrupts available, refer to Interrupt Handling on page 96.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

reset_enable Specifies the recurring counter resets on index signals can be prevented by setting reset_enable to 0. If you set reset_enable to 1, each recurring index signal causes a reset. There are the following predefined symbols:

Predefined Symbol	Meaning
DS1104_INC_IDXMODE_OFF	No reset on index
DS1104_INC_IDXMODE_ON	Reset on index

Note

It is important, when initializing with ds1104_inc_set_idxmode on page 243 that the reset-on-index is activated.

Return value

The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_INC_IDX_SET	If the channel bit in the interrupt register has been set
DS1104_INC_IDX_NOT_SET	Otherwise

Related topics

References

ds1104_inc_trigger_setup

Syntax

```
void ds1104_inc_trigger_setup(
     UInt16 channel,
     UInt16 state)
```

Include file

`io1104.h`

Purpose

To enable or disable triggering of the specified encoder channel via synchronous trigger (SyncIn).

Description

Incremental Encoder channels that are enabled for triggering can be triggered via software, by a slave PWM signal or an external trigger. For further information, refer to Synchronous I/O Trigger on page 206.

I/O mapping

For information on the I/O mapping, refer to Incremental Encoder Interface (DS1104 Features 📖).

Parameters

channel Specifies the channel number within the range 1 … 2.

state Specifies the state of the external trigger. The following symbols are predefined:

Predefined Symbol	Meaning
DS1104_TRIGGER_DISABLE	Disables the external trigger
DS1104_TRIGGER_ENABLE	Enables the external trigger

Related topics

References

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

Information in other sections

Serial Interface (DS1104 Features 📖)
The board contains a universal asynchronous receiver and transmitter (UART) to communicate with external devices.

Serial Interface (DS1104 RTI Reference 📖)

Basic Principles of Serial Communication

Where to go from here

Information in this section

Information in other sections

Serial Interface (DS1104 Features 📖)
The board contains a universal asynchronous receiver and transmitter (UART) to communicate with external devices.

Trigger Levels

Introduction

Two different trigger levels can be configured.

UART trigger level

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

User trigger level

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

Related topics

Basics

HowTos

How to Handle Subinterrupts in Serial Communication

Introduction

The interrupt functions must be used only in handcoded applications. Using them in Simulink applications (user code or S-functions) conflicts with the internal interrupt handling.

The following subinterrupts can be passed to your application:

Subinterrupt	Meaning
`DSSER_TRIGGER_LEVEL_SUBINT`	Generated when the receive buffer is filled with the number of bytes specified as the trigger level (see Trigger Levels on page 257).

Subinterrupt	Meaning
DSSER_TX_FIFO_EMPTY_SUBINT	Generated when the transmit buffer has no data.
DSSER_RECEIVER_LINE_SUBINT	Line status interrupt provided by the UART.
DSSER_MODEM_STATE_SUBINT	Modem status interrupt provided by the UART.
DSSER_NO_SUBINT	Generated after the last subinterrupt. This subinterrupt tells your application that no further subinterrupts were generated.

Method

To install a subinterrupt handler within your application

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

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

2 Initialize your subinterrupt handler:

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

3 Enable the required subinterrupts:

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

Related topics

Basics

References

Example of a Serial Interface Communication

Example

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

```
#include <brtenv.h>
void callback(dsserChannel* serCh, UInt32 subint)
{
   UInt32 count;
   UInt8 data[32];
   switch (subint)
   {
      case DSSER_TRIGGER_LEVEL_SUBINT:
         msg_info_set(0,0,"DSSER_TRIGGER_LEVEL_SUBINT");
         dsser_receive(serCh,32,data,&count);
         dsser_transmit(serCh,count,data,&count);
         break;
      case DSSER_TX_FIFO_EMPTY_SUBINT:
         msg_info_set(0,0,"DSSER_TX_FIFO_EMPTY_SUBINT");
         break;
      default:
         break;
   }
}
main()
{
   dsserChannel* serCh;
   init();

/* 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

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..284

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 284

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_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 257 or dsser_subint_handler_inst on page 287).

Members

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

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 257).

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........257

References

dsser_init........268

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 260.

lineStatusReg Line status register. Refer to `dsser_LSR` on page 262.

modemStatusReg Modem status register. Refer to `dsser_MSR` on page 263.

Related topics

References

dsser_status_read 284

Generic Serial Interface Communication Functions

Where to go from here

Information in this section

dsser_init

Syntax

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

Include file

`dsser.h`

Purpose

To initialize the serial interface and install the interrupt handler.

Note

Pay attention to the initialization sequence. First, initialize the processor board, then the I/O boards, and then the serial interface.

Parameters

base Specifies the base address of the serial interface. This value has to be set to 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 `channel` parameter is out of range.
700	Error	x, ch=y, Buffersize: Illegal	The `fifo_size` parameter is out of range.

Related topics

Basics

Basic Principles of Serial Communication........256

Examples

Example of a Serial Interface Communication........259

References

Data Types for Serial Communication........260
dsser_config........270
dsser_free........269

dsser_free

Syntax

```
Int32 dsser_free(dsserChannel*serCh)
```

Include file

`dsser.h`

Purpose

To close a serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

Return value

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

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation. The specified serial interface is closed. Its memory for the buffer is freed and the interrupts are released. A serial interface can be created again using the `dsser_init` function.
DSSER_TX_FIFO_NOT_EMPTY	The serial interface is not closed, because the transmit buffer is not empty.
DSSER_CHANNEL_INIT_ERROR	There is no serial interface to be closed (serCh == NULL).

Related topics

Basics

Basic Principles of Serial Communication....................256

References

dsser_init....................268

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 268).

fifo_mode Specifies the mode of the receive buffer (see Serial Interface (DS1104 Features 📖)):

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

baudrate Specifies the baud rate in bits per second:

Mode	Baud Rate Range
RS232	300 … 115,200 baud
RS422	300 … 1,000,000 baud
RS485	300 … 1,000,000 baud

For further information, refer to Specifying the Baud Rate of the Serial Interface (DS1104 Features 📖).

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

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

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

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

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

uart_trigger_level Sets the UART trigger level (see Trigger Levels on page 257). 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 257):

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)
`DSSER_RS422`	RS422 mode
`DSSER_RS485`	RS485 mode

Messages

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

ID	Type	Message	Description
101	Warning	x, ch=y, Mixed usage of high and low level API!	It is not allowed to use the generic functions (high-level access functions) and the low-level access functions of the serial interface on the same channel. It is recommended to use only the generic functions.
601	Error	x, serCh: The UART channel was not initialized.	The `dsser_config` function was called before the serial interface was initialized with `dsser_init`.
602	Error	x, ch=y, baudrate: Illegal!	The `baudrate` parameter is out of range.

ID	Type	Message	Description
603	Error	x, ch=y, databits: Use range 5 … 8 bits!	The `databits` parameter is out of range.
604	Error	x, ch=y, stopbits: Illegal number (1-2 bits allowed)!	The `stopbits` parameter is out of range.
605	Error	x, ch=y, parity: Illegal parity!	The `parity` parameter is out of range.
606	Error	x, ch=y, trigger_level: Illegal UART trigger level!	The `uart_trigger_level` parameter is out of range.
607	Error	x, ch=y, trigger_level: Illegal user trigger level!	The `user_trigger_level` parameter is out of range.
608	Error	x, ch=y, fifo_mode: Use range 0 … (fifo_size-1) bytes!	The `uart_mode` parameter is out of range.
609	Error	x, ch=y, uart_mode: Transceiver not supported!	The selected UART mode does not exist for this serial interface.
611	Error	x, ch=y, uart_mode: Autoflow is not supported!	Autoflow does not exist for this serial interface.

Related topics

Basics

Basic Principles of Serial Communication....256

Examples

Example of a Serial Interface Communication....259

References

dsser_init....268

dsser_transmit

Syntax

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

Include file

`dsser.h`

Purpose

To transmit data through the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

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

Examples

References

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 268).

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

data Specifies the pointer to the destination buffer.

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

Return value

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

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

Example

The following example shows how to receive 4 bytes.

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

Related topics

Basics

Basic Principles of Serial Communication....256

Examples

Example of a Serial Interface Communication....259

References

dsser_init....268

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 268).

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

data Specifies the pointer to the destination buffer.

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

term Specifies the character that terminates the reception of bytes.

Return value

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

Predefined Symbol	Meaning
`DSSER_NO_ERROR`	No error occurred during the operation.
`DSSER_NO_DATA`	No new data is read from the FIFO.
`DSSER_FIFO_OVERFLOW`	The FIFO is filled. The behavior depends on the `fifo_mode` adjusted with `dsser_config`: ▪ `fifo_mode = DSSER_FIFO_MODE_BLOCKED` Not all new data could be placed in the FIFO. ▪ `fifo_mode = DSSER_FIFO_MODE_OVERWRITE` The old data is rejected.

Predefined Symbol	Meaning
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....................256

References

dsser_init....................268

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 268).

Return value

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

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

Related topics

Basics

References

dsser_enable

Syntax

```
Int32 dsser_enable(const dsserChannel* serCh)
```

Include file

`dsser.h`

Purpose

To enable the serial interface.

Description

The UART interrupt is enabled, the serial interface starts transmitting and receiving data.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

Return value

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

Predefined Symbol	Meaning
`DSSER_NO_ERROR`	No error occurred during the operation.
`DSSER_COMMUNICATION_FAILED`	The function failed. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics

Basics

References

dsser_disable

Syntax

```
Int32 dsser_disable(const dsserChannel* serCh)
```

Include file

`dsser.h`

Purpose

To disable the serial interface.

Description

The serial interface stops transmitting data, incoming data is no longer stored in the receive buffer and the UART subinterrupts are disabled.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

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

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

Related topics

Basics

References

dsser_error_read

Syntax

```
Int32 dsser_error_read(const dsserChannel* serCh)
```

Include file `dsser.h`

Purpose To read an error flag of the serial interface.

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

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

Return value

This function returns an error flag.

The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error flag set
DSSER_FIFO_OVERFLOW	Too many bytes for the buffer
DSSER_SLAVE_DATA_LOST	Data was lost
DSSER_SLAVE_FIFO_OVERFLOW	Overflow of the software FIFO
DSSER_SLAVE_INIT_ACK	No error, it is an acknowledge code from the slave
DSSER_SLAVE_ALLOC_ERROR	Memory allocation on the slave failed
DSSER_SLAVE_BUFFER_OVERFLOW	Buffer overflow of the communication queue between master and slave
DSSER_SLAVE_UNDEF_ERROR	Undefined error

Related topics

Basics

Basic Principles of Serial Communication........256

References

dsser_config........270
dsser_init........268

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 268).

Return value

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

Related topics

Basics

References

dsser_receive_fifo_level

Syntax

```
Int32 dsser_receive_fifo_level(const dsserChannel* serCh)
```

Include file

`dsser.h`

Purpose

To get the number of bytes in the receive buffer.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

Return value

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

Related topics

Basics

References

dsser_status_read

Syntax

```
Int32 dsser_status_read(
      dsserChannel*serCh,
      const UInt8 register_type)
```

Include file

`dsser.h`

Purpose

To read the value of one or more status registers and to store the values in the appropriate fields of the channel structure.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

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

Predefined Symbol	Meaning
DSSER_STATUS_IIR_FCR	Interrupt status register, see `dsser_ISR` data type.
DSSER_STATUS_LSR	Line status register, see `dsser_ISR` data type.
DSSER_STATUS_MSR	Modem status register, see `dsser_ISR` data type.

Return value

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

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

Example

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

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

Related topics

Basics

References

dsser_handle_get

Syntax

```
dsserChannel* dsser_handle_get(
    UInt32 base,
    UInt32 channel)
```

Include file

`dsser.h`

Purpose

To check whether the serial interface is in use.

Parameters

base Specifies the base address of the serial interface. This value has to be set to 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

References

dsser_set

Syntax

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

Include file

`dsser.h`

Purpose

To set a property of the UART.

Description

The DS1104 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 268).

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

References

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 288).

> **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 268).

subint_handler Specifies the pointer to the subinterrupt handler.

Return value

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

Related topics

Basics

Examples

References

dsser_subint_enable

Syntax

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

Include file

`dsser.h`

Purpose

To enable one or several subinterrupts of the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

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 257)
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

Examples

References

dsser_subint_disable

Syntax

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

Include file

`dsser.h`

Purpose

To disable one or several subinterrupts of the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see dsser_init on page 268).

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 257)
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

References

dsser_word2bytes

Syntax

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

Include file

dsser.h

Purpose

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

Parameters

word Specifies the pointer to the input word.

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

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

Return value

This function returns the pointer to a byte array.

Example

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

```
void word_transmit(dsserChannel* serCh, UInt32* word, UInt32* count)
{
   UInt8    bytes[4];
   UInt8*   data_p;
   if(dsser_transmit_fifo_level(serCh) < serCh->fifo_size - 4)
   {
      data_p = dsser_word2bytes(word, bytes, 4);
      dsser_transmit(serCh, 4, data_p, count);
   }
   else
   {
      *count = 0;
   }
}
```

Use of the function:

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

Related topics

Basics

Basic Principles of Serial Communication....256

References

dsser_bytes2word....292
dsser_transmit....273
dsser_transmit_fifo_level....282

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

References

Special Processor Functions

Purpose

To ensure proper operation of the PowerPC.

Where to go from here

Information in this section

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 PowerPC executes I/O accesses in the right order. For example, when two I/O accesses are performed sequentially, the PowerPC 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....295
Standard Macros....296

RTLIB_SYNC

Syntax	`void RTLIB_SYNC(void)`
Include file	`dsstd.h`
Purpose	To force the PowerPC to perform all pending memory accesses.
Description	This macro ensures that the PowerPC performs all memory accesses that were issued before the macro was called.
Return value	None
Related topics	References

RTLIB_FORCE_IN_ORDER 294
Standard Macros 296

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 ...
`init`	init() on page 298

End of application

The include file `dsstd.h` defines a macro, which you can use to stop the application immediately.

Macro	Refer to ...
`RTLIB_EXIT`	RTLIB_EXIT on page 298

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 ...
`RTLIB_GET_SERIAL_NUMBER`	RTLIB_GET_SERIAL_NUMBER() on page 299

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 ...
`RTLIB_BACKGROUND_SERVICE`	RTLIB_BACKGROUND_SERVICE on page 25
`rtlib_background_hook`	rtlib_background_hook on page 26
`rtlib_background_hook_process`	rtlib_background_hook_process on page 28

Interrupt handling

The include file `dsstd.h` defines macros, which you can use to enable or disable the interrupts globally.

Macro	Refer to ...
`RTLIB_INT_ENABLE`	DS1104_GLOBAL_INTERRUPT_ENABLE() on page 109
`RTLIB_INT_DISABLE`	DS1104_GLOBAL_INTERRUPT_DISABLE() on page 109

Sample rate timer

The DS1104 PPC Board uses the Timer 0 as the sample rate timer. The include file `dsstd.h` defines macros to handle this sample rate timer:

Macro	Refer to ...
`RTLIB_SRT_START`	ds1104_start_isr_timer0 on page 83
`RTLIB_SRT_ISR_BEGIN`	ds1104_begin_isr_timer0 on page 88
`RTLIB_SRT_ISR_END`	ds1104_end_isr_timer0 on page 89
`RTLIB_SRT_ENABLE`	ds1104_enable_hardware_int on page 103
`RTLIB_SRT_DISABLE`	ds1104_disable_hardware_int on page 104

Time interval measurement

There are macros to be used for time interval measurement.

- **`RTLIB_TIC_START`** on page 47
- **`RTLIB_TIC_READ`** on page 45
- **`RTLIB_TIC_READ_TOTAL`** on page 46
- **`RTLIB_TIC_HALT`** on page 44
- **`RTLIB_TIC_CONTINUE`** on page 40
- **`RTLIB_TIC_DELAY`** on page 42
- **`RTLIB_TIC_COUNT`** on page 41
- **`RTLIB_TIC_DIFF`** on page 42
- **`RTLIB_TIC_ELAPSED`** on page 43

Memory allocation

The include file `dsstd.h` defines macros to handle memory allocation that is protected against interrupt activities.

Macros	Refer to ...
`RTLIB_MALLOC_PROT`	RTLIB_MALLOC_PROT on page 299
`RTLIB_CALLOC_PROT`	RTLIB_CALLOC_PROT on page 300
`RTLIB_REALLOC_PROT`	RTLIB_REALLOC_PROT on page 300
`RTLIB_FREE_PROT`	RTLIB_FREE_PROT on page 301

PowerPC functions

The include file `dsstd.h` defines macros to handle the following Assembler commands.

Macros	Refer to ...
`RTLIB_FORCE_IN_ORDER`	RTLIB_FORCE_IN_ORDER on page 294
`RTLIB_SYNC`	RTLIB_SYNC on page 295

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

- 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 300
RTLIB_FREE_PROT 301
RTLIB_REALLOC_PROT 300

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_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 300
RTLIB_FREE_PROT 301
RTLIB_MALLOC_PROT 299

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 300
RTLIB_MALLOC_PROT 299
RTLIB_REALLOC_PROT 300

Function Execution Times

Introduction

The execution times of the C functions can vary, since they depend on different factors. The measured execution times are influenced by the test environment used. This section gives you basic information on the test environment and contains the mean function execution times.

Where to go from here

Information in this section

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 `downxxxx` 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 used DS1104 Controller Board are:

CPU clock	250 MHz
Bus clock	100 MHz
Global RAM size	32 MB

Related topics

References

Measured Execution Times 303

Measured Execution Times

ADC unit

The following execution times have been measured for the functions of the ADC unit:

Function	Execution Time (in µs)
ds1104_adc_start	0.08
ds1104_adc_delayed_start	1.35
ds1104_adc_mux	0.08
ds1104_adc_read_mux	3.58 (1 channel) 5.85 (2 channels) 8.05 (3 channels) 10.2 (4 channels)
ds1104_adc_read_conv	0.71
ds1104_adc_read_conv_immediately	0.43
ds1104_adc_read_ch	0.71
ds1104_adc_read_ch_immediately	0.4
ds1104_adc_read_all	2.35
ds1104_adc_trigger_setup	0.92

Note

The execution times of the read functions do not include the conversion time.

DAC unit

The following execution times have been measured for the functions of the DAC unit:

Function	Execution Time (in µs)
ds1104_dac_init	1.39
ds1104_dac_reset	0.84
ds1104_dac_write	0.2

Function	Execution Time (in µs)
ds1104_dac_strobe	0.08
ds1104_dac_trigger_setup	0.92

Incremental Encoder Interface

The following execution times have been measured for the functions of the Incremental Encoder Interface:

Function	Execution Time (in µs)
ds1104_inc_init	1.44
ds1104_inc_set_idxmode	0.88
ds1104_inc_position_read	0.76
ds1104_inc_position_read_immediately	0.4
ds1104_inc_delta_position_read	1.73
ds1104_inc_delta_position_read_immediately	1.28
ds1104_inc_position_write	0.72
ds1104_inc_counter_read	0.64
ds1104_inc_counter_read_immediately	0.36
ds1104_inc_counter_clear	0.64
ds1104_inc_counter_write	0.64
ds1104_inc_index_read	1.8
ds1104_inc_trigger_setup	0.92

Digital I/O

The following execution times have been measured for the functions of the Digital I/O:

Function	Execution Time (in µs)
ds1104_bit_io_init	0.08
ds1104_bit_io_init_with_preset	0.12
ds1104_bit_io_write	0.08
ds1104_bit_io_read	0.32
ds1104_bit_io_set	0.84
ds1104_bit_io_clear	0.84

Trigger functions

The following execution times have been measured for the functions of the external I/O trigger:

Function	Execution Time (in µs)
ds1104_syncin_edge_setup	0.88
ds1104_syncout_edge_setup	0.88

Function	Execution Time (in μs)
ds1104_syncin_trigger	0.08
ds1104_syncout_trigger	0.08
ds1104_external_trigger_enable	1.06 ms (without slave communication)

Related topics

Basics

Information on the Test Environment...302

Slave DSP Access Functions

Introduction

This section comprises the master PowerPC functions for slave control and the functions you need to access the features served by the slave DSP.

Where to go from here

Information in this section

Basic Communication Principles

Basic Principles of Master-Slave Communication

Introduction

The master PPC controls, with the help of the slave access functions, the actions of the slave DSP and exchanges data with the slave interface.

Note

You have to initialize the communication between master and slaves. For initializing the PPC to slave DSP communication, see ds1104_slave_dsp_communication_init on page 310.

Communication Process

- The master PPC application initializes the necessary slave function or a group of slave functions based on a particular module, for example serial interface. Whether or not initialization is necessary, depends on the slave application and the I/O interface (e.g., ADC unit, Bit I/O unit) used.
- The master PPC registers the slave function and with it the parameters in the command table. The function can then be identified by the command table index, which is returned when registering the function.
- To perform a read operation, the master PPC requests the slave function previously registered to be carried out. The slave then performs the required functions independently and writes the results back into the dual-ported memory. If more than one function is required simultaneously – for example as a result of different tasks on the PPC – priorities must be considered.
- The master PPC application reads/writes the input/output data from/to the slave. The read/write functions can also carry out format conversions and scaling if necessary.

Note

It is important to remember that the master PPC reads the slave results from the dual-ported memory in the order in which they occur, and then reads them into a buffer, regardless whether a particular result may or may not be needed. The read functions are the ones which copy data results from the buffer into the PPC application variables.

Function classes

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

- *Initialization functions* initialize the slave functions.
- *Register functions* make the slave functions known to the slave.

- *Request functions* require the slave function previously registered to be carried out by the slave.
- *Read functions* fetch data from the dual-ported memory, and convert or scale the data, if necessary.
- *Write functions* convert or scale the data if necessary and write them into the dual-ported memory.

Error handling

When an error occurs with initialization or register functions, an error message appears from the global message module. Then the program ends. Request, read, and write functions return an error code. The application can then deal with the error code.

Communication channels and priorities

This communication method along with the command table and the transfer buffer can be initialized in parallel for three statically defined communication channels with fixed priorities (0 … 2). As well as communication buffers, each communication channel has access to memory space in the dual-ported memory so that slave error codes can be transferred.

Overall Slave DSP Access Functions

Where to go from here

Information in this section

Information in other sections

DS1104 Features
Provides the feature information you need to implement your real-time models on your dSPACE hardware.

ds1104_slave_dsp_communication_init

Syntax

```
void ds1104_slave_dsp_communication_init(void)
```

Include file

`slvdsp1104.h`

Purpose	To initialize the communication between master PPC and slave DSP.

Description

This function also initializes three communication channels with fixed task_ids (0 … 2) for the master-slave communication, and starts the slave DSP. The communication channel with task_id = 0 has the highest priority. This function also starts a version check that compares the version of the dSPACE firmware with the one that is expected by the current RTLib.

> **Note**
>
> This initialization function must be performed at the beginning of every application and every S-function accessing the slave DSP features.
>
> Regarding S-functions, you need not take care of multiple calls. Even if this function is called more than one time within a model, it is executed only once.

Return value

None

ds1104_slave_dsp_error_read

Syntax

```
void ds1104_slave_dsp_error_read(
      Int16 channel,
      UInt32 *slave_error)
```

Include file

`slvdsp1104.h`

Purpose

To read the current slave DSP error code concerning the specified communication channel from the dual-ported memory.

Description

The error codes deal with those communication errors that occur to the slave DSP part of communication. You can use this function to monitor the slave DSP error state continuously in the background task of your application.

Parameters

channel Specifies the communication channel within the range 0 … 2.

slave_error Specifies the address where the slave DSP error code is written. The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_SLV_ALLOC_ERROR	The allocation of the slave DSP dynamic memory has failed. This could lead to an application abort.
SLVDSP1104_SLV_BUFFER_ OVERFLOW	The communication buffer from slave DSP to master PPC has overflown. This could lead to an application abort.

Return value

None

ds1104_slave_dsp_int_init

Syntax

```
void ds1104_slave_dsp_int_init(isr_function_name)
```

Include file

`slvdsp1104.h`

Purpose

To initialize and enable the slave DSP interrupt.

Description

This macro initializes the slave DSP interrupt `DS1104_INT_SLAVE_DSP` by means of ds1104_set_interrupt_vector on page 97. Then the interrupt will be enabled via `ds1104_enable_hardware_int` and `DS1104_GLOBAL_INTERRUPT_ENABLE()`.

Parameters

isr_function_name Specifies the name of the interrupt service routine.

Return value

None

Related topics

References

ds1104_slave_dsp_firmware_rev_read

Syntax

```
Int16 ds1104_slave_dsp_firmware_rev_read(
      Int16 channel,
      UInt32 *revision)
```

Include file

`slvdsp1104.h`

Purpose

To read the current slave DSP firmware revision concerning the specified communication channel.

Parameters

channel Specifies the communication channel within the range 0 … 2.

revision Specifies the address where the slave DSP firmware revision of the specified communication channel is written.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_ALLOC_ERROR	The allocation of the DSMCOM dynamic memory has failed.
SLVDSP1104_ILLEGAL_TASK_ID	The task id (channel) is out of range.
SLVDSP1104_ILLEGAL_INDEX	The command index is out of range.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer of the DSMCOM is full.
SLVDSP1104_NO_DATA	No data available.
SLVDSP1104_DATA_LOST	Return data lost.
SLVDSP1104_TIMEOUT	Slave DSP is not responding.
SLVDSP1104_SLV_ALLOC_ERROR	The allocation of the slave DSP dynamic memory has failed. This could lead to an application abort.

Predefined Symbol	Meaning
SLVDSP1104_SLV_BUFFER_ OVERFLOW	The communication buffer from slave DSP to master PPC has overflown. This could lead to an application abort.
SLVDSP1104_SLV_UNDEFINED	The error flag of the slave DSP is undefined.
SLVDSP1104_SLV_ILLEGAL_USR_IDX	The user index of the slave DSP is out of range.

Related topics

Basics

Basic Communication Principles..308

ds1104_slave_dsp_reset

Syntax

```
void ds1104_slave_dsp_reset(void)
```

Include file

`slvdsp1104.h`

Purpose

To reset the slave DSP.

Return value

None

Related topics

Basics

Basic Communication Principles..308

References

ds1104_slave_dsp_start..314

ds1104_slave_dsp_start

Syntax

```
void ds1104_slave_dsp_start(void)
```

Include file	`slvdsp1104.h`
Purpose	To start the slave DSP.
Return value	None
Related topics	Basics

References

ds1104_slave_dsp_ram_boot

Syntax	`void ds1104_slave_dsp_ram_boot(void)`
Include file	`slvdsp1104.h`
Purpose	To run the slave DSP in microprocessor mode from external RAM.
Return value	None
Related topics	Basics

References

ds1104_slave_dsp_flash_boot

Syntax	`void ds1104_slave_dsp_flash_boot(void)`
Include file	`slvdsp1104.h`
Purpose	To run the slave DSP in micro computer mode from internal flash.
Return value	None

Related topics

Basics

Basic Communication Principles..308

References

ds1104_slave_dsp_ram_boot..315

ds1104_slave_dsp_appl_load

Syntax	`void ds1104_slave_dsp_appl_load(Int32 *appl_addr)`
Include file	`slvdsp1104.h`
Purpose	To load a slave DSP application.

Description

This function loads the specified slave DSP application into the program memory of the slave DSP. After the slave's boot sequence the slave starts in microprocessor mode from external RAM.

> **Note**
>
> The slave DSP boot sequence takes some milliseconds, if the overall RAM is used for the application.

Parameters	**appl_addr**	Specifies the address of the slave DSP application.

Return value

None

Related topics

Basics

Basic Communication Principles........308

References

ds1104_slave_dsp_ram_boot........315

Slave DSP Bit I/O Unit

Where to go from here

Information in this section

Information in other sections

Example of Using the Bit I/O Functions of the Slave DSP

Example source code

The following example demonstrates how to use bit I/O functions of the slave DSP. You find the relevant files in the directory `<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\SlaveDSP\Slv_BitIo_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <Brtenv.h>
#define DT 1.0e-3               /* 1 ms simulation step size */
/* variables for ControlDesk*/
volatile UInt32 writevalue = 0x01;   /* set bits for output */
UInt32 readvalue;
Float64 exec_time;                     /* execution time */
UInt32 writestate[6];
UInt32 readstate[4];
/* variables for communication with Slave DSP */
Int16 index = -1;                    /* command table index */
Int16 index1 = -1;
Int16 task_id = 0;                   /* communication channel */
void isr_srt(void)
{
   UInt8 temp;
   Int16 slave_err_read;
   Int bitpos;
   ts_timestamp_type ts;
```

```
   RTLIB_SRT_ISR_BEGIN();          /* overload check */
   RTLIB_TIC_START();              /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);   /* data acquisition service*/
```

```
   /* update writevalues from the writestate array */
   for (bitpos = 0; bitpos < 6; bitpos++)
   {
      if (writestate[bitpos])
         writevalue |= 0x1 << bitpos;
      else
         writevalue &= ~(0x1 << bitpos);
   }
   /* write writevalue to the specified I/O group */
   ds1104_slave_dsp_bit_io_write(task_id, index, writevalue );
   ds1104_slave_dsp_bit_io_read_request(task_id, index1);
   /* read bitmap from the specified I/O group */
   do
   {
      slave_err_read = ds1104_slave_dsp_bit_io_read(task_id,
         index1, &temp );
   }
   while(slave_err_read == SLVDSP1104_NO_DATA);
```

```
   readvalue = temp;
   /* update the readstate array from readvalue */
   for (bitpos = 0M bitpos < 4; bitpos++)
   {
      if (readvalue & (0x1 << (bitpos + 4)))
         readstate[bitpos] = 1;
      else
         readstate[bitpos] = 0;
   }
   exec_time = RTLIB_TIC_READ();
   RTLIB_SRT_ISR_END();
}
void main(void)
{
   /* DS1104 and RTLib1104 initialization */
   init();
   /* init communication with slave DSP*/
   ds1104_slave_dsp_communication_init();
   /* Initialize whole group 2 for Bit Out */
   ds1104_slave_dsp_bit_io_init(task_id, 2,
     SLVDSP1104_BIT_IO_BIT0_MSK | SLVDSP1104_BIT_IO_BIT1_MSK |
     SLVDSP1104_BIT_IO_BIT2_MSK | SLVDSP1104_BIT_IO_BIT3_MSK |
     SLVDSP1104_BIT_IO_BIT4_MSK | SLVDSP1104_BIT_IO_BIT5_MSK,
     SLVDSP1104_BIT_IO_BIT0_OUT | SLVDSP1104_BIT_IO_BIT1_OUT |
     SLVDSP1104_BIT_IO_BIT2_OUT | SLVDSP1104_BIT_IO_BIT3_OUT |
     SLVDSP1104_BIT_IO_BIT4_OUT | SLVDSP1104_BIT_IO_BIT5_OUT);
   /* register write function in the command table */
   ds1104_slave_dsp_bit_io_write_register(task_id,
      &index, 2);
   /* Initialize Bit 4, 5, 6, 7 group 3 for Bit In */
   ds1104_slave_dsp_bit_io_init(task_id, 3,
     SLVDSP1104_BIT_IO_BIT4_MSK | SLVDSP1104_BIT_IO_BIT5_MSK |
     SLVDSP1104_BIT_IO_BIT6_MSK | SLVDSP1104_BIT_IO_BIT7_MSK,
     SLVDSP1104_BIT_IO_BIT4_IN | SLVDSP1104_BIT_IO_BIT5_IN |
     SLVDSP1104_BIT_IO_BIT6_IN | SLVDSP1104_BIT_IO_BIT7_IN );
   /* register read function in the command table
      for group 1 */
   ds1104_slave_dsp_bit_io_read_register(task_id,
      &index1, 3);
   /* periodic event in ISR */
   RTLIB_SRT_START(DT, isr_srt);
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();   /* background service */
   }
}
```

Related topics

Basics

Slave DSP Bit I/O Unit (DS1104 Features 📖)

ds1104_slave_dsp_bit_io_init

Syntax

```
void ds1104_slave_dsp_bit_io_init(
      Int16 task_id,
      UInt16 channel,
      UInt8 sel_mask,
      UInt8 dir_mask)
```

Include file

`slvdsp1104.h`

Purpose

To initialize the slave DSP bit I/O unit.

Description

With this function you can reserve I/O pins (bits) for input or output purposes.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Bit I/O Unit (DS1104 Features 📖).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

channel Specifies the group number within the range 2 … 4.

sel_mask Specifies the reserves single bits (pins) of the specified group for bit I/O purposes. You can specify a bit mask where each bit (0 … 7) represents a pin (0 … 7) of the specified group. A "0" has no effect, a "1" reserves the pin. You can use the symbols predefined as follows. To reserve more than one bit at once, you can combine the predefined symbols by using the logical operator OR.

Predefined Symbol	Meaning
SLVDSP1104_BIT_IO_BIT0_MSK	Reserves shared I/O pin for bit 0
SLVDSP1104_BIT_IO_BIT1_MSK	Reserves shared I/O pin for bit 1
…	…
SLVDSP1104_BIT_IO_BIT7_MSK	Reserves shared I/O pin for bit 7

dir_mask Specifies the configures the bits (pins) of the specified group for input or output. You can specify a bit mask where each bit (0 … 7) represents a pin (0 … 7) of the specified group. A "0" configures the bit for input, a "1" configures the pin for output. Or you can use the symbols that are predefined below. To define the whole group, you must specify a list of predefined symbols combined by using the logical operator OR.

Predefined Symbol	Meaning
SLVDSP1104_BIT_IO_BIT0_IN	Sets bit 0 to input
SLVDSP1104_BIT_IO_BIT0_OUT	Sets bit 0 to output
...	...
SLVDSP1104_BIT_IO_BIT7_IN	Sets bit 7 to input
SLVDSP1104_BIT_IO_BIT7_OUT	Sets bit 7 to output

Related topics

Examples

Example of Using the Bit I/O Functions of the Slave DSP..319

ds1104_slave_dsp_bit_io_read_register

Syntax

```
void ds1104_slave_dsp_bit_io_read_register(
      Int16 task_id,
      Int16 *index,
      UInt32 channel)
```

Include file

`slvdsp1104.h`

Purpose

To register the bit I/O read function in the command table.

Description

Use the returned table index when calling `ds1104_slave_dsp_bit_io_read_request` and one of the functions `ds1104_slave_dsp_bit_io_read` or `ds1104_slave_dsp_bit_io_read_new` to read from the specified I/O group.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Bit I/O Unit (DS1104 Features 📖).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address where the command table index is written:

- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

channel Specifies the group number within the range 2 … 4.

Related topics

Examples

Example of Using the Bit I/O Functions of the Slave DSP..319

References

ds1104_slave_dsp_bit_io_read..324
ds1104_slave_dsp_bit_io_read_new...325
ds1104_slave_dsp_bit_io_read_request..323

ds1104_slave_dsp_bit_io_read_request

Syntax

```
Int16 ds1104_slave_dsp_bit_io_read_request(
      Int16 task_id,
      Int32 index)
```

Include file

`slvdsp1104.h`

Purpose

To request a read from the digital I/O port.

Description

The slave DSP performs the read function independently and writes the results back into the dual-ported memory. To fetch the data from the dual-ported memory, use one of the functions `ds1104_slave_dsp_bit_io_read` or `ds1104_slave_dsp_bit_io_read_new`.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed register function `ds1104_slave_dsp_bit_io_read_register`.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

Examples

Example of Using the Bit I/O Functions of the Slave DSP...319

References

ds1104_slave_dsp_bit_io_read...324
ds1104_slave_dsp_bit_io_read_new...325
ds1104_slave_dsp_bit_io_read_register...322

ds1104_slave_dsp_bit_io_read

Syntax

```
Int16 ds1104_slave_dsp_bit_io_read(
      Int16 task_id,
      Int32 index,
      UInt8 *value)
```

Include file

`slvdsp1104.h`

Purpose

To read the I/O group value from the dual-ported memory.

Description

Prior to this, the read operation must have been requested by the master PPC using the function `ds1104_slave_dsp_bit_io_read_request` that asks for a slave DSP I/O port read.

Note

The specified bits must be reserved for input purposes before by calling `ds1104_slave_dsp_bit_io_init`.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed register function `ds1104_slave_dsp_bit_io_read_register`.

value Specifies the address where the value is written.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_NO_DATA	There is no current data for the specified slave DSP function. So, the data from the previous request has been read.
SLVDSP1104_DATA_LOST	The input data of a previous request for the specified slave DSP function has been overwritten. The current request has been performed without error.

Related topics

Examples

Example of Using the Bit I/O Functions of the Slave DSP..319

References

ds1104_slave_dsp_bit_io_init..321
ds1104_slave_dsp_bit_io_read_new..325
ds1104_slave_dsp_bit_io_read_register...322
ds1104_slave_dsp_bit_io_read_request...323

ds1104_slave_dsp_bit_io_read_new

Syntax

```
Int16 ds1104_slave_dsp_bit_io_read_new(
      Int16 task_id,
      Int32 index,
      UInt8 *value)
```

Include file

`slvdsp1104.h`

Purpose

To poll for a new value of a digital I/O group.

Description

Unlike `ds1104_slave_dsp_bit_io_read`, this function polls for a new value until the slave DSP has delivered a new value.

Note

The function may lead to a deadlock when you do not request a new value. Use `ds1104_slave_dsp_bit_io_read_request` to request the new value.

When polling was successful, the function reads the new value from the dual-ported memory.

Note

The specified bits must be reserved for input purposes before by calling `ds1104_slave_dsp_bit_io_init`.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed register function `ds1104_slave_dsp_bit_io_read_register`.

value Specifies the address where the value is written.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_DATA_LOST	The input data of a previous request for the specified slave DSP function has been overwritten. The current request has been performed without error.

Related topics

References

ds1104_slave_dsp_bit_io_write_register

Syntax

```
void ds1104_slave_dsp_bit_io_write_register(
      Int16 task_id,
      Int16 *index,
      UInt32 channel)
```

Include file

`slvdsp1104.h`

Purpose

To register the write function in the command table.

Description

The returned table index can be used by `ds1104_slave_dsp_bit_io_write` to write a byte to the specified I/O group.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Bit I/O Unit (DS1104 Features 🕮).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 🕮).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address where the command table index is written:

- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

channel Specifies the group number within the range 2 … 4.

Related topics

Examples

References

ds1104_slave_dsp_bit_io_write

Syntax

```
Int16 ds1104_slave_dsp_bit_io_write(
      Int16 task_id,
      Int32 index,
      UInt8 value)
```

Include file

`slvdsp1104.h`

Purpose

To write a value to the specified digital I/O group.

Description

The bits (pins) that are configured for input and the pins that are reserved by other I/O units are ignored (see `ds1104_slave_dsp_bit_io_init`).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function `ds1104_slave_dsp_bit_io_write_register`.

value Specifies the value to be written.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

Examples

Example of Using the Bit I/O Functions of the Slave DSP..319

References

ds1104_slave_dsp_bit_io_init...321
ds1104_slave_dsp_bit_io_write_register..327

ds1104_slave_dsp_bit_io_set_register

Syntax

```
void ds1104_slave_dsp_bit_io_set_register(
      Int16 task_id,
      Int16 *index,
      UInt32 channel)
```

Include file

`slvdsp1104.h`

Purpose

To register the set function in the command table.

Description

The returned table index can be used by `ds1104_slave_dsp_bit_io_set` to set the specified I/O group.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Bit I/O Unit (DS1104 Features).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address where the command table index is written:

- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

channel Specifies the group number within the range 2 … 4.

Related topics

References

ds1104_slave_dsp_bit_io_set

Syntax

```
Int16 ds1104_slave_dsp_bit_io_set(
      Int16 task_id,
      Int32 index,
      UInt8 mask)
```

Include file

`slvdsp1104.h`

Purpose

To set the specified I/O group according to the specified mask.

Description

The pins that are configured for input as well as the pins that are reserved by other I/O units are ignored (see **`ds1104_slave_dsp_bit_io_init`**).

Use **`ds1104_slave_dsp_bit_io_clear`** to clear single bits of an I/O group.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Bit I/O Unit (DS1104 Features 🕮).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 🕮).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function **`ds1104_slave_dsp_bit_io_set_register`**.

mask Specifies the bit mask where each bit (0 … 7) represents a pin (0 … 7) of the specified group. A "0" does not change the bit setting, a "1" resets the associated bit to "0".

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

References

ds1104_slave_dsp_bit_io_clear_register

Syntax

```
void ds1104_slave_dsp_bit_io_clear_register(
      Int16 task_id,
      Int16 *index,
      UInt32 channel)
```

Include file

`slvdsp1104.h`

Purpose

To register the clear function in the command table.

Description

The returned table index can be used by `ds1104_slave_dsp_bit_io_clear` to actually clear the specified I/O group.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Bit I/O Unit (DS1104 Features 🕮).

Note

The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 🕮).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address where the command table index is written:
- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

channel Specifies the group number within the range 2 … 4.

Related topics

References

ds1104_slave_dsp_bit_io_clear 332

ds1104_slave_dsp_bit_io_clear

Syntax

```
Int16 ds1104_slave_dsp_bit_io_clear(
      Int16 task_id,
      Int32 index,
      UInt8 mask)
```

Include file

`slvdsp1104.h`

Purpose

To clear the specified I/O group according to the specified mask.

Description

The pins that are configured for input and the pins that are reserved by other I/O units are ignored (see **`ds1104_slave_dsp_bit_io_init`**).

Use **`ds1104_slave_dsp_bit_io_set`** to set an I/O group.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function **`ds1104_slave_dsp_bit_io_clear_register`**.

mask Specifies the bit mask where each bit (0 … 7) represents a pin (0 … 7) of the specified group. A "0" does not change the bit setting, a "1" resets the associated bit to "0".

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

References

ds1104_slave_dsp_bit_io_clear_register 331
ds1104_slave_dsp_bit_io_init 321
ds1104_slave_dsp_bit_io_set 330

Slave DSP Timing I/O Unit

Where to go from here

Information in this section

Slave DSP PWM Generation

Where to go from here

Information in this section

Information in other sections

DS1104SL_DSP_PWM (DS1104 RTI Reference 📖)

1-Phase PWM Signal Generation (PWM) (DS1104 Features 📖)

Example of Using PWM Functions of the Slave DSP

Example source code

The following example demonstrates how to use PWM functions of the slave DSP. You find the relevant files in the directory `<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\SlaveDSP\Slv_Pwm_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <brtenv.h>
#define DT 1.0e-3               /* 1 ms simulation step size */
/* variables for communication with slave */
Int16 task_id = 0; /* communication channel */
Int16 ch1_index = -1; /* slave DSP command index for ch. 1 */
Int16 ch2_index = -1; /* slave DSP command index for ch. 2 */
Int16 ch3_index = -1; /* slave DSP command index for ch. 3 */
Int16 ch4_index = -1; /* slave DSP command index for ch. 4 */
/* parameters for PWM initialization */
Float64 period = DT;   /* PWM period = simulation step size */
UInt16 mode = SLVDSP1104_PWM_MODE_SYM;      /* PWM mode */
UInt16 pol = SLVDSP1104_PWM_POL_HIGH;       /* PWM polarity */
/* parameters accessed by ControlDesk */
volatile Float64 duty = 0.1;
volatile Int32 channel = 1;             /* PWM channel */
Int32 err;
Float64 exec_time;                      /* execution time */
/* interrupt service routine */
void isr_srt(void)
{
   ts_timestamp_type ts;
```

```
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);        /* data acquisition service */
```

```
    /* write PWM Duty cycle to slave DSP and test for error */
    switch(channel)
    {
        case 1:/* channel 1 selected */
            err = ds1104_slave_dsp_pwm_duty_write(task_id,
                    ch1_index, duty);
            break;
       case 2:/* channel 2 selected */
            err = ds1104_slave_dsp_pwm_duty_write(task_id,
                    ch2_index, duty);
            break;
       case 3:/* channel 3 selected */
            err = ds1104_slave_dsp_pwm_duty_write(task_id,
                    ch3_index, duty);
            break;
       case 4:/* channel 4 selected */
            err = ds1104_slave_dsp_pwm_duty_write(task_id,
                    ch4_index, duty);
            break;
       default:
             break;
    }
    exec_time = RTLIB_TIC_READ();
    RTLIB_SRT_ISR_END();                   /* overload check */
}
```

```
void main(void)
{
   init();          /* DS1104 and RTLib1104 initialization */
   /* init communication with slave_DSP */
   ds1104_slave_dsp_communication_init();
   /* initialization of PWM generation on slave DSP */
   ds1104_slave_dsp_pwm_init(task_id, period, duty, mode, pol,
                             SLVDSP1104_PWM_CH1_MSK |
                             SLVDSP1104_PWM_CH2_MSK |
                             SLVDSP1104_PWM_CH3_MSK |
                             SLVDSP1104_PWM_CH4_MSK);
   /* start of PWM generation on slave DSP */
   ds1104_slave_dsp_pwm_start(task_id,
                              SLVDSP1104_PWM_CH1_MSK |
                              SLVDSP1104_PWM_CH2_MSK |
                              SLVDSP1104_PWM_CH3_MSK |
                              SLVDSP1104_PWM_CH4_MSK);
   /* registration of PWM duty cycle update commands */
   /* channel 1 */
   ds1104_slave_dsp_pwm_duty_write_register(task_id,
     &ch1_index, 1);
   /* channel 2 */
   ds1104_slave_dsp_pwm_duty_write_register(task_id,
      &ch2_index, 2);
   /* channel 3 */
   ds1104_slave_dsp_pwm_duty_write_register(task_id,
      &ch3_index, 3);
   /* channel 4 */
   ds1104_slave_dsp_pwm_duty_write_register(task_id,
      &ch4_index, 4);
   RTLIB_SRT_START(DT, isr_srt);  /* start sample rate timer */
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();      /* background service */
   }
}
```

ds1104_slave_dsp_pwm_init

Syntax

```
void ds1104_slave_dsp_pwm_init(
      Int16 task_id,
      Float64 period,
      Float64 duty,
      UInt16 mode,
      UInt16 pol,
      UInt16 mask)
```

Include file

`slvdsp1104.h`

Purpose

To initialize up to four channels for 1-phase PWM generation on the slave DSP.

Description

After each initialization, you must start the PWM generation using the function `ds1104_slave_dsp_pwm_start`.

You can call the function more than once to initialize the channels not yet initialized. The resolution depends on the PWM period and the mode.

I/O mapping

For information on the I/O mapping, refer to 1-Phase PWM Signal Generation (PWM) (DS1104 Features 🕮).

Note

- The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 🕮).
- When using D2F channel 4, you cannot generate standard PWM signals.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

period Specifies the duration of the PWM period in seconds. The minimum and maximum periods depend on the mode. The following symbols are predefined:

Predefined Symbol	Minimum Period	Maximum Period
SLVDSP1104_PWM_MODE_ASYM	200 ns	400 ms
SLVDSP1104_PWM_MODE_SYM	200 ns	800 ms

The period must be the same for all channels. If you try to set different periods, the channels keep the previous value. For further information on the period values, refer to Basics of Slave DSP PWM Signal Generation (DS1104 Features 🕮).

duty Specifies the duty cycle within the range 0.0 … 1.0. It is scaled according to the basic frequency. The following table shows the relation to the duty cycle given in percent.

Range	Duty Cycle
0.0 … 1.0	0 … 100%

mode Determines whether a mid-symmetrical or begin-synchronized PWM should be used. The mode must be the same for all channels. If you try to set different modes, the channels keep the previous mode. The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_PWM_MODE_ASYM	Sets begin-synchronized PWM
SLVDSP1104_PWM_MODE_SYM	Sets mid-symmetrical PWM

pol Specifies the output polarity of the PWM signals. You can specify different polarities for the PWM channels. The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_PWM_POL_LOW	Set polarity to active low
SLVDSP1104_PWM_POL_HIGH	Set polarity to active high

mask Reserves shared I/O pins for PWM purposes. To reserve more than one channel at once, you can combine the predefined symbols by using the logical operator OR. The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_PWM_CH1_MSK	Reserves shared I/O pin for PWM channel 1
SLVDSP1104_PWM_CH2_MSK	Reserves shared I/O pin for PWM channel 2
SLVDSP1104_PWM_CH3_MSK	Reserves shared I/O pin for PWM channel 3
SLVDSP1104_PWM_CH4_MSK	Reserves shared I/O pin for PWM channel 4

Related topics

Examples

Example of Using PWM Functions of the Slave DSP........335

References

ds1104_slave_dsp_pwm_start........341

ds1104_slave_dsp_pwm_duty_write_register

Syntax

```
void ds1104_slave_dsp_pwm_duty_write_register(
      Int16 task_id,
      Int16 *index,
      UInt32 channel)
```

Include file

`slvdsp1104.h`

Purpose

To register the write function in the command table.

Description

The returned table index can be used by `ds1104_slave_dsp_pwm_duty_write` to actually write to the specified PWM channel.

I/O mapping

For information on the I/O mapping, refer to 1-Phase PWM Signal Generation (PWM) (DS1104 Features 📖).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 ... 2.

index Specifies the address where the command table index is written:
- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

channel Specifies the number of the PWM channel within the range 1 ... 4.

Related topics

Examples

Example of Using PWM Functions of the Slave DSP...335

References

ds1104_slave_dsp_pwm_duty_write..340

ds1104_slave_dsp_pwm_duty_write

Syntax

```
Int16 ds1104_slave_dsp_pwm_duty_write(
      Int16 task_id,
      Int32 index,
      Float64 duty)
```

Include file

`slvdsp1104.h`

Purpose

To set the PWM duty cycle for the PWM channel specified by `ds1104_slave_dsp_pwm_duty_write_register`.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function `ds1104_slave_dsp_pwm_duty_write_register`.

duty Specifies the duty cycle within the range 0.0 … 1.0. It is scaled according to the basic frequency. The following table shows the relation to the duty cycle given in percent.

Range	Duty Cycle
0.0 … 1.0	0 … 100%

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

Examples

Example of Using PWM Functions of the Slave DSP.. 335

References

ds1104_slave_dsp_pwm_duty_write_register.. 339

ds1104_slave_dsp_pwm_start

Syntax

```
Int16 ds1104_slave_dsp_pwm_start(
      Int16 task_id,
      UInt32 mask)
```

Include file

`slvdsp1104.h`

Purpose

To start PWM generation.

Description

Use this function to start the signal generation on the slave DSP. This function is not registered but carried out directly instead.

> **Note**
>
> - PWM generation must have been initialized by using the `ds1104_slave_dsp_pwm_init` function.
> - The bits 0, 1, 2, 4 of the digital I/O port 2 conflict with the simple PWM channels 1 … 4.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

mask Specifies the channels to be started separately. To start more than one PWM channel at once, you can combine the predefined symbols by using the logical operator OR. The following predefined symbols are used:

Predefined Symbol	Meaning
SLVDSP1104_PWM_CH1_MSK	Starts PWM channel 1
SLVDSP1104_PWM_CH2_MSK	Starts PWM channel 2
SLVDSP1104_PWM_CH3_MSK	Starts PWM channel 3
SLVDSP1104_PWM_CH4_MSK	Starts PWM channel 4

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

Examples

Example of Using PWM Functions of the Slave DSP....335

References

ds1104_slave_dsp_pwm_init....337
ds1104_slave_dsp_pwm_stop....343

ds1104_slave_dsp_pwm_stop

Syntax

```
Int16 ds1104_slave_dsp_pwm_stop(
     Int16 task_id,
     UInt32 mask)
```

Include file

`slvdsp1104.h`

Purpose

To stop PWM generation.

Description

Use this function to stop the signal generation on the slave DSP. Only the output of the PWM signal is disabled. Signal calculation is still running and if you start PWM generation the currently calculated signal is output. This function is not registered but carried out directly instead.

Note

- PWM generation must have been initialized by using the `ds1104_slave_dsp_pwm_init` function.
- The bits 0, 1, 2, 4 of the digital I/O port 2 conflict with the simple PWM channels 1 … 4.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

mask Specifies the channels to be stopped separately. To stop more than one PWM channel simultaneously, you can combine the predefined symbols by using the logical operator OR. The following predefined symbols are used:

Predefined Symbol	Meaning
SLVDSP1104_PWM_CH1_MSK	Stops PWM channel 1
SLVDSP1104_PWM_CH2_MSK	Stops PWM channel 2
SLVDSP1104_PWM_CH3_MSK	Stops PWM channel 3
SLVDSP1104_PWM_CH4_MSK	Stops PWM channel 4

Return value

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

Predefined Symbols	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

References

Slave DSP PWM3 Generation

Where to go from here

Information in this section

Information in other sections

DS1104SL_DSP_PWM3 (DS1104 RTI Reference 📖)

3-Phase PWM Signal Generation (PWM3) (DS1104 Features 📖)

Example of Using 3-Phase PWM Functions of the Slave DSP

Example source code

The following example demonstrates how to use 3-phase PWM functions of the slave DSP. You find the relevant files in the directory `<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\SlaveDSP\Slv_Pwm3_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <brtenv.h>          /* basic real-time environment */
/* variables for communication with Slave DSP */
Int16 task_id = 0;               /* communication channel */
Int16 index = -1;                /* slave DSP command index */
/* parameters for PWM initialization */
Float64 period = 10e-3;                  /* PWM period */
Float64 deadband = 0.0;                  /* deadband period */
Float64 sync_pos = 0.5;                  /* position of the synchronization interrupt signal */
/* parameters accessed by ControlDesk */
volatile Float64 duty1 = 0.1;
volatile Float64 duty2 = 0.2;
volatile Float64 duty3 = 0.3;
Float64 exec_time;
/* interrupt service routine for PWM sync interrupt */
void PWM_sync_interrupt(void)
{
   ts_timestamp_type ts;
```

```
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);        /* data acquisition service */
```

```
   /* write PWM3 duty cycle to slave DSP and test for error */
   ds1104_slave_dsp_pwm3_duty_write(task_id, index,
      duty1, duty2, duty3);
   exec_time = RTLIB_TIC_READ();
}
void main(void)
{
   /* basic initialization of DS1104 */
   init();
   /* initialization of slave DSP communication */
   ds1104_slave_dsp_communication_init();
   /* init and start of 3-phase PWM generation on slave DSP */
   ds1104_slave_dsp_pwm3_init(task_id, period, duty1, duty2, duty3, deadband, sync_pos);
   ds1104_slave_dsp_pwm3_start(task_id);
   /* registration of PWM duty cycle update command */
   ds1104_slave_dsp_pwm3_duty_write_register(task_id, &index);
   /* initialization of PWM sync interrupt */
   ds1104_set_interrupt_vector(DS1104_INT_SLAVE_DSP_PWM,
      (DS1104_Int_Handler_Type) &PWM_sync_interrupt,
      SAVE_REGS_ON);
   ds1104_enable_hardware_int(DS1104_INT_SLAVE_DSP_PWM);
   RTLIB_INT_ENABLE();
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();   /* background service */
   }
}
```

ds1104_slave_dsp_pwm3_init

Syntax

```
void ds1104_slave_dsp_pwm3_init(
      Int16 task_id,
      Float64 period,
      Float64 duty1,
      Float64 duty2,
      Float64 duty3,
      Float64 dead_band,
      Float64 sync_pos)
```

Include file

`slvdsp1104.h`

Purpose

To initialize the PWM3 on the slave DSP.

Description

Use `ds1104_slave_dsp_pwm3_duty_write_register` and `ds1104_slave_dsp_pwm3_duty_write` to set the duty cycles. Use `ds1104_slave_dsp_pwm3_start`, `ds1104_slave_dsp_pwm3_stop` to start and stop the generation of the PWM3. The 3-phase PWM (PWM3) and PWMSV generations use the same connector pins.

Note

- When using 3-phase PWM (PWM3), you cannot generate the D2F square wave signals and the 3-phase Space Vector PWM (PWMSV).
- For PWM3 generation, the PWM interrupt from the slave DSP to the master PPC is available. It can be generated at any position within the PWM period. See ds1104_slave_dsp_pwm3_int_init on page 348 for how to make the slave DSP PWM interrupt available.

I/O mapping

For information on the I/O mapping, refer to 3-Phase PWM Signal Generation (PWM3) (DS1104 Features 🕮).

Note

The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 🕮).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

period Specifies the duration of the PWM period given in seconds. For detailed information on the dependency of period and resolution, refer to Basics of Slave DSP PWM Signal Generation (DS1104 Features 📖).

duty1 Specifies the initial duty cycle for the 3-phase PWM channel 1 within the range 0.0 … 1.0.

duty2 Specifies the initial duty cycle for the 3-phase PWM channel 2 within the range 0.0 … 1.0.

duty3 Specifies the initial duty cycle for the 3-phase PWM channel 3 within the range 0.0 … 1.0. The following table shows the relation to the duty cycle given in percent.

Range	Duty Cycle
0.0 … 1.0	0 … 100%

dead_band Specifies the time delay between the edges of the original and the inverted output signals given in seconds. Values are within the range 0 … 100 µs.

sync_pos Specifies the position of the synchronization interrupt. The interrupt is triggered by the falling edge of the low-active synchronization interrupt signal. The value must be given within the range 0.0 … 1.0. The following table shows the relation to the synchronization interrupt position in percent.

Range	Synchronization Interrupt Position
0.0 … 1.0	0 … 100%

The three basic interrupt states are defined by the following symbols:

Predefined Symbol	Meaning
SLVDSP1104_PWM3_SYNC_OFF	0 (no interrupt)
SLVDSP1104_PWM3_SYNC_LEFT	1.0 (right from the start of the PWM period)
SLVDSP1104_PWM3_SYNC_CENT	0xFFFF (in the middle of the PWM period)

Note

If you use `sync_pos ≤ 0` or `sync_pos > 1`, the interrupt generation is disabled.

Related topics

Examples

References

ds1104_slave_dsp_pwm3_int_init

Syntax

```
void ds1104_slave_dsp_pwm3_int_init(isr_function_name)
```

Include file

`slvdsp1104.h`

Purpose

To initialize and enable the slave DSP interrupt for the PWM3 generation.

Description

This macro initializes the slave DSP interrupt `DS1104_INT_SLAVE_DSP_PWM` by means of `ds1104_set_interrupt_vector`. Then the interrupt will be enabled via `ds1104_enable_hardware_int` and `DS1104_GLOBAL_INTERRUPT_ENABLE()`.

Parameters

isr_function_name	Specifies the name of the interrupt service routine.

Related topics

Examples

References

ds1104_slave_dsp_pwm3_duty_write_register

Syntax

```
void ds1104_slave_dsp_pwm3_duty_write_register(
      Int16 task_id,
      Int16 *index)
```

Include file

`slvdsp1104.h`

Purpose

To register the write function in the command table.

Description

The returned table index can be used by `ds1104_slave_dsp_pwm3_duty_write` to actually set the duty cycles of a 3-phase PWM on the slave DSP.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address where the command table index is written:

- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

Related topics

Examples

Example of Using 3-Phase PWM Functions of the Slave DSP.. 344

References

ds1104_slave_dsp_pwm3_duty_write.. 349

ds1104_slave_dsp_pwm3_duty_write

Syntax

```
Int16 ds1104_slave_dsp_pwm3_duty_write(
      Int16 task_id,
      Int32 index,
      Float64 duty1,
      Float64 duty2,
      Float64 duty3)
```

Include file `slvdsp1104.h`

Purpose To set the three duty cycles of a 3-phase PWM on the slave DSP.

Description Use `ds1104_slave_dsp_pwm3_start` and `ds1104_slave_dsp_pwm3_stop` to start and stop generation of the PWM3.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function `ds1104_slave_dsp_pwm3_duty_write_register`.

duty1 Specifies the duty cycle for the 3-phase PWM channel 1 within the range 0.0 … 1.0. It is scaled according to the basic frequency.

duty2 Specifies the duty cycle for the 3-phase PWM channel 2 within the range 0.0 … 1.0. It is scaled according to the basic frequency.

duty3 Specifies the duty cycle for the 3-phase PWM channel 3 within the range 0.0 … 1.0. It is scaled according to the basic frequency. The following table shows the relation to the duty cycle given in percent.

Range	Duty Cycle
0.0 … 1.0	0 … 100%

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

Predefined Symbols	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

Examples

Example of Using 3-Phase PWM Functions of the Slave DSP.......... 344

References

ds1104_slave_dsp_pwm3_duty_write_register.......... 349
ds1104_slave_dsp_pwm3_start.......... 351
ds1104_slave_dsp_pwm3_stop.......... 351

ds1104_slave_dsp_pwm3_start

Syntax

```
Int16 ds1104_slave_dsp_pwm3_start(Int16 task_id)
```

Include file

`slvdsp1104.h`

Purpose

To start the 3-phase PWM and 3-phase Space Vector PWM (PWMSV) generation.

Description

Use this function to start the signal generation on the slave DSP. This function is carried out directly without registration.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

Examples

Example of Using 3-Phase PWM Functions of the Slave DSP......344

References

ds1104_slave_dsp_pwm3_stop......351

ds1104_slave_dsp_pwm3_stop

Syntax

```
Int16 ds1104_slave_dsp_pwm3_stop(Int16 task_id)
```

Include file

`slvdsp1104.h`

Purpose

To stop the 3-phase PWM and 3-phase Space Vector PWM (PWMSV) generation.

Description

Use this function to stop the signal generation on the slave DSP. This function is carried out directly without registration.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

References

ds1104_slave_dsp_pwm3_start..351

Slave DSP PWMSV Generation

Where to go from here

Information in this section

Information in other sections

Example of Using 3-Phase PWMSV Functions of the Slave DSP

Example source code

The following example demonstrates how to use 3-phase PWMSV functions of the slave DSP.

```
#include <brtenv.h>          /* basic real-time environment */
/* variables for communication with Slave DSP */
Int16 task_id = 0;               /* communication channel */
Int16 index = -1;                /* slave DSP command index */
/* parameters for PWM initialization */
Float64 period = 10e-3;                  /* PWM period */
Float64 deadband = 0.0;                  /* deadband period */
Float64 sync_pos = 0.75;                 /* position of the synch. interrupt signal */;
/* parameters accessed by ControlDesk */
volatile Float64 t1 = 0.5e-3;
volatile Float64 t2 = 0.5e-3;
Int16 sector = 1;
/* interrupt service routine for PWM sync interrupt */
void PWM_sync_interrupt(void)
{
   ts_timestamp_type ts;

   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);        /* data acquisition service */

   /* write PWM3 duty cycle to slave DSP and test for error */
   error = ds1104_slave_dsp_pwm3sv_duty_write(task_id, index, sector, t1, t2);
   if ( error != DSCOMDEF_NO_ERROR)
   {
      …
   }
}
void main(void)
{
   /* basic initialization of DS1104 */
   init();
   /* initialization of slave DSP communication */
   ds1104_slave_dsp_communication_init();
```

```
   /* init and start of 3-phase PWMSV generation on slave DSP */
   ds1104_slave_dsp_pwm3sv_init(task_id, period, sector, t1, t2, deadband, sync_pos);
   ds1104_slave_dsp_pwm3_start(task_id);
   /* registration of PWM duty cycle update command */
   ds1104_slave_dsp_pwm3sv_duty_write_register(task_id, &index);
   /* initialization of PWM sync interrupt */
   ds1104_set_interrupt_vector(DS1104_INT_SLAVE_DSP_PWM,
      (DS1104_Int_Handler_Type) &PWM_sync_interrupt,
      SAVE_REGS_ON);
   ds1104_enable_hardware_int(DS1104_INT_SLAVE_DSP_PWM);
   RTLIB_INT_ENABLE();
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();   /* background service */
   }
}
```

ds1104_slave_dsp_pwm3sv_init

Syntax

```
void ds1104_slave_dsp_pwm3sv_init(
      Int16 task_id,
      Float64 period,
      UInt16 sector,
      Float64 t1,
      Float64 t2,
      Float64 dead_band,
      Float64 sync_pos)
```

Include file

`slvdsp1104.h`

Purpose

To initialize the 3-phase Space Vector PWM (PWMSV) on the slave DSP.

Description

Use `ds1104_slave_dsp_pwm3sv_duty_write_register` and `ds1104_slave_dsp_pwm3sv_duty_write` to set the duty cycles. Use `ds1104_slave_dsp_pwm3_start` and `ds1104_slave_dsp_pwm3_stop` to start and stop the generation of the PWMSV. PWMSV and 3-phase PWM (PWM3) use the same connector pins.

Note

- When using 3-phase Space Vector PWM (PWMSV), you cannot generate the D2F square wave signals and the 3-phase PWM.
- For PWMSV generation, the PWM interrupt from the slave DSP to the master PPC is available. It can be generated at any position within the PWM period. See ds1104_slave_dsp_pwm3_int_init on page 348 for how to make the slave DSP PWM interrupt available.

I/O mapping

For information on the I/O mapping, refer to Space Vector PWM Signal Generation (PWMSV) (DS1104 Features 📖).

Note

The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

period Specifies the duration of the PWM period given in seconds. Valid values are within the range 0 … 0.8 s. For detailed information on the dependency of period and resolution, refer to Space Vector PWM Signal Generation (PWMSV) (DS1104 Features 📖).

sector Specifies the initial sector of the PWM. Values are within the range 1 … 6.

t1 Specifies the initial duration of the first vector given in seconds.

t2 Specifies the initial duration of the second vector given in seconds.

Note

The sum of t1 and t2 must be less or equal to the value of the *period*.

dead_band Specifies the time delay between the edges of the original and the inverted output signals given in seconds. Values are within the range 0 … 100 µs.

sync_pos Specifies the position of the synchronization interrupt. The interrupt is triggered by the falling edge of the low-active synchronization interrupt signal. The value must be given within the range 0.0 … 1.0. The following table shows the relation to the synchronization interrupt position in percent.

Range	Synchronization Interrupt Position
0.0 … 1.0	0 … 100%

The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_PWM3_SYNC_OFF	0 (no interrupt)
SLVDSP1104_PWM3_SYNC_LEFT	1.0 (right from the start of the PWM period)
SLVDSP1104_PWM3_SYNC_CENT	0xFFFF (in the middle of the PWM period)

Note

If you use `sync_pos ≤ 0` or `sync_pos > 1`, the interrupt generation is disabled.

Return value

None

Related topics

Examples

Example of Using 3-Phase PWMSV Functions of the Slave DSP....353

References

ds1104_slave_dsp_pwm3_int_init....348
ds1104_slave_dsp_pwm3_start....351
ds1104_slave_dsp_pwm3_stop....351
ds1104_slave_dsp_pwm3sv_duty_write....357
ds1104_slave_dsp_pwm3sv_duty_write_register....356

ds1104_slave_dsp_pwm3sv_duty_write_register

Syntax

```
void ds1104_slave_dsp_pwm3sv_duty_write_register(
      Int16 task_id,
      Int16 *index)
```

Include file

`slvdsp1104.h`

Purpose

To register the write function in the command table.

Description

The returned table index can be used by `ds1104_slave_dsp_pwm3sv_duty_write` to actually set the duration of the vectors of a 3-phase Space Vector PWM on the slave DSP.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address where the command table index is written:
- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

Related topics

Examples

Example of Using 3-Phase PWMSV Functions of the Slave DSP... 353

References

ds1104_slave_dsp_pwm3sv_duty_write.. 357

ds1104_slave_dsp_pwm3sv_duty_write

Syntax

```
Int16 ds1104_slave_dsp_pwm3sv_duty_write(
      Int16 task_id,
      Int32 index,
      UInt16 sector,
      Float64 t1,
      Float64 t2)
```

Include file

`slvdsp1104.h`

Purpose

To set the 3-phase Space Vector PWM duty cycles on the slave DSP.

Description

Use `ds1104_slave_dsp_pwm3_start` and `ds1104_slave_dsp_pwm3_stop` to start and stop the generation of the PWMSV.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function `ds1104_slave_dsp_pwm3sv_duty_write_register`.

sector Specifies the sector of the PWM. Values are within the range 1 … 6.

t1 Specifies the initial duration of the first vector given in seconds.

t2 Specifies the initial duration of the second vector given in seconds.

Note

The sum of t1 and t2 must be less or equal to the value of the *period* (see `ds1104_slave_dsp_pwm3sv_init`).

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

Examples

Example of Using 3-Phase PWMSV Functions of the Slave DSP....353

References

ds1104_slave_dsp_pwm3_start....351
ds1104_slave_dsp_pwm3_stop....351
ds1104_slave_dsp_pwm3sv_duty_write_register....356
ds1104_slave_dsp_pwm3sv_init....354

Square Wave Signal Generation (D2F)

Where to go from here

Information in this section

Information in other sections

Slave DSP Square-Wave Signal Generation (D2F) (DS1104 Features 📖)

DS1104SL_DSP_D2F (DS1104 RTI Reference 📖)

Example of Using the Square Wave Signal Generation of the Slave DSP

Example source code

The following example demonstrates how to use square wave signal generation of the slave DSP. You find the relevant files in the directory `<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\SlaveDSP\Slv_D2F_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <brtenv.h>          /* basic real-time environment */
#define DT 1.0e-1              /* 1 ms simulation step size */
/* parameters for initialization */
UInt16 range = 6;            /* frequency range within 1...8 */
UInt16 ch4_enable = SLVDSP1104_D2F_CH4_ENABLE;
/* parameters accessed by ControlDesk */
volatile Float64 freq = 400;             /* frequency in Hz */
volatile Int32 channel = 1;          /* Channel number 1...4 */
Float64 exec_time;                        /* execution time */
/* variables for communication with Slave DSP */
Int16 task_id = 0;                 /* communication channel */
Int16 ch1_index = -1; /* slave DSP command index for ch.1 */
Int16 ch2_index = -1; /* slave DSP command index for ch.2 */
Int16 ch3_index = -1; /* slave DSP command index for ch.3 */
Int16 ch4_index = -1; /* slave DSP command index for ch.4 */
```

```
/* interrupt service routine */
void isr_srt(void)
{
   ts_timestamp_type ts;
```

```
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);   /* data acquisition service*/
```

```
   /* write D2F frequency to slave DSP and test for error */
   switch(channel)
   {
      case 1: /* channel 1 selected */
         ds1104_slave_dsp_d2f_write(task_id, ch1_index, freq);
         break;
      case 2: /* channel 2 selected */
         ds1104_slave_dsp_d2f_write(task_id, ch2_index, freq);
         break;
      case 3: /* channel 3 selected */
         ds1104_slave_dsp_d2f_write(task_id, ch3_index, freq);
         break;
      case 4: /* channel 4 selected */
         ds1104_slave_dsp_d2f_write(task_id, ch4_index, freq);
         break;
      default:
         break;
   }
   exec_time = RTLIB_TIC_READ();
   RTLIB_SRT_ISR_END()
}
void main(void)
{
   /* DS1104 and RTLib1104 initialization */
   init();
   /* initialization of slave DSP communication */
   ds1104_slave_dsp_communication_init();
   /* init of D2F generation on slave DSP */
   ds1104_slave_dsp_d2f_init(task_id, range, freq, ch4_enable);
   /* registration of D2F write commands */
   /* channel 1 */
   ds1104_slave_dsp_d2f_write_register(task_id,
      &ch1_index, 1);
   /* channel 2 */
   ds1104_slave_dsp_d2f_write_register(task_id,
      &ch2_index, 2);
   /* channel 3 */
   ds1104_slave_dsp_d2f_write_register(task_id,
      &ch3_index, 3);
   /* channel 4 */
   ds1104_slave_dsp_d2f_write_register(task_id,
      &ch4_index, 4);
   msg_info_set(MSG_SM_RTLIB, 0, "System started.");
   /* start sample rate timer */
   RTLIB_SRT_START(DT, isr_srt);
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();   /* background service */
   }
}
```

ds1104_slave_dsp_d2f_init

Syntax

```
void ds1104_slave_dsp_d2f_init(
      Int16 task_id,
      UInt16 range,
      Float64 freq,
      UInt16 ch4_enable)
```

Include file

`slvdsp1104.h`

Purpose

To initialize the generation of up to 4 square wave signals (D2F).

Description

If a frequency below the lower limit is chosen, square wave signal generation will stop.

Note

- When using D2F square wave signal generation, you cannot generate 3-phase PWM or 3-phase Space Vector PWM signals.
- When using D2F channel 4, you cannot generate standard PWM signals.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Square-Wave Signal Generation (D2F) (DS1104 Features 🕮).

Note

The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 🕮).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

range Specifies the frequency range within the range 1 … 8. For detailed information about the frequency ranges, refer to Slave DSP Square-Wave Signal Generation (D2F) (DS1104 Features 🕮).

freq Specifies the initial signal frequency. Values must remain within the selected range. If a frequency below the lower limit is chosen, signal generation will stop.

Note

To minimize the quantization effect on the frequency resolution, you should select the smallest possible frequency range. For detailed information, refer to Slave DSP Timing I/O Unit (DS1104 Features 📖).

ch4_enable Specifies the selection of channel 4. The following predefined symbols are used:

Predefined Symbol	Meaning
SLVDSP1104_D2F_CH4_ENABLE	Channel 4 enabled
SLVDSP1104_D2F_CH4_DISABLE	Channel 4 disabled

Related topics

Examples

Example of Using the Square Wave Signal Generation of the Slave DSP.................................359

References

ds1104_slave_dsp_d2f_write...363
ds1104_slave_dsp_d2f_write_register...362

ds1104_slave_dsp_d2f_write_register

Syntax

```
void ds1104_slave_dsp_d2f_write_register(
      Int16 task_id,
      Int16 *index,
      UInt32 channel)
```

Include file

`slvdsp1104.h`

Purpose

To register the write function in the command table.

Description

The returned table index can be used by `ds1104_slave_dsp_d2f_write` to actually set the frequency of the square wave generation on the slave DSP.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Square-Wave Signal Generation (D2F) (DS1104 Features 📖).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address where the command table index is written:
- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

channel Specifies the channel of the square wave signal generation within the range 1 … 4.

Related topics

Examples

References

ds1104_slave_dsp_d2f_write

Syntax

```
Int16 ds1104_slave_dsp_d2f_write(
      Int16 task_id,
      Int32 index,
      Float64 freq)
```

Include file

`slvdsp1104.h`

Purpose

To set the frequency of the square wave generation.

Description

If a frequency below the lower limit is chosen, the square wave signal generation will stop.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function `ds1104_slave_dsp_d2f_write_register`.

freq Specifies the frequency of the square wave generation. Values must remain within the selected range.

> **Note**
>
> To minimize the quantization effect on the frequency resolution, you should select the smallest possible frequency range. For detailed information, refer to Slave DSP Timing I/O Unit (DS1104 Features 📖).

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

Examples

Example of Using the Square Wave Signal Generation of the Slave DSP.................................359

References

ds1104_slave_dsp_d2f_write_register..362

Square Wave Signal Measurement (F2D)

Where to go from here

Information in this section

Information in other sections

Slave DSP Square-Wave Signal Measurement (F2D) (DS1104 Features 🕮)

DS1104SL_DSP_F2D (DS1104 RTI Reference 🕮)

Example of Using the Square Wave Signal Measurement of the Slave DSP

Example source code

The following example demonstrates how to use square wave signal measurement of the slave DSP. You find the relevant files in the directory `<RCP_HIL_InstallationPath>\Demos\DS1104\RTLib\SlaveDSP\Slv_F2d_1104_hc`. You can use ControlDesk to load and start the application on the DS1104.

```
#include <brtenv.h>          /* basic real-time environment */
#define DT 1.0e-2            /* 10 ms simulation step size */
/* variables for communication with Slave DSP */
Int16 task_id = 0;      /* communication channel */
Int16 ch1_index = -1;   /* slave DSP command index for ch.1 */
Int16 ch2_index = -1;   /* slave DSP command index for ch.2 */
Int16 ch3_index = -1;   /* slave DSP command index for ch.3 */
Int16 ch4_index = -1;   /* slave DSP command index for ch.4 */
```

```
* parameters for initialization */
Float64 fmin1 =   1;         /* minimum frequency for ch. 1 */
Float64 fmin2 =   5;         /* minimum frequency for ch. 2 */
Float64 fmin3 =  50;         /* minimum frequency for ch. 3 */
Float64 fmin4 = 100;         /* minimum frequency for ch. 4 */
UInt16 int_enable = SLVDSP1104_INT_DISABLE;
/* variables accessed by ControlDesk */
volatile Int32 channel = 1;             /* Input via SCAP1 */
Int32 err_f2d, err_read;
Int16 index;
UInt32 status;
Float64 exec_time;                       /* execution time */
Float64 frequency;
/* interrupt service routine for timer 0 interrupt */
void isr_srt(void)
{
   Float64 freq;
   UInt16 temp;
   ts_timestamp_type ts;
```

```
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);        /* data acquisition service */
```

```
   switch(channel)
   {
      case 1: /* channel 1 selected */
         index = ch1_index;
         break;
      case 2: /* channel 2 selected */
         index = ch2_index;
         break;
      case 3: /* channel 3 selected */
         index = ch3_index;
         break;
      case 4: /* channel 4 selected */
         index = ch4_index;
         break;
      default:
         break;
   }
   /* request read frequency from slave DSP */
   err_f2d = ds1104_slave_dsp_f2d_read_request(task_id,
      index);
   /* read F2D frequency from slave DSP*/
   do
   {
      err_read = ds1104_slave_dsp_f2d_read(task_id, index,
         &freq, &temp);
   } while (err_read == SLVDSP1104_NO_DATA);
   status = temp;
   frequency = freq;
   exec_time = RTLIB_TIC_READ();
   RTLIB_SRT_ISR_END();
}
```

```
void main(void)
{
   /* DS1104 and RTLib1104 initialization */
   init();
   /* initialization of slave DSP communication */
   ds1104_slave_dsp_communication_init();
   /* init of F2D frequency measurement on slave DSP */
   ds1104_slave_dsp_f2d_init(task_id, fmin1, fmin2,
      fmin3, fmin4);
   /* registration of F2D read commands */
   /* channel 1 */
   ds1104_slave_dsp_f2d_read_register(task_id, &ch1_index,
      1, int_enable);
   /* channel 2 */
   ds1104_slave_dsp_f2d_read_register(task_id, &ch2_index,
      2, int_enable);
   /* channel 3 */
   ds1104_slave_dsp_f2d_read_register(task_id, &ch3_index,
      3, int_enable);
   /* channel 4 */
   ds1104_slave_dsp_f2d_read_register(task_id, &ch4_index,
      4, int_enable);
   /* periodic event in ISR */
   RTLIB_SRT_START(DT, isr_srt);
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE();   /* background service */
   }
}
```

ds1104_slave_dsp_f2d_init

Syntax

```
void ds1104_slave_dsp_f2d_init(
      Int16 task_id,
      Float64 fmin1,
      Float64 fmin2,
      Float64 fmin3,
      Float64 fmin4)
```

Include file

`slvdsp1104.h`

Purpose

To initialize the frequency measurements on the slave DSP for channels 1 … 4.

Description

Use `ds1104_slave_dsp_f2d_read_register`, `ds1104_slave_dsp_f2d_read_request`, and `ds1104_slave_dsp_f2d_read` to read the frequency measurement. For detailed information about the ranges for frequency measurement, refer to Slave DSP Square-Wave Signal Measurement (F2D) (DS1104 Features 📖).

Note

- When using the F2D frequency measurement, you cannot perform PWM2D measurement.
- The values of the maximum frequency depend on the number of used channels. When exceeding these ranges the measurement may be faulty. When using other interrupt-based functions at the same time – for example: square wave signals generation (D2F) – there may be measurement faults even in lower frequency ranges.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Square-Wave Signal Measurement (F2D) (DS1104 Features 📖).

Note

The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

fmin1 Specifies the minimum frequency to be measured for channel 1.

fmin2 Specifies the minimum frequency to be measured for channel 2.

fmin3 Specifies the minimum frequency to be measured for channel 3.

fmin4 Specifies the minimum frequency to be measured for channel 4.

The values must be given within the range of 0.005 … 150 Hz. For frequencies below fmin, the measurement returns "0". If you choose a very small value for fmin, the time to detect the frequency f = 0 will increase.

Note

To minimize the deviations, which are caused by a quantization effect, between the input frequency and the measured frequency value, you should select the smallest possible frequency range. For detailed information, refer to Slave DSP Square-Wave Signal Measurement (F2D) (DS1104 Features 📖).

Related topics

Basics

Slave DSP Square-Wave Signal Measurement (F2D) (DS1104 Features 📖)

Examples

References

ds1104_slave_dsp_f2d_read_register

Syntax

```
void ds1104_slave_dsp_f2d_read_register(
      Int16 task_id,
      Int16 *index,
      UInt16 channel,
      UInt16 int_enable)
```

Include file

`slvdsp1104.h`

Purpose

To register the read function in the command table.

Description

The returned table index can be used by `ds1104_slave_dsp_f2d_read_request` to request the measurement, and by `ds1104_slave_dsp_f2d_read` to read the data.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Square-Wave Signal Measurement (F2D) (DS1104 Features 📖).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address where the command table index is written:

- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

channel Specifies the channel of the frequency generation within the range 1 … 4.

int_enable Specifies the interrupt setting. The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_INT_DISABLE	Disables interrupt to the master PPC
SLVDSP1104_INT_ENABLE	Generates an interrupt to the master PPC when read is successfully completed and a new frequency result is available

Related topics

Examples

Example of Using the Square Wave Signal Measurement of the Slave DSP............................365

References

ds1104_slave_dsp_f2d_read..371
ds1104_slave_dsp_f2d_read_request..370

ds1104_slave_dsp_f2d_read_request

Syntax

```
Int16 ds1104_slave_dsp_f2d_read_request(
      Int16 task_id,
      Int32 index)
```

Include file

`slvdsp1104.h`

Purpose

To request a frequency measurement from the slave DSP.

Description

The slave DSP performs the measurement independently and writes the result back into the dual-ported memory. To fetch the data from the dual-ported memory use `ds1104_slave_dsp_f2d_read`.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function `ds1104_slave_dsp_f2d_read_register`.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

Examples

Example of Using the Square Wave Signal Measurement of the Slave DSP.............................. 365

References

ds1104_slave_dsp_f2d_read.. 371
ds1104_slave_dsp_f2d_read_register.. 369

ds1104_slave_dsp_f2d_read

Syntax

```
Int16 ds1104_slave_dsp_f2d_read(
      Int16 task_id,
      Int32 index,
      Float64 *freq,
      UInt16 *status)
```

Include file

`slvdsp1104.h`

Purpose

To read the frequency measurement data from the dual-ported memory.

Description

Prior to this, the data must have been requested by the master PPC using the function `ds1104_slave_dsp_f2d_read_request` that asks for a slave DSP frequency measurement read.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function `ds1104_slave_dsp_f2d_read_register`.

freq Specifies the value of the measured frequency.

Note

To minimize the deviations, which are caused by a quantization effect, between the input frequency and the measured frequency value, you should select the smallest possible frequency range. For detailed information, refer to Slave DSP Square-Wave Signal Measurement (F2D) (DS1104 Features).

status Specifies the measurement status. Each rising edge of the signal generates a slave DSP interrupt and thus a new measurement value. That means for example: frequency of the signal is 10 Hz, the measurement is performed every 1 ms, so only every 100th measurement represents a new value. The following predefined symbols are used:

Predefined Symbol	Meaning
SLVDSP1104_F2D_OLD	Old measurement value
SLVDSP1104_F2D_NEW	New measurement value

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_NO_DATA	There is no current data for the specified slave DSP function. So, the data from the previous request has been read.
SLVDSP1104_DATA_LOST	The input data of a previous request for the specified slave DSP function has been overwritten. The current request has been performed without error.

Related topics

Examples

Example of Using the Square Wave Signal Measurement of the Slave DSP.............................365

References

ds1104_slave_dsp_f2d_read_register...369
ds1104_slave_dsp_f2d_read_request...370

Slave DSP PWM Measurement (PWM2D)

Where to go from here

Information in this section

Information in other sections

Slave DSP PWM Signal Measurement (PWM2D) (DS1104 Features 🕮)

DS1104SL_DSP_PWM2D (DS1104 RTI Reference 🕮)

ds1104_slave_dsp_pwm2d_init

Syntax

```
void ds1104_slave_dsp_pwm2d_init(Int16 task_id)
```

Include file

`slvdsp1104.h`

Purpose

To initialize the PWM period and duty cycle measurements on the slave DSP for channels 1 … 4.

Description

Use `ds1104_slave_dsp_pwm2d_read_register`, `ds1104_slave_dsp_pwm2d_read_request`, and `ds1104_slave_dsp_pwm2d_read` to read the measurement data. For detailed information about the ranges for PWM period and duty cycle measurement, refer to Slave DSP PWM Signal Measurement (PWM2D) (DS1104 Features 🕮).

Note

- When using the PWM measurement, you cannot perform F2D frequency measurement.
- The values of the minimum period depend on the number of channels used. When these ranges are exceeded, the measurement may be faulty. When using other interrupt based functions at the same time – for example: square wave signal generation (D2F) – there may be measurement faults even in higher ranges.

I/O mapping

For information on the I/O mapping, refer to Slave DSP PWM Signal Measurement (PWM2D) (DS1104 Features 📖).

Note

The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

task_id	Specifies the communication channel within the range 0 … 2.

Return value

None

Related topics

Basics

Slave DSP PWM Signal Measurement (PWM2D) (DS1104 Features 📖)

References

ds1104_slave_dsp_pwm2d_read_register

Syntax

```
void ds1104_slave_dsp_pwm2d_read_register(
      Int16 task_id,
      Int16 *index,
      UInt16 channel,
      UInt16 int_enable)
```

Include file

`slvdsp1104.h`

Purpose

To register the read function in the command table.

Description

The returned table index can be used by `ds1104_slave_dsp_pwm2d_read_request` to request the measurement, and by `ds1104_slave_dsp_pwm2d_read` to read the data.

I/O mapping

For information on the I/O mapping, refer to Slave DSP PWM Signal Measurement (PWM2D) (DS1104 Features 📖).

> **Note**
>
> The I/O mapping of this function can conflict with other I/O functions. For further information, refer to Conflicting I/O Features (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address where the command table index is written:
- input: If (index value = –1) an available command table index is chosen, otherwise the input index value is used.
- output: address where the selected index is written.

channel Specifies the channel of the frequency generation within the range 1 … 4.

int_enable Specifies the interrupt setting. The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_INT_DISABLE	Disables interrupt to the master PPC
SLVDSP1104_INT_ENABLE	Generates an interrupt to the master PPC when read is successfully completed and a new measurement result is available

Return value

None

Related topics

References

ds1104_slave_dsp_pwm2d_read 377
ds1104_slave_dsp_pwm2d_read_request 376

ds1104_slave_dsp_pwm2d_read_request

Syntax

```
Int16 ds1104_slave_dsp_pwm2d_read_request(
      Int16 task_id,
      Int32 index)
```

Include file

`slvdsp1104.h`

Purpose

To request a PWM period and duty cycle measurement from the slave DSP.

Description

The slave DSP performs the measurement independently and writes the result back into the dual-ported memory. To fetch the data from the dual-ported memory use `ds1104_slave_dsp_pwm2d_read`.

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function `ds1104_slave_dsp_pwm2d_read_register`.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_BUFFER_OVERFLOW	The communication buffer from master PPC to slave DSP has overflown.

Related topics

References

ds1104_slave_dsp_pwm2d_read..377
ds1104_slave_dsp_pwm2d_read_register..374

ds1104_slave_dsp_pwm2d_read

Syntax

```
Int16 ds1104_slave_dsp_pwm2d_read(
      Int16 task_id,
      Int32 index,
      Float64 *period,
      Float64 *duty,
      UInt16 *status)
```

Include file

`slvdsp1104.h`

Purpose

To read the PWM period and duty cycle measurement data from the dual-ported memory.

Description

The measurement algorithm used is accurate if the PWM period starts with the falling or rising edge of the corresponding PWM signal (asymmetric signal).

The DS1104 can also be used to measure PWM signals that are centered around the middle of the PWM period (symmetric signals). However, the measurement of the PWM frequency of symmetric PWM signals is faulty if the duty cycle of the PWM signal changes during measurement. For details, refer to Limitation for the Measurement of Symmetric PWM Signals (DS1104 Features 📖).

Note

- Prior to this, the data must have been requested by the master PPC using the function `ds1104_slave_dsp_pwm2d_read_request` that asks for a slave DSP PWM period and duty cycle measurement read.
- Due to a complex interrupt handling, the function works well within certain limits only. For detailed information, refer to Slave DSP PWM Signal Measurement (PWM2D) (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the table index already allocated by the previously performed function `ds1104_slave_dsp_pwm2d_read_register`.

period Specifies the measured PWM period in seconds.

duty Specifies the measured duty cycle within the range 0.0 … 1.0.

status Specifies the measurement status. Each rising edge of the signal generates a slave DSP interrupt and thus a new measurement of PWM period and duty cycle. That is for example: the frequency of the signal is 10 Hz, the measurement is performed every 1 ms, so only every 100th measurement represents a new value. The following predefined symbols are used:

Predefined Symbol	Meaning
SLVDSP1104_PWM2D_OLD	Old measurement value
SLVDSP1104_PWM2D_NEW	New measurement value

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	The function has been performed without error.
SLVDSP1104_NO_DATA	There is no current data for the specified slave DSP function. So, the data from the previous request has been read.
SLVDSP1104_DATA_LOST	The input data of a previous request for the specified slave DSP function has been overwritten. The current request has been performed without error.

Related topics

Basics

Limitation for the Measurement of Symmetric PWM Signals (DS1104 Features 🕮)
Slave DSP PWM Signal Measurement (PWM2D) (DS1104 Features 🕮)

References

ds1104_slave_dsp_pwm2d_read_register......374
ds1104_slave_dsp_pwm2d_read_request......376

Slave DSP Serial Peripheral Interface

Where to go from here

Information in this section

Information in other sections

Slave DSP Serial Peripheral Interface (SPI) (DS1104 Features 📖)

Example of Using the Serial Peripheral Interface

Example source code

The following example contains the source code for initializing the serial port as master with a baudrate of 500 kBd. The clock signal triggers a transfer of 8-bit length on the rising edge with delayed output. One byte (0x62) will be sent, and one byte will be received, if available.

The read function is registered in the slave DSP's command table. Before you can read the data from the communication buffer, the data of the specified read function must be requested. To write data to the communication buffer, the write function must be registered in the command table before.

```
#include <Brtenv.h>
#define DT 1e-3                         /* 1 ms simulation step size */
Int16 task_id0 = 0;                     /* communication channel 0 */
Int16 task_id1 = 1;                     /* communication channel 1 */
UInt32 baudrate = 500000;               /* set baudrate to 500000 bps */
UInt16 databits = 8;                    /* character is 8 bit long */
/* use automatic mode to assign new index */
Int16 idx0 = DSCOMDEF_AUTO_INDEX;
Int16 idx1 = DSCOMDEF_AUTO_INDEX;
UInt32 count;
UInt32 status;
UInt16 slave_err;
UInt32 rec_data[15];
UInt16 send_data = 0x62;                /* send 'B' */
Float64 exec_time;                      /* execution time */
/*-----------------------------------------------------*/
void isr_timerA(void)
{
   ts_timestamp_type ts;
```

```
   RTLIB_SRT_ISR_BEGIN();         /* overload check */
   RTLIB_TIC_START();             /* start time measurement */
   ts_timestamp_read(&ts);
   host_service(1, &ts);   /* data acquisition service*/
```

```
   /* request the data for read function with index idx */
   ds1104_slave_dsp_spi_read_request(task_id0, idx0);
   /* read data until communication buffer is empty */
   do
   {
      slave_err = ds1104_slave_dsp_spi_read(task_id0, idx0,
         &count, &status, &rec_data);
   }
   while (slave_err == SLVDSP1104_NO_DATA);
   /*- SPI send -----------------------------------------*/
   /* write data to the communication buffer */
   ds1104_slave_dsp_spi_write(task_id1, idx1, send_data);
   exec_time = RTLIB_TIC_READ() * 1e6;
   RTLIB_SRT_ISR_END();                  /* overload check */
}
/*-----------------------------------------------------*/
void main(void)
{
   /* DS1104 and RTLib1104 initialization */
   init();
   /* init communication with slave_dsp */
   ds1104_slave_dsp_communication_init();
   /* define serial port as master with following parameters*/
   ds1104_slave_dsp_spi_init(task_id0, SLVDSP1104_SPI_MASTER,
      baudrate, SLVDSP1104_SPI_CLKPOL_RISE,
      SLVDSP1104_SPI_CLKPHS_WD, databits);
   /* register the read function in the command table */
   ds1104_slave_dsp_spi_read_register(task_id0, &idx0);
   /* register a write function in the command table */
   ds1104_slave_dsp_spi_write_register(task_id1, &idx1);
   /* start sample rate timer */
   RTLIB_SRT_START(DT, isr_timerA);
```

```
   /* Background tasks */
   while(1)
   {
      RTLIB_BACKGROUND_SERVICE(); /* background service */
   }
}
```

ds1104_slave_dsp_spi_init

Syntax

```
void ds1104_slave_dsp_spi_init(
      Int16 task_id,
      UInt16 spimode,
      UInt32 baudrate,
      UInt16 clk_polarity,
      UInt16 clk_phase,
      UInt16 databits)
```

Include file

`slvdsp1104.h`

Purpose

To initialize the serial peripheral interface.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Serial Peripheral Interface (SPI) (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

spimode Specifies the mode of the serial port. In master mode the data is send on SSIMO pin and latched from SSOMI pin. In slave mode, the data output is on SSOMI pin and the data input on SSIMO pin. The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_SPI_MASTER	Master mode
SLVDSP1104_SPI_SLAVE	Slave mode

baudrate Specifies the baudrate of the serial port within the range 78125 Bd … 2.5 MBd.

clk_polarity Specifies the polarity of the clock signal. The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_SPI_CLKPOL_RISE	Data is output on the rising edge of the clock signal
SLVDSP1104_SPI_CLKPOL_FALL	Data is output on the falling edge of the clock signal

clk_phase Specifies the phase of the clock signal. The following symbols are predefined:

Predefined Symbol	Meaning
SLVDSP1104_SPI_CLKPHS_WOD	Data is output without delay
SLVDSP1104_SPI_CLKPHS_WD	Data is output one half-cycle before the first falling or rising edge

Tip

Using the clock phase with delay, master and slave are able to send and receive data simultaneously.

databits Specifies the number of databits within the range 1 … 8.

Note

If a character is shorter than 8 bits, the transmit data must be written in left justified form and the received data must be read in right justified form.

Return value

None

Related topics

Examples

Example of Using the Serial Peripheral Interface 379

References

ds1104_slave_dsp_spi_read 384
ds1104_slave_dsp_spi_read_register 382
ds1104_slave_dsp_spi_read_request 383
ds1104_slave_dsp_spi_write 387
ds1104_slave_dsp_spi_write_register 386

ds1104_slave_dsp_spi_read_register

Syntax

```
void ds1104_slave_dsp_spi_read_register(
      Int16 task_id,
      Int16 *index)
```

Include file

`slvdsp1104.h`

Purpose

To register the read function in the slave DSP's command table.

Description

The registration of the read function is to be implemented only once within the initialization phase of your application.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Serial Peripheral Interface (SPI) (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address of the command table index. Using DSCOMDEF_AUTO_INDEX for the index, you will get the next free index in the slave DSP's command table.

Return value

None

Related topics

Examples

Example of Using the Serial Peripheral Interface....379

References

ds1104_slave_dsp_spi_init....381
ds1104_slave_dsp_spi_read....384
ds1104_slave_dsp_spi_read_request....383

ds1104_slave_dsp_spi_read_request

Syntax

```
Int16 ds1104_slave_dsp_spi_read_request(
      Int16 task_id,
      Int32 index)
```

Include file

`slvdsp1104.h`

Purpose

To read up to 15 bytes from the serial peripheral interface and store it in the communication buffer at the slave DSP.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Serial Peripheral Interface (SPI) (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the command table index.

Return value

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

Predefined Symbol	Meaning
`SLVDSP1104_NO_ERROR`	No error
`SLVDSP1104_BUFFER_OVERFLOW`	Communication buffer overflow

Related topics

Examples

References

ds1104_slave_dsp_spi_read

Syntax

```
Int16 ds1104_slave_dsp_spi_read(
      Int16 task_id,
      Int32 index,
      UInt32 *count,
      UInt32 *status,
      UInt32 *data)
```

Include file

`slvdsp1104.h`

Purpose

To read the received data from the communication buffer.

Description

Because the master is not able to recognize an incoming byte at the serial peripheral interface, the received bytes are stored temporarily in a slave DSP's

FIFO of 16 byte capacity. Each call of the read function delivers the current content of this FIFO. If the FIFO has overflown (the slave received more than 15 bytes since the last call of the read function), old data has been overwritten and the status bit, which will be sent with the next function call, is set to "1".

Note

Note the following preconditions:

- The read function must be registered within the slave DSP initialization using the ds1104_slave_dsp_spi_read_register function.
- The data to be read must be requested beforehand using the ds1104_slave_dsp_spi_read_request function.

For a demo source code, refer to Example of Using the Serial Peripheral Interface on page 379.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Serial Peripheral Interface (SPI) (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the command table index.

count Specifies the pointer to the variable containing the number of received data bytes.

status Specifies the pointer to the variable containing the status of the slave DSP's FIFO buffer since last reading. The status bit can be set to the following values:

Value	Meaning
0	No overflow
1	Overflow

data Specifies the pointer to the variable containing the array of received data bytes (max. 15 bytes).

Return value

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

Predefined Symbol	Meaning
`SLVDSP1104_NO_ERROR`	No error
`SLVDSP1104_NO_DATA`	There is no current data in the communication buffer. Data has been read from the previous request.
`SLVDSP1104_DATA_LOST`	Data of a previous request has been overwritten. The current request has been performed without error.

Example

If the slave received 18 bytes since the last read operation, the value of the count parameter is “2”, the status is set to “1” and only the bytes 17 and 18 are stored in the data array. The return value is `SLVDSP1104_DATA_LOST`.

Related topics

Examples

References

ds1104_slave_dsp_spi_write_register

Syntax

```
void ds1104_slave_dsp_spi_write_register(
      Int16 task_id,
      Int16 *index)
```

Include file

`slvdsp1104.h`

Purpose

To register the write function in the slave DSP’s command table.

Description

The registration of the write function is to be implemented only once within the initialization phase of your application.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Serial Peripheral Interface (SPI) (DS1104 Features).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the address of the command table index. Using DSCOMDEF_AUTO_INDEX for the index, you will get the next free index in the slave DSP’s command table.

Return value

None

Related topics

Examples

References

ds1104_slave_dsp_spi_write

Syntax

```
Int16 ds1104_slave_dsp_spi_write(
      Int16 task_id,
      Int32 index,
      UInt32 value)
```

Include file

`slvdsp1104.h`

Purpose

To write a byte to the FIFO of the serial peripheral interface.

Note

The write function must be registered once within the slave DSP initialization using the ds1104_slave_dsp_spi_write_register function.
For a demo source code, refer to Example of Using the Serial Peripheral Interface on page 379.

I/O mapping

For information on the I/O mapping, refer to Slave DSP Serial Peripheral Interface (SPI) (DS1104 Features 📖).

Parameters

task_id Specifies the communication channel within the range 0 … 2.

index Specifies the command table index.

value Specifies the data byte to be sent.

Return value

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

Predefined Symbol	Meaning
SLVDSP1104_NO_ERROR	No error
SLVDSP1104_BUFFER_OVERFLOW	Communication buffer overflow

Related topics

Examples

Example of Using the Serial Peripheral Interface........ 379

References

ds1104_slave_dsp_spi_init........ 381
ds1104_slave_dsp_spi_write_register........ 386

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

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 PowerPC processor of DS1104 R&D Controller Board.

Where to go from here

Information in this section

Compiler and C Run-Time Libraries

Compiler and C run-time libraries

The Microtec PowerPC C/C++ Compiler is installed as encrypted archive when you install the Real-Time Interface product set. If you ordered the required licenses, you can decrypt the archive and use the compiler afterwards. After decryption, the compiler is available in `%ProgramData%\dSPACE\<RCPandHIL_InstallationGUID>`.

If you use the Command Prompt for dSPACE RCP and HIL shortcut in the Windows Start menu, the required paths and environment settings for calling the compiler are automatically set.

For information on the compiler, refer to the documentation provided with the Microtec PowerPC C/C++ Compiler.

For information on decrypting, refer to How to Decrypt Encrypted Archives of dSPACE Software Installations (Managing dSPACE Software Installations).

For information on the C++ support, refer to Integrating C++ Code on page 401.

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 DS1104 software is as follows:

Folder	Contents
`<RCP_HIL_InstallationPath>\DS1104\RTLib`	Source and library files of the DS1104 Real-Time Library, makefiles, and linker command file
`<RCP_HIL_InstallationPath>\DS1104\RTLib`	Source and object files of Slave DSP applications, makefiles, and linker command file
`<RCP_HIL_InstallationPath>\DS1104\PAL`	PAL update application
`<RCP_HIL_InstallationPath>\Exe`	Batch files for manually programming the DS1104, host programs
`<RCP_HIL_InstallationPath>\Demos\DS1104`	Demo examples

DS1104 Real-Time Library

DS1104.lib

All functions of the DS1104 Real-Time Library were compiled with the highest optimization level and collected in the library `ds1104.lib`. Required objects from this library are automatically linked to any application when `Ds1104.lk` is used for linking. The header files are located in `<RCP_HIL_InstallationPath>DS1104\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 `ds1104.lib`:

Module (Header File)	Contents
`brtenv.h`	Basic real-time environment
`ds1104.h`	Addresses and error codes
`dsmcom.h`, `dscomdef.h`	General master-slave communication
`dsmsg.h`	Message module support
`dsser.h, dsser1104.h, dsserdef.h`	Serial interface
`dssint.h`, `sint1104.h`	Subinterrupt handling
`dsstd.h`	Standard definitions
`dsstimul.h, dsstimul_msg.h`	Stimulus engine for real-time systems
`dsts.h`, `ts1104.h`	Time-stamping
`dstypes.h`	dSPACE type definitions
`dsvcm.h`, `dsmodule.h`	Version and config section management
`exc1104.h, ppcexc.h`	Exception handling
`hsvc1104.h`	Host service support
`info1104.h`	Information from config section (hardware configuration, ...)
`init1104.h`	Initialization functions
`int1104.h`	Interrupt handling
`io1104.h`	I/O functions
`pm1104.h`	Performance measurement
`ppcstack.h`	Stack control
`ser1104.h`	Serial interface
`slvdsp1104.h`	Slave DSP access functions
`tic1104.h`	Time measurement and delay
`tmr1104.h`	Timer access functions (Timer 0, Timer 1, Timer 2, Timer 3, Decrementer, Timebase)

File Extensions

File extensions

The following file extensions are used:

File Extension	Meaning
`.asm`	Assembler source files generated by a conversion utility such as bin2asm or coffconv
`.c`[1]	C source files
`.lib`	Library files
`.lk`	Linker command file
`.mk`	Makefiles
`.o03`	Relocatable object files for PowerPC 603
`.o24`	Relocatable object files for slave DSP TMS320F240
`.obj`	Executable programs for the slave DSP TMS320F240 and object files for the PowerPC
`.ppc`	Executable programs for the PowerPC
`.s`	Assembler source files

[1] For C++ support, refer to Integrating C++ Code on page 401.

Compiling, Linking and Downloading an Application

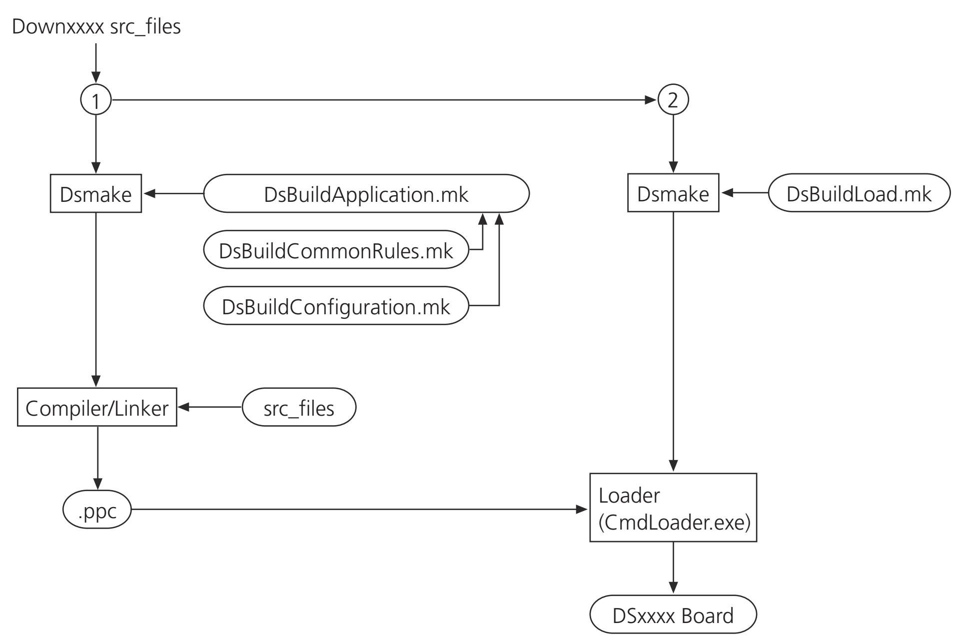
Introduction

If you want to build a user application and download it to the target hardware, you can use the `Down` tool for your board.

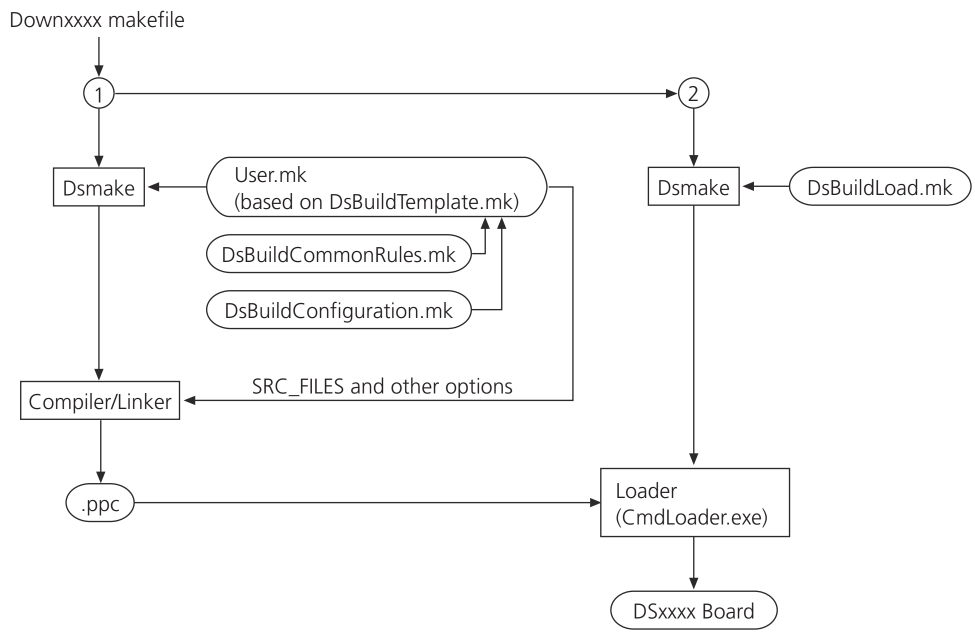
> **Tip**
>
> The executable file Down1104 can be called in a Command Prompt window (DOS window) of your host PC.
> If you use the Command Prompt for dSPACE RCP and HIL shortcut in the Windows Start menu, the required paths and environment settings are automatically set.

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

Down1104.exe

Syntax

```
down1104 file.mk [options] [/?]
```

or

```
down1104 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.

Assembler-coded source file with preprocessing directives (.ss) To specify the file(s) to be assembled and linked using DsBuildTemplate.mk. The resulting program file is named according to the name of the specified makefile.

Assembler-coded source file (.s) To specify the file(s) to be assembled 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, `Down1104` 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 DS1104 platform named 'ds1104'. 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 394.

For further information on the C++ support, refer to Integrating C++ Code on page 401.

Options

The following command line options are available:

Option	Meaning
/ao <option>	To specify additional assembler options; refer to the Microtec PowerPC Assembler documentation.
/co <option>	To specify additional compiler options; refer to the Microtec PowerPC C Compiler documentation.
/d	To disable downloading the application; only compiling and linking.
/g	To compile for source level debugging;`Ds1104dbg.lib` is used for linking.
/l	To write all output to `down1104.log`.
/lib <lib_file>	To specify an additional library to be linked.
/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 ds1104 if you call `Down1104.exe`.
/pause	To pause execution of `Down1104` 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.

Message	Description
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 DOWN1104 version. For example, if you are using DOWN1104 and the working board is of type DS1007, the board name ds1104 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.
ERROR: unable to access file <file name>!	The specified file cannot be accessed by Down1104. 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: ...	Down1104 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!	DOWN1104 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

Description

This makefile is used to compile or assemble the application source files. It is called by Down1104.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 Down1104 or a custom makefile based on DsBuildTemplate.mk instead.

Related topics

References

DsBuildLoad.mk

Description

This file is automatically invoked by Down1104.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

DsBuildTemplate.mk

Description

If you want to call Down1104.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 401.

Related topics

References

Ds1104.lk

Description

This linker command file is automatically used by DsBuildApplication.mk or the custom makefile based on DsBuildTemplate.mk to link DS1104 PowerPC 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

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.o03 MyModule4.cppo03
...
# 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 400.

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 183).

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 148. For detailed information about exceptions, refer to Handling Exceptions (RTI and RTI-MP Implementation Guide).

PPCObjdump

Syntax

```
PPCObjdump [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 PPCObjdump 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\PPCTools\bin`.

Options

The following command line options are available:

Option		Meaning
-a	--archive-headers	Shows object file format and header information from an archive object file.
	--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. PPCObjdump 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:

```
PPCObjdump -S -l -D appl.ppc > result.dmp
```

Numerics

A

B

C

D

E

F

H

I

L

M

O

P