# I2C/TWI Driver

1.0

Generated by Doxygen 1.9.5

# Chapter 1

# File Index

## 1.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 2

# File Documentation

## 2.1 I2C_1.h File Reference

```
#include <avr/io.h>
```

### Functions

- void setSCLFrequency (unsigned char bitRateRegister)
- void enableTWI ()
- void disableTWI ()
- void setDeviceAddress (char addr, char generalCall)
- void I2CInit ()
- char initiateWrite (char addr)
- char burstWrite (char *data, char n)
- char initiateRead (char addr)
- char burstRead (char *data, char n)
- void stopTransmission ()

### 2.1.1 Detailed Description

I2C/TWI Driver. This is a driver for the I2C/TWI peripheral on the AVR 8 bit ATMega32 family of microcontrollers.

### 2.1.2 Function Documentation

#### 2.1.2.1 setSCLFrequency()

```
void setSCLFrequency (
            unsigned char bitRateRegister )
```

This function is used to set the frequency of the SCL line according to the value of the bitRateRegister.

**Parameters**

| | |
|---|---|
| *bitRateRegister* | This parameter is used to set the SCL frequency and should be chosen according to the following equation: $$\text{bitRateRegister} = \frac{(\frac{Foscillator}{Fdesired} - 16)}{8}$$ |

**Examples**

I2C_SetupExample.cpp.

#### 2.1.2.2 enableTWI()

```
void enableTWI ( )
```

This function will enable the TWI and only needs to be called once in the setup.

**Examples**

I2C_SetupExample.cpp.

#### 2.1.2.3 disableTWI()

```
void disableTWI ( )
```

This function will disable the TWI.

#### 2.1.2.4 setDeviceAddress()

```
void setDeviceAddress (
            char addr,
            char generalCall )
```

This function sets the I2C address and general call recognition for the ATMega32 microcontroller.

**Parameters**

| | |
|---|---|
| *addr* | This parameter represents the 7 bit I2C address. |
| *generalCall* | This parameter sets the microcontroller's response to general calls made on the I2C bus where 1 means that the microcontroller will respond to general calls and 0 it will ignore them. |

**Examples**

I2C_SetupExample.cpp.

#### 2.1.2.5 I2CInit()

```
void I2CInit ( )
```

This function initializes the I2C peripheral with some default settings where: F_SCL = 400Khz, Device address = 0x32, and General calls are ignored.

**Examples**

I2C_CommunicationExample.cpp, and I2C_SetupExample.cpp.

#### 2.1.2.6 initiateWrite()

```
char initiateWrite (
            char addr )
```

This function will try to take control of the I2C bus and make a write request to the address specified.

**Parameters**

| | |
|---|---|
| *addr* | This parameter represents the 7 bit I2C address of the target device. |

**Returns**

Will return 1 if it was successful and 0 otherwise.

**Examples**

I2C_CommunicationExample.cpp.

#### 2.1.2.7 burstWrite()

```
char burstWrite (
            char * data,
            char n )
```

This function will load n bytes into the TWDR serially till all the data specified has been written to the target device.

**Parameters**

| | |
|---|---|
| *data* | This parameter is a pointer to the start of the array of bytes that will be sent. |
| *n* | This parameter represents the number of bytes to send and should be less than or equal to the length of the byte array. |

**Attention**

This function must be preceded by a call to initiateWrite(char addr).

**Returns**

Will return 1 if it was successful and 0 otherwise.

**See also**

initiateWrite(char addr)

**Examples**

I2C_CommunicationExample.cpp.

#### 2.1.2.8 initiateRead()

```
char initiateRead (
            char addr )
```

This function will try to take control of the I2C bus and make a read request from the address specified.

**Parameters**

| | |
|---|---|
| *addr* | This parameter represents the 7 bit I2C address of the target device. |

**Returns**

Will return 1 if it was successful and 0 otherwise.

**Examples**

I2C_CommunicationExample.cpp.

#### 2.1.2.9 burstRead()

```
char burstRead (
            char * data,
            char n )
```

This function will read n bytes from the TWDR serially till all the data specified has been read from the target device.

**Parameters**

| | |
|---|---|
| *data* | This parameter is a pointer to the start of the array of bytes that will be used to store the incoming data. |
| *n* | This parameter represents the number of bytes to read and should be less than or equal to the length of |
| | the byte array. |

**Attention**

This function must be preceded by a call to initiateRead(char addr).

**Returns**

Will return 1 if it was successful and 0 otherwise.

**See also**

initiateRead(char addr)

**Examples**

I2C_CommunicationExample.cpp.

#### 2.1.2.10 stopTransmission()

```
void stopTransmission ( )
```

This function is used to trigger the I2C stop signal which ends the ongoing communication and frees the I2C bus.

**Examples**

I2C_CommunicationExample.cpp.

## 2.2 I2C_1.h

Go to the documentation of this file.

```
1
6 #ifndef I2C_1_H
7 #define I2C_1_H
8
9 #include <avr/io.h>
10
19 void setSCLFrequency (unsigned char bitRateRegister);
20
24 void enableTWI ();
25
29 void disableTWI ();
30
37 void setDeviceAddress(char addr, char generalCall);
38
43 void I2CInit();
44
50 char initiateWrite (char addr);
51
62 char burstWrite(char *data, char n);
63
69 char initiateRead (char addr);
70
81 char burstRead(char* data, char n);
82
86 void stopTransmission();
87
96 #endif
```

# Chapter 3

# Example Documentation

## 3.1 I2C_SetupExample.cpp

An example on how to initialize the TWI before starting communication.

```
#include "I2C_1.h"
int main(void)
{
    /* Initialize with default settings */
    I2CInit();

    /* Or initialize with custom settings */
    enableTWI();
    setDeviceAddress(0x52, 1); // 7 bit address = 0x52 and General call recognition is enabled.
    setSCLFrequency(18); // SCL Frequency = 100Khz.
}
```

## 3.2 I2C_CommunicationExample.cpp

An example of how to use the I2C driver for readings and writing bytes from and to a target device.

```
#include "I2C_1.h"
#define TARGET_ADDRESS 0x34
int main(void)
{
    char status = 0;
    char dataOut[3] = {'a', 'r', 'm'};
    char dataIn;

    /* Initialize TWI */
    I2CInit();

    /* Write 3 bytes to the target device */
    status = initiateWrite(TARGET_ADDRESS);
    if(status)
    {
        status = burstWrite(dataOut, 3);
    }

    /* Read 1 byte from the target device */
    status = initiateRead(TARGET_ADDRESS); // Initiates a repeated start without releasing control of the
      I2C bus.
    if(status)
    {
        status = burstRead(&dataIn, 1);
    }
    stopTransmission();
```

# Index