7 Uhr und Zeitraster

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

7. Uhr und Zeitraster

ACHTUNG

Diese Seite ist gerade in Bearbeitung ...

Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie man eine einfache Menüführung auf einem Display implementiert.

Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. mexleuhr_spi.simu
    2. mexleuhr_master.hex
    3. mexleuhr_master.zip

II. Analyse des fertigen Programms

1. Initialisieren des Programms
    1. Öffnen Sie SimulIDE und öffnen Sie dort mittels  die Datei `mexleuhr_spi.simu`. In der Simulation sind einige Änderungen zu finden:
        1. Die Schalter sind an den Pins C0...C3 angeschlossen, statt an den Pins B1...B4. Viele der Pins haben - neben er Möglichkeit digitale Werte auszugeben - weitere Funktionen. Im Datenblatt ist diese Belegung beschrieben:
            - Bei PB2 steht auch $\overline{SS}$ für “Slave Select”
            - Bei PB3 steht auch $MOSI$ für “Master out, slave in”
            - Bei PB4 steht auch $MISO$ für “Master in, slave out”
            - zusätzlich steht bei PB5 steht auch SCK für “serial clock”
        2. Diese Anschlüsse sind an einem weiteren Display “Pcd8544” angeschlossen. Zusätzlich ist PB1 an “D/C” des Displays angeschlossen
        3. PB3 ($MOSI$) ist zusätzlich an einem Oszilloskop und einer Probe angeschlossen. Die Probe ist mit einem Plotterkanal verbunden.
    2. Laden Sie `mexleuhr_master.hex` als firmware auf den 328 Chip
    3. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    4. Als nächstes ist im Display eine Uhr mit dem Format HH:MM:SS Menu zu sehen
    5. Die Tasten $S2$ und $S3$ ermöglichen das Einstellen der Stunde und Minute.
    6. Bleibt die Taste $S1$ gedrückt, so werden die Zehntelsekunden auf dem Display Pcd8544 ausgegeben.
    7. Beim Druck auf die Taste $S4$ wird eine Linie zwischen zwei zufälligen Punkten gezeichnet
2. Das Programm zu diesem Hexfile soll nun erstellt und erklärt werden

III. Eingabe in Atmel Studio

```
/*
================================================================================

Experiment 7:   MiniMEXLE-Uhr mit hh:mm:ss-Anzeige und SPI-Master
=============
================================================================================

Dateiname:      MEXLEuhr_Master.c

Autoren:        Prof. T. Fischer (Hochschule Heilbronn)
                Prof. G. Gruhler (Hochschule Heilbronn)
                D. Chilachava     (Georgische Technische Universitaet)

Version:        0.2 vom 23.05.2020

Hardware:       Simulide

Software:       Atmel Studio Ver. 7.xx

Funktion:       Digitaluhr mit Anzeige von Stunden, Minuten und Sekunden. Eine
                einfache Stellfunktion ist mit den Tasten S2 und S3 realisiert.
                Mit S1 und S4 kann die SPI-Kommunikation mit einem Slave-Display
                gestartet werden
```

```
/*================================================================================

=
```

Ändern Sie auch hier wieder die Beschreibung am Anfang des C-Files, je nachdem was Sie entwickeln

```
Displayanzeige: Start (fuer 2s):        Betrieb:
                                +-----------------+        +-------------------+
                                | MEXLEuhr - SPI  |        |=== 00:00:00 ===|
                                |     Master      |        |10tl Std Min Lin|
                                +-----------------+        +-------------------+

Tastenfunktion: S1: uebertraegt die Zehntelsekunde vom Master zum Slave
                S2:    Std (zaehlt Stunden bei Flanke aufwaerts. Ueberlauf bei 24)
                S3: Min (zaehlt Minuten bei Flanke aufwaerts. Ueberlauf bei 60) (setzt Sekunden beim Druecken zurueck auf 00)
                S4: uebertraegt die Info zum Darstellen einer Linie zum Master

Fuses im uC:    CKDIV8: Aus (keine generelle Vorteilung des Takts)

Header-Files:   lcd_lib_de.h (Library zur Ansteuerung LCD-Display Ver. 1.2)

Libraries:      pcd8544.c (Library        fuer die Ansteuerung des Displays)
                pcd8544.h (Header-Datei    fuer die Ansteuerung des Displays)

Module:         1) Taktgenerator
                2) Zaehler fuer Uhr    (Takt: 1 s)
                3)
```

Deklarationen
==========

1. Hier wird wieder geprüft ob die Frequenz des Quarz bereits eingestellt wurde und - falls nicht - dessen Frequenz eingestellt. Die Frequenz ist diesmal merklich niedriger, da die Ansteuerung des Display keine höheren Raten erlaubt

2. Zusätzlich zu den bisherigen Header-Dateien sind nun folgende hinzugekommen:

    1. `<stdlib.h>` - Standard-Bibliothek für Typenumwandlung und mehr. Hiervon wird die Erstellung von Zufallswerten genutzt

    2. “`pcd8544.h`” - Bibliothek für das einbinden des neuen Displays

3. Die Makros entsprechen denen der letzten Programme.

4. Die Konstanten entsprechen im Wesentlichen denen der letzten Programme. Der Vorteiler Wert entspricht aber hier der Hälfte des bisherigen Wertes, da die Taktfrequenz ebenso halb so groß ist.

5. Für die Zeichen `0` und `:` werden für die ASCII-Codes Makros definiert. Dadurch wird das Lesen des am Display ausgegebenen Textes im Code einfacher.

6. Bei den Variablen entsprechen einige denen der letzten Programme.

7. Für die Uhr werden Stunden, Minuten,

```
Anzeigetreiber     (Takt:
100 ms)
                4)
Stellfunktion      (Takt: 10
ms)
                5) SPI-
Funktionen

    Die Kopplung der Module
wird ueber global definierte
Variable realisiert:

    1-Bit-Variable:
takt10ms:    Taktgenerator
=> Stellfunktion
takt100ms:    Taktgenerator
=> Anzeigetreiber
takt1s:        Taktgenerator
=> Zaehler fuer Uhr

    8-Bit-Variable:
sekunden    Stellfunktion =>
Zaehler => Anzeige
minuten
stunden

============================
============================
====================*/


// Deklarationen
============================
============================
======

// Festlegung der
Quarzfrequenz

#ifndef F_CPU
// optional definieren
#define F_CPU 6144000UL
// MiniMEXLE mit 6,144 MHz
Quarz
#endif


// Include von Header-
Dateien

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

Sekunden und Zehntelsekunden mit Anfangswerten deklariert.

8. Für das neue Display werden Variablen für die Textposition und für das auszugebenden Zeichen deklariert.

9. Das Flag PcdSendMessage zeigt an, ob Zeichen regelmäßig zu übertragen sind.

10. Bei den Funktionsprototypen sind einige bekannte Unterprogramme vorhanden. Details werden weiter unten erklärt.

```
Hauptprogramm ========================
```

1. Zunächst werden zwei Initialisierungsroutinen aufgerufen (siehe weiter unten)
2. Dann werden wieder “Timer/Counter Control Register” und “Timer Interrupt MaSK” konfiguriert.
3. Die “Data Direction Register” wurden auch bereits beschrieben. Diese werden hier so konfiguriert, dass zwei Anschlüsse für Lautsprecher und LED als Ausgang definiert sind.

4. Auch die Konfiguration der Anschlüsse für die Schalter wurde bereits erklärt. Die an Port C angeschlossenen Taster erhalten dadurch einen Pull-up Widerstand.
5. Mit dem Befehl `sei()` wird die Bearbeitung von Interrupts aktiv.

```
// I/O-Konfiguration (intern
weitere Dateien)
#include <stdbool.h>
// Bibliothek fuer Bit-
Variable
#include <stdlib.h>
// Bibliothek fuer Bit-
Variable
#include <avr/interrupt.h>
// Definition von Interrupts
#include <util/delay.h>
// Definition von Delays
(Wartezeiten)
#include "lcd_lib_de.h"
// Header-Datei fuer LCD-
Anzeige
#include "pcd8544.h"
// Header Datei des Displays

// Makros

#define SET_BIT(PORT, BIT)
((PORT) |=  (1 << (BIT))) //
Port-Bit Setzen
#define CLR_BIT(PORT, BIT)
((PORT) &= ~(1 << (BIT))) //
Port-Bit Loeschen
#define TGL_BIT(PORT, BIT)
((PORT) ^=  (1 << (BIT))) //
Port-Bit Toggeln

// Konstanten

#define VORTEILER_WERT
30            // Faktor
Vorteiler = 90
#define HUNDERTSTEL_WERT
10             // Faktor
Hundertstel = 10
#define ZEHNTEL_WERT
10             // Faktor
Zehntel = 10

#define SPEAK_PORT
PORTD        // Port-Adresse
fuer Lautsprecher
#define SPEAK_BIT
5            // Port-Bit
fuer Lautsprecher
#define LED_PORT
PORTB        // Port-Adresse
```

6. In der Endlosschleife sind verschiedene Zeitzyklen vorgesehen (wie beim Up/Down Counter).
    1. im $10ms$ Raster (auch $10ms$ Zyklus genannt) wird die Unterfunktion `uhrStellen` aufgerufen.
    2. im $100ms$ Raster werden die Unterfunktionen `uhrAnzeigen` und `UhrZaehlen` aufgerufen. Davor wird, falls das Flag `PcdSendMessage` gesetzt ist, wird dieses zurückgesetzt, die LED blinkt und das Unterprogramm `zehntelSekundenAufPcdAnzeigen` wird aufgerufen.
    3. im $1s$ Raster blinkt die LED und das Unterprogramm `pcd_init` zum initialisieren des PCD Displays wird aufgerufen.

Interrupt Routine

1. Mit dem Befehl `ISR()` wird wieder die Interrupt Service Routine für den OVerFlow Interrupt des TIMER0 angelegt.
2. Die Ermittlung von `Timertick`, `vorteiler`, `takt10ms`, `hundertstel` und `takt100ms` ist hier wieder gleich dem im Up/Down Counter.
3. eine Erweiterung auf `zehntel` und `takt1s` wurde hier mit eingefügt.

Stellfunktion

1. Das Einstellen des Data Direction Registers und der Pullups wurde bereits in vorherigen

```
fuer LED
#define LED_BIT
0            // Port-Bit
fuer gelbe LED an PB2

#define ASC_NULL
0x30        // Das Zeichen
'0' in ASCII
#define ASC_COLON
0x3A        // Das Zeichen
':' in ASCII

// Variable

unsigned char vorteiler
= VORTEILER_WERT;    //
Zaehlvariable Vorteiler
unsigned char hundertstel
= HUNDERTSTEL_WERT; //
Zaehlvariable Hundertstel
unsigned char zehntel
= ZEHNTEL_WERT;        //
Zaehlvariable Zehntel

unsigned char
zehntelSekunden = 0;      //
Variable Sekunden
unsigned char sekunden
= 56;        // Variable
Sekunden
unsigned char minuten
= 34;        // Variable
Minuten
unsigned char stunden
= 12;        // Variable
Stunden

unsigned char zeile
= 0;        // x-Koordinate
unsigned char pos
= 0;        // y-Koordinate
unsigned char zeichen
='a'-1;        //
auszugebendes Zeichen

bool timertick;
// Bit-Botschaft alle
0,111ms (bei Timer-
Interrupt)
bool takt10ms;
// Bit-Botschaft alle 10ms
```

Programmen erklärt.

`Funktion Tasten einlesen ==========`

1. In dieser Funktion werden zunächst die Stellungen aller Taster eingelesen (vgl. `counterCounting(void)` bei Up/down Counter).

2. Die Flankenerkennung in den if-Bedingungen wurde auch bereits beim Up/down Counter erklärt.
3. Wenn die Taste S1 gedrückt ist, so wird das Flag `PcdSendMessage` gesetzt, welches in `main` zum Aufrufen des Unterprogramm `zehntelSekundenAufPcdAnzeigen` in jedem $100ms$ Raster führt.
4. Die Tasten S2 und S3 führen zum einmaligem Hochzählen der Stunden bzw. Minuten. Wenn der jeweilige Wert über das Maximum hinausläuft, so beginnt dieser wieder beim Minimum.
5. Die Taste S4 zeichnet eine Linie mittels `pcd_putLine` und aktualisiert das Displays des PCD8544

`Anzeigefunktion Uhr ========================`

1. Die Funktion `initDisplay()` wird zu Beginn des Programms aufgerufen und führt zunächst

```
bool takt100ms;
// Bit-Botschaft alle 100ms
bool takt1s;
// Bit-Botschaft alle 1s

bool sw1_neu = 1;
// Bitspeicher fuer Taste 1
bool sw2_neu = 1;
// Bitspeicher fuer Taste 2
bool sw3_neu = 1;
// Bitspeicher fuer Taste 3
bool sw4_neu = 1;
// Bitspeicher fuer Taste 4
bool sw1_alt = 1;
// alter Wert von Taste 1
bool sw2_alt = 1;
// alter Wert von Taste 2
bool sw3_alt = 1;
// alter Wert von Taste 3
bool sw4_alt = 1;
// alter Wert von Taste 4

bool PcdSendMessage = 0;
// Flag fuer sendebereite
SPI-Nachricht

// Funktionsprototypen

void timerInt0(void);
// Init Zeitbasis mit Timer
0
void uhrStellen(void);
// Stellfunktion
void uhrAnzeigen(void);
// Anzeigefunktion
void uhrZaehlen(void);
// Uhrfunktion
void initDisplay(void);
// Init Anzeige
void
zehntelSekundenAufPcdAnzeige
n(void);//

// Hauptprogramm
============================
============================
======

int main()
{
    // Initialisierung
```

die Initialisierung des Displays aus.
2. Danach wird der erste Text auf den Bildschirm geschrieben und damit der Programmname dargestellt.
3. Nach zwei Sekunden wird der Auswahlbildschirm angezeigt.

`Anzeige Hauptmenu`
`==========`

1. Da der Auswahlbildschirm mit dem Hauptmenu nicht nur beim Start, sondern auch nach jeder Rückkehr aus Unterprogrammen dargestellt werden muss, wird der Auswahlbildschirm in einem neuen Unterprogramm angezeigt.

`/* Teilprogramm 1: Blinkende LED =====`

Hier ist das Programm der Blinking LED etwas angepasst eingefügt.

1. Zunächst wird ein Unterprogramm zur Anzeige das Displays aufgerufen
2. `SET_BIT(DDRB, DDB0)` wandelt den Anschluss B0 in einen Ausgang um
3. Die Schleife wird solange ausgeführt, bis die Flanke des Schalters 1 über `sw1_slope`

```
    initDisplay();
// Initialisierung LCD-
Anzeige
    pcd_init();

    TCCR0A = 0;
// Timer 0 auf "Normal Mode"
schalten
    SET_BIT(TCCR0B, CS01);
// mit Prescaler /8
betreiben
    SET_BIT(TIMSK0, TOIE0);
// Overflow-Interrupt
aktivieren

    SET_BIT(DDRD,
SPEAK_BIT);            //
Speaker-Bit auf Ausgabe
    PORTC |= 0b00001111;
// Taster Anschluesse auf
Pullup R
    SET_BIT(DDRB, LED_BIT);
// LED-Bit auf Ausgabe

    sei();
// generell Interrupts
einschalten
    // Hauptprogrammschleife

    while(1)
// unendliche Warteschleife
mit Aufruf der
// Funktionen abhaengig von
Taktbotschaften
    {
        if (takt10ms)
// alle 10ms:
        {
            takt10ms = 0;
//        Botschaft "10ms"
loeschen
            uhrStellen();
//        Tasten abfragen,
Stellen, SPI-Komm.
        }

        if (takt100ms)
// alle 100ms:
        {
            takt100ms = 0;
//        Botschaft "100ms"
```

erkannt wurde

4. Beim Aktivieren der LED wird auch auf dem Display eine 1 geschrieben.

5. Nach einer Sekunde wird die LED ausgeschalten und auf dem Display eine geschrieben.

6. Nach Beendigung der Schleife werden alle Flanken gelöscht. Damit wird verhindert, dass beim Aufruf des Hauptmenus sofort ein Sprung in ein Unterprogramm ausgeführt wird.

```
/* Teilprogramm 2: Soundgenerierung
====
```

Hier ist das Programm Sound und Timer etwas angepasst eingefügt.

```
loeschen
            if
(PcdSendMessage)
// wenn SPI-Nachricht
gesendet werden soll:
            {
PcdSendMessage = 0;
// Botschaft loeschen
TGL_BIT(LED_PORT, LED_BIT);
// LED Zustand wechseln
zehntelSekundenAufPcdAnzeige
n();// Anzeige auf PCD
Display
            }
            uhrAnzeigen();
//        Uhrzeit auf
Anzeige ausgeben
            uhrZaehlen();
//        Uhr weiterzaehlen
        }

        if (takt1s)
// alle Sekunden:
        {
            takt1s = 0;
//        Botschaft "1s"
loeschen
TGL_BIT(LED_PORT, LED_BIT);
// LED Zustand wechseln
            pcd_init();
        }
    }
    return 0;
}


// Interrupt-Routine
============================
============================
==

ISR (TIMER0_OVF_vect)

/*  In der Interrupt-Routine
sind die Softwareteiler
realisiert, die die Takt-
    botschaften (10ms,
100ms, 1s) fuer die gesamte
Uhr erzeugen. Die Interrupts
    werden von Timer 0
ausgeloest (Interrupt Nr. 1)
```

1. Die Port Initialisierung, um Lautsprecher und LED anzusteuern, wurde übernommen.
2. Hier wird Timer 0 genutzt, um das gepulste Signal an den Lautsprecher zu verändern.
3. Die while-Schleife wird wieder abgebrochen, wenn die Taste 1 gedrückt wurde.
4. Neben dem Herunterzählen der Periodenlänge (über `OCR0A--`), wird auch der Periodenzähler ausgegeben. Die Ausgabe ähnelt `counterDisplay` aus dem Programm Up/Down Counter.
5. Da die for-Schleife zum Herunterzählen der Periodenlänge sehr lange dauert (etwa 2 Sekunden) wird auch darin der Tastendruck der Taste 1 abgefragt werden.
6. Falls die Taste 1 gedrückt wurde, wird sowohl in der for-Schleife, als auch nach der while-Schleife der Timer gestoppt und die Flanken zurückgesetzt.
7. Das Heraufzählen der Frequenz gleich dem Herunterzählen, bis auf die Werte der for-Schleife.

```
    Veraenderte Variable:
vorteiler
hunderstel
zehntel

    Ausgangsvariable:
takt10ms
takt100ms
takt1s
*/

{
    timertick = 1;
// Botschaft 0,111ms senden
    --vorteiler;
// Vorteiler dekrementieren
    if (vorteiler==0)
// wenn 0 erreicht: 10ms
abgelaufen
    {
        vorteiler =
VORTEILER_WERT;        //
Vorteiler auf Startwert
        takt10ms = 1;
//    Botschaft 10ms senden
        --hundertstel;
//    Hunderstelzaehler
dekrementieren

        if (hundertstel==0)
// wenn 0 erreicht: 100ms
abgelaufen
        {
            hundertstel =
HUNDERTSTEL_WERT; // Teiler
auf Startwert
            takt100ms = 1;
//    Botschaft 100ms senden
            --zehntel;
//    Zehntelzaehler
dekrementieren

            if (zehntel==0)
// wenn 0 erreicht: 1s
abgelaufen
            {
                zehntel =
ZEHNTEL_WERT;    //
Teiler auf Startwert
                takt1s = 1;
//    Botschaft 1s senden
```

```
/* Teilprogramm 3: Logische Funktionen
====
```

Hier ist das Programm Logische Funktionen etwas angepasst eingefügt.

1. Durch den Anschluss des Tasters zwischen Port und Masse erzeugt ein geschlossener ein LOW Signal (logisch 0). Hier sollen aber nun der Tastendruck dem Wert HIGH (logisch 1) entsprechen. Aus diesem Grund sind die Tasterwerte in den Bedingungen negiert, z.B. `(!sw3_alt)&&(!sw4_alt)`

```
            }
        }
    }
}

// Stellfunktion
============================
============================
======

void uhrStellen(void)

/*  Die Stellfunktion der
Uhr wird alle 10ms
aufgerufen. Dadurch wir eine
    Entprellung der
Tastensignale realisiert.
Das Stellen wir bei einer
    fallenden Flanke des
jeweiligen Tastensignals
durchgefuehrt. Darum
    muss fuer einen weiteren
Stellschritt die Taste
erneut betaetigt werden.
    Ebenso wird die SPI-
Funktion hier aufgerufen.

    Eine Flanke wird durch
(alter Wert == 1) UND
(aktueller Wert == 0)
erkannt.

    Mit der Taste S1 wird
die Uebergabe der Zeit
Master > Slave gestartet
    Mit der Taste S2 werden
die Stunden aufwaerts
gestellt.
    Mit der Taste S3 werden
die Minuten aufwaerts
gestellt (kein Uebertrag)
    Solange Taste S3
gedrueckt ist werden die
Sekunden auf 00 gehalten
    Mit der Taste S4 wird
die Uebergabe der Zeit
Master < Slave gestartet

    Veraenderte Variable:
stunden
minuten
```

```
/* Teilprogramm 4: Up-Down-Counter ====
```

Hier ist das Programm Up/Down Counter etwas angepasst eingefügt.

1. Im wesentlichen gleicht das Programm dem bereits bekanntem. Es kann aber auf die bereits berechnete Flanken `sw2_slope` bis `sw4_slope` zurückgegriffen.

```
sekunden

    Speicher fuer Bits:
sw1Alt
sw2Alt
sw3Alt
sw4Alt
*/

{
    sw1_neu = (PINC & (1 <<
PC0));        // Tasten von
Port einlesen
    sw2_neu = (PINC & (1 <<
PC1));
    sw3_neu = (PINC & (1 <<
PC2));
    sw4_neu = (PINC & (1 <<
PC3));

    if ((sw1_neu==0))
// wenn Taste 1 gedrueckt
ist:
    {
        PcdSendMessage = 1;
//    Senden der SPI-
Nachricht aktivieren
    }
    if
((sw2_neu==0)&(sw2_alt==1))
// wenn Taste 2 eben
gedrueckt wurde:
    {
        stunden++;
//    Stunden hochzaehlen,
Ueberlauf bei 23
        if (stunden==24)
            stunden = 00;
    }
    if
((sw3_neu==0)&(sw3_alt==1))
// wenn Taste 3 eben
gedrueckt wurde:
    {
        minuten++;
//    Minuten hochzaehlen,
Ueberlauf bei 59
        if (minuten==60)
            minuten = 00;
    }
    if (sw3_neu==0)
```

Auswahl im Hauptmenu ermitteln
==========

1. Je nach gedrückter Taste wird hier die Variable `modus` gesetzt

```
// solange Taste 3
gedrueckt:
    sekunden = 00;
//       Sekunden auf 00
setzen

    if
((sw4_neu==0)&(sw4_alt==1))
// wenn Taste 3 eben
gedrueckt wurde:
    {
pcd_putLine(rand()%83,rand()
%47,rand()%83,rand()%47);
        pcd_updateDisplay();
    }
    sw1_alt = sw1_neu;
// aktuelle Tastenwerte
speichern
    sw2_alt = sw2_neu;
//    in Variable fuer alte
Werte
    sw3_alt = sw3_neu;
    sw4_alt = sw4_neu;
}

// Anzeigefunktion Uhr
============================
============================

void uhrAnzeigen(void)

/*  Die Umrechnung der
binaeren Zaehlwerte auf BCD
ist folgendermaßen geloest:
    Zehner: einfache
Integer-Teilung (/10)
    Einer:  Modulo-
Ermittlung (%10), d.h. Rest
bei der Teilung durch 10
*/

{
    lcd_gotoxy(0,4);
// Cursor auf Start der
Zeitausgabe setzen

    lcd_putc(ASC_NULL +
stunden/10);    // Stunden
Zehner als ASCII ausgeben
    lcd_putc(ASC_NULL +
stunden%10);    // Stunden
```

```
Einer als ASCII ausgeben
    lcd_putc(ASC_COLON);
// Doppelpunkt ausgeben

    lcd_putc(ASC_NULL +
minuten/10);    // Minuten
als ASCII ausgeben
    lcd_putc(ASC_NULL +
minuten%10);    //
    lcd_putc(ASC_COLON);
// Doppelpunkt ausgeben

    lcd_putc(ASC_NULL +
sekunden/10);    // Sekunden
als ASCII ausgeben
    lcd_putc(ASC_NULL +
sekunden%10);    //
}

// Anzeigefunktion fuer PCD
Display
============================
============================

void
zehntelSekundenAufPcdAnzeige
n(void)

/*  Anzeigen der
Zenhtelsekunden auf dem
Display PCD8544
*/

{
    pcd_gotoxy(zeile, pos);
// Setze Position am Display
pcd_putc(zehntelSekunden+0x3
0);        // Schreibe
Zehntelsekunden
    pcd_updateDisplay();
// Aktualisiere das Display
des PCD8544
    if (++pos > 13)
// naechste Position, und
wenn diese ausserhalb der
Anzeige
    {
        pos = 0;
// zurueck auf erste
Position
        if (++zeile > 5)
```

```
// naechste Zeile, und wenn
diese ausserhalb der Anzeige
        {
            zeile = 0;
// zurueck auf erste Zeile
pcd_clearDisplay();
// loesche Anzeige
        };
    }
}

// Initialisierung Display-
Anzeige
============================
================

void initDisplay()
// Start der Funktion
{
    lcd_init();
// Initialisierungsroutine
aus der lcd_lib
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("- Experiment
7a-");        // Ausgabe
Festtext: 16 Zeichen

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("Uhr + SPI-
Master");        // Ausgabe
Festtext: 16 Zeichen

    _delay_ms(1000);
// Wartezeit nach
Initialisierung

    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("=== 00:00:00
===");        // Ausgabe
Festtext: 16 Zeichen

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("10tl Std Min
```

```
Lin.");        // Ausgabe
Festtext: 16 Zeichen

}
// Ende der Funktion


// Zaehlfunktion Uhr
============================
============================
==

void uhrZaehlen (void)
// wird jede Sekunde
gestartet

/*  Die Uhr wird im
Sekundentakt gezaehlt. Bei
jedem Aufruf wird auch ein
    "Tick" auf dem
Lautsprecher ausgegeben.
Ueberlaeufe der Sekunden
zaehlen
    die Minuten, die
Ueberlaeufe der Minuten die
Stunden hoch.

    Veraenderte Variable:
sekunden
minuten
stunden
*/

{
    TGL_BIT (SPEAK_PORT,
SPEAK_BIT);    // "Tick" auf
Lautsprecher ausgeben
// durch Invertierung des
Portbits
    zehntelSekunden++;
    if (zehntelSekunden==10)
// bei Ueberlauf:
    {
        zehntelSekunden=0;
        sekunden++;
// Sekunden hochzaehlen
        if (sekunden==60)
// bei Ueberlauf:
        {
            sekunden = 0;
//   Sekunden auf 00 setzen
```

```
            minuten++;
//   Minuten hochzaehlen
            if (minuten==60)
//     bei Ueberlauf:
            {
                minuten = 0;
//       Minuten auf 00
setzen
                stunden++;
//       Stunden hochzaehlen
                if
(stunden==24)        //
bei Ueberlauf:
                    stunden
= 0;        //
Stunden auf 00 setzen
            }
        }
    }
}
```

IV. Ausführung in Simulide

1. Geben Sie die oben dargestellten Codezeilen ein und kompilieren Sie den Code.
2. Öffnen Sie Ihre hex-Datei in SimulIDE und testen Sie, ob diese die gleiche Ausgabe erzeugt

Bitte arbeiten Sie folgende Aufgaben durch:

Aufgaben

1. Speicherauslastung und Programmoptimierung:
    1. Merken Sie sich die Speicherauslastung des bisherigen Programms. Diese finden Sie z.B. über den Solution Explorer: `Output Files` » `5_Program_Menu.elf` » rechte Maustaste (Kontextmenu) » `Properties` » `Flash size` und `RAM size` (in Bytes).
    2. Der oben gezeigte Code wurde optimiert: 5_program_menu_opt.c . Aus funktionaler Sicht sind die beiden Programme gleich. Kompilieren Sie diesen Code und überprüfen Sie die Speicherauslastung.
    3. Wie funktioniert die optimierte Funktionen `void getChoiceInMainMenu()`?
    4. Für was wird der Array `DisplayText` verwendet?

<--