OpenOsci Reference Manual

0.01

Generated by Doxygen 1.4.6

Fri Aug 11 21:47:30 2006

Contents

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

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:

Chapter 3

OpenOsci Data Structure Index

3.1 OpenOsci Data Structures

Here are the data structures with brief descriptions:

Chapter 4

OpenOsci File Index

4.1 OpenOsci File List

Here is a list of all files with brief descriptions:

Chapter 5

OpenOsci Page Index

5.1 OpenOsci Related Pages

Here is a list of all related documentation pages:

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"

```
                                {
    adc_count = 0;
    adc_stop_flag = 0;

    adc_select_channel(channel);    //set the channel
    adc_current_channel = channel;  //save the number of the current channel
    adc_LED();

    ADCSRA = adc_prescaler & 7;     //& 7: only assign first three bits
    //AD Enable, AD Start Conversion, AD Free Running, AD Interrupt Enable
    adc_starttime = us_time_get();
    ADCSRA |= (1<<ADEN) | (1<<ADSC) | (1<<ADFR);//interrupt method: | (1<<ADIE);

    if(adc_channels == 1) adc_single_channel();
    else adc_multi_channels();

    //stop adc again
    adc_stop();
}
```

7.1.1.2 void adc_LED (void)

handels the LED output

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

7.1.1.3 void adc_multi_channels (void)

```
                                {
    //fast measurement
    loop_until_bit_is_set(ADCSRA,ADIF);
    ADCSRA |= (1<<ADIF);
    adc[0] = ADCH;
    for(uint16_t x=0; x < ADC_BUF_SIZE; x++){
        adc_next_channel();
        loop_until_bit_is_set(ADCSRA,ADIF);
        adc[x] = ADCH;
        ADCSRA |= (1<<ADIF);

        loop_until_bit_is_set(ADCSRA,ADIF);
        ADCSRA |= (1<<ADIF);
        loop_until_bit_is_set(ADCSRA,ADIF);
        ADCSRA |= (1<<ADIF);
    }
}
```

7.1.1.4 void adc_next_channel (void)

switches to the next channel

```
                              {
    uint8_t new = adc_current_channel;

    new++;
    if(new > adc_channels) new = 1;
    adc_select_channel(new);
}
```

7.1.1.5 void adc_off (void)

switches the ADC off

```
                   {
    PORTA = PORTA & 15; //sets the upper 4 bits to zero

    ADCSRA = 0;
}
```

7.1.1.6 void adc_select_channel (uint8_t *channel*)

select "channel"

```
                                           {
    if(channel == adc_current_channel) return;
    adc_current_channel = channel;

    //Input Channel selection; obersten beiden bits wÃd’hlen externe Referenz, wenn 0
    //=> ADMUX entspricht Channel
    ADMUX = (channel-1) + (1<<ADLAR) + (1<<REFS0);
    //ADMUX |= (1<<ADLAR);  //left-align bits to allow to read the 8bit-value from ADCH
}
```

7.1.1.7 void adc_set_nr_channels (int8_t *nr*)

sets the nr of active channels

```
                                      {
    adc_channels = nr;
}
```

7.1.1.8 void adc_set_presc (int8_t *presc*)

sets the prescaler

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

7.1.1.9 void adc_single_channel (void)

```
                                {
    //fast measurement
    loop_until_bit_is_set(ADCSRA,ADIF);
    ADCSRA |= (1<<ADIF);
    adc[0] = ADCH;
    for(uint16_t x=0; x < ADC_BUF_SIZE; x++){
        loop_until_bit_is_set(ADCSRA,ADIF);
        adc[x] = ADCH;
//      adc_next_channel();
        ADCSRA |= (1<<ADIF);
    }
}
```

7.1.1.10 void adc_stop (void)

stops the ADC

switches off the free running mode.

Todo

still needed?

```
                      {
    ADCSRA &= ~(1<<ADSC) & ~(1<<ADFR) & ~(1<<ADIE);
    adc_stop_flag = 1;

    adc_duration = us_time_get_difference_d(adc_starttime) / ((double)ADC_BUF_SIZE / (double)adc_chann
}
```

7.1.1.11 int8_t adc_stopped (void)

checks, whether the ADC has stopped

Todo

still needed?

```
                          {
        return adc_stop_flag;
  }
```

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"

```
                                {
    adc_count = 0;
    adc_stop_flag = 0;

    adc_select_channel(channel);    //set the channel
    adc_current_channel = channel;  //save the number of the current channel
    adc_LED();

    ADCSRA = adc_prescaler & 7;     //& 7: only assign first three bits
    //AD Enable, AD Start Conversion, AD Free Running, AD Interrupt Enable
    adc_starttime = us_time_get();
    ADCSRA |= (1<<ADEN) | (1<<ADSC) | (1<<ADFR);//interrupt method: | (1<<ADIE);

    if(adc_channels == 1) adc_single_channel();
    else adc_multi_channels();

    //stop adc again
    adc_stop();
}
```

7.2.3.2 void adc_LED (void)

handels the LED output

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

7.2.3.3 void adc_next_channel (void)

switches to the next channel

```
                         {
    uint8_t new = adc_current_channel;

    new++;
    if(new > adc_channels) new = 1;
    adc_select_channel(new);
}
```

7.2.3.4 void adc_off (void)

switches the ADC off

```
                 {
    PORTA = PORTA & 15; //sets the upper 4 bits to zero

    ADCSRA = 0;
}
```

7.2.3.5 void adc_select_channel (uint8_t *channel*)

select "channel"

```
                                     {
    if(channel == adc_current_channel) return;
    adc_current_channel = channel;

    //Input Channel selection; obersten beiden bits wÃd’hlen externe Referenz, wenn 0
    //=> ADMUX entspricht Channel
    ADMUX = (channel-1) + (1<<ADLAR) + (1<<REFS0);
    //ADMUX |= (1<<ADLAR);  //left-align bits to allow to read the 8bit-value from ADCH
}
```

7.2.3.6 void adc_set_nr_channels (int8_t *nr*)

sets the nr of active channels

```
                               {
    adc_channels = nr;
}
```

7.2.3.7 void adc_set_presc (int8_t *presc*)

sets the prescaler

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

7.2.3.8 void adc_stop (void)

stops the ADC

switches off the free running mode.

Todo
still needed?

```
                         {
       ADCSRA &= ~(1<<ADSC) & ~(1<<ADFR) & ~(1<<ADIE);
       adc_stop_flag = 1;

       adc_duration = us_time_get_difference_d(adc_starttime) / ((double)ADC_BUF_SIZE / (double)adc_chan
   }
```

7.2.3.9 int8_t adc_stopped (void)

checks, whether the ADC has stopped

Todo
still needed?

```
                            {
       return adc_stop_flag;
   }
```

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"

```
                         {
       //usart_off();
}
```

7.3.1.2 void control_init (void)

initialise control

```
                        {
}
```

7.3.1.3 void control_refresh (void)

refresh control

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

7.3.1.4 void control_term (void)

the control function for mode "term"

```
                        {
    //usart_init();
}
```

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"

```
                           {
    //usart_off();
}
```

7.4.2.2 void control_init (void)

initialise control

```
                          {
 }
```

7.4.2.3 void control_play (void)

the control function for mode "play"

7.4.2.4 void control_refresh (void)

refresh control

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

7.4.2.5 void control_term (void)

the control function for mode "term"

```
                          {
    //usart_init();
 }
```

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

```
                      {
        glcdSetAddr(0,0, 131, 175);   // set RAM access pointer of display
        //glcdSetBkColor(BkColor);

        //width and height also defined in header
```

```
    for(uint8_t x = 0; x < 132; x++){
        for(uint8_t y = 0; y < 176; y++){
            glcdPutPixel(BkColor);
        }
    }
}
```

7.5.2.2 void disp_debugging (void)

displays a debugging screen

```
                          {
    glcdSetColors(WHITE,BkColor);

    static uint16_t count = 0;
    int8_t y = 30;
    glcdMoveTo(20,y+=11); printf(" Count: %d    \n",count++);
    glcdMoveTo(20,y+=11); printf(" Menue: %d    \n",(uint16_t)ADCSRA);
    glcdMoveTo(20,y+=11); printf(" Presc: %d    \n",(uint16_t)(ADCSRA & ADIF));
    glcdMoveTo(20,y+=11); printf(" spi_control: %d   \n",SPCR);
    glcdMoveTo(20,y+=11); printf(" spi_status: %d   \n",SPSR);
}
```

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.

```
                                                                          {
    //draw grid
    glcdSetColors(ShColor, BkColor);

    glcdLine(GR_X+xaxis, GR_Y, GR_X+xaxis, GR_Y+GR_HH); //horizontal
    glcdLine(GR_X, GR_Y+yaxis, GR_X+GR_WW, GR_Y+yaxis); //vertical

    /* x and y need to be signed, because for small xaxis and yaxis, the start
     * can be negative */
    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+yax
    for(int16_t x = GR_X+xaxis-GR_W; x >= GR_X      ; x-=GR_W) glcdLine(x, GR_Y-3+yaxis, x, GR_Y+3+yax

    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,
    for(int16_t y = GR_Y+yaxis-GR_H; y >= GR_Y      ; y-=GR_H) glcdLine(GR_X-3+xaxis, y, GR_X+3+xaxis,


    //replot data
    //init
    static uint8_t buffer[4][GR_HH];
    uint8_t old_coord[4] = {buffer[0][0], buffer[1][0], buffer[2][0], buffer[3][0]};
    static uint8_t old_adc_nr_channels = 1;
    for(uint8_t ch=0; ch < adc_channels; ch++)
        buffer[ch][0] = toDraw[ch] >> 1;
    glcdSetBkColor(BkColor);
    //start
    for(uint8_t y = 1; y < GR_HH; y++){
        //erase old lines
        glcdSetFgColor(BkColor);
        for(uint8_t ch=0; ch < old_adc_nr_channels; ch++){ //old_number!!!
            glcdLine(GR_X + old_coord[ch], y-1 + GR_Y, GR_X + buffer[ch][y], y + GR_Y);
```

```
                old_coord[ch] = buffer[ch][y];
                buffer[ch][y] = toDraw[y*adc_channels+ch] >> 1;
            }
            //draw new lines
            for(uint8_t ch=0; ch < adc_channels; ch++){
                switch(ch){
                    case 0:
                        glcdSetFgColor(WHITE);
                        break;
                    case 1:
                        glcdSetFgColor(FgColor);
                        break;
                    case 2:
                        glcdSetFgColor(GREEN);
                        break;
                    case 3:
                        glcdSetFgColor(RED);
                        break;
                }
                glcdLine(GR_X + buffer[ch][y-1], y-1 + GR_Y, GR_X + buffer[ch][y], y + GR_Y);
            }
        }
    old_adc_nr_channels = adc_channels;

    //info box
    #define INFO_X GR_X+GR_WW-55
    #define INFO_Y GR_Y+GR_HH-25
    glcdSetColors(FgColor, BkColor);
    int y = INFO_Y - 9;
    glcdMoveTo(INFO_X+2, y+=11); printf("td:");
    glcdMoveTo(INFO_X+2, y+=11); printf("fq:");
    y = INFO_Y - 9;
    uint32_t time = adc_duration*GR_H;
    if(time < 10000){
        glcdMoveTo(INFO_X+15, y+=11); printf(" %4dus ",(int16_t)(time));
    }else{
        glcdMoveTo(INFO_X+15, y+=11); printf(" %4dms ",(int16_t)(time/1000));
    }
    if(freq[0] < 10000){
        glcdMoveTo(INFO_X+15, y+=11); printf(" %4dHz ",(int16_t)(freq[0]));
    }else{
        glcdMoveTo(INFO_X+15, y+=11); printf(" %3dkHz ",(int16_t)(freq[0]/1000));
    }
}
```

7.5.2.4 void disp_graph (void)

displays the osci graph

```
                      {
    if(adc_stopped() == 0) return;

    #define GR_W    20  //width of sub ... dash? unterteilung
    #define GR_H    20  //height of ???
    #define GR_WW   128 //width
    #define GR_HH   144 //height
    #define GR_X    2   //x-coord: upper-left
    #define GR_Y    15  //y-coord: upper-left


    //trigger
    uint16_t trigger_shift = 72*adc_channels;
    uint16_t GR_TRIG_BEFORE = ADC_BUF_SIZE - (GR_HH*adc_channels) + trigger_shift;

    int8_t trigger_old = 0;
    int8_t trigger_new = 0;
    int8_t trigger_highest = 0;
    uint16_t offset = 0;

    for(uint16_t x = trigger_shift; x < GR_TRIG_BEFORE; x+=adc_channels){
        trigger_old = adc[x] >> 1;
        trigger_new = adc[x + 3*adc_channels] >> 1;
        if((trigger_new-trigger_old) > trigger_highest){
            offset = x - trigger_shift;
            trigger_highest = trigger_new-trigger_old;
        }
    }

    min[0] = 255; min[1] = 255; min[2] = 255; min[3] = 255;
    max[0] = 0; max[1] = 0; max[2] = 0; max[3] = 0;
    mid[0] = 0; mid[1] = 0; mid[2] = 0; mid[3] = 0;
    freq[0] = 0; freq[1] = 0; freq[2] = 0; freq[3] = 0;
    for(uint8_t ch = 0; ch < adc_channels; ch++){
        for(uint16_t x = ch; x < ADC_BUF_SIZE; x+=adc_channels){
            if(adc[x] < min[ch]) min[ch] = adc[x];
            if(adc[x] > max[ch]) max[ch] = adc[x];
        }
    }
    for(uint8_t x = 0; x < adc_channels; x++) mid[x] = (max[x]-min[x])/2  + min[x];


    for(uint8_t ch = 0; ch < adc_channels; ch++){
        uint16_t old_pos = 0;
        int8_t under;       //currently under mid? 0 for no, 1 for yes
        int8_t old_under;
        double period = 0.0;

        if(adc[ch] < mid[ch]){ under = 0; old_under = 0; }
            else { under = 1; old_under = 1; }

        for(uint16_t x = ch; x < ADC_BUF_SIZE; x+=adc_channels){
            if(adc[x] > mid[ch]) under = 0;
            if(adc[x] < mid[ch]) under = 1;

            if(old_under == 1 && under == 0){
                period = (double)(x - old_pos)*adc_duration /adc_channels/ 1000000.0;
                if(old_pos != 0  &&  period != 0.0){
                    if(freq[ch] == 0.0) freq[ch] = 1.0 / period;
                    else                freq[ch] = 0.8*freq[ch] + 0.2/period;
                }
                old_pos = x;
            }

            old_under = under;
        }

    }


    //draw in the display
    disp_drawGrid(&adc[offset], 64, trigger_shift/adc_channels);

    adc_init(1);
}
```

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.

```
                    {
    //**** Hardware ****
    #ifndef USE_AUTOINIT
        glcdDisplayInit();
    #endif

    #ifdef DISP_SPLASH
        disp_load_bitmap();
    #else
        disp_clean();
    #endif

    glcd_Flags.AutoLineFeed = 0;
    glcdSelectFont(f8x11, 0); // font is stored in FLASH, thus no need for own read callback
    fdevopen(glcdPut,NULL,0);

    //window for text-output
    glcd_Window.X1 = 5;
    glcd_Window.X2 = 110;
    glcd_Window.Y1 = 16;
    glcd_Window.Y2 = 162;
}
```

7.5.2.6 void disp_menu (void)

displays the menu

Refreshes the menu bar when the menu has changed.

```
                    {
    static int8_t main_old = 0;

    //*** left ***
    if(menu_now != main_old){
        main_old = menu_now;
        disp_clean();


        glcdFillRect(0,0,132,13,ShColor); //shadow color for menu bg
        #define SPACELEFT 6
        for(uint8_t x=0; x < MENU_ANZ_MAIN; x++){
            if(menu_now == (x+1)){
                glcdFillRect(SPACELEFT+x*40,1, SPACELEFT+(x+1)*40,12, BkColor);
                glcdSetColors(FgColor, BkColor);
            }else{
                glcdSetColors(FgColor, ShColor);
            }
            uint8_t w = glcdCharsWidth(menu_mains[x].name, 0);
            glcdMoveTo(SPACELEFT+21+x*40 - w/2, 2);
            printf(menu_mains[x].name);
        }
    }
}
```

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.

```
                    {
    bk_LED(1);
    //for(uint32_t x=0; x < 5000000; x++) asm("nop");

    //clean
    glcdSetAddr(0,0, 131, 175);   // set RAM access pointer of display
    //width and height also defined in header
    for(uint8_t x = 0; x < 132; x++){
        for(uint8_t y = 0; y < 176; y++){
            glcdPutPixel(WHITE);
        }
    }
    for(uint32_t x=0; x < 500000; x++) asm("nop");
    glcdDisplayOff();
    glcdWait(100);
}
```

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.

```
                    {
    static int8_t main_old = 0;
    static int8_t prop_old = 0;
    static int8_t value_old = 0;

    if( menu_now != main_old ||
        MENU_CURRENT_PROP_NR != prop_old ||
        MENU_PROP_NOW.current_value != value_old){

        main_old = menu_now;
        prop_old = MENU_CURRENT_PROP_NR;
        value_old = MENU_PROP_NOW.current_value;

        #define DISP_PROP_W 33
        glcdFillRect(0, 162, 132, 176, ShColor);
        for(uint8_t x=0; x < 3; x++){
            if(x == 1){
                glcdFillRect(SPACELEFT+x*DISP_PROP_W,163, SPACELEFT+(x+1)*DISP_PROP_W,174, BkColor);
                glcdSetColors(FgColor, BkColor);
            }else{
                glcdSetColors(FgColor, ShColor);
            }
            int8_t current_value = value_old-1+x;
            char *label;
            if(current_value < 1)
                label = "=>";
            else if(current_value > MENU_PROP_NOW.nr_values)
                label = "<=";
            else
                label = MENU_PROP_NOW.value_name[current_value-1];
            uint8_t w = glcdCharsWidth(label, 0);
            glcdMoveTo(SPACELEFT+21+x*DISP_PROP_W - w, 164);
            printf(label);

            //prop-name
            w = glcdCharsWidth(MENU_PROP_NOW.prop_name, 0);
            glcdFillRect(132 - 2*SPACELEFT - w, 163, 130, 174, BkColor);
            glcdSetColors(FgColor, BkColor);
            glcdMoveTo(132 - SPACELEFT - w, 164);
            printf(MENU_PROP_NOW.prop_name);
        }
    }
}
```

7.5.2.9 void disp_refresh (void)

refreshes the display

```
                         {
    disp_menu();
    menu_mains[menu_now-1].disp_func();
    disp_prop();
}
```

7.5.2.10 void disp_terminal (void)

displays a terminal

The terminal for the serial interface.

Todo

Testing necessary!

```
                          {
//  if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162) glcdMoveTo(5,16);
    if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162 || glcd_Cursor.X > 100) glcdMove
    glcdSetColors(WHITE,BkColor);

    uint16_t rx_data;
    while(! ((rx_data=uart_getc()) & UART_NO_DATA)){
        //glcdPut(rx_data & 255);
        glcdDrawChar((uint8_t)rx_data);
    }
    while(! ((rx_data=uart1_getc()) & UART_NO_DATA)){
        //glcdPut(rx_data & 255);
        glcdDrawChar((uint8_t)rx_data);
    }

//  if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162 || glcd_Cursor.X > 120) glcdMove
    //printf("%c",(rx_data & 255));

    //glcdWait(100);
}
```

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

```
                     {
    glcdSetAddr(0,0, 131, 175);   // set RAM access pointer of display
    //glcdSetBkColor(BkColor);

    //width and height also defined in header
    for(uint8_t x = 0; x < 132; x++){
        for(uint8_t y = 0; y < 176; y++){
            glcdPutPixel(BkColor);
        }
    }
}
```

7.6.2.2 void disp_debugging (void)

displays a debugging screen

```
                          {
    glcdSetColors(WHITE,BkColor);

    static uint16_t count = 0;
    int8_t y = 30;
    glcdMoveTo(20,y+=11); printf(" Count: %d    \n",count++);
    glcdMoveTo(20,y+=11); printf(" Menue: %d    \n",(uint16_t)ADCSRA);
    glcdMoveTo(20,y+=11); printf(" Presc: %d    \n",(uint16_t)(ADCSRA & ADIF));
    glcdMoveTo(20,y+=11); printf(" spi_control: %d   \n",SPCR);
    glcdMoveTo(20,y+=11); printf(" spi_status: %d   \n",SPSR);
}
```

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.

```
                                                                           {
    //draw grid
    glcdSetColors(ShColor, BkColor);

    glcdLine(GR_X+xaxis, GR_Y, GR_X+xaxis, GR_Y+GR_HH); //horizontal
    glcdLine(GR_X, GR_Y+yaxis, GR_X+GR_WW, GR_Y+yaxis); //vertical

    /* x and y need to be signed, because for small xaxis and yaxis, the start
     * can be negative */
    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+yax
    for(int16_t x = GR_X+xaxis-GR_W; x >= GR_X      ; x-=GR_W) glcdLine(x, GR_Y-3+yaxis, x, GR_Y+3+yax

    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,
    for(int16_t y = GR_Y+yaxis-GR_H; y >= GR_Y      ; y-=GR_H) glcdLine(GR_X-3+xaxis, y, GR_X+3+xaxis,


    //replot data
    //init
    static uint8_t buffer[4][GR_HH];
    uint8_t old_coord[4] = {buffer[0][0], buffer[1][0], buffer[2][0], buffer[3][0]};
    static uint8_t old_adc_nr_channels = 1;
    for(uint8_t ch=0; ch < adc_channels; ch++)
        buffer[ch][0] = toDraw[ch] >> 1;
    glcdSetBkColor(BkColor);
    //start
    for(uint8_t y = 1; y < GR_HH; y++){
        //erase old lines
        glcdSetFgColor(BkColor);
        for(uint8_t ch=0; ch < old_adc_nr_channels; ch++){ //old_number!!!
            glcdLine(GR_X + old_coord[ch], y-1 + GR_Y, GR_X + buffer[ch][y], y + GR_Y);
            old_coord[ch] = buffer[ch][y];
            buffer[ch][y] = toDraw[y*adc_channels+ch] >> 1;
        }
        //draw new lines
        for(uint8_t ch=0; ch < adc_channels; ch++){
            switch(ch){
                case 0:
                    glcdSetFgColor(WHITE);
                    break;
                case 1:
                    glcdSetFgColor(FgColor);
                    break;
                case 2:
                    glcdSetFgColor(GREEN);
                    break;
                case 3:
                    glcdSetFgColor(RED);
                    break;
            }
            glcdLine(GR_X + buffer[ch][y-1], y-1 + GR_Y, GR_X + buffer[ch][y], y + GR_Y);
        }
    }
    old_adc_nr_channels = adc_channels;

    //info box
    #define INFO_X GR_X+GR_WW-55
    #define INFO_Y GR_Y+GR_HH-25
    glcdSetColors(FgColor, BkColor);
    int y = INFO_Y - 9;
    glcdMoveTo(INFO_X+2, y+=11); printf("td:");
    glcdMoveTo(INFO_X+2, y+=11); printf("fq:");
    y = INFO_Y - 9;
    uint32_t time = adc_duration*GR_H;
    if(time < 10000){
        glcdMoveTo(INFO_X+15, y+=11); printf(" %4dus ",(int16_t)(time));
    }else{
        glcdMoveTo(INFO_X+15, y+=11); printf(" %4dms ",(int16_t)(time/1000));
    }
    if(freq[0] < 10000){
        glcdMoveTo(INFO_X+15, y+=11); printf(" %4dHz ",(int16_t)(freq[0]));
    }else{
        glcdMoveTo(INFO_X+15, y+=11); printf(" %3dkHz ",(int16_t)(freq[0]/1000));
    }
}
```

7.6.2.4 void disp_graph (void)

displays the osci graph

```
                    {
    if(adc_stopped() == 0) return;

    #define GR_W    20  //width of sub ... dash? unterteilung
    #define GR_H    20  //height of ???
    #define GR_WW   128 //width
    #define GR_HH   144 //height
    #define GR_X    2   //x-coord: upper-left
    #define GR_Y    15  //y-coord: upper-left


    //trigger
    uint16_t trigger_shift = 72*adc_channels;
    uint16_t GR_TRIG_BEFORE = ADC_BUF_SIZE - (GR_HH*adc_channels) + trigger_shift;

    int8_t trigger_old = 0;
    int8_t trigger_new = 0;
    int8_t trigger_highest = 0;
    uint16_t offset = 0;

    for(uint16_t x = trigger_shift; x < GR_TRIG_BEFORE; x+=adc_channels){
        trigger_old = adc[x] >> 1;
        trigger_new = adc[x + 3*adc_channels] >> 1;
        if((trigger_new-trigger_old) > trigger_highest){
            offset = x - trigger_shift;
            trigger_highest = trigger_new-trigger_old;
        }
    }

    min[0] = 255; min[1] = 255; min[2] = 255; min[3] = 255;
    max[0] = 0; max[1] = 0; max[2] = 0; max[3] = 0;
    mid[0] = 0; mid[1] = 0; mid[2] = 0; mid[3] = 0;
    freq[0] = 0; freq[1] = 0; freq[2] = 0; freq[3] = 0;
    for(uint8_t ch = 0; ch < adc_channels; ch++){
        for(uint16_t x = ch; x < ADC_BUF_SIZE; x+=adc_channels){
            if(adc[x] < min[ch]) min[ch] = adc[x];
            if(adc[x] > max[ch]) max[ch] = adc[x];
        }
    }
    for(uint8_t x = 0; x < adc_channels; x++) mid[x] = (max[x]-min[x])/2  + min[x];


    for(uint8_t ch = 0; ch < adc_channels; ch++){
        uint16_t old_pos = 0;
        int8_t under;       //currently under mid? 0 for no, 1 for yes
        int8_t old_under;
        double period = 0.0;

        if(adc[ch] < mid[ch]){ under = 0; old_under = 0; }
            else { under = 1; old_under = 1; }

        for(uint16_t x = ch; x < ADC_BUF_SIZE; x+=adc_channels){
            if(adc[x] > mid[ch]) under = 0;
            if(adc[x] < mid[ch]) under = 1;

            if(old_under == 1 && under == 0){
                period = (double)(x - old_pos)*adc_duration /adc_channels/ 1000000.0;
                if(old_pos != 0  &&  period != 0.0){
                    if(freq[ch] == 0.0) freq[ch] = 1.0 / period;
                    else                freq[ch] = 0.8*freq[ch] + 0.2/period;
                }
                old_pos = x;
            }

            old_under = under;
        }

    }


    //draw in the display
    disp_drawGrid(&adc[offset], 64, trigger_shift/adc_channels);

    adc_init(1);
}
```

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.

```
                    {
    //**** Hardware ****
    #ifndef USE_AUTOINIT
        glcdDisplayInit();
    #endif

    #ifdef DISP_SPLASH
        disp_load_bitmap();
    #else
        disp_clean();
    #endif

    glcd_Flags.AutoLineFeed = 0;
    glcdSelectFont(f8x11, 0); // font is stored in FLASH, thus no need for own read callback
    fdevopen(glcdPut,NULL,0);

    //window for text-output
    glcd_Window.X1 = 5;
    glcd_Window.X2 = 110;
    glcd_Window.Y1 = 16;
    glcd_Window.Y2 = 162;
}
```

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.

```
                    {
    static int8_t main_old = 0;

    //*** left ***
    if(menu_now != main_old){
        main_old = menu_now;
        disp_clean();


        glcdFillRect(0,0,132,13,ShColor); //shadow color for menu bg
        #define SPACELEFT 6
        for(uint8_t x=0; x < MENU_ANZ_MAIN; x++){
            if(menu_now == (x+1)){
                glcdFillRect(SPACELEFT+x*40,1, SPACELEFT+(x+1)*40,12, BkColor);
                glcdSetColors(FgColor, BkColor);
            }else{
                glcdSetColors(FgColor, ShColor);
            }
            uint8_t w = glcdCharsWidth(menu_mains[x].name, 0);
            glcdMoveTo(SPACELEFT+21+x*40 - w/2, 2);
            printf(menu_mains[x].name);
        }
    }
}
```

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.

```
                   {
    bk_LED(1);
    //for(uint32_t x=0; x < 5000000; x++) asm("nop");

    //clean
    glcdSetAddr(0,0, 131, 175);   // set RAM access pointer of display
    //width and height also defined in header
    for(uint8_t x = 0; x < 132; x++){
        for(uint8_t y = 0; y < 176; y++){
            glcdPutPixel(WHITE);
        }
    }
    for(uint32_t x=0; x < 500000; x++) asm("nop");
    glcdDisplayOff();
    glcdWait(100);
}
```

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.

```
                    {
    static int8_t main_old = 0;
    static int8_t prop_old = 0;
    static int8_t value_old = 0;

    if( menu_now != main_old ||
        MENU_CURRENT_PROP_NR != prop_old ||
        MENU_PROP_NOW.current_value != value_old){

        main_old = menu_now;
        prop_old = MENU_CURRENT_PROP_NR;
        value_old = MENU_PROP_NOW.current_value;

        #define DISP_PROP_W 33
        glcdFillRect(0, 162, 132, 176, ShColor);
        for(uint8_t x=0; x < 3; x++){
            if(x == 1){
                glcdFillRect(SPACELEFT+x*DISP_PROP_W,163, SPACELEFT+(x+1)*DISP_PROP_W,174, BkColor);
                glcdSetColors(FgColor, BkColor);
            }else{
                glcdSetColors(FgColor, ShColor);
            }
            int8_t current_value = value_old-1+x;
            char *label;
            if(current_value < 1)
                label = "=>";
            else if(current_value > MENU_PROP_NOW.nr_values)
                label = "<=";
            else
                label = MENU_PROP_NOW.value_name[current_value-1];
            uint8_t w = glcdCharsWidth(label, 0);
            glcdMoveTo(SPACELEFT+21+x*DISP_PROP_W - w, 164);
            printf(label);

            //prop-name
            w = glcdCharsWidth(MENU_PROP_NOW.prop_name, 0);
            glcdFillRect(132 - 2*SPACELEFT - w, 163, 130, 174, BkColor);
            glcdSetColors(FgColor, BkColor);
            glcdMoveTo(132 - SPACELEFT - w, 164);
            printf(MENU_PROP_NOW.prop_name);
        }
    }
}
```

7.6.2.10 void disp_refresh (void)

refreshes the display

```
                       {
    disp_menu();
    menu_mains[menu_now-1].disp_func();
    disp_prop();
}
```

7.6.2.11 void disp_terminal (void)

displays a terminal

The terminal for the serial interface.

Todo

Testing necessary!

```
                         {
//  if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162) glcdMoveTo(5,16);
    if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162 || glcd_Cursor.X > 100) glcdMove
    glcdSetColors(WHITE,BkColor);

    uint16_t rx_data;
    while(! ((rx_data=uart_getc()) & UART_NO_DATA)){
        //glcdPut(rx_data & 255);
        glcdDrawChar((uint8_t)rx_data);
    }
    while(! ((rx_data=uart1_getc()) & UART_NO_DATA)){
        //glcdPut(rx_data & 255);
        glcdDrawChar((uint8_t)rx_data);
    }

//  if(glcd_Cursor.X < 5 || glcd_Cursor.Y < 16 || glcd_Cursor.Y > 162 || glcd_Cursor.X > 120) glcdMove
    //printf("%c",(rx_data & 255));

    //glcdWait(100);
}
```

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

```
                        {
       taste[0] = &menu_main_incr;
       taste[1] = &menu_start;
       taste[2] = &menu_left;
       taste[3] = &menu_up;
       taste[4] = &menu_down;
       taste[5] = &menu_right;
       taste[6] = &voidfunc;
       taste[7] = &voidfunc;
       taste_long[0] = &switch_off;
       taste_long[1] = &voidfunc;
       taste_long[2] = &voidfunc;
```

```
    taste_long[3] = &voidfunc;
    taste_long[4] = &voidfunc;
    taste_long[5] = &voidfunc;
    taste_long[6] = &voidfunc;
    taste_long[7] = &voidfunc;
}
```

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?

```
                        {
    tasten_status();
}
```

7.7.2.3 void switch_off (void)

```
                     {
    disp_off();
    PORTA = 0;
}
```

7.7.2.4 void tasten_status (void)

```
                         {
    static uint8_t taster = 255;
    static uint32_t last_down = 0;

    uint8_t taster_neu = PINC;
    uint8_t diff = taster_neu ^ taster; //contains changes; "^" is xor
    diff &= taster_neu; //on_release; substitute taster_neu to taster to get on_push

    //if(count < 65535) count++;
    double timeFromLastDown = us_time_get_difference_d(last_down);
    #define ENTPR 1000
    #define PRESS_LONG 1000000
    if(timeFromLastDown > ENTPR){   //entprellen: sicherstellen, dass bestimmte Zeit vergangen ist
        if      (bit_is_set(diff,0)) taste[0]();     //Taste 1
        else if(bit_is_set(diff,1)) taste[1]();     //TASTE 2
        else if(bit_is_set(diff,2)) taste[2]();     //TASTE 3
        else if(bit_is_set(diff,3)) taste[3]();     //TASTE 4
        else if(bit_is_set(diff,4)) taste[4]();     //TASTE 5
        else if(bit_is_set(diff,5)) taste[5]();     //TASTE 6
        else if(bit_is_set(diff,6)) taste[6]();     //TASTE 7
        else if(bit_is_set(diff,7)) taste[7]();     //TASTE 8
    }
    if(taster != taster_neu) last_down = us_time_get();     //entpreltime neu setzen
    timeFromLastDown = us_time_get_difference_d(last_down);
    if(timeFromLastDown > PRESS_LONG){
        if     (bit_is_clear(taster_neu,0) && bit_is_clear(diff,0)) taste_long[0]();
        else if(bit_is_clear(taster_neu,1) && bit_is_clear(diff,1)) taste_long[1]();
        else if(bit_is_clear(taster_neu,2) && bit_is_clear(diff,2)) taste_long[2]();
        else if(bit_is_clear(taster_neu,3) && bit_is_clear(diff,3)) taste_long[3]();
        else if(bit_is_clear(taster_neu,4) && bit_is_clear(diff,4)) taste_long[4]();
        else if(bit_is_clear(taster_neu,5) && bit_is_clear(diff,5)) taste_long[5]();
        else if(bit_is_clear(taster_neu,6) && bit_is_clear(diff,6)) taste_long[6]();
        else if(bit_is_clear(taster_neu,7) && bit_is_clear(diff,7)) taste_long[7]();
    }
    taster = taster_neu;
}
```

7.7.2.5 void voidfunc (void)

```
{}
```

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

```
                    {
    taste[0] = &menu_main_incr;
    taste[1] = &menu_start;
    taste[2] = &menu_left;
    taste[3] = &menu_up;
    taste[4] = &menu_down;
    taste[5] = &menu_right;
    taste[6] = &voidfunc;
    taste[7] = &voidfunc;
    taste_long[0] = &switch_off;
    taste_long[1] = &voidfunc;
    taste_long[2] = &voidfunc;
    taste_long[3] = &voidfunc;
    taste_long[4] = &voidfunc;
    taste_long[5] = &voidfunc;
    taste_long[6] = &voidfunc;
    taste_long[7] = &voidfunc;
}
```

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?

```
                        {
    tasten_status();
}
```

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.

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

7.9.1.2 void call_inits (void)

```
{
    init_ports();
    menu_init();
    control_init();
    input_init();
    adc_init(0);
    disp_init();
    usart_init();
    interrupt_init();
}
```

7.9.1.3 void init_ports (void)

```
{
    DDRA=255;   //OUTPUT
    PORTA=255;  //on

    DDRC = 0;       //Input
    PORTC = 255;    //pull ups
    DDRD = (1<<PD3);    //PD3 Output
    PORTD = 255;
    DDRE = 0;   //input
    PORTE = 255;
}
```

7.9.1.4 void interrupt_init (void)

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

```
{
    //timer 2 in fast PWM for backlight
    PORTB &= ~(1<<PB7);
    DDRB |= (1<<PB7);   //OC2 pin is output
    TCCR2 = (1<<WGM21) | (1<<WGM20) | (1<<COM21) | (1<<CS20); //no prescaler
    TCNT2 = 0;
    OCR2 = 108; //max 120

    //timer 3 for us_timer
    us_timer_init();

    // enable interrupts
    sei();
}
```

7.9.1.5 int main (void)

Main, with test-code.

```
{
    //#define cnt  1
    DDRE = (1<<PE2); //sets data direction for xck0 to output

    call_inits();
    #ifdef DISP_SPLASH
        for(uint32_t x=0; x < 5000000; x++) asm("nop");
    #endif

    while(1==1){
        //uart_empfang();
        control_refresh();
        disp_refresh();
        input_refresh();
    }
}
```

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.

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

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".

```
                    {
    if(MENU_PROP_NOW.current_value > 1) MENU_PROP_NOW.current_value--;
    MENU_PROP_NOW.set_value(MENU_PROP_NOW.current_value);
}
```

7.11.1.2 void menu_init (void)

initialise menu

```
                    {
    menu_main_set(1);
    menu_prop_set(1);
}
```

7.11.1.3 void menu_left (void)

Handels pressed signal for button "left".

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

7.11.1.4 void menu_main_decr (void)

menu before

```
                         {
    if(menu_now > 1)    menu_main_set(menu_now - 1);
    else                menu_main_set(MENU_ANZ_MAIN);
}
```

7.11.1.5 void menu_main_incr (void)

next menu

```
                         {
    if(menu_now < MENU_ANZ_MAIN)    menu_main_set(menu_now + 1);
    else                            menu_main_set(1);
}
```

7.11.1.6 void menu_main_set (int8_t *nr*)

set to menu "nr"

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

7.11.1.7 void menu_prop_set (int8_t *nr*)

set prop in argument active

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

7.11.1.8 void menu_right (void)

Handels pressed signal for button "right".

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

7.11.1.9 void menu_start (void)

Handels pressed signal for button "start".

```
{}
```

7.11.1.10 void menu_up (void)

Handels pressed signal for button "up".

```
                     {
    if(MENU_PROP_NOW.current_value < MENU_PROP_NOW.nr_values)
        MENU_PROP_NOW.current_value++;
    MENU_PROP_NOW.set_value(MENU_PROP_NOW.current_value);
}
```

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".

```
                        {
       if(MENU_PROP_NOW.current_value > 1) MENU_PROP_NOW.current_value--;
       MENU_PROP_NOW.set_value(MENU_PROP_NOW.current_value);
   }
```

7.12.3.2 volatile void menu_init (void)

initialise menu

```
                        {
       menu_main_set(1);
       menu_prop_set(1);
   }
```

7.12.3.3 void menu_left (void)

Handels pressed signal for button "left".

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

7.12.3.4 void menu_main_decr (void)

menu before

```
                             {
       if(menu_now > 1)    menu_main_set(menu_now - 1);
       else                menu_main_set(MENU_ANZ_MAIN);
   }
```

7.12.3.5 void menu_main_incr (void)

next menu

```
                            {
    if(menu_now < MENU_ANZ_MAIN)     menu_main_set(menu_now + 1);
    else                             menu_main_set(1);
}
```

7.12.3.6 volatile void menu_main_set (int8_t *nr*)

set to menu "nr"

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

7.12.3.7 void menu_prop_set (int8_t)

set prop in argument active

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

7.12.3.8 void menu_right (void)

Handels pressed signal for button "right".

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

7.12.3.9 void menu_start (void)

Handels pressed signal for button "start".

```
{}
```

7.12.3.10 void menu_up (void)

Handels pressed signal for button "up".

```
                     {
    if(MENU_PROP_NOW.current_value < MENU_PROP_NOW.nr_values)
        MENU_PROP_NOW.current_value++;
    MENU_PROP_NOW.set_value(MENU_PROP_NOW.current_value);
}
```

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

```
                             {
    uint16_t reg = 0;

    //all values for 11.0592MHz oscillator
    if      (br == 0) reg = 287; //2k4
    else if(br == 1) reg = 143; //4k8
    else if(br == 2) reg = 71; //9k6
    else if(br == 3) reg = 47; //14k4
    else if(br == 4) reg = 35; //19k2
    else if(br == 5) reg = 23; //28k8
    else if(br == 6) reg = 17; //38k4
    else if(br == 7) reg = 11; //57k6
    else if(br == 8) reg = 8; //76k8
    else if(br == 9) reg = 5; //115k2
    else if(br == 10) reg = 2; //230k4

    UBRR0H = ((reg>>8) & 255); UBRR0L = reg & 255;
    UBRR1H = ((reg>>8) & 255); UBRR1L = reg & 255;

//  UBRR0H = 0; UBRR0L = 5;
//  UBRR1H = 0; UBRR1L = 5;
}
```

7.13.1.2 void usart_dataBits (int8_t *db*)

property: sets the nr of data bits

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

7.13.1.3 void usart_init (void)

initialises both USARTs

```
                      {
    //baudrate
//  UBRR0H = 0; UBRR0L = 0;
//  UBRR1H = 0; UBRR1L = 0;
    usart_baudrate(0);

    //erase data overflow flag
    UCSR0A &= ~(1<<DOR);
    UCSR1A &= ~(1<<DOR);

    //RXEN, TXEN, INTERRUPT ENABLE
    UCSR0B |= (1<<RXEN) | (1<<TXEN) | (1<<RXCIE);
    UCSR1B |= (1<<RXEN) | (1<<TXEN) | (1<<RXCIE);

    //2 stop bit, 8 data bit, synchronous mode
//  UCSR0C = (1<<USBS) | (3<<UCSZ0);// | (1<<UMSEL);
//  UCSR1C = (1<<USBS) | (3<<UCSZ0);// | (1<<UMSEL);
    usart_stopBits(1);
    usart_setSync(0);
    usart_dataBits(3);

    //activate internal PullUp for RX and XCK(if input)
    //DDRE = 255;
    PORTE |= (1<<PE0) | (1<<PE2);
    PORTD |= (1<<PD2) | (1<<PD5);

    uart_init(UART_BAUD_SELECT(9600, 11059200UL));
    uart1_init(UART_BAUD_SELECT(9600, 11059200UL));
}
```

7.13.1.4 void usart_off (void)

switches off both USARTs

```
                     {
    UCSR0B &= ~(1<<RXEN) & ~(1<<TXEN);
    UCSR1B &= ~(1<<RXEN) & ~(1<<TXEN);
}
```

7.13.1.5 void usart_setSync (int8_t *enable*)

property: enables synchronous communication

```
                                   {
    if(enable == 1){    //sync enabled
        UCSR0C |= (1<<UMSEL);
        UCSR1C |= (1<<UMSEL);
    }
    else if(enable == 0){               //sync disabled
        UCSR0C &= ~(1<<UMSEL);
        UCSR1C &= ~(1<<UMSEL);
    }
}
```

7.13.1.6 void usart_stopBits (int8_t *sb*)

property: sets the nr of stop bits

```
                                {
    if(sb == 0){        //1 stop bit
        UCSR0C &= ~(1<<USBS);
        UCSR1C &= ~(1<<USBS);
    }
    else if(sb == 1){   //2 stop bits
        UCSR0C |= (1<<USBS);
        UCSR1C |= (1<<USBS);
    }
}
```

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

```
                                 {
    uint16_t reg = 0;

    //all values for 11.0592MHz oscillator
    if      (br == 0) reg = 287; //2k4
    else if(br == 1) reg = 143; //4k8
    else if(br == 2) reg = 71; //9k6
    else if(br == 3) reg = 47; //14k4
    else if(br == 4) reg = 35; //19k2
    else if(br == 5) reg = 23; //28k8
    else if(br == 6) reg = 17; //38k4
    else if(br == 7) reg = 11; //57k6
    else if(br == 8) reg = 8; //76k8
    else if(br == 9) reg = 5; //115k2
    else if(br == 10) reg = 2; //230k4

    UBRR0H = ((reg>>8) & 255); UBRR0L = reg & 255;
    UBRR1H = ((reg>>8) & 255); UBRR1L = reg & 255;

//  UBRR0H = 0; UBRR0L = 5;
//  UBRR1H = 0; UBRR1L = 5;
}
```

7.14.3.2 void usart_dataBits (int8_t *db*)

property: sets the nr of data bits

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

7.14.3.3 void usart_init (void)

initialises both USARTs

```
                        {
    //baudrate
//  UBRR0H = 0; UBRR0L = 0;
//  UBRR1H = 0; UBRR1L = 0;
    usart_baudrate(0);

    //erase data overflow flag
    UCSR0A &= ~(1<<DOR);
    UCSR1A &= ~(1<<DOR);

    //RXEN, TXEN, INTERRUPT ENABLE
    UCSR0B |= (1<<RXEN) | (1<<TXEN) | (1<<RXCIE);
    UCSR1B |= (1<<RXEN) | (1<<TXEN) | (1<<RXCIE);

    //2 stop bit, 8 data bit, synchronous mode
//  UCSR0C = (1<<USBS) | (3<<UCSZ0);// | (1<<UMSEL);
//  UCSR1C = (1<<USBS) | (3<<UCSZ0);// | (1<<UMSEL);
    usart_stopBits(1);
    usart_setSync(0);
    usart_dataBits(3);

    //activate internal PullUp for RX and XCK(if input)
    //DDRE = 255;
    PORTE |= (1<<PE0) | (1<<PE2);
    PORTD |= (1<<PD2) | (1<<PD5);

    uart_init(UART_BAUD_SELECT(9600, 11059200UL));
    uart1_init(UART_BAUD_SELECT(9600, 11059200UL));
}
```

7.14.3.4 void usart_off (void)

switches off both USARTs

```
                    {
    UCSR0B &= ~(1<<RXEN) & ~(1<<TXEN);
    UCSR1B &= ~(1<<RXEN) & ~(1<<TXEN);
}
```

7.14.3.5 void usart_setSync (int8_t *enable*)

property: enables synchronous communication

```
                                 {
    if(enable == 1){    //sync enabled
        UCSR0C |= (1<<UMSEL);
        UCSR1C |= (1<<UMSEL);
    }
    else if(enable == 0){               //sync disabled
        UCSR0C &= ~(1<<UMSEL);
        UCSR1C &= ~(1<<UMSEL);
    }
}
```

7.14.3.6 void usart_stopBits (int8_t *sb*)

property: sets the nr of stop bits

```
                              {
    if(sb == 0){        //1 stop bit
        UCSR0C &= ~(1<<USBS);
        UCSR1C &= ~(1<<USBS);
    }
    else if(sb == 1){   //2 stop bits
        UCSR0C |= (1<<USBS);
        UCSR1C |= (1<<USBS);
    }
}
```

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.

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

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.

```
{
    uint16_t timer;

    /*  It is really, really important to stop global interrupts before
     *  reading 16bit registers. See the avr-libc FAQ! */
    cli();
    timer = TCNT3;
    sei();

    return ((us_time << 16) + timer);
}
```

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.

```
{
    uint32_t time2;
    time2 = us_time_get();
    if(time2 >= time1) return (time2-time1);
    else return (0xFFFFFFFF - time1+time2); //2^32
}
```

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.

```
{
    return ((double)(us_time_get_difference(time1) * US_TIMER_PRESCALER) / (double)XTAL);
}
```

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

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

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

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.

```
{
    uint16_t timer;

    /*  It is really, really important to stop global interrupts before
     *  reading 16bit registers. See the avr-libc FAQ! */
    cli();
    timer = TCNT3;
    sei();

    return ((us_time << 16) + timer);
}
```

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.

```
{
    uint32_t time2;
    time2 = us_time_get();
    if(time2 >= time1) return (time2-time1);
    else return (0xFFFFFFFF - time1+time2); //2^32
}
```

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.

```
{
    return ((double)(us_time_get_difference(time1) * US_TIMER_PRESCALER) / (double)XTAL);
}
```

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

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

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

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