# MonitoringBoard

Generated by Doxygen 1.7.6.1

Wed Mar 20 2013 11:28:09

# Contents

Chapter 1

# MonitoringBoard

# 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 documented files with brief descriptions:

# Chapter 5

# Module Documentation

## 5.1 Main Control Loop

This is the main file of the firmware Here the necessary Functions to measure and scale Values, communicate and react are implemented.

### Defines

- #define **TIMERCOUNTER** TCNT0

  *led test counter*
- #define **TIMERCONTROL_B** TCCR0B

  *prescaler led counter*
- #define **ACK_CAN** 10

  *ports are ranged from 0 to 7 --- 10 for ACK*

### Enumerations

- enum **portMeasureModes** { **NOT_USED**, **Temp**, **Light**, **Vol**, **Noise**, **Distance**, **Led** }

  *values interval when reaction is triggerd, without interval the relais jitters*

### Functions

- void **sleep** (uint32_t val)

  *sleep methode self-made sleep method for better timing*
- void **initPorts** (void)

  *init the ports used as in or output and init the ADC with 125 kHz*
- void **initVoltageMultiplikator** (void)

  *init the multiplicator for the voltage calc 1, 5,24, or 111 for 400 V*

- double **getTemp** (void)

  *get the temperature measured with the temp-sensor*
- double **getLightIntensity** (void)

  *get the light intensity with a level of 0-100, 0 means its dark and 100 is very bright*
- double **getNoise** (void)

  *get noise level with the mic sensor, with level 0-100 100 is very loud*
- double **getDistance** (void)

  *get the distance to the next object near to the ultrasonic sensor*
- uint16_t **startMeasure** (unsigned int X)

  *measure voltage on port x*
- void **measureAndScale** (uint8_t port)

  *use startMeasure to measure and scale with given scalingfactor*
- double **getUnit** (uint8_t port)

  *calculate the value of the real unit e.g. Distance in cm from measure*
- void **sendReactCAN** (uint8_t port, char HoL)

  *send a msg over CAN that the system has reacted format: Hn , Ln; H when switched off (relais high) and L when back on, n is the port*
- void **reactOnMeasure** (uint8_t port)

  *reacts on measured values e.g. by setting a port High or Low to switch the relais --- sends a Hn*
  *when set High Ln*
  *when Low, n is the port*
- void **sendMeasureUART** (uint8_t port)

  *send measured data by uart Format: P:n;x y*
  *--- n = portnumber, x = value, y = unit*
- void **generateCANMsg** (**tExtendedCAN** ∗M, uint8_t port)

  *generates a can message format: bits [0-5] value, bit[6] unit, bit [7] port*
- uint8_t **readUARTConfig** (unsigned char conf)
- uint8_t **readUARTConfModes** (unsigned char conf)
- uint8_t **reactOnUARTCommand** (unsigned char c)

  *reaction for given commands*
- uint8_t **reactOnCANCommand** (char ∗data)

  *reaction for given commands*
- void **can_init** (void)

  *initialize the can driver*
- int **main** (void)

  *the main method with statemachine loop as follows: statemachine ----!!!*

## Variables

- uint8_t **measuringVoltageFaktor** [8]

  *the scalings (1,5,26,111 to calc the voltage, 0 means no measurement. 400 -> 111 as voltagefaktor*
- char **voltageValueBuffer** [7]

*char buffer for measure conversion in string*

- double **measurement**

  *the measurement*

- double **ledLastMeasure**

  *led test last measure*

- uint8_t **ledTick** = 150
- uint8_t **serialCommunikation** = 1

  *en-/disable serial line*

- uint8_t **canCommunikation** = 1

  *en-/disable can*

- uint8_t **ethCommunikation** = 0

  *en-/disable eternet line*

- uint8_t **portToRelais** [8]
- uint8_t **reactionValueMode** [8]

  *which port korresponds to which relais to react*

- uint32_t **reactionValues** [8]

  *should be switched when measure is higher or lower val*

- uint8_t **portModeMeasure** [ ]

  *modes how we can measure*

- double **intervalValues** [7]

  *which measure mode on specific port*

- double **distanceScale** = 250

  *the specific relais-switch interval to handle relais jittering: Temp 1C, Light 7%, Vol 0,5, Distance 5cm*

- double **heartBeat** = 180.0

  *Vcc/512 ∗ 2,54 (inch -> cm)*

- double **mea** [200]

  *standard beat 330 ms tune up to 1,5 s with poti*

- double **loopCounter** = 0.0
- char **te** [33]

  *the counter for the inner measure loop*

### 5.1.1 Detailed Description

This is the main file of the firmware Here the necessary Functions to measure and scale Values, communicate and react are implemented.

**Author**

christian schaub

### 5.1.2 Function Documentation

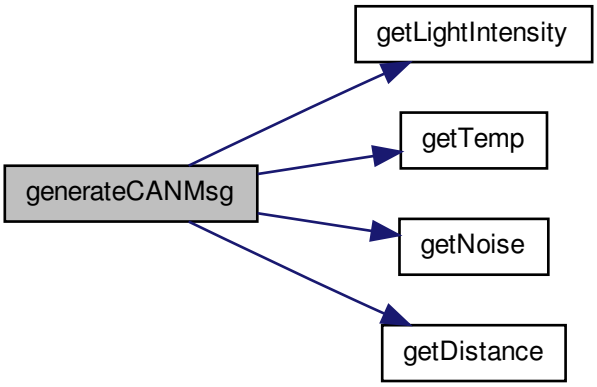
#### 5.1.2.1 void generateCANMsg ( tExtendedCAN * *M,* uint8_t *port* )

generates a can message format: bits [0-5] value, bit[6] unit, bit [7] port

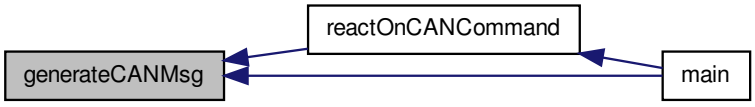
**Parameters**

| | |
|---|---|
| *pointer* | to the message, port which is measured |

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.1.2.2 double getUnit ( uint8_t *port* )

calculate the value of the real unit e.g. Distance in cm from measure

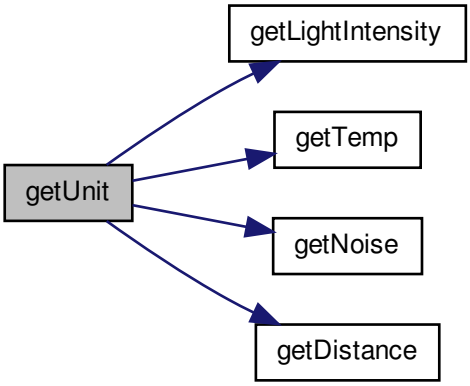
**Parameters**

| | |
|---|---|
| *the* | port which was measured |

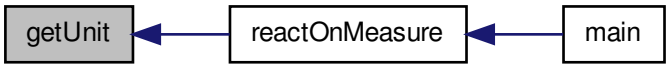
**Return values**

| | |
|---|---|
| *the* | recalculated measurement |

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.1.2.3 int main ( void )

the main method with statemachine loop as follows: statemachine ----!!!

enable interrupts

init ports

init scalings

init uart

welcome message uart

init can

CS12 1 CS11 0 CS10 1 clkI/O/1024 (From prescaler)

main loop

toggle status led

the measure loop

measure required ??

reaction required ??

led check required ??

-1 is the initial value

check the status of the led with the light sensor

check uart commands and send data by uart when activated

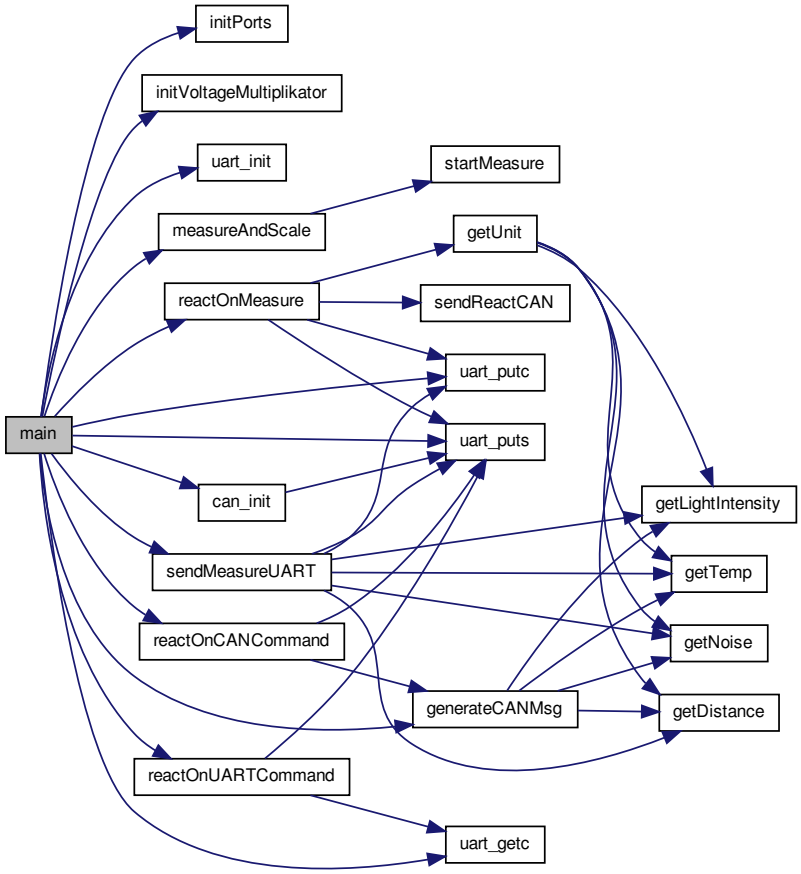
reaction required ??

measure required ??

check can commands and send data when activated

recalculate the heartBeat if the poti port is measured

Here is the call graph for this function:



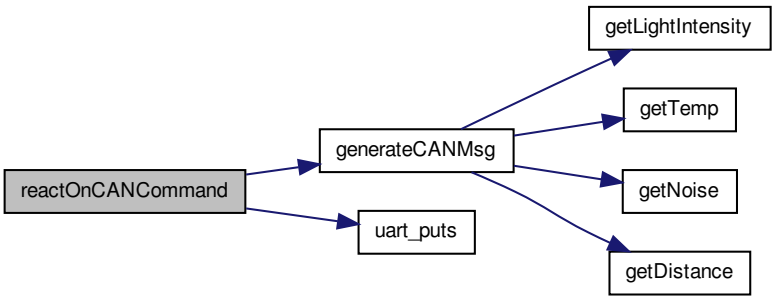

#### 5.1.2.4 uint8_t reactOnCANCommand ( char * *data* )

reaction for given commands

**Parameters**

| | |
|---|---|
| *the* | command given by can |

Here is the call graph for this function:




Here is the caller graph for this function:

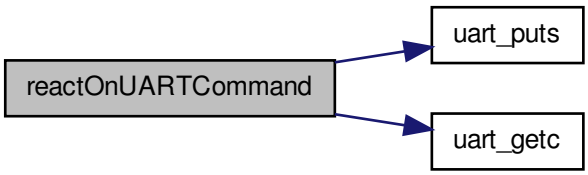

#### 5.1.2.5 uint8_t reactOnUARTCommand ( unsigned char *c* )

reaction for given commands

**Parameters**

| | |
|---|---|
| *the* | command given by uart |

Here is the call graph for this function:




Here is the caller graph for this function:


#### 5.1.2.6 void sendMeasureUART ( uint8_t *port* )

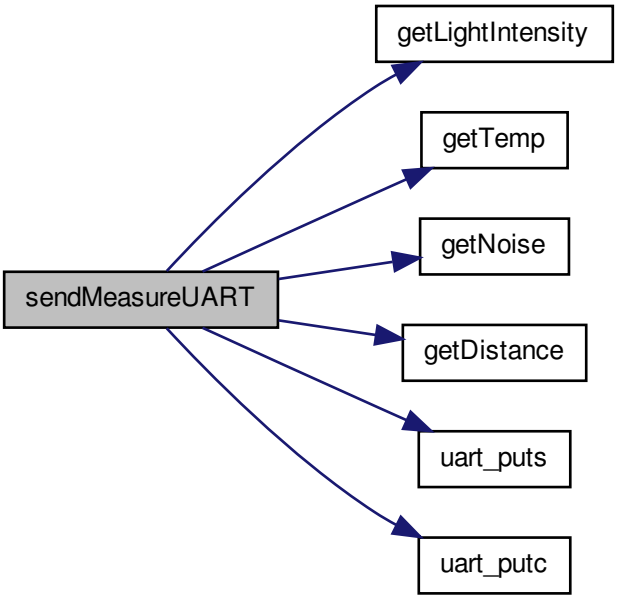
send measured data by uart Format: P:n;x y

--- n = portnumber, x = value, y = unit

**Parameters**

| | |
|---|---|
| *port* | to measure |

Here is the call graph for this function:




Here is the caller graph for this function:

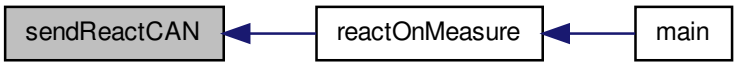

#### 5.1.2.7 void sendReactCAN ( uint8_t *port,* char *HoL* )

send a msg over CAN that the system has reacted format: Hn , Ln; H when switched off (relais high) and L when back on, n is the port

**Parameters**

| | |
|---|---|
| *the* | port, high or low (H,L) |

Here is the caller graph for this function:




#### 5.1.2.8 void sleep ( uint32_t *val* )

sleep methode self-made sleep method for better timing

**Parameters**

| | |
|---|---|
| *the* | val in ms of sleeping |

### 5.1.3 Variable Documentation

#### 5.1.3.1 uint8_t portModeMeasure[]

**Initial value:**

```
{Temp,
                Light,
                Distance,
                Vol,
                Distance,
                Distance,
                Vol,
                0}
```

modes how we can measure

## 5.2 CAN bus library

CAN Library This library can be used to transmit and receive data through the CAN Bus.

### Data Structures

- struct **tCAN**

  *The normal CAN message length 4 byte.*
- struct **tExtendedCAN**

  *The extended CAN message length 8 byte This message type is used in this projekt, dont use other, this will cause problems with the CAN-usb cable.*

### Defines

- #define **MCP2515_FILTER_EXTENDED**(id)
- #define **MCP2515_FILTER**(id)

### Functions

- uint8_t **spi_putc** (uint8_t data)
- void **mcp2515_write_register** (uint8_t adress, uint8_t data)
- uint8_t **mcp2515_read_register** (uint8_t adress)
- void **mcp2515_bit_modify** (uint8_t adress, uint8_t mask, uint8_t data)
- uint8_t **mcp2515_read_status** (uint8_t type)
- uint8_t **mcp2515_init** (void)
- uint8_t **mcp2515_check_message** (void)
- uint8_t **can_get_message** (**tCAN** ∗message)
- uint8_t **mcp2515_get_extmessage** (**tExtendedCAN** ∗message)
- uint8_t **can_send_message** (**tCAN** ∗message)
- uint8_t **mcp2515_send_extmessage** (**tExtendedCAN** ∗message)
- void **generate_extCAN_ID** (uint8_t ∗bytes, char ∗resultchars)
- void **mcp2515_static_filter** (PGM_P filter)
- void **print_can_message** (**tCAN** ∗message)
- void **testCan** (void)

### Variables

- uint16_t **id**
- int8_t **rtr**: 1
- uint8_t **length**: 4
- struct {
  int8_t **rtr**: 1
  uint8_t **length**: 4
  } **header**

- uint8_t **data** [8]
- char **id** [4]
- int8_t **rtr**: 1
- uint8_t **length**: 4
- struct {
  int8_t **rtr**: 1
  uint8_t **length**: 4
  } **header**

- uint8_t **data** [8]

### 5.2.1 Detailed Description

CAN Library This library can be used to transmit and receive data through the CAN Bus.

### 5.2.2 Define Documentation

#### 5.2.2.1 #define MCP2515_FILTER( *id* )

**Value:**

```
(uint8_t)((uint32_t) id >> 3), \
                (uint8_t)((uint32_t) id << 5), \
                0, \
                0
```

#### 5.2.2.2 #define MCP2515_FILTER_EXTENDED( *id* )

**Value:**

```
(uint8_t)  ((uint32_t) id >> 21), \
                (uint8_t)((((uint32_t) id >> 13) & 0xe0) | (1<<3) | \
                        (((uint32_t) id >> 16) & 0x3)), \
                (uint8_t)  ((uint32_t) id >> 8), \
                (uint8_t)  ((uint32_t) id)
```

## 5.3 UART Library

Interrupt UART library using the built-in UART with transmit and receive circular buffers.

### Defines

- #define **UART_BAUD_SELECT**(baudRate, xtalCpu) ((xtalCpu)/((baudRate)∗16l)-1)

  *UART Baudrate Expression.*
- #define **UART_BAUD_SELECT_DOUBLE_SPEED**(baudRate, xtalCpu) (((xtalCpu)/((baudRate)∗8l)-1)|0x8000)

  *UART Baudrate Expression for ATmega double speed mode.*
- #define **UART_FRAME_ERROR** 0x0800 /∗ Framing Error by UART ∗/
- #define **UART_OVERRUN_ERROR** 0x0400 /∗ Overrun condition by UART ∗/
- #define **UART_BUFFER_OVERFLOW** 0x0200 /∗ receive ringbuffer overflow ∗/
- #define **UART_NO_DATA** 0x0100 /∗ no receive data available ∗/
- #define **uart_puts_P**(__s) **uart_puts_p**(PSTR(__s))

  *Macro to automatically put a string constant into program memory.*
- #define **uart1_puts_P**(__s) **uart1_puts_p**(PSTR(__s))

  *Macro to automatically put a string constant into program memory.*

### Functions

- void **uart_init** (unsigned int baudrate)

  *Initialize UART and set baudrate.*
- unsigned int **uart_getc** (void)

  *Get received byte from ringbuffer.*
- void **uart_putc** (unsigned char data)

  *Put byte to ringbuffer for transmitting via UART.*
- void **uart_puts** (const char ∗s)

  *Put string to ringbuffer for transmitting via UART.*
- void **uart_puts_p** (const char ∗s)

  *Put string from program memory to ringbuffer for transmitting via UART.*
- void **uart1_init** (unsigned int baudrate)

  *Initialize USART1 (only available on selected ATmegas)*
- unsigned int **uart1_getc** (void)

  *Get received byte of USART1 from ringbuffer. (only available on selected ATmega)*
- void **uart1_putc** (unsigned char data)

  *Put byte to ringbuffer for transmitting via USART1 (only available on selected ATmega)*
- void **uart1_puts** (const char ∗s)

  *Put string to ringbuffer for transmitting via USART1 (only available on selected ATmega)*
- void **uart1_puts_p** (const char ∗s)

  *Put string from program memory to ringbuffer for transmitting via USART1 (only available on selected ATmega)*

### 5.3.1 Detailed Description

Interrupt UART library using the built-in UART with transmit and receive circular buffers.

```
#include <uart.h>
```

This library can be used to transmit and receive data through the built in UART.

An interrupt is generated when the UART has finished transmitting or receiving a byte. The interrupt handling routines use circular buffers for buffering received and transmitted data.

The UART_RX_BUFFER_SIZE and UART_TX_BUFFER_SIZE constants define the size of the circular buffers in bytes. Note that these constants must be a power of 2. You may need to adapt this constants to your target and your application by adding -CDEFS += -DUART_RX_BUFFER_SIZE=nn -DUART_RX_BUFFER_SIZE=nn to your Makefile.

**Note**

Based on Atmel Application Note AVR306

**Author**

Peter Fleury `pfleury@gmx.ch` `http://jump.to/fleury`

### 5.3.2 Define Documentation

#### 5.3.2.1 #define UART_BAUD_SELECT( *baudRate, xtalCpu* ) ((xtalCpu)/((baudRate)∗16l)-1)

UART Baudrate Expression.

**Parameters**

| | |
|---|---|
| *xtalcpu* | system clock in Mhz, e.g. 4000000L for 4Mhz |
| *baudrate* | baudrate in bps, e.g. 1200, 2400, 9600 |

#### 5.3.2.2 #define UART_BAUD_SELECT_DOUBLE_SPEED( *baudRate, xtalCpu* ) (((xtalCpu)/((baudRate)∗8l)-1)|0x8000)

UART Baudrate Expression for ATmega double speed mode.

**Parameters**

| | |
|---|---|
| *xtalcpu* | system clock in Mhz, e.g. 4000000L for 4Mhz |
| *baudrate* | baudrate in bps, e.g. 1200, 2400, 9600 |

#### 5.3.2.3 #define UART_FRAME_ERROR 0x0800 /* Framing Error by UART */

Size of the circular receive buffer, must be power of 2 Size of the circular transmit buffer, must be power of 2

### 5.3.3 Function Documentation

#### 5.3.3.1 unsigned int uart1_getc ( void )

Get received byte of USART1 from ringbuffer. (only available on selected ATmega)

**See also**

**uart_getc** (p. 25)

#### 5.3.3.2 void uart1_init ( unsigned int *baudrate* )

Initialize USART1 (only available on selected ATmegas)

**See also**

**uart_init** (p. 25)

#### 5.3.3.3 void uart1_putc ( unsigned char *data* )

Put byte to ringbuffer for transmitting via USART1 (only available on selected ATmega)

**See also**

**uart_putc** (p. 26)

#### 5.3.3.4 void uart1_puts ( const char * *s* )

Put string to ringbuffer for transmitting via USART1 (only available on selected ATmega)

**See also**

**uart_puts** (p. 27)

#### 5.3.3.5 void uart1_puts_p ( const char * *s* )

Put string from program memory to ringbuffer for transmitting via USART1 (only available on selected ATmega)

**See also**

**uart_puts_p** (p. 28)

#### 5.3.3.6 unsigned int uart_getc ( void )

Get received byte from ringbuffer.

Returns in the lower byte the received character and in the higher byte the last receive error. UART_NO_DATA is returned when no data is available.

**Parameters**

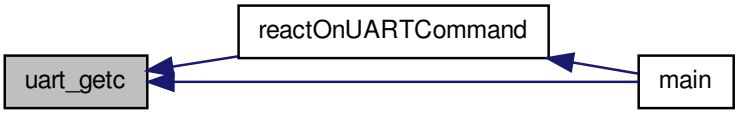
| | |
|---|---|
| *void* | |

**Returns**

lower byte: received byte from ringbuffer
higher byte: last receive status

- **0** successfully received data from UART
- **UART_NO_DATA**
  no receive data available
- **UART_BUFFER_OVERFLOW**
  Receive ringbuffer overflow. We are not reading the receive buffer fast enough, one or more received character have been dropped
- **UART_OVERRUN_ERROR**
  Overrun condition by UART. A character already present in the UART UDR register was not read by the interrupt handler before the next character arrived, one or more received characters have been dropped.
- **UART_FRAME_ERROR**
  Framing Error by UART

Here is the caller graph for this function:




#### 5.3.3.7 void uart_init ( unsigned int *baudrate* )

Initialize UART and set baudrate.

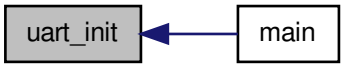
**Parameters**

| | |
|---|---|
| *baudrate* | Specify baudrate using macro **UART_BAUD_SELECT()** (p. 23) |

**Returns**

none

Here is the caller graph for this function:




#### 5.3.3.8 void uart_putc ( unsigned char *data* )

Put byte to ringbuffer for transmitting via UART.

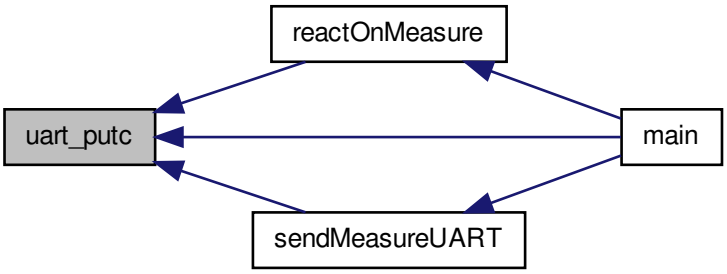
**Parameters**

| | |
|---|---|
| *data* | byte to be transmitted |

**Returns**

none

Here is the caller graph for this function:




#### 5.3.3.9 void uart_puts ( const char * s )

Put string to ringbuffer for transmitting via UART.

The string is buffered by the uart library in a circular buffer and one character at a time is transmitted to the UART using interrupts. Blocks if it can not write the whole string into the circular buffer.

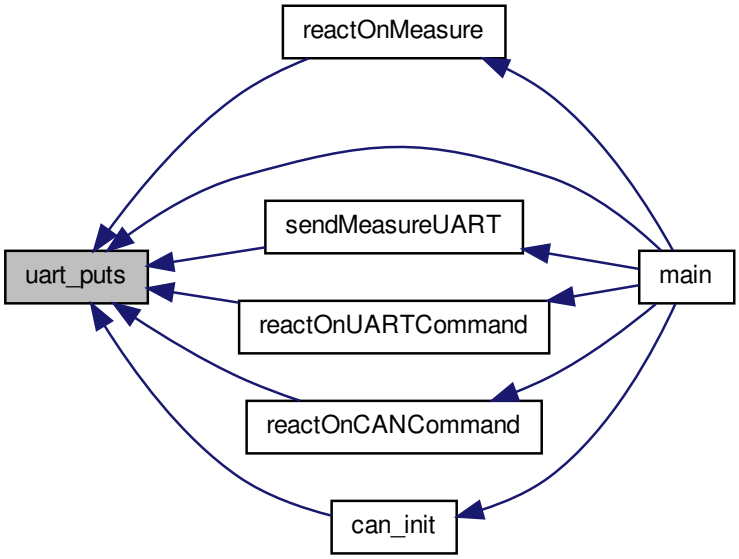
**Parameters**

| | |
|---|---|
| *s* | string to be transmitted |

**Returns**

none

Here is the caller graph for this function:




#### 5.3.3.10 void uart_puts_p ( const char * s )

Put string from program memory to ringbuffer for transmitting via UART.

The string is buffered by the uart library in a circular buffer and one character at a time is transmitted to the UART using interrupts. Blocks if it can not write the whole string into the circular buffer.

**Parameters**

| | |
|---|---|
| *s* | program memory string to be transmitted |

**Returns**

none

See also

**uart_puts_P** (p. 22)

# Chapter 6

# Data Structure Documentation

## 6.1 tCAN Struct Reference

The normal CAN message length 4 byte.

`#include <mcp2515.h>`

### Data Fields

- uint16_t **id**
- struct {
  int8_t **rtr**: 1
  uint8_t **length**: 4
  } **header**

- uint8_t **data** [8]

### 6.1.1 Detailed Description

The normal CAN message length 4 byte.

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

- workspace/Uniprojekt/Platine16Mhz/Abgabe/**mcp2515.h**

## 6.2 tExtendedCAN Struct Reference

The extended CAN message length 8 byte This message type is used in this projekt, dont use other, this will cause problems with the CAN-usb cable.

`#include <mcp2515.h>`

### Data Fields

- char **id** [4]
- struct {
  int8_t **rtr**: 1
  uint8_t **length**: 4
  } **header**

- uint8_t **data** [8]

### 6.2.1 Detailed Description

The extended CAN message length 8 byte This message type is used in this projekt, dont use other, this will cause problems with the CAN-usb cable.

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

- workspace/Uniprojekt/Platine16Mhz/Abgabe/**mcp2515.h**

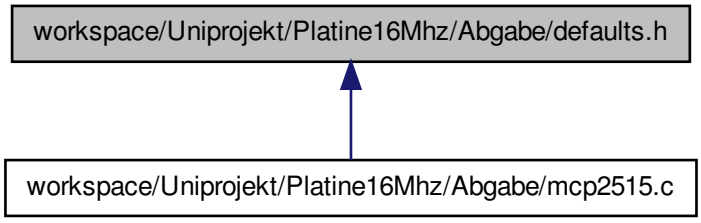
# Chapter 7

# File Documentation

## 7.1 workspace/Uniprojekt/Platine16Mhz/Abgabe/defaults.h File - Reference

DefaultConfig here the pins for the relaispins, SPI, status and error leds and some names are defined.

This graph shows which files directly or indirectly include this file:




### Defines

- #define **LOGGING** 1
- #define **isReactive** 1
- #define **DONT_REAC** 500
- #define **NO_RELAIS** 111
- #define **HIGHER** 1
- #define **LOWER** 0

- #define **CLOCK_REGULATOR** 6
- #define **isON** PB0
- #define **showON** PC0
- #define **lastPin1** PC1
- #define **lastPin2** PC2
- #define **lastPin3** PC3
- #define **errorLed** PC7
- #define **P_MOSI** PB5
- #define **P_MISO** PB6
- #define **P_SCK** PB7
- #define **P_CS** PB4
- #define **ETH2CAN_ID** 0x00
- #define **MOTOR_ID** 0x20
- #define **COMPASS_ID** 0x40
- #define **REKICK_ID** 0x60
- #define **SERVO_ID** 0x80
- #define **M_BOARD_ID** 0x50
- #define **PRIORITY_HIGH** 0x20
- #define **PRIORITY_NORM** 0x40
- #define **PRIORITY_LOW** 0x80

### 7.1.1 Detailed Description

DefaultConfig here the pins for the relaispins, SPI, status and error leds and some names are defined.

## 7.2 workspace/Uniprojekt/Platine16Mhz/Abgabe/mBoard.h File - Reference

### Defines

- #define **TIMERCOUNTER** TCNT0

  *led test counter*
- #define **TIMERCONTROL_B** TCCR0B

  *prescaler led counter*
- #define **ACK_CAN** 10

  *ports are ranged from 0 to 7 --- 10 for ACK*

### Enumerations

- enum **portMeasureModes** { **NOT_USED**, **Temp**, **Light**, **Vol**, **Noise**, **Distance**, **Led** }

  *values interval when reaction is triggerd, without interval the relais jitters*

## Functions

- void **sleep** (uint32_t val)

  *sleep methode self-made sleep method for better timing*
- void **initPorts** (void)

  *init the ports used as in or output and init the ADC with 125 kHz*
- void **initVoltageMultiplikator** (void)

  *init the multiplicator for the voltage calc 1, 5,24, or 111 for 400 V*
- double **getTemp** (void)

  *get the temperature measured with the temp-sensor*
- double **getLightIntensity** (void)

  *get the light intensity with a level of 0-100, 0 means its dark and 100 is very bright*
- double **getNoise** (void)

  *get noise level with the mic sensor, with level 0-100 100 is very loud*
- double **getDistance** (void)

  *get the distance to the next object near to the ultrasonic sensor*
- uint16_t **startMeasure** (unsigned int X)

  *measure voltage on port x*
- void **measureAndScale** (uint8_t port)

  *use startMeasure to measure and scale with given scalingfactor*
- double **getUnit** (uint8_t port)

  *calculate the value of the real unit e.g. Distance in cm from measure*
- void **sendReactCAN** (uint8_t port, char HoL)

  *send a msg over CAN that the system has reacted format: Hn , Ln; H when switched off (relais high) and L when back on, n is the port*
- void **reactOnMeasure** (uint8_t port)

  *reacts on measured values e.g. by setting a port High or Low to switch the relais --- sends a Hn*
  *when set High Ln*
  *when Low, n is the port*
- void **sendMeasureUART** (uint8_t port)

  *send measured data by uart Format: P:n;x y*
  *--- n = portnumber, x = value, y = unit*
- void **generateCANMsg** (**tExtendedCAN** ∗M, uint8_t port)

  *generates a can message format: bits [0-5] value, bit[6] unit, bit [7] port*
- uint8_t **readUARTConfig** (unsigned char conf)
- uint8_t **readUARTConfModes** (unsigned char conf)
- uint8_t **reactOnUARTCommand** (unsigned char c)

  *reaction for given commands*
- uint8_t **reactOnCANCommand** (char ∗data)

  *reaction for given commands*
- void **can_init** (void)

  *initialize the can driver*
- int **main** (void)

  *the main method with statemachine loop as follows: statemachine ----!!!*

### Variables

- uint8_t **measuringVoltageFaktor** [8]

  *the scalings (1,5,26,111 to calc the voltage, 0 means no measurement. 400 -> 111 as voltagefaktor*
- char **voltageValueBuffer** [7]

  *char buffer for measure conversion in string*
- double **measurement**

  *the measurement*
- double **ledLastMeasure**

  *led test last measure*
- uint8_t **ledTick** = 150
- uint8_t **serialCommunikation** = 1

  *en-/disable serial line*
- uint8_t **canCommunikation** = 1

  *en-/disable can*
- uint8_t **ethCommunikation** = 0

  *en-/disable eternet line*
- uint8_t **portToRelais** [8]
- uint8_t **reactionValueMode** [8]

  *which port korresponds to which relais to react*
- uint32_t **reactionValues** [8]

  *should be switched when measure is higher or lower val*
- uint8_t **portModeMeasure** [ ]

  *modes how we can measure*
- double **intervalValues** [7]

  *which measure mode on specific port*
- double **distanceScale** = 250

  *the specific relais-switch interval to handle relais jittering: Temp 1C, Light 7%, Vol 0,5, Distance 5cm*
- double **heartBeat** = 180.0

  *Vcc/512 ∗ 2,54 (inch -> cm)*
- double **mea** [200]

  *standard beat 330 ms tune up to 1,5 s with poti*
- double **loopCounter** = 0.0
- char **te** [33]

  *the counter for the inner measure loop*

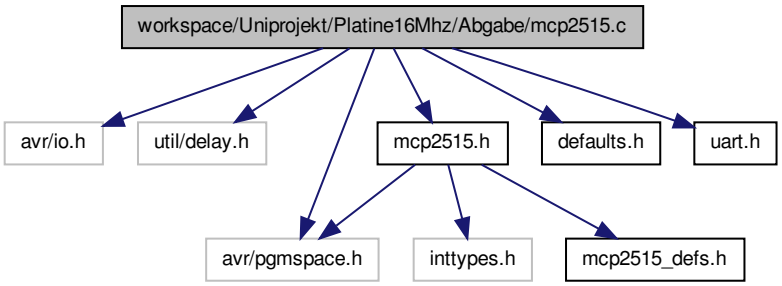
#### 7.2.1 Detailed Description

The main loop

## 7.3 workspace/Uniprojekt/Platine16Mhz/Abgabe/mcp2515.c File - Reference

the functions to interact with the can bus controller and the spi bus

`#include <avr/io.h>` `#include <util/delay.h>` `#include <avr/pgmspace.h>` `#include "mcp2515.h"` `#include "defaults.-h"` `#include "uart.h"` Include dependency graph for mcp2515.c:




### Functions

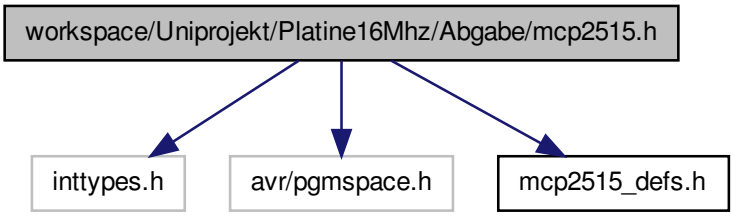
- uint8_t **spi_putc** (uint8_t data)
- void **spi_init** (void)
- void **mcp2515_bit_modify** (uint8_t adress, uint8_t mask, uint8_t data)
- void **testCan** (void)
- void **print_can_message** (**tCAN** ∗message)
- uint8_t **mcp2515_init** (void)
- void **mcp2515_write_register** (uint8_t adress, uint8_t data)
- uint8_t **mcp2515_read_register** (uint8_t adress)
- uint8_t **mcp2515_read_rx_status** (void)
- uint8_t **can_get_message** (**tCAN** ∗p_message)
- uint8_t **mcp2515_read_status** (uint8_t type)
- uint8_t **can_send_message** (**tCAN** ∗p_message)
- uint8_t **mcp2515_get_extmessage** (**tExtendedCAN** ∗message)
- uint8_t **mcp2515_send_extmessage** (**tExtendedCAN** ∗message)

### 7.3.1 Detailed Description

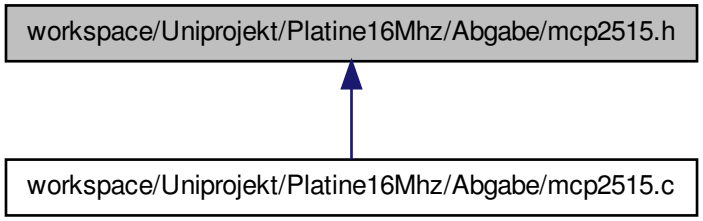
the functions to interact with the can bus controller and the spi bus

## 7.4 workspace/Uniprojekt/Platine16Mhz/Abgabe/mcp2515.h File - Reference

`#include <inttypes.h> #include <avr/pgmspace.h> #include "mcp2515_defs.h"` Include dependency graph for mcp2515.h:




This graph shows which files directly or indirectly include this file:




### Data Structures

- struct **tCAN**

  *The normal CAN message length 4 byte.*

- struct **tExtendedCAN**

  *The extended CAN message length 8 byte This message type is used in this projekt, dont use other, this will cause problems with the CAN-usb cable.*

### Defines

- #define **MCP2515_FILTER_EXTENDED**(id)
- #define **MCP2515_FILTER**(id)

### Functions

- uint8_t **spi_putc** (uint8_t data)
- void **mcp2515_write_register** (uint8_t adress, uint8_t data)
- uint8_t **mcp2515_read_register** (uint8_t adress)
- void **mcp2515_bit_modify** (uint8_t adress, uint8_t mask, uint8_t data)
- uint8_t **mcp2515_read_status** (uint8_t type)
- uint8_t **mcp2515_init** (void)
- uint8_t **mcp2515_check_message** (void)
- uint8_t **can_get_message** (**tCAN** ∗message)
- uint8_t **mcp2515_get_extmessage** (**tExtendedCAN** ∗message)
- uint8_t **can_send_message** (**tCAN** ∗message)
- uint8_t **mcp2515_send_extmessage** (**tExtendedCAN** ∗message)
- void **generate_extCAN_ID** (uint8_t ∗bytes, char ∗resultchars)
- void **mcp2515_static_filter** (PGM_P filter)
- void **print_can_message** (**tCAN** ∗message)
- void **testCan** (void)

#### 7.4.1 Detailed Description

# Index