Übungsprotokoll

SYTI – Systemtechnik Industrielle Systeme

htlkrems Bautechnik & IT	**Übungsdatum:** KW 00/17 – 99/17	**Klasse:** 5AHIT	**Name:** Yi Liu
	Abgabedatum: 13.3.2024	**Gruppe:** SYTI	**Note:**
Leitung: DI (FH) Markus BRUNNER	**Mitübende:** Gegebenenfalls hier anführen, muss mit Aufgabenteilung in der Aufgabenstellung korrespondieren!		

Übungsbezeichnung:

Messomat 7k

Inhaltsverzeichnis:

1 Aufgabenstellung

Ziel ist die Entwicklung eines intelligenten Temperatur- und Feuchtigkeitsüberwachungssystems, das (Mess-)Daten erfasst und an einen Rechner übermittelt. Zur Visualisierung Daten und Steuerung der Messanwendung ist im ersten Schritt eine einfache Terminalanwendung einzusetzen.

Weitere genauere Anforderungen sind im Dokument *„SYTI5_UE2-Messomat7k-v1.4.pdf“* zu sehen.

2 Abstract (English)

The aim is to develop an intelligent temperature and humidity monitoring system that records (measurement) data and transmits it to a computer. The first step is to use a simple terminal application to visualize data and control the measurement application.

More detailed information can be found in the document *“SYTI5_UE2-Messomat7k-v1.4.pdf”*.

3 Theoretische Grundlagen

Die theoretischen Grundlagen für die Übung mit dem ATMega328p und C-Programmierung umfassen mehrere Bereiche:

3.1 Mikrocontroller ATMega328p

1. Architektur: 8-Bit RISC, 32 Register, interne Peripheriegeräte (ADC, Timer, UART etc.)
2. Speicher: Flash (für Code), SRAM (für Variablen), EEPROM (für nichtflüchtige Speicherung)
3. GPIOs: Steuerung von Sensoren, Aktoren und LCD
4. Energie-Modi: Normalbetrieb vs. Energiesparmodus

3.2 Sensoren & Peripheriegeräte

1. **Temperatur- und Feuchtigkeitssensor** (z. B. DHT11, DHT22 oder SHT3x)
2. **Hochleistungsventilator** (Steuerung per PWM oder einfacher GPIO-Schaltung)
3. **LCD-Display** (z. B. HD44780, I2C- oder SPI-Ansteuerung)
4. **Taster-Eingaben** (z. B. Interrupt-gesteuert oder per Polling)

3.3 Kommunikation & Datenübertragung

1. **Serielle Kommunikation (UART)** zur Datenübertragung an den Host-Computer
2. **Protokoll mit STX/ETX** zur Rahmenbildung der Datenpakete
3. **Fehlermanagement**: Erkennung von Verbindungsabbrüchen, ACK/NACK-Prüfungen

3.4 Embedded-C Programmierung

1. Verwendung von **Header-Dateien (.h)** für systemweite Definitionen
2. **Bedingte Kompilierung** (Mock-Mode für Testzwecke)
3. **Interrupts & Timer** für präzise Steuerung (z. B. 1s vs. 4s Messintervall)
4. **EEPROM-Speicherung** zur Sicherung der letzten 10 Messwerte

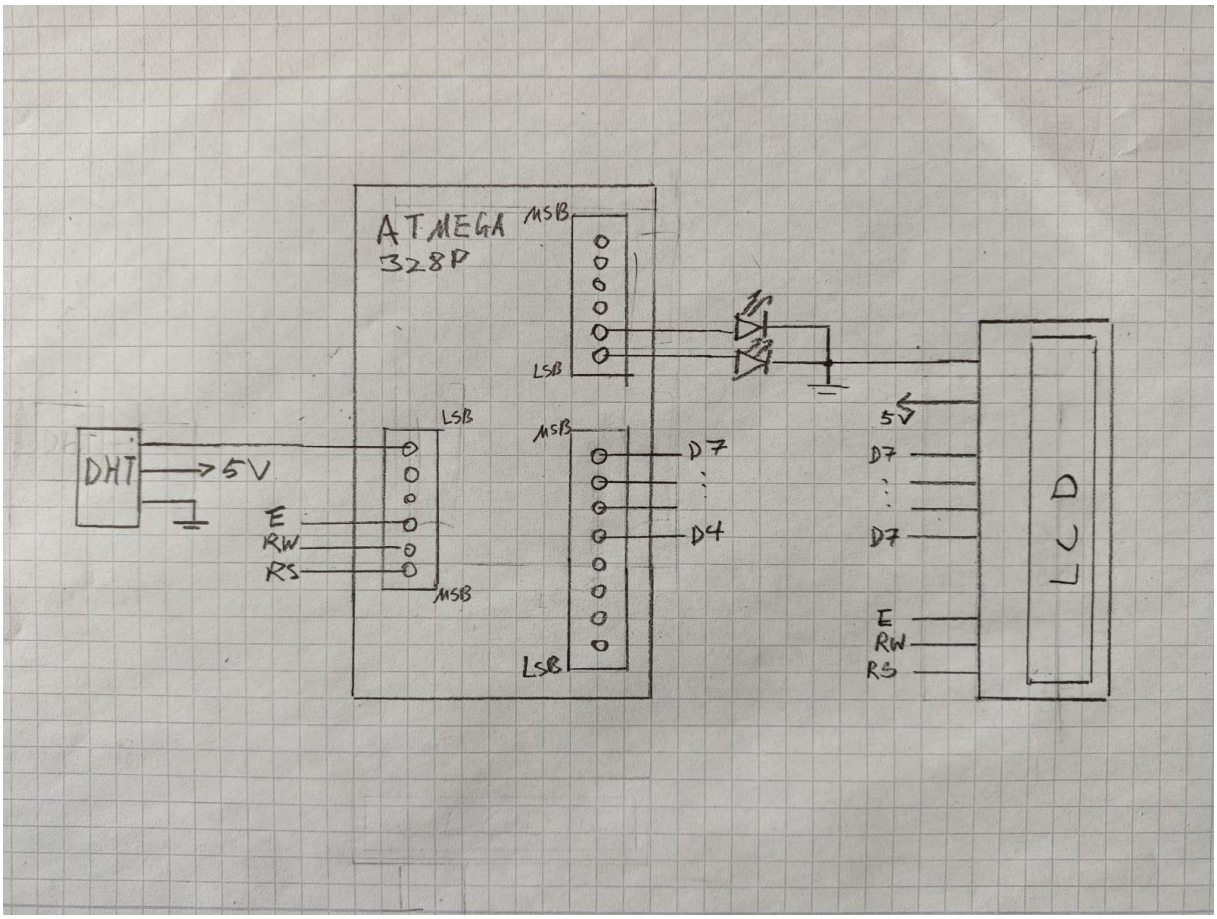
3.5 Fehlerbehandlung

1. **Fehlermanagement** durch Reset-Taste oder Software-Reset
2. **LED-Statusanzeige** für Verbindungsstatus

4 Schaltplan

Verwendete Teile sind:

- ATMega328p Board
- 2 LEDs
- DHT-7k
- LCD



5 Übungsdurchführung / Überlegungen beim Code-Design

5.1 Timer-Initialisierung

Die Messungen sollen in festen Intervallen (1 oder 4 Sekunden) erfolgen.

Ein Timer-Interrupt ist effizienter als eine `_delay_ms()`-Schleife, da er den Prozessor nicht blockiert.

Vorgang:

Timer1 im CTC-Modus (Clear Timer on Compare Match) mit Prescaler 1024.

Compare Match Value:

1 Sekunde: `OCR1A = 15624` (16.000.000 Hz / 1024 / 1 Hz - 1)

4 Sekunden: `OCR1A = 62499`.

```
void timer1_init(void) {          // Set Timer1 für CTC-Modus
    TCCR1B |= (1 << WGM12);       // CTC-Modus
    OCR1A = 15624;                // 1 Sekunde bei 16 MHz und Prescaler 1024
    TCCR1B |= (1 << CS12) | (1 << CS10);    // Prescaler 1024
    TIMSK1 |= (1 << OCIE1A);      // Output Compare A Interrupt aktivieren
}
```

5.2 UART-Kommunikation

Die Daten müssen zuverlässig an das Host-System übertragen werden.

Ein Framing-Protokoll (STX/ETX) sorgt für eine klare Struktur der Nachrichten.

ACK-Bestätigungen gewährleisten, dass die Daten korrekt empfangen wurden.

Vorgang:

Framing: Jede Nachricht wird mit <STX> (0x02) und <ETX> (0x03) gerahmt.

Beispiel: <STX>DATE22|HU55|SN1<ETX>.

Retry-Logik: Nach 3 fehlgeschlagenen Übertragungen:

LED leuchtet (**PB0**).

Messdaten werden im EEPROM gespeichert.

```
void send_data() {
sprintf(txBuffer, "\x02DATE%d|HU%d|SN%d\x03", currentTemp, currentHumidity, seqNumber);
uart_puts(txBuffer);
retryCount++;
if (retryCount >= 3) PORTB |= (1 << STATUS_LED_PIN); // Fehler-LED
}
```

5.3 EEPROM-Datenspeicherung

Bei Verbindungsabbrüchen sollen die letzten 10 Messwerte gespeichert werden, um sie später erneut zu senden.

Ein Ringpuffer (Kreis) ist effizient, da er Speicherplatz wiederverwendet und keine Verschiebung der Daten erfordert.

Vorgang:

- **Ringpuffer-Struktur:**
  - Byte 0: Aktuelle Position im Puffer.
  - Byte 1: Letzte Sequenznummer.
  - Byte 2–21: Temperatur- und Feuchtigkeitsdaten (10 × 2 Byte).

```
void store_in_eeprom(){
    uint8_t pos = eeprom_read_byte(&eepromStorage[0]);
    if(pos ⩾ 10) pos = 0;

    eeprom_write_byte(&eepromStorage[pos*2 + 2], currentTemp);
    eeprom_write_byte(&eepromStorage[pos*2 + 3], currentHumidity);
    eeprom_write_byte(&eepromStorage[0], pos + 1);
    eeprom_write_byte(&eepromStorage[1], seqNumber);

}
```

Vorteile des Ringpuffers:

Speichereffizienz: Es werden nur 22 Byte benötigt.

Einfache Implementierung: Die Position wird durch Modulo-Operation automatisch zurückgesetzt.

Robustheit: Daten gehen nicht verloren, selbst wenn der Puffer voll ist.

5.4 LCD-Anzeige

Zeile 1: T:22C H:55%

Zeile 2: F:ON I:1s ERR! (bei Verbindungsabbruch).

```
void update_display(uint8_t stopped) {
    lcd_clrscr();
    if(stopped) {
        lcd_puts("**ME gestoppt**");
    } else {
        // Measurements
        sprintf(displayBuffer, "T:%dC H:%d%%", currentTemp, currentHumidity);
        lcd_puts(displayBuffer);

        // Fan status & interval
        lcd_gotoxy(0,1);
        sprintf(displayBuffer, "F:%s I:%ds %s",
            fanStatus ? "ON " : "OFF",
            interval,
            connectionLost ? "ERR!" : "    ");
        lcd_puts(displayBuffer);
    }
}
```

5.5 Protokollengineering

5.5.1 Framing (STX/ETX)

Ein Rahmen (STX/ETX) ermöglicht es dem Empfänger, den Anfang und das Ende einer Nachricht zu erkennen. Dies ist besonders wichtig bei serieller Kommunikation, da Datenströme kontinuierlich sind.

Implementierung:

- Jede Nachricht beginnt mit <STX> (0x02) und endet mit <ETX> (0x03).
- Beispiel: <STX>DATE22|HU55|SN1<ETX>.

5.6 ACK-Bestätigung

Der Sender muss sicherstellen, dass die Nachricht korrekt empfangen wurde. Ein ACK (0x06) bestätigt den erfolgreichen Empfang.

Implementierung:

- Der Host sendet 0x06 nach erfolgreichem Empfang.
- Der Mikrocontroller setzt den Retry-Zähler zurück und erhöht die Sequenznummer.

5.6.1.1 C. Sequenznummern

Sequenznummern ermöglichen es, Nachrichten in der richtigen Reihenfolge zu verarbeiten. Sie helfen auch bei der Identifikation von verlorenen oder doppelten Nachrichten.

Implementierung:

- Jede Nachricht enthält eine Sequenznummer (SNx).
- Die Sequenznummer wird nach jedem erfolgreichen ACK erhöht.

6 Code

6.1 Main.c

```
/*
 * Messomat.c
 *
 * Created: 2024/12/5 13:16:47
 * Author : Yi
 */
#define F_CPU 16000000UL
#define UART_BAUD_RATE 9600
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include "dht.h"
#include "lcd.h"
#include "uart.h"

//Reservierungen
#define STATUS_LED_PIN  PORTB0
#define FAN_PIN         PORTB1
#define BTN_T1_PIN      PORTD2
#define BTN_T2_PIN      PORTD3
#define EEPROM_SIZE     22  // 2 header + 10*2 data

// Globale Statuse
volatile uint8_t interval = 1;
volatile uint8_t sendFlag = 0;
volatile uint8_t seqNumber = 1;
volatile uint8_t ackReceived = 0;
volatile uint8_t retryCount = 0;
volatile uint8_t measureTimeFlag = 0;
volatile uint8_t fanStatus = 0; // 0=off, 1=on
volatile uint8_t connectionLost = 0;

// EEPROM Specific
uint8_t EEMEM eepromStorage[EEPROM_SIZE];

//Sensor Daten
volatile int8_t currentTemp;
volatile int8_t currentHumidity;
volatile char displayBuffer[20];
volatile char txBuffer[32];

void timer1_init(void) {          // Set Timer1 für CTC-Modus
	TCCR1B |= (1 << WGM12);           // CTC-Modus
	OCR1A = 15624;                            // 1 Sekunde bei 16 MHz und Prescaler
1024
	TCCR1B |= (1 << CS12) | (1 << CS10);     // Prescaler 1024
	TIMSK1 |= (1 << OCIE1A);   // Output Compare A Interrupt aktivieren
}

ISR(TIMER1_COMPA_vect) {
	measureTimeFlag = 1;
}

void update_display(uint8_t stopped) {
	lcd_clrscr();
	if(stopped) {
		lcd_puts("**ME gestoppt**");
```

```
	} else {
		// Measurements
		sprintf(displayBuffer, "T:%dC H:%d%%", currentTemp, currentHumidity);
		lcd_puts(displayBuffer);

		// Fan status & interval
		lcd_gotoxy(0,1);
		sprintf(displayBuffer, "F:%s I:%ds %s",
			fanStatus ? "ON " : "OFF",
			interval,
			connectionLost ? "ERR!" : "    ");
		lcd_puts(displayBuffer);
	}
}

void send_data(){
	sprintf(txBuffer, "DATE%d|HU%d|SN%d", currentTemp, currentHumidity, seqNumber);
	uart_putc(0x02);
	uart_puts(txBuffer);
	uart_putc(0x03);
	retryCount++;
}


//	EEPROM: erste Stelle sag die neuest gespeicherten Daten aus, die zweite die
Seriennummer
//	folgenden 10*2 positionen bilden ein Kreis
void store_in_eeprom(){
	uint8_t pos = eeprom_read_byte(&eepromStorage[0]);
	if(pos >= 10) pos = 0;

	eeprom_write_byte(&eepromStorage[pos*2 + 2], currentTemp);
	eeprom_write_byte(&eepromStorage[pos*2 + 3], currentHumidity);
	eeprom_write_byte(&eepromStorage[0], pos + 1);
	eeprom_write_byte(&eepromStorage[1], seqNumber);

}

void resend_eeprom_data(){
	uint8_t start_pos = eeprom_read_byte(&eepromStorage[0]);
	uint8_t start_seq = eeprom_read_byte(&eepromStorage[1]) > 10
				? eeprom_read_byte(&eepromStorage[1]) - 10
				: 0;

	for(uint8_t i=0; i<10; i++) {
		uint8_t pos = (start_pos + i) % 10;
		int8_t temp = eeprom_read_byte(&eepromStorage[pos*2 + 2]);
		int8_t hum = eeprom_read_byte(&eepromStorage[pos*2 + 3]);

		sprintf(txBuffer, "DATE%d|HU%d|SN%d", temp, hum, start_seq + i);
		uart_putc(0x02);
		uart_puts(txBuffer);
		uart_putc(0x03);
	}
}

void check_input(void){
	char command = uart_getc();
	switch (command) {
		case 0x06:
			ackReceived = 1;
			if(connectionLost) {
				connectionLost = 0;
```

```
                        resend_eeprom_data();
                }
                break;
        case '1':
                OCR1A = 15624;
                interval = 1;
                TCNT1 = 0;// Reset timer
                update_display(0);
                break;
        case '4':
                OCR1A = 62499;
                interval = 4;
                TCNT1 = 0;  // Reset timer
                update_display(0);
                break;
        case 'd':
                sendFlag = 1;
                retryCount = 0;  // Reset retry counter
                update_display(0);
                break;
        case 'q':
                sendFlag = 0;
                PORTB &= ~(1 << FAN_PIN);
                fanStatus = 0;
                update_display(1);
                break;
        case 'e':
                PORTB |= (1 << FAN_PIN);
                fanStatus = 1;
                update_display(0);
                break;
        case 'a':
                PORTB &= ~(1 << FAN_PIN);
                fanStatus = 0;
                update_display(0);
                break;
        case 's':
                uart_putc(0x02);
                uart_puts(fanStatus ? "FAN1" : "FAN0");
                uart_putc(0x03);
                break;
        case 'r':  // Reset command
                retryCount = 0;
                ackReceived = 0;
                connectionLost = 0;
                PORTB &= ~(1 << STATUS_LED_PIN);
                break;
    }
}


int main(void)
{
    lcd_init(LCD_DISP_ON);
    lcd_clrscr();
    uart_init(UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU));
    timer1_init();

    //I/O-Konfigurationen
    DDRB |=(1<< STATUS_LED_PIN)|(1<<FAN_PIN);//DDB0/DDB1
    PORTB &= ~((1 << PORTB0) | (1 << PORTB1));// Sicherstellen dass 2 LEDs am Anfang
ausgeschaltet sind
```

```
	//Buttons (unnoetig)
	DDRD &= ~((1 << BTN_T1_PIN)|(1 << BTN_T2_PIN));
	PORTD |= (1 << BTN_T1_PIN)|(1 << BTN_T2_PIN); // pull-ups einschalten

	sei();
	update_display(1);

	while(1){
		check_input();
		if(ackReceived) {
			PORTB &= ~(1 << STATUS_LED_PIN);  // Turn off LED
			seqNumber++;
			retryCount = 0;
			ackReceived = 0;
		}

		if(measureTimeFlag) {
			measureTimeFlag = 0;
			if(dht_gettemperaturehumidity(&currentTemp, &currentHumidity) ==
DHT_ERROR_NOERR) {
				store_in_eeprom(); //alle Messungen werden gespeichert
				update_display(0);

				if (sendFlag) {
					if (retryCount < 3) {
						send_data();
						} else {
						PORTB |= (1 << STATUS_LED_PIN);  // Turn on
LED after 3 retries
						connectionLost = 1;
					}
				}
			}
		}
		// Handle LED state
		/*if (retryCount >= 3 && !ackReceived) {
			PORTB |= (1 << PORTB0);
		}*/

	}
}
```

6.2 Lcd.h

```
#ifndef LCD_H
#define LCD_H
/************************************************************************
 Title :   C include file for the HD44780U LCD library (lcd.c)
 Author:    Peter Fleury <pfleury@gmx.ch>  http://tinyurl.com/peterfleury
 File:     $Id: lcd.h,v 1.14.2.4 2015/01/20 17:16:07 peter Exp $
 Software:  AVR-GCC 4.x
 Hardware:  any AVR device, memory mapped mode only for AVR with
            memory mapped interface (AT90S8515/ATmega8515/ATmega128)
************************************************************************/

/**
 @mainpage
 Collection of libraries for AVR-GCC
 @author Peter Fleury pfleury@gmx.ch http://tinyurl.com/peterfleury
 @copyright (C) 2015 Peter Fleury, GNU General Public License Version 3

 @file
 @defgroup pfleury_lcd LCD library <lcd.h>
 @code #include <lcd.h> @endcode

 @brief Basic routines for interfacing a HD44780U-based character LCD display

 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 The chapter <a href=" http://homepage.hispeed.ch/peterfleury/avr-lcd44780.html"
target="_blank">Interfacing a HD44780 Based LCD to an AVR</a>
      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

*/

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

#if (__GNUC__ * 100 + __GNUC_MINOR__) < 405
#error "This library requires AVR-GCC 4.5 or later, update to newer AVR-GCC
compiler !"
#endif


/**@{*/

/*
```

```
 * LCD and target specific definitions below can be defined in a separate include file
with name 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
 */
#ifdef _LCD_DEFINITIONS_FILE
#include "lcd_definitions.h"
#endif


/**
 * @name  Definition for LCD controller type
 * Use 0 for HD44780 controller, change to 1 for displays with KS0073 controller.
 */
#ifndef LCD_CONTROLLER_KS0073
#define LCD_CONTROLLER_KS0073 0  /**< Use 0 for HD44780 controller, 1 for KS0073
controller */
#endif

/**
 * @name  Definitions for Display Size
 * Change these definitions to adapt setting to your display
 *
 * These definitions can be defined in a separate include file \b 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
 *
 */
#ifndef LCD_LINES
#define LCD_LINES           2     /**< number of visible lines of the display */
#endif
#ifndef LCD_DISP_LENGTH
#define LCD_DISP_LENGTH    16     /**< visibles characters per line of the display */
#endif
#ifndef LCD_LINE_LENGTH
#define LCD_LINE_LENGTH  0x40     /**< internal line length of the display    */
#endif
#ifndef LCD_START_LINE1
#define LCD_START_LINE1  0x00     /**< DDRAM address of first char of line 1 */
#endif
#ifndef LCD_START_LINE2
#define LCD_START_LINE2  0x40     /**< DDRAM address of first char of line 2 */
#endif
#ifndef LCD_START_LINE3
#define LCD_START_LINE3  0x14     /**< DDRAM address of first char of line 3 */
#endif
#ifndef LCD_START_LINE4
#define LCD_START_LINE4  0x54     /**< DDRAM address of first char of line 4 */
#endif
#ifndef LCD_WRAP_LINES
#define LCD_WRAP_LINES      0     /**< 0: no wrap, 1: wrap at end of visibile line */
#endif


/**
 * @name 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.\n
 * These definitions can be defined in a separate include file \b lcd_definitions.h
instead modifying this file by
 * adding \b -D_LCD_DEFINITIONS_FILE to the \b 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            /**< 0: memory mapped mode, 1: IO port mode */

#if LCD_IO_MODE

#ifndef LCD_PORT
#define LCD_PORT         PORTA        /**< port for the LCD lines   */
#endif
#ifndef LCD_DATA0_PORT
#define LCD_DATA0_PORT   LCD_PORT     /**< port for 4bit data bit 0 */
#endif
#ifndef LCD_DATA1_PORT
#define LCD_DATA1_PORT   LCD_PORT     /**< port for 4bit data bit 1 */
#endif
#ifndef LCD_DATA2_PORT
#define LCD_DATA2_PORT   LCD_PORT     /**< port for 4bit data bit 2 */
#endif
#ifndef LCD_DATA3_PORT
#define LCD_DATA3_PORT   LCD_PORT     /**< port for 4bit data bit 3 */
#endif
#ifndef LCD_DATA0_PIN
#define LCD_DATA0_PIN    4            /**< pin for 4bit data bit 0  */
#endif
#ifndef LCD_DATA1_PIN
#define LCD_DATA1_PIN    5            /**< pin for 4bit data bit 1  */
#endif
#ifndef LCD_DATA2_PIN
#define LCD_DATA2_PIN    6            /**< pin for 4bit data bit 2  */
#endif
#ifndef LCD_DATA3_PIN
#define LCD_DATA3_PIN    7            /**< pin for 4bit data bit 3  */
#endif
#ifndef LCD_RS_PORT
#define LCD_RS_PORT      PORTC     /**< port for RS line         */
#endif
#ifndef LCD_RS_PIN
#define LCD_RS_PIN       5            /**< pin  for RS line         */
#endif
#ifndef LCD_RW_PORT
#define LCD_RW_PORT      PORTC     /**< port for RW line         */
#endif
#ifndef LCD_RW_PIN
#define LCD_RW_PIN       4            /**< pin  for RW line         */
#endif
#ifndef LCD_E_PORT
#define LCD_E_PORT       PORTC     /**< port for Enable line     */
#endif
#ifndef LCD_E_PIN
#define LCD_E_PIN        3            /**< pin  for Enable line     */
#endif
```

```
#elif defined(__AVR_AT90S4414__) || defined(__AVR_AT90S8515__) ||
defined(__AVR_ATmega64__) || \
      defined(__AVR_ATmega8515__)|| defined(__AVR_ATmega103__) ||
defined(__AVR_ATmega128__) || \
      defined(__AVR_ATmega161__) || defined(__AVR_ATmega162__)
/*
 * memory mapped mode is only supported when the device has an external data memory
interface
 */
#define LCD_IO_DATA      0xC000    /* A15=E=1, A14=RS=1                */
#define LCD_IO_FUNCTION  0x8000    /* A15=E=1, A14=RS=0                */
#define LCD_IO_READ      0x0100    /* A8 =R/W=1 (R/W: 1=Read, 0=Write   */

#else
#error "external data memory interface not available for this device, use 4-bit IO
port mode"

#endif


/**
 * @name 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.\n
 * These definitions can be defined in a separate include file \b lcd_definitions.h
instead modifying this file by
 * adding \b -D_LCD_DEFINITIONS_FILE to the \b CDEFS section in the Makefile.
 * All definitions added to the file lcd_definitions.h will override the default
definitions from lcd.h
 */
#ifndef LCD_DELAY_BOOTUP
#define LCD_DELAY_BOOTUP   16000      /**< delay in micro seconds after power-on  */
#endif
#ifndef LCD_DELAY_INIT
#define LCD_DELAY_INIT      5000      /**< delay in micro seconds after initialization
command sent  */
#endif
#ifndef LCD_DELAY_INIT_REP
#define LCD_DELAY_INIT_REP    64      /**< delay in micro seconds after initialization
command repeated */
#endif
#ifndef LCD_DELAY_INIT_4BIT
#define LCD_DELAY_INIT_4BIT   64      /**< delay in micro seconds after setting 4-bit
mode */
#endif
#ifndef LCD_DELAY_BUSY_FLAG
#define LCD_DELAY_BUSY_FLAG    4      /**< time in micro seconds the address counter
is updated after busy flag is cleared */
#endif
#ifndef LCD_DELAY_ENABLE_PULSE
#define LCD_DELAY_ENABLE_PULSE 1      /**< enable signal pulse width in micro seconds
*/
#endif


/**
 * @name 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.
```

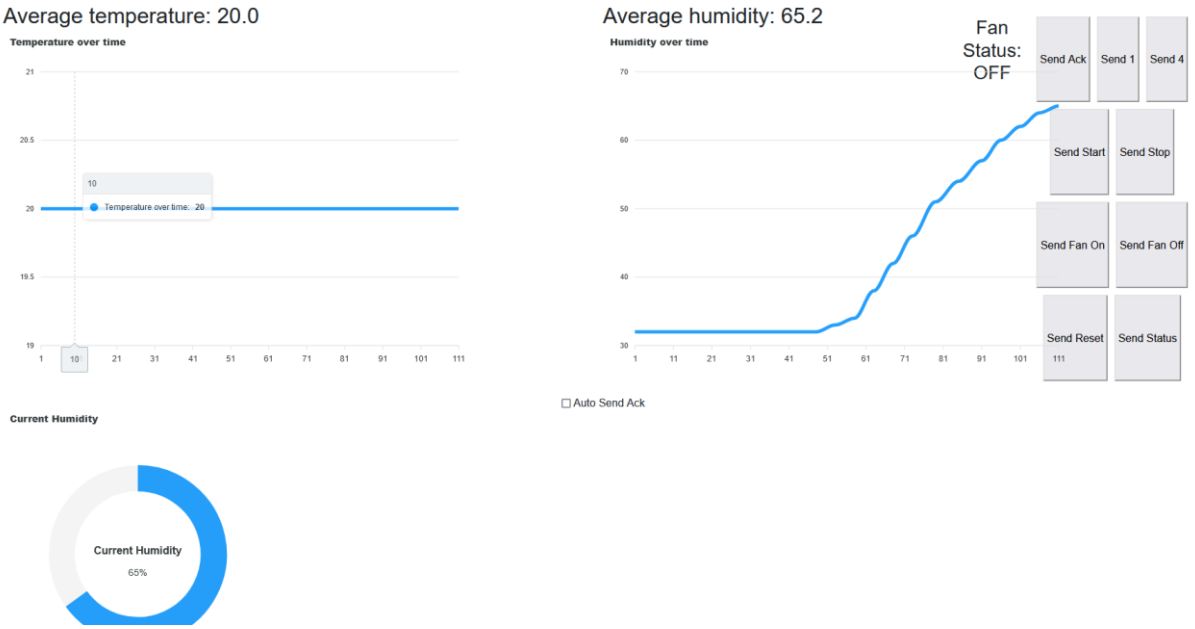
```
 */

/* instruction register bit positions, see HD44780U data sheet */
#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                    */

/* set entry mode: display shift on/off, dec/inc cursor move direction */
#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 */

/* display on/off, cursor on/off, blinking char at cursor position */
#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   */

/* move cursor/shift display */
#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                 */

/* function set: set interface data length and number of display lines */
#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) )



/**
 *  @name Functions
 */


/**
 @brief    Initialize display and select type of cursor
 @param    dispAttr \b LCD_DISP_OFF display off\n
                    \b LCD_DISP_ON display on, cursor off\n
                    \b LCD_DISP_ON_CURSOR display on, cursor on\n
```

```
                    \b LCD_DISP_ON_CURSOR_BLINK display on, cursor on flashing
 @return  none
*/
extern void lcd_init(uint8_t dispAttr);


/**
 @brief    Clear display and set cursor to home position
 @return   none
*/
extern void lcd_clrscr(void);


/**
 @brief    Set cursor to home position
 @return   none
*/
extern void lcd_home(void);


/**
 @brief    Set cursor to specified position

 @param    x horizontal position\n (0: left most position)
 @param    y vertical position\n   (0: first line)
 @return   none
*/
extern void lcd_gotoxy(uint8_t x, uint8_t y);


/**
 @brief    Display character at current cursor position
 @param    c character to be displayed
 @return   none
*/
extern void lcd_putc(char c);


/**
 @brief    Display string without auto linefeed
 @param    s string to be displayed
 @return   none
*/
extern void lcd_puts(const char *s);


/**
 @brief    Display string from program memory without auto linefeed
 @param    progmem_s string from program memory be be displayed
 @return   none
 @see      lcd_puts_P
*/
extern void lcd_puts_p(const char *progmem_s);


/**
 @brief    Send LCD controller instruction command
 @param    cmd instruction to send to LCD controller, see HD44780 data sheet
 @return   none
*/
extern void lcd_command(uint8_t cmd);
```

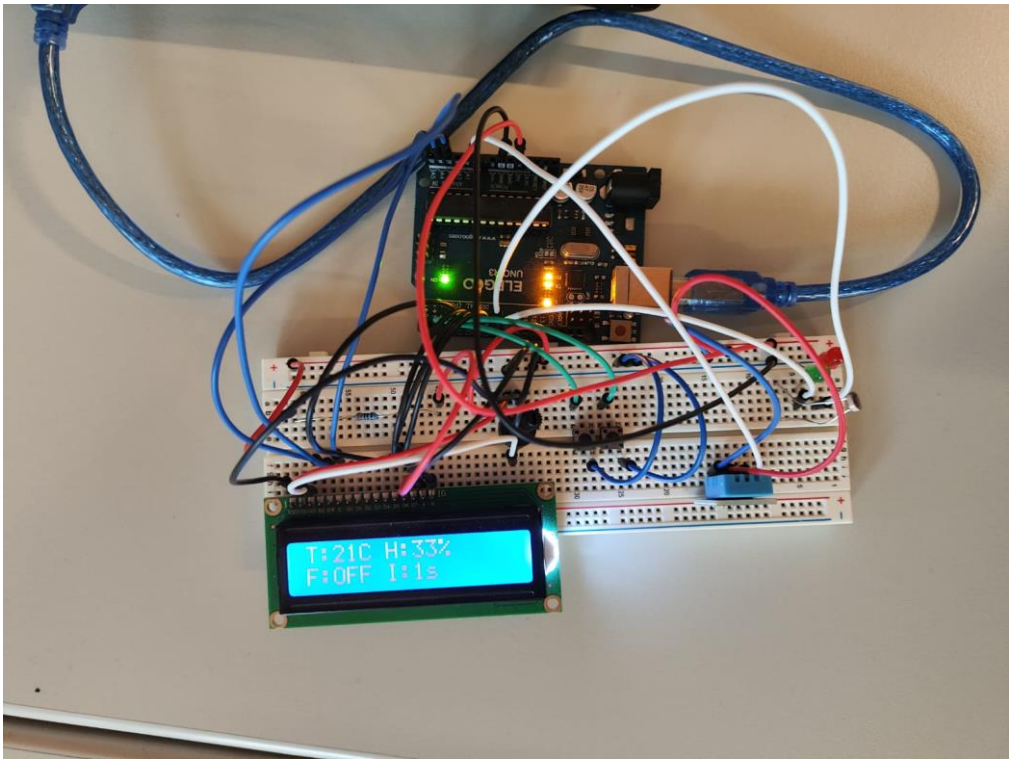
```
/**
 @brief    Send data byte to LCD controller

 Similar to lcd_putc(), but without interpreting LF
 @param    data byte to send to LCD controller, see HD44780 data sheet
 @return   none
*/
extern void lcd_data(uint8_t data);


/**
 @brief macros for automatically storing string constant in program memory
*/
#define lcd_puts_P(__s)         lcd_puts_p(PSTR(__s))

/**@}*/

#endif //LCD_H
```

7 Ergebnis

Blazor Visualisierung:



Breadboard:



8 Kommentar

Die Übung war *frrrrrrrr* FIRE