SENSOR TERMINAL BOARD by dresden elektronik
Reference Manual

Generated by Doxygen 1.5.1-p1

Mon Nov 5 13:22:14 2007

Contents

1 SENSOR TERMINAL BOARD by dresden elektronik Main Page **1**

1.1 Introduction 1

1.2 Copyright 1

1.3 Installation 1

1.4 Getting Startet 2

1.5 Examples 3

1.6 CDROM Directory Structure 3

2 SENSOR TERMINAL BOARD by dresden elektronik Data Structure Index **5**

2.1 SENSOR TERMINAL BOARD by dresden elektronik Data Structures 5

3 SENSOR TERMINAL BOARD by dresden elektronik File Index **7**

3.1 SENSOR TERMINAL BOARD by dresden elektronik File List . 7

4 SENSOR TERMINAL BOARD by dresden elektronik Page Index **9**

4.1 SENSOR TERMINAL BOARD by dresden elektronik Related Pages 9

5 SENSOR TERMINAL BOARD by dresden elektronik Data Structure Documentation **11**

5.1 association_entry_t Struct Reference 11

5.2 coord_status_t Struct Reference 13

5.3 device_status_t Struct Reference 14

6 SENSOR TERMINAL BOARD by dresden elektronik File Documentation 17

6.1 Doc/Doxygen/SensTermBoard.dox File Reference 17

6.2 Examples/Button/main.c File Reference 18

6.3 Examples/LEDLoop/main.c File Reference 25

6.4 Examples/LEDTimer/main.c File Reference 29

6.5 Examples/Temperature/main.c File Reference 34

6.6 Examples/TWIEeprom/main.c File Reference 44

6.7 Examples/Uart/main.c File Reference 61

6.8 Examples/Usb/main.c File Reference 66

6.9 Examples/XRAM/main.c File Reference 71

6.10 Examples/Z-LinkWireless/coord.c File Reference 77

6.11 Examples/Z-LinkWireless/device.c File Reference 97

7 SENSOR TERMINAL BOARD by dresden elektronik Page Documentation 111

7.1 copyright of the example applications 111

7.2 Getting Startet 113

7.3 PCB SensTermBoard 114

7.4 PCB Radio Controller Board 115

Chapter 1

SENSOR TERMINAL BOARD by dresden elektronik Main Page

1.1 Introduction

This CDROM contains all necessary tools for developing, building and debugging software for the SensTermBoard on the Windows (TM) plattform. All **examples** (p. 3) are tested with the listed and delivered software versions.

1.2 Copyright

The different included software packages are subject to different licences. See its detailed conditions while installing an application. Every included application was choosen instead of others because its special licence allows the use and redistribution for personal and commercial purposes. The included examples are also open source as you can read on the **copyright page** (p. 111).

1.3 Installation

1.3.1 Applications

For installation of the required applications you can start the software installers directly from this documentation:

- `AVR-Studio version 4.13 B557`: Free IDE from ATMEL
- `WinAVR version 20070525`: gnu compiler collection tool chain, gcc version 4.1.1

- `rdk230wpan version 1.3`: free MAC layer software stack from ATMEL for 802.15.4 mesh networks for AT89RF230 radio and ATMEGA1281 controller

1.3.2 Attention!

The electrical circuit of the SensTermBoard does not allow the parallel use of the batteries and the USB/DC connection. Please make sure that the batteries are taken out of the RCB when an external power source is connected.

1.3.3 USB-Driver

When you plug the PC's USB cable into the SensTermBoard, Windows (TM) asks you for the location of the USB driver for the board. It can be found in `".\3dParty\FTDI\CDM 2.00.00"` on the CDROM. There is also a newer version (CDM 2.02.04 WHQL Certified), but this one has a serious bug at the API layer and is not recommended. FTDI is working on a new version.

1.3.4 Building The Documentation

This documentation was generated by the use of the following tools

- `Doxygen 1.5.1 p1`: free source documentation system from Dimitri van Heesch
- `graphviz-2.12.exe`: free generator for tree graphs from AT&T Research Labs

which are called in the batch file

```
.\Doc\Doxygen\doxygen.bat
```

1.3.5 Documentation As PDF

The related manual `refman.pdf` was generated out of the doxygen output by using the tools

- `LaTeX`
- `GhostScript`

1.4 Getting Startet

For a short guide how to build and debug applications on the SensTermBoard see the page **Getting Startet** (p. 113). There you can find a step by step guide from powering up to debugging.

1.5 Examples

On the CDROM are a lot of miniature example applications which are demonstrating a special part of the SensTermBoard. You can find its documentation on the `files page`.

1.6 CDROM Directory Structure

```
+--- 3dParty              Applications, Sources, Drivers
|   +--- ATMEL            IDE, RF Sources
|   |   +--- AVR-Studio   IDE
|   |   +--- rdk230wpan   RF Sources
|   +--- Doxygen          Source documentation
|   +--- FTDI             USB-Chip Driver
|   +--- WinAVR           Open Source ANSI-C compiler tool chain
+--- Doc                  Documentation around the SensTermBoard
|   +--- DataSheets       Data sheets for  ATMEGA1281,  AT86RF230
|   +--- Doxygen          this documentation and its config files
|   +--- Images           Images used for this documentation
|   +--- PCB              PCB files (schematic, layout) for SensTermBoard
(p. 114) and RCB (p. 115)
+--- Examples             examples for the the board's components
    +---Button (p. 18)          demonstrates init and use of the button
    +---LEDLoop (p. 25)         Blinking LED under use of a delay loop
    +---LEDTimer (p. 29)        Blinking LED under use of timer 4
    +---Temperature (p. 34)     Using the thermistor for temperature measurement
    +---TWIEeprom (p. 44)       Using the I2C / TWI modul for access to an external eeprom
    +---Uart (p. 61)            "Hello World" with 9600 Baud over the UART0
    +---Usb (p. 66)             "Hello World" over the USB chip FT245
    +---XRAM (p. 71)            How to use the external 32k (XRAM)
    +--- Z-LinkWireless   Devices (p. 97) notify a Coordinator
(p. 77) via RF about its temperatures, which it sends over USB to a PC
```

Chapter 2

SENSOR TERMINAL BOARD by dresden elektronik Data Structure Index

2.1 SENSOR TERMINAL BOARD by dresden elektronik Data Structures

Here are the data structures with brief descriptions:

association_entry_t 11
coord_status_t 13
device_status_t 14

Chapter 3

SENSOR TERMINAL BOARD by dresden elektronik File Index

3.1 SENSOR TERMINAL BOARD by dresden elektronik File List

Here is a list of all files with brief descriptions:

Examples/Button/**main.c** (SensTermBoard Example: LED blinks with timer interrupt) 18
Examples/LEDLoop/**main.c** (SensTermBoard Example: LED Blinks with delay loop) 25
Examples/LEDTimer/**main.c** (SensTermBoard Example: LED blinks with timer interrupt) 29
Examples/Temperature/**main.c** (SensTermBoard Example: thermometer function over USB) 34
Examples/TWIEeprom/**main.c** (SensTermBoard Example: using of eeprom function over USB) 44
Examples/Uart/**main.c** (SensTermBoard Example: "Hello World" over UART 0) 61
Examples/Usb/**main.c** (SensTermBoard Example: "Hello World" over USB chip FT245) 66
Examples/XRAM/**main.c** (SensTermBoard Example: How to use the external 32k (XRAM)) 71
Examples/Z-LinkWireless/**coord.c** (SensTermBoard Example: Coordinator receives temperature measures from devices) 77
Examples/Z-LinkWireless/**device.c** (SensTermBoard Example: Device measures temperature and sends it to a coordinator) . . 97

Chapter 4

SENSOR TERMINAL BOARD by dresden elektronik Page Index

4.1 SENSOR TERMINAL BOARD by dresden elektronik Related Pages

Here is a list of all related documentation pages:

copyright of the example applications 111
Getting Startet 113
PCB SensTermBoard 114
PCB Radio Controller Board 115

Chapter 5

SENSOR TERMINAL BOARD by dresden elektronik Data Structure Documentation

5.1 association_entry_t Struct Reference

Data Fields

- bool **associated**
- uint64_t **long_addr**
- uint16_t **temperature**

5.1.1 Detailed Description

Definition at line 96 of file coord.c.

5.1.2 Field Documentation

5.1.2.1 bool association_entry_t::associated

Definition at line 98 of file coord.c.

Referenced by usr_mcps_data_ind(), and usr_mlme_comm_status_ind().

5.1.2.2 uint64_t association_entry_t::long_addr

Definition at line 99 of file coord.c.

Referenced by association_table_init(), and mac_register_device().

5.1.2.3 uint16_t association_entry_t::temperature

Definition at line 100 of file coord.c.

Referenced by usr_mcps_data_ind().

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

- Examples/Z-LinkWireless/**coord.c**

5.2 coord_status_t Struct Reference

Data Fields

- uint8_t **handle**
- uint8_t **led_value**
- **coord_state_t state**

5.2.1 Detailed Description

Definition at line 114 of file coord.c.

5.2.2 Field Documentation

5.2.2.1 uint8_t coord_status_t::handle

Definition at line 116 of file coord.c.

Referenced by application_init(), and mac_send_data().

5.2.2.2 uint8_t coord_status_t::led_value

Definition at line 117 of file coord.c.

Referenced by application_init(), and mac_send_data().

5.2.2.3 coord_state_t coord_status_t::state

Definition at line 118 of file coord.c.

Referenced by usr_mlme_associate_ind(), usr_mlme_reset_conf(), usr_mlme_scan_conf(), usr_mlme_set_conf(), and usr_mlme_start_conf().

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

- Examples/Z-LinkWireless/**coord.c**

5.3 device_status_t Struct Reference

Data Fields

- bool **led**
- bool **switch_pressed**
- uint16_t **device_short_address**
- uint8_t **coord_address_mode**
- uint64_t **coord_address**
- uint16_t **pan_id**
- uint8_t **logical_channel**
- uint8_t **msdu_handle**
- **device_state_t state**

5.3.1 Detailed Description

Definition at line 86 of file device.c.

5.3.2 Field Documentation

5.3.2.1 bool device_status_t::led

Definition at line 88 of file device.c.

5.3.2.2 bool device_status_t::switch_pressed

Definition at line 89 of file device.c.

5.3.2.3 uint16_t device_status_t::device_short_address

Definition at line 90 of file device.c.

Referenced by switch_task(), and usr_mlme_associate_conf().

5.3.2.4 uint8_t device_status_t::coord_address_mode

Definition at line 91 of file device.c.

Referenced by mac_associate(), switch_task(), and usr_mlme_scan_conf().

5.3.2.5 uint64_t device_status_t::coord_address

Definition at line 92 of file device.c.

Referenced by mac_associate(), switch_task(), and usr_mlme_scan_conf().

5.3.2.6 uint16_t device_status_t::pan_id

Definition at line 93 of file device.c.

Referenced by mac_associate(), switch_task(), usr_mcps_data_ind(), and usr_mlme_scan_conf().

5.3.2.7 uint8_t device_status_t::logical_channel

Definition at line 94 of file device.c.

Referenced by mac_associate(), and usr_mlme_scan_conf().

5.3.2.8 uint8_t device_status_t::msdu_handle

Definition at line 95 of file device.c.

Referenced by switch_task().

5.3.2.9 device_state_t device_status_t::state

Definition at line 96 of file device.c.

Referenced by switch_task(), usr_mcps_data_ind(), usr_mlme_associate_conf(), usr_mlme_reset_conf(), and usr_mlme_scan_conf().

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

- Examples/Z-LinkWireless/**device.c**

Chapter 6

SENSOR TERMINAL BOARD by dresden elektronik File Documentation

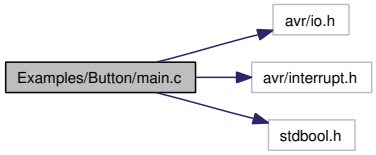
6.1 Doc/Doxygen/SensTermBoard.dox File Reference

6.2 Examples/Button/main.c File Reference

SensTermBoard Example: LED blinks with timer interrupt.

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

Include dependency graph for main.c:



Defines

- #define **IO_LED_AD** (0x4000)
- #define **IO_BUT_AD** (0x4000)

Functions

- static void **led_init** (void)

  Initializes the led hardware.

- static void **led_set** (int led_nr, bool OnOff)

  Set LED nr's state to on or off.

- static void **button_init** (void)

  Initializes the button hardware.

- static bool **button_pressed** (void)

  Gives the button's state back.

- void **timer_init** (void)

  Init timer 4 for 1 millisecond interrupting.

- **ISR** (TIMER4_COMPB_vect)

  timer 4 capture and compare interrupt routine

- int **main** (void)

  Main function of the demo application.

Variables

- static volatile unsigned char * **pIO_LED** = (unsigned char *) IO_LED_AD
- static volatile unsigned char * **pIO_BUT** = (unsigned char *) IO_BUT_AD
- static uint8_t **LED** = 0x00

6.2.1 Detailed Description

SensTermBoard Example: LED blinks with timer interrupt.

Changes the active LED when the button is pressed. The button state is scaned from the timer 4 interrupt and debaunced through testing more times for the new state

Author:

Definition in file **main.c**.

6.2.2 Define Documentation

6.2.2.1 #define IO_BUT_AD (0x4000)

Definition at line 50 of file main.c.

6.2.2.2 #define IO_LED_AD (0x4000)

Definition at line 49 of file main.c.

6.2.3 Function Documentation

6.2.3.1 static void button_init (void) [static]

Initializes the button hardware.

Initializes the button hardware so that further calls to button_pressed works

Definition at line 103 of file main.c.

Referenced by main().

```
104 {
105   ;
106 }
```

Here is the caller graph for this function:

6.2.3.2 static bool button_pressed (void) [static]

Gives the button's state back.

Returns:

bool false: not pressed, true: pressed

Definition at line 116 of file main.c.

References pIO_BUT.

Referenced by ISR().

```
117 {
118   return ((*pIO_BUT & 0x01) ? true : false);
119 }
```

Here is the caller graph for this function:

6.2.3.3 ISR (TIMER4_COMPB_vect)

timer 4 capture and compare interrupt routine

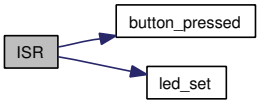
timer for interrupts every 1 millisecond with a compare event. The compare counter gets resetet and every 10 milliseconds gets the state of the button scanned for changes. On the first push button the active LED is changed and then waited for a 50 milliseconds long release of the button

Definition at line 148 of file main.c.

References button_pressed(), and led_set().

```
149 {
150   static int ms = 0;                          // memory for milliseconds
151   static bool led_0 = true;                   // memory for active led
152   static int  button_cnt = 0;                 // counter for button debounce
153   static bool button_state = false;           // memory for button state
154
155   TCNT4 = 0;                                  // clear timer
156   if (10 <= ++ms)                             // 10 milliseconds over?
157   {
158     ms = 0;                                   // reset milliseconds counter
159
160     switch (button_state)                     // button is in...
161     {
162       case false:                             // Off State?
163         if (button_pressed())                 // button pressed?
164         {
165           button_state = true;                // On State
166           led_set(0, led_0);                  // Set LED 0 to on, if activ, otherwise off
167           led_set(1,!led_0);                  // Set LED 1 to on, if activ, otherwise off
168           led_0 = !led_0;                     // change active LED
169         }
170         break;
171
172       case true:                              // On State?
173         if (!button_pressed())                // button released?
174         {
175           if (5 <= ++button_cnt)              // for a minimum time of 50 milliseconds?
176           {
177             button_state = false;
178           }
179         }
180         else                                  // button bounced? reset counter
181         {
182           button_cnt = 0;
183         }
184         break;
185     }
186   }
187 }
```

Here is the call graph for this function:



6.2.3.4 static void led_init (void) [static]

Initializes the led hardware.

Initializes the led hardware so that further calls to led_set works

Definition at line 71 of file main.c.

Referenced by main().

```
72 {
73   XMCRA |= _BV(SRE); XMCRB = _BV(XMBK);            // enable external Memory Interface (IO, LED)
74 }
```

Here is the caller graph for this function:

6.2.3.5 static void led_set (int *led_nr*, bool *OnOff*) [static]

Set LED nr's state to on or off.

set LED numer 1 or 2's state to On or Off

Parameters:

led_nr 0 or 1
OnOff true or false

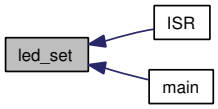
Definition at line 86 of file main.c.

References LED, and pIO_LED.

Referenced by ISR(), and main().

```
87 {
88   if (OnOff)                                      // when On
89     LED &= ~_BV(led_nr);                          // reset Bit
90   else                                            // otherwise
91     LED |= _BV(led_nr);                           // set it
92
93   *pIO_LED = ~LED | ~0x03;                        // Memory mapped IO
94 }
```

Here is the caller graph for this function:



6.2.3.6 int main (void)

Main function of the demo application.

This function consists of

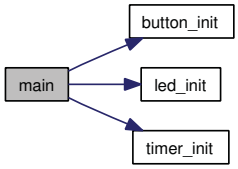
- The Initialicing Part (Sets all parts to default states)
- The Main-Loop (Endless loop with calls to all modules)

Definition at line 199 of file main.c.

References button_init(), led_init(), and timer_init().

```
200 {
201   led_init();                                   // init LED's hardware
202   timer_init();                                 // init timer 4 for interrupting
203   button_init();                                // init button for scanning
204   sei();                                        // enable interrupts
205
206   while(1)                                      // endless main loop does nothing
207   {                                             // work is done in timer 4 interrupt
208   }
209 }
```

Here is the call graph for this function:



6.2.3.7 void timer_init (void)

Init timer 4 for 1 millisecond interrupting.

Definition at line 126 of file main.c.

Referenced by main().

```
127 {
128
129   TCCR4B = _BV(CS42);                              // set clock source to prescaler F_CPU/256
130   OCR4B = (F_CPU / 256) / 1000;                    // timer interrupt every 1 millisecond
131   TCNT4 = 0;                                       // clear timer
132   TIMSK4 |= (1 << OCIE4B);                         // Enable timer interrupt
133   TIFR4  |= (1 << OCF4B);                          // clear capture flag
134 }
```

Here is the caller graph for this function:

6.2.4 Variable Documentation

6.2.4.1 uint8_t LED = 0x00 [static]

Definition at line 60 of file main.c.

Referenced by led_set().

6.2.4.2 volatile unsigned char* pIO_BUT = (unsigned char *) IO_BUT_AD [static]

Definition at line 58 of file main.c.

Referenced by button_pressed().

6.2.4.3 volatile unsigned char* pIO_LED = (unsigned char *) IO_LED_AD [static]

Definition at line 56 of file main.c.

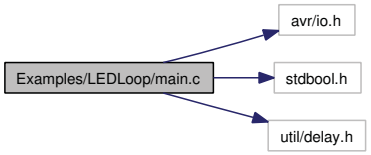
Referenced by led_set().

6.3 Examples/LEDLoop/main.c File Reference

SensTermBoard Example: LED Blinks with delay loop.

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

Include dependency graph for main.c:



Defines

- #define **IO_LED_AD** (0x4000)

Functions

- static void **delay_ms** (int ms)
  Delays program for ms milliseconds.
- static void **led_init** (void)
  Initializes the led hardware.
- static void **led_set** (int led_nr, bool OnOff)
  Set LED nr's state to on or off.
- int **main** (void)
  Main function of the demo application.

Variables

- static volatile unsigned char * **pIO_LED** = (unsigned char *) IO_LED_AD
- static uint8_t **LED** = 0x00

6.3.1 Detailed Description

SensTermBoard Example: LED Blinks with delay loop.

Blinks the LEDs with a busy waiting delay loop

Author:

Definition in file **main.c**.

6.3.2 Define Documentation

6.3.2.1 #define IO_LED_AD (0x4000)

Definition at line 47 of file main.c.

6.3.3 Function Documentation

6.3.3.1 static void delay_ms (int *ms*) [static]

Delays program for ms milliseconds.

the AVR-LIBC function _delay_ms is only able to delay for about 30ms (at a clock speed of 8 MHz). This function loops in steps of 1 millisecond and decrements the ms counter until it is zero.

Parameters:

ms Delay in milliseconds

Definition at line 71 of file main.c.

Referenced by main().

```
72 {
73   while (ms > 0)
74   {
75     _delay_ms(1);
76     ms--;
77   }
78 }
```

Here is the caller graph for this function:

6.3.3.2 static void led_init (void) [static]

Initializes the led hardware.

Initializes the led hardware so that further calls to led_set works

Definition at line 87 of file main.c.

```
88 {
89   XMCRA |= _BV(SRE); XMCRB = _BV(XMBK);            // enable external Memory Interface (IO, LED)
90 }
```

6.3.3.3 static void led_set (int *led_nr*, bool *OnOff*) [static]

Set LED nr's state to on or off.

set LED numer 1 or 2's state to On or Off

Parameters:

led_nr 0 or 1
OnOff true or false

Definition at line 102 of file main.c.

References LED, and pIO_LED.

```
103 {
104   if (OnOff)                                       // when On
105     LED &= ~_BV(led_nr);                           // reset Bit
```

```
106   else                                          // otherwise
107     LED |= _BV(led_nr);                         // set it
108
109   *pIO_LED = ~LED | ~0x03;                      // Memory mapped IO
110 }
```

6.3.3.4 int main (void)

Main function of the demo application.

This function consists of

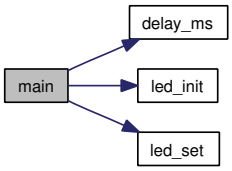
- The Initialicing Part (Sets all parts to default states)
- The Main-Loop (Endless loop with calls to all modules)

Definition at line 122 of file main.c.

References delay_ms(), led_init(), and led_set().

```
123 {
124   bool led_state = true;                        // memory for led state
125   led_init();
126
127   while(1)
128   {
129     led_set(0,  led_state);                     // Set actual LED state
130     led_set(1, !led_state);                     // Set actual LED state
131     led_state = !led_state;                     // invert led state
132     delay_ms(500);                              // busy waiting for 500 milliseconds
133   }
134 }
```

Here is the call graph for this function:



6.3.4 Variable Documentation

6.3.4.1 uint8_t LED = 0x00 [static]

Definition at line 55 of file main.c.

6.3.4.2 volatile unsigned char* pIO_LED = (unsigned char *) IO_LED_AD [static]

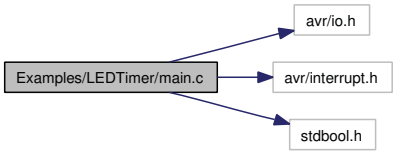
Definition at line 53 of file main.c.

6.4 Examples/LEDTimer/main.c File Reference

SensTermBoard Example: LED blinks with timer interrupt.

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

Include dependency graph for main.c:



Defines

- #define **IO_LED_AD** (0x4000)

Functions

- static void **led_init** (void)

  Initializes the led hardware.

- static void **led_set** (int led_nr, bool OnOff)

  Set LED nr's state to on or off.

- void **timer_init** (void)

  Init timer 4 for 1 millisecond interrupting.

- **ISR** (TIMER4_COMPB_vect)

  timer 4 capture and compare interrupt routine

- int **main** (void)

  Main function of the demo application.

Variables

- static volatile unsigned char * **pIO_LED** = (unsigned char *) IO_LED_AD
- static uint8_t **LED** = 0x00

6.4.1 Detailed Description

SensTermBoard Example: LED blinks with timer interrupt.

Blinks the LEDs with the assistance of a timer interrupt

Author:

Definition in file **main.c**.

6.4.2 Define Documentation

6.4.2.1 #define IO_LED_AD (0x4000)

Definition at line 47 of file main.c.

6.4.3 Function Documentation

6.4.3.1 ISR (TIMER4_COMPB_vect)

timer 4 capture and compare interrupt routine

timer for interrupts every 1 millisecond with a compare event. The compare counter gets resetet and the LED inverted.

Definition at line 115 of file main.c.

References led_set().

```
116 {
117   static int ms = 0;                              // memory for milliseconds
118   static bool led_state = true;                   // memory for led state
119
120   TCNT4 = 0;                                      // clear timer
121   if (500 <= ++ms)                                // 500 milliseconds over?
122   {
123     ms = 0;
124     led_set(0,led_state);
125     led_state = !led_state;                       // invert led state
126   }
127 }
```

Here is the call graph for this function:

6.4.3.2 static void led_init (void) [static]

Initializes the led hardware.

Initializes the led hardware so that further calls to led_set works

Definition at line 64 of file main.c.

```
65 {
66   XMCRA |= _BV(SRE); XMCRB = _BV(XMBK);            // enable external Memory Interface (IO, LED)
67 }
```

6.4.3.3 static void led_set (int *led_nr*, bool *OnOff*) [static]

Set LED nr's state to on or off.

set LED numer 1 or 2's state to On or Off

Parameters:

led_nr 0 or 1

OnOff true or false

Definition at line 79 of file main.c.

References LED, and pIO_LED.

```
80 {
81   if (OnOff)                                    // when On
82     LED &= ~_BV(led_nr);                        // reset Bit
83   else                                          // otherwise
84     LED |= _BV(led_nr);                         // set it
85
86   *pIO_LED = ~LED | ~0x03;                      // Memory mapped IO
87 }
```

6.4.3.4 int main (void)

Main function of the demo application.

This function consists of

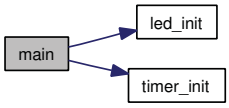
- The Initialicing Part (Sets all parts to default states)
- The Main-Loop (Endless loop with calls to all modules)

Definition at line 139 of file main.c.

References led_init(), and timer_init().

```
140 {
141   led_init();                                   // init LED's hardware
142   timer_init();                                 // init timer 4 for interrupting
143   sei();                                        // enable interrupts
144
145   while(1)                                      // endless main loop does nothing
146   {                                             // work is done in timer interrupt
147   }
148 }
```

Here is the call graph for this function:



6.4.3.5 void timer_init (void)

Init timer 4 for 1 millisecond interrupting.

Definition at line 96 of file main.c.

```
97 {
98
99   TCCR4B = _BV(CS42);                              // set clock source to prescaler F_CPU/256
100   OCR4B = (F_CPU / 256) / 1000;                    // timer interrupt every 1 millisecond
101   TCNT4 = 0;                                       // clear timer
102   TIMSK4 |= (1 << OCIE4B);                         // Enable timer interrupt
103   TIFR4  |= (1<<OCF4B);                            // clear capture flag
104 }
```

6.4.4 Variable Documentation

6.4.4.1 uint8_t LED = 0x00 [static]

Definition at line 55 of file main.c.

6.4.4.2 volatile unsigned char* pIO_LED = (unsigned char *) IO_LED_AD [static]

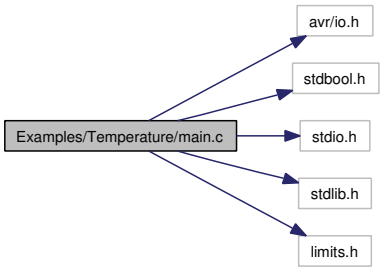
Definition at line 53 of file main.c.

6.5 Examples/Temperature/main.c File Reference

SensTermBoard Example: thermometer function over USB.

```
#include <avr/io.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
```

Include dependency graph for main.c:



Defines

- #define **IO_THERM_AD** (0x4000)
- #define **USB_FIFO_AD** (0x2200)
- #define **USB_BIT_RXF** _BV(7)
- #define **USB_BIT_TXE** _BV(6)
- #define **USB_STATE** (PINE)
- #define **VT100_CLR_LINE** "\x1B[2K\r"

Functions

- static int **usb_putc** (char c, struct __file *dummy_file)

  sends a character over the USB connection

- static int **usb_getc** (struct __file *dummy_file)

  Waits till there is a character available on the USB chip and gives it back.

- static void **usb_init** (void)

  Initialize interface to USB chip.

- static bool **usb_keypressed** (void)

  returns true if the USB chip has a character available for read

- uint16_t **pwr_read_adc** (uint8_t mux)

  gibt am ADC-Eingang mux gemessene Spannung in mV zurück

- static void **temp_init** (void)

  Initialize temperature hardware (ADC, Thermistor enable).

- static double **temp_get_degrcelc** (void)

  Gives back the actual temperature.

- int **main** (void)

  Main function of the demo application.

Variables

- FILE **usb_stream** = FDEV_SETUP_STREAM(usb_putc, usb_getc,_FDEV_SETUP_RW)
- static volatile unsigned char * **pUSB_Fifo** = (unsigned char *) USB_FIFO_AD
- static volatile unsigned char * **pIO_THERM** = (unsigned char *) IO_THERM_AD

6.5.1 Detailed Description

SensTermBoard Example: thermometer function over USB.

This demo inits the interface to the USB Chip FT245 an continuously writes the actual measured temperature to it.

Open Hyperterminal and connect to the virtual COM-Port that the SensTermBoard has got from Windows (Baudrate 115kBit, 8 data bits, no parity, 1 stop bit)

Note

- Input and Output is done by mechanism of the "stdio.h" interface. One need to setup a FILE stream with links to the putchar and getchar functions.
- the application uses floating point arithmetics. AVR-Studio has to be told to include the float version of the printf library:

  Author:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Definition in file **main.c**.

6.5.2 Define Documentation

6.5.2.1 #define IO_THERM_AD (0x4000)

Definition at line 61 of file main.c.

6.5.2.2 #define USB_BIT_RXF _BV(7)

Definition at line 63 of file main.c.

Referenced by usb_init(), and usb_keypressed().

6.5.2.3 #define USB_BIT_TXE _BV(6)

Definition at line 64 of file main.c.

Referenced by usb_init(), and usb_putc().

6.5.2.4 #define USB_FIFO_AD (0x2200)

Definition at line 62 of file main.c.

6.5.2.5 #define USB_STATE (PINE)

Definition at line 65 of file main.c.

Referenced by usb_keypressed(), and usb_putc().

6.5.2.6 #define VT100_CLR_LINE "\x1B[2K\r"

Definition at line 68 of file main.c.

Referenced by main().

6.5.3 Function Documentation

6.5.3.1 int main (void)

Main function of the demo application.

This function consists of

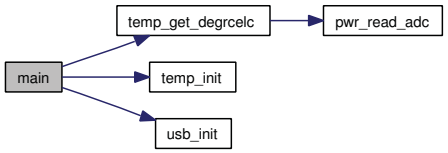
- The Initialicing Part (Sets all parts to default states)
- The Main-Loop (Endless loop with calls to all modules)

Definition at line 286 of file main.c.

References temp_get_degrcelc(), temp_init(), usb_init(), usb_stream, and VT100_CLR_LINE.

```
287 {
288   double last_temp;                                 // memory for last measurement
289   uint32_t cnt = 0;                                // counter for terminal update
290
291   temp_init();                                     // init temperature hardware
292   usb_init();                                      // init USB's hardware
293   stdout = &usb_stream;                            // init standard output over USB
294   stdin  = &usb_stream;                            // init standard input over USB
295
296   while(1)                                         // endless main loop
297   {
298     if (0 == cnt--)
299     {
300       cnt = 10000;                                 // reset counter for terminal update
301
302       last_temp = temp_get_degrcelc();
303       printf(VT100_CLR_LINE);                      // VT100-Terminal Sequence: clear line
304       printf("Actual temperature is: %1.1f",       // print temperature value
305         last_temp
306       );
307     }
308   }
309 }
```

Here is the call graph for this function:



6.5.3.2 uint16_t pwr_read_adc (uint8_t *mux*)

gibt am ADC-Eingang mux gemessene Spannung in mV zurück

Kanal 0: Batteriespannung, 1: Netzspannung

Parameters:

mux 0..1: Kanal

Returns:

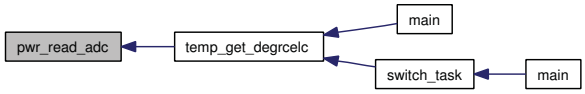
uint16_t Spannung in mV

Definition at line 160 of file main.c.

Referenced by temp_get_degrcelc().

```
161 {
162   uint32_t result;
163 
164   ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);    // Frequenzvorteiler
165                                 // setzen auf 8 (1) und ADC aktivieren (1)
166 
167   ADMUX = mux;                        // Kanal waehlen
168   ADMUX |= (1<<REFS0); // Referenz ist 3,3V VCC
169 
170   /* nach Aktivieren des ADC wird ein "Dummy-Readout" empfohlen, man liest
171      also einen Wert und verwirft diesen, um den ADC "warmlaufen zu lassen" */
172   ADCSRA |= (1<<ADSC);                // eine ADC-Wandlung
173   while ( ADCSRA & (1<<ADSC) ) {
174     ;     // auf Abschluss der Konvertierung warten
175   }
176   result  = ADC;
177   result  = (3223UL*result);   // 3300mV bei 1024 Schritten sind 3223mV je Schritt
178   result /= 1000;
179 
180   return (uint16_t)result;
181 }
```

Here is the caller graph for this function:



6.5.3.3 static double temp_get_degrcelc (void) [static]

Gives back the actual temperature.

uses a table with precalculated voltage measures and does linear interpolation between two sampling points.

Returns:

double The measured temperature in degrees celsius

Definition at line 223 of file main.c.

References pwr_read_adc().

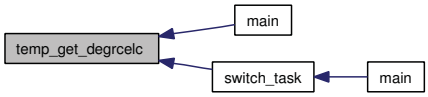
Referenced by main(), and switch_task().

```
224 {
225   static struct _ttable                          // table with precalculated volatages
226   {
227     int temp; int millivolts;
228   } ttable[] =
229   {
230     { -40 ,  86   } ,
231     { -35 ,  120  } ,
232     { -30 ,  164  } ,
233     { -25 ,  221  } ,
234     { -20 ,  292  } ,
235     { -15 ,  380  } ,
236     { -10 ,  487  } ,
237     { -5  ,  612  } ,
238     { 0   ,  756  } ,
239     { 5   ,  916  } ,
240     { 10  ,  1091 } ,
241     { 15  ,  1274 } ,
242     { 20  ,  1462 } ,
243     { 25  ,  1650 } ,
244     { 30  ,  1832 } ,
245     { 35  ,  2006 } ,
246     { 40  ,  2166 } ,
247     { 45  ,  2313 } ,
248     { 50  ,  2445 } ,
249     { 55  ,  2562 } ,
250     { 60  ,  2665 } ,
251     { 65  ,  2754 } ,
252     { 70  ,  2830 } ,
253     { 75  ,  2897 } ,
254   };
255
256   static double temp;
257   uint16_t i, volt;
258
259   volt = pwr_read_adc(3);                        // read ADC in millivolts
260
261   for (i = 1; i < (sizeof(ttable)/sizeof(ttable[0])-1); i++)
262   {
263     if (volt < ttable[i].millivolts)             // look for a fitting table entry
264     {
265       i--;
266       temp = (double)ttable[i].temp              // linear Interpolation
```

```
267               + 5.0 * (double) (volt - ttable[i].millivolts)
268                     / (double)(ttable[i+1].millivolts - ttable[i].millivolts);
269       break;
270     }
271   }
272
273   return temp;                                        // give back the temperature
274 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



6.5.3.4 static void temp_init (void) [static]

Initialize temperature hardware (ADC, Thermistor enable).

As reference the 3,3V system power is used. So you have a more accurate reference than the internal of the AVR. The full range of the ADC can be used.

Definition at line 192 of file main.c.

References pIO_THERM.

Referenced by main().

```
193 {
194   PRR0 &= ~(1 << PRADC);                              // power up ADC
195
196   ADCSRA = (1 << ADPS2) | (1 << ADPS1)                // divider 128 on
197            | (1 << ADPS0);
198   ADCSRB = 0x00;                                      // not the free running mode
199
200   DIDR0 = _BV(1) | _BV(0);                            // disable digital input buffers
201
202   ACSR = (1 << ACD);                                  // disable analog comperator
203
204
205   XMCRA |= _BV(SRE);XMCRB = _BV(XMBK);                // enable external Memory Interface (XRAM)
206
207   *pIO_THERM = 0x03;                                  // Thermistor On, LEDs off
208
209 }
```

Here is the caller graph for this function:

6.5.3.5 static int usb_getc (struct __file * *dummy_file*) [static]

Waits till there is a character available on the USB chip and gives it back.

waits forever when no char comes in

Returns:

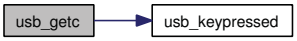
char This is the character on the USB chip

Definition at line 120 of file main.c.

References pUSB_Fifo, and usb_keypressed().

```
121 {
122   while (!usb_keypressed());                      // check for incomming data
123   return *pUSB_Fifo;                              // Return the data
124 }
```

Here is the call graph for this function:



6.5.3.6 static void usb_init (void) [static]

Initialize interface to USB chip.

Definition at line 91 of file main.c.

References USB_BIT_RXF, and USB_BIT_TXE.

Referenced by main().

```
92 {
93   XMCRA |= _BV(SRE);XMCRB = _BV(XMBK);             // enable external Memory Interface (XRAM)
94   DDRE  &= ~(USB_BIT_RXF | USB_BIT_TXE);           // USB's status signals are inputs
95   PORTE |=  (USB_BIT_RXF | USB_BIT_TXE);           // switch internal pull up resitors on
96 }
```

Here is the caller graph for this function:

6.5.3.7 static bool usb_keypressed (void) [static]

returns true if the USB chip has a character available for read

Returns:

bool false: No Char available, true: a char is available

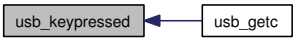
Definition at line 106 of file main.c.

References USB_BIT_RXF, and USB_STATE.

Referenced by usb_getc().

```
107 {
108   return (USB_STATE & USB_BIT_RXF) == 0;             // FIFO is not full? return true
109 }
```

Here is the caller graph for this function:



6.5.3.8 static int usb_putc (char *c*, struct __file * *dummy_file*) [static]

sends a character over the USB connection

waits while the USB chip anounces a full FIFO buffer

Parameters:

c the char to send

dummy_file not used

Definition at line 137 of file main.c.

References pUSB_Fifo, USB_BIT_TXE, and USB_STATE.

Referenced by usb_putc().

```
138 {
139   if ('\n' == c)                                  // automatic conversion from "\n" to "\r\n"
140   {
141     usb_putc('\r', dummy_file);
142   }
143
144   while (!((USB_STATE & USB_BIT_TXE) == 0));      // Wait for empty transmit buffer
145   *pUSB_Fifo = c;                                 // write the byte into the USB FIFO
146   return c;                                       // return the char
147 }
```

Here is the caller graph for this function:

6.5.4 Variable Documentation

6.5.4.1 volatile unsigned char* pIO_THERM = (unsigned char *) IO_THERM_AD [static]

Definition at line 82 of file main.c.

Referenced by temp_init().

6.5.4.2 volatile unsigned char* pUSB_Fifo = (unsigned char *) USB_FIFO_AD [static]

Definition at line 80 of file main.c.

Referenced by usb_getc(), and usb_putc().

6.5.4.3 FILE usb_stream = FDEV_SETUP_STREAM(usb_putc, usb_getc,_FDEV_SETUP_RW)

Definition at line 77 of file main.c.

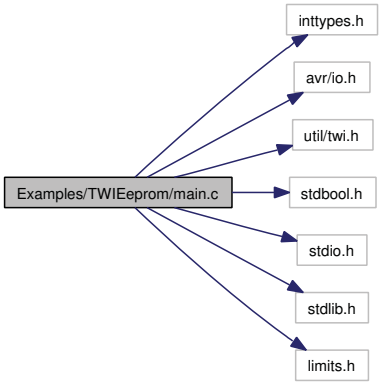
Referenced by main().

6.6 Examples/TWIEeprom/main.c File Reference

SensTermBoard Example: using of eeprom function over USB.

```
#include <inttypes.h>
#include <avr/io.h>
#include <util/twi.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
```

Include dependency graph for main.c:



Defines

- #define **USB_FIFO_AD** (0x2200)
- #define **USB_BIT_RXF** _BV(7)
- #define **USB_BIT_TXE** _BV(6)
- #define **USB_STATE** (PINE)
- #define **VT100_CLR_LINE** "\x1B[2K\r"
- #define **TWI_SLA_24CXX** 0xa0

  TWI address for 24Cxx EEPROM: 1 0 1 0 0 A1 A0 R/~W 24C128 (A0=L, A1=L, WP=L).

- #define **MAX_ITER** 200

  Maximal number of iterations to wait for a device to respond for a selection.

- #define **PAGE_SIZE** 64

  Number of bytes that can be written in a row, see comments for ***ee24xx_write_page()*** *(p. 52) below.*

- #define **EE_WRITE**(addr, str) ee24xx_write_bytes(addr, sizeof(str)-1, (uint8_t*)str)

Functions

- static int **usb_putc** (char c, struct __file *dummy_file)
  sends a character over the USB connection
- static int **usb_getc** (struct __file *dummy_file)
  Waits till there is a character available on the USB chip and gives it back.
- static void **usb_init** (void)
  Initialize interface to USB chip.
- static bool **usb_keypressed** (void)
  returns true if the USB chip has a character available for read
- void **ioinit** (void)
  Do all the startup-time peripheral initializations: TWI clock, .
- int **ee24xx_read_bytes** (uint16_t eeaddr, int len, uint8_t *buf)
  Read "len" bytes from EEPROM starting at "eeaddr" into "buf".
- int **ee24xx_write_page** (uint16_t eeaddr, int len, uint8_t *buf)
  Write "len" bytes into EEPROM starting at "eeaddr" from "buf".
- int **ee24xx_write_bytes** (uint16_t eeaddr, int len, uint8_t *buf)
  *Wrapper around **ee24xx_write_page()** (p. 52) that repeats calling this function until either an error has been returned, or all bytes have been written.*
- void **error** (void)
  Error Routine.
- int **main** (void)
  Main function of the demo application.

Variables

- FILE **usb_stream** = FDEV_SETUP_STREAM(usb_putc, usb_getc, _FDEV_SETUP_RW)
- static volatile unsigned char * **pUSB_Fifo** = (unsigned char *) USB_-FIFO_AD
- uint8_t **twst**
  Saved TWI status register, for error messages only.

6.6.1 Detailed Description

SensTermBoard Example: using of eeprom function over USB.

This demo inits the interface to the USB Chip FT245 and reads and writes values to and from EEPROM AT24C128 using the builtin TWI interface of an ATmega device.

Open Hyperterminal and connect to the virtual COM-Port that the SensTermBoard has got from Windows (Baudrate 115kBit, 8 data bits, no parity, 1 stop bit)

Note

- Input and Output is done by mechanism of the "stdio.h" interface. One need to setup a FILE stream with links to the putchar and getchar functions.

Author:

based on twitest.c of Joerg Wunsch <joerg@FreeBSD.ORG>

Definition in file **main.c**.

6.6.2 Define Documentation

6.6.2.1 #define EE_WRITE(addr, str) ee24xx_write_bytes(addr, sizeof(str)-1, (uint8_t*)str)

Referenced by main().

6.6.2.2 #define MAX_ITER 200

Maximal number of iterations to wait for a device to respond for a selection.

Should be large enough to allow for a pending write to complete, but low enough to properly abort an infinite loop in case a slave is broken or not present at all. With 100 kHz TWI clock, transfering the start condition and SLA+R/W packet takes about 10 μs. The longest write period is supposed to not exceed $\sim$ 10 ms. Thus, normal operation should not require more than 100 iterations to get the device to respond to a selection.

Definition at line 174 of file main.c.

Referenced by ee24xx_read_bytes(), and ee24xx_write_page().

6.6.2.3 #define PAGE_SIZE 64

Number of bytes that can be written in a row, see comments for **ee24xx_write_page()** (p. 52) below.

Some vendor's devices would accept 16, but 8 seems to be the lowest common denominator.

Note that the page size must be a power of two, this simplifies the page boundary calculations below.

Definition at line 187 of file main.c.

Referenced by ee24xx_write_page().

6.6.2.4 #define TWI_SLA_24CXX 0xa0

TWI address for 24Cxx EEPROM: 1 0 1 0 0 A1 A0 R/$\sim$W 24C128 (A0=L, A1=L, WP=L).

Definition at line 160 of file main.c.

Referenced by ee24xx_read_bytes(), and ee24xx_write_page().

6.6.2.5 #define USB_BIT_RXF _BV(7)

Definition at line 64 of file main.c.

6.6.2.6 #define USB_BIT_TXE _BV(6)

Definition at line 65 of file main.c.

6.6.2.7 #define USB_FIFO_AD (0x2200)

Definition at line 63 of file main.c.

6.6.2.8 #define USB_STATE (PINE)

Definition at line 66 of file main.c.

6.6.2.9 #define VT100_CLR_LINE "\x1B[2K\r"

Definition at line 70 of file main.c.

6.6.3 Function Documentation

6.6.3.1 int ee24xx_read_bytes (uint16_t *eeaddr*, int *len*, uint8_t * *buf*)

Read "len" bytes from EEPROM starting at "eeaddr" into "buf".

This requires two bus cycles: during the first cycle, the device will be selected (master transmitter mode), and the address transfered.

The second bus cycle will reselect the device (repeated start condition, going into master receiver mode), and transfer the data from the device to the TWI master. Multiple bytes can be transfered by ACKing the client's transfer. The last transfer will be NACKed, which the client will take as an indication to not initiate further transfers.

Definition at line 234 of file main.c.

References error(), MAX_ITER, TWI_SLA_24CXX, and twst.

Referenced by main().

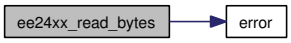
```
235 {
236   uint8_t sla, twcr, n = 0;
237   int rv = 0;
238 
239   sla = TWI_SLA_24CXX;
240   /*
241    * First cycle: master transmitter mode
242    */
243 restart:
244   if (n++ >= MAX_ITER)
245     return -1;
246 begin:
247 
248   TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN); /* send start condition */
```

```
249   while ((TWCR & _BV(TWINT)) == 0)
250     ; /* wait for transmission */
251   switch ((twst = TW_STATUS))
252   {
253     case TW_REP_START:    /* OK, but should not happen */
254     case TW_START:
255       break;
256
257     case TW_MT_ARB_LOST:
258       goto begin;
259
260     default:
261       return -1;  /* error: not in start condition */
262       /* NB: do /not/ send stop condition */
263   }
264
265   TWDR = sla | TW_WRITE;         /* send SLA+W */
266   TWCR = _BV(TWINT) | _BV(TWEN); /* clear interrupt to start transmission */
267   while ((TWCR & _BV(TWINT)) == 0)
268     ; /* wait for transmission */
269   switch ((twst = TW_STATUS))
270   {
271     case TW_MT_SLA_ACK:
272       break;
273
274     case TW_MT_SLA_NACK:   /* nack during select: device busy writing */
275       goto restart;
276
277     case TW_MT_ARB_LOST:   /* re-arbitrate */
278       goto begin;
279
280     default:
281       goto error;  /* must send stop condition */
282   }
283
284   TWDR = (eeaddr >> 8) & 0xFF; /* high 8 bits of addr */
285   TWCR = _BV(TWINT) | _BV(TWEN);  /* clear interrupt to start transmission */
286   while ((TWCR & _BV(TWINT)) == 0)
287     ; /* wait for transmission */
288   switch ((twst = TW_STATUS))
289   {
290     case TW_MT_DATA_ACK:
291       break;
292
293     case TW_MT_DATA_NACK:
294       goto quit;
295
296     case TW_MT_ARB_LOST:
297       goto begin;
298
299     default:
300       goto error;  /* must send stop condition */
301   }
302
303   TWDR = eeaddr & 0xFF;  /* low 8 bits of addr */
304   TWCR = _BV(TWINT) | _BV(TWEN); /* clear interrupt to start transmission */
305   while ((TWCR & _BV(TWINT)) == 0)
306     ; /* wait for transmission */
307   switch ((twst = TW_STATUS))
308   {
309     case TW_MT_DATA_ACK:
310       break;
```

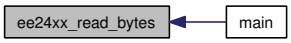
```
311
312     case TW_MT_DATA_NACK:
313       goto quit;
314
315     case TW_MT_ARB_LOST:
316       goto begin;
317
318     default:
319       goto error;  /* must send stop condition */
320   }
321
322   /*
323    * Next cycle(s): master receiver mode
324    */
325   TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN); /* send (rep.) start condition */
326   while ((TWCR & _BV(TWINT)) == 0)
327     ; /* wait for transmission */
328   switch ((twst = TW_STATUS))
329   {
330     case TW_START:    /* OK, but should not happen */
331     case TW_REP_START:
332       break;
333
334     case TW_MT_ARB_LOST:
335       goto begin;
336
337     default:
338       goto error;
339   }
340
341   /* send SLA+R */
342   TWDR = sla | TW_READ;
343   TWCR = _BV(TWINT) | _BV(TWEN); /* clear interrupt to start transmission */
344   while ((TWCR & _BV(TWINT)) == 0)
345     ; /* wait for transmission */
346   switch ((twst = TW_STATUS))
347   {
348     case TW_MR_SLA_ACK:
349       break;
350
351     case TW_MR_SLA_NACK:
352       goto quit;
353
354     case TW_MR_ARB_LOST:
355       goto begin;
356
357     default:
358       goto error;
359   }
360
361   for (twcr = _BV(TWINT) | _BV(TWEN) | _BV(TWEA);
362       len > 0;
363       len--)
364   {
365     if (len == 1)
366       twcr = _BV(TWINT) | _BV(TWEN); /* send NAK this time */
367     TWCR = twcr;  /* clear int to start transmission */
368     while ((TWCR & _BV(TWINT)) == 0)
369       ; /* wait for transmission */
370     switch ((twst = TW_STATUS))
371     {
372       case TW_MR_DATA_NACK:
```

```
373         len = 0;  /* force end of loop */
374         /* FALLTHROUGH */
375       case TW_MR_DATA_ACK:
376         *buf++ = TWDR;
377         rv++;
378         break;
379
380       default:
381         goto error;
382     }
383   }
384 quit:
385   TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); /* send stop condition */
386
387   return rv;
388
389 error:
390   rv = -1;
391   goto quit;
392 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.2 int ee24xx_write_bytes (uint16_t *eeaddr*, int *len*, uint8_t * *buf*)

Wrapper around **ee24xx_write_page()** (p. 52) that repeats calling this function until either an error has been returned, or all bytes have been written.

Definition at line 549 of file main.c.

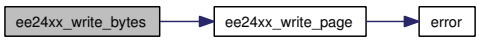
References ee24xx_write_page().

Referenced by main().

```
550 {
551   int rv, total;
552
553   total = 0;
554   do
555   {
556     printf("Calling ee24xx_write_page(%d, %d, %p)", eeaddr, len, buf);
557     rv = ee24xx_write_page(eeaddr, len, buf);
558     printf(" => %d\n", rv);
559     if (rv == -1)
560       return -1;
```

```
561     eeaddr += rv;
562     len -= rv;
563     buf += rv;
564     total += rv;
565   }
566   while (len > 0);
567
568   return total;
569 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.6.3.3 int ee24xx_write_page (uint16_t *eeaddr*, int *len*, uint8_t * *buf*)

Write "len" bytes into EEPROM starting at "eeaddr" from "buf".

This is a bit simpler than the previous function since both, the address and the data bytes will be transfered in master transmitter mode, thus no reselection of the device is necessary. However, the EEPROMs are only capable of writing one "page" simultaneously, so care must be taken to not cross a page boundary within one write cycle. The amount of data one page consists of varies from manufacturer to manufacturer: some vendors only use 8-byte pages for the smaller devices, and 16-byte pages for the larger devices, while other vendors generally use 16-byte pages. We thus use the smallest common denominator of 8 bytes per page, declared by the macro PAGE_SIZE above.

The function simply returns after writing one page, returning the actual number of data byte written. It is up to the caller to re-invoke it in order to write further data.

Definition at line 415 of file main.c.

References error(), MAX_ITER, PAGE_SIZE, TWI_SLA_24CXX, and twst.

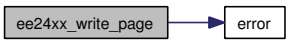
Referenced by ee24xx_write_bytes().

```
416 {
417   uint8_t sla, n = 0;
418   int rv = 0;
419   uint16_t endaddr;
420
421   if (eeaddr + len < (eeaddr | (PAGE_SIZE - 1)))
422   {
```

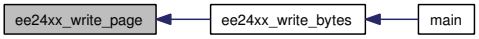
```
423     endaddr = eeaddr + len;
424   }
425   else
426   {
427     endaddr = (eeaddr | (PAGE_SIZE - 1)) + 1;
428   }
429   len = endaddr - eeaddr;
430 
431   sla = TWI_SLA_24CXX;
432 
433 restart:
434   if (n++ >= MAX_ITER)
435   {
436     return -1;
437   }
438 begin:
439 
440   TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN); /* send start condition */
441   while ((TWCR & _BV(TWINT)) == 0)
442     ; /* wait for transmission */
443   switch ((twst = TW_STATUS))
444   {
445     case TW_REP_START:    /* OK, but should not happen */
446     case TW_START:
447       break;
448 
449     case TW_MT_ARB_LOST:
450       goto begin;
451 
452     default:
453       return -1;  /* error: not in start condition */
454       /* NB: do /not/ send stop condition */
455   }
456 
457   TWDR = sla | TW_WRITE; /* send SLA+W */
458   TWCR = _BV(TWINT) | _BV(TWEN); /* clear interrupt to start transmission */
459   while ((TWCR & _BV(TWINT)) == 0)
460     ; /* wait for transmission */
461   switch ((twst = TW_STATUS))
462   {
463     case TW_MT_SLA_ACK:
464       break;
465 
466     case TW_MT_SLA_NACK:   /* nack during select: device busy writing */
467       goto restart;
468 
469     case TW_MT_ARB_LOST:   /* re-arbitrate */
470       goto begin;
471 
472     default:
473       goto error;  /* must send stop condition */
474   }
475 
476   TWDR = (eeaddr >> 8) & 0xFF; /* high 8 bits of addr */
477   TWCR = _BV(TWINT) | _BV(TWEN); /* clear interrupt to start transmission */
478   while ((TWCR & _BV(TWINT)) == 0)
479     ; /* wait for transmission */
480   switch ((twst = TW_STATUS))
481   {
482     case TW_MT_DATA_ACK:
483       break;
484 
```

```
485     case TW_MT_DATA_NACK:
486       goto quit;
487 
488     case TW_MT_ARB_LOST:
489       goto begin;
490 
491     default:
492       goto error;  /* must send stop condition */
493   }
494 
495   TWDR = eeaddr & 0xFF; /* low 8 bits of addr */
496   TWCR = _BV(TWINT) | _BV(TWEN); /* clear interrupt to start transmission */
497   while ((TWCR & _BV(TWINT)) == 0)
498     ; /* wait for transmission */
499   switch ((twst = TW_STATUS))
500   {
501     case TW_MT_DATA_ACK:
502       break;
503 
504     case TW_MT_DATA_NACK:
505       goto quit;
506 
507     case TW_MT_ARB_LOST:
508       goto begin;
509 
510     default:
511       goto error;  /* must send stop condition */
512   }
513 
514   for (; len > 0; len--)
515   {
516     TWDR = *buf++;
517     TWCR = _BV(TWINT) | _BV(TWEN); /* start transmission */
518     while ((TWCR & _BV(TWINT)) == 0)
519       ; /* wait for transmission */
520     switch ((twst = TW_STATUS))
521     {
522       case TW_MT_DATA_NACK:
523         goto error;  /* device write protected -- Note [16] */
524 
525       case TW_MT_DATA_ACK:
526         rv++;
527         break;
528 
529       default:
530         goto error;
531     }
532   }
533 quit:
534   TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); /* send stop condition */
535   return rv;
536 
537 error:
538   rv = -1;
539   goto quit;
540 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.4 void error (void)

Error Routine.

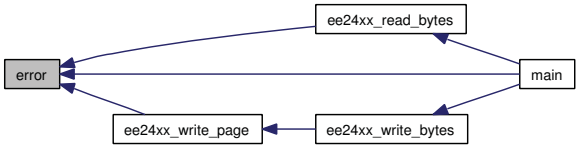
Definition at line 576 of file main.c.

References twst.

Referenced by ee24xx_read_bytes(), ee24xx_write_page(), and main().

```
577 {
578   printf("error: TWI status %#x\n", twst);
579   exit(0);
580 }
```

Here is the caller graph for this function:



6.6.3.5 void ioinit (void)

Do all the startup-time peripheral initializations: TWI clock, .

..

Definition at line 203 of file main.c.

Referenced by main().

```
204 {
205   /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */
206   /* has prescaler (mega128 & newer) */
207   TWSR = 0;
208
209 #if F_CPU < 3600000UL
210
211   TWBR = 10;   /* smallest TWBR value, see note [5] */
212 #else
213
214   TWBR = (F_CPU / 100000UL - 16) / 2;
215 #endif
216 }
```

Here is the caller graph for this function:

6.6.3.6 int main (void)

Main function of the demo application.

This function consists of

- The Initialising Part (Sets all parts to default states)
- The Main-Sequence with read and write EEPROM
- The Endless-Loop to stop the mcu

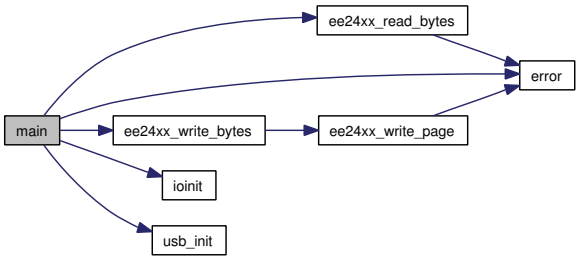
Definition at line 592 of file main.c.

References ee24xx_read_bytes(), ee24xx_write_bytes(), EE_WRITE, error(), ioinit(), usb_init(), and usb_stream.

```
593 {
594   uint16_t a;
595   int rv;
596   uint8_t b[16];
597   uint8_t x;
598
599   ioinit();
600
601   usb_init();                                    // init USB's hardware
602   stdout = &usb_stream;                          // init standard output over USB
603   stdin = &usb_stream;                          // init standard input over USB
604
605   for (a = 0; a < sizeof(b); a++)
606     b[a] = 0xFF;
607   for (a = 0; a < 256;)
608   {
609     rv = ee24xx_write_bytes(a, sizeof(b), (uint8_t*)b);
610     if (rv <= 0)
611     {
612       error();
613     }
614     a += rv;
615   }
616
617   for (a = 0; a < 256;)
618   {
619     printf("%#04x: ", a);
620     rv = ee24xx_read_bytes(a, 16, b);
621     if (rv <= 0)
622     {
623       error();
624     }
625     if (rv < 16)
626     {
```

```
627       printf("warning: short read %d\n", rv);
628     }
629     a += rv;
630     for (x = 0; x < rv; x++)
631     {
632       printf("%02x ", b[x]);
633     }
634     putchar('\n');
635   }
636 #define EE_WRITE(addr, str) ee24xx_write_bytes(addr, sizeof(str)-1, (uint8_t*)str)
637   rv = EE_WRITE(55, "The quick brown fox jumps over the lazy dog.");
638   if (rv < 0)
639   {
640     error();
641   }
642   printf("Wrote %d bytes.\n", rv);
643   for (a = 0; a < 256;)
644   {
645     printf("%#04x: ", a);
646     rv = ee24xx_read_bytes(a, 16, b);
647     if (rv <= 0)
648     {
649       error();
650     }
651     if (rv < 16)
652     {
653       printf("warning: short read %d\n", rv);
654     }
655     a += rv;
656     for (x = 0; x < rv; x++)
657     {
658       printf("%02x ", b[x]);
659     }
660     putchar('\n');
661   }
662   printf("done.\n");
663
664   while (1) ;
665 }
```

Here is the call graph for this function:



6.6.3.7 static int usb_getc (struct __file * *dummy_file*) [static]

Waits till there is a character available on the USB chip and gives it back.

waits forever when no char comes in

Returns:

char This is the character on the USB chip

Definition at line 122 of file main.c.

References pUSB_Fifo, and usb_keypressed().

```
123 {
124   while (!usb_keypressed())
125     ;                       // check for incomming data
126   return *pUSB_Fifo;                               // Return the data
127 }
```

Here is the call graph for this function:

6.6.3.8 static void usb_init (void) [static]

Initialize interface to USB chip.

Definition at line 93 of file main.c.

References USB_BIT_RXF, and USB_BIT_TXE.

```
94 {
95   XMCRA |= _BV(SRE);
96   XMCRB = _BV(XMBK);             // enable external Memory Interface (XRAM)
97   DDRE &= ~(USB_BIT_RXF | USB_BIT_TXE);           // USB's status signals are inputs
98   PORTE |= (USB_BIT_RXF | USB_BIT_TXE);           // switch internal pull up resitors on
99 }
```

6.6.3.9 static bool usb_keypressed (void) [static]

returns true if the USB chip has a character available for read

Returns:

bool false: No Char available, true: a char is available

Definition at line 108 of file main.c.

References USB_BIT_RXF, and USB_STATE.

```
109 {
110   return (USB_STATE & USB_BIT_RXF) == 0;             // FIFO is not full? return true
111 }
```

6.6.3.10 static int usb_putc (char *c*, struct __file * *dummy_file*) [static]

sends a character over the USB connection

waits while the USB chip anounces a full FIFO buffer

Parameters:

c the char to send

dummy_file not used

Definition at line 139 of file main.c.

References pUSB_Fifo, USB_BIT_TXE, usb_putc(), and USB_STATE.

```
140 {
141   if ('\n' == c)                                  // automatic conversion from "\n" to "\r\n"
142   {
143     usb_putc('\r', dummy_file);
144   }
145
146   while (!((USB_STATE & USB_BIT_TXE) == 0))
147     ;       // Wait for empty transmit buffer
148   *pUSB_Fifo = c;                                 // write the byte into the USB FIFO
149   return c;                                       // return the char
150 }
```

Here is the call graph for this function:

6.6.4 Variable Documentation

6.6.4.1 volatile unsigned char* pUSB_Fifo = (unsigned char *) USB_FIFO_AD [static]

Definition at line 84 of file main.c.

6.6.4.2 uint8_t twst

Saved TWI status register, for error messages only.

We need to save it in a variable, since the datasheet only guarantees the TWSR register to have valid contents while the TWINT bit in TWCR is set.

Definition at line 196 of file main.c.

Referenced by ee24xx_read_bytes(), ee24xx_write_page(), and error().

6.6.4.3 FILE usb_stream = FDEV_SETUP_STREAM(usb_putc, usb_getc, _FDEV_SETUP_RW)

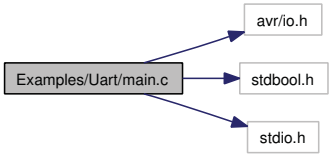
Definition at line 82 of file main.c.

6.7 Examples/Uart/main.c File Reference

SensTermBoard Example: "Hello World" over UART 0.

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

Include dependency graph for main.c:



Defines

- #define **BAUDRATE**(BR) (((unsigned long)F_CPU/((unsigned long)16*(unsigned long)BR))-1)

Functions

- static int **uart_putc** (char c, struct __file *dummy_file)

  sends a character over the UART connection

- static int **uart_getc** (struct __file *dummy_file)

  Waits till there is a character available on the UART and gives it back.

- static void **uart_init** (unsigned char BaudRate)

  Initialize interface to UART.

- static bool **uart_keypressed** (void)

  returns true if the UART has a character available for read

- int **main** (void)

  Main function of the demo application.

Variables

- FILE **uart_stream** = FDEV_SETUP_STREAM(uart_putc, uart_getc,_FDEV_SETUP_RW)

6.7.1 Detailed Description

SensTermBoard Example: "Hello World" over UART 0.

This demo inits the interface to the UART an writes the message "Hello World" to it. Any key press results in a further "Hello World".

Open Hyperterminal and open the COM-Port which the SensTermBoard is connected to (Baudrate 9600Baud, 8 data bits, no parity, 1 stop bit)

Note:

- You need a cable with a RS232 level converter from the 3,3VDC logic level of the board to the +/-12VDC of the RS232 connector. There are a lot of cheap cables for mobile phones available on the market, one can find them at google's product search: `http://www.google.com/products?q=-usb+siemens++Data+Cable&btnG=Search&hl=en&show=dd`
- Input and Output is done by mechanism of the "stdio.h" interface. One need to setup a FILE stream with links to the putchar and getchar functions.

Author:

THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file **main.c**.

6.7.2 Define Documentation

6.7.2.1 #define BAUDRATE(BR) (((unsigned long)F_CPU/((unsigned long)16*(unsigned long)BR))-1)

Definition at line 60 of file main.c.

Referenced by main().

6.7.3 Function Documentation

6.7.3.1 int main (void)

Main function of the demo application.

This function consists of

- The Initialicing Part (Sets all parts to default states)
- The Main-Loop (Endless loop with calls to all modules)

Definition at line 139 of file main.c.

References BAUDRATE, uart_init(), and uart_stream.

```
140 {
141   uart_init(BAUDRATE(9600));                    // init UARTS's hardware
142   stdout = &uart_stream;                        // init standard output over UART
143   stdin  = &uart_stream;                        // init standard input over UART
144
145   while(1)                                      // endless main loop
146   {
147     printf("Hello World\r\n");                  // print the message
148     getchar();                                  // wait for a key press
149   }
150 }
```

Here is the call graph for this function:

6.7.3.2 static int uart_getc (struct __file * *dummy_file*) [static]

Waits till there is a character available on the UART and gives it back.

waits forever when no char comes in

Returns:

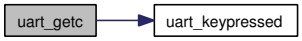
char This is the character on the UART

Definition at line 106 of file main.c.

References uart_keypressed().

```
107 {
108   while (!uart_keypressed());                    // check for incomming data
109   return UDR0;                                   // Return the data
110 }
```

Here is the call graph for this function:



6.7.3.3 static void uart_init (unsigned char *BaudRate*) [static]

Initialize interface to UART.

Definition at line 78 of file main.c.

Referenced by main().

```
79 {
80   UBRR0L = BaudRate;                             // set the baudrate that is choosen
81   UCSR0B = (1<<RXEN0) | (1<<TXEN0);              // enable Receiver and Transmitter
82   UCSR0C = (3<<UCSZ00);                          // 8 Data, No Parity, 1 Stop bit
83 }
```

Here is the caller graph for this function:

6.7.3.4 static bool uart_keypressed (void) [static]

returns true if the UART has a character available for read

Returns:

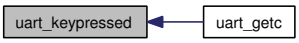
bool false: No Char available, true: a char is available

Definition at line 92 of file main.c.

Referenced by uart_getc().

```
93 {
94   return (UCSR0A & _BV(RXC0)) != 0;               // FIFO is not full? return true
95 }
```

Here is the caller graph for this function:



6.7.3.5 static int uart_putc (char *c*, struct __file * *dummy_file*) [static]

sends a character over the UART connection

waits while the UART anounces a full FIFO buffer

Parameters:

c the char to send

dummy_file not used

Definition at line 122 of file main.c.

```
123 {
124   while ( !(UCSR0A & _BV(UDRE0)) );                // wait for empty tx buffer
125   UDR0 = c;                                        // Start transmittion
126   return c;                                        // return the char
127 }
```

6.7.4 Variable Documentation

6.7.4.1 FILE uart_stream = FDEV_SETUP_STREAM(uart_putc, uart_getc, _FDEV_SETUP_RW)

Definition at line 69 of file main.c.

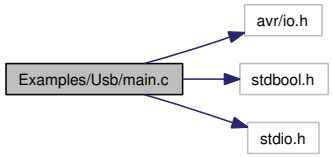
Referenced by main().

6.8 Examples/Usb/main.c File Reference

SensTermBoard Example: "Hello World" over USB chip FT245.

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

Include dependency graph for main.c:



Defines

- #define **USB_FIFO_AD** (0x2200)
- #define **USB_BIT_RXF** _BV(7)
- #define **USB_BIT_TXE** _BV(6)
- #define **USB_STATE** (PINE)

Functions

- static int **usb_putc** (char c, struct __file *dummy_file)

  sends a character over the USB connection

- static int **usb_getc** (struct __file *dummy_file)

  Waits till there is a character available on the USB chip and gives it back.

- static void **usb_init** (void)

  Initialize interface to USB chip.

- static bool **usb_keypressed** (void)

  returns true if the USB chip has a character available for read

- int **main** (void)

  Main function of the demo application.

Variables

- FILE **usb_stream** = FDEV_SETUP_STREAM(usb_putc, usb_getc,_FDEV_SETUP_RW)

- static volatile unsigned char * **pUSB_Fifo** = (unsigned char *) USB_FIFO_AD

6.8.1 Detailed Description

SensTermBoard Example: "Hello World" over USB chip FT245.

This demo inits the interface to the USB Chip FT245 an writes the message "Hello World:X" to it. Any key press results in a further "Hello World:X". The X is replaced by the pressed key.

Open Hyperterminal and connect to the virtual COM-Port that the SensTermBoard has got from Windows (Baudrate 115kBit, 8 data bits, no parity, 1 stop bit)

Note: Input and Output is done by mechanism of the "stdio.h" interface. One need to setup a FILE stream with links to the putchar and getchar functions.

Author:

Definition in file **main.c**.

6.8.2 Define Documentation

6.8.2.1 #define USB_BIT_RXF _BV(7)

Definition at line 56 of file main.c.

6.8.2.2 #define USB_BIT_TXE _BV(6)

Definition at line 57 of file main.c.

6.8.2.3 #define USB_FIFO_AD (0x2200)

Definition at line 55 of file main.c.

6.8.2.4 #define USB_STATE (PINE)

Definition at line 58 of file main.c.

6.8.3 Function Documentation

6.8.3.1 int main (void)

Main function of the demo application.

This function consists of

- The Initialicing Part (Sets all parts to default states)
- The Main-Loop (Endless loop with calls to all modules)

Definition at line 145 of file main.c.

References usb_init(), and usb_stream.

```
146 {
147   char c = ' ';
148
149   usb_init();                                   // init USB's hardware
150   stdout = &usb_stream;                         // init standard output over USB
151   stdin  = &usb_stream;                         // init standard input over USB
152
153   while(1)                                      // endless main loop
154   {
155     printf("Hello World:%c\r\n",c);             // print the message
156     c = getchar();                              // wait for a key press
157   }
158 }
```

Here is the call graph for this function:

6.8.3.2 static int usb_getc (struct __file * *dummy_file*) [static]

Waits till there is a character available on the USB chip and gives it back.

waits forever when no char comes in

Returns:

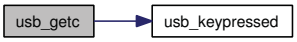
char This is the character on the USB chip

Definition at line 110 of file main.c.

References pUSB_Fifo, and usb_keypressed().

```
111 {
112   while (!usb_keypressed());                      // check for incomming data
113   return *pUSB_Fifo;                              // Return the data
114 }
```

Here is the call graph for this function:



6.8.3.3 static void usb_init (void) [static]

Initialize interface to USB chip.

Definition at line 81 of file main.c.

References USB_BIT_RXF, and USB_BIT_TXE.

```
82 {
83   XMCRA |= _BV(SRE);XMCRB = _BV(XMBK);            // enable external Memory Interface (XRAM)
84   DDRE  &= ~(USB_BIT_RXF | USB_BIT_TXE);          // USB's status signals are inputs
85   PORTE |=  (USB_BIT_RXF | USB_BIT_TXE);          // switch internal pull up resitors on
86 }
```

6.8.3.4 static bool usb_keypressed (void) [static]

returns true if the USB chip has a character available for read

Returns:

bool false: No Char available, true: a char is available

Definition at line 96 of file main.c.

References USB_BIT_RXF, and USB_STATE.

```
97 {
98   return (USB_STATE & USB_BIT_RXF) == 0;             // FIFO is not full? return true
99 }
```

6.8.3.5 static int usb_putc (char *c*, struct __file * *dummy_file*) [static]

sends a character over the USB connection

waits while the USB chip anounces a full FIFO buffer

Parameters:

c the char to send

dummy_file not used

Definition at line 127 of file main.c.

References pUSB_Fifo, USB_BIT_TXE, and USB_STATE.

```
128 {
129   while (!((USB_STATE & USB_BIT_TXE) == 0));       // Wait for empty transmit buffer
130   *pUSB_Fifo = c;                                  // write the byte into the USB FIFO
131   return c;                                        // return the char
132 }
```

6.8.4 Variable Documentation

6.8.4.1 volatile unsigned char* pUSB_Fifo = (unsigned char *) USB_FIFO_AD [static]

Definition at line 72 of file main.c.

6.8.4.2 FILE usb_stream = FDEV_SETUP_STREAM(usb_putc, usb_getc,_FDEV_SETUP_RW)

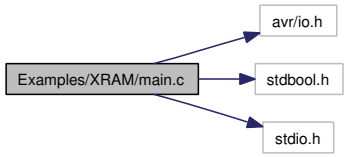
Definition at line 69 of file main.c.

6.9 Examples/XRAM/main.c File Reference

SensTermBoard Example: How to use the external 32k (XRAM).

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

Include dependency graph for main.c:



Defines

- #define **USB_FIFO_AD** (0x2200)
- #define **XRAM_MEM_AD** (0x8000)
- #define **USB_BIT_RXF** _BV(7)
- #define **USB_BIT_TXE** _BV(6)
- #define **USB_STATE** (PINE)

Functions

- static int **usb_putc** (char c, struct __file *dummy_file)

  sends a character over the USB connection

- static int **usb_getc** (struct __file *dummy_file)

  Waits till there is a character available on the USB chip and gives it back.

- static void **usb_init** (void)

  Initialize interface to USB chip.

- static bool **usb_keypressed** (void)

  returns true if the USB chip has a character available for read

- int **main** (void)

  Main function of the demo application.

Variables

- FILE **usb_stream** = FDEV_SETUP_STREAM(usb_putc, usb_getc,_FDEV_SETUP_RW)
- static volatile unsigned char * **pUSB_Fifo** = (unsigned char *) USB_FIFO_AD
- static volatile unsigned char * **pXRAM** = (unsigned char *) XRAM_MEM_AD

6.9.1 Detailed Description

SensTermBoard Example: How to use the external 32k (XRAM).

This demo inits the XRAM interface to the external static memory (32K) and uses pointer based access to the memory to write and compare a test pattern.

Note:

: Because the lower adress range is occupied by the AVR's internal static memory the external RAM is mapped onto the upper 32K (0x8000..0xFFFF) You have to tell your linker to map external memory to this range if your want to use it for additional heap or stack space: -Wl,–defsym=__heap_start=0x802000,–defsym=__heap_end=0x803fff A good description can be found at the avr-libc project: `http://www.nongnu.org/avr-libc/user-manual/malloc.html`

Author:

DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file **main.c**.

6.9.2 Define Documentation

6.9.2.1 #define USB_BIT_RXF _BV(7)

Definition at line 57 of file main.c.

6.9.2.2 #define USB_BIT_TXE _BV(6)

Definition at line 58 of file main.c.

6.9.2.3 #define USB_FIFO_AD (0x2200)

Definition at line 55 of file main.c.

6.9.2.4 #define USB_STATE (PINE)

Definition at line 59 of file main.c.

6.9.2.5 #define XRAM_MEM_AD (0x8000)

Definition at line 56 of file main.c.

6.9.3 Function Documentation

6.9.3.1 int main (void)

Main function of the demo application.

This function consists of

- The Initialicing Part (Sets all parts to default states)
- The Main-Loop (Endless loop with calls to all modules)

Definition at line 147 of file main.c.

References pXRAM, usb_init(), and usb_stream.

```
148 {
149   uint16_t adr;
150 
151   usb_init();                                     // init USB's hardware
152   stdout = &usb_stream;                           // init standard output over USB
153   stdin  = &usb_stream;                           // init standard input over USB
154 
155   printf("Hello XRAM, what about a test?\r\n");   // print the message
156 
157   for (adr = 0x0000; adr < 0x8000; adr++)         // write test sample to XRAM
158   {
159     pXRAM[adr] = (uint8_t)adr;
160   }
161 
162   for (adr = 0x0000; adr < 0x8000; adr++)         // compare test samples in XRAM
163   {
164     if (pXRAM[adr] != (uint8_t)adr)
165     {
166       printf("Test failed at address 0x%04X!",adr);
167       return 1;
168     }
169   }
170 
171   printf("Test passed successfully!\r\n");        // if we come here, the test passed successfully
172   return 0;
173 }
```

Here is the call graph for this function:

6.9.3.2 static int usb_getc (struct __file * *dummy_file*) [static]

Waits till there is a character available on the USB chip and gives it back.

waits forever when no char comes in

Returns:

char This is the character on the USB chip

Definition at line 112 of file main.c.

References pUSB_Fifo, and usb_keypressed().

```
113 {
114   while (!usb_keypressed());                      // check for incomming data
115   return *pUSB_Fifo;                              // Return the data
116 }
```

Here is the call graph for this function:

6.9.3.3 static void usb_init (void) [static]

Initialize interface to USB chip.

Definition at line 83 of file main.c.

References USB_BIT_RXF, and USB_BIT_TXE.

```
84 {
85   XMCRA |= _BV(SRE);XMCRB = _BV(XMBK);             // enable external Memory Interface (XRAM)
86   DDRE  &= ~(USB_BIT_RXF | USB_BIT_TXE);           // USB's status signals are inputs
87   PORTE |=  (USB_BIT_RXF | USB_BIT_TXE);           // switch internal pull up resitors on
88 }
```

6.9.3.4 static bool usb_keypressed (void) [static]

returns true if the USB chip has a character available for read

Returns:

bool false: No Char available, true: a char is available

Definition at line 98 of file main.c.

References USB_BIT_RXF, and USB_STATE.

```
99 {
100   return (USB_STATE & USB_BIT_RXF) == 0;             // FIFO is not full? return true
101 }
```

6.9.3.5 static int usb_putc (char *c*, struct __file * *dummy_file*) [static]

sends a character over the USB connection

waits while the USB chip anounces a full FIFO buffer

Parameters:

c the char to send

dummy_file not used

Definition at line 129 of file main.c.

References pUSB_Fifo, USB_BIT_TXE, and USB_STATE.

```
130 {
131   while (!((USB_STATE & USB_BIT_TXE) == 0));       // Wait for empty transmit buffer
132   *pUSB_Fifo = c;                                  // write the byte into the USB FIFO
133   return c;                                        // return the char
134 }
```

6.9.4 Variable Documentation

6.9.4.1 volatile unsigned char* pUSB_Fifo = (unsigned char *) USB_FIFO_AD [static]

Definition at line 73 of file main.c.

6.9.4.2 volatile unsigned char* pXRAM = (unsigned char *) XRAM_MEM_AD [static]

Definition at line 74 of file main.c.

Referenced by main().

6.9.4.3 FILE usb_stream = FDEV_SETUP_STREAM(usb_putc, usb_getc,_FDEV_SETUP_RW)

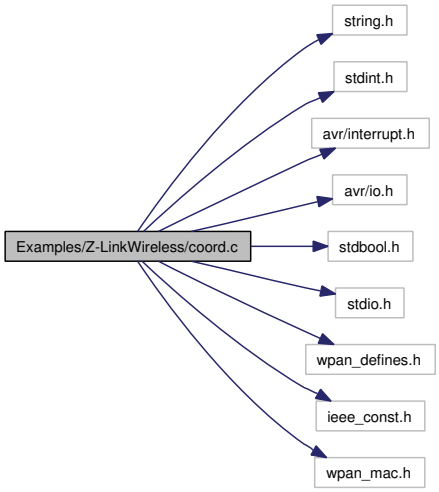
Definition at line 70 of file main.c.

6.10 Examples/Z-LinkWireless/coord.c File Reference

SensTermBoard Example: Coordinator receives temperature measures from devices.

```
#include <string.h>
#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdbool.h>
#include <stdio.h>
#include "wpan_defines.h"
#include "ieee_const.h"
#include "wpan_mac.h"
```

Include dependency graph for coord.c:



Data Structures

- struct **association_entry_t**
- struct **coord_status_t**

Defines

- #define **RF_CHANNEL** (18)
- #define **PANCOORD_SHORT_ADD** (0xBABE)

- #define **BROADCAST_SHORT_ADD** (0xFFFF)
- #define **PANID** (0xCAFE)
- #define **PANCOORD_SCAN_CHANNELS** ((uint32_t)0x00000000)
- #define **SCANDURATION** (3)
- #define **BEACON_ORDER** (15)
- #define **SUPERFRAME_ORDER** (15)
- #define **EMPTY_LONG_ADDRESS** (~(uint64_t)0)
- #define **MAX_ENTRIES** (8)
- #define **NO_ENTRY** (0xFF)
- #define **SET_STATE**(x)
- #define **USB_FIFO_AD** (0x2200)
- #define **USB_BIT_RXF** _BV(7)
- #define **USB_BIT_TXE** _BV(6)
- #define **USB_STATE** (PINE)
- #define **VT100_CLR_SCREEN** "\x1B[2J"
- #define **VT100_CUR_HOME** "\x1B[H"

Enumerations

- enum **coord_state_t** {

  INIT_DONE, **PEND_RESET**, **PEND_SET_SHORT_ADDR**, **PEND_INITIAL_SCAN**,

  PEND_START, **PEND_ASSOC_PERMIT**, **RUN** }

Functions

- void **application_init** (void)
- void **mac_do_reset** (void)
- void **mac_active_scan** (void)
- void **mac_start_pan** (void)
- void **mac_set_short_addr** (uint16_t addr)
- void **mac_set_assoc_permit** (uint8_t permit)
- void **mac_register_device** (uint64_t DeviceAddress)
- void **mac_send_data** (void)
- static void **association_table_init** (void)
- static uint8_t **get_empty_association_slot** (void)
- static uint8_t **search_association_entry** (uint64_t addr)
- static int **usb_putc** (char c, struct __file *dummy_file)

  sends a character over the USB connection

- static int **usb_getc** (struct __file *dummy_file)

  Waits till there is a character available on the USB chip and gives it back.

- static void **show_device_list** (void)

- static void **usb_init** (void)

  Initialize interface to USB chip.

- static bool **usb_keypressed** (void)

  returns true if the USB chip has a character available for read

- void **usr_mlme_reset_conf** (uint8_t status)
- void **usr_mlme_set_conf** (uint8_t status, uint8_t PIBAttribute)
- void **usr_mlme_scan_conf** (uint8_t status, uint8_t ScanType, uint32_t UnscannedChannels, uint8_t ResultListSize, uint8_t *data, uint8_t data_length)
- void **usr_mlme_start_conf** (uint8_t status)
- void **usr_mlme_associate_ind** (uint64_t DeviceAddress, uint8_t CapabilityInformation, uint8_t SecurityUse, uint8_t ACLEntry)
- void **usr_mlme_comm_status_ind** (wpan_commstatus_addr_t *pAddrInfo, uint8_t status)
- void **usr_mcps_data_ind** (wpan_mcpsdata_addr_t *addrInfo, uint8_t mpduLinkQuality, uint8_t SecurityUse, uint8_t ACLEntry, uint8_t msduLength, uint8_t *msdu)
- int **main** (void)

Variables

- static **coord_status_t c_status**
- static **association_entry_t association_table** [MAX_ENTRIES]
- uint8_t **data_buffer** [127]
- FILE **usb_stream** = FDEV_SETUP_STREAM(usb_putc, usb_getc,_FDEV_SETUP_RW)
- static volatile unsigned char * **pUSB_Fifo** = (unsigned char *) USB_FIFO_AD

6.10.1 Detailed Description

SensTermBoard Example: Coordinator receives temperature measures from devices.

The application shows a list of associated devices and lists there measured temperature on a connected VT100 terminal programm. Open Hyperterminal and connect to the virtual COM-Port that the SensTermBoard has got from Windows (Baudrate 115kBit, 8 data bits, no parity, 1 stop bit)

The device is implemented in **coord.c** (p. 77)

Note

- Input and Output is done by mechanism of the "stdio.h" interface. One need to setup a FILE stream with links to the putchar and getchar functions.

- the application uses floating point arithmetics. AVR-Studio has to be told to include the float version of the printf library:

Author:

Copyright (c) 2007 dresden elektronik, All rights reserved. `www.dresden-elektronik.de` Based on the example published by ATMEL Corporation, Copyright (c) 2006, Atmel Corporation All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file **coord.c**.

6.10.2 Define Documentation

6.10.2.1 #define BEACON_ORDER (15)

Definition at line 77 of file coord.c.

Referenced by mac_start_pan().

6.10.2.2 #define BROADCAST_SHORT_ADD (0xFFFF)

Definition at line 72 of file coord.c.

Referenced by mac_send_data().

6.10.2.3 #define EMPTY_LONG_ADDRESS (~(uint64_t)0)

Definition at line 79 of file coord.c.

Referenced by association_table_init(), and get_empty_association_slot().

6.10.2.4 #define MAX_ENTRIES (8)

Definition at line 81 of file coord.c.

Referenced by association_table_init(), get_empty_association_slot(), search_association_entry(), show_device_list(), and usr_mlme_comm_status_ind().

6.10.2.5 #define NO_ENTRY (0xFF)

Definition at line 82 of file coord.c.

Referenced by get_empty_association_slot(), mac_register_device(), search_association_entry(), and usr_mlme_comm_status_ind().

6.10.2.6 #define PANCOORD_SCAN_CHANNELS ((uint32_t)0x00000000)

Definition at line 75 of file coord.c.

Referenced by mac_active_scan().

6.10.2.7 #define PANCOORD_SHORT_ADD (0xBABE)

Definition at line 71 of file coord.c.

Referenced by mac_send_data(), and usr_mlme_reset_conf().

6.10.2.8 #define PANID (0xCAFE)

Definition at line 73 of file coord.c.

Referenced by mac_send_data(), mac_start_pan(), and usr_mlme_comm_status_ind().

6.10.2.9 #define RF_CHANNEL (18)

Definition at line 67 of file coord.c.

Referenced by mac_scan(), and mac_start_pan().

6.10.2.10 #define SCANDURATION (3)

Definition at line 76 of file coord.c.

Referenced by mac_active_scan().

6.10.2.11 #define SET_STATE(x)

Value:

```
do { c_status.state=(x); \
                         /* PORTE |= (_BV((uint8_t)(x))); */} while(0)
```

Definition at line 85 of file coord.c.

Referenced by application_init(), mac_active_scan(), mac_associate(), mac_do_reset(), mac_scan(), mac_set_assoc_permit(), mac_set_short_addr(), mac_start_pan(), usr_mlme_associate_conf(), and usr_mlme_set_conf().

6.10.2.12 #define SUPERFRAME_ORDER (15)

Definition at line 78 of file coord.c.

Referenced by mac_start_pan().

6.10.2.13 #define USB_BIT_RXF _BV(7)

Definition at line 89 of file coord.c.

6.10.2.14 #define USB_BIT_TXE _BV(6)

Definition at line 90 of file coord.c.

6.10.2.15 #define USB_FIFO_AD (0x2200)

Definition at line 88 of file coord.c.

6.10.2.16 #define USB_STATE (PINE)

Definition at line 91 of file coord.c.

6.10.2.17 #define VT100_CLR_SCREEN "\x1B[2J"

Definition at line 93 of file coord.c.

Referenced by show_device_list().

6.10.2.18 #define VT100_CUR_HOME "\x1B[H"

Definition at line 94 of file coord.c.

Referenced by show_device_list().

6.10.3 Enumeration Type Documentation

6.10.3.1 enum coord_state_t

Enumerator:

INIT_DONE
PEND_RESET
PEND_SET_SHORT_ADDR
PEND_INITIAL_SCAN
PEND_START
PEND_ASSOC_PERMIT
RUN

Definition at line 103 of file coord.c.

```
104 {
105     INIT_DONE,
106     PEND_RESET,
107     PEND_SET_SHORT_ADDR,
108     PEND_INITIAL_SCAN,
109     PEND_START,
110     PEND_ASSOC_PERMIT,
111     RUN,
112 } coord_state_t;
```

6.10.4 Function Documentation

6.10.4.1 static void application_init (void)

Definition at line 212 of file coord.c.

References association_table_init(), c_status, coord_status_t::handle, INIT_DONE, coord_status_t::led_value, and SET_STATE.

Referenced by main().

```
213 {
214
215     c_status.led_value = 0;
216     c_status.handle = 0;
217     association_table_init();
218
219     /* init IO ports */
220     DDRE |=  (_BV(2) | _BV(3) | _BV(4))/* 0xFF */;  /* all bits of PORT E are outputs */
```

```
221     PORTE &= (_BV(2) | _BV(3) | _BV(4)) /* 0x00 */; /* all LED's ON */
222
223     /* init mac layer */
224     wpan_init();
225     SET_STATE(INIT_DONE);
226
227     /* enable interrupts */
228     sei();
229     return;
230 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.10.4.2 void association_table_init (void) [static]

Definition at line 338 of file coord.c.

References association_table, EMPTY_LONG_ADDRESS, association_entry_t::long_addr, and MAX_ENTRIES.

Referenced by application_init().

```
339 {
340     for(uint16_t i = 0; i < MAX_ENTRIES; i++)
341     {
342         association_table[i].associated = false;
343         association_table[i].long_addr = EMPTY_LONG_ADDRESS;
344     }
345     return;
346 }
```

Here is the caller graph for this function:

6.10.4.3 uint8_t get_empty_association_slot (void) [static]

Definition at line 349 of file coord.c.

References association_table, EMPTY_LONG_ADDRESS, MAX_ENTRIES, and NO_ENTRY.

Referenced by mac_register_device().

```
350 {
351     uint8_t ret = NO_ENTRY, idx;
352
353     for(idx = 0; idx < MAX_ENTRIES; idx++)
354     {
355         if(association_table[idx].long_addr == EMPTY_LONG_ADDRESS)
356         {
357             ret = idx;
358             break;
359         }
360     }
361
362     return ret;
363 }
```

Here is the caller graph for this function:

6.10.4.4 void mac_active_scan (void)

Definition at line 290 of file coord.c.

References PANCOORD_SCAN_CHANNELS, PEND_INITIAL_SCAN, SCANDURATION, and SET_STATE.

Referenced by usr_mlme_set_conf().

```
291 {
292     wpan_mlme_scan_request (MLME_SCAN_TYPE_ACTIVE, PANCOORD_SCAN_CHANNELS, SCANDURATION);
293     SET_STATE(PEND_INITIAL_SCAN);
294     return;
295 }
```

Here is the caller graph for this function:

6.10.4.5 void mac_do_reset (void)

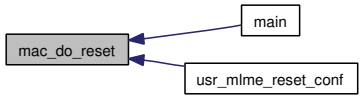
Definition at line 232 of file coord.c.

References PEND_RESET, and SET_STATE.

Referenced by main(), and usr_mlme_reset_conf().

```
233 {
234     wpan_mlme_reset_request( true );
235     SET_STATE( PEND_RESET );
236 }
```

Here is the caller graph for this function:



6.10.4.6 void mac_register_device (uint64_t *DeviceAddress*)

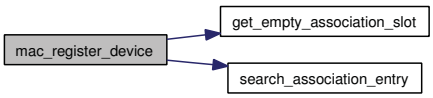
Definition at line 393 of file coord.c.

References association_table, get_empty_association_slot(), association_entry_t::long_addr, NO_ENTRY, and search_association_entry().

Referenced by usr_mlme_associate_ind().

```
394 {
395     uint8_t entry_index;
396 
397     /*  Search if device is already associated. */
398 
399     entry_index = search_association_entry(DeviceAddress);
400     if(entry_index == NO_ENTRY)
401     {
402         entry_index = get_empty_association_slot();
403     }
404 
405     if (entry_index == NO_ENTRY)
406     {
407         wpan_mlme_associate_response (DeviceAddress, 0,
408             PAN_AT_CAPACITY, false);
409     }
410     else
411     {
412         association_table[entry_index].long_addr = DeviceAddress;
413         wpan_mlme_associate_response(DeviceAddress, entry_index,
414             ASSOCIATION_SUCCESSFUL, false);
415     }
416     return;
417 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.10.4.7 void mac_send_data (void)

Definition at line 462 of file coord.c.

References BROADCAST_SHORT_ADD, c_status, coord_status_t::handle, coord_status_t::led_value, PANCOORD_SHORT_ADD, and PANID.

```
463 {
464     wpan_mcpsdata_addr_t addr_info;
465     addr_info.SrcAddrMode = WPAN_ADDRMODE_SHORT;
466     addr_info.SrcPANId = PANID;
467     addr_info.SrcAddr = PANCOORD_SHORT_ADD;
468     addr_info.DstAddrMode = WPAN_ADDRMODE_SHORT;
469     addr_info.DstPANId = PANID;
470     addr_info.DstAddr = BROADCAST_SHORT_ADD;
471     wpan_mcps_data_request( &addr_info, c_status.handle++, WPAN_TXOPT_OFF,
472                             (void *)&c_status.led_value,
473                             sizeof(c_status.led_value));
474     return;
475 }
```

6.10.4.8 void mac_set_assoc_permit (uint8_t *permit*)

Definition at line 246 of file coord.c.

References PEND_ASSOC_PERMIT, and SET_STATE.

Referenced by usr_mlme_start_conf().

```
247 {
248     wpan_mlme_set_request (macAssociationPermit, &permit, sizeof(permit));
249     SET_STATE( PEND_ASSOC_PERMIT );
250 }
```

Here is the caller graph for this function:

6.10.4.9 void mac_set_short_addr (uint16_t *addr*)

Definition at line 239 of file coord.c.

References PEND_SET_SHORT_ADDR, and SET_STATE.

Referenced by usr_mlme_reset_conf().

```
240 {
241     wpan_mlme_set_request (macShortAddress, &addr, sizeof(addr));
242     SET_STATE ( PEND_SET_SHORT_ADDR );
243 }
```

Here is the caller graph for this function:

6.10.4.10 void mac_start_pan (void)

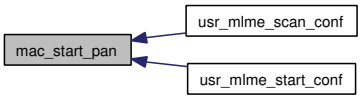
Definition at line 311 of file coord.c.

References BEACON_ORDER, PANID, PEND_START, RF_CHANNEL, SET_STATE, and SUPERFRAME_ORDER.

Referenced by usr_mlme_scan_conf(), and usr_mlme_start_conf().

```
312 {
313     wpan_mlme_start_request ( PANID, RF_CHANNEL,
314                               BEACON_ORDER, SUPERFRAME_ORDER,
315                               true, false, false, false);
316     SET_STATE(PEND_START);
317     return;
318 }
```

Here is the caller graph for this function:



6.10.4.11 int main (void)

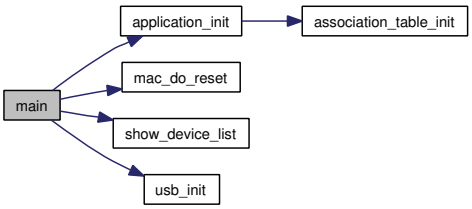
Definition at line 498 of file coord.c.

References application_init(), mac_do_reset(), show_device_list(), usb_init(), and usb_stream.

```
499 {
500     usb_init();                                     // init USB's hardware
501     stdout = &usb_stream;                           // init standard output over USB
502     stdin  = &usb_stream;                           // init standard input over USB
503
504     printf("Hello\n");
505     application_init();
506     printf("App init\n");
507     mac_do_reset();
```

```
508     printf("MAC reset\n");
509     show_device_list();
510     while(1)
511     {
512         while(wpan_task())
513         {
514         }
515     }
516 }
```

Here is the call graph for this function:



6.10.4.12 uint8_t search_association_entry (uint64_t *addr*) [static]

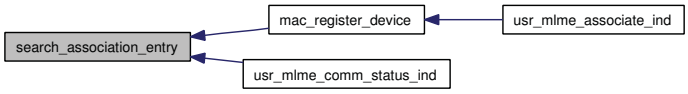
Definition at line 366 of file coord.c.

References association_table, MAX_ENTRIES, and NO_ENTRY.

Referenced by mac_register_device(), and usr_mlme_comm_status_ind().

```
367 {
368 uint8_t ret = NO_ENTRY, idx;
369     /*  Look for the address in the table. */
370     for(idx = 0; idx < MAX_ENTRIES; idx++)
371     {
372         if(association_table[idx].long_addr == addr)
373         {
374             ret = idx;
375             break;
376         }
377     }
378
379     return ret;
380 }
```

Here is the caller graph for this function:



6.10.4.13 static void show_device_list (void) [static]

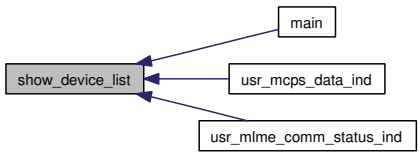
Definition at line 478 of file coord.c.

References association_table, MAX_ENTRIES, VT100_CLR_SCREEN, and VT100_CUR_HOME.

Referenced by main(), usr_mcps_data_ind(), and usr_mlme_comm_status_ind().

```
479 {
480   int i;
481
482   printf(VT100_CLR_SCREEN);
483   printf(VT100_CUR_HOME);
484
485   for ( i = 0; i < MAX_ENTRIES; i++)
486   {
487     if (association_table[i].associated)
488     {
489       printf("Device %d: %02.1f\n", i, ((double)association_table[i].temperature)/10.0);
490     }
491     else
492     {
493       printf("Device %d: not found\n",i);
494     }
495   }
496 }
```

Here is the caller graph for this function:



6.10.4.14 static int usb_getc (struct __file * *dummy_file*) [static]

Waits till there is a character available on the USB chip and gives it back.

waits forever when no char comes in

Returns:

char This is the character on the USB chip

Definition at line 187 of file coord.c.

References pUSB_Fifo, and usb_keypressed().

```
188 {
```

```
189   while (!usb_keypressed());                    // check for incomming data
190   return *pUSB_Fifo;                            // Return the data
191 }
```

Here is the call graph for this function:

6.10.4.15 static void usb_init (void) [static]

Initialize interface to USB chip.

Definition at line 158 of file coord.c.

References USB_BIT_RXF, and USB_BIT_TXE.

```
159 {
160   XMCRA |= _BV(SRE);XMCRB = _BV(XMBK);           // enable external Memory Interface (XRAM)
161   DDRE  &= ~(USB_BIT_RXF | USB_BIT_TXE);        // USB's status signals are inputs
162   PORTE |=  (USB_BIT_RXF | USB_BIT_TXE);        // switch internal pull up resitors on
163 }
```

6.10.4.16 static bool usb_keypressed (void) [static]

returns true if the USB chip has a character available for read

Returns:

bool false: No Char available, true: a char is available

Definition at line 173 of file coord.c.

References USB_BIT_RXF, and USB_STATE.

```
174 {
175   return (USB_STATE & USB_BIT_RXF) == 0;          // FIFO is not full? return true
176 }
```

6.10.4.17 static int usb_putc (char *c*, struct __file * *dummy_file*) [static]

sends a character over the USB connection

waits while the USB chip anounces a full FIFO buffer

Parameters:

c the char to send

dummy_file not used

Definition at line 204 of file coord.c.

References pUSB_Fifo, USB_BIT_TXE, and USB_STATE.

```
205 {
206   while (!((USB_STATE & USB_BIT_TXE) == 0));       // Wait for empty transmit buffer
207   *pUSB_Fifo = c;                                  // write the byte into the USB FIFO
208   return c;                                        // return the char
209 }
```

6.10.4.18 void usr_mcps_data_ind (wpan_mcpsdata_addr_t * *addrInfo*, uint8_t *mpduLinkQuality*, uint8_t *SecurityUse*, uint8_t *ACLEntry*, uint8_t *msduLength*, uint8_t * *msdu*)

Definition at line 445 of file coord.c.

References association_entry_t::associated, association_table, show_device_list(), and association_entry_t::temperature.

```
448 {
449     /* Determine if the indication comes from a device,
450      * that has previously associated.
451      */
452     if (addrInfo->SrcAddrMode == WPAN_ADDRMODE_SHORT
453         && association_table[(uint8_t)addrInfo->SrcAddr].associated)
454     {
455         association_table[(uint8_t)addrInfo->SrcAddr].temperature = *(uint16_t *)msdu;
456         show_device_list();
457     }
458     return;
459 }
```

Here is the call graph for this function:

6.10.4.19 void usr_mlme_associate_ind (uint64_t *DeviceAddress*, uint8_t *CapabilityInformation*, uint8_t *SecurityUse*, uint8_t *ACLEntry*)

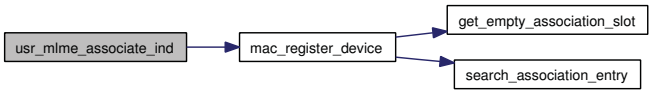
Definition at line 383 of file coord.c.

References c_status, mac_register_device(), RUN, and coord_status_t::state.

```
385 {
386     if (c_status.state == RUN)
387     {
```

```
388         mac_register_device(DeviceAddress);
389     }
390 }
```

Here is the call graph for this function:



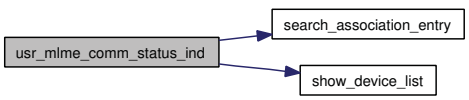
6.10.4.20 void usr_mlme_comm_status_ind (wpan_commstatus_addr_t * *pAddrInfo*, uint8_t *status*)

Definition at line 420 of file coord.c.

References association_entry_t::associated, association_table, MAX_ENTRIES, NO_ENTRY, PANID, search_association_entry(), and show_device_list().

```
421 {
422 uint16_t assoc_idx = NO_ENTRY;
423
424     if (pAddrInfo->PANId == PANID) /* if it is our PAN */
425     {
426         if (pAddrInfo->DstAddrMode == WPAN_ADDRMODE_SHORT)
427         {
428             assoc_idx = pAddrInfo->DstAddr;
429         }
430         else if (pAddrInfo->DstAddrMode == WPAN_ADDRMODE_LONG)
431         {
432             assoc_idx = search_association_entry(pAddrInfo->DstAddr);
433         }
434
435         if ((status == MAC_SUCCESS) && (assoc_idx < MAX_ENTRIES))
436         {
437             association_table[assoc_idx].associated = true;
438         }
439
440         show_device_list();
441     }
442 }
```

Here is the call graph for this function:



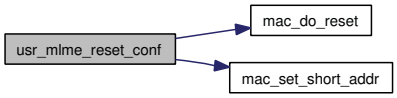
6.10.4.21 void usr_mlme_reset_conf (uint8_t *status*)

Definition at line 253 of file coord.c.

References c_status, mac_do_reset(), mac_set_short_addr(), PANCOORD_SHORT_ADD, PEND_RESET, and coord_status_t::state.

```
254 {
255     if ((status == MAC_SUCCESS) && (c_status.state == PEND_RESET))
256     {
257         mac_set_short_addr(PANCOORD_SHORT_ADD);
258     }
259     else
260     {
261         mac_do_reset();
262     }
263     return;
264 }
```

Here is the call graph for this function:



6.10.4.22 void usr_mlme_scan_conf (uint8_t *status*, uint8_t *ScanType*, uint32_t *UnscannedChannels*, uint8_t *ResultListSize*, uint8_t * *data*, uint8_t *data_length*)

Definition at line 298 of file coord.c.

References c_status, mac_start_pan(), PEND_INITIAL_SCAN, and coord_status_t::state.

```
300 {
301     if (c_status.state == PEND_INITIAL_SCAN)
302     {
303         /* We don't care about the confirm of the scan request because the scan */
304         /* request just puts the MAC into the correct state for a pancoord. */
305         mac_start_pan();
306     }
307     return;
308 }
```

Here is the call graph for this function:

6.10.4.23 void usr_mlme_set_conf (uint8_t *status*, uint8_t *PIBAttribute*)

Definition at line 266 of file coord.c.

References c_status, mac_active_scan(), PEND_ASSOC_PERMIT, PEND_SET_SHORT_ADDR, RUN, SET_STATE, and coord_status_t::state.

```
267 {
268     switch (c_status.state)
269     {
270         case PEND_SET_SHORT_ADDR:
271             if ((status == MAC_SUCCESS) && (PIBAttribute == macShortAddress))
272             {
273                 mac_active_scan();
274             }
275             break;
276         case PEND_ASSOC_PERMIT:
277             if ((status == MAC_SUCCESS) && (PIBAttribute == macAssociationPermit))
278             {
279                 SET_STATE( RUN );
280                 PORTE |=  (_BV(2) | _BV(3) | _BV(4)); /* LED's off if we come to here */
281             }
282         default:
283             break;
284     }
285
286     return;
287 }
```

Here is the call graph for this function:

6.10.4.24 void usr_mlme_start_conf (uint8_t *status*)

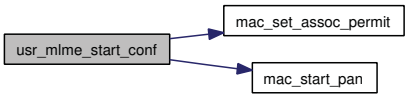
Definition at line 321 of file coord.c.

References c_status, mac_set_assoc_permit(), mac_start_pan(), PEND_START, and coord_status_t::state.

```
322 {
323     if  (c_status.state == PEND_START)
324     {
325         if (status == MAC_SUCCESS)
326         {
327             mac_set_assoc_permit( 1 );
328         }
329         else
330         {
331             mac_start_pan();
332         }
333     }
```

```
334     return;
335 }
```

Here is the call graph for this function:



6.10.5 Variable Documentation

6.10.5.1 association_entry_t association_table[MAX_ENTRIES] [static]

Definition at line 141 of file coord.c.

Referenced by association_table_init(), get_empty_association_slot(), mac_register_device(), search_association_entry(), show_device_list(), usr_mcps_data_ind(), and usr_mlme_comm_status_ind().

6.10.5.2 coord_status_t c_status [static]

Definition at line 140 of file coord.c.

Referenced by application_init(), mac_send_data(), usr_mlme_associate_ind(), usr_mlme_reset_conf(), usr_mlme_scan_conf(), usr_mlme_set_conf(), and usr_mlme_start_conf().

6.10.5.3 uint8_t data_buffer[127]

Definition at line 143 of file coord.c.

6.10.5.4 volatile unsigned char* pUSB_Fifo = (unsigned char *) USB_FIFO_AD [static]

Definition at line 149 of file coord.c.

6.10.5.5 FILE usb_stream = FDEV_SETUP_STREAM(usb_putc, usb_getc, _FDEV_SETUP_RW)

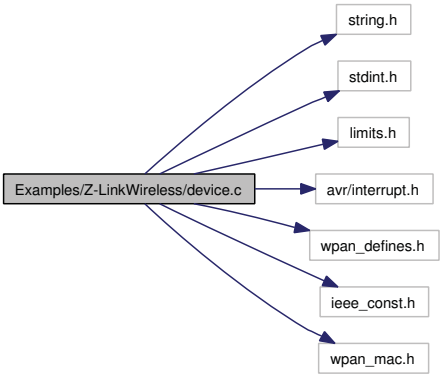
Definition at line 146 of file coord.c.

6.11 Examples/Z-LinkWireless/device.c File Reference

SensTermBoard Example: Device measures temperature and sends it to a coordinator.

```
#include <string.h>
#include <stdint.h>
#include <limits.h>
#include <avr/interrupt.h>
#include "wpan_defines.h"
#include "ieee_const.h"
#include "wpan_mac.h"
```

Include dependency graph for device.c:



Data Structures

- struct **device_status_t**

Defines

- #define **RF_CHANNEL** (18)
- #define **ALL_HIGH_BAND_CHANNELS** ((uint32_t)0x07FFF800)
- #define **CHANNELMASK**(a) (1UL<<(a))
- #define **SCAN_DURATION** (3)
- #define **SET_STATE**(x) do { d_status.state=(x); } while(0)
- #define **IO_THERM_AD** (0x4000)

Enumerations

- enum **device_state_t** {

  INIT_DONE, **PEND_RESET**, **PEND_SCAN**, **PEND_ASSOCIATE**,

  PEND_SET_SHORT_ADDR, **PEND_START**, **RUN** }

Functions

- static void **application_init** (void)
- static void **switch_task** (void)
- void **mac_do_reset** (void)
- static void **mac_scan** (void)
- static void **mac_associate** (void)
- uint16_t **pwr_read_adc** (uint8_t mux)

  gibt am ADC-Eingang mux gemessene Spannung in mV zurück

- static void **temp_init** (void)

  Initialize temperature hardware (ADC, Thermistor enable).

- static uint16_t **temp_get_degrcelc** (void)

  Gives back the actual temperature.

- void **usr_mlme_reset_conf** (uint8_t status)
- void **usr_mlme_scan_conf** (uint8_t status, uint8_t ScanType, uint32_t UnscannedChannels, uint8_t ResultListSize, uint8_t *data, uint8_t data_length)
- void **usr_mlme_associate_conf** (uint16_t AssocShortAddress, uint8_t status)
- void **usr_mcps_data_ind** (wpan_mcpsdata_addr_t *pAddrInfo, uint8_t mpduLinkQuality, uint8_t SecurityUse, uint8_t ACLEntry, uint8_t msduLength, uint8_t *msdu)
- int **main** (void)

Variables

- static **device_status_t d_status**
- static volatile unsigned char * **pIO_THERM** = (unsigned char *) IO_THERM_AD

6.11.1 Detailed Description

SensTermBoard Example: Device measures temperature and sends it to a coordinator.

The application measures the temperature and sends it to a coordinator found on the 802.15.4 network.

The coordinator is implemented in **coord.c** (p. 77)

Note

- Input and Output is done by mechanism of the "stdio.h" interface. One need to setup a FILE stream with links to the putchar and getchar functions.

- the application uses floating point arithmetics. AVR-Studio has to be told to include the float version of the printf library:

Author:

Definition in file **device.c**.

6.11.2 Define Documentation

6.11.2.1 #define ALL_HIGH_BAND_CHANNELS ((uint32_t)0x07FFF800)

Definition at line 66 of file device.c.

6.11.2.2 #define CHANNELMASK(a) (1UL<<(a))

Definition at line 67 of file device.c.

Referenced by mac_scan().

6.11.2.3 #define IO_THERM_AD (0x4000)

Definition at line 72 of file device.c.

6.11.2.4 #define RF_CHANNEL (18)

Definition at line 62 of file device.c.

6.11.2.5 #define SCAN_DURATION (3)

Definition at line 68 of file device.c.

Referenced by mac_scan().

6.11.2.6 #define SET_STATE(x) do { d_status.state=(x); } while(0)

Definition at line 71 of file device.c.

6.11.3 Enumeration Type Documentation

6.11.3.1 enum device_state_t

Enumerator:

INIT_DONE
PEND_RESET
PEND_SCAN
PEND_ASSOCIATE
PEND_SET_SHORT_ADDR
PEND_START
RUN

Definition at line 75 of file device.c.

```
76 {
77     INIT_DONE,
78     PEND_RESET,
79     PEND_SCAN,
80     PEND_ASSOCIATE,
81     PEND_SET_SHORT_ADDR,
82     PEND_START,
83     RUN,
84 } device_state_t;
```

6.11.4 Function Documentation

6.11.4.1 static void application_init (void) [static]

6.11.4.2 static void mac_associate (void) [static]

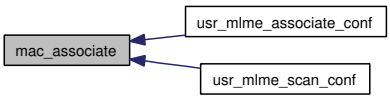
Definition at line 360 of file device.c.

References device_status_t::coord_address, device_status_t::coord_address_mode, d_status, device_status_t::logical_channel, device_status_t::pan_id, PEND_ASSOCIATE, and SET_STATE.

Referenced by usr_mlme_associate_conf(), and usr_mlme_scan_conf().

```
361 {
362     uint8_t capability_info;
363     capability_info = WPAN_CAP_FFD | WPAN_CAP_PWRSOURCE |\
364                       WPAN_CAP_RXONWHENIDLE | WPAN_CAP_ALLOCADDRESS;
365
366     wpan_mlme_associate_request(d_status.logical_channel,
367                                 d_status.coord_address_mode,
368                                 d_status.pan_id, d_status.coord_address,
369                                 capability_info, false);
370
371
372     SET_STATE( PEND_ASSOCIATE );
373     return;
374 }
```

Here is the caller graph for this function:



6.11.4.3 void mac_do_reset (void)

6.11.4.4 static void mac_scan (void) [static]

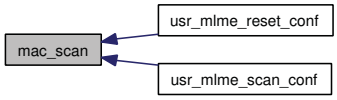
Definition at line 316 of file device.c.

References CHANNELMASK, PEND_SCAN, RF_CHANNEL, SCAN_DURATION, and SET_STATE.

Referenced by usr_mlme_reset_conf(), and usr_mlme_scan_conf().

```
317 {
318     uint32_t chanmsk;
319     chanmsk = CHANNELMASK(RF_CHANNEL);
320     wpan_mlme_scan_request(MLME_SCAN_TYPE_ACTIVE, chanmsk, SCAN_DURATION);
321     SET_STATE( PEND_SCAN );
322    return;
323 }
```

Here is the caller graph for this function:



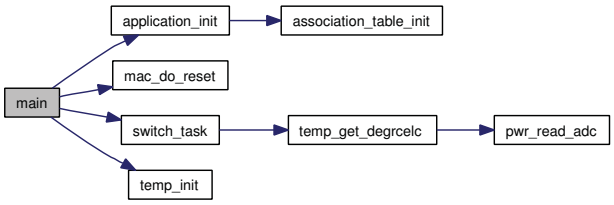
6.11.4.5 int main (void)

Definition at line 421 of file device.c.

References application_init(), mac_do_reset(), switch_task(), and temp_init().

```
422 {
423     temp_init();
424     application_init();
425
426     mac_do_reset();
427
428     while(1)
429     {
430         while(wpan_task())
431         {
432             /* only short running tasks are called here */
433         }
434         /* main user task */
435         switch_task();
436     }
437 }
```

Here is the call graph for this function:



6.11.4.6 uint16_t pwr_read_adc (uint8_t *mux*)

gibt am ADC-Eingang mux gemessene Spannung in mV zurück

Kanal 0: Batteriespannung, 1: Netzspannung

Parameters:

mux 0..1: Kanal

Returns:

uint16_t Spannung in mV

Definition at line 149 of file device.c.

```
150 {
151   uint32_t result;
152
153   ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);    // Frequenzvorteiler
154                                 // setzen auf 8 (1) und ADC aktivieren (1)
155
156   ADMUX = mux;                        // Kanal waehlen
157   ADMUX |= (1<<REFS0); // Referenz ist 3,3V VCC
158
159   /* nach Aktivieren des ADC wird ein "Dummy-Readout" empfohlen, man liest
160      also einen Wert und verwirft diesen, um den ADC "warmlaufen zu lassen" */
161   ADCSRA |= (1<<ADSC);              // eine ADC-Wandlung
162   while ( ADCSRA & (1<<ADSC) ) {
163      ;     // auf Abschluss der Konvertierung warten
164   }
165   result  = ADC;
166   result  = (3223UL*result);   // 3300mV bei 1024 Schritten sind 3223mV je Schritt
167   result /= 1000;
168
169   return (uint16_t)result;
170 }
```

6.11.4.7 static void switch_task (void) [static]

Definition at line 265 of file device.c.

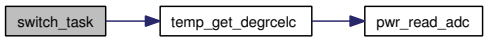
References device_status_t::coord_address, device_status_t::coord_address_mode, d_status, device_status_t::device_short_address, device_status_t::msdu_handle, device_status_t::pan_id, RUN, device_status_t::state, and temp_get_degrcelc().

Referenced by main().

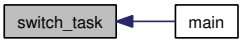
```
266 {
267     if (d_status.state == RUN)
268     {
269         static uint16_t last_temp = INT_MAX;
270         uint16_t temp = temp_get_degrcelc();
271         bool send_data = false;
272
```

```
273         // temperature change more then 0.5 degrees?
274         if ((temp > (last_temp + 5)) || (temp < (last_temp -5)))
275         {
276           last_temp = temp;
277           send_data = true;
278         }
279
280         if (send_data)
281         {
282             /* send data */
283             wpan_mcpsdata_addr_t addr_info;
284             addr_info.SrcAddrMode = WPAN_ADDRMODE_SHORT;
285             addr_info.SrcPANId = d_status.pan_id;
286             addr_info.SrcAddr = d_status.device_short_address;
287             addr_info.DstAddrMode = d_status.coord_address_mode;
288             addr_info.DstPANId = d_status.pan_id;
289             addr_info.DstAddr = d_status.coord_address;
290
291             wpan_mcps_data_request(&addr_info,
292                                    d_status.msdu_handle++, WPAN_TXOPT_ACK,
293                                    (void *)&last_temp, sizeof(last_temp));
294         }
295     }
296     return;
297 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.4.8 static uint16_t temp_get_degrcelc (void) [static]

Gives back the actual temperature.

uses a table with precalculated voltage measures and does linear interpolation between two sampling points.

Returns:

double The measured temperature in degrees celsius

Definition at line 212 of file device.c.

References pwr_read_adc().

```
213 {
214   static struct _ttable                          // table with precalculated volatages
```

```
215   {
216     int temp; int millivolts;
217   } ttable[] =
218   {
219     { -40 ,  86   } ,
220     { -35 ,  120  } ,
221     { -30 ,  164  } ,
222     { -25 ,  221  } ,
223     { -20 ,  292  } ,
224     { -15 ,  380  } ,
225     { -10 ,  487  } ,
226     { -5  ,  612  } ,
227     { 0   ,  756  } ,
228     { 5   ,  916  } ,
229     { 10  ,  1091 } ,
230     { 15  ,  1274 } ,
231     { 20  ,  1462 } ,
232     { 25  ,  1650 } ,
233     { 30  ,  1832 } ,
234     { 35  ,  2006 } ,
235     { 40  ,  2166 } ,
236     { 45  ,  2313 } ,
237     { 50  ,  2445 } ,
238     { 55  ,  2562 } ,
239     { 60  ,  2665 } ,
240     { 65  ,  2754 } ,
241     { 70  ,  2830 } ,
242     { 75  ,  2897 } ,
243   };
244
245   uint16_t temp = 0;
246   uint16_t i, volt;
247
248   volt = pwr_read_adc(3);                          // read ADC in millivolts
249
250   for (i = 1; i < (sizeof(ttable)/sizeof(ttable[0])-1); i++)
251   {
252     if (volt < ttable[i].millivolts)               // look for a fitting table entry
253     {
254       i--;
255       temp = (uint16_t)(10.0 *                                 // scale for transmission
256                         ((double)ttable[i].temp                // linear Interpolation
257                 + 5.0 * (double) (volt - ttable[i].millivolts)
258                       / (double)(ttable[i+1].millivolts - ttable[i].millivolts)));
259       break;
260     }
261   }
262
263   return temp;                                     // give back the temperature
264 }
```

Here is the call graph for this function:

6.11.4.9 static void temp_init (void) [static]

Initialize temperature hardware (ADC, Thermistor enable).

As reference the 3,3V system power is used. So you have a more accurate reference than the internal of the AVR. The full range of the ADC can be used.

Definition at line 181 of file device.c.

References pIO_THERM.

```
182 {
183   PRR0 &= ~(1 << PRADC);                          // power up ADC
184
185   ADCSRA = (1 << ADPS2) | (1 << ADPS1)            // divider 128 on
186            | (1 << ADPS0);
187   ADCSRB = 0x00;                                  // not the free running mode
188
189   DIDR0 = _BV(1) | _BV(0);                        // disable digital input buffers
190
191   ACSR = (1 << ACD);                              // disable analog comperator
192
193
194   XMCRA |= _BV(SRE);XMCRB = _BV(XMBK);            // enable external Memory Interface (XRAM)
195
196   *pIO_THERM = 0x03;                              // Thermistor On, LEDs off
197
198 }
```

6.11.4.10 void usr_mcps_data_ind (wpan_mcpsdata_addr_t * *pAddrInfo*, uint8_t *mpduLinkQuality*, uint8_t *SecurityUse*, uint8_t *ACLEntry*, uint8_t *msduLength*, uint8_t * *msdu*)

Definition at line 399 of file device.c.

References d_status, device_status_t::pan_id, RUN, and device_status_t::state.

```
402 {
403     if ((d_status.state == RUN) && (pAddrInfo->DstPANId == d_status.pan_id))
404     {
405         /* Data packet contains LED information. */
406         /*  Mask the data byte with the address info. */
407         if(*msdu)
408         {
409             PORTE &= ~_BV(2);
410             //c_status.led_value |= (1 << (uint8_t)addrInfo->SrcAddr);
411         }
412         else
413         {
414             PORTE |= _BV(2);
415             //c_status.led_value &= ~(1 << (uint8_t)addrInfo->SrcAddr);
416         }
417     }
418 }
```

6.11.4.11 void usr_mlme_associate_conf (uint16_t *AssocShortAddress*, uint8_t *status*)

Definition at line 376 of file device.c.

References d_status, device_status_t::device_short_address, mac_associate(), PEND_ASSOCIATE, RUN, SET_STATE, and device_status_t::state.

```
377 {
378     if ((status == MAC_SUCCESS) && (d_status.state == PEND_ASSOCIATE))
379     {
380         /* save the device short address */
381         d_status.device_short_address = AssocShortAddress;
382 
383         /* mark that association is complete */
384         SET_STATE( RUN );
385 
386         /* turn all LEDs off after successfull association */
387         PORTE |= (_BV(2) | _BV(3) | _BV(4));
388     }
389     else
390     {
391         /* somethig went wrong, try  association again */
392         mac_associate();
393     }
394 
395     return;
396 }
```

Here is the call graph for this function:

6.11.4.12 void usr_mlme_reset_conf (uint8_t *status*)

Definition at line 305 of file device.c.

References d_status, mac_scan(), PEND_RESET, and device_status_t::state.

```
306 {
307     if ((status == MAC_SUCCESS) && (d_status.state == PEND_RESET))
308     {
309         mac_scan();
310     }
311 
312     return;
313 }
```

Here is the call graph for this function:

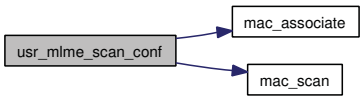
6.11.4.13 void usr_mlme_scan_conf (uint8_t *status*, uint8_t *ScanType*, uint32_t *UnscannedChannels*, uint8_t *ResultListSize*, uint8_t * *data*, uint8_t *data_length*)

Definition at line 326 of file device.c.

References device_status_t::coord_address, device_status_t::coord_address_mode, d_status, device_status_t::logical_channel, mac_associate(), mac_scan(), device_status_t::pan_id, PEND_SCAN, and device_status_t::state.

```
329 {
330     bool scan_success = false;
331
332     if ((status == MAC_SUCCESS) && (d_status.state == PEND_SCAN))
333     {
334         /* there should only be one PAN descriptor */
335         if (ResultListSize == 1)
336         {
337             scan_success = true;
338             wpan_pandescriptor_t *pandesc = (wpan_pandescriptor_t *)data;
339
340             /* save information from the PAN Descriptor */
341             d_status.coord_address_mode = pandesc->CoordAddrMode;
342             d_status.coord_address = pandesc->CoordAddress;
343             d_status.pan_id = pandesc->CoordPANId;
344             d_status.logical_channel = pandesc->LogicalChannel;
345
346             /* associate to the PAN coordinator */
347             mac_associate();
348         }
349     }
350
351     if (!scan_success)
352     {
353         /* no success, scan again */
354         mac_scan();
355     }
356
357     return;
358 }
```

Here is the call graph for this function:



6.11.5 Variable Documentation

6.11.5.1 device_status_t d_status [static]

Definition at line 101 of file device.c.

Referenced by mac_associate(), switch_task(), usr_mcps_data_ind(), usr_mlme_associate_conf(), usr_mlme_reset_conf(), and usr_mlme_scan_conf().

6.11.5.2 volatile unsigned char* pIO_THERM = (unsigned char *) IO_THERM_AD [static]

Definition at line 103 of file device.c.

Chapter 7

SENSOR TERMINAL BOARD by dresden elektronik Page Documentation

7.1 copyright of the example applications

Author:

RIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7.2 Getting Startet

This guide shows the preparation and programming of applications for the SensTermBoard togehter with a JTAGICE MKII debugger (which you can buy separatly).

1. First you should install `AVR Studio` and `WinAVR` if you do not have it on you computer till now. Simply following all instructions and rebooting the computer if the installer recommends it.

2. Second the parts has to be put together. The image shows a working setup. The colors of the LEDs are showing the right connection of the JTAGICE MKII debugger.

3. Then we open an AVR-Studio project file out of the `Examples` directory. In this example we use the `Button_changes_led.aps` project. AVR-Studio starts with the choosen project.

4. Now we are ready for building our first application. Just choosing the function "Build/Rebuild All" in the main menu of AVR-Studio.

5. In the next step we prepare the device and AVR-Studio for downloading and debugging. We have to tell AVR-Studio the kind of programmer and AVR device:

6. After this we have to set some important so called **fuses** of the AVR device. The fuses are configuration options and determine the way the AVR is generating its CPU clock, getting its programm memory contents and so on. We are using the internal 8 MHz RC-clock. After one has shoosen all necessary fuses in the window, the button "Program" sets the fuses in the AVR device to the desired values. Your AVR-Studio should show the same message at the bottom of the window: "reading fuses .. 0xFD0x, 0x12, 0xD2 .. OK!". The right values are important for proer function.

7. Now we will start the download and debugging process. Just follow the steps in the pictures:

8. Here we are! The picture shows the appearance of AV-Studio at debugging the first application.

7.3 PCB SensTermBoard

7.3.1 Schematic files for the SensTermBoard

The schematic of the SensTermBoard is available as `Schematic_SensTerm.pdf`.

Here is a `larger picture`.

7.4 PCB Radio Controller Board

7.4.1 Schematic files for the SensTermBoard

The ZIP-File `ATAVRRZ200_DemonstrationKit.zip` with documentation and schematic and Gerber files of the RCB can be found on ATMEL's website.

Index

ALL_HIGH_BAND_CHANNELS
    device.c, 100
application_init
    coord.c, 83
    device.c, 101
associated
    association_entry_t, 11
association_entry_t, 11
    associated, 11
    long_addr, 11
    temperature, 12
association_table
    coord.c, 96
association_table_init
    coord.c, 84

BAUDRATE
    Uart/main.c, 63
BEACON_ORDER
    coord.c, 80
BROADCAST_SHORT_ADD
    coord.c, 80
Button/main.c
    button_init, 20
    button_pressed, 20
    IO_BUT_AD, 20
    IO_LED_AD, 20
    ISR, 21
    LED, 24
    led_init, 22
    led_set, 22
    main, 23
    pIO_BUT, 24
    pIO_LED, 24
    timer_init, 23
button_init
    Button/main.c, 20
button_pressed
    Button/main.c, 20

c_status
    coord.c, 96
CHANNELMASK
    device.c, 100
coord.c
    application_init, 83
    association_table, 96
    association_table_init, 84
    BEACON_ORDER, 80
    BROADCAST_SHORT_ADD, 80
    c_status, 96
    coord_state_t, 83
    data_buffer, 96
    EMPTY_LONG_ADDRESS, 81
    get_empty_association_slot, 84
    INIT_DONE, 83
    mac_active_scan, 85
    mac_do_reset, 85
    mac_register_device, 86
    mac_send_data, 87
    mac_set_assoc_permit, 87
    mac_set_short_addr, 87
    mac_start_pan, 88
    main, 88
    MAX_ENTRIES, 81
    NO_ENTRY, 81
    PANCOORD_SCAN_-CHANNELS, 81
    PANCOORD_SHORT_ADD, 81
    PANID, 81
    PEND_ASSOC_PERMIT, 83
    PEND_INITIAL_SCAN, 83
    PEND_RESET, 83
    PEND_SET_SHORT_ADDR, 83
    PEND_START, 83
    pUSB_Fifo, 96
    RF_CHANNEL, 81
    RUN, 83
    SCANDURATION, 81
    search_association_entry, 89

SET_STATE, 82
show_device_list, 89
SUPERFRAME_ORDER, 82
USB_BIT_RXF, 82
USB_BIT_TXE, 82
USB_FIFO_AD, 82
usb_getc, 90
usb_init, 91
usb_keypressed, 91
usb_putc, 91
USB_STATE, 82
usb_stream, 96
usr_mcps_data_ind, 92
usr_mlme_associate_ind, 92
usr_mlme_comm_status_ind, 93
usr_mlme_reset_conf, 93
usr_mlme_scan_conf, 94
usr_mlme_set_conf, 94
usr_mlme_start_conf, 95
VT100_CLR_SCREEN, 82
VT100_CUR_HOME, 82
coord_address
device_status_t, 14
coord_address_mode
device_status_t, 14
coord_state_t
coord.c, 83
coord_status_t, 13
handle, 13
led_value, 13
state, 13

d_status
device.c, 108
data_buffer
coord.c, 96
delay_ms
LEDLoop/main.c, 26
device.c
ALL_HIGH_BAND_CHANNELS, 100
application_init, 101
CHANNELMASK, 100
d_status, 108
device_state_t, 100
INIT_DONE, 100
IO_THERM_AD, 100
mac_associate, 101
mac_do_reset, 101
mac_scan, 101
main, 102
PEND_ASSOCIATE, 100
PEND_RESET, 100
PEND_SCAN, 100
PEND_SET_SHORT_ADDR, 100
PEND_START, 100
pIO_THERM, 109
pwr_read_adc, 102
RF_CHANNEL, 100
RUN, 100
SCAN_DURATION, 100
SET_STATE, 100
switch_task, 103
temp_get_degrcelc, 104
temp_init, 105
usr_mcps_data_ind, 106
usr_mlme_associate_conf, 106
usr_mlme_reset_conf, 107
usr_mlme_scan_conf, 108
device_short_address
device_status_t, 14
device_state_t
device.c, 100
device_status_t, 14
coord_address, 14
coord_address_mode, 14
device_short_address, 14
led, 14
logical_channel, 15
msdu_handle, 15
pan_id, 14
state, 15
switch_pressed, 14
Doc/Doxygen/SensTermBoard.dox, 17

ee24xx_read_bytes
TWIEeprom/main.c, 48
ee24xx_write_bytes
TWIEeprom/main.c, 51
ee24xx_write_page
TWIEeprom/main.c, 52
EE_WRITE
TWIEeprom/main.c, 47
EMPTY_LONG_ADDRESS
coord.c, 81
error
TWIEeprom/main.c, 55

Examples/Button/main.c, 18
Examples/LEDLoop/main.c, 25
Examples/LEDTimer/main.c, 29
Examples/Temperature/main.c, 34
Examples/TWIEeprom/main.c, 44
Examples/Uart/main.c, 61
Examples/Usb/main.c, 66
Examples/XRAM/main.c, 71
Examples/Z-LinkWireless/coord.c, 77
Examples/Z-LinkWireless/device.c, 97

get_empty_association_slot
    coord.c, 84

handle
    coord_status_t, 13

INIT_DONE
    coord.c, 83
    device.c, 100
IO_BUT_AD
    Button/main.c, 20
IO_LED_AD
    Button/main.c, 20
    LEDLoop/main.c, 26
    LEDTimer/main.c, 30
IO_THERM_AD
    device.c, 100
    Temperature/main.c, 36
ioinit
    TWIEeprom/main.c, 55
ISR
    Button/main.c, 21
    LEDTimer/main.c, 31

LED
    Button/main.c, 24
    LEDLoop/main.c, 28
    LEDTimer/main.c, 33
led
    device_status_t, 14
led_init
    Button/main.c, 22
    LEDLoop/main.c, 27
    LEDTimer/main.c, 31
led_set
    Button/main.c, 22
    LEDLoop/main.c, 27
    LEDTimer/main.c, 31
led_value
    coord_status_t, 13
LEDLoop/main.c
    delay_ms, 26
    IO_LED_AD, 26
    LED, 28
    led_init, 27
    led_set, 27
    main, 28
    pIO_LED, 28
LEDTimer/main.c
    IO_LED_AD, 30
    ISR, 31
    LED, 33
    led_init, 31
    led_set, 31
    main, 32
    pIO_LED, 33
    timer_init, 32
logical_channel
    device_status_t, 15
long_addr
    association_entry_t, 11

mac_active_scan
    coord.c, 85
mac_associate
    device.c, 101
mac_do_reset
    coord.c, 85
    device.c, 101
mac_register_device
    coord.c, 86
mac_scan
    device.c, 101
mac_send_data
    coord.c, 87
mac_set_assoc_permit
    coord.c, 87
mac_set_short_addr
    coord.c, 87
mac_start_pan
    coord.c, 88
main
    Button/main.c, 23
    coord.c, 88
    device.c, 102
    LEDLoop/main.c, 28
    LEDTimer/main.c, 32
    Temperature/main.c, 37
    TWIEeprom/main.c, 56

Uart/main.c, 63
Usb/main.c, 68
XRAM/main.c, 73
MAX_ENTRIES
coord.c, 81
MAX_ITER
TWIEeprom/main.c, 47
msdu_handle
device_status_t, 15

NO_ENTRY
coord.c, 81

PAGE_SIZE
TWIEeprom/main.c, 47
pan_id
device_status_t, 14
PANCOORD_SCAN_CHANNELS
coord.c, 81
PANCOORD_SHORT_ADD
coord.c, 81
PANID
coord.c, 81
PEND_ASSOC_PERMIT
coord.c, 83
PEND_ASSOCIATE
device.c, 100
PEND_INITIAL_SCAN
coord.c, 83
PEND_RESET
coord.c, 83
device.c, 100
PEND_SCAN
device.c, 100
PEND_SET_SHORT_ADDR
coord.c, 83
device.c, 100
PEND_START
coord.c, 83
device.c, 100
pIO_BUT
Button/main.c, 24
pIO_LED
Button/main.c, 24
LEDLoop/main.c, 28
LEDTimer/main.c, 33
pIO_THERM
device.c, 109
Temperature/main.c, 43
pUSB_Fifo
coord.c, 96
Temperature/main.c, 43
TWIEeprom/main.c, 59
Usb/main.c, 70
XRAM/main.c, 76
pwr_read_adc
device.c, 102
Temperature/main.c, 38
pXRAM
XRAM/main.c, 76

RF_CHANNEL
coord.c, 81
device.c, 100
RUN
coord.c, 83
device.c, 100

SCAN_DURATION
device.c, 100
SCANDURATION
coord.c, 81
search_association_entry
coord.c, 89
SET_STATE
coord.c, 82
device.c, 100
show_device_list
coord.c, 89
state
coord_status_t, 13
device_status_t, 15
SUPERFRAME_ORDER
coord.c, 82
switch_pressed
device_status_t, 14
switch_task
device.c, 103

temp_get_degrcelc
device.c, 104
Temperature/main.c, 38
temp_init
device.c, 105
Temperature/main.c, 40
temperature
association_entry_t, 12
Temperature/main.c
IO_THERM_AD, 36
main, 37

pIO_THERM, 43
pUSB_Fifo, 43
pwr_read_adc, 38
temp_get_degrcelc, 38
temp_init, 40
USB_BIT_RXF, 36
USB_BIT_TXE, 36
USB_FIFO_AD, 36
usb_getc, 41
usb_init, 41
usb_keypressed, 41
usb_putc, 42
USB_STATE, 36
usb_stream, 43
VT100_CLR_LINE, 37
timer_init
Button/main.c, 23
LEDTimer/main.c, 32
TWI_SLA_24CXX
TWIEeprom/main.c, 47
TWIEeprom/main.c
ee24xx_read_bytes, 48
ee24xx_write_bytes, 51
ee24xx_write_page, 52
EE_WRITE, 47
error, 55
ioinit, 55
main, 56
MAX_ITER, 47
PAGE_SIZE, 47
pUSB_Fifo, 59
TWI_SLA_24CXX, 47
twst, 59
USB_BIT_RXF, 47
USB_BIT_TXE, 47
USB_FIFO_AD, 48
usb_getc, 57
usb_init, 58
usb_keypressed, 58
usb_putc, 58
USB_STATE, 48
usb_stream, 59
VT100_CLR_LINE, 48
twst
TWIEeprom/main.c, 59

Uart/main.c
BAUDRATE, 63
main, 63
uart_getc, 63
uart_init, 64
uart_keypressed, 64
uart_putc, 65
uart_stream, 65
uart_getc
Uart/main.c, 63
uart_init
Uart/main.c, 64
uart_keypressed
Uart/main.c, 64
uart_putc
Uart/main.c, 65
uart_stream
Uart/main.c, 65
Usb/main.c
main, 68
pUSB_Fifo, 70
USB_BIT_RXF, 68
USB_BIT_TXE, 68
USB_FIFO_AD, 68
usb_getc, 69
usb_init, 69
usb_keypressed, 69
usb_putc, 70
USB_STATE, 68
usb_stream, 70
USB_BIT_RXF
coord.c, 82
Temperature/main.c, 36
TWIEeprom/main.c, 47
Usb/main.c, 68
XRAM/main.c, 73
USB_BIT_TXE
coord.c, 82
Temperature/main.c, 36
TWIEeprom/main.c, 47
Usb/main.c, 68
XRAM/main.c, 73
USB_FIFO_AD
coord.c, 82
Temperature/main.c, 36
TWIEeprom/main.c, 48
Usb/main.c, 68
XRAM/main.c, 73
usb_getc
coord.c, 90
Temperature/main.c, 41
TWIEeprom/main.c, 57
Usb/main.c, 69
XRAM/main.c, 74

usb_init
coord.c, 91
Temperature/main.c, 41
TWIEeprom/main.c, 58
Usb/main.c, 69
XRAM/main.c, 74
usb_keypressed
coord.c, 91
Temperature/main.c, 41
TWIEeprom/main.c, 58
Usb/main.c, 69
XRAM/main.c, 75
usb_putc
coord.c, 91
Temperature/main.c, 42
TWIEeprom/main.c, 58
Usb/main.c, 70
XRAM/main.c, 75
USB_STATE
coord.c, 82
Temperature/main.c, 36
TWIEeprom/main.c, 48
Usb/main.c, 68
XRAM/main.c, 73
usb_stream
coord.c, 96
Temperature/main.c, 43
TWIEeprom/main.c, 59
Usb/main.c, 70
XRAM/main.c, 76
usr_mcps_data_ind
coord.c, 92
device.c, 106
usr_mlme_associate_conf
device.c, 106
usr_mlme_associate_ind
coord.c, 92
usr_mlme_comm_status_ind
coord.c, 93
usr_mlme_reset_conf
coord.c, 93
device.c, 107
usr_mlme_scan_conf
coord.c, 94
device.c, 108
usr_mlme_set_conf
coord.c, 94
usr_mlme_start_conf
coord.c, 95
VT100_CLR_LINE
Temperature/main.c, 37
TWIEeprom/main.c, 48
VT100_CLR_SCREEN
coord.c, 82
VT100_CUR_HOME
coord.c, 82

XRAM/main.c
main, 73
pUSB_Fifo, 76
pXRAM, 76
USB_BIT_RXF, 73
USB_BIT_TXE, 73
USB_FIFO_AD, 73
usb_getc, 74
usb_init, 74
usb_keypressed, 75
usb_putc, 75
USB_STATE, 73
usb_stream, 76
XRAM_MEM_AD, 73
XRAM_MEM_AD
XRAM/main.c, 73