Collection of AVR libraries

2022

Generated by Doxygen 1.8.17

Tue Nov 15 2022 20:34:10

1 Main Page

Collection of AVR libraries used in bachelor course Digital Electronics 2 at Brno University of Technology, Czechia.

Author

Tomas Fryza, Peter Fleury

Copyright

2 Module Index

2.1 Modules

Here is a list of all modules:

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

4 Module Documentation

4.1 GPIO Library <gpio.h>

GPIO library for AVR-GCC.

Functions

- void GPIO_mode_output (volatile uint8_t ∗reg, uint8_t pin)

  Configure one output pin.
- void GPIO_mode_input_pullup (volatile uint8_t ∗reg, uint8_t pin)

  Configure one input pin and enable pull-up.
- void GPIO_write_low (volatile uint8_t ∗reg, uint8_t pin)

  Write one pin to low value.
- void GPIO_write_high (volatile uint8_t ∗reg, uint8_t pin)

  Write one pin to high value.
- uint8_t GPIO_read (volatile uint8_t ∗reg, uint8_t pin)

  Read a value from input pin.

4.1.1 Detailed Description

GPIO library for AVR-GCC.

```
#include <gpio.h>
```

The library contains functions for controlling AVRs' gpio pin(s).

Note

Based on AVR Libc Reference Manual. Tested on ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2.

Author

Tomas Fryza, Dept. of Radio Electronics, Brno University of Technology, Czechia

Copyright

(c) 2019 Tomas Fryza, This work is licensed under the terms of the MIT license

4.1.2 Function Documentation

4.1.2.1 GPIO_mode_input_pullup()

```
void GPIO_mode_input_pullup (
        volatile uint8_t * reg,
        uint8_t pin )
```

Configure one input pin and enable pull-up.

Parameters

reg	Address of Data Direction Register, such as &DDRB
pin	Pin designation in the interval 0 to 7

Returns

none

4.1.2.2 GPIO_mode_output()

```
void GPIO_mode_output (
            volatile uint8_t * reg,
            uint8_t pin )
```

Configure one output pin.

Parameters

reg	Address of Data Direction Register, such as &DDRB
pin	Pin designation in the interval 0 to 7

Returns

none

4.1.2.3 GPIO_read()

```
uint8_t GPIO_read (
            volatile uint8_t * reg,
            uint8_t pin )
```

Read a value from input pin.

Parameters

reg	Address of Pin Register, such as &PINB
pin	Pin designation in the interval 0 to 7

Returns

Pin value

4.1.2.4 GPIO_write_high()

```
void GPIO_write_high (
            volatile uint8_t * reg,
            uint8_t pin )
```

Write one pin to high value.

Parameters

reg	Address of Port Register, such as &PORTB
pin	Pin designation in the interval 0 to 7

Returns

none

4.1.2.5 GPIO_write_low()

```
void GPIO_write_low (
            volatile uint8_t * reg,
            uint8_t pin )
```

Write one pin to low value.

Parameters

reg	Address of Port Register, such as &PORTB
pin	Pin designation in the interval 0 to 7

Returns

none

4.2 LCD library <lcd.h>

Basic routines for interfacing a HD44780U-based character LCD display.

Definition for LCD controller type

Use 0 for HD44780 controller, change to 1 for displays with KS0073 controller.

- #define LCD_CONTROLLER_KS0073 0

Definitions for Display Size

Change these definitions to adapt setting to your display

These definitions can be defined in a separate include file **lcd_definitions.h** instead modifying this file by adding -D_LCD_DEFINITIONS_FILE to the CDEFS section in the Makefile. All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h

- #define LCD_LINE_LENGTH 0x40
- #define LCD_START_LINE1 0x00
- #define LCD_START_LINE2 0x40
- #define LCD_START_LINE3 0x14
- #define LCD_START_LINE4 0x54
- #define LCD_WRAP_LINES 0

Definitions for 4-bit IO mode

The four LCD data lines and the three control lines RS, RW, E can be on the same port or on different ports. Change LCD_RS_PORT, LCD_RW_PORT, LCD_E_PORT if you want the control lines on different ports.

Normally the four data lines should be mapped to bit 0..3 on one port, but it is possible to connect these data lines in different order or even on different ports by adapting the LCD_DATAx_PORT and LCD_DATAx_PIN definitions.

Adjust these definitions to your target.
These definitions can be defined in a separate include file **lcd_definitions.h** instead modifying this file by adding **-D_LCD_DEFINITIONS_FILE** to the **CDEFS** section in the Makefile. All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h

- #define LCD_IO_MODE 1
- #define LCD_RW_PORT LCD_PORT
- #define LCD_RW_PIN 5

Definitions of delays

Used to calculate delay timers. Adapt the F_CPU define in the Makefile to the clock frequency in Hz of your target

These delay times can be adjusted, if some displays require different delays.
These definitions can be defined in a separate include file **lcd_definitions.h** instead modifying this file by adding **-D_LCD_DEFINITIONS_FILE** to the **CDEFS** section in the Makefile. All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h

- #define LCD_DELAY_BOOTUP 16000
- #define LCD_DELAY_INIT 5000
- #define LCD_DELAY_INIT_REP 64
- #define LCD_DELAY_INIT_4BIT 64
- #define LCD_DELAY_BUSY_FLAG 4
- #define LCD_DELAY_ENABLE_PULSE 1

Definitions for LCD command instructions

The constants define the various LCD controller instructions which can be passed to the function lcd_command(), see HD44780 data sheet for a complete description.

- #define **LCD_CLR** 0 /∗ DB0: clear display ∗/
- #define **LCD_HOME** 1 /∗ DB1: return to home position ∗/
- #define **LCD_ENTRY_MODE** 2 /∗ DB2: set entry mode ∗/
- #define **LCD_ENTRY_INC** 1 /∗ DB1: 1=increment, 0=decrement ∗/
- #define **LCD_ENTRY_SHIFT** 0 /∗ DB2: 1=display shift on ∗/
- #define **LCD_ON** 3 /∗ DB3: turn lcd/cursor on ∗/
- #define **LCD_ON_DISPLAY** 2 /∗ DB2: turn display on ∗/
- #define **LCD_ON_CURSOR** 1 /∗ DB1: turn cursor on ∗/
- #define **LCD_ON_BLINK** 0 /∗ DB0: blinking cursor ? ∗/
- #define **LCD_MOVE** 4 /∗ DB4: move cursor/display ∗/
- #define **LCD_MOVE_DISP** 3 /∗ DB3: move display (0-> cursor) ? ∗/
- #define **LCD_MOVE_RIGHT** 2 /∗ DB2: move right (0-> left) ? ∗/
- #define **LCD_FUNCTION** 5 /∗ DB5: function set ∗/
- #define **LCD_FUNCTION_8BIT** 4 /∗ DB4: set 8BIT mode (0->4BIT mode) ∗/
- #define **LCD_FUNCTION_2LINES** 3 /∗ DB3: two lines (0->one line) ∗/
- #define **LCD_FUNCTION_10DOTS** 2 /∗ DB2: 5x10 font (0->5x7 font) ∗/
- #define **LCD_CGRAM** 6 /∗ DB6: set CG RAM address ∗/
- #define **LCD_DDRAM** 7 /∗ DB7: set DD RAM address ∗/
- #define **LCD_BUSY** 7 /∗ DB7: LCD is busy ∗/
- #define **LCD_ENTRY_DEC** 0x04 /∗ display shift off, dec cursor move dir ∗/
- #define **LCD_ENTRY_DEC_SHIFT** 0x05 /∗ display shift on, dec cursor move dir ∗/
- #define **LCD_ENTRY_INC_** 0x06 /∗ display shift off, inc cursor move dir ∗/
- #define **LCD_ENTRY_INC_SHIFT** 0x07 /∗ display shift on, inc cursor move dir ∗/
- #define **LCD_DISP_OFF** 0x08 /∗ display off ∗/
- #define **LCD_DISP_ON** 0x0C /∗ display on, cursor off ∗/
- #define **LCD_DISP_ON_BLINK** 0x0D /∗ display on, cursor off, blink char ∗/
- #define **LCD_DISP_ON_CURSOR** 0x0E /∗ display on, cursor on ∗/
- #define **LCD_DISP_ON_CURSOR_BLINK** 0x0F /∗ display on, cursor on, blink char ∗/
- #define **LCD_MOVE_CURSOR_LEFT** 0x10 /∗ move cursor left (decrement) ∗/
- #define **LCD_MOVE_CURSOR_RIGHT** 0x14 /∗ move cursor right (increment) ∗/
- #define **LCD_MOVE_DISP_LEFT** 0x18 /∗ shift display left ∗/
- #define **LCD_MOVE_DISP_RIGHT** 0x1C /∗ shift display right ∗/
- #define **LCD_FUNCTION_4BIT_1LINE** 0x20 /∗ 4-bit interface, single line, 5x7 dots ∗/
- #define **LCD_FUNCTION_4BIT_2LINES** 0x28 /∗ 4-bit interface, dual line, 5x7 dots ∗/
- #define **LCD_FUNCTION_8BIT_1LINE** 0x30 /∗ 8-bit interface, single line, 5x7 dots ∗/
- #define **LCD_FUNCTION_8BIT_2LINES** 0x38 /∗ 8-bit interface, dual line, 5x7 dots ∗/
- #define **LCD_MODE_DEFAULT** ((1 << LCD_ENTRY_MODE) | (1 << LCD_ENTRY_INC))

Functions

- void lcd_init (uint8_t dispAttr)

  Initialize display and select type of cursor.
- void lcd_clrscr (void)

  Clear display and set cursor to home position.
- void lcd_home (void)

  Set cursor to home position.
- void lcd_gotoxy (uint8_t x, uint8_t y)

  Set cursor to specified position.
- void lcd_putc (char c)

  Display character at current cursor position.
- void lcd_puts (const char ∗s)

  Display string without auto linefeed.
- void lcd_puts_p (const char ∗progmem_s)

  Display string from program memory without auto linefeed.
- void lcd_command (uint8_t cmd)

  Send LCD controller instruction command.
- void lcd_data (uint8_t data)

  Send data byte to LCD controller.
- #define lcd_puts_P(__s) lcd_puts_p(PSTR(__s))

  macros for automatically storing string constant in program memory

4.2.1 Detailed Description

Basic routines for interfacing a HD44780U-based character LCD display.

```
#include <lcd.h>
```

LCD character displays can be found in many devices, like espresso machines, laser printers. The Hitachi HD44780 controller and its compatible controllers like Samsung KS0066U have become an industry standard for these types of displays.

This library allows easy interfacing with a HD44780 compatible display and can be operated in memory mapped mode (LCD_IO_MODE defined as 0 in the include file lcd.h.) or in 4-bit IO port mode (LCD_IO_MODE defined as 1). 8-bit IO port mode is not supported.

Memory mapped mode is compatible with old Kanda STK200 starter kit, but also supports generation of R/W signal through A8 address line.

See also

The chapter `Interfacing a HD44780 Based LCD to an AVR` on my home page, which shows example circuits how to connect an LCD to an AVR controller.

Author

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

Version

2.0

Copyright

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

4.2.2 Macro Definition Documentation

4.2.2.1 LCD_CONTROLLER_KS0073 `#define LCD_CONTROLLER_KS0073 0`

Use 0 for HD44780 controller, 1 for KS0073 controller

4.2.2.2 LCD_DELAY_BOOTUP `#define LCD_DELAY_BOOTUP 16000`

delay in micro seconds after power-on

4.2.2.3 LCD_DELAY_BUSY_FLAG `#define LCD_DELAY_BUSY_FLAG 4`

time in micro seconds the address counter is updated after busy flag is cleared

4.2.2.4 LCD_DELAY_ENABLE_PULSE `#define LCD_DELAY_ENABLE_PULSE 1`

enable signal pulse width in micro seconds

4.2.2.5 LCD_DELAY_INIT `#define LCD_DELAY_INIT 5000`

delay in micro seconds after initialization command sent

4.2.2.6 LCD_DELAY_INIT_4BIT `#define LCD_DELAY_INIT_4BIT 64`

delay in micro seconds after setting 4-bit mode

4.2.2.7 LCD_DELAY_INIT_REP `#define LCD_DELAY_INIT_REP 64`

delay in micro seconds after initialization command repeated

4.2.2.8 LCD_IO_MODE `#define LCD_IO_MODE 1`

0: memory mapped mode, 1: IO port mode

4.2.2.9 LCD_LINE_LENGTH `#define LCD_LINE_LENGTH 0x40`

internal line length of the display

4.2.2.10 LCD_RW_PIN `#define LCD_RW_PIN 5`

pin for RW line

4.2.2.11 LCD_RW_PORT `#define LCD_RW_PORT LCD_PORT`

port for RW line

4.2.2.12 LCD_START_LINE1 `#define LCD_START_LINE1 0x00`

DDRAM address of first char of line 1

4.2.2.13 LCD_START_LINE2 `#define LCD_START_LINE2 0x40`

DDRAM address of first char of line 2

4.2.2.14 LCD_START_LINE3 `#define LCD_START_LINE3 0x14`

DDRAM address of first char of line 3

4.2.2.15 LCD_START_LINE4 `#define LCD_START_LINE4 0x54`

DDRAM address of first char of line 4

4.2.2.16 LCD_WRAP_LINES `#define LCD_WRAP_LINES 0`

0: no wrap, 1: wrap at end of visibile line

4.2.3 Function Documentation

4.2.3.1 lcd_clrscr()

```
void lcd_clrscr (
            void  )
```

Clear display and set cursor to home position.

Returns

none

4.2.3.2 lcd_command()

```
void lcd_command (
            uint8_t cmd )
```

Send LCD controller instruction command.

Parameters

cmd	instruction to send to LCD controller, see HD44780 data sheet

Returns

none

4.2.3.3 lcd_data()

```
void lcd_data (
        uint8_t data )
```

Send data byte to LCD controller.

Similar to lcd_putc(), but without interpreting LF

Parameters

data	byte to send to LCD controller, see HD44780 data sheet

Returns

none

4.2.3.4 lcd_gotoxy()

```
void lcd_gotoxy (
        uint8_t x,
        uint8_t y )
```

Set cursor to specified position.

Parameters

x	horizontal position (0: left most position)
y	vertical position (0: first line)

Returns

none

4.2.3.5 lcd_home()

```
void lcd_home (
        void )
```

Set cursor to home position.

Returns

none

4.2.3.6 lcd_init()

```
void lcd_init (
        uint8_t dispAttr )
```

Initialize display and select type of cursor.

Parameters

dispAttr	**LCD_DISP_OFF** display off **LCD_DISP_ON** display on, cursor off **LCD_DISP_ON_CURSOR** display on, cursor on **LCD_DISP_ON_CURSOR_BLINK** display on, cursor on flashing

Returns

none

4.2.3.7 lcd_putc()

```
void lcd_putc (
        char c )
```

Display character at current cursor position.

Parameters

c	character to be displayed

Returns

none

4.2.3.8 lcd_puts()

```
void lcd_puts (
        const char * s )
```

Display string without auto linefeed.

Parameters

s	string to be displayed

Returns

none

4.2.3.9 lcd_puts_p()

```
void lcd_puts_p (
        const char * progmem_s )
```

Display string from program memory without auto linefeed.

Parameters

progmem↩_s	string from program memory be be displayed

Returns

none

See also

lcd_puts_P

4.3 LCD Definitions <lcd_definitions.h>

Adjusting the display settings.

Definitions for Display Size

Number of visible lines and characters per line of the display.

Note

All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h. Add -D_LCD↩
_DEFINITIONS_FILE to the CDEFS section in the Makefile.

- #define LCD_LINES 2

  Number of visible lines of the display.
- #define LCD_DISP_LENGTH 16

  Visibles characters per line of the display.

Definitions for 4-bit IO mode

4-bit mode definition of LCD signals on the Arduino Uno LCD Keypad Shield.

The four LCD data lines and the two control lines RS, E can be on the same port or on different ports. R/W pin is directly connected to GND on LCD Keypad Shield and cannot be controlled.

Note

All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h. Add -D_LCD↩
_DEFINITIONS_FILE to the CDEFS section in the Makefile.

- #define **LCD_PORT** PORTD
- #define **LCD_DATA0_PORT** LCD_PORT
- #define **LCD_DATA1_PORT** LCD_PORT
- #define **LCD_DATA2_PORT** LCD_PORT
- #define **LCD_DATA3_PORT** LCD_PORT
- #define LCD_DATA0_PIN PD4

  Pin for HD44780 data pin D4.
- #define LCD_DATA1_PIN PD5

  Pin for HD44780 data pin D5.
- #define LCD_DATA2_PIN PD6

  Pin for HD44780 data pin D6.
- #define LCD_DATA3_PIN PD7

  Pin for HD44780 data pin D7.
- #define **LCD_RS_PORT** PORTB
- #define **LCD_RS_PIN** PB0
- #define **LCD_E_PORT** PORTB
- #define **LCD_E_PIN** PB1

4.3.1 Detailed Description

Adjusting the display settings.

```
#include <lcd_definitions.h>
```

All definitions added to the file "lcd_definitions.h" will override the default definitions from "lcd.h" (see Peter Fleury's LCD library for HD44780 based LCDs).

Author

Tomas Fryza, Peter Fleury, Dept. of Radio Electronics, Brno University of Technology, Czechia

Copyright

4.4 Seven-segment Library <segment.h>

Seven-segment display library for AVR-GCC.

Definition of SSD interface

Note

Connection is based on Multi-function shield.

- void SEG_init (void)

  Configure SSD signals LATCH, CLK, and DATA as output.
- void SEG_update_shift_regs (uint8_t segments, uint8_t position)

  Display segments at one position of the SSD.
- #define **SEG_LATCH** PD4
- #define **SEG_CLK** PD7
- #define **SEG_DATA** PB0

4.4.1 Detailed Description

Seven-segment display library for AVR-GCC.

```
#include <segment.h>
```

The library contains functions for controlling the seven-segment display (SSD) using two shift registers 74HC595.

Author

Tomas Fryza, Dept. of Radio Electronics, Brno University of Technology, Czechia

Copyright

(c) 2019 Tomas Fryza, This work is licensed under the terms of the MIT license

4.4.2 Function Documentation

4.4.2.1 SEG_init()

```
void SEG_init (
          void   )
```

Configure SSD signals LATCH, CLK, and DATA as output.

Returns

none

4.4.2.2 SEG_update_shift_regs()

```
void SEG_update_shift_regs (
          uint8_t segments,
          uint8_t position )
```

Display segments at one position of the SSD.

Parameters

segments	Segments to be displayed (abcdefgDP, active low)
position	Position of the display where the segments are to be displayed (p3 p2 p1 p0 xxxx, active high)

Note

Two shift registers are connected in series, ie 16 bits are transmitted.

Returns

none

4.5 Timer Library <timer.h>

Timer library for AVR-GCC.

Definitions for 16-bit Timer/Counter1

Note

t_OVF = 1/F_CPU ∗ prescaler ∗ 2^n where n = 16, F_CPU = 16 MHz

- #define TIM1_stop() TCCR1B &= ∼((1<<CS12) | (1<<CS11) | (1<<CS10));

  Stop timer, prescaler 000 --> STOP.
- #define TIM1_overflow_4ms() TCCR1B &= ∼((1<<CS12) | (1<<CS11)); TCCR1B |= (1<<CS10);

  Set overflow 4ms, prescaler 001 --> 1.
- #define TIM1_overflow_33ms() TCCR1B &= ∼((1<<CS12) | (1<<CS10)); TCCR1B |= (1<<CS11);

  Set overflow 33ms, prescaler 010 --> 8.
- #define TIM1_overflow_262ms() TCCR1B &= ∼(1<<CS12); TCCR1B |= (1<<CS11) | (1<<CS10);

  Set overflow 262ms, prescaler 011 --> 64.
- #define TIM1_overflow_1s() TCCR1B &= ∼((1<<CS11) | (1<<CS10)); TCCR1B |= (1<<CS12);

  Set overflow 1s, prescaler 100 --> 256.
- #define TIM1_overflow_4s() TCCR1B &= ∼(1<<CS11); TCCR1B |= (1<<CS12) | (1<<CS10);

  Set overflow 4s, prescaler // 101 --> 1024.
- #define TIM1_overflow_interrupt_enable() TIMSK1 |= (1<<TOIE1);

  Enable overflow interrupt, 1 --> enable.
- #define TIM1_overflow_interrupt_disable() TIMSK1 &= ∼(1<<TOIE1);

  Disable overflow interrupt, 0 --> disable.

4.5.1 Detailed Description

Timer library for AVR-GCC.

```
#include <timer.h>
```

The library contains macros for controlling the timer modules.

Note

Based on Microchip Atmel ATmega328P manual and no source file is needed for the library.

Author

Tomas Fryza, Dept. of Radio Electronics, Brno University of Technology, Czechia

Copyright

(c) 2019 Tomas Fryza, This work is licensed under the terms of the MIT license

4.6 TWI Library <twi.h>

I2C/TWI library for AVR-GCC.

Definition of frequencies

- #define F_CPU 16000000

    CPU frequency in Hz required TWI_BIT_RATE_REG.
- #define F_SCL 50000

    I2C/TWI bit rate. Must be greater than 31000.
- #define TWI_BIT_RATE_REG ((F_CPU/F_SCL - 16) / 2)

    TWI bit rate register value.

Definition of ports and pins

- #define TWI_PORT PORTC

    Port of TWI unit.
- #define TWI_SDA_PIN 4

    SDA pin of TWI unit.
- #define TWI_SCL_PIN 5

    SCL pin of TWI unit.

Other definitions

- void twi_init (void)

    Initialize TWI unit, enable internal pull-ups, and set SCL frequency.
- uint8_t twi_start (uint8_t address, uint8_t mode)

    Start communication on I2C/TWI bus and send address byte.
- void twi_write (uint8_t data)

    Send one data byte to I2C/TWI Slave device.
- uint8_t twi_read_ack (void)

    Read one byte from the I2C/TWI Slave device and acknowledge it with ACK, i.e. communication will continue.
- uint8_t twi_read_nack (void)

    Read one byte from the I2C/TWI Slave device and acknowledge it with NACK, i.e. communication will not continue.
- void twi_stop (void)

    Generates stop condition on I2C/TWI bus.
- #define TWI_READ 1

    Mode for reading from I2C/TWI device.
- #define TWI_WRITE 0

    Mode for writing to I2C/TWI device.
- #define DDR(_x) (∗(&_x - 1))

    Define address of Data Direction Register of port _x.
- #define PIN(_x) (∗(&_x - 2))

    Define address of input register of port _x.

4.6.1 Detailed Description

I2C/TWI library for AVR-GCC.

```
#include <twi.h>
```

This library defines functions for the TWI (I2C) communication between AVR and Slave device(s). Functions use internal TWI module of AVR.

Note

Based on Microchip Atmel ATmega16 and ATmega328P manuals.

Author

Tomas Fryza, Dept. of Radio Electronics, Brno University of Technology, Czechia

Copyright

(c) 2018 Tomas Fryza, This work is licensed under the terms of the MIT license

4.6.2 Function Documentation

4.6.2.1 twi_init()

```
void twi_init (
        void )
```

Initialize TWI unit, enable internal pull-ups, and set SCL frequency.

Implementation notes:

- AVR internal pull-up resistors at pins TWI_SDA_PIN and TWI_SCL_PIN are enabled
- TWI bit rate register value is calculated as follows fscl = fcpu/(16 + 2∗TWBR)

Returns

none

4.6.2.2 twi_read_ack()

```
uint8_t twi_read_ack (
        void )
```

Read one byte from the I2C/TWI Slave device and acknowledge it with ACK, i.e. communication will continue.

Returns

Received data byte

4.6.2.3 twi_read_nack()

```
uint8_t twi_read_nack (
        void )
```

Read one byte from the I2C/TWI Slave device and acknowledge it with NACK, i.e. communication will not continue.

Returns

Received data byte

4.6.2.4 twi_start()

```
uint8_t twi_start (
        uint8_t address,
        uint8_t mode )
```

Start communication on I2C/TWI bus and send address byte.

Parameters

address	Slave address
mode	TWI_READ or TWI_WRITE

Return values

0	- Slave device accessible
1	- Failed to access Slave device

Note

Function returns 0 only if 0x18 or 0x40 status code is detected
0x18: SLA+W has been transmitted and ACK has been received
0x40: SLA+R has been transmitted and ACK has been received

4.6.2.5 twi_stop()

```
void twi_stop (
        void )
```

Generates stop condition on I2C/TWI bus.

Returns

none

4.6.2.6 twi_write()

```
void twi_write (
        uint8_t data )
```

Send one data byte to I2C/TWI Slave device.

Parameters

data	Byte to be transmitted

Returns

none

4.7 UART Library <uart.h>

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

Macros

- #define UART_BAUD_SELECT(baudRate, xtalCpu) (((xtalCpu) + 8UL ∗ (baudRate)) / (16UL ∗ (baudRate)) - 1UL)

    UART Baudrate Expression.
- #define UART_BAUD_SELECT_DOUBLE_SPEED(baudRate, xtalCpu) (((((xtalCpu) + 4UL ∗ (baudRate)) / (8UL ∗ (baudRate)) - 1UL)) | 0x8000)

    UART Baudrate Expression for ATmega double speed mode.
- #define UART_RX_BUFFER_SIZE 64

    Size of the circular receive buffer, must be power of 2.
- #define UART_TX_BUFFER_SIZE 64

    Size of the circular transmit buffer, must be power of 2.
- #define UART_FRAME_ERROR 0x1000

    Framing Error by UART
- #define UART_OVERRUN_ERROR 0x0800

    Overrun condition by UART
- #define UART_PARITY_ERROR 0x0400

    Parity Error by UART
- #define UART_BUFFER_OVERFLOW 0x0200

    receive ringbuffer overflow
- #define UART_NO_DATA 0x0100

    no receive data available
- #define uart_puts_P(__s) uart_puts_p(PSTR(__s))

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

    Macro to automatically put a string constant into program memory.

Functions

- void uart_init (unsigned int baudrate)

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

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

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

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

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

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

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

- void uart1_putc (unsigned char data)

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

- void uart1_puts (const char ∗s)

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

- void uart1_puts_p (const char ∗s)

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

4.7.1 Detailed Description

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

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

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

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

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

Note

Based on Atmel Application Note AVR306

Author

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

Copyright

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

4.7.2 Macro Definition Documentation

4.7.2.1 UART_BAUD_SELECT

```
#define UART_BAUD_SELECT(
            baudRate,
            xtalCpu ) (((xtalCpu) + 8UL * (baudRate)) / (16UL * (baudRate)) - 1UL)
```

UART Baudrate Expression.

Parameters

xtalCpu	system clock in Mhz, e.g. 4000000UL for 4Mhz
baudRate	baudrate in bps, e.g. 1200, 2400, 9600

4.7.2.2 UART_BAUD_SELECT_DOUBLE_SPEED

```
#define UART_BAUD_SELECT_DOUBLE_SPEED(
            baudRate,
            xtalCpu ) ( ((((xtalCpu) + 4UL * (baudRate)) / (8UL * (baudRate)) - 1UL)) | 0x8000)
```

UART Baudrate Expression for ATmega double speed mode.

Parameters

xtalCpu	system clock in Mhz, e.g. 4000000UL for 4Mhz
baudRate	baudrate in bps, e.g. 1200, 2400, 9600

4.7.2.3 UART_RX_BUFFER_SIZE

```
#define UART_RX_BUFFER_SIZE 64
```

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

You may need to adapt this constant to your target and your application by adding CDEFS += -DUART_RX_BUF↩ FER_SIZE=nn to your Makefile.

4.7.2.4 UART_TX_BUFFER_SIZE

```
#define UART_TX_BUFFER_SIZE 64
```

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

You may need to adapt this constant to your target and your application by adding CDEFS += -DUART_TX_BUF↩ FER_SIZE=nn to your Makefile.

4.7.3 Function Documentation

4.7.3.1 uart1_getc()

```
unsigned int uart1_getc (
            void )
```

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

See also

uart_getc

4.7.3.2 uart1_init()

```
void uart1_init (
        unsigned int baudrate )
```

Initialize USART1 (only available on selected ATmegas)

See also

uart_init

4.7.3.3 uart1_putc()

```
void uart1_putc (
        unsigned char data )
```

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

See also

uart_putc

4.7.3.4 uart1_puts()

```
void uart1_puts (
        const char * s )
```

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

See also

uart_puts

4.7.3.5 uart1_puts_p()

```
void uart1_puts_p (
        const char * s )
```

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

See also

uart_puts_p

4.7.3.6 uart_getc()

```
unsigned int uart_getc (
        void )
```

Get received byte from ringbuffer.

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

Returns

lower byte: received byte from ringbuffer

higher byte: last receive status

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

4.7.3.7 uart_init()

```
void uart_init (
        unsigned int baudrate )
```

Initialize UART and set baudrate.

Parameters

baudrate	Specify baudrate using macro UART_BAUD_SELECT()

Returns

none

4.7.3.8 uart_putc()

```
void uart_putc (
        unsigned char data )
```

Put byte to ringbuffer for transmitting via UART.

Parameters

data	byte to be transmitted

Returns

none

4.7.3.9 uart_puts()

```
void uart_puts (
        const char * s )
```

Put string to ringbuffer for transmitting via UART.

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

Parameters

s	string to be transmitted

Returns

none

4.7.3.10 uart_puts_p()

```
void uart_puts_p (
        const char * s )
```

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

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

Parameters

s	program memory string to be transmitted

Returns

none

See also

uart_puts_P

5 File Documentation

5.1 gpio.h File Reference

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

Functions

- void GPIO_mode_output (volatile uint8_t ∗reg, uint8_t pin)

  Configure one output pin.
- void GPIO_mode_input_pullup (volatile uint8_t ∗reg, uint8_t pin)

  Configure one input pin and enable pull-up.
- void GPIO_write_low (volatile uint8_t ∗reg, uint8_t pin)

  Write one pin to low value.
- void GPIO_write_high (volatile uint8_t ∗reg, uint8_t pin)

  Write one pin to high value.
- uint8_t GPIO_read (volatile uint8_t ∗reg, uint8_t pin)

  Read a value from input pin.

5.2 lcd.h File Reference

```
#include <inttypes.h>
#include <avr/pgmspace.h>
#include "lcd_definitions.h"
```

Macros

Definition for LCD controller type

Use 0 for HD44780 controller, change to 1 for displays with KS0073 controller.

- #define LCD_CONTROLLER_KS0073 0

Definitions for Display Size

Change these definitions to adapt setting to your display

*These definitions can be defined in a separate include file **lcd_definitions.h** instead modifying this file by adding -D_LCD_DEFINITIONS_FILE to the CDEFS section in the Makefile. All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h*

- #define LCD_LINE_LENGTH 0x40
- #define LCD_START_LINE1 0x00
- #define LCD_START_LINE2 0x40
- #define LCD_START_LINE3 0x14
- #define LCD_START_LINE4 0x54
- #define LCD_WRAP_LINES 0

Definitions for 4-bit IO mode

The four LCD data lines and the three control lines RS, RW, E can be on the same port or on different ports. Change LCD_RS_PORT, LCD_RW_PORT, LCD_E_PORT if you want the control lines on different ports.

Normally the four data lines should be mapped to bit 0..3 on one port, but it is possible to connect these data lines in different order or even on different ports by adapting the LCD_DATAx_PORT and LCD_DATAx_PIN definitions.

Adjust these definitions to your target.
These definitions can be defined in a separate include file ***lcd_definitions.h*** *instead modifying this file by adding* ***-D_LCD_DEFINITIONS_FILE*** *to the* ***CDEFS*** *section in the Makefile. All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h*

- #define LCD_IO_MODE 1
- #define LCD_RW_PORT LCD_PORT
- #define LCD_RW_PIN 5

Definitions of delays

Used to calculate delay timers. Adapt the F_CPU define in the Makefile to the clock frequency in Hz of your target

These delay times can be adjusted, if some displays require different delays.
These definitions can be defined in a separate include file ***lcd_definitions.h*** *instead modifying this file by adding* ***-D_LCD_DEFINITIONS_FILE*** *to the* ***CDEFS*** *section in the Makefile. All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h*

- #define LCD_DELAY_BOOTUP 16000
- #define LCD_DELAY_INIT 5000
- #define LCD_DELAY_INIT_REP 64
- #define LCD_DELAY_INIT_4BIT 64
- #define LCD_DELAY_BUSY_FLAG 4
- #define LCD_DELAY_ENABLE_PULSE 1

Definitions for LCD command instructions

The constants define the various LCD controller instructions which can be passed to the function lcd_command(), see HD44780 data sheet for a complete description.

- #define **LCD_CLR** 0 /∗ DB0: clear display ∗/
- #define **LCD_HOME** 1 /∗ DB1: return to home position ∗/
- #define **LCD_ENTRY_MODE** 2 /∗ DB2: set entry mode ∗/
- #define **LCD_ENTRY_INC** 1 /∗ DB1: 1=increment, 0=decrement ∗/
- #define **LCD_ENTRY_SHIFT** 0 /∗ DB2: 1=display shift on ∗/
- #define **LCD_ON** 3 /∗ DB3: turn lcd/cursor on ∗/
- #define **LCD_ON_DISPLAY** 2 /∗ DB2: turn display on ∗/
- #define **LCD_ON_CURSOR** 1 /∗ DB1: turn cursor on ∗/
- #define **LCD_ON_BLINK** 0 /∗ DB0: blinking cursor ? ∗/
- #define **LCD_MOVE** 4 /∗ DB4: move cursor/display ∗/
- #define **LCD_MOVE_DISP** 3 /∗ DB3: move display (0-> cursor) ? ∗/
- #define **LCD_MOVE_RIGHT** 2 /∗ DB2: move right (0-> left) ? ∗/
- #define **LCD_FUNCTION** 5 /∗ DB5: function set ∗/
- #define **LCD_FUNCTION_8BIT** 4 /∗ DB4: set 8BIT mode (0->4BIT mode) ∗/
- #define **LCD_FUNCTION_2LINES** 3 /∗ DB3: two lines (0->one line) ∗/
- #define **LCD_FUNCTION_10DOTS** 2 /∗ DB2: 5x10 font (0->5x7 font) ∗/
- #define **LCD_CGRAM** 6 /∗ DB6: set CG RAM address ∗/
- #define **LCD_DDRAM** 7 /∗ DB7: set DD RAM address ∗/
- #define **LCD_BUSY** 7 /∗ DB7: LCD is busy ∗/
- #define **LCD_ENTRY_DEC** 0x04 /∗ display shift off, dec cursor move dir ∗/
- #define **LCD_ENTRY_DEC_SHIFT** 0x05 /∗ display shift on, dec cursor move dir ∗/
- #define **LCD_ENTRY_INC_** 0x06 /∗ display shift off, inc cursor move dir ∗/
- #define **LCD_ENTRY_INC_SHIFT** 0x07 /∗ display shift on, inc cursor move dir ∗/
- #define **LCD_DISP_OFF** 0x08 /∗ display off ∗/
- #define **LCD_DISP_ON** 0x0C /∗ display on, cursor off ∗/

- #define **LCD_DISP_ON_BLINK** 0x0D /∗ display on, cursor off, blink char ∗/
- #define **LCD_DISP_ON_CURSOR** 0x0E /∗ display on, cursor on ∗/
- #define **LCD_DISP_ON_CURSOR_BLINK** 0x0F /∗ display on, cursor on, blink char ∗/
- #define **LCD_MOVE_CURSOR_LEFT** 0x10 /∗ move cursor left (decrement) ∗/
- #define **LCD_MOVE_CURSOR_RIGHT** 0x14 /∗ move cursor right (increment) ∗/
- #define **LCD_MOVE_DISP_LEFT** 0x18 /∗ shift display left ∗/
- #define **LCD_MOVE_DISP_RIGHT** 0x1C /∗ shift display right ∗/
- #define **LCD_FUNCTION_4BIT_1LINE** 0x20 /∗ 4-bit interface, single line, 5x7 dots ∗/
- #define **LCD_FUNCTION_4BIT_2LINES** 0x28 /∗ 4-bit interface, dual line, 5x7 dots ∗/
- #define **LCD_FUNCTION_8BIT_1LINE** 0x30 /∗ 8-bit interface, single line, 5x7 dots ∗/
- #define **LCD_FUNCTION_8BIT_2LINES** 0x38 /∗ 8-bit interface, dual line, 5x7 dots ∗/
- #define **LCD_MODE_DEFAULT** ((1 << LCD_ENTRY_MODE) | (1 << LCD_ENTRY_INC))

Functions

- #define lcd_puts_P(__s) lcd_puts_p(PSTR(__s))

  macros for automatically storing string constant in program memory
- void lcd_init (uint8_t dispAttr)

  Initialize display and select type of cursor.
- void lcd_clrscr (void)

  Clear display and set cursor to home position.
- void lcd_home (void)

  Set cursor to home position.
- void lcd_gotoxy (uint8_t x, uint8_t y)

  Set cursor to specified position.
- void lcd_putc (char c)

  Display character at current cursor position.
- void lcd_puts (const char ∗s)

  Display string without auto linefeed.
- void lcd_puts_p (const char ∗progmem_s)

  Display string from program memory without auto linefeed.
- void lcd_command (uint8_t cmd)

  Send LCD controller instruction command.
- void lcd_data (uint8_t data)

  Send data byte to LCD controller.

5.3 lcd_definitions.h File Reference

Macros

Definitions for Display Size

Number of visible lines and characters per line of the display.

Note

All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h. Add -D_L↩ CD_DEFINITIONS_FILE to the CDEFS section in the Makefile.

- #define LCD_LINES 2

  Number of visible lines of the display.
- #define LCD_DISP_LENGTH 16

  Visibles characters per line of the display.

Definitions for 4-bit IO mode

4-bit mode definition of LCD signals on the Arduino Uno LCD Keypad Shield.

The four LCD data lines and the two control lines RS, E can be on the same port or on different ports. R/W pin is directly connected to GND on LCD Keypad Shield and cannot be controlled.

Note

All definitions added to the file lcd_definitions.h will override the default definitions from lcd.h. Add -D_L↩ CD_DEFINITIONS_FILE to the CDEFS section in the Makefile.

- #define **LCD_PORT** PORTD
- #define **LCD_DATA0_PORT** LCD_PORT
- #define **LCD_DATA1_PORT** LCD_PORT
- #define **LCD_DATA2_PORT** LCD_PORT
- #define **LCD_DATA3_PORT** LCD_PORT
- #define LCD_DATA0_PIN PD4

  Pin for HD44780 data pin D4.
- #define LCD_DATA1_PIN PD5

  Pin for HD44780 data pin D5.
- #define LCD_DATA2_PIN PD6

  Pin for HD44780 data pin D6.
- #define LCD_DATA3_PIN PD7

  Pin for HD44780 data pin D7.
- #define **LCD_RS_PORT** PORTB
- #define **LCD_RS_PIN** PB0
- #define **LCD_E_PORT** PORTB
- #define **LCD_E_PIN** PB1

5.4 segment.h File Reference

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

Definition of SSD interface

Note

Connection is based on Multi-function shield.

- #define **SEG_LATCH** PD4
- #define **SEG_CLK** PD7
- #define **SEG_DATA** PB0
- void SEG_init (void)

  Configure SSD signals LATCH, CLK, and DATA as output.
- void SEG_update_shift_regs (uint8_t segments, uint8_t position)

  Display segments at one position of the SSD.

5.5 timer.h File Reference

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

Macros

Definitions for 16-bit Timer/Counter1

Note

t_OVF = 1/F_CPU ∗ prescaler ∗ 2^n where n = 16, F_CPU = 16 MHz

- #define TIM1_stop() TCCR1B &= ∼((1<<CS12) | (1<<CS11) | (1<<CS10));
  Stop timer, prescaler 000 --> STOP.
- #define TIM1_overflow_4ms() TCCR1B &= ∼((1<<CS12) | (1<<CS11)); TCCR1B |= (1<<CS10);
  Set overflow 4ms, prescaler 001 --> 1.
- #define TIM1_overflow_33ms() TCCR1B &= ∼((1<<CS12) | (1<<CS10)); TCCR1B |= (1<<CS11);
  Set overflow 33ms, prescaler 010 --> 8.
- #define TIM1_overflow_262ms() TCCR1B &= ∼(1<<CS12); TCCR1B |= (1<<CS11) | (1<<CS10);
  Set overflow 262ms, prescaler 011 --> 64.
- #define TIM1_overflow_1s() TCCR1B &= ∼((1<<CS11) | (1<<CS10)); TCCR1B |= (1<<CS12);
  Set overflow 1s, prescaler 100 --> 256.
- #define TIM1_overflow_4s() TCCR1B &= ∼(1<<CS11); TCCR1B |= (1<<CS12) | (1<<CS10);
  Set overflow 4s, prescaler // 101 --> 1024.
- #define TIM1_overflow_interrupt_enable() TIMSK1 |= (1<<TOIE1);
  Enable overflow interrupt, 1 --> enable.
- #define TIM1_overflow_interrupt_disable() TIMSK1 &= ∼(1<<TOIE1);
  Disable overflow interrupt, 0 --> disable.

5.6 twi.h File Reference

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

Macros

Definition of frequencies

- #define F_CPU 16000000
  CPU frequency in Hz required TWI_BIT_RATE_REG.
- #define F_SCL 50000
  I2C/TWI bit rate. Must be greater than 31000.
- #define TWI_BIT_RATE_REG ((F_CPU/F_SCL - 16) / 2)
  TWI bit rate register value.

Definition of ports and pins

- #define TWI_PORT PORTC
  Port of TWI unit.
- #define TWI_SDA_PIN 4
  SDA pin of TWI unit.
- #define TWI_SCL_PIN 5
  SCL pin of TWI unit.

Other definitions

- #define TWI_READ 1

  Mode for reading from I2C/TWI device.
- #define TWI_WRITE 0

  Mode for writing to I2C/TWI device.
- #define DDR(_x) (∗(&_x - 1))

  Define address of Data Direction Register of port _x.
- #define PIN(_x) (∗(&_x - 2))

  Define address of input register of port _x.
- void twi_init (void)

  Initialize TWI unit, enable internal pull-ups, and set SCL frequency.
- uint8_t twi_start (uint8_t address, uint8_t mode)

  Start communication on I2C/TWI bus and send address byte.
- void twi_write (uint8_t data)

  Send one data byte to I2C/TWI Slave device.
- uint8_t twi_read_ack (void)

  Read one byte from the I2C/TWI Slave device and acknowledge it with ACK, i.e. communication will continue.
- uint8_t twi_read_nack (void)

  Read one byte from the I2C/TWI Slave device and acknowledge it with NACK, i.e. communication will not continue.
- void twi_stop (void)

  Generates stop condition on I2C/TWI bus.

5.7 uart.h File Reference

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

Macros

- #define UART_BAUD_SELECT(baudRate, xtalCpu) (((xtalCpu) + 8UL ∗ (baudRate)) / (16UL ∗ (baudRate)) - 1UL)

  UART Baudrate Expression.
- #define UART_BAUD_SELECT_DOUBLE_SPEED(baudRate, xtalCpu) (((((xtalCpu) + 4UL ∗ (baudRate)) / (8UL ∗ (baudRate)) - 1UL)) | 0x8000)

  UART Baudrate Expression for ATmega double speed mode.
- #define UART_RX_BUFFER_SIZE 64

  Size of the circular receive buffer, must be power of 2.
- #define UART_TX_BUFFER_SIZE 64

  Size of the circular transmit buffer, must be power of 2.
- #define UART_FRAME_ERROR 0x1000

  Framing Error by UART
- #define UART_OVERRUN_ERROR 0x0800

  Overrun condition by UART
- #define UART_PARITY_ERROR 0x0400

  Parity Error by UART
- #define UART_BUFFER_OVERFLOW 0x0200

receive ringbuffer overflow
- #define UART_NO_DATA 0x0100

  no receive data available

- #define uart_puts_P(__s) uart_puts_p(PSTR(__s))

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

  Macro to automatically put a string constant into program memory.

Functions

- void uart_init (unsigned int baudrate)

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

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

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

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

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

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

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

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

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

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