

OpenOsci Reference Manual

0.01

Generated by Doxygen 1.4.6

Fri Aug 11 21:47:30 2006

Contents

1	OpenOsci Main Page	1
1.1	Thanks!	1
1.2	License	1
1.3	Documentation	1
1.4	Installation	2
1.5	FuseBits	2
1.6	Contact	2
2	OpenOsci Hierarchical Index	3
2.1	OpenOsci Class Hierarchy	3
3	OpenOsci Data Structure Index	5
3.1	OpenOsci Data Structures	5
4	OpenOsci File Index	7
4.1	OpenOsci File List	7
5	OpenOsci Page Index	9
5.1	OpenOsci Related Pages	9
6	OpenOsci Data Structure Documentation	11
6.1	menu_main_struct Struct Reference	11
6.2	menu_prop_main_struct Struct Reference	12
6.3	menu_prop_struct Struct Reference	13
7	OpenOsci File Documentation	15
7.1	adc.c File Reference	15
7.2	adc.h File Reference	20
7.3	control.c File Reference	24
7.4	control.h File Reference	26

7.5	display.c File Reference	28
7.6	display.h File Reference	36
7.7	input.c File Reference	44
7.8	input.h File Reference	47
7.9	main.c File Reference	49
7.10	main.h File Reference	51
7.11	menu.c File Reference	52
7.12	menu.h File Reference	56
7.13	uart.c File Reference	61
7.14	uart.h File Reference	64
7.15	ustimer.c File Reference	68
7.16	ustimer.h File Reference	71
8	OpenOsci Page Documentation	73
8.1	Todo List	73
8.2	Bug List	74

Chapter 1

OpenOsci Main Page

1.1 Thanks!

Thanks to all the people who write open source code. This project is based on a couple of other projects (barely did anything myself):

- Hagen Reddmann's and Christian Kranz's glcd lib for the Siemens S65 display
- Peter Fleury's UART lib
- of course: the avr-libc project

Hope I did not forget anyone.

1.2 License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

1.3 Documentation

An up-to-date HTML and PDF version of the documentation is located at <http://www.svenkreiss.com/private/openosci.php>.

1.4 Installation

The easiest way to install the code should be to program the fuse bits, download the *.hex-file and write it directly to the controller.

1.5 FuseBits

Be very careful with these commands. It is absolutely necessary that you know in detail what each of these does and whether you can apply them to your system.

read ext fuse bits:

```
avrdude -c avr910 -p m128 -P /dev/ttyUSB0 -U efuse:r:-:r | xxd
```

write ext fuse (m103C off, watchdog off):

```
avrdude -c avr910 -p m128 -P /dev/ttyUSB0 -U efuse:w:0xFF:m
```

write high fuse (disable JTAG, CKOPT to 0 for high freq cryst > 8MHz):

```
avrdude -c avr910 -p m128 -P /dev/ttyUSB0 -U hfuse:w:0xC9:m
```

write low fuse (switch to external chrystal osc):

```
avrdude -c avr910 -p m128 -P /dev/ttyUSB0 -U lfuse:w:0xEF:m
```

1.6 Contact

Sven Kreiss

eMail: sk@svenkreiss.com – sk at svenkreiss dot com

web: <http://www.svenkreiss.com/> – www dot svenkreiss dot com

Chapter 2

OpenOsci Hierarchical Index

2.1 OpenOsci Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

menu_main_struct	11
menu_prop_main_struct	12
menu_prop_struct	13

Chapter 3

OpenOsci Data Structure Index

3.1 OpenOsci Data Structures

Here are the data structures with brief descriptions:

menu_main_struct (Main menu – Mode)	11
menu_prop_main_struct (Container for properties)	12
menu_prop_struct (Properties)	13

Chapter 4

OpenOsci File Index

4.1 OpenOsci File List

Here is a list of all files with brief descriptions:

adc.c	15
adc.h (ADC)	20
control.c	24
control.h (Control)	26
display.c	28
display.h (Display)	36
input.c	44
input.h (Input)	47
main.c	49
main.h (Main)	51
menu.c	52
menu.h (Menu)	56
uart.c	61
uart.h (UART)	64
ustimer.c	68
ustimer.h (Micro-second(us) timer)	71

Chapter 5

OpenOsci Page Index

5.1 OpenOsci Related Pages

Here is a list of all related documentation pages:

Todo List	73
Bug List	74

Chapter 6

OpenOsci Data Structure Documentation

6.1 menu_main_struct Struct Reference

Main menu – Mode.

```
#include <menu.h>
```

Data Fields

- void(* [disp_func](#))(void)
- void(* [control_func](#))(void)
- char * [name](#)

6.1.1 Detailed Description

Main menu – Mode.

Structur which holds the "interface"-information for the main menu entries.

6.1.2 Field Documentation

6.1.2.1 void(* [menu_main_struct::control_func](#))(void)

6.1.2.2 void(* [menu_main_struct::disp_func](#))(void)

6.1.2.3 char* [menu_main_struct::name](#)

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

- [menu.h](#)

6.2 menu_prop_main_struct Struct Reference

container for properties

```
#include <menu.h>
```

Data Fields

- [int8_t nr_props](#)
- [int8_t prop_now](#)
- [menu_prop_struct properties](#) [10]

6.2.1 Detailed Description

container for properties

6.2.2 Field Documentation

6.2.2.1 [int8_t menu_prop_main_struct::nr_props](#)

6.2.2.2 [int8_t menu_prop_main_struct::prop_now](#)

6.2.2.3 [menu_prop_struct menu_prop_main_struct::properties](#)[10]

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

- [menu.h](#)

6.3 menu_prop_struct Struct Reference

Properties.

```
#include <menu.h>
```

Data Fields

- void(* [set_value](#))(int8_t value)
- int8_t [current_value](#)
- int8_t [nr_values](#)
- char * [prop_name](#)
- char * [value_name](#) [11]

6.3.1 Detailed Description

Properties.

"Interface"-information for the property-list.

Note: This is a single property. The structure containing all the properties for one menu is [menu_prop_main_struct](#). Names are chosen badly here.

6.3.2 Field Documentation

6.3.2.1 [int8_t menu_prop_struct::current_value](#)

6.3.2.2 [int8_t menu_prop_struct::nr_values](#)

6.3.2.3 [char* menu_prop_struct::prop_name](#)

6.3.2.4 [void\(* menu_prop_struct::set_value\)\(int8_t value\)](#)

6.3.2.5 [char* menu_prop_struct::value_name\[11\]](#)

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

- [menu.h](#)

Chapter 7

OpenOsci File Documentation

7.1 adc.c File Reference

```
#include "adc.h"
```

Functions

- void [adc_select_channel](#) (uint8_t channel)
select "channel"
- void [adc_LED](#) (void)
handels the LED output
- void [adc_off](#) (void)
switches the ADC off
- void [adc_set_nr_channels](#) (int8_t nr)
sets the nr of active channels
- void [adc_next_channel](#) (void)
switches to the next channel
- void [adc_set_presc](#) (int8_t presc)
sets the prescaler
- void [adc_single_channel](#) (void)
- void [adc_multi_channels](#) (void)
- void [adc_init](#) (uint8_t channel)
initialise ADC with "channel"
- void [adc_stop](#) (void)
stops the ADC
- int8_t [adc_stopped](#) (void)
checks, whether the ADC has stopped

Variables

- volatile uint8_t `adc_prescaler` = 1
the current prescaler
- volatile uint16_t `adc_count` = 0
- volatile int8_t `adc_stop_flag` = 1
still needed ???
- volatile uint32_t `adc_starttime` = 0
- volatile double `adc_duration` = 0
- volatile uint8_t `adc_channels` = 1
nr of active channels
- volatile uint8_t `adc_current_channel` = 0

7.1.1 Function Documentation

7.1.1.1 void `adc_init` (uint8_t *channel*)

initialise ADC with "channel"

```

104                                     {
105     adc_count = 0;
106     adc_stop_flag = 0;
107
108     adc_select_channel(channel);    //set the channel
109     adc_current_channel = channel;  //save the number of the current channel
110     adc_LED();
111
112     ADCSRA = adc_prescaler & 7;    //& 7: only assign first three bits
113     //AD Enable, AD Start Conversion, AD Free Running, AD Interrupt Enable
114     adc_starttime = us_time_get();
115     ADCSRA |= (1<<ADEN) | (1<<ADSC) | (1<<ADFR); //interrupt method: | (1<<ADIE);
116
117     if(adc_channels == 1) adc_single_channel();
118     else adc_multi_channels();
119
120     //stop adc again
121     adc_stop();
122 }
```

7.1.1.2 void `adc_LED` (void)

handels the LED output

```

39     {
40     if(adc_channels == 1) PORTA = (PORTA & 15) + (8 << 4); //& 15: assign only first 4 bits
41     if(adc_channels == 2) PORTA = (PORTA & 15) + (12 << 4); //& 15: assign only first 4 bits
42     if(adc_channels == 3) PORTA = (PORTA & 15) + (14 << 4); //& 15: assign only first 4 bits
43     if(adc_channels == 4) PORTA = (PORTA & 15) + (15 << 4); //& 15: assign only first 4 bits
44 }
```

7.1.1.3 void adc_multi_channels (void)

```

85         {
86     //fast measurement
87     loop_until_bit_is_set (ADCSRA, ADIF);
88     ADCSRA |= (1<<ADIF);
89     adc[0] = ADCH;
90     for(uint16_t x=0; x < ADC_BUF_SIZE; x++){
91         adc_next_channel();
92         loop_until_bit_is_set (ADCSRA, ADIF);
93         adc[x] = ADCH;
94         ADCSRA |= (1<<ADIF);
95
96         loop_until_bit_is_set (ADCSRA, ADIF);
97         ADCSRA |= (1<<ADIF);
98         loop_until_bit_is_set (ADCSRA, ADIF);
99         ADCSRA |= (1<<ADIF);
100     }
101 }

```

7.1.1.4 void adc_next_channel (void)

switches to the next channel

```

57         {
58     uint8_t new = adc_current_channel;
59
60     new++;
61     if(new > adc_channels) new = 1;
62     adc_select_channel (new);
63 }

```

7.1.1.5 void adc_off (void)

switches the ADC off

```

47         {
48     PORTA = PORTA & 15; //sets the upper 4 bits to zero
49
50     ADCSRA = 0;
51 }

```

7.1.1.6 void adc_select_channel (uint8_t channel)

select "channel"

```

29         {
30     if(channel == adc_current_channel) return;
31     adc_current_channel = channel;
32
33     //Input Channel selection; obersten beiden bits wÃd'hlen externe Referenz, wenn 0
34     //=> ADMUX entspricht Channel
35     ADMUX = (channel-1) + (1<<ADLAR) + (1<<REFS0);
36     //ADMUX |= (1<<ADLAR); //left-align bits to allow to read the 8bit-value from ADCH
37 }

```

7.1.1.7 void adc_set_nr_channels (int8_t nr)

sets the nr of active channels

```
53                                     {
54     adc_channels = nr;
55 }
```

7.1.1.8 void adc_set_presc (int8_t presc)

sets the prescaler

```
66                                     {
67     if(presc <= 7 && presc >= 1) adc_prescaler = presc;
68 }
```

7.1.1.9 void adc_single_channel (void)

```
72                                     {
73     //fast measurement
74     loop_until_bit_is_set (ADCSRA, ADIF);
75     ADCSRA |= (1<<ADIF);
76     adc[0] = ADCH;
77     for(uint16_t x=0; x < ADC_BUF_SIZE; x++){
78         loop_until_bit_is_set (ADCSRA, ADIF);
79         adc[x] = ADCH;
80     //     adc_next_channel();
81         ADCSRA |= (1<<ADIF);
82     }
83 }
```

7.1.1.10 void adc_stop (void)

stops the ADC

switches off the free running mode.

Todo

still needed?

```
127                                     {
128     ADCSRA &= ~(1<<ADSC) & ~(1<<ADFR) & ~(1<<ADIE);
129     adc_stop_flag = 1;
130
131     adc_duration = us_time_get_difference_d(adc_starttime) / ((double)ADC_BUF_SIZE / (double)adc_chann
132 }
```

7.1.1.11 int8_t adc_stopped (void)

checks, whether the ADC has stopped

Todo

still needed?

```
135     {  
136     return adc_stop_flag;  
137 }
```

7.1.2 Variable Documentation

7.1.2.1 volatile uint8_t `adc_channels` = 1

nr of active channels

7.1.2.2 volatile uint16_t `adc_count` = 0

7.1.2.3 volatile uint8_t `adc_current_channel` = 0

7.1.2.4 volatile double `adc_duration` = 0

7.1.2.5 volatile uint8_t `adc_prescaler` = 1

the current prescaler

7.1.2.6 volatile uint32_t `adc_starttime` = 0

7.1.2.7 volatile int8_t `adc_stop_flag` = 1

still needed ???

7.2 adc.h File Reference

ADC.

```
#include "main.h"
```

Defines

- #define `ADC_BUF_SIZE` 1700
Size of the input-buffer.

Functions

- void `adc_init` (uint8_t channel)
initialise ADC with "channel"
- void `adc_select_channel` (uint8_t channel)
select "channel"
- void `adc_LED` (void)
handels the LED output
- void `adc_off` (void)
switches the ADC off
- void `adc_set_nr_channels` (int8_t nr)
sets the nr of active channels
- void `adc_set_presc` (int8_t presc)
sets the prescaler
- void `adc_next_channel` (void)
switches to the next channel
- int8_t `adc_stopped` (void)
checks, whether the ADC has stopped
- void `adc_stop` (void)
stops the ADC

Variables

- volatile uint8_t `adc` [ADC_BUF_SIZE]
input-buffer
- volatile uint8_t `adc_channels`
nr of active channels

- volatile double `adc_duration`
time for one sample point
- volatile uint8_t `adc_prescaler`
the current prescaler

7.2.1 Detailed Description

ADC.

Data acquisition.

30 March 2006

Sven Kreiss

7.2.2 Define Documentation

7.2.2.1 #define ADC_BUF_SIZE 1700

Size of the input-buffer.

7.2.3 Function Documentation

7.2.3.1 void adc_init (uint8_t channel)

initialise ADC with "channel"

```
104                                     {
105     adc_count = 0;
106     adc_stop_flag = 0;
107
108     adc_select_channel(channel); //set the channel
109     adc_current_channel = channel; //save the number of the current channel
110     adc_LED();
111
112     ADCSRA = adc_prescaler & 7; // & 7: only assign first three bits
113     //AD Enable, AD Start Conversion, AD Free Running, AD Interrupt Enable
114     adc_starttime = us_time_get();
115     ADCSRA |= (1<<ADEN) | (1<<ADSC) | (1<<ADFR); //interrupt method: | (1<<ADIE);
116
117     if(adc_channels == 1) adc_single_channel();
118     else adc_multi_channels();
119
120     //stop adc again
121     adc_stop();
122 }
```

7.2.3.2 void adc_LED (void)

handles the LED output

```

39         {
40     if (adc_channels == 1) PORTA = (PORTA & 15) + (8 << 4); //& 15: assign only first 4 bits
41     if (adc_channels == 2) PORTA = (PORTA & 15) + (12 << 4); //& 15: assign only first 4 bits
42     if (adc_channels == 3) PORTA = (PORTA & 15) + (14 << 4); //& 15: assign only first 4 bits
43     if (adc_channels == 4) PORTA = (PORTA & 15) + (15 << 4); //& 15: assign only first 4 bits
44 }

```

7.2.3.3 void adc_next_channel (void)

switches to the next channel

```

57         {
58     uint8_t new = adc_current_channel;
59
60     new++;
61     if (new > adc_channels) new = 1;
62     adc_select_channel (new);
63 }

```

7.2.3.4 void adc_off (void)

switches the ADC off

```

47         {
48     PORTA = PORTA & 15; //sets the upper 4 bits to zero
49
50     ADCSRA = 0;
51 }

```

7.2.3.5 void adc_select_channel (uint8_t channel)

select "channel"

```

29         {
30     if (channel == adc_current_channel) return;
31     adc_current_channel = channel;
32
33     //Input Channel selection; obersten beiden bits wÃd'hlen externe Referenz, wenn 0
34     //=> ADMUX entspricht Channel
35     ADMUX = (channel-1) + (1<<ADLAR) + (1<<REFS0);
36     //ADMUX |= (1<<ADLAR); //left-align bits to allow to read the 8bit-value from ADCH
37 }

```

7.2.3.6 void adc_set_nr_channels (int8_t nr)

sets the nr of active channels

```

53         {
54     adc_channels = nr;
55 }

```

7.2.3.7 void adc_set_presc (int8_t presc)

sets the prescaler

```
66     {
67     if(presc <= 7 && presc >= 1) adc_prescaler = presc;
68 }
```

7.2.3.8 void adc_stop (void)

stops the ADC

switches off the free running mode.

Todo

still needed?

```
127     {
128     ADCSRA &= ~(1<<ADSC) & ~(1<<ADFR) & ~(1<<ADIE);
129     adc_stop_flag = 1;
130
131     adc_duration = us_time_get_difference_d(adc_starttime) / ((double)ADC_BUF_SIZE / (double)adc_chanr
132 }
```

7.2.3.9 int8_t adc_stopped (void)

checks, whether the ADC has stopped

Todo

still needed?

```
135     {
136     return adc_stop_flag;
137 }
```

7.2.4 Variable Documentation

7.2.4.1 volatile uint8_t adc[ADC_BUF_SIZE]

input-buffer

7.2.4.2 volatile uint8_t adc_channels

nr of active channels

7.2.4.3 volatile double adc_duration

time for one sample point

7.2.4.4 volatile uint8_t adc_prescaler

the current prescaler

7.3 control.c File Reference

```
#include "control.h"
```

Functions

- void [control_init](#) (void)
initialise control
- void [control_refresh](#) (void)
refresh control
- void [control_graph](#) (void)
the control function for mode "graph"
- void [control_term](#) (void)
the control function for mode "term"

7.3.1 Function Documentation

7.3.1.1 void control_graph (void)

the control function for mode "graph"

```
38                                     {  
39     //usart_off();  
40 }
```

7.3.1.2 void control_init (void)

initialise control

```
30                                     {  
31 }
```

7.3.1.3 void control_refresh (void)

refresh control

```
34                                     {  
35     menu_mains[menu_now-1].control_func();  
36 }
```

7.3.1.4 void control_term (void)

the control function for mode "term"

```
42                                     {  
43     //usart_init();  
44 }
```

7.4 control.h File Reference

control

```
#include "main.h"
```

Functions

- void [control_init](#) (void)
initialise control
- void [control_refresh](#) (void)
refresh control
- void [control_play](#) (void)
the control function for mode "play"
- void [control_graph](#) (void)
the control function for mode "graph"
- void [control_term](#) (void)
the control function for mode "term"

7.4.1 Detailed Description

control

Control-backend.

Todo

!!! Need to think about this concept again !!! Only [control_refresh\(\)](#) is not empty :-S .

13 December 2005

Sven Kreiss

7.4.2 Function Documentation

7.4.2.1 void control_graph (void)

the control function for mode "graph"

```
38         {  
39     //usart_off();  
40 }
```

7.4.2.2 void control_init (void)

initialise control

```
30                                     {  
31 }
```

7.4.2.3 void control_play (void)

the control function for mode "play"

7.4.2.4 void control_refresh (void)

refresh control

```
34                                     {  
35     menu_mains[menu_now-1].control_func();  
36 }
```

7.4.2.5 void control_term (void)

the control function for mode "term"

```
42                                     {  
43     //usart_init();  
44 }
```

7.5 display.c File Reference

```
#include "display.h"  
#include "avr/pgmspace.h"
```

Defines

- #define [BkColor](#) BLACK
- #define [FgColor](#) YELLOW
- #define [ShColor](#) BLUE
- #define [SPACELEFT](#) 6
- #define [DISP_PROP_W](#) 33
- #define [GR_W](#) 20
- #define [GR_H](#) 20
- #define [GR_WW](#) 128
- #define [GR_HH](#) 144
- #define [GR_X](#) 2
- #define [GR_Y](#) 15
- #define [INFO_X](#) GR_X+GR_WW-55
- #define [INFO_Y](#) GR_Y+GR_HH-25

Functions

- void [disp_init](#) (void)
initialises display
- void [disp_off](#) (void)
switches the display off
- void [disp_refresh](#) (void)
refreshes the display
- void [disp_menu](#) (void)
displays the menu
- void [disp_prop](#) (void)
displays the properties
- void [disp_clean](#) (void)
clean the display
- void [disp_debugging](#) (void)
displays a debugging screen
- void [disp_terminal](#) (void)
displays a terminal
- void [disp_graph](#) (void)

displays the osci graph

- void `disp_drawGrid` (volatile uint8_t toDraw[], uint8_t xaxis, uint8_t yaxis)
draws the grid and the data in the argument

Variables

- volatile uint8_t `min` [4] = {255,255,255,255}
- volatile uint8_t `max` [4] = {0,0,0,0}
- volatile uint8_t `mid` [4] = {0,0,0,0}
- volatile double `freq` [4] = {0,0,0,0}

7.5.1 Define Documentation

7.5.1.1 #define BkColor BLACK

7.5.1.2 #define DISP_PROP_W 33

7.5.1.3 #define FgColor YELLOW

7.5.1.4 #define GR_H 20

7.5.1.5 #define GR_HH 144

7.5.1.6 #define GR_W 20

7.5.1.7 #define GR_WW 128

7.5.1.8 #define GR_X 2

7.5.1.9 #define GR_Y 15

7.5.1.10 #define INFO_X GR_X+GR_WW-55

7.5.1.11 #define INFO_Y GR_Y+GR_HH-25

7.5.1.12 #define ShColor BLUE

7.5.1.13 #define SPACELEFT 6

7.5.2 Function Documentation

7.5.2.1 void disp_clean (void)

clean the display

```
165         {
166     glcdSetAddr(0,0, 131, 175); // set RAM access pointer of display
167     //glcdSetBkColor(BkColor);
168
169     //width and height also defined in header
```

```

170     for(uint8_t x = 0; x < 132; x++){
171         for(uint8_t y = 0; y < 176; y++){
172             glcdPutPixel(BkColor);
173         }
174     }
175 }

```

7.5.2.2 void disp_debugging (void)

displays a debugging screen

```

187     {
188     glcdSetColors(WHITE,BkColor);
189
190     static uint16_t count = 0;
191     int8_t y = 30;
192     glcdMoveTo(20,y+=11); printf(" Count: %d \n",count++);
193     glcdMoveTo(20,y+=11); printf(" Menue: %d \n", (uint16_t)ADCSRA);
194     glcdMoveTo(20,y+=11); printf(" Presc: %d \n", (uint16_t)(ADCSRA & ADIF));
195     glcdMoveTo(20,y+=11); printf(" spi_control: %d \n",SPCR);
196     glcdMoveTo(20,y+=11); printf(" spi_status: %d \n",SPSR);
197 }

```

7.5.2.3 void disp_drawGrid (volatile uint8_t toDraw[], uint8_t xaxis, uint8_t yaxis)

draws the grid and the data in the argument

Todo

Change from grid to coordinate axes. Origin is the trigger point and zero Volt.

```

308     {
309     //draw grid
310     glcdSetColors(ShColor, BkColor);
311
312     glcdLine(GR_X+xaxis, GR_Y, GR_X+xaxis, GR_Y+GR_HH); //horizontal
313     glcdLine(GR_X, GR_Y+yaxis, GR_X+GR_WW, GR_Y+yaxis); //vertical
314
315     /* x and y need to be signed, because for small xaxis and yaxis, the start
316     * can be negative */
317     for(int16_t x = GR_X+xaxis+GR_W; x <= GR_X+GR_WW; x+=GR_W) glcdLine(x, GR_Y-3+yaxis, x, GR_Y+3+yaxis);
318     for(int16_t x = GR_X+xaxis-GR_W; x >= GR_X-GR_W; x-=GR_W) glcdLine(x, GR_Y-3+yaxis, x, GR_Y+3+yaxis);
319
320     for(int16_t y = GR_Y+yaxis+GR_H; y <= GR_Y+GR_HH; y+=GR_H) glcdLine(GR_X-3+xaxis, y, GR_X+3+xaxis, y);
321     for(int16_t y = GR_Y+yaxis-GR_H; y >= GR_Y-GR_H; y-=GR_H) glcdLine(GR_X-3+xaxis, y, GR_X+3+xaxis, y);
322
323
324     //replot data
325     //init
326     static uint8_t buffer[4][GR_HH];
327     uint8_t old_coord[4] = {buffer[0][0], buffer[1][0], buffer[2][0], buffer[3][0]};
328     static uint8_t old_adc_nr_channels = 1;
329     for(uint8_t ch=0; ch < adc_channels; ch++)
330         buffer[ch][0] = toDraw[ch] >> 1;
331     glcdSetBkColor(BkColor);
332     //start
333     for(uint8_t y = 1; y < GR_HH; y++){
334         //erase old lines
335         glcdSetFgColor(BkColor);
336         for(uint8_t ch=0; ch < old_adc_nr_channels; ch++){ //old_number!!!
337             glcdLine(GR_X + old_coord[ch], y-1 + GR_Y, GR_X + buffer[ch][y], y + GR_Y);

```

```

338         old_coord[ch] = buffer[ch][y];
339         buffer[ch][y] = toDraw[y*adc_channels+ch] >> 1;
340     }
341     //draw new lines
342     for(uint8_t ch=0; ch < adc_channels; ch++){
343         switch(ch){
344             case 0:
345                 glcdSetFgColor(WHITE);
346                 break;
347             case 1:
348                 glcdSetFgColor(FgColor);
349                 break;
350             case 2:
351                 glcdSetFgColor(GREEN);
352                 break;
353             case 3:
354                 glcdSetFgColor(RED);
355                 break;
356         }
357         glcdLine(GR_X + buffer[ch][y-1], y-1 + GR_Y, GR_X + buffer[ch][y], y + GR_Y);
358     }
359 }
360 old_adc_nr_channels = adc_channels;
361
362 //info box
363 #define INFO_X GR_X+GR_WW-55
364 #define INFO_Y GR_Y+GR_HH-25
365 glcdSetColors(FgColor, BkColor);
366 int y = INFO_Y - 9;
367 glcdMoveTo(INFO_X+2, y+=11); printf("td:");
368 glcdMoveTo(INFO_X+2, y+=11); printf("fq:");
369 y = INFO_Y - 9;
370 uint32_t time = adc_duration*GR_H;
371 if(time < 10000){
372     glcdMoveTo(INFO_X+15, y+=11); printf(" %4dus ", (int16_t)(time));
373 }else{
374     glcdMoveTo(INFO_X+15, y+=11); printf(" %4dms ", (int16_t)(time/1000));
375 }
376 if(freq[0] < 10000){
377     glcdMoveTo(INFO_X+15, y+=11); printf(" %4dHz ", (int16_t)(freq[0]));
378 }else{
379     glcdMoveTo(INFO_X+15, y+=11); printf(" %3dkHz ", (int16_t)(freq[0]/1000));
380 }
381 }

```

7.5.2.4 void disp_graph (void)

displays the osci graph

```

230     {
231     if(adc_stopped() == 0) return;
232
233     #define GR_W    20 //width of sub ... dash? unterteilung
234     #define GR_H    20 //height of ???
235     #define GR_WW   128 //width
236     #define GR_HH   144 //height
237     #define GR_X    2 //x-coord: upper-left
238     #define GR_Y    15 //y-coord: upper-left
239
240
241     //trigger
242     uint16_t trigger_shift = 72*adc_channels;
243     uint16_t GR_TRIG_BEFORE = ADC_BUF_SIZE - (GR_HH*adc_channels) + trigger_shift;
244

```

```

245     int8_t trigger_old = 0;
246     int8_t trigger_new = 0;
247     int8_t trigger_highest = 0;
248     uint16_t offset = 0;
249
250     for(uint16_t x = trigger_shift; x < GR_TRIG_BEFORE; x+=adc_channels){
251         trigger_old = adc[x] >> 1;
252         trigger_new = adc[x + 3*adc_channels] >> 1;
253         if((trigger_new-trigger_old) > trigger_highest){
254             offset = x - trigger_shift;
255             trigger_highest = trigger_new-trigger_old;
256         }
257     }
258
259     min[0] = 255; min[1] = 255; min[2] = 255; min[3] = 255;
260     max[0] = 0; max[1] = 0; max[2] = 0; max[3] = 0;
261     mid[0] = 0; mid[1] = 0; mid[2] = 0; mid[3] = 0;
262     freq[0] = 0; freq[1] = 0; freq[2] = 0; freq[3] = 0;
263     for(uint8_t ch = 0; ch < adc_channels; ch++){
264         for(uint16_t x = ch; x < ADC_BUF_SIZE; x+=adc_channels){
265             if(adc[x] < min[ch]) min[ch] = adc[x];
266             if(adc[x] > max[ch]) max[ch] = adc[x];
267         }
268     }
269     for(uint8_t x = 0; x < adc_channels; x++) mid[x] = (max[x]-min[x])/2 + min[x];
270
271
272     for(uint8_t ch = 0; ch < adc_channels; ch++){
273         uint16_t old_pos = 0;
274         int8_t under; //currently under mid? 0 for no, 1 for yes
275         int8_t old_under;
276         double period = 0.0;
277
278         if(adc[ch] < mid[ch]){ under = 0; old_under = 0; }
279         else { under = 1; old_under = 1; }
280
281         for(uint16_t x = ch; x < ADC_BUF_SIZE; x+=adc_channels){
282             if(adc[x] > mid[ch]) under = 0;
283             if(adc[x] < mid[ch]) under = 1;
284
285             if(old_under == 1 && under == 0){
286                 period = (double)(x - old_pos)*adc_duration /adc_channels/ 1000000.0;
287                 if(old_pos != 0 && period != 0.0){
288                     if(freq[ch] == 0.0) freq[ch] = 1.0 / period;
289                     else freq[ch] = 0.8*freq[ch] + 0.2/period;
290                 }
291                 old_pos = x;
292             }
293
294             old_under = under;
295         }
296     }
297
298
299
300     //draw in the display
301     disp_drawGrid(&adc[offset], 64, trigger_shift/adc_channels);
302
303     adc_init(1);
304 }

```

7.5.2.5 void disp_init (void)

initialises display

Calls the lib's routine to init the display. Displays the splash-screen. Sets the font. Sets the window for text-output.

```

26         {
27     //**** Hardware ****
28     #ifndef USE_AUTOINIT
29         glcdDisplayInit();
30     #endif
31
32     #ifdef DISP_SPLASH
33         disp_load_bitmap();
34     #else
35         disp_clean();
36     #endif
37
38     glcd_Flags.AutoLineFeed = 0;
39     glcdSelectFont(f8x11, 0); // font is stored in FLASH, thus no need for own read callback
40     fdevopen(glcdPut, NULL, 0);
41
42     //window for text-output
43     glcd_Window.X1 = 5;
44     glcd_Window.X2 = 110;
45     glcd_Window.Y1 = 16;
46     glcd_Window.Y2 = 162;
47 }

```

7.5.2.6 void disp_menu (void)

displays the menu

Refreshes the menu bar when the menu has changed.

```

75         {
76     static int8_t main_old = 0;
77
78     //**** left ***
79     if(menu_now != main_old){
80         main_old = menu_now;
81         disp_clean();
82
83
84         glcdFillRect(0,0,132,13,ShColor); //shadow color for menu bg
85         #define SPACELEFT 6
86         for(uint8_t x=0; x < MENU_ANZ_MAIN; x++){
87             if(menu_now == (x+1)){
88                 glcdFillRect(SPACELEFT+x*40,1, SPACELEFT+(x+1)*40,12, BkColor);
89                 glcdSetColors(FgColor, BkColor);
90             }else{
91                 glcdSetColors(FgColor, ShColor);
92             }
93             uint8_t w = glcdCharsWidth(menu_mains[x].name, 0);
94             glcdMoveTo(SPACELEFT+21+x*40 - w/2, 2);
95             printf(menu_mains[x].name);
96         }
97     }
98 }

```

7.5.2.7 void disp_off (void)

switches the display off

The original shutdown sequence for the display is not known. The current workaround is to activate the stand-by for the display.

```

51         {
52     bk_LED(1);
53     //for(uint32_t x=0; x < 5000000; x++) asm("nop");
54
55     //clean
56     glcdSetAddr(0,0, 131, 175); // set RAM access pointer of display
57     //width and height also defined in header
58     for(uint8_t x = 0; x < 132; x++){
59         for(uint8_t y = 0; y < 176; y++){
60             glcdPutPixel(WHITE);
61         }
62     }
63     for(uint32_t x=0; x < 500000; x++) asm("nop");
64     glcdDisplayOff();
65     glcdWait(100);
66 }

```

7.5.2.8 void disp_prop (void)

displays the properties

Refreshes the property bar if the menu, the property or the value of the property has changed.

```

102         {
103     static int8_t main_old = 0;
104     static int8_t prop_old = 0;
105     static int8_t value_old = 0;
106
107     if( menu_now != main_old ||
108         MENU_CURRENT_PROP_NR != prop_old ||
109         MENU_PROP_NOW.current_value != value_old){
110
111         main_old = menu_now;
112         prop_old = MENU_CURRENT_PROP_NR;
113         value_old = MENU_PROP_NOW.current_value;
114
115         #define DISP_PROP_W 33
116         glcdFillRect(0, 162, 132, 176, ShColor);
117         for(uint8_t x=0; x < 3; x++){
118             if(x == 1){
119                 glcdFillRect(SPACELEFT+x*DISP_PROP_W,163, SPACELEFT+(x+1)*DISP_PROP_W,174, BkColor);
120                 glcdSetColors(FgColor, BkColor);
121             }else{
122                 glcdSetColors(FgColor, ShColor);
123             }
124             int8_t current_value = value_old-1+x;
125             char *label;
126             if(current_value < 1)
127                 label = "=>";
128             else if(current_value > MENU_PROP_NOW.nr_values)
129                 label = "<=";
130             else
131                 label = MENU_PROP_NOW.value_name[current_value-1];
132             uint8_t w = glcdCharsWidth(label, 0);
133             glcdMoveTo(SPACELEFT+21+x*DISP_PROP_W - w, 164);
134             printf(label);
135
136             //prop-name
137             w = glcdCharsWidth(MENU_PROP_NOW.prop_name, 0);
138             glcdFillRect(132 - 2*SPACELEFT - w, 163, 130, 174, BkColor);
139             glcdSetColors(FgColor, BkColor);

```

```

140         glcdMoveTo(132 - SPACELEFT - w, 164);
141         printf(MENU_PROP_NOW.prop_name);
142     }
143 }
144 }

```

7.5.2.9 void disp_refresh (void)

refreshes the display

```

68         {
69     disp_menu();
70     menu_mains[menu_now-1].disp_func();
71     disp_prop();
72 }

```

7.5.2.10 void disp_terminal (void)

displays a terminal

The terminal for the serial interface.

Todo

Testing necessary!

```

201         {
202 //   if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162) glcdMoveTo(5,16);
203   if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162 || glcd_Cursor.X > 100) glcdMoveTo(5,16);
204   glcdSetColors(WHITE,BkColor);
205
206   uint16_t rx_data;
207   while(! ((rx_data=uart_getc()) & UART_NO_DATA)){
208       //glcdPut(rx_data & 255);
209       glcdDrawChar((uint8_t)rx_data);
210   }
211   while(! ((rx_data=uart1_getc()) & UART_NO_DATA)){
212       //glcdPut(rx_data & 255);
213       glcdDrawChar((uint8_t)rx_data);
214   }
215
216 //   if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162 || glcd_Cursor.X > 120) glcdMoveTo(5,16);
217   //printf("%c", (rx_data & 255));
218
219   //glcdWait(100);
220 }

```

7.5.3 Variable Documentation

7.5.3.1 volatile double **freq**[4] = {0,0,0,0}

7.5.3.2 volatile uint8_t **max**[4] = {0,0,0,0}

7.5.3.3 volatile uint8_t **mid**[4] = {0,0,0,0}

7.5.3.4 volatile uint8_t **min**[4] = {255,255,255,255}

Reads the input buffer from the ADC. Searches for the best trigger point. Calls [disp_drawGrid\(\)](#) to display the output.

7.6 display.h File Reference

Display.

```
#include <glcd.h>
#include "../libs/font/f9x14.h"
#include "../libs/font/f8x11.h"
#include <inttypes.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include "main.h"
```

Functions

- void `disp_init` (void)
initialises display
- void `disp_off` (void)
switches the display off
- void `disp_refresh` (void)
refreshes the display
- void `disp_menu` (void)
displays the menu
- void `disp_prop` (void)
displays the properties
- void `disp_load_bitmap` (void)
displays a bitmap (used by splash)
- void `disp_clean` (void)
clean the display
- void `disp_debugging` (void)
displays a debugging screen
- void `disp_graph` (void)
displays the osci graph
- void `disp_drawGrid` (volatile uint8_t toDraw[], uint8_t xaxis, uint8_t yaxis)
draws the grid and the data in the argument
- void `disp_terminal` (void)

displays a terminal

7.6.1 Detailed Description

Display.

All graphical output is handled here.

03 April 2006

Sven Kreiss

7.6.2 Function Documentation

7.6.2.1 void disp_clean (void)

clean the display

```
165         {
166     glcdSetAddr(0,0, 131, 175); // set RAM access pointer of display
167     //glcdSetBkColor(BkColor);
168
169     //width and height also defined in header
170     for(uint8_t x = 0; x < 132; x++){
171         for(uint8_t y = 0; y < 176; y++){
172             glcdPutPixel(BkColor);
173         }
174     }
175 }
```

7.6.2.2 void disp_debugging (void)

displays a debugging screen

```
187         {
188     glcdSetColors(WHITE,BkColor);
189
190     static uint16_t count = 0;
191     int8_t y = 30;
192     glcdMoveTo(20,y+=11); printf(" Count: %d \n",count++);
193     glcdMoveTo(20,y+=11); printf(" Menue: %d \n", (uint16_t)ADCSRA);
194     glcdMoveTo(20,y+=11); printf(" Presc: %d \n", (uint16_t)(ADCSRA & ADIF));
195     glcdMoveTo(20,y+=11); printf(" spi_control: %d \n",SPCR);
196     glcdMoveTo(20,y+=11); printf(" spi_status: %d \n",SPSR);
197 }
```

7.6.2.3 void disp_drawGrid (volatile uint8_t toDraw[], uint8_t xaxis, uint8_t yaxis)

draws the grid and the data in the argument

Todo

Change from grid to coordinate axes. Origin is the trigger point and zero Volt.

```

308
309 //draw grid
310 glcdSetColors(ShColor, BkColor);
311
312 glcdLine(GR_X+xaxis, GR_Y, GR_X+xaxis, GR_Y+GR_HH); //horizontal
313 glcdLine(GR_X, GR_Y+yaxis, GR_X+GR_WW, GR_Y+yaxis); //vertical
314
315 /* x and y need to be signed, because for small xaxis and yaxis, the start
316 * can be negative */
317 for(int16_t x = GR_X+xaxis+GR_W; x <= GR_X+GR_WW; x+=GR_W) glcdLine(x, GR_Y-3+yaxis, x, GR_Y+3+yaxis);
318 for(int16_t x = GR_X+xaxis-GR_W; x >= GR_X-GR_W; x-=GR_W) glcdLine(x, GR_Y-3+yaxis, x, GR_Y+3+yaxis);
319
320 for(int16_t y = GR_Y+yaxis+GR_H; y <= GR_Y+GR_HH; y+=GR_H) glcdLine(GR_X-3+xaxis, y, GR_X+3+xaxis, y);
321 for(int16_t y = GR_Y+yaxis-GR_H; y >= GR_Y-GR_H; y-=GR_H) glcdLine(GR_X-3+xaxis, y, GR_X+3+xaxis, y);
322
323
324 //replot data
325 //init
326 static uint8_t buffer[4][GR_HH];
327 uint8_t old_coord[4] = {buffer[0][0], buffer[1][0], buffer[2][0], buffer[3][0]};
328 static uint8_t old_adc_nr_channels = 1;
329 for(uint8_t ch=0; ch < adc_channels; ch++)
330     buffer[ch][0] = toDraw[ch] >> 1;
331 glcdSetBkColor(BkColor);
332 //start
333 for(uint8_t y = 1; y < GR_HH; y++){
334     //erase old lines
335     glcdSetFgColor(BkColor);
336     for(uint8_t ch=0; ch < old_adc_nr_channels; ch++){ //old_number!!!
337         glcdLine(GR_X + old_coord[ch], y-1 + GR_Y, GR_X + buffer[ch][y], y + GR_Y);
338         old_coord[ch] = buffer[ch][y];
339         buffer[ch][y] = toDraw[y*adc_channels+ch] >> 1;
340     }
341     //draw new lines
342     for(uint8_t ch=0; ch < adc_channels; ch++){
343         switch(ch){
344             case 0:
345                 glcdSetFgColor(WHITE);
346                 break;
347             case 1:
348                 glcdSetFgColor(FgColor);
349                 break;
350             case 2:
351                 glcdSetFgColor(GREEN);
352                 break;
353             case 3:
354                 glcdSetFgColor(RED);
355                 break;
356         }
357         glcdLine(GR_X + buffer[ch][y-1], y-1 + GR_Y, GR_X + buffer[ch][y], y + GR_Y);
358     }
359 }
360 old_adc_nr_channels = adc_channels;
361
362 //info box
363 #define INFO_X GR_X+GR_WW-55
364 #define INFO_Y GR_Y+GR_HH-25
365 glcdSetColors(FgColor, BkColor);
366 int y = INFO_Y - 9;
367 glcdMoveTo(INFO_X+2, y+=11); printf("td:");
368 glcdMoveTo(INFO_X+2, y+=11); printf("fq:");
369 y = INFO_Y - 9;
370 uint32_t time = adc_duration*GR_H;
371 if(time < 10000){
372     glcdMoveTo(INFO_X+15, y+=11); printf(" %4dus ", (int16_t)(time));
373 }else{
374     glcdMoveTo(INFO_X+15, y+=11); printf(" %4dms ", (int16_t)(time/1000));

```

```

375     }
376     if(freq[0] < 10000){
377         glcdMoveTo(INFO_X+15, y+=11); printf(" %4dHz ",(int16_t)(freq[0]));
378     }else{
379         glcdMoveTo(INFO_X+15, y+=11); printf(" %3dkHz ",(int16_t)(freq[0]/1000));
380     }
381 }

```

7.6.2.4 void disp_graph (void)

displays the osci graph

```

230         {
231     if(adc_stopped() == 0) return;
232
233     #define GR_W      20 //width of sub ... dash? unterteilung
234     #define GR_H      20 //height of ???
235     #define GR_WW     128 //width
236     #define GR_HH     144 //height
237     #define GR_X      2 //x-coord: upper-left
238     #define GR_Y      15 //y-coord: upper-left
239
240
241     //trigger
242     uint16_t trigger_shift = 72*adc_channels;
243     uint16_t GR_TRIG_BEFORE = ADC_BUF_SIZE - (GR_HH*adc_channels) + trigger_shift;
244
245     int8_t trigger_old = 0;
246     int8_t trigger_new = 0;
247     int8_t trigger_highest = 0;
248     uint16_t offset = 0;
249
250     for(uint16_t x = trigger_shift; x < GR_TRIG_BEFORE; x+=adc_channels){
251         trigger_old = adc[x] >> 1;
252         trigger_new = adc[x + 3*adc_channels] >> 1;
253         if((trigger_new-trigger_old) > trigger_highest){
254             offset = x - trigger_shift;
255             trigger_highest = trigger_new-trigger_old;
256         }
257     }
258
259     min[0] = 255; min[1] = 255; min[2] = 255; min[3] = 255;
260     max[0] = 0; max[1] = 0; max[2] = 0; max[3] = 0;
261     mid[0] = 0; mid[1] = 0; mid[2] = 0; mid[3] = 0;
262     freq[0] = 0; freq[1] = 0; freq[2] = 0; freq[3] = 0;
263     for(uint8_t ch = 0; ch < adc_channels; ch++){
264         for(uint16_t x = ch; x < ADC_BUF_SIZE; x+=adc_channels){
265             if(adc[x] < min[ch]) min[ch] = adc[x];
266             if(adc[x] > max[ch]) max[ch] = adc[x];
267         }
268     }
269     for(uint8_t x = 0; x < adc_channels; x++) mid[x] = (max[x]-min[x])/2 + min[x];
270
271
272     for(uint8_t ch = 0; ch < adc_channels; ch++){
273         uint16_t old_pos = 0;
274         int8_t under; //currently under mid? 0 for no, 1 for yes
275         int8_t old_under;
276         double period = 0.0;
277
278         if(adc[ch] < mid[ch]){ under = 0; old_under = 0; }
279         else { under = 1; old_under = 1; }
280
281         for(uint16_t x = ch; x < ADC_BUF_SIZE; x+=adc_channels){

```

```

282         if(adc[x] > mid[ch]) under = 0;
283         if(adc[x] < mid[ch]) under = 1;
284
285         if(old_under == 1 && under == 0){
286             period = (double)(x - old_pos)*adc_duration /adc_channels/ 1000000.0;
287             if(old_pos != 0 && period != 0.0){
288                 if(freq[ch] == 0.0) freq[ch] = 1.0 / period;
289                 else freq[ch] = 0.8*freq[ch] + 0.2/period;
290             }
291             old_pos = x;
292         }
293
294         old_under = under;
295     }
296 }
297
298
299
300 //draw in the display
301 disp_drawGrid(&adc[offset], 64, trigger_shift/adc_channels);
302
303 adc_init(1);
304 }

```

7.6.2.5 void disp_init (void)

initialises display

Calls the lib's routine to init the display. Displays the splash-screen. Sets the font. Sets the window for text-output.

```

26         {
27         //**** Hardware ****
28         #ifndef USE_AUTOINIT
29             glcdDisplayInit();
30         #endif
31
32         #ifdef DISP_SPLASH
33             disp_load_bitmap();
34         #else
35             disp_clean();
36         #endif
37
38         glcd_Flags.AutoLineFeed = 0;
39         glcdSelectFont(f8x11, 0); // font is stored in FLASH, thus no need for own read callback
40         fdevopen(glcdPut,NULL,0);
41
42         //window for text-output
43         glcd_Window.X1 = 5;
44         glcd_Window.X2 = 110;
45         glcd_Window.Y1 = 16;
46         glcd_Window.Y2 = 162;
47     }

```

7.6.2.6 void disp_load_bitmap (void)

displays a bitmap (used by splash)

7.6.2.7 void disp_menu (void)

displays the menu

Refreshes the menu bar when the menu has changed.

```

75         {
76     static int8_t main_old = 0;
77
78     /*** left ***/
79     if(menu_now != main_old){
80         main_old = menu_now;
81         disp_clean();
82
83
84         glcdFillRect(0,0,132,13,ShColor); //shadow color for menu bg
85         #define SPACELEFT 6
86         for(uint8_t x=0; x < MENU_ANZ_MAIN; x++){
87             if(menu_now == (x+1)){
88                 glcdFillRect(SPACELEFT+x*40,1, SPACELEFT+(x+1)*40,12, BkColor);
89                 glcdSetColors(FgColor, BkColor);
90             }else{
91                 glcdSetColors(FgColor, ShColor);
92             }
93             uint8_t w = glcdCharsWidth(menu_mains[x].name, 0);
94             glcdMoveTo(SPACELEFT+21+x*40 - w/2, 2);
95             printf(menu_mains[x].name);
96         }
97     }
98 }

```

7.6.2.8 void disp_off (void)

switches the display off

The original shutdown sequence for the display is not known. The current workaround is to activate the stand-by for the display.

```

51         {
52     bk_LED(1);
53     //for(uint32_t x=0; x < 5000000; x++) asm("nop");
54
55     //clean
56     glcdSetAddr(0,0, 131, 175); // set RAM access pointer of display
57     //width and height also defined in header
58     for(uint8_t x = 0; x < 132; x++){
59         for(uint8_t y = 0; y < 176; y++){
60             glcdPutPixel(WHITE);
61         }
62     }
63     for(uint32_t x=0; x < 500000; x++) asm("nop");
64     glcdDisplayOff();
65     glcdWait(100);
66 }

```

7.6.2.9 void disp_prop (void)

displays the properties

Refreshes the property bar if the menu, the property or the value of the property has changed.

```

102         {
103     static int8_t main_old = 0;
104     static int8_t prop_old = 0;

```

```

105     static int8_t value_old = 0;
106
107     if( menu_now != main_old ||
108         MENU_CURRENT_PROP_NR != prop_old ||
109         MENU_PROP_NOW.current_value != value_old){
110
111         main_old = menu_now;
112         prop_old = MENU_CURRENT_PROP_NR;
113         value_old = MENU_PROP_NOW.current_value;
114
115         #define DISP_PROP_W 33
116         glcdFillRect(0, 162, 132, 176, ShColor);
117         for(uint8_t x=0; x < 3; x++){
118             if(x == 1){
119                 glcdFillRect(SPACELEFT+x*DISP_PROP_W,163, SPACELEFT+(x+1)*DISP_PROP_W,174, BkColor);
120                 glcdSetColors(FgColor, BkColor);
121             }else{
122                 glcdSetColors(FgColor, ShColor);
123             }
124             int8_t current_value = value_old-1+x;
125             char *label;
126             if(current_value < 1)
127                 label = "=>";
128             else if(current_value > MENU_PROP_NOW.nr_values)
129                 label = "<=";
130             else
131                 label = MENU_PROP_NOW.value_name[current_value-1];
132             uint8_t w = glcdCharsWidth(label, 0);
133             glcdMoveTo(SPACELEFT+21+x*DISP_PROP_W - w, 164);
134             printf(label);
135
136             //prop-name
137             w = glcdCharsWidth(MENU_PROP_NOW.prop_name, 0);
138             glcdFillRect(132 - 2*SPACELEFT - w, 163, 130, 174, BkColor);
139             glcdSetColors(FgColor, BkColor);
140             glcdMoveTo(132 - SPACELEFT - w, 164);
141             printf(MENU_PROP_NOW.prop_name);
142         }
143     }
144 }

```

7.6.2.10 void disp_refresh (void)

refreshes the display

```

68         {
69     disp_menu();
70     menu_mains[menu_now-1].disp_func();
71     disp_prop();
72 }

```

7.6.2.11 void disp_terminal (void)

displays a terminal

The terminal for the serial interface.

Todo

Testing necessary!

```
201         {
202 //   if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162) glcdMoveTo(5,16);
203   if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162 || glcd_Cursor.X > 100) glcdMove
204   glcdSetColors(WHITE,BkColor);
205
206   uint16_t rx_data;
207   while(! ((rx_data=uart_getc()) & UART_NO_DATA)){
208     //glcdPut(rx_data & 255);
209     glcdDrawChar((uint8_t)rx_data);
210   }
211   while(! ((rx_data=uart1_getc()) & UART_NO_DATA)){
212     //glcdPut(rx_data & 255);
213     glcdDrawChar((uint8_t)rx_data);
214   }
215
216 //   if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162 || glcd_Cursor.X > 120) glcdMove
217 //   //printf("%c", (rx_data & 255));
218
219 //   //glcdWait(100);
220 }
```

7.7 input.c File Reference

```
#include "input.h"
```

Defines

- #define [ENTPR](#) 1000
- #define [PRESS_LONG](#) 1000000

Functions

- void [voidfunc](#) (void)
- void [tasten_status](#) (void)
- void [switch_off](#) (void)
- void [input_init](#) (void)
initialises the input
- void [input_refresh](#) (void)
refreshes the input

Variables

- int8_t [joy_adc_ready](#) = 0
- void(* [taste](#) [8])(void)
- void(* [taste_long](#) [8])(void)

7.7.1 Define Documentation

7.7.1.1 #define ENTPR 1000

7.7.1.2 #define PRESS_LONG 1000000

7.7.2 Function Documentation

7.7.2.1 void input_init (void)

initialises the input

```
30         {
31     taste[0] = &menu_main_incr;
32     taste[1] = &menu_start;
33     taste[2] = &menu_left;
34     taste[3] = &menu_up;
35     taste[4] = &menu_down;
36     taste[5] = &menu_right;
37     taste[6] = &voidfunc;
38     taste[7] = &voidfunc;
39     taste_long[0] = &switch_off;
40     taste_long[1] = &voidfunc;
41     taste_long[2] = &voidfunc;
```



```

42     taste_long[3] = &voidfunc;
43     taste_long[4] = &voidfunc;
44     taste_long[5] = &voidfunc;
45     taste_long[6] = &voidfunc;
46     taste_long[7] = &voidfunc;
47 }

```

7.7.2.2 void input_refresh (void)

refreshes the input

this function needs to be called regularly in order to register all button events.

Bug

With prescaler 7 `input_refresh()` does not get called often enough.

Todo

Copy code from `tasten_status()` directly in here?

```

54     {
55     taste_status();
56 }

```

7.7.2.3 void switch_off (void)

```

25     {
26     disp_off();
27     PORTA = 0;
28 }

```

7.7.2.4 void tasten_status (void)

```

61     {
62     static uint8_t taster = 255;
63     static uint32_t last_down = 0;
64
65     uint8_t taster_neu = PINC;
66     uint8_t diff = taster_neu ^ taster; //contains changes; "^" is xor
67     diff &= taster_neu; //on_release; substitute taster_neu to taster to get on_push
68
69     //if(count < 65535) count++;
70     double timeFromLastDown = us_time_get_difference_d(last_down);
71     #define ENTPR 1000
72     #define PRESS_LONG 1000000
73     if(timeFromLastDown > ENTPR){ //entprellen: sicherstellen, dass bestimmte Zeit vergangen ist
74         if (bit_is_set(diff,0)) taste[0] (); //Taste 1
75         else if(bit_is_set(diff,1)) taste[1] (); //TASTE 2
76         else if(bit_is_set(diff,2)) taste[2] (); //TASTE 3
77         else if(bit_is_set(diff,3)) taste[3] (); //TASTE 4
78         else if(bit_is_set(diff,4)) taste[4] (); //TASTE 5
79         else if(bit_is_set(diff,5)) taste[5] (); //TASTE 6
80         else if(bit_is_set(diff,6)) taste[6] (); //TASTE 7
81         else if(bit_is_set(diff,7)) taste[7] (); //TASTE 8
82     }
83     if(taster != taster_neu) last_down = us_time_get(); //entprelltime neu setzen
84     timeFromLastDown = us_time_get_difference_d(last_down);
85     if(timeFromLastDown > PRESS_LONG){

```

```
86     if      (bit_is_clear(taster_neu,0) && bit_is_clear(diff,0)) taste_long[0] ();
87     else if (bit_is_clear(taster_neu,1) && bit_is_clear(diff,1)) taste_long[1] ();
88     else if (bit_is_clear(taster_neu,2) && bit_is_clear(diff,2)) taste_long[2] ();
89     else if (bit_is_clear(taster_neu,3) && bit_is_clear(diff,3)) taste_long[3] ();
90     else if (bit_is_clear(taster_neu,4) && bit_is_clear(diff,4)) taste_long[4] ();
91     else if (bit_is_clear(taster_neu,5) && bit_is_clear(diff,5)) taste_long[5] ();
92     else if (bit_is_clear(taster_neu,6) && bit_is_clear(diff,6)) taste_long[6] ();
93     else if (bit_is_clear(taster_neu,7) && bit_is_clear(diff,7)) taste_long[7] ();
94 }
95 taster = taster_neu;
96 }
```

7.7.2.5 void voidfunc (void)

```
17 {}
```

7.7.3 Variable Documentation

7.7.3.1 int8_t joy_adc_ready = 0

7.7.3.2 void(* taste[8])(void)

7.7.3.3 void(* taste_long[8])(void)

7.8 input.h File Reference

input

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/signal.h>
#include "main.h"
```

Functions

- void [input_init](#) (void)
initialises the input
- void [input_refresh](#) (void)
refreshes the input

7.8.1 Detailed Description

input

Handles the button events.

12 Octobre 2005

Sven Kreiss

7.8.2 Function Documentation

7.8.2.1 void input_init (void)

initialises the input

```
30         {
31     taste[0] = &menu_main_incr;
32     taste[1] = &menu_start;
33     taste[2] = &menu_left;
34     taste[3] = &menu_up;
35     taste[4] = &menu_down;
36     taste[5] = &menu_right;
37     taste[6] = &voidfunc;
38     taste[7] = &voidfunc;
39     taste_long[0] = &switch_off;
40     taste_long[1] = &voidfunc;
41     taste_long[2] = &voidfunc;
42     taste_long[3] = &voidfunc;
43     taste_long[4] = &voidfunc;
44     taste_long[5] = &voidfunc;
45     taste_long[6] = &voidfunc;
46     taste_long[7] = &voidfunc;
47 }
```

7.8.2.2 void input_refresh (void)

refreshes the input

this function needs to be called regularly in order to register all button events.

Bug

With prescaler 7 [input_refresh\(\)](#) does not get called often enough.

Todo

Copy code from [tasten_status\(\)](#) directly in here?

```
54         {
55     tasten_status();
56 }
```

7.9 main.c File Reference

```
#include <main.h>
```

Functions

- void [interrupt_init](#) (void)
- void [init_ports](#) (void)
- void [call_inits](#) (void)
- void [bk_LED](#) (int8_t value)
Function to connect to property "Backlight brightness".
- int [main](#) (void)

7.9.1 Function Documentation

7.9.1.1 void bk_LED (int8_t value)

Function to connect to property "Backlight brightness".

Form defined through the first function-pointer in struct `menue_prop_struct` in [menu.h](#).

```
86 {  
87     if(value >= 1 && value <= 11) OCR2 = (value-1) * 12;    //10*12 = 120  
88 }
```

7.9.1.2 void call_inits (void)

```
42 {  
43     init_ports();  
44     menu_init();  
45     control_init();  
46     input_init();  
47     adc_init(0);  
48     disp_init();  
49     usart_init();  
50     interrupt_init();  
51 }
```

7.9.1.3 void init_ports (void)

```
29 {  
30     DDRA=255;    //OUTPUT  
31     PORTA=255;  //on  
32  
33     DDRC = 0;    //Input  
34     PORTC = 255; //pull ups  
35     DDRD = (1<<PD3); //PD3 Output  
36     PORTD = 255;  
37     DDRE = 0;   //input  
38     PORTE = 255;  
39 }
```

7.9.1.4 void interrupt_init (void)

Enables Timer2 for Backlight. Enables Timer3 for micro-second measurement. Enables interrupts globally.

```
66 {
67     //timer 2 in fast PWM for backlight
68     PORTB &= ~(1<<PB7);
69     DDRB |= (1<<PB7); //OC2 pin is output
70     TCCR2 = (1<<WGM21) | (1<<WGM20) | (1<<COM21) | (1<<CS20); //no prescaler
71     TCNT2 = 0;
72     OCR2 = 108; //max 120
73
74     //timer 3 for us_timer
75     us_timer_init();
76
77     // enable interrupts
78     sei();
79 }
```

7.9.1.5 int main (void)

Main, with test-code.

```
97 {
98     //define cnt 1
99     DDRE = (1<<PE2); //sets data direction for xck0 to output
100
101     call_inits();
102     #ifdef DISP_SPLASH
103         for(uint32_t x=0; x < 5000000; x++) asm("nop");
104     #endif
105
106     while(1==1){
107         //uart_empfang();
108         control_refresh();
109         disp_refresh();
110         input_refresh();
111     }
112 }
```

7.10 main.h File Reference

main

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "menu.h"
#include "control.h"
#include "input.h"
#include "display.h"
#include "uart.h"
#include "adc.h"
#include "ustimer.h"
```

Functions

- void [bk_LED](#) (int8_t value)
Function to connect to property "Backlight brightness".

7.10.1 Detailed Description

main

Contains the main method.

ca. 01/01/2006

Sven Kreiss

7.10.2 Function Documentation

7.10.2.1 void [bk_LED](#) (int8_t value)

Function to connect to property "Backlight brightness".

Form defined through the first function-pointer in struct `menue_prop_struct` in [menu.h](#).

```
86 {
87     if(value >= 1 && value <= 11) OCR2 = (value-1) * 12;    //10*12 = 120
88 }
```

7.11 menu.c File Reference

```
#include "menu.h"
```

Functions

- void `menu_init` (void)
initialise menu
- void `menu_main_set` (int8_t nr)
set to menu "nr"
- void `menu_main_incr` (void)
next menu
- void `menu_main_decr` (void)
menu before
- void `menu_prop_set` (int8_t nr)
set prop in argument active
- void `menu_start` (void)
Handels pressed signal for button "start".
- void `menu_left` (void)
Handels pressed signal for button "left".
- void `menu_right` (void)
Handels pressed signal for button "right".
- void `menu_up` (void)
Handels pressed signal for button "up".
- void `menu_down` (void)
Handels pressed signal for button "down".

Variables

- volatile `menu_main_struct menu_mains` []
array which holds one `menu_main_struct` for each entry
- volatile `menu_prop_main_struct menu_props` [MENU_ANZ_MAIN]
array holding one `menu_prop_main_struct` for each property

7.11.1 Function Documentation

7.11.1.1 void menu_down (void)

Handels pressed signal for button "down".

```
85         {
86     if (MENU_PROP_NOW.current_value > 1) MENU_PROP_NOW.current_value--;
87     MENU_PROP_NOW.set_value (MENU_PROP_NOW.current_value);
88 }
```

7.11.1.2 void menu_init (void)

initialise menu

```
43         {
44     menu_main_set (1);
45     menu_prop_set (1);
46 }
```

7.11.1.3 void menu_left (void)

Handels pressed signal for button "left".

```
71         {
72     if (MENU_CURRENT_PROP_NR > 1) MENU_PROPS_NOW.prop_now--;
73     else MENU_PROPS_NOW.prop_now = MENU_PROPS_NOW.nr_props;
74 }
```

7.11.1.4 void menu_main_decr (void)

menu before

```
58         {
59     if (menu_now > 1) menu_main_set (menu_now - 1);
60     else menu_main_set (MENU_ANZ_MAIN);
61 }
```

7.11.1.5 void menu_main_incr (void)

next menu

```
53         {
54     if (menu_now < MENU_ANZ_MAIN) menu_main_set (menu_now + 1);
55     else menu_main_set (1);
56 }
```

7.11.1.6 void menu_main_set (int8_t nr)

set to menu "nr"

```
49         {
50     if(nr <= MENU_ANZ_MAIN) menu_now = nr;
51 }
```

7.11.1.7 void menu_prop_set (int8_t nr)

set prop in argument active

```
63         {
64     if(nr <= MENU_PROPS_NOW.nr_props) MENU_PROPS_NOW.prop_now = nr;
65 }
```

7.11.1.8 void menu_right (void)

Handels pressed signal for button "right".

```
75         {
76     if(MENU_CURRENT_PROP_NR < MENU_PROPS_NOW.nr_props) MENU_PROPS_NOW.prop_now++;
77     else MENU_PROPS_NOW.prop_now = 1;
78 }
```

7.11.1.9 void menu_start (void)

Handels pressed signal for button "start".

```
69 {}
```

7.11.1.10 void menu_up (void)

Handels pressed signal for button "up".

```
80         {
81     if(MENU_PROP_NOW.current_value < MENU_PROP_NOW.nr_values)
82         MENU_PROP_NOW.current_value++;
83     MENU_PROP_NOW.set_value(MENU_PROP_NOW.current_value);
84 }
```

7.11.2 Variable Documentation**7.11.2.1 volatile [menu_main_struct](#) [menu_mains](#) []**

Initial value:

```
{
  { &disp_graph, &control_graph, "Graph" },
  { &disp_terminal, &control_graph, "TTY" },
  { &disp_debugging, &control_graph, "FFT" },
}
```

array which holds one [menu_main_struct](#) for each entry

Initialise the modes (or main menu entries).

Todo

changes needed: -FFT, +Logic Analyzer, +multimeter function (U, I, R)

7.11.2.2 volatile [menu_prop_main_struct](#) [menu_props](#)[MENU_ANZ_MAIN]

Initial value:

```
{
  { 3, 1, {
    { &adc_set_presc, 1, 7, "Pr", {"1","2","3","4","5","6","7"} },
    { &adc_set_nr_channels, 1, 4, "Ch", {"1","2","3","4"} },
    { &bk_LED, 10, 11, "LED", {"0","1","2","3","4","5","6","7","8","9","10"} },
  } },
  { 4, 1, {
    { &usart_baudrate, 1, 11, "BR", {"2k4","4k8","9k6","14k4","19k2","28k8","38k4","57k6","76k8","115k"} },
    { &usart_setSync, 1, 2, "SY", {"dis","en"} },
    { &usart_dataBits, 4, 5, "DB", {"5","6","7","8","9"} },
    { &usart_stopBits, 2, 2, "SB", {"1","2"} },
  } },
  { 1, 1,
    {{ &adc_set_presc, 1, 5, "TB", {"1","2","3","4","5"} }} },
}
```

array holding one [menu_prop_main_struct](#) for each property

Initialises all properties

7.12 menu.h File Reference

menu

```
#include "main.h"
```

Data Structures

- struct [menu_main_struct](#)
Main menu – Mode.
- struct [menu_prop_struct](#)
Properties.
- struct [menu_prop_main_struct](#)
container for properties

Defines

- #define [MENU_ANZ_MAIN](#) 3
nr of main menu entries
- #define [MENU_PROPS_NOW](#) menu_props[menu_now-1]
current properties array
- #define [MENU_CURRENT_PROP_NR](#) menu_props[menu_now-1].prop_now
nr of the current property
- #define [MENU_PROP_NOW](#) MENU_PROPS_NOW.properties[MENU_CURRENT_PROP_NR-1]
the current property

Functions

- volatile void [menu_init](#) (void)
initialise menu
- volatile void [menu_main_set](#) (int8_t nr)
set to menu "nr"
- void [menu_main_incr](#) (void)
next menu
- void [menu_main_decr](#) (void)
menu before
- void [menu_prop_set](#) (int8_t)

set prop in argument active

- void `menu_start` (void)
Handels pressed signal for button "start".
- void `menu_left` (void)
Handels pressed signal for button "left".
- void `menu_up` (void)
Handels pressed signal for button "up".
- void `menu_down` (void)
Handels pressed signal for button "down".
- void `menu_right` (void)
Handels pressed signal for button "right".

Variables

- volatile int8_t `menu_now`
nr of current menu point
- volatile `menu_main_struct` `menu_mains` []
array which holds one `menu_main_struct` for each entry
- volatile `menu_prop_main_struct` `menu_props` [MENU_ANZ_MAIN]
array holding one `menu_prop_main_struct` for each property

7.12.1 Detailed Description

menu

"Data-backend" for the menu.

03 April 2006

Sven Kreiss

7.12.2 Define Documentation

7.12.2.1 #define MENU_ANZ_MAIN 3

nr of main menu entries

7.12.2.2 #define MENU_CURRENT_PROP_NR `menu_props[menu_now-1].prop_now`

nr of the current property

7.12.2.3 #define MENU_PROP_NOW MENU_PROPS_NOW.properties[MENU_CURRENT_PROP_NR-1]

the current property

7.12.2.4 #define MENU_PROPS_NOW menu_props[menu_now-1]

current properties array

7.12.3 Function Documentation

7.12.3.1 void menu_down (void)

Handels pressed signal for button "down".

```

85         {
86     if (MENU_PROP_NOW.current_value > 1) MENU_PROP_NOW.current_value--;
87     MENU_PROP_NOW.set_value (MENU_PROP_NOW.current_value);
88 }
```

7.12.3.2 volatile void menu_init (void)

initialise menu

```

43         {
44     menu_main_set (1);
45     menu_prop_set (1);
46 }
```

7.12.3.3 void menu_left (void)

Handels pressed signal for button "left".

```

71         {
72     if (MENU_CURRENT_PROP_NR > 1) MENU_PROPS_NOW.prop_now--;
73     else MENU_PROPS_NOW.prop_now = MENU_PROPS_NOW.nr_props;
74 }
```

7.12.3.4 void menu_main_decr (void)

menu before

```

58         {
59     if (menu_now > 1) menu_main_set (menu_now - 1);
60     else menu_main_set (MENU_ANZ_MAIN);
61 }
```

7.12.3.5 void menu_main_incr (void)

next menu

```
53         {
54     if (menu_now < MENU_ANZ_MAIN)    menu_main_set (menu_now + 1);
55     else                             menu_main_set (1);
56 }
```

7.12.3.6 volatile void menu_main_set (int8_t nr)

set to menu "nr"

```
49         {
50     if (nr <= MENU_ANZ_MAIN) menu_now = nr;
51 }
```

7.12.3.7 void menu_prop_set (int8_t)

set prop in argument active

```
63         {
64     if (nr <= MENU_PROPS_NOW.nr_props) MENU_PROPS_NOW.prop_now = nr;
65 }
```

7.12.3.8 void menu_right (void)

Handels pressed signal for button "right".

```
75         {
76     if (MENU_CURRENT_PROP_NR < MENU_PROPS_NOW.nr_props) MENU_PROPS_NOW.prop_now++;
77     else MENU_PROPS_NOW.prop_now = 1;
78 }
```

7.12.3.9 void menu_start (void)

Handels pressed signal for button "start".

```
69 {}
```

7.12.3.10 void menu_up (void)

Handels pressed signal for button "up".

```
80         {
81     if (MENU_PROP_NOW.current_value < MENU_PROP_NOW.nr_values)
82         MENU_PROP_NOW.current_value++;
83     MENU_PROP_NOW.set_value (MENU_PROP_NOW.current_value);
84 }
```

7.12.4 Variable Documentation

7.12.4.1 volatile `menu_main_struct menu_mains[]`

array which holds one `menu_main_struct` for each entry

Initialise the modes (or main menu entries).

Todo

changes needed: -FFT, +Logic Analyzer, +multimeter function (U, I, R)

7.12.4.2 volatile `int8_t menu_now`

nr of current menu point

7.12.4.3 volatile `menu_prop_main_struct menu_props[MENU_ANZ_MAIN]`

array holding one `menu_prop_main_struct` for each property

Initialises all properties

7.13 uart.c File Reference

```
#include "uart.h"
```

Functions

- void `usart_init` (void)
initialises both USARTs
- void `usart_off` (void)
switches off both USARTs
- void `usart_baudrate` (int8_t br)
property: sets baudrate
- void `usart_setSync` (int8_t enable)
property: enables synchronous communication
- void `usart_stopBits` (int8_t sb)
property: sets the nr of stop bits
- void `usart_dataBits` (int8_t db)
property: sets the nr of data bits

7.13.1 Function Documentation

7.13.1.1 void usart_baudrate (int8_t br)

property: sets baudrate

```
52                                     {
53     uint16_t reg = 0;
54
55     //all values for 11.0592MHz oscillator
56     if      (br == 0) reg = 287; //2k4
57     else if (br == 1) reg = 143; //4k8
58     else if (br == 2) reg = 71;  //9k6
59     else if (br == 3) reg = 47;  //14k4
60     else if (br == 4) reg = 35;  //19k2
61     else if (br == 5) reg = 23;  //28k8
62     else if (br == 6) reg = 17;  //38k4
63     else if (br == 7) reg = 11;  //57k6
64     else if (br == 8) reg = 8;   //76k8
65     else if (br == 9) reg = 5;   //115k2
66     else if (br == 10) reg = 2;  //230k4
67
68     UBRR0H = ((reg>>8) & 255); UBRR0L = reg & 255;
69     UBRR1H = ((reg>>8) & 255); UBRR1L = reg & 255;
70
71 // UBRR0H = 0; UBRR0L = 5;
72 // UBRR1H = 0; UBRR1L = 5;
73 }
```

7.13.1.2 void usart_dataBits (int8_t db)

property: sets the nr of data bits

```

97                                     {
98     if      (db == 0){ UCSR0C |= (0<<UCSZ0); UCSR1C |= (0<<UCSZ1); } //5bits
99     else if (db == 1){ UCSR0C |= (1<<UCSZ0); UCSR1C |= (1<<UCSZ1); } //6bits
100    else if (db == 2){ UCSR0C |= (2<<UCSZ0); UCSR1C |= (2<<UCSZ1); } //7bits
101    else if (db == 3){ UCSR0C |= (3<<UCSZ0); UCSR1C |= (3<<UCSZ1); } //8bits
102    else if (db == 4){ UCSR0C |= (15<<UCSZ0); UCSR1C |= (15<<UCSZ1); } //9bits
103 }

```

7.13.1.3 void usart_init (void)

initialises both USARTs

```

17                                     {
18     //baudrate
19     // UBRR0H = 0; UBRR0L = 0;
20     // UBRR1H = 0; UBRR1L = 0;
21     usart_baudrate(0);
22
23     //erase data overflow flag
24     UCSR0A &= ~(1<<DOR);
25     UCSR1A &= ~(1<<DOR);
26
27     //RXEN, TXEN, INTERRUPT ENABLE
28     UCSR0B |= (1<<RXEN) | (1<<TXEN) | (1<<RXCIE);
29     UCSR1B |= (1<<RXEN) | (1<<TXEN) | (1<<RXCIE);
30
31     //2 stop bit, 8 data bit, synchronous mode
32     // UCSR0C = (1<<USBS) | (3<<UCSZ0); // | (1<<UMSEL);
33     // UCSR1C = (1<<USBS) | (3<<UCSZ0); // | (1<<UMSEL);
34     usart_stopBits(1);
35     usart_setSync(0);
36     usart_dataBits(3);
37
38     //activate internal PullUp for RX and XCK(if input)
39     //DDRE = 255;
40     PORTE |= (1<<PE0) | (1<<PE2);
41     PORTD |= (1<<PD2) | (1<<PD5);
42
43     uart_init(UART_BAUD_SELECT(9600, 11059200UL));
44     uart1_init(UART_BAUD_SELECT(9600, 11059200UL));
45 }

```

7.13.1.4 void usart_off (void)

switches off both USARTs

```

47                                     {
48     UCSR0B &= ~(1<<RXEN) & ~(1<<TXEN);
49     UCSR1B &= ~(1<<RXEN) & ~(1<<TXEN);
50 }

```

7.13.1.5 void usart_setSync (int8_t enable)

property: enables synchronous communication

```
75     {
76     if(enable == 1){ //sync enabled
77         UCSR0C |= (1<<UMSEL);
78         UCSR1C |= (1<<UMSEL);
79     }
80     else if(enable == 0){ //sync disabled
81         UCSR0C &= ~(1<<UMSEL);
82         UCSR1C &= ~(1<<UMSEL);
83     }
84 }
```

7.13.1.6 void usart_stopBits (int8_t sb)

property: sets the nr of stop bits

```
86     {
87     if(sb == 0){ //1 stop bit
88         UCSR0C &= ~(1<<USBS);
89         UCSR1C &= ~(1<<USBS);
90     }
91     else if(sb == 1){ //2 stop bits
92         UCSR0C |= (1<<USBS);
93         UCSR1C |= (1<<USBS);
94     }
95 }
```

7.14 uart.h File Reference

UART.

```
#include "main.h"
#include "../libs/uartlibrary/uart.h"
```

Defines

- #define [UART_RX_BUFFER_SIZE](#) 8
- #define [UART_TX_BUFFER_SIZE](#) 8

Functions

- void [usart_init](#) (void)
initialises both USARTs
- void [usart_off](#) (void)
switches off both USARTs
- void [usart_baudrate](#) (int8_t br)
property: sets baudrate
- void [usart_setSync](#) (int8_t enable)
property: enables synchronous communication
- void [usart_stopBits](#) (int8_t sb)
property: sets the nr of stop bits
- void [usart_dataBits](#) (int8_t db)
property: sets the nr of data bits

7.14.1 Detailed Description

UART.

Serial communication.

Todo

All the functions need to be completed.
Peter Fleury's lib?

ca 01 January 2006

Sven Kreiss

7.14.2 Define Documentation

7.14.2.1 #define UART_RX_BUFFER_SIZE 8

7.14.2.2 #define UART_TX_BUFFER_SIZE 8

7.14.3 Function Documentation

7.14.3.1 void usart_baudrate (int8_t br)

property: sets baudrate

```

52                                     {
53     uint16_t reg = 0;
54
55     //all values for 11.0592MHz oscillator
56     if      (br == 0) reg = 287; //2k4
57     else if (br == 1) reg = 143; //4k8
58     else if (br == 2) reg = 71;  //9k6
59     else if (br == 3) reg = 47;  //14k4
60     else if (br == 4) reg = 35;  //19k2
61     else if (br == 5) reg = 23;  //28k8
62     else if (br == 6) reg = 17;  //38k4
63     else if (br == 7) reg = 11;  //57k6
64     else if (br == 8) reg = 8;   //76k8
65     else if (br == 9) reg = 5;   //115k2
66     else if (br == 10) reg = 2;  //230k4
67
68     UBRR0H = ((reg>>8) & 255); UBRR0L = reg & 255;
69     UBRR1H = ((reg>>8) & 255); UBRR1L = reg & 255;
70
71 // UBRR0H = 0; UBRR0L = 5;
72 // UBRR1H = 0; UBRR1L = 5;
73 }
```

7.14.3.2 void usart_dataBits (int8_t db)

property: sets the nr of data bits

```

97                                     {
98     if      (db == 0){ UCSR0C |= (0<<UCSZ0); UCSR1C |= (0<<UCSZ1); } //5bits
99     else if (db == 1){ UCSR0C |= (1<<UCSZ0); UCSR1C |= (1<<UCSZ1); } //6bits
100    else if (db == 2){ UCSR0C |= (2<<UCSZ0); UCSR1C |= (2<<UCSZ1); } //7bits
101    else if (db == 3){ UCSR0C |= (3<<UCSZ0); UCSR1C |= (3<<UCSZ1); } //8bits
102    else if (db == 4){ UCSR0C |= (15<<UCSZ0); UCSR1C |= (15<<UCSZ1); } //9bits
103 }
```

7.14.3.3 void usart_init (void)

initialises both USARTs

```

17                                     {
18     //baudrate
19 // UBRR0H = 0; UBRR0L = 0;
20 // UBRR1H = 0; UBRR1L = 0;
21     usart_baudrate(0);
22 }
```

```

23 //erase data overflow flag
24 UCSR0A &= ~(1<<DOR);
25 UCSR1A &= ~(1<<DOR);
26
27 //RXEN, TXEN, INTERRUPT ENABLE
28 UCSR0B |= (1<<RXEN) | (1<<TXEN) | (1<<RXCIE);
29 UCSR1B |= (1<<RXEN) | (1<<TXEN) | (1<<RXCIE);
30
31 //2 stop bit, 8 data bit, synchronous mode
32 // UCSR0C = (1<<USBS) | (3<<UCSZ0); // | (1<<UMSEL);
33 // UCSR1C = (1<<USBS) | (3<<UCSZ0); // | (1<<UMSEL);
34 usart_stopBits(1);
35 usart_setSync(0);
36 usart_dataBits(3);
37
38 //activate internal PullUp for RX and XCK(if input)
39 //DDRE = 255;
40 PORTE |= (1<<PE0) | (1<<PE2);
41 PORTD |= (1<<PD2) | (1<<PD5);
42
43 uart_init(UART_BAUD_SELECT(9600, 11059200UL));
44 uart1_init(UART_BAUD_SELECT(9600, 11059200UL));
45 }

```

7.14.3.4 void usart_off (void)

switches off both USARTs

```

47 {
48     UCSR0B &= ~(1<<RXEN) & ~(1<<TXEN);
49     UCSR1B &= ~(1<<RXEN) & ~(1<<TXEN);
50 }

```

7.14.3.5 void usart_setSync (int8_t enable)

property: enables synchronous communication

```

75 {
76     if(enable == 1){ //sync enabled
77         UCSR0C |= (1<<UMSEL);
78         UCSR1C |= (1<<UMSEL);
79     }
80     else if(enable == 0){ //sync disabled
81         UCSR0C &= ~(1<<UMSEL);
82         UCSR1C &= ~(1<<UMSEL);
83     }
84 }

```

7.14.3.6 void usart_stopBits (int8_t sb)

property: sets the nr of stop bits

```

86 {
87     if(sb == 0){ //1 stop bit
88         UCSR0C &= ~(1<<USBS);
89         UCSR1C &= ~(1<<USBS);
90     }

```

```
91     else if (sb == 1) { //2 stop bits
92         UCSR0C |= (1<<USBS);
93         UCSR1C |= (1<<USBS);
94     }
95 }
```

7.15 ustimer.c File Reference

```
#include "ustimer.h"
```

Defines

- #define [US_TIMER_PRESCALER](#) 8
- #define [XTAL](#) 11.0562

Functions

- void [us_timer_init](#) (void)
initiates the timer
- [SIGNAL](#) (SIG_OVERFLOW3)
- uint32_t [us_time_get](#) (void)
get current time
- uint32_t [us_time_get_difference](#) (uint32_t time1)
calculates the difference between a saved and the current time
- double [us_time_get_difference_d](#) (uint32_t time1)
calculates the difference between a saved and the current time in micro-seconds

Variables

- volatile uint32_t [us_time](#) = 0
internal counter

7.15.1 Define Documentation

7.15.1.1 #define [US_TIMER_PRESCALER](#) 8

7.15.1.2 #define [XTAL](#) 11.0562

7.15.2 Function Documentation

7.15.2.1 [SIGNAL](#) (SIG_OVERFLOW3)

The interrupt handler for the micro-second(us) timer.

```
36 {  
37     if(us_time < 0xFFFF) us_time++; //with 16Bit-Timer use 0xFFFF; 8bit: 0xFFFFF  
38     else us_time = 0;  
39 }
```


7.15.2.2 uint32_t us_time_get (void)

get current time

Calculates the current time from an incremented variable and the counter register of the timer.

```

46 {
47     uint16_t timer;
48
49     /* It is really, really important to stop global interrupts before
50      * reading 16bit registers. See the avr-libc FAQ! */
51     cli();
52     timer = TCNT3;
53     sei();
54
55     return ((us_time << 16) + timer);
56 }

```

7.15.2.3 uint32_t us_time_get_difference (uint32_t time1)

calculates the difference between a saved and the current time

Uses [us_time_get\(\)](#) to get the current time. The if-condition at the end checks whether the later time is smaller. If so, then an timer overflow is assumed and the appropriate action is taken that the correct time can still be calculated. Therefore, the maximum time one can measure is 2^{32} micro-seconds.

```

68 {
69     uint32_t time2;
70     time2 = us_time_get();
71     if(time2 >= time1) return (time2-time1);
72     else return (0xFFFFFFFF - time1+time2); //2^32
73 }

```

7.15.2.4 double us_time_get_difference_d (uint32_t time1)

calculates the difference between a saved and the current time in micro-seconds

Same as [us_time_get_difference\(\)](#), but returns the value in micro seconds.

```

79 {
80     return ((double)(us_time_get_difference(time1) * US_TIMER_PRESCALER) / (double)XTAL);
81 }

```

7.15.2.5 void us_timer_init (void)

initiates the timer

Initiates 16bit-Timer3 for the precise measurement. Enables Timer overflow interrupt.

< for XTAL > 8 MHz: 8, else: 1

< cpu-freq in MHz

```

22 {
23     #define US_TIMER_PRESCALER 8    ///< for XTAL > 8 MHz: 8, else: 1
24     #define XTAL 11.0562           ///< cpu-freq in MHz

```

```
25
26     TCCR3A = 0;                //normal mode
27     TCCR3B = (1<<CS31);       //CS31: Prescaler 8, CS00: Prescaler 1
28     ETIMSK = (1<<TOIE3);     //enable Timer3 overflow interrupt
29     //TIMSK = (1<<TOIE1);     //enable Timer1 overflow interrupt
30     TCNT3 = 0;
31 }
```

7.15.3 Variable Documentation

7.15.3.1 volatile uint32_t `us_time` = 0

internal counter

7.16 ustimer.h File Reference

micro-second(us) timer

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
```

Functions

- void [us_timer_init](#) (void)
initiates the timer
- uint32_t [us_time_get](#) (void)
get current time
- uint32_t [us_time_get_difference](#) (uint32_t time1)
calculates the difference between a saved and the current time
- double [us_time_get_difference_d](#) (uint32_t time1)
calculates the difference between a saved and the current time in micro-seconds

7.16.1 Detailed Description

micro-second(us) timer

Functions for precise time measurements. Tested with

- Timer3, ATmega128, 16bit
- Timer1, ATmega16, 16bit

No tests with 8bit-Timers so far, but it should work with minor changes.

Todo

ustimer could become a project on its own.

29 March 2006

Sven Kreiss

7.16.2 Function Documentation

7.16.2.1 uint32_t us_time_get (void)

get current time

Calculates the current time from an incremented variable and the counter register of the timer.

```

46 {
47     uint16_t timer;
48
49     /* It is really, really important to stop global interrupts before
50      * reading 16bit registers. See the avr-libc FAQ! */
51     cli();
52     timer = TCNT3;
53     sei();
54
55     return ((us_time << 16) + timer);
56 }

```

7.16.2.2 uint32_t us_time_get_difference (uint32_t time1)

calculates the difference between a saved and the current time

Uses [us_time_get\(\)](#) to get the current time. The if-condition at the end checks whether the later time is smaller. If so, then an timer overflow is assumed and the appropriate action is taken that the correct time can still be calculated. Therefore, the maximum time one can measure is 2^{32} micro-seconds.

```

68 {
69     uint32_t time2;
70     time2 = us_time_get();
71     if(time2 >= time1) return (time2-time1);
72     else return (0xFFFFFFFF - time1+time2); //2^32
73 }

```

7.16.2.3 double us_time_get_difference_d (uint32_t time1)

calculates the difference between a saved and the current time in micro-seconds

Same as [us_time_get_difference\(\)](#), but returns the value in micro seconds.

```

79 {
80     return ((double)(us_time_get_difference(time1) * US_TIMER_PRESCALER) / (double)XTAL);
81 }

```

7.16.2.4 void us_timer_init (void)

initiates the timer

Initiates 16bit-Timer3 for the precise measurement. Enables Timer overflow interrupt.

< for XTAL > 8 MHz: 8, else: 1

< cpu-freq in MHz

```

22 {
23     #define US_TIMER_PRESCALER 8    ///< for XTAL > 8 MHz: 8, else: 1
24     #define XTAL 11.0562           ///< cpu-freq in MHz
25
26     TCCR3A = 0;                    //normal mode
27     TCCR3B = (1<<CS31);            //CS31: Prescaler 8, CS00: Prescaler 1
28     ETIMSK = (1<<TOIE3);           //enable Timer3 overflow interrupt
29     //TIMSK = (1<<TOIE1);           //enable Timer1 overflow interrupt
30     TCNT3 = 0;
31 }

```

Chapter 8

OpenOsci Page Documentation

8.1 Todo List

Global [adc_stop](#) still needed?

Global [adc_stopped](#) still needed?

File [control.h](#) !!! Need to think about this concept again !!! Only [control_refresh\(\)](#) is not empty :-S .

Global [disp_drawGrid](#) Change from grid to coordinate axes. Origin is the trigger point and zero Volt.

Global [disp_terminal](#) Testing necessary!

Global [input_refresh](#) Copy code from [tasten_status\(\)](#) directly in here?

Global [menu_mains](#) changes needed: -FFT, +Logic Analyzer, +multimeter function (U, I, R)

File [uart.h](#) All the functions need to be completed.

Peter Fleury's lib?

File [ustimer.h](#) ustimer could become a project on its own.

8.2 Bug List

Global `input_refresh` With prescaler 7 `input_refresh()` does not get called often enough.

Index

adc
 adc.h, 23
adc.c, 15
 adc_channels, 19
 adc_count, 19
 adc_current_channel, 19
 adc_duration, 19
 adc_init, 16
 adc_LED, 16
 adc_multi_channels, 16
 adc_next_channel, 17
 adc_off, 17
 adc_prescaler, 19
 adc_select_channel, 17
 adc_set_nr_channels, 17
 adc_set_presc, 18
 adc_single_channel, 18
 adc_starttime, 19
 adc_stop, 18
 adc_stop_flag, 19
 adc_stopped, 18
adc.h, 20
 adc, 23
 ADC_BUF_SIZE, 21
 adc_channels, 23
 adc_duration, 23
 adc_init, 21
 adc_LED, 21
 adc_next_channel, 22
 adc_off, 22
 adc_prescaler, 23
 adc_select_channel, 22
 adc_set_nr_channels, 22
 adc_set_presc, 22
 adc_stop, 23
 adc_stopped, 23
ADC_BUF_SIZE
 adc.h, 21
adc_channels
 adc.c, 19
 adc.h, 23
adc_count
 adc.c, 19
adc_current_channel
 adc.c, 19
 adc_duration
 adc.c, 19
 adc.h, 23
 adc_init
 adc.c, 16
 adc.h, 21
 adc_LED
 adc.c, 16
 adc.h, 21
 adc_multi_channels
 adc.c, 16
 adc_next_channel
 adc.c, 17
 adc.h, 22
 adc_off
 adc.c, 17
 adc.h, 22
 adc_prescaler
 adc.c, 19
 adc.h, 23
 adc_select_channel
 adc.c, 17
 adc.h, 22
 adc_set_nr_channels
 adc.c, 17
 adc.h, 22
 adc_set_presc
 adc.c, 18
 adc.h, 22
 adc_single_channel
 adc.c, 18
 adc_starttime
 adc.c, 19
 adc_stop
 adc.c, 18
 adc.h, 23
 adc_stop_flag
 adc.c, 19
 adc_stopped
 adc.c, 18
 adc.h, 23
bk_LED
 main.c, 49
 main.h, 51

- BkColor
 - display.c, 29
- call_inits
 - main.c, 49
- control.c, 24
 - control_graph, 24
 - control_init, 24
 - control_refresh, 24
 - control_term, 24
- control.h, 26
 - control_graph, 26
 - control_init, 26
 - control_play, 27
 - control_refresh, 27
 - control_term, 27
- control_func
 - menu_main_struct, 11
- control_graph
 - control.c, 24
 - control.h, 26
- control_init
 - control.c, 24
 - control.h, 26
- control_play
 - control.h, 27
- control_refresh
 - control.c, 24
 - control.h, 27
- control_term
 - control.c, 24
 - control.h, 27
- current_value
 - menu_prop_struct, 13
- disp_clean
 - display.c, 29
 - display.h, 37
- disp_debugging
 - display.c, 30
 - display.h, 37
- disp_drawGrid
 - display.c, 30
 - display.h, 37
- disp_func
 - menu_main_struct, 11
- disp_graph
 - display.c, 31
 - display.h, 39
- disp_init
 - display.c, 32
 - display.h, 40
- disp_load_bitmap
 - display.h, 40
- disp_menu
 - display.c, 33
 - display.h, 40
- disp_off
 - display.c, 33
 - display.h, 41
- disp_prop
 - display.c, 34
 - display.h, 41
- DISP_PROP_W
 - display.c, 29
- disp_refresh
 - display.c, 35
 - display.h, 42
- disp_terminal
 - display.c, 35
 - display.h, 42
- display.c, 28
 - BkColor, 29
 - disp_clean, 29
 - disp_debugging, 30
 - disp_drawGrid, 30
 - disp_graph, 31
 - disp_init, 32
 - disp_menu, 33
 - disp_off, 33
 - disp_prop, 34
 - DISP_PROP_W, 29
 - disp_refresh, 35
 - disp_terminal, 35
 - FgColor, 29
 - freq, 35
 - GR_H, 29
 - GR_HH, 29
 - GR_W, 29
 - GR_WW, 29
 - GR_X, 29
 - GR_Y, 29
 - INFO_X, 29
 - INFO_Y, 29
 - max, 35
 - mid, 35
 - min, 35
 - ShColor, 29
 - SPACELEFT, 29
- display.h, 36
 - disp_clean, 37
 - disp_debugging, 37
 - disp_drawGrid, 37
 - disp_graph, 39
 - disp_init, 40
 - disp_load_bitmap, 40
 - disp_menu, 40
 - disp_off, 41

- disp_prop, 41
- disp_refresh, 42
- disp_terminal, 42
- ENTPR
 - input.c, 44
- FgColor
 - display.c, 29
- freq
 - display.c, 35
- GR_H
 - display.c, 29
- GR_HH
 - display.c, 29
- GR_W
 - display.c, 29
- GR_WW
 - display.c, 29
- GR_X
 - display.c, 29
- GR_Y
 - display.c, 29
- INFO_X
 - display.c, 29
- INFO_Y
 - display.c, 29
- init_ports
 - main.c, 49
- input.c, 44
 - ENTPR, 44
 - input_init, 44
 - input_refresh, 45
 - joy_adc_ready, 46
 - PRESS_LONG, 44
 - switch_off, 45
 - taste, 46
 - taste_long, 46
 - tasten_status, 45
 - voidfunc, 46
- input.h, 47
 - input_init, 47
 - input_refresh, 47
- input_init
 - input.c, 44
 - input.h, 47
- input_refresh
 - input.c, 45
 - input.h, 47
- interrupt_init
 - main.c, 49
- joy_adc_ready
 - input.c, 46
- main
 - main.c, 50
- main.c, 49
 - bk_LED, 49
 - call_inits, 49
 - init_ports, 49
 - interrupt_init, 49
 - main, 50
- main.h, 51
 - bk_LED, 51
- max
 - display.c, 35
- menu.c, 52
 - menu_down, 53
 - menu_init, 53
 - menu_left, 53
 - menu_main_decr, 53
 - menu_main_incr, 53
 - menu_main_set, 53
 - menu_mains, 54
 - menu_prop_set, 54
 - menu_props, 55
 - menu_right, 54
 - menu_start, 54
 - menu_up, 54
- menu.h, 56
 - MENU_ANZ_MAIN, 57
 - MENU_CURRENT_PROP_NR, 57
 - menu_down, 58
 - menu_init, 58
 - menu_left, 58
 - menu_main_decr, 58
 - menu_main_incr, 58
 - menu_main_set, 59
 - menu_mains, 60
 - menu_now, 60
 - MENU_PROP_NOW, 57
 - menu_prop_set, 59
 - menu_props, 60
 - MENU_PROPS_NOW, 58
 - menu_right, 59
 - menu_start, 59
 - menu_up, 59
- MENU_ANZ_MAIN
 - menu.h, 57
- MENU_CURRENT_PROP_NR
 - menu.h, 57
- menu_down
 - menu.c, 53
 - menu.h, 58
- menu_init
 - menu.c, 53

- menu.h, 58
- menu_left
 - menu.c, 53
 - menu.h, 58
- menu_main_decr
 - menu.c, 53
 - menu.h, 58
- menu_main_incr
 - menu.c, 53
 - menu.h, 58
- menu_main_set
 - menu.c, 53
 - menu.h, 59
- menu_main_struct, 11
 - control_func, 11
 - disp_func, 11
 - name, 11
- menu_mains
 - menu.c, 54
 - menu.h, 60
- menu_now
 - menu.h, 60
- menu_prop_main_struct, 12
 - nr_props, 12
 - prop_now, 12
 - properties, 12
- MENU_PROP_NOW
 - menu.h, 57
- menu_prop_set
 - menu.c, 54
 - menu.h, 59
- menu_prop_struct, 13
 - current_value, 13
 - nr_values, 13
 - prop_name, 13
 - set_value, 13
 - value_name, 13
- menu_props
 - menu.c, 55
 - menu.h, 60
- MENU_PROPS_NOW
 - menu.h, 58
- menu_right
 - menu.c, 54
 - menu.h, 59
- menu_start
 - menu.c, 54
 - menu.h, 59
- menu_up
 - menu.c, 54
 - menu.h, 59
- mid
 - display.c, 35
- min
 - display.c, 35
- name
 - menu_main_struct, 11
- nr_props
 - menu_prop_main_struct, 12
- nr_values
 - menu_prop_struct, 13
- PRESS_LONG
 - input.c, 44
- prop_name
 - menu_prop_struct, 13
- prop_now
 - menu_prop_main_struct, 12
- properties
 - menu_prop_main_struct, 12
- set_value
 - menu_prop_struct, 13
- ShColor
 - display.c, 29
- SIGNAL
 - ustimer.c, 68
- SPACELEFT
 - display.c, 29
- switch_off
 - input.c, 45
- taste
 - input.c, 46
- taste_long
 - input.c, 46
- tasten_status
 - input.c, 45
- uart.c, 61
 - usart_baudrate, 61
 - usart_dataBits, 61
 - usart_init, 62
 - usart_off, 62
 - usart_setSync, 62
 - usart_stopBits, 63
- uart.h, 64
 - UART_RX_BUFFER_SIZE, 65
 - UART_TX_BUFFER_SIZE, 65
 - usart_baudrate, 65
 - usart_dataBits, 65
 - usart_init, 65
 - usart_off, 66
 - usart_setSync, 66
 - usart_stopBits, 66
- UART_RX_BUFFER_SIZE
 - uart.h, 65
- UART_TX_BUFFER_SIZE

- uart.h, 65
- us_time
 - ustimer.c, 70
- us_time_get
 - ustimer.c, 68
 - ustimer.h, 71
- us_time_get_difference
 - ustimer.c, 69
 - ustimer.h, 72
- us_time_get_difference_d
 - ustimer.c, 69
 - ustimer.h, 72
- us_timer_init
 - ustimer.c, 69
 - ustimer.h, 72
- US_TIMER_PRESCALER
 - ustimer.c, 68
- usart_baudrate
 - uart.c, 61
 - uart.h, 65
- usart_dataBits
 - uart.c, 61
 - uart.h, 65
- usart_init
 - uart.c, 62
 - uart.h, 65
- usart_off
 - uart.c, 62
 - uart.h, 66
- usart_setSync
 - uart.c, 62
 - uart.h, 66
- usart_stopBits
 - uart.c, 63
 - uart.h, 66
- ustimer.c, 68
 - SIGNAL, 68
 - us_time, 70
 - us_time_get, 68
 - us_time_get_difference, 69
 - us_time_get_difference_d, 69
 - us_timer_init, 69
 - US_TIMER_PRESCALER, 68
 - XTAL, 68
- ustimer.h, 71
 - us_time_get, 71
 - us_time_get_difference, 72
 - us_time_get_difference_d, 72
 - us_timer_init, 72
- value_name
 - menu_prop_struct, 13
- voidfunc
 - input.c, 46
- XTAL
 - ustimer.c, 68