

DS4201-S Serial Interface Board

RTLib Reference

Release 2021-A – May 2021

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

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

© 2001 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

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

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

Contents

About This Reference	5
Macros	7
Base Address of the I/O Board.....	7
Board Initialization	9
ds4201s_init.....	9
Serial Interface Communication	11
Basic Principles of Serial Communication.....	12
Software FIFO Buffer.....	12
Trigger Levels.....	13
How to Handle Subinterrupts in Serial Communication.....	14
Example of a Serial Interface Communication.....	15
Data Types for Serial Communication.....	17
dsser_ISR.....	17
dsser_LSR.....	19
dsser_MSR.....	20
dsser_subint_handler_t.....	21
dsserChannel.....	22
Generic Serial Interface Communication Functions.....	24
dsser_init.....	25
dsser_free.....	26
dsser_config.....	27
dsser_transmit.....	31
dsser_receive.....	32
dsser_receive_term.....	34
dsser_fifo_reset.....	35
dsser_enable.....	36
dsser_disable.....	37
dsser_error_read.....	38
dsser_transmit_fifo_level.....	39
dsser_receive_fifo_level.....	40
dsser_status_read.....	41

dsser_handle_get.....	42
dsser_set.....	43
dsser_subint_handler_inst.....	44
dsser_subint_enable.....	45
dsser_subint_disable.....	46
dsser_word2bytes.....	48
dsser_bytes2word.....	49

Index	51
-------	----

About This Reference

Content

This RTLib Reference (Real-Time Library) gives detailed descriptions of the C functions needed to program a DS4201-S Serial Interface 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
	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
	Indicates a hazard that, if not avoided, could result in property damage.
	Indicates important information that you should take into account to avoid malfunctions.
	Indicates tips that can make your work easier.
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files


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

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

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

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

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

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

Macros

Introduction

The base address of an I/O board in a PHS-bus-based system has to be defined by using the `DSxxxx_n_BASE` macro.

Base Address of the I/O Board

DSxxxx_n_BASE Macros

When using I/O board functions, you always need the board's base address as a parameter. This address can easily be obtained by using the `DSxxxx_n_BASE` macros, where `DSxxxx` is the board name (for example, `DS2001`) and `n` is an index which counts boards of the same type. The board with the lowest base address is given index 1. The other boards of the same type are given consecutive numbers in order of their base addresses.

The macros reference an internal data structure which holds the addresses of all I/O boards in the system. The initialization function of the processor board (named `init`) creates this data structure. Hence, when you change an I/O board base address, it is not necessary to recompile the code of your application. For more information on the processor board's initialization function, refer to [ds1006_init \(DS1006 RTLib Reference\)](#) or [init \(DS1007 RTLib Reference\)](#).

Note

The `DSxxxx_n_BASE` macros can be used only after the processor board's initialization function `init` is called.

Example

This example demonstrates the use of the `DSxxxx_n_BASE` macros. There are two `DS2001` boards, two `DS2101` boards, and one `DS2002` board connected to a PHS bus. Their base addresses have been set to different addresses. The following table shows the I/O boards, their base addresses, and the macros which can be used as base addresses:

Board	Base Address	Macro
DS2001	00H	DS2001_1_BASE
DS2002	20H	DS2002_1_BASE
DS2101	80H	DS2101_1_BASE
DS2001	90H	DS2001_2_BASE
DS2101	A0H	DS2101_2_BASE

Board Initialization

Introduction

Before you can use the DS4201-S board, you have to perform the initialization process.

Note

The initialization function of the processor board must be called before the DS4201-S board's initialization function.

ds4201s_init

Syntax

```
void ds4201s_init(phs_addr_t base)
```

Include file

ds4201.h

Purpose

To initialize the DS4201-S board.

Description

All DS4201-S registers are initialized to default values.

Note

This function must be called before any of the DS4201-S functions can be used.

I/O mapping

For details on the I/O mapping, refer to [Mapping of I/O Signals \(PHS Bus System Hardware Reference !\[\]\(1f56542a42e2413e44a2b2023033aa2e_img.jpg\)](#)).

Parameters **base** Specifies the PHS-bus base address. Refer to [Base Address of the I/O Board](#) on page 7.

Return value None

Messages The following messages are defined:

ID	Type	Message	Description
201	Error	ds4102s_init(): Invalid PHS-bus base address 0x????????	The value of the base parameter is not a valid PHS-bus address. This error may be caused if the PHS-bus connection of the I/O board is missing. Check the connection.
-180	Error	ds4201s_init(0x?): Board not found!	No DS4201-S board could be found at the specified PHS-bus base address. Check if the DSxxxx_n_BASE macro corresponds to the I/O board used.
-53	Warning	ds4201s_init(0x?): Jumper setup is not matching SW default initialization! STP register: 0x???????? instead of 0x????????	The value of the STP register could not be verified because the jumper setting is not correct. If you have not changed the jumper settings a hardware failure could cause this message.

Related topics

References

Base Address of the I/O Board	7
Macros	7

Serial Interface Communication

Introduction

This section contains the generic functions for communication via a serial interface.

The DS4201-S Serial Interface Board provides 4 serial communication channels with selectable line transceivers (RS232, RS422 or RS485).

Note

You have to initialize the DS4201-S with the `ds4201s_init` function before you can use one of these functions.

Where to go from here

Information in this section

Basic Principles of Serial Communication.....	12
Data Types for Serial Communication.....	17
Generic Serial Interface Communication Functions.....	24

Basic Principles of Serial Communication

Where to go from here

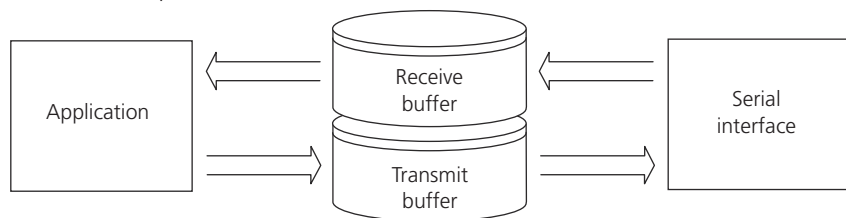
Information in this section

Software FIFO Buffer.....	12
To get information about the receive and transmit buffers.	
Trigger Levels.....	13
To get information about the trigger levels.	
How to Handle Subinterrupts in Serial Communication.....	14
Instructions on handling subinterrupts in serial communication.	
Example of a Serial Interface Communication.....	15
Shows you how to implement serial interface communication.	

Software FIFO Buffer

Introduction

The software FIFO buffer is a memory section that provides the UART with additional space for data storage and ensures that the generic functions are hardware-independent.



The software FIFO buffer stores data that will be written to (transmit buffer) or has been read by (receive buffer) the UART.

Buffer size

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

Transmit buffer

The transmit buffer is filled with data to be sent as long as free space is available. It cannot be overwritten. You can write data to the transmit buffer with the function `dsrser_transmit`.

Receive buffer

The receive buffer is filled with data received by the UART as long as free space is available. If an overflow occurs, old data in the receive buffer is overwritten or new data is rejected. This depends on the mode of the FIFO. You can access the

receive buffer by using the functions `ds-ser_receive` and `ds-ser_receive_term`.

Related topics

Basics	
Trigger Levels.....	13
References	
ds-ser_receive.....	32
ds-ser_receive_term.....	34
ds-ser_transmit.....	31

Trigger Levels

Introduction	Two different trigger levels can be configured.
UART trigger level	The UART trigger level is hardware-dependent. After the specified number of bytes is received, the UART generates an interrupt and the bytes are copied into the receive buffer.
User trigger level	The user trigger level is hardware-independent and can be adjusted in smaller or larger steps than the UART trigger level. After a specified number of bytes is received in the receive buffer, the subinterrupt handler is called.

Related topics

Basics	
Basic Principles of Serial Communication.....	12
HowTos	
How to Handle Subinterrupts in Serial Communication.....	14

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 13).
DSSER_TX_FIFO_EMPTY_SUBINT	Generated when the transmit buffer has no data.
DSSER_RECEIVER_LINE_SUBINT	Line status interrupt provided by the UART.
DSSER_MODEM_STATE_SUBINT	Modem status interrupt provided by the UART.
DSSER_NO_SUBINT	Generated after the last subinterrupt. This subinterrupt tells your application that no further subinterrupts were generated.

Method

To install a subinterrupt handler within your application

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

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

- 2 Initialize your subinterrupt handler:

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

- 3 Enable the required subinterrupts:

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

Related topics

Basics

[Trigger Levels.....](#) 13

References

[dsSer_subint_enable.....](#) 45
[dsSer_subint_handler_inst.....](#) 44
[dsSer_subint_handler_t.....](#) 21
[dsSerChannel.....](#) 22

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();
    ds4201s_init(DS4201S_1_BASE);

    /* allocate a new 1024 byte SW-FIFO */
    serCh = dsSer_init(DS4201S_1_BASE, 0, 1024);
    dsSer_subint_handler_inst(serCh,
        (dsSer_subint_handler_t)callback);
}
```

```
dsser_subint_enable(serCh,  
    DSSER_TRIGGER_LEVEL_SUBINT_MASK |  
    DSSER_TX_FIFO_EMPTY_SUBINT_MASK);  
/* config and start the UART */  
dsser_config(serCh, DSSER_FIFO_MODE_OVERWRITE,  
    9600, 8, DSSER_1_STOPBIT, DSSER_NO_PARITY,  
    DSSER_14_BYTE_TRIGGER_LEVEL, 32, DSSER_RS232);  
RTLIB_INT_ENABLE();  
for(;;)  
{  
    RTLIB_BACKGROUND_SERVICE();  
}
```


Data Types for Serial Communication

Introduction

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

Where to go from here	Information in this section
	dsser_ISR..... 17 Provides information about the interrupt identification register.
	dsser_LSR..... 19 Provides information about the status of data transfers.
	dsser_MSR..... 20 Provides information about the state of the control lines.
	dsser_subint_handler_t..... 21 Provides information about the subinterrupt handler.
	dsserChannel..... 22 Provides information about the serial channel.

dsser_ISR

Syntax

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

Include file

dsserdef.h

Description

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

Note

The data type contains the value of the UART's register.
The register conforms to a standard 16550 UART such as the TEXAS INSTRUMENTS PC16550D. 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, PC16550D*.

Related topics**References**

[dsser_status_read.....](#) 41

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 PC16550D. 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, PC16550D*.

Related topics

References

[dsser_status_read](#)..... 41

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 PC16550D. 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, PC16550D*.

Related topics**References**

[dsser_status_read](#)..... 41

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 14 or [dsser_subint_handler_inst](#) on page 44).

Members

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 25).

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 13).

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..... 13](#)

References

[dsser_init..... 25](#)

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 17.
 - lineStatusReg** Line status register. Refer to [dsser_LSR](#) on page 19.
 - modemStatusReg** Modem status register. Refer to [dsser_MSR](#) on page 20.

Related topics

References	
dsser_status_read	41

Generic Serial Interface Communication Functions

Where to go from here

Information in this section

dsser_init	25
To initialize the serial interface and install the interrupt handler.	
dsser_free	26
To close a serial interface.	
dsser_config	27
To configure and start the serial interface.	
dsser_transmit	31
To transmit data through the serial interface.	
dsser_receive	32
To receive data through the serial interface.	
dsser_receive_term	34
To receive data through the serial interface.	
dsser_fifo_reset	35
To reset the serial interface.	
dsser_enable	36
To enable the serial interface.	
dsser_disable	37
To disable the serial interface.	
dsser_error_read	38
To read an error flag of the serial interface.	
dsser_transmit_fifo_level	39
To get the number of bytes in the transmit buffer.	
dsser_receive_fifo_level	40
To get the number of bytes in the receive buffer.	
dsser_status_read	41
To read the value of one or more status registers and store the values in the appropriate fields of the channel structure.	
dsser_handle_get	42
To check whether the serial interface is in use.	
dsser_set	43
To set a property of the UART.	
dsser_subint_handler_inst	44
To install a subinterrupt handler for the serial interface.	
dsser_subint_enable	45
To enable one or several subinterrupts of the serial interface.	

dsser_subint_disable	46
To disable one or several subinterrupts of the serial interface.	
dsser_word2bytes	48
To convert a word (max. 4 bytes long) into a byte array.	
dsser_bytes2word	49
To convert a byte array with a maximum of 4 elements into a single word.	

dsser_init

Syntax

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

Include file

dsser.h

Purpose

To initialize the serial interface and install the interrupt handler.

Note

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

Parameters

base Specifies the base address of the serial interface. This value has to be set to DS4201S_y_BASE, with y as a consecutive number within the range of 1 ... 16. For example, if there is only one DS4201S board, use DS4201S_1_BASE.

channel Specifies the number of the channel to be used for the serial interface. The permitted values are within the range 0 ... 3.

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

Return value

This function returns the pointer to the serial channel structure.

Messages

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

ID	Type	Message	Description
100	Error	x, ch=y, Board not found!	I/O board was not found.
101	Warning	x, ch=y, Mixed usage of high and low level API!	It is not allowed to use the generic functions (high-level access functions) and the low-level access functions of the serial interface on the same channel. It is recommended to use only the generic functions.
501	Error	x, ch=y, memory: Allocation error on master.	Memory allocation error. No free memory on the master.
508	Error	x, ch=y, channel: out of range!	The <code>channel</code> parameter is out of range.
700	Error	x, ch=y, Buffersize: Illegal	The <code>fifo_size</code> parameter is out of range.

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 12

Examples

[Example of a Serial Interface Communication.....](#) 15

References

[Data Types for Serial Communication.....](#) 17
[dsr_config.....](#) 27
[dsr_free.....](#) 26

dsr_free

Syntax

```
Int32 dsr_free(dsrChannel*serCh)
```

Include file

`dsr.h`

Purpose

To close a serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsr_init](#) on page 25).

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

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

Related topics

Basics

[Basic Principles of Serial Communication..... 12](#)

References

[dsser_init..... 25](#)

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 25).

fifo_mode Specifies the mode of the receive buffer (see [Software FIFO Buffer](#) on page 12):

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:

The serial interface of the DS4201-S can be driven by an oscillator with a frequency up to $f_{osc} = 24$ MHz. The baud rate range depends on the selected transceiver mode and the oscillator frequency.

Note

You have to ensure that the specified transceiver type is installed. For more information, refer to [Component Settings for Transceiver Setup \(PHS Bus System Hardware Reference\)](#).

For an example, the following table shows the baud rate ranges for two different oscillator frequencies:

Mode	Baud Rate Range ($f_{osc} = 1.8432$ MHz)	Baud Rate Range ($f_{osc} = 24$ MHz)
RS232	5 ... 115,200 baud	5 ... 115,200 baud
RS422	5 ... 115,200 baud	5 ... 1,500,000 baud
RS485	5 ... 115,200 baud	5 ... 1,500,000 baud

You can specify any baud rate in the range listed above. However, the baud rate actually used by the DS4201-S depends on the oscillator frequency f_{osc} since the baud rate is a fraction of f_{osc} .

The available baud rates can be calculated according to

$$f = f_{osc} / 16 \cdot n,$$

where n is a positive integer.

When you specify a baud rate, the closest available baud rate is actually used for serial communication. For example, if you specify 70,000 baud as the baud rate,

the baud rate actually used is 57,600 baud ($f_{osc} = 1.8432$ MHz) or 71,429 baud ($f_{osc} = 24$ MHz).

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 13). 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 13):

Value	Meaning
DSSER_DEFAULT_TRIGGER_LEVEL	Synchronizes the UART trigger level and the user trigger level.
1 ... (<code>fifo_size - 1</code>)	Sets the user trigger level.

Value	Meaning
DSSER_TRIGGER_LEVEL_DISABLE	No receive subinterrupt handling for the serial interface

uart_mode Sets the mode of the UART transceiver.

The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_RS232	RS232 mode
DSSER_RS422	RS422 mode
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.
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.....](#) 12

Examples

[Example of a Serial Interface Communication.....](#) 15

References

[dsser_init.....](#) 25

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 25).

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.

Predefined Symbol	Meaning
DSSER_COMMUNICATION_FAILED	<p>The function failed with no effect on the input or output data. No data is written to the FIFO.</p> <p>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.</p>

Example

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

```

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

```

Related topics**Basics**

[Basic Principles of Serial Communication..... 12](#)

Examples

[Example of a Serial Interface Communication..... 15](#)

References

[dsser_init..... 25](#)
[dsser_transmit_fifo_level..... 39](#)

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 25).

datalen Specifies the number of data bytes to be read. The value must not be greater than the FIFO size defined with [dsser_init](#).

data Specifies the pointer to the destination buffer.

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

Return value

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

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_NO_DATA	No new data is read from the FIFO.
DSSER_FIFO_OVERFLOW	The FIFO is filled. The behavior depends on the <code>fifo_mode</code> adjusted with dsser_config : <ul style="list-style-type: none"> ▪ <code>fifo_mode = DSSER_FIFO_MODE_BLOCKED</code> Not all new data could be placed in the FIFO. ▪ <code>fifo_mode = DSSER_FIFO_MODE_OVERWRITE</code> The old data is rejected.
DSSER_COMMUNICATION_FAILED	The function failed with no effect on the input or output data. No data is read from the FIFO. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Example

The following example shows how to receive 4 bytes.

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

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 12

Examples

[Example of a Serial Interface Communication.....](#) 15

References

[dsser_init.....](#) 25

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 25).

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

data Specifies the pointer to the destination buffer.

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

term Specifies the character that terminates the reception of bytes.

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

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_NO_DATA	No new data is read from the FIFO.
DSSER_FIFO_OVERFLOW	The FIFO is filled. The behavior depends on the <code>fifo_mode</code> adjusted with <code>dsser_config</code> : <ul style="list-style-type: none"> ▪ <code>fifo_mode = DSSER_FIFO_MODE_BLOCKED</code> Not all new data could be placed in the FIFO. ▪ <code>fifo_mode = DSSER_FIFO_MODE_OVERWRITE</code> The old data is rejected.
DSSER_COMMUNICATION_FAILED	The function failed with no effect on the input or output data. No data is read from the FIFO. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

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

```
UInt8 data[4];
UInt32 count;
Int32 error;
error = dsser_receive_term(serCh, 4, data, &count, '\r');
```

Related topics

Basics

[Basic Principles of Serial Communication..... 12](#)

References

[dsser_init..... 25](#)

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 25).

Return value

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

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

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 12

References

[dsser_enable.....](#) 36
[dsser_init.....](#) 25

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 25).
Return value	This function returns an error code. The following symbols are predefined:
Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 12

References

[dsser_disable.....](#) 37
[dsser_init.....](#) 25

dsser_disable

Syntax	<code>Int32 dsser_disable(const dsserChannel* serCh)</code>
Include file	<code>dsser.h</code>
Purpose	To disable the serial interface.
Description	The serial interface stops transmitting data, incoming data is no longer stored 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 25).

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

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

Related topics

Basics

[Basic Principles of Serial Communication..... 12](#)

References

[dsser_enable..... 36](#)
[dsser_init..... 25](#)

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 25).

Return value

This function returns an error flag.

The following symbols are predefined:

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

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 12

References

[dsser_config.....](#) 27
[dsser_init.....](#) 25

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 25).

Return value

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

Related topics**Basics**[Basic Principles of Serial Communication.....](#) 12**References**[dsser_init.....](#) 25
[dsser_receive_fifo_level.....](#) 40

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 25).**Return value**

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

Related topics**Basics**[Basic Principles of Serial Communication.....](#) 12**References**[dsser_init.....](#) 25
[dsser_transmit_fifo_level.....](#) 39

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 25).

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

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

Return value

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

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

Example

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

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

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 12

References

[dsser_init.....](#) 25
[dsser_ISR.....](#) 17
[dsser_LSR.....](#) 19
[dsser_MSR.....](#) 20

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 DS4201S_y_BASE, with y as a consecutive number within the range of 1 ... 16. For example, if there is only one DS4201S board, use DS4201S_1_BASE.

channel Specifies the number of the channel to be used for the serial interface. The permitted values are within the range 0 ... 3.

Return value

This function returns:

- NULL if the specified serial interface is not used.
- A pointer to the serial channel structure of the serial interface that has been created by using the **dsser_init** function.

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 12

References

[dsser_init.....](#) 25

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 DS4201-S 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 25).

type Specifies the property to be changed (`DSSER_SET_UART_FREQUENCY`).

value_p Specifies the pointer to a UInt32-variable with the new value, for example, a variable which contains the quartz frequency.

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

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

Example

This example sets a new value for the frequency.

```
UInt32 freq = 1843200;           /* 1.8432 MHz */
Int32 error;
error = dsser_set(serCh, DSSER_SET_UART_FREQUENCY, &freq);
```

Related topics

Basics

[Basic Principles of Serial Communication..... 12](#)

References

[dsser_config..... 27](#)
[dsser_init..... 25](#)

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	<p>After installing the handler, the specified subinterrupt type must be enabled (see dsser_subint_enable on page 45).</p> <div>Note<p>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.</p></div>
Parameters	<p>serCh Specifies the pointer to the serial channel structure (see dsser_init on page 25).</p> <p>subint_handler Specifies the pointer to the subinterrupt handler.</p>
Return value	<p>This function returns the pointer to the previously installed subinterrupt handler.</p>
Related topics	<p>Basics</p> <div>Basic Principles of Serial Communication..... 12</div> <p>Examples</p> <div>Example of a Serial Interface Communication..... 15</div> <p>References</p> <div>dsser_init..... 25 dsser_subint_disable..... 46 dsser_subint_enable..... 45</div>

dsser_subint_enable

Syntax	<pre>Int32 dsser_subint_enable(dsserChannel* serCh, const UInt8 subint)</pre>
Include file	<code>dsser.h</code>
Purpose	<p>To enable one or several subinterrupts of the serial interface.</p>

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 25).

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 13)
DSSER_TX_FIFO_EMPTY_SUBINT_MASK	Interrupt triggered when the transmit buffer is empty
DSSER_RECEIVER_LINE_SUBINT_MASK	Line status interrupt of the UART
DSSER_MODEM_STATE_SUBINT_MASK	Modem status interrupt of the UART

Return value

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

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

Related topics**Basics**

[Basic Principles of Serial Communication..... 12](#)

Examples

[Example of a Serial Interface Communication..... 15](#)

References

[dsser_init..... 25](#)
[dsser_subint_disable..... 46](#)
[dsser_subint_handler_inst..... 44](#)

dsser_subint_disable

Syntax

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

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

Purpose	To disable one or several subinterrupts of the serial interface.
----------------	--

Parameters	<p>serCh Specifies the pointer to the serial channel structure (see dsser_init on page 25).</p> <p>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:</p>
-------------------	---

Predefined Symbol	Meaning
DSSER_TRIGGER_LEVEL_SUBINT_MASK	Interrupt triggered when the user trigger level is reached (see Trigger Levels on page 13)
DSSER_TX_FIFO_EMPTY_SUBINT_MASK	Interrupt triggered when the transmit buffer is empty
DSSER_RECEIVER_LINE_SUBINT_MASK	Line status interrupt of the UART
DSSER_MODEM_STATE_SUBINT_MASK	Modem status interrupt of the UART

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

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

Related topics

Basics

[Basic Principles of Serial Communication](#)..... 12

References

[dsser_init](#)..... 25
[dsser_subint_enable](#)..... 45
[dsser_subint_handler_inst](#)..... 44

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.....](#) 12

References

[dsser_bytes2word.....](#) 49
[dsser_transmit.....](#) 31
[dsser_transmit_fifo_level.....](#) 39

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(dsserChannel1* serCh, UInt32* word_p, UInt32* count)
{
    UInt8 bytes[4];
```

```
if(dsser_receive_fifo_level(serCh) > 3)
{
    dsser_receive(serCh, 4, bytes, count);
    word_p = dsser_bytes2word(bytes, word_p, 4);
}
else
{
    *count = 0;
}
}
```

Use of the function:

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

Related topics

Basics	
Basic Principles of Serial Communication.....	12
References	
dsser_receive.....	32
dsser_receive_fifo_level.....	40
dsser_word2bytes.....	48

B

base address 7

C

Common Program Data folder 6

D

data type

- dserr_ISR 17
- dserr_LSR 19
- dserr_MSR 20
- dserr_subint_handler_t 21
- dserrChannel 22

Documents folder 6

dserr_bytes2word 49

dserr_config 27

dserr_disable 37

dserr_enable 36

dserr_error_read 38

dserr_fifo_reset 35

dserr_free 26

dserr_handle_get 42

dserr_init 25

dserr_ISR 17

dserr_LSR 19

dserr_MSR 20

dserr_receive 32

dserr_receive_fifo_level 40

dserr_receive_term 34

dserr_set 43

dserr_status_read 41

dserr_subint_disable 46

dserr_subint_enable 45

dserr_subint_handler_inst 44

dserr_subint_handler_t 21

dserr_transmit 31

dserr_transmit_fifo_level 39

dserr_word2bytes 48

dserrChannel 22

DSxxxx_n_BASE 7

L

Local Program Data folder 6

R

receive buffer 12

S

serial interface communication 11

subinterrupt

- serial communication 14

T

transmit buffer 12

trigger level 13

U

UART 11

