Smart home

1.0

Generated by Doxygen 1.8.20

Chapter 1

ESP8266 WiFi module

Thingspeak example

Here is an example of the AT commands that we send to the module. More documentation can be found in the datasheet ESP8266_AT Instruction set V1.3

Connect to wifi

AT+CWMODE=3 // softAP+station mode AT+CWJAP="MyFi","nicolai5"

Send data

AT+CIPSTART="TCP","api.thingspeak.com",80 AT+CIPSEND=48 GET /update?api_key=9XZN1CY19NFLB7↩ RB&field1=32

Start Server

AT+CWMODE=3 // softAP+station mode AT+CIPMUX=1 // Enable multiple connections page 53 AT+CIPSERV↩ ER=1,80 //Start TCP server on port 80 page 54 AT+CWSAP? // Lists SSID page 27 FaryLink_46D8FA AT+CIFSR // Lists IP look at CIFSR:APIP page page 52

Sending data to the device

To send commands to the wifi module, you start with sending 1 byte specifying the register address you want to communicate with.

After that you send the data bytes that are defined by the register, this is being terminated by sending 0xFF. Which then ends the transmission.

example

Address	data byte 0	data byte 1	data byte 2	data byte 3	data byte 4	data byte 5	data byte 6	data byte 7	data byte 8	data byte 9	data byte 10	data byte 11	data byte 12	Terminator
0x00	M	y	F	i	,	n	i	c	o	l	a	i	5	0x←FF

WIFI connection (SSID, password)

- address: 0x00 First you send the SSID follow by a comma 0x2C and then followed by the password, until termination byte 0xFF

Clock set

- address: 0x01 This only has 7 bytes you can send, in this order hour, minute, second, day, date, month, year, followed by the termination byte 0xFF so a total of 9 bytes including the address and termination byte

Temperature limit

- address: 0x02 This only has 1 byte of data you can send, which is the temperature threshold, before it turns the fan on. So 3 bytes in total should be send including the address and termination byte

Fan strength

- address: 0x03 This only has 1 byte of data you can send, which is the duty cycle that the fan runs on. So 3 bytes in total should be send including the address and termination byte

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 WIFI Library

ESP8266 WIFI library.

Macros

- #define F_CPU 16000000UL
- #define ESP8266_H_
- #define SREG _SFR_IO8(0x3F)
- #define DEFAULT_BUFFER_SIZE 160
- #define DEFAULT_TIMEOUT 10000
- #define SINGLE 0
- #define MULTIPLE 1
- #define NORMAL 0
- #define TRANSPERANT 1
- #define STATION 1
- #define ACCESSPOINT 2
- #define BOTH_STATION_AND_ACCESPOINT 3
- #define DOMAIN "api.thingspeak.com"
- #define PORT "80"
- #define API_WRITE_KEY "9XZN1CY19NFLB7RB"
- #define CHANNEL_ID "1217032"
- #define SSID "OP6T"
- #define PASSWORD "11111111"

Enumerations

- enum ESP8266_RESPONSE_STATUS {
  ESP8266_RESPONSE_WAITING, ESP8266_RESPONSE_FINISHED, ESP8266_RESPONSE_TIMEOUT,
  ESP8266_RESPONSE_BUFFER_FULL,
  ESP8266_RESPONSE_STARTING, ESP8266_RESPONSE_ERROR }

  ESP8266_RESPONSE_STATUS is the status that the ESP8266, can respond with, these are being used for every connection with the device.

- enum ESP8266_CONNECT_STATUS {
ESP8266_CONNECTED_TO_AP, ESP8266_CREATED_TRANSMISSION, ESP8266_TRANSMISSION_DISCONNECTED, ESP8266_NOT_CONNECTED_TO_AP,
ESP8266_CONNECT_UNKNOWN_ERROR }

  ESP8266_CONNECT_STATUS is the status that the ESP8266, can be in describing weather or not is is connect to an Access point.
- enum ESP8266_JOINAP_STATUS {
ESP8266_WIFI_CONNECTED, ESP8266_CONNECTION_TIMEOUT, ESP8266_WRONG_PASSWORD, ESP8266_NOT_FOUND_TARGET_AP,
ESP8266_CONNECTION_FAILED, ESP8266_JOIN_UNKNOWN_ERROR }

  ESP8266_JOINAP_STATUS is the status that the ESP8266, responds with when you connect to an Access point.

Functions

- void Read_Response (char ∗_Expected_Response)
- void ESP8266_Clear ()
- void Start_Read_Response (char ∗_ExpectedResponse)
- void GetResponseBody (char ∗Response, uint16_t ResponseLength)
- bool WaitForExpectedResponse (char ∗ExpectedResponse)

  WaitForExpectedResponse is used after the SendATandExpectResponse, its used to wait for a certain string being send by the ESP8266.
- bool SendATandExpectResponse (char ∗ATCommand, char ∗ExpectedResponse)

  SendATandExpectResponse sends a certain AT command to the ESP8266 and there after uses the WaitFor←ExpectedResponse to wait until a certain message has been send by the ESP8266.
- bool ESP8266_ApplicationMode (uint8_t Mode)

  ESP8266_ApplicationMode Sets the transmission mode that should be used can be either NORMAL or TRANSP←ERANT check datasheet page 55. Calls the AT+CIPMODE command to the ESP8266.
- bool ESP8266_ConnectionMode (uint8_t Mode)

  ESP8266_ConnectionMode Set the connection mode that should be used This decides how many connections, there are allowed to be. If hosting your own server on the device this need to be set to MULTIPLE, otherwise just use SINGLE, as long as you are sure only one request is running at a time. Page 53 Calls the AT+CIPMUX command to the ESP8266.
- bool ESP8266_Begin ()
- bool ESP8266_Close ()
- bool ESP8266_WIFIMode (uint8_t _mode)

  ESP8266_WIFIMode sets the wifi mode that should be used This decides weather you want to act as its own AP or connect to an already existing one. You can also use it as both at the same time. STATION if you want to connect to WIFI ACCESSPOINT if you want it to act as an Access point BOTH_STATION_AND_ACCESPOINT is you want it to act as both Page 19 Calls the AT+CWMODE command to the ESP8266.
- bool ESP8266_StartServer (uint8_t _port)

  ESP8266_StartServer starts a TCP server on a given Port This starts the internal TCP server on a given port page 54 Calls the AT+CIPSERVER command to the ESP8266.
- bool ESP8266_StopServer ()

  ESP8266_StopServer stops the started TCP server page 54 Calls the AT+CIPSERVER command to the ESP8266.
- uint8_t ESP8266_JoinAccessPoint (char ∗_SSID, char ∗_PASSWORD)

  ESP8266_JoinAccessPoint joins the given access point This is used to connect to a certain wifi page 22 Calls the AT+CWJAP,"_SSID","_PASSWORD".
- uint8_t ESP8266_connected ()
- uint8_t ESP8266_Start (uint8_t _ConnectionNumber, char ∗Domain, char ∗Port)

  ESP8266_Start connects to a certain IP and port over TCP Connect to a given TCP server page 46 Calls the AT+←CIPSTaRT="TCP","Domain",Port.
- uint8_t ESP8266_Send (char ∗Data)

  ESP8266_Send sends data to a previously established connection Sends the given data to the already established conenction. Takes the data and calculates the byte length, this is then sent to the AT+CIPSEND command. After that it sends the given data page 47.

- int16_t ESP8266_DataAvailable ()
- uint8_t ESP8266_DataRead ()
- uint16_t Read_Data (char ∗_buffer)

5.1.1 Detailed Description

ESP8266 WIFI library.

```
#include <ESP8266.h>
```

Connect the ESP8266 device using the following schematic and pins.

The library can be used to both, connect to a wifi network, and then open TCP connections, and send data and receive data.

You can also use it to start your own access point and receive data, acting as a TCP server.

Author

Morten Nissen, Nicolai De Jong Bjerg & Kevin Pike Darmer

Copyright

(C) 2020 Morten Nissen, Nicolai De Jung Berg & Kevin Pike Darmer, GNU General Public License Version 3

5.1.2 Macro Definition Documentation

5.1.2.1 ACCESSPOINT

```
#define ACCESSPOINT 2
```

Definition at line 51 of file ESP8266.h.

5.1.2.2 API_WRITE_KEY

```
#define API_WRITE_KEY "9XZN1CY19NFLB7RB"
```

Definition at line 56 of file ESP8266.h.

5.1.2.3 BOTH_STATION_AND_ACCESPOINT

```
#define BOTH_STATION_AND_ACCESPOINT 3
```

Definition at line 52 of file ESP8266.h.

5.1.2.4 CHANNEL_ID

```
#define CHANNEL_ID "1217032"
```

Definition at line 57 of file ESP8266.h.

5.1.2.5 DEFAULT_BUFFER_SIZE

```
#define DEFAULT_BUFFER_SIZE 160
```

Definition at line 38 of file ESP8266.h.

5.1.2.6 DEFAULT_TIMEOUT

```
#define DEFAULT_TIMEOUT 10000
```

Definition at line 39 of file ESP8266.h.

5.1.2.7 DOMAIN

```
#define DOMAIN "api.thingspeak.com"
```

Definition at line 54 of file ESP8266.h.

5.1.2.8 ESP8266_H_

```
#define ESP8266_H_
```

Definition at line 35 of file ESP8266.h.

5.1.2.9 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 32 of file ESP8266.h.

5.1.2.10 MULTIPLE

```
#define MULTIPLE 1
```

Definition at line 43 of file ESP8266.h.

5.1.2.11 NORMAL

```
#define NORMAL 0
```

Definition at line 46 of file ESP8266.h.

5.1.2.12 PASSWORD

```
#define PASSWORD "11111111"
```

Definition at line 60 of file ESP8266.h.

5.1.2.13 PORT

```
#define PORT "80"
```

Definition at line 55 of file ESP8266.h.

5.1.2.14 SINGLE

```
#define SINGLE 0
```

Definition at line 42 of file ESP8266.h.

5.1.2.15 SREG

```
#define SREG _SFR_IO8(0x3F)
```

Definition at line 36 of file ESP8266.h.

5.1.2.16 SSID

```
#define SSID "OP6T"
```

Definition at line 59 of file ESP8266.h.

5.1.2.17 STATION

```
#define STATION 1
```

Definition at line 50 of file ESP8266.h.

5.1.2.18 TRANSPERANT

```
#define TRANSPERANT 1
```

Definition at line 47 of file ESP8266.h.

5.1.3 Enumeration Type Documentation

5.1.3.1 ESP8266_CONNECT_STATUS

```
enum ESP8266_CONNECT_STATUS
```

ESP8266_CONNECT_STATUS is the status that the ESP8266, can be in describing weather or not is is connect to an Access point.

Enumerator

ESP8266_CONNECTED_TO_AP	
ESP8266_CREATED_TRANSMISSION	
ESP8266_TRANSMISSION_DISCONNECTED	
ESP8266_NOT_CONNECTED_TO_AP	
ESP8266_CONNECT_UNKNOWN_ERROR	

Definition at line 79 of file ESP8266.h.

5.1.3.2 ESP8266_JOINAP_STATUS

```
enum ESP8266_JOINAP_STATUS
```

ESP8266_JOINAP_STATUS is the status that the ESP8266, responds with when you connect to an Access point.

Enumerator

ESP8266_WIFI_CONNECTED	
ESP8266_CONNECTION_TIMEOUT	
ESP8266_WRONG_PASSWORD	
ESP8266_NOT_FOUND_TARGET_AP	
ESP8266_CONNECTION_FAILED	
ESP8266_JOIN_UNKNOWN_ERROR	

Definition at line 91 of file ESP8266.h.

5.1.3.3 ESP8266_RESPONSE_STATUS

```
enum ESP8266_RESPONSE_STATUS
```

ESP8266_RESPONSE_STATUS is the status that the ESP8266, can respond with, these are being used for every connection with the device.

Enumerator

ESP8266_RESPONSE_WAITING	
ESP8266_RESPONSE_FINISHED	
ESP8266_RESPONSE_TIMEOUT	
ESP8266_RESPONSE_BUFFER_FULL	
ESP8266_RESPONSE_STARTING	
ESP8266_RESPONSE_ERROR	

Definition at line 66 of file ESP8266.h.

5.1.4 Function Documentation

5.1.4.1 ESP8266_ApplicationMode()

```
bool ESP8266_ApplicationMode (
             uint8_t Mode )
```

ESP8266_ApplicationMode Sets the transmission mode that should be used can be either NORMAL or TRANS↩ PERANT check datasheet page 55. Calls the AT+CIPMODE command to the ESP8266.

Parameters

uint8↩ _t	Mode is the mode it should be set to NORMAL or TRANSPERANT

Returns

Returns true for succes and false for failure

Definition at line 117 of file ESP8266.c.

5.1.4.2 ESP8266_Begin()

```
bool ESP8266_Begin ( )
```

Definition at line 135 of file ESP8266.c.

5.1.4.3 ESP8266_Clear()

```
void ESP8266_Clear ( )
```

Definition at line 67 of file ESP8266.c.

5.1.4.4 ESP8266_Close()

```
bool ESP8266_Close ( )
```

Definition at line 145 of file ESP8266.c.

5.1.4.5 ESP8266_connected()

```
uint8_t ESP8266_connected ( )
```

Definition at line 199 of file ESP8266.c.

5.1.4.6 ESP8266_ConnectionMode()

```
bool ESP8266_ConnectionMode (
             uint8_t Mode )
```

ESP8266_ConnectionMode Set the connection mode that should be used This decides how many connections, there are allowed to be. If hosting your own server on the device this need to be set to MULTIPLE, otherwise just use SINGLE, as long as you are sure only one request is running at a time. Page 53 Calls the AT+CIPMUX command to the ESP8266.

Parameters

uint8←_t	Mode is the mode it should be set to SINGLE or MULTIPLE

Returns

Returns true for succes and false for failure

Definition at line 126 of file ESP8266.c.

5.1.4.7 ESP8266_DataAvailable()

```
int16_t ESP8266_DataAvailable ( )
```

Definition at line 252 of file ESP8266.c.

5.1.4.8 ESP8266_DataRead()

```
uint8_t ESP8266_DataRead ( )
```

Definition at line 257 of file ESP8266.c.

5.1.4.9 ESP8266_JoinAccessPoint()

```
uint8_t ESP8266_JoinAccessPoint (
            char * _SSID,
            char * _PASSWORD )
```

ESP8266_JoinAccessPoint joins the given access point This is used to connect to a certain wifi page 22 Calls the AT+CWJAP,"_SSID","_PASSWORD".

Parameters

char∗	_SSID is the name of the SSID to joins
char∗	_Password is the password that is used to connect

Returns

ESP8266_JOINAP_STATUS with how the connection went

Definition at line 177 of file ESP8266.c.

5.1.4.10 ESP8266_Send()

```
uint8_t ESP8266_Send (
             char * Data )
```

ESP8266_Send sends data to a previously established connection Sends the given data to the already established conenction. Takes the data and calculates the byte length, this is then sent to the AT+CIPSEND command. After that it sends the given data page 47.

Parameters

char∗	Data is the data that should be sent

Returns

ESP8266_RESPONSE_STATUS with what the TCP server responds with

Definition at line 236 of file ESP8266.c.

5.1.4.11 ESP8266_Start()

```
uint8_t ESP8266_Start (
             uint8_t _ConnectionNumber,
             char * Domain,
             char * Port )
```

ESP8266_Start connects to a certain IP and port over TCP Connect to a given TCP server page 46 Calls the AT+CIPSTaRT="TCP","Domain",Port.

Parameters

uint8↩ _t	_ConnectionNumber is used in MULTIPLE connectionmode to select what connection to use single mode should just use 0
char∗	Domain is the ip or domain that it should connect to
char∗	Port is the port that it should connect to

Returns

ESP8266_CONNECT_STATUS with how the connection went

Definition at line 214 of file ESP8266.c.

5.1.4.12 ESP8266_StartServer()

```
bool ESP8266_StartServer (
             uint8_t _port )
```

ESP8266_StartServer starts a TCP server on a given Port This starts the internal TCP server on a given port page 54 Calls the AT+CIPSERVER command to the ESP8266.

Parameters

uint8←_t	_port is the port that it should listen on

Returns

Returns true for succes and false for failure

Definition at line 159 of file ESP8266.c.

5.1.4.13 ESP8266_StopServer()

```
bool ESP8266_StopServer ( )
```

ESP8266_StopServer stops the started TCP server page 54 Calls the AT+CIPSERVER command to the ESP8266.

Returns

Returns true for succes and false for failure

Definition at line 168 of file ESP8266.c.

5.1.4.14 ESP8266_WIFIMode()

```
bool ESP8266_WIFIMode (
            uint8_t _mode )
```

ESP8266_WIFIMode sets the wifi mode that should be used This decides weather you want to act as its own AP or connect to an already existing one. You can also use it as both at the same time. STATION if you want to connect to WIFI ACCESSPOINT if you want it to act as an Access point BOTH_STATION_AND_ACCESPOINT is you want it to act as both Page 19 Calls the AT+CWMODE command to the ESP8266.

Parameters

uint8←_t	_mode is the mode it should be set to STATION, ACCESSPOINT or BOTH_STATION_AND_ACCESPOINT

Returns

Returns true for succes and false for failure

Definition at line 150 of file ESP8266.c.

5.1.4.15 GetResponseBody()

```
void GetResponseBody (
            char * Response,
            uint16_t ResponseLength )
```

Definition at line 82 of file ESP8266.c.

5.1.4.16 Read_Data()

```
uint16_t Read_Data (
            char * _buffer )
```

Definition at line 267 of file ESP8266.c.

5.1.4.17 Read_Response()

```
void Read_Response (
            char * _Expected_Response )
```

Definition at line 23 of file ESP8266.c.

5.1.4.18 SendATandExpectResponse()

```
bool SendATandExpectResponse (
            char * ATCommand,
            char * ExpectedResponse )
```

SendATandExpectResponse sends a certain AT command to the ESP8266 and there after uses the WaitFor↩ ExpectedResponse to wait until a certain message has been send by the ESP8266.

Parameters

*char**	ATCommand is the string that should be send to the ESP8266
*char**	EcpectedResponse is the string that it should wait for

Returns

Returns true for succes and false for failure

Definition at line 109 of file ESP8266.c.

5.1.4.19 Start_Read_Response()

```
void Start_Read_Response (
             char * _ExpectedResponse )
```

Definition at line 73 of file ESP8266.c.

5.1.4.20 WaitForExpectedResponse()

```
bool WaitForExpectedResponse (
             char * ExpectedResponse )
```

WaitForExpectedResponse is used after the SendATandExpectResponse, its used to wait for a certain string being send by the ESP8266.

This is a dangerous function to use, as there is no timeout so if the string never occures its stuck in a loop.

Parameters

*char**	ExpectedResponse is the string that it should wait for

Returns

returns true for success and false for failure

Definition at line 101 of file ESP8266.c.

5.2 I2C Master library

I2C (TWI) Master Software Library.

Macros

- #define I2C_READ 1
- #define I2C_WRITE 0
- #define i2c_read(ack) (ack) ? i2c_readAck() : i2c_readNak();

Functions

- void i2c_init (void)

  initialize the I2C master interace. Need to be called only once
- void i2c_stop (void)

  Terminates the data transfer and releases the I2C bus.
- unsigned char i2c_start (unsigned char addr)

  Issues a start condition and sends address and transfer direction.
- unsigned char i2c_rep_start (unsigned char addr)

  Issues a repeated start condition and sends address and transfer direction.
- void i2c_start_wait (unsigned char addr)

  Issues a start condition and sends address and transfer direction.
- unsigned char i2c_write (unsigned char data)

  Send one byte to I2C device.
- unsigned char i2c_readAck (void)

  read one byte from the I2C device, request more data from device
- unsigned char i2c_readNak (void)

  read one byte from the I2C device, read is followed by a stop condition
- unsigned char i2c_read (unsigned char ack)

  read one byte from the I2C device

5.2.1 Detailed Description

I2C (TWI) Master Software Library.
```
#include <i2cmaster.h>
```

Basic routines for communicating with I2C slave devices. This single master implementation is limited to one bus master on the I2C bus.

This I2c library is implemented as a compact assembler software implementation of the I2C protocol which runs on any AVR (i2cmaster.S) and as a TWI hardware interface for all AVR with built-in TWI hardware (twimaster.c). Since the API for these two implementations is exactly the same, an application can be linked either against the software I2C implementation or the hardware I2C implementation.

Use 4.7k pull-up resistor on the SDA and SCL pin.

Adapt the SCL and SDA port and pin definitions and eventually the delay routine in the module i2cmaster.S to your target when using the software I2C implementation !

Adjust the CPU clock frequence F_CPU in twimaster.c or in the Makfile when using the TWI hardware implementaion.

Note

The module i2cmaster.S is based on the Atmel Application Note AVR300, corrected and adapted to GNU assembler and AVR-GCC C call interface. Replaced the incorrect quarter period delays found in AVR300 with half period delays.

Author

Peter Fleury pfleury@gmx.ch http://tinyurl.com/peterfleury

Copyright

(C) 2015 Peter Fleury, GNU General Public License Version 3

API Usage Example

The following code shows typical usage of this library, see example test_i2cmaster.c

```
#include <i2cmaster.h>
#define Dev24C02  0xA2      // device address of EEPROM 24C02, see datasheet
int main(void)
{
    unsigned char ret;
    i2c_init();                               // initialize I2C library
    // write 0x75 to EEPROM address 5 (Byte Write)
    i2c_start_wait(Dev24C02+I2C_WRITE);      // set device address and write mode
    i2c_write(0x05);                          // write address = 5
    i2c_write(0x75);                          // write value 0x75 to EEPROM
    i2c_stop();                               // set stop conditon = release bus
    // read previously written value back from EEPROM address 5
    i2c_start_wait(Dev24C02+I2C_WRITE);      // set device address and write mode
    i2c_write(0x05);                          // write address = 5
    i2c_rep_start(Dev24C02+I2C_READ);        // set device address and read mode
    ret = i2c_readNak();                      // read one byte from EEPROM
    i2c_stop();
    for(;;);
}
```

5.2.2 Macro Definition Documentation

5.2.2.1 I2C_READ

```
#define I2C_READ 1
```

defines the data direction (reading from I2C device) in i2c_start(),i2c_rep_start()

Definition at line 92 of file I2Clibrary.h.

5.2.2.2 i2c_read

```
#define i2c_read(
                ack ) (ack) ? i2c_readAck() : i2c_readNak();
```

Definition at line 173 of file I2Clibrary.h.

5.2.2.3 I2C_WRITE

```
#define I2C_WRITE 0
```

defines the data direction (writing to I2C device) in i2c_start(),i2c_rep_start()

Definition at line 95 of file I2Clibrary.h.

5.2.3 Function Documentation

5.2.3.1 i2c_init()

```
void i2c_init (
             void  )
```

initialize the I2C master interace. Need to be called only once

Returns

none

Definition at line 27 of file I2Clibrary.c.

5.2.3.2 i2c_read()

```
unsigned char i2c_read (
             unsigned char ack )
```

read one byte from the I2C device

Implemented as a macro, which calls either i2c_readAck or i2c_readNak

Parameters

ack	1 send ack, request more data from device 0 send nak, read is followed by a stop condition

Returns

byte read from I2C device

5.2.3.3 i2c_readAck()

```
unsigned char i2c_readAck (
             void  )
```

read one byte from the I2C device, request more data from device

Returns

byte read from I2C device

Definition at line 180 of file I2Clibrary.c.

5.2.3.4 i2c_readNak()

```
unsigned char i2c_readNak (
             void  )
```

read one byte from the I2C device, read is followed by a stop condition

Returns

byte read from I2C device

Definition at line 195 of file I2Clibrary.c.

5.2.3.5 i2c_rep_start()

```
unsigned char i2c_rep_start (
             unsigned char addr )
```

Issues a repeated start condition and sends address and transfer direction.

Parameters

addr	address and transfer direction of I2C device

Return values

0	device accessible
1	failed to access device

Definition at line 128 of file I2Clibrary.c.

5.2.3.6 i2c_start()

```
unsigned char i2c_start (
             unsigned char addr )
```

Issues a start condition and sends address and transfer direction.

Parameters

addr	address and transfer direction of I2C device

Return values

0	device accessible
1	failed to access device

Definition at line 41 of file I2Clibrary.c.

5.2.3.7 i2c_start_wait()

```
void i2c_start_wait (
             unsigned char addr )
```

Issues a start condition and sends address and transfer direction.

If device is busy, use ack polling to wait until device ready

Parameters

addr	address and transfer direction of I2C device

Returns

none

Definition at line 77 of file I2Clibrary.c.

5.2.3.8 i2c_stop()

```
void i2c_stop (
             void  )
```

Terminates the data transfer and releases the I2C bus.

Returns

none

Definition at line 138 of file I2Clibrary.c.

5.2.3.9 i2c_write()

```
unsigned char i2c_write (
             unsigned char data )
```

Send one byte to I2C device.

Parameters

data	byte to be transfered

Return values

0	write successful
1	write failed

Definition at line 156 of file I2Clibrary.c.

5.3 RTC Library

RTC Library.

Data Structures

- struct DateTime

  Structure used to save and get datetime from the RTC device. Day is what day of the week it is, and date is the date of the year.

Macros

- #define F_CPU 16000000UL
- #define SLAVE_ADDRESS 0xD0

  The slave address of the RTC device.
- #define SECONDS 0x00

  The address register of the seconds.

Functions

- void DS3231_setDateTime (DateTime ∗time)

  Sets the time in the RTC device.
- void DS3231_getDateTime (DateTime ∗now)

  Gets the date from the RTC device and saves it in the given struct.

Variables

- uint8_t Year
- uint8_t Month
- uint8_t Day
- uint8_t Date
- uint8_t Hour
- uint8_t Minute
- uint8_t Second

5.3.1 Detailed Description

RTC Library.
```
#include <RTClibrary.h>
```

Basic routines for communicating with an RTC device over i2c, this library uses the I2CLib from peter fleury. So be sure, to have that included in the project you wish to use this library from.

Connect the RTC using given pins

Author

Morten Nissen, Nicolai De Jong Bjerg & Kevin Pike Darmer

Copyright

(C) 2020 Morten Nissen, Nicolai De Jung Berg & Kevin Pike Darmer, GNU General Public License Version 3

5.3.2 Macro Definition Documentation

5.3.2.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 25 of file RTClibrary.h.

5.3.2.2 SECONDS

```
#define SECONDS 0x00
```

The address register of the seconds.

Definition at line 49 of file RTClibrary.h.

5.3.2.3 SLAVE_ADDRESS

```
#define SLAVE_ADDRESS 0xD0
```

The slave address of the RTC device.

Definition at line 44 of file RTClibrary.h.

5.3.3 Function Documentation

5.3.3.1 DS3231_getDateTime()

```
void DS3231_getDateTime (
             DateTime * now )
```

Gets the date from the RTC device and saves it in the given struct.

Parameters

DateTime	pointer where the data should be put into

Definition at line 21 of file RTClibrary.c.

5.3.3.2 DS3231_setDateTime()

```
void DS3231_setDateTime (
            DateTime * time )
```

Sets the time in the RTC device.

Parameters

DateTime	struct with the time, you wish to set the device to

Definition at line 7 of file RTClibrary.c.

5.3.4 Variable Documentation

5.3.4.1 Date

```
uint8_t Date
```

Definition at line 35 of file RTClibrary.h.

5.3.4.2 Day

```
uint8_t Day
```

Definition at line 34 of file RTClibrary.h.

5.3.4.3 Hour

```
uint8_t Hour
```

Definition at line 36 of file RTClibrary.h.

5.3.4.4 Minute

```
uint8_t Minute
```

Definition at line 37 of file RTClibrary.h.

5.3.4.5 Month

```
uint8_t Month
```

Definition at line 33 of file RTClibrary.h.

5.3.4.6 Second

```
uint8_t Second
```

Definition at line 38 of file RTClibrary.h.

5.3.4.7 Year

```
uint8_t Year
```

Definition at line 32 of file RTClibrary.h.

Chapter 6

Data Structure Documentation

6.1 DateTime Struct Reference

Structure used to save and get datetime from the RTC device. Day is what day of the week it is, and date is the date of the year.

```
#include <RTClibrary.h>
```

Data Fields

- uint8_t Year
- uint8_t Month
- uint8_t Day
- uint8_t Date
- uint8_t Hour
- uint8_t Minute
- uint8_t Second

6.1.1 Detailed Description

Structure used to save and get datetime from the RTC device. Day is what day of the week it is, and date is the date of the year.

Definition at line 31 of file RTClibrary.h.

The documentation for this struct was generated from the following file:

- RTClib/RTClibrary.h

6.2 DHT11 Struct Reference

```
#include <DHT11library.h>
```

Data Fields

- uint8_t Temperatur
- uint8_t Humidity

6.2.1 Detailed Description

Definition at line 8 of file DHT11library.h.

6.2.2 Field Documentation

6.2.2.1 Humidity

```
uint8_t Humidity
```

Definition at line 10 of file DHT11library.h.

6.2.2.2 Temperatur

```
uint8_t Temperatur
```

Definition at line 9 of file DHT11library.h.

The documentation for this struct was generated from the following file:

- DHT11lib/DHT11library.h

Chapter 7

File Documentation

7.1 DC_Motor/DCLibrary.c File Reference

```
#include "DClibrary.h"
```

Macros

- #define F_CPU 16000000UL

Functions

- void DCMotor_init ()
- void Start_Fan ()
- void Stop_Fan ()

7.1.1 Macro Definition Documentation

7.1.1.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 10 of file DCLibrary.c.

7.1.2 Function Documentation

7.1.2.1 DCMotor_init()

```
void DCMotor_init ( )
```

Definition at line 13 of file DCLibrary.c.

7.1.2.2 Start_Fan()

```
void Start_Fan ( )
```

Definition at line 28 of file DCLibrary.c.

7.1.2.3 Stop_Fan()

```
void Stop_Fan ( )
```

Definition at line 32 of file DCLibrary.c.

7.2 DC_Motor/DCLibrary.h File Reference

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

Macros

- #define Fan_Start OCR0A = 255;
- #define Fan_Stop OCR0A = 0;

Functions

- void DCMotor_init ()
- void Start_Fan ()
- void Stop_Fan ()

7.2.1 Macro Definition Documentation

7.2.1.1 Fan_Start

```
#define Fan_Start OCR0A = 255;
```

Definition at line 14 of file DCLibrary.h.

7.2.1.2 Fan_Stop

```
#define Fan_Stop OCR0A = 0;
```

Definition at line 17 of file DCLibrary.h.

7.2.2 Function Documentation

7.2.2.1 DCMotor_init()

```
void DCMotor_init ( )
```

Definition at line 13 of file DCLibrary.c.

7.2.2.2 Start_Fan()

```
void Start_Fan ( )
```

Definition at line 28 of file DCLibrary.c.

7.2.2.3 Stop_Fan()

```
void Stop_Fan ( )
```

Definition at line 32 of file DCLibrary.c.

7.3 DClib/DClibrary.c File Reference

```
#include "DClibrary.h"
#include <stdbool.h>
```

Macros

- #define F_CPU 16000000UL

Functions

- void DCMotor_init ()
- void Set_Speed (uint8_t speed)
- void Start_Fan ()
- void Stop_Fan ()

7.3.1 Macro Definition Documentation

7.3.1.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 10 of file DClibrary.c.

7.3.2 Function Documentation

7.3.2.1 DCMotor_init()

```
void DCMotor_init ( )
```

Definition at line 12 of file DClibrary.c.

7.3.2.2 Set_Speed()

```
void Set_Speed (
             uint8_t speed )
```

Definition at line 27 of file DClibrary.c.

7.3.2.3 Start_Fan()

```
void Start_Fan ( )
```

Definition at line 31 of file DClibrary.c.

7.3.2.4 Stop_Fan()

```
void Stop_Fan ( )
```

Definition at line 35 of file DClibrary.c.

7.4 DClib/DClibrary.h File Reference

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

Macros

- #define F_CPU 16000000UL
- #define Fan OCR0A
- #define Stop 0;

Functions

- void DCMotor_init ()
- void Set_Speed (uint8_t speed)
- void Start_Fan ()
- void Stop_Fan ()

Variables

- uint8_t Speed

7.4.1 Macro Definition Documentation

7.4.1.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 9 of file DClibrary.h.

7.4.1.2 Fan

```
#define Fan OCR0A
```

Definition at line 15 of file DClibrary.h.

7.4.1.3 Stop

```
#define Stop 0;
```

Definition at line 18 of file DClibrary.h.

7.4.2 Function Documentation

7.4.2.1 DCMotor_init()

```
void DCMotor_init ( )
```

Definition at line 13 of file DCLibrary.c.

7.4.2.2 Set_Speed()

```
void Set_Speed (
             uint8_t speed )
```

Definition at line 27 of file DClibrary.c.

7.4.2.3 Start_Fan()

```
void Start_Fan ( )
```

Definition at line 28 of file DCLibrary.c.

7.4.2.4 Stop_Fan()

```
void Stop_Fan ( )
```

Definition at line 32 of file DCLibrary.c.

7.4.3 Variable Documentation

7.4.3.1 Speed

```
uint8_t Speed
```

Definition at line 21 of file DClibrary.h.

7.5 DHT11lib/Debug/DHT11library.d File Reference

7.6 DHT11lib/DHT11library.c File Reference

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/sfr_defs.h>
#include <stdbool.h>
#include <stdio.h>
#include "DHT11library.h"
```

Macros

- #define F_CPU 16000000UL
- #define LED_TOOGLE PORTA ^= (1 <<PA1)
- #define LED_ON PORTA |= (1 << PA1)

Functions

- bool DHT11_Wait (int maxWaitTime, bool waitForHigh)
- void DHT11_init ()
- void DHT11_reset ()
- uint8_t DHT11_ReadTemp ()
- void DHT11_ReadRaw (DHT11 *result)

7.6.1 Macro Definition Documentation

7.6.1.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 1 of file DHT11library.c.

7.6.1.2 LED_ON

```
#define LED_ON PORTA |= (1 << PA1)
```

Definition at line 10 of file DHT11library.c.

7.6.1.3 LED_TOOGLE

```
#define LED_TOOGLE PORTA ^= (1 <<PA1)
```

Definition at line 9 of file DHT11library.c.

7.6.2 Function Documentation

7.6.2.1 DHT11_init()

```
void DHT11_init ( )
```

Definition at line 41 of file DHT11library.c.

7.6.2.2 DHT11_ReadRaw()

```
void DHT11_ReadRaw (
            DHT11 * result )
```

Definition at line 59 of file DHT11library.c.

7.6.2.3 DHT11_ReadTemp()

```
uint8_t DHT11_ReadTemp ( )
```

Definition at line 52 of file DHT11library.c.

7.6.2.4 DHT11_reset()

```
void DHT11_reset ( )
```

Definition at line 47 of file DHT11library.c.

7.6.2.5 DHT11_Wait()

```
bool DHT11_Wait (
             int maxWaitTime,
             bool waitForHigh )
```

Definition at line 13 of file DHT11library.c.

7.7 DHT11lib/DHT11library.h File Reference

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

Data Structures

- struct DHT11

Macros

- #define F_CPU 16000000UL
- #define SET_BIT(port, bit) ((port) |= (1<<bit))
- #define CLR_BIT(port, bit) ((port) &= ~((1<<bit)))
- #define DHT11_DDR DDRA
- #define DHT11_PORT PORTA
- #define DHT11_PIN PINA
- #define DHT11_BIT PA0

Functions

- void DHT11_init ()
- void DHT11_ReadRaw (DHT11 *result)
- uint8_t DHT11_ReadTemp ()
- uint8_t DHT11_ReadHumid ()

7.7.1 Macro Definition Documentation

7.7.1.1 CLR_BIT

```
#define CLR_BIT(
              port,
              bit ) ((port) &= ~((1<<bit)))
```

Definition at line 14 of file DHT11library.h.

7.7.1.2 DHT11_BIT

```
#define DHT11_BIT PA0
```

Definition at line 19 of file DHT11library.h.

7.7.1.3 DHT11_DDR

```
#define DHT11_DDR DDRA
```

Definition at line 16 of file DHT11library.h.

7.7.1.4 DHT11_PIN

```
#define DHT11_PIN PINA
```

Definition at line 18 of file DHT11library.h.

7.7.1.5 DHT11_PORT

```
#define DHT11_PORT PORTA
```

Definition at line 17 of file DHT11library.h.

7.7.1.6 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 4 of file DHT11library.h.

7.7.1.7 SET_BIT

```
#define SET_BIT(
              port,
              bit ) ((port) |= (1<<bit))
```

Definition at line 13 of file DHT11library.h.

7.7.2 Function Documentation

7.7.2.1 DHT11_init()

```
void DHT11_init ( )
```

Definition at line 41 of file DHT11library.c.

7.7.2.2 DHT11_ReadHumid()

```
uint8_t DHT11_ReadHumid ( )
```

7.7.2.3 DHT11_ReadRaw()

```
void DHT11_ReadRaw (
            DHT11 * result )
```

Definition at line 59 of file DHT11library.c.

7.7.2.4 DHT11_ReadTemp()

```
uint8_t DHT11_ReadTemp ( )
```

Definition at line 52 of file DHT11library.c.

7.8 ESP8266lib/Debug/ESP8266.d File Reference

7.9 ESP8266lib/Debug/USART/USART_RS232_C_file.d File Reference

7.10 ESP8266lib/ESP8266.c File Reference

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include "USART/USART_RS232_H_file.h"
#include "ESP8266.h"
```

Macros

- #define F_CPU 16000000UL

Functions

- void Read_Response (char ∗_Expected_Response)
- void ESP8266_Clear ()
- void Start_Read_Response (char ∗_ExpectedResponse)
- void GetResponseBody (char ∗Response, uint16_t ResponseLength)
- bool WaitForExpectedResponse (char ∗ExpectedResponse)

  WaitForExpectedResponse is used after the SendATandExpectResponse, its used to wait for a certain string being send by the ESP8266.
- bool SendATandExpectResponse (char ∗ATCommand, char ∗ExpectedResponse)

  SendATandExpectResponse sends a certain AT command to the ESP8266 and there after uses the WaitFor←↩ ExpectedResponse to wait until a certain message has been send by the ESP8266.
- bool ESP8266_ApplicationMode (uint8_t Mode)

  ESP8266_ApplicationMode Sets the transmission mode that should be used can be either NORMAL or TRANSP←↩ ERANT check datasheet page 55. Calls the AT+CIPMODE command to the ESP8266.
- bool ESP8266_ConnectionMode (uint8_t Mode)

  ESP8266_ConnectionMode Set the connection mode that should be used This decides how many connections, there are allowed to be. If hosting your own server on the device this need to be set to MULTIPLE, otherwise just use SINGLE, as long as you are sure only one request is running at a time. Page 53 Calls the AT+CIPMUX command to the ESP8266.
- bool ESP8266_Begin ()
- bool ESP8266_Close ()
- bool ESP8266_WIFIMode (uint8_t _mode)

  ESP8266_WIFIMode sets the wifi mode that should be used This decides weather you want to act as its own AP or connect to an already existing one. You can also use it as both at the same time. STATION if you want to connect to WIFI ACCESSPOINT if you want it to act as an Access point BOTH_STATION_AND_ACCESPOINT is you want it to act as both Page 19 Calls the AT+CWMODE command to the ESP8266.
- bool ESP8266_StartServer (uint8_t _port)

  ESP8266_StartServer starts a TCP server on a given Port This starts the internal TCP server on a given port page 54 Calls the AT+CIPSERVER command to the ESP8266.
- bool ESP8266_StopServer ()

  ESP8266_StopServer stops the started TCP server page 54 Calls the AT+CIPSERVER command to the ESP8266.
- uint8_t ESP8266_JoinAccessPoint (char ∗_SSID, char ∗_PASSWORD)

  ESP8266_JoinAccessPoint joins the given access point This is used to connect to a certain wifi page 22 Calls the AT+CWJAP,"_SSID","_PASSWORD".
- uint8_t ESP8266_connected ()
- uint8_t ESP8266_Start (uint8_t _ConnectionNumber, char ∗Domain, char ∗Port)

  ESP8266_Start connects to a certain IP and port over TCP Connect to a given TCP server page 46 Calls the AT+←↩ CIPSTaRT="TCP","Domain",Port.
- uint8_t ESP8266_Send (char ∗Data)

  ESP8266_Send sends data to a previously established connection Sends the given data to the already established conenction. Takes the data and calculates the byte length, this is then sent to the AT+CIPSEND command. After that it sends the given data page 47.
- int16_t ESP8266_DataAvailable ()
- uint8_t ESP8266_DataRead ()
- uint16_t Read_Data (char ∗_buffer)
- ISR (USART1_RX_vect)

Variables

- int8_t Response_Status
- volatile int16_t Counter = 0
- volatile int16_t pointer = 0
- uint32_t TimeOut = 0
- char RESPONSE_BUFFER [DEFAULT_BUFFER_SIZE]

7.10.1 Macro Definition Documentation

7.10.1.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 16 of file ESP8266.c.

7.10.2 Function Documentation

7.10.2.1 ISR()

```
ISR (
            USART1_RX_vect )
```

Definition at line 276 of file ESP8266.c.

7.10.3 Variable Documentation

7.10.3.1 Counter

```
volatile int16_t Counter = 0
```

Definition at line 19 of file ESP8266.c.

7.10.3.2 pointer

```
volatile int16_t pointer = 0
```

Definition at line 19 of file ESP8266.c.

7.10.3.3 RESPONSE_BUFFER

```
char RESPONSE_BUFFER[DEFAULT_BUFFER_SIZE]
```

Definition at line 21 of file ESP8266.c.

7.10.3.4 Response_Status

```
int8_t Response_Status
```

Definition at line 18 of file ESP8266.c.

7.10.3.5 TimeOut

```
uint32_t TimeOut = 0
```

Definition at line 20 of file ESP8266.c.

7.11 ESP8266lib/ESP8266.h File Reference

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/interrupt.h>
```

Macros

- #define F_CPU 16000000UL
- #define ESP8266_H_
- #define SREG _SFR_IO8(0x3F)
- #define DEFAULT_BUFFER_SIZE 160
- #define DEFAULT_TIMEOUT 10000
- #define SINGLE 0
- #define MULTIPLE 1
- #define NORMAL 0
- #define TRANSPERANT 1
- #define STATION 1
- #define ACCESSPOINT 2
- #define BOTH_STATION_AND_ACCESPOINT 3
- #define DOMAIN "api.thingspeak.com"
- #define PORT "80"
- #define API_WRITE_KEY "9XZN1CY19NFLB7RB"
- #define CHANNEL_ID "1217032"
- #define SSID "OP6T"
- #define PASSWORD "11111111"

Enumerations

- enum ESP8266_RESPONSE_STATUS {
  ESP8266_RESPONSE_WAITING, ESP8266_RESPONSE_FINISHED, ESP8266_RESPONSE_TIMEOUT,
  ESP8266_RESPONSE_BUFFER_FULL,
  ESP8266_RESPONSE_STARTING, ESP8266_RESPONSE_ERROR }

  ESP8266_RESPONSE_STATUS is the status that the ESP8266, can respond with, these are being used for every connection with the device.
- enum ESP8266_CONNECT_STATUS {
  ESP8266_CONNECTED_TO_AP, ESP8266_CREATED_TRANSMISSION, ESP8266_TRANSMISSION_DISCONNECTED,
  ESP8266_NOT_CONNECTED_TO_AP,
  ESP8266_CONNECT_UNKNOWN_ERROR }

  ESP8266_CONNECT_STATUS is the status that the ESP8266, can be in describing weather or not is is connect to an Access point.
- enum ESP8266_JOINAP_STATUS {
  ESP8266_WIFI_CONNECTED, ESP8266_CONNECTION_TIMEOUT, ESP8266_WRONG_PASSWORD,
  ESP8266_NOT_FOUND_TARGET_AP,
  ESP8266_CONNECTION_FAILED, ESP8266_JOIN_UNKNOWN_ERROR }

  ESP8266_JOINAP_STATUS is the status that the ESP8266, responds with when you connect to an Access point.

Functions

- void Read_Response (char ∗_Expected_Response)
- void ESP8266_Clear ()
- void Start_Read_Response (char ∗_ExpectedResponse)
- void GetResponseBody (char ∗Response, uint16_t ResponseLength)
- bool WaitForExpectedResponse (char ∗ExpectedResponse)

  WaitForExpectedResponse is used after the SendATandExpectResponse, its used to wait for a certain string being send by the ESP8266.
- bool SendATandExpectResponse (char ∗ATCommand, char ∗ExpectedResponse)

  SendATandExpectResponse sends a certain AT command to the ESP8266 and there after uses the WaitFor↩ ExpectedResponse to wait until a certain message has been send by the ESP8266.
- bool ESP8266_ApplicationMode (uint8_t Mode)

  ESP8266_ApplicationMode Sets the transmission mode that should be used can be either NORMAL or TRANSP↩ ERANT check datasheet page 55. Calls the AT+CIPMODE command to the ESP8266.
- bool ESP8266_ConnectionMode (uint8_t Mode)

  ESP8266_ConnectionMode Set the connection mode that should be used This decides how many connections, there are allowed to be. If hosting your own server on the device this need to be set to MULTIPLE, otherwise just use SINGLE, as long as you are sure only one request is running at a time. Page 53 Calls the AT+CIPMUX command to the ESP8266.
- bool ESP8266_Begin ()
- bool ESP8266_Close ()
- bool ESP8266_WIFIMode (uint8_t _mode)

  ESP8266_WIFIMode sets the wifi mode that should be used This decides weather you want to act as its own AP or connect to an already existing one. You can also use it as both at the same time. STATION if you want to connect to WIFI ACCESSPOINT if you want it to act as an Access point BOTH_STATION_AND_ACCESPOINT is you want it to act as both Page 19 Calls the AT+CWMODE command to the ESP8266.
- bool ESP8266_StartServer (uint8_t _port)

  ESP8266_StartServer starts a TCP server on a given Port This starts the internal TCP server on a given port page 54 Calls the AT+CIPSERVER command to the ESP8266.
- bool ESP8266_StopServer ()

  ESP8266_StopServer stops the started TCP server page 54 Calls the AT+CIPSERVER command to the ESP8266.
- uint8_t ESP8266_JoinAccessPoint (char ∗_SSID, char ∗_PASSWORD)

  ESP8266_JoinAccessPoint joins the given access point This is used to connect to a certain wifi page 22 Calls the AT+CWJAP,"_SSID","_PASSWORD".

- uint8_t ESP8266_connected ()
- uint8_t ESP8266_Start (uint8_t _ConnectionNumber, char ∗Domain, char ∗Port)

  ESP8266_Start connects to a certain IP and port over TCP Connect to a given TCP server page 46 Calls the AT+↩ CIPSTaRT="TCP","Domain",Port.
- uint8_t ESP8266_Send (char ∗Data)

  ESP8266_Send sends data to a previously established connection Sends the given data to the already established conenction. Takes the data and calculates the byte length, this is then sent to the AT+CIPSEND command. After that it sends the given data page 47.
- int16_t ESP8266_DataAvailable ()
- uint8_t ESP8266_DataRead ()
- uint16_t Read_Data (char ∗_buffer)

7.12 ESP8266lib/ESP8266.md File Reference

7.13 ESP8266lib/USART/USART_RS232_C_file.c File Reference

```
#include "USART_RS232_H_file.h"
```

Functions

- void USART_Init (unsigned long BAUDRATE)
- char USART_RxChar ()
- void USART_TxChar (char data)
- void USART_SendString (char ∗str)

7.13.1 Function Documentation

7.13.1.1 USART_Init()

```
void USART_Init (
            unsigned long BAUDRATE )
```

Definition at line 9 of file USART_RS232_C_file.c.

7.13.1.2 USART_RxChar()

```
char USART_RxChar ( )
```

Definition at line 20 of file USART_RS232_C_file.c.

7.13.1.3 USART_SendString()

```
void USART_SendString (
             char * str )
```

Definition at line 32 of file USART_RS232_C_file.c.

7.13.1.4 USART_TxChar()

```
void USART_TxChar (
             char data )
```

Definition at line 26 of file USART_RS232_C_file.c.

7.14 ESP8266lib/USART/USART_RS232_H_file.h File Reference

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

Macros

- #define F_CPU 16000000UL
- #define DOUBLE_SPEED_MODE
- #define BAUD_PRESCALE (int)round(((((double)F_CPU / ((double)BAUDRATE * 8.0))) - 1.0)) /* Define prescale value */

Functions

- void USART_Init (unsigned long)
- char USART_RxChar ()
- void USART_TxChar (char)
- void USART_SendString (char *)

7.14.1 Macro Definition Documentation

7.14.1.1 BAUD_PRESCALE

```
#define BAUD_PRESCALE (int)round(((((double)F_CPU / ((double)BAUDRATE * 8.0))) - 1.0)) /*
Define prescale value */
```

Definition at line 17 of file USART_RS232_H_file.h.

7.14.1.2 DOUBLE_SPEED_MODE

```
#define DOUBLE_SPEED_MODE
```

Definition at line 14 of file USART_RS232_H_file.h.

7.14.1.3 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 7 of file USART_RS232_H_file.h.

7.14.2 Function Documentation

7.14.2.1 USART_Init()

```
void USART_Init (
            unsigned long  )
```

Definition at line 9 of file USART_RS232_C_file.c.

7.14.2.2 USART_RxChar()

```
char USART_RxChar ( )
```

Definition at line 20 of file USART_RS232_C_file.c.

7.14.2.3 USART_SendString()

```
void USART_SendString (
            char *  )
```

Definition at line 32 of file USART_RS232_C_file.c.

7.14.2.4 USART_TxChar()

```
void USART_TxChar (
            char  )
```

Definition at line 26 of file USART_RS232_C_file.c.

7.15 I2Clib/Debug/I2Clibrary.d File Reference

7.16 I2Clib/I2Clibrary.c File Reference

```
#include <inttypes.h>
#include <compat/twi.h>
#include "I2Clibrary.h"
```

Macros

- #define F_CPU 4000000UL
- #define SCL_CLOCK 100000L

Functions

- void i2c_init (void)

  initialize the I2C master interace. Need to be called only once
- unsigned char i2c_start (unsigned char address)

  Issues a start condition and sends address and transfer direction.
- void i2c_start_wait (unsigned char address)

  Issues a start condition and sends address and transfer direction.
- unsigned char i2c_rep_start (unsigned char address)

  Issues a repeated start condition and sends address and transfer direction.
- void i2c_stop (void)

  Terminates the data transfer and releases the I2C bus.
- unsigned char i2c_write (unsigned char data)

  Send one byte to I2C device.
- unsigned char i2c_readAck (void)

  read one byte from the I2C device, request more data from device
- unsigned char i2c_readNak (void)

  read one byte from the I2C device, read is followed by a stop condition

7.16.1 Macro Definition Documentation

7.16.1.1 F_CPU

```
#define F_CPU 4000000UL
```

Definition at line 17 of file I2Clibrary.c.

7.16.1.2 SCL_CLOCK

```
#define SCL_CLOCK 100000L
```

Definition at line 21 of file I2Clibrary.c.

7.17 I2Clib/I2Clibrary.h File Reference

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

Macros

- #define I2C_READ 1
- #define I2C_WRITE 0
- #define i2c_read(ack) (ack) ? i2c_readAck() : i2c_readNak();

Functions

- void i2c_init (void)

  initialize the I2C master interace. Need to be called only once
- void i2c_stop (void)

  Terminates the data transfer and releases the I2C bus.
- unsigned char i2c_start (unsigned char addr)

  Issues a start condition and sends address and transfer direction.
- unsigned char i2c_rep_start (unsigned char addr)

  Issues a repeated start condition and sends address and transfer direction.
- void i2c_start_wait (unsigned char addr)

  Issues a start condition and sends address and transfer direction.
- unsigned char i2c_write (unsigned char data)

  Send one byte to I2C device.
- unsigned char i2c_readAck (void)

  read one byte from the I2C device, request more data from device
- unsigned char i2c_readNak (void)

  read one byte from the I2C device, read is followed by a stop condition
- unsigned char i2c_read (unsigned char ack)

  read one byte from the I2C device

7.18 Master/Debug/main.d File Reference

7.19 Projekt/Debug/main.d File Reference

7.20 Master/Debug/Setup/stdio_setup.d File Reference

7.21 Master/main.c File Reference

```
#include <avr/io.h>
#include <stdbool.h>
#include <util/delay.h>
#include "DHT11library.h"
#include "ESP8266.h"
#include "SPIlibrary.h"
```

Macros

- #define F_CPU 16000000UL
- #define MAX_TEMP 25

Functions

- int main (void)

7.21.1 Macro Definition Documentation

7.21.1.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 1 of file main.c.

7.21.1.2 MAX_TEMP

```
#define MAX_TEMP 25
```

Definition at line 10 of file main.c.

7.21.2 Function Documentation

7.21.2.1 main()

```
int main (
            void  )
```

Definition at line 13 of file main.c.

7.22 Slave/main.c File Reference

```
#include <avr/io.h>
#include "DClibrary.h"
#include "SPIlibrary.h"
#include <avr/interrupt.h>
#include <stdio.h>
```

Macros

- #define F_CPU 16000000UL

Functions

- ISR (PCINT0_vect)
- void Slave_Init ()
- int main (void)

Variables

- uint8_t reg
- uint8_t val

7.22.1 Macro Definition Documentation

7.22.1.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 1 of file main.c.

7.22.2 Function Documentation

7.22.2.1 ISR()

```
ISR (
            PCINT0_vect )
```

Definition at line 12 of file main.c.

7.22.2.2 main()

```
int main (
            void )
```

Definition at line 45 of file main.c.

7.22.2.3 Slave_Init()

```
void Slave_Init ( )
```

Definition at line 37 of file main.c.

7.22.3 Variable Documentation

7.22.3.1 reg

```
uint8_t reg
```

Definition at line 9 of file main.c.

7.22.3.2 val

```
uint8_t val
```

Definition at line 10 of file main.c.

7.23 Master/Setup/stdio_setup.c File Reference

```
#include <avr/io.h>
#include <stdio.h>
#include <util/setbaud.h>
```

Macros

- #define F_CPU 16000000UL
- #define BAUD 9600
- #define BAUD_TOL 2

Functions

- void UartPutchar (char data)
- int UartGetchar (void)
- void UartInit (void)

Variables

- static FILE the_stdio = FDEV_SETUP_STREAM(UartPutchar, UartGetchar, _FDEV_SETUP_RW)

7.23.1 Macro Definition Documentation

7.23.1.1 BAUD

```
#define BAUD 9600
```

Definition at line 2 of file stdio_setup.c.

7.23.1.2 BAUD_TOL

```
#define BAUD_TOL 2
```

Definition at line 3 of file stdio_setup.c.

7.23.1.3 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 1 of file stdio_setup.c.

7.23.2 Function Documentation

7.23.2.1 UartGetchar()

```
int UartGetchar (
             void  )
```

Definition at line 40 of file stdio_setup.c.

7.23.2.2 UartInit()

```
void UartInit (
             void  )
```

Definition at line 14 of file stdio_setup.c.

7.23.2.3 UartPutchar()

```
void UartPutchar (
             char data )
```

Definition at line 34 of file stdio_setup.c.

7.23.3 Variable Documentation

7.23.3.1 the_stdio

```
FILE the_stdio = FDEV_SETUP_STREAM(UartPutchar, UartGetchar, _FDEV_SETUP_RW)  [static]
```

Definition at line 12 of file stdio_setup.c.

7.24 Slave/stdio_setup.c File Reference

```
#include <avr/io.h>
#include <stdio.h>
#include <util/setbaud.h>
```

Macros

- #define BAUD 9600
- #define BAUD_TOL 2

Functions

- void UartPutchar (char data)
- int UartGetchar (void)
- void UartInit (void)

Variables

- static FILE the_stdio = FDEV_SETUP_STREAM(UartPutchar, UartGetchar, _FDEV_SETUP_RW)

7.24.1 Macro Definition Documentation

7.24.1.1 BAUD

```
#define BAUD 9600
```

Definition at line 1 of file stdio_setup.c.

7.24.1.2 BAUD_TOL

```
#define BAUD_TOL 2
```

Definition at line 2 of file stdio_setup.c.

7.24.2 Function Documentation

7.24.2.1 UartGetchar()

```
int UartGetchar (
             void  )
```

Definition at line 39 of file stdio_setup.c.

7.24.2.2 UartInit()

```
void UartInit (
             void  )
```

Definition at line 13 of file stdio_setup.c.

7.24.2.3 UartPutchar()

```
void UartPutchar (
             char data )
```

Definition at line 33 of file stdio_setup.c.

7.24.3 Variable Documentation

7.24.3.1 the_stdio

```
FILE the_stdio = FDEV_SETUP_STREAM(UartPutchar, UartGetchar, _FDEV_SETUP_RW)  [static]
```

Definition at line 11 of file stdio_setup.c.

7.25 Master/Setup/stdio_setup.h File Reference

Functions

- void UartInit (void)

7.25.1 Function Documentation

7.25.1.1 UartInit()

```
void UartInit (
             void  )
```

Definition at line 14 of file stdio_setup.c.

7.26 Slave/stdio_setup.h File Reference

Functions

- void UartInit (void)

7.26.1 Function Documentation

7.26.1.1 UartInit()

```
void UartInit (
            void  )
```

Definition at line 14 of file stdio_setup.c.

7.27 Projekt/Debug/DHT11/DHT11.d File Reference

7.28 Projekt/Debug/i2c/twimaster.d File Reference

7.29 Projekt/Debug/RTC/RTC.d File Reference

7.30 RTClib/Debug/RTClibrary.d File Reference

7.31 RTClib/RTClibrary.c File Reference

```
#include <avr/io.h>
#include <ctype.h>
#include "I2Clibrary.h"
#include "RTClibrary.h"
```

Macros

- #define F_CPU 16000000UL

Functions

- void DS3231_setDateTime (DateTime ∗time)

  Sets the time in the RTC device.
- void DS3231_getDateTime (DateTime ∗now)

  Gets the date from the RTC device and saves it in the given struct.

7.31.1 Macro Definition Documentation

7.31.1.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 1 of file RTClibrary.c.

7.32 RTClib/RTClibrary.h File Reference

Data Structures

- struct DateTime

  Structure used to save and get datetime from the RTC device. Day is what day of the week it is, and date is the date of the year.

Macros

- #define F_CPU 16000000UL
- #define SLAVE_ADDRESS 0xD0

  The slave address of the RTC device.
- #define SECONDS 0x00

  The address register of the seconds.

Functions

- void DS3231_setDateTime (DateTime ∗time)

  Sets the time in the RTC device.
- void DS3231_getDateTime (DateTime ∗now)

  Gets the date from the RTC device and saves it in the given struct.

7.33 SPIlib/Debug/SPIlibrary.d File Reference

7.34 SPIlib/SPIlibrary.c File Reference

```
#include <avr/io.h>
#include <stdbool.h>
#include <util/delay.h>
#include "SPIlibrary.h"
```

Macros

- #define F_CPU 16000000UL

Functions

- void SPI_Init (bool isMaster)
- char SPI_Read ()
- void SPI_Write (char data)
- void SPI_WriteData (char address, char data)

7.34.1 Macro Definition Documentation

7.34.1.1 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 1 of file SPIlibrary.c.

7.34.2 Function Documentation

7.34.2.1 SPI_Init()

```
void SPI_Init (
             bool isMaster )
```

Definition at line 7 of file SPIlibrary.c.

7.34.2.2 SPI_Read()

```
char SPI_Read ( )
```

Definition at line 33 of file SPIlibrary.c.

7.34.2.3 SPI_Write()

```
void SPI_Write (
             char data )
```

Definition at line 40 of file SPIlibrary.c.

7.34.2.4 SPI_WriteData()

```
void SPI_WriteData (
             char address,
             char data )
```

Definition at line 46 of file SPIlibrary.c.

7.35 SPIlib/SPIlibrary.h File Reference

```
#include <stdbool.h>
```

Macros

- #define F_CPU 16000000UL
- #define BAUD 9600
- #define BAUD_TOL 2
- #define SS_ENABLE PORTB &= ~(1<<PB0)
- #define SS_DISABLE PORTB |= (1<<PB0)

Functions

- void SPI_Init (bool isMaster)
- char SPI_Read ()
- void SPI_Write (char data)
- void SPI_WriteData (char address, char data)

7.35.1 Macro Definition Documentation

7.35.1.1 BAUD

```
#define BAUD 9600
```

Definition at line 5 of file SPIlibrary.h.

7.35.1.2 BAUD_TOL

```
#define BAUD_TOL 2
```

Definition at line 6 of file SPIlibrary.h.

7.35.1.3 F_CPU

```
#define F_CPU 16000000UL
```

Definition at line 4 of file SPIlibrary.h.

7.35.1.4 SS_DISABLE

```
#define SS_DISABLE PORTB |= (1<<PB0)
```

Definition at line 8 of file SPIlibrary.h.

7.35.1.5 SS_ENABLE

```
#define SS_ENABLE PORTB &= ~(1<<PB0)
```

Definition at line 7 of file SPIlibrary.h.

7.35.2 Function Documentation

7.35.2.1 SPI_Init()

```
void SPI_Init (
             bool isMaster )
```

Definition at line 7 of file SPIlibrary.c.

7.35.2.2 SPI_Read()

```
char SPI_Read ( )
```

Definition at line 33 of file SPIlibrary.c.

7.35.2.3 SPI_Write()

```
void SPI_Write (
            char data )
```

Definition at line 40 of file SPIlibrary.c.

7.35.2.4 SPI_WriteData()

```
void SPI_WriteData (
            char address,
            char data )
```

Definition at line 46 of file SPIlibrary.c.

Index