5 Menüführung

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

5 Menüführung

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. 5._menuefuehrung.sim1
    2. 5._menufuehrung.hex
    3. lcd_lib_de.h

II. Analyse des fertigen Programms

1. Initialisieren des Programms
    1. Öffnen Sie SimulIDE und öffnen Sie dort mittels  die Datei `5._menuefuehrung.sim1`
    2. Laden Sie `5._menuefuehrung.hex` als firmware auf den 88 Chip
    3. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    4. Als nächstes ist im Display ein Menu zu sehen, in dem verschiedene Programme P1 ... P4 durch Tastendruck auswählbar ist. Dadurch sind die bisherigen Programme auswählbar. Im Unterprogramm ermöglicht der Schalter S1 das Zurückspringen ins Menu.
2. Das Programm zu diesem Hexfile soll nun erstellt werden

III. Eingabe in Atmel Studio

```
/*============================================================
============================
Experiment 5:   Programm-Menu
=============
=============

Dateiname:    Program_Menu.c
Autoren:        Peter Blinzinger
                Prof. G. Gruhler (Hochschule Heilbronn)
                D.
```

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

```
Chilachava    (Georgische
Technische Universitaet)
Version:        1.2 vom
29.04.2020
Hardware:      MEXLE2020
Ver. 1.0 oder höher
                AVR-USB-
PROGI Ver. 2.0
Software:
Entwicklungsumgebung:
AtmelStudio 7.0
                C-Compiler:
AVR/GNU C Compiler 5.4.0
Funktion:      Unter einer
gemeinsamen
Programmoberflaeche werden
vier Teil-
                programme
verwaltet. Dies sind:
                P1: Blinking
LED
                P2: Creating
Sound
                P3: Logic
Functions
                P4: Up/Down-
Counter
                Der Start
der Teilprogramme erfolgt
den zugeordneten Funktions-
                tasten. Nach
dem Abbruch eines
Teilprogramms (immer mit S1)
                wird wieder
die Programmauswahl
gestartet.
Displayanzeige: Start (fuer
2s):        Betrieb
(Hauptebene):
                +----------
-----+      +--------------
-+
                |-
Experiment 5 -|      |
Main Level   |
                |  Program
Menu  |      | P1  P2  P3
P4 |
                +----------
-----+      +--------------
-+
```

Deklarationen
==========

1. Hier wird wieder geprüft ob die Frequenz des Quarz bereits eingestellt wurde und - falls nicht - dessen Frequenz eingestellt.
2. Die Header-Dateien entsprechen denen der letzten Programme.
3. Auch die Makros entsprechen denen der letzten Programme.
4. Die Konstanten entsprechen denen der letzten Programme.
5. Auch die anfänglichen Variablen entsprechen denen der letzten Programme. Hierbei sind alle vier Schalter berücksichtigt.
6. Wird die Taste S1 gedrückt, so wird `sw1_neu` gesetzt. `sw1_alt` entspricht dem vorherigen Wert. Gleiches gibt es für die anderen Taster.

```
                Anzeige fuer
Teilprogramme siehe bei
einzelnen Programmen
Tastenfunktion: Im
Hauptprogramm rufen S1 .. S4
die 4 Teiprogramme auf.
                Im
Teilprogramm ist die
Funktion unterschiedlich
(siehe dort)
Jumperstellung: Auswirkung
nur im Teilprogramm "Sound":
                Schalter
muss fuer des Buzzer
zwischen geschlossen sein
Fuses im uC:    CKDIV8: Aus
(keine generelle Vorteilung
des Takts)
Header-Files:   lcd_lib_de.h
(Library zur Ansteuerung
LCD-Display Ver. 1.3)
============================
============================
====================*/
// Deklarationen
============================
============================ Hauptprogramm ========================
======
// Festlegung der
Quarzfrequenz
#ifndef F_CPU
// optional definieren
#define F_CPU 18432000UL
// ATmega 88 mit 18,432 MHz
Quarz
#endif
// Include von Header-
Dateien
#include <avr/io.h>
// I/O-Konfiguration (intern
weitere Dateien)
#include <stdbool.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-
```

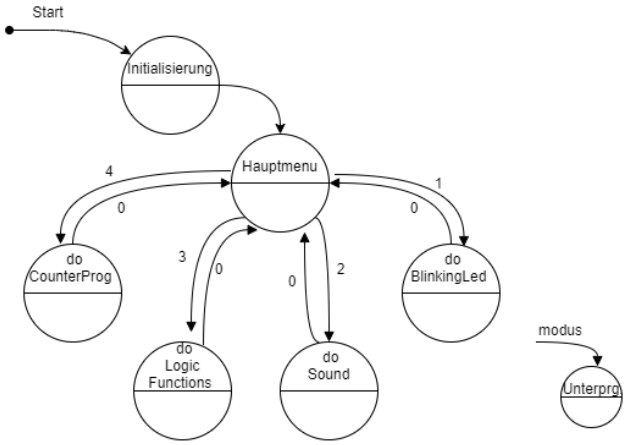
7. Wird eine ansteigende Flanke der Taste S1 gedrückt, so wird `sw1_slope` gesetzt. Das heißt, wenn die Taste gerade von 'nicht gedrückt' auf 'gedrückt' gewechselt hat, so wird `sw1_slope` gesetzt. Gleiches gibt es für die anderen Taster.

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

1. Zunächst werden zwei Initialisierungsroutinen aufgerufen (siehe weiter unten)
2. Dann werden die “Timer/Counter Control Register” des Timers 2 TCCR2A und TCCR2B gesetzt. Der Timer 2 ist im wesentlichen mit dem Timer 0 aus dem Up/Down Counter vergleichbar. Er ist ein 8-Bit Timer und auch hier wird der “Normal Mode” zum hochzählen genutzt. Auch hier gibt das Register TCCR2B den Prescaler an.
3. Auch hier gibt es eine “Timer Interrupt MaSK” TIMSK2. Auch hier wird mit dem Bit TOIE2 (“Timer Overflow Interrupt Enable”) der Interrupt bei Überlauf aktiviert.
4. Mit dem Befehl `sei()` wird die Bearbeitung von Interrupts aktiv
5. in der Endlosschleife ist nur eine switch-case Anweisung zu finden. Diese stellt den Auswahlteil einer Zustandsmaschine dar:

```
Anzeige
// Makros
#define SET_BIT(BYTE, BIT)
((BYTE) |=  (1 << (BIT))) //
Bit Zustand in Byte setzen
#define CLR_BIT(BYTE, BIT)
((BYTE) &= ~(1 << (BIT))) //
Bit Zustand in Byte loeschen
#define TGL_BIT(BYTE, BIT)
((BYTE) ^=  (1 << (BIT))) //
Bit Zustand in Byte wechseln
(toggle)
// Konstanten
#define VORTEILER_WERT
90  // Faktor Vorteiler = 60
#define HUNDERTSTEL_WERT
10  // Faktor Hundertstel =
10
#define ZEHNTEL_WERT
10  // Faktor Zehntel = 10
#define ON_TIME
100    // "Ein-Zeit" in
Inkrementen zu 100 ms
#define OFF_TIME
100    // "Aus-Zeit" in
Inkrementen zu 100 ms
#define MIN_PER
143    // minimale
Periodendauer in
"Timerticks"
#define MAX_PER
239    // maximale
Periodendauer in
"Timerticks"
#define WAIT_SND
2000// Wartezeit zwischen
zum Tonwechsel in ms
#define WAIT_LED
1000// Wartezeit zwischen
zum Blinkwechsel der LED in
ms

#define ASC_ZERO
0x30// ASCII-Zeichen '0'
#define ASC_ONE
0x31// ASCII-Zeichen '1'
// Variable
unsigned char vorteiler   =
VORTEILER_WERT;   //
Zaehlvariable Vorteiler
unsigned char hundertstel =
```



Aus jedem Unterprogramm wird wieder zurück ins Hauptmenü gesprungen.

6. Beim `case 1…4` wird zunächst das jeweilige Programm aufgerufen. Nachdem Rückkehr aus diesem Programm wird zunächst der `modus` wieder auf 0 zurückgesetzt, sodass beim nächsten Durchlauf der Schleife der `case 0` ausgeführt wird. Jeder case wird mit `break` beendet.

Interrupt Routine

1. Mit dem Befehl `ISR()` wird eine Interrupt Service Routine für den OVerFlow Interrupt für TIMER2 angelegt.
2. Der Überlauf-Interrupt durch den Timer2 wird erst bei Überlauf des 8-Bit Wert ausgeführt. Auch hier ergibt sich durch den Prescaler und Modus (TCCR2A und TCCR2B) eine Periode von $T_{\rm ISR}= 0,16\bar{6}~\rm ms$.
3. Die Ermittlung von `Timertick`, `vorteiler`, `takt10ms`, `hundertstel` und `takt100ms` ist hier wieder gleich dem im Up/Down Counter.
4. Eine große Änderung ist, dass bereits im Interrupt alle 10ms die Unterfunktion `readButton()` aufgerufen wird.

```
HUNDERTSTEL_WERT; //
Zaehlvariable Hundertstel
unsigned char modus       =
0;                //
Programmmodus
int counter = 0000;
// Variable fuer Zaehler
bool timertick;
// Bit-Botschaft alle
0,111ms (Timer-Interrupt)
bool takt10ms;
// Bit-Botschaft alle 10ms
bool takt100ms;
// Bit-Botschaft alle 100ms
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 sw1_slope = 0;
// Flankenspeicher fuer
Taste 1
bool sw2_slope = 0;
// Flankenspeicher fuer
Taste 2
bool sw3_slope = 0;
// Flankenspeicher fuer
Taste 3
bool sw4_slope = 0;
// Flankenspeicher fuer
Taste 4
// Funktionsprototypen
void initTimer0(void);
// Timer 0 initialisieren
(Soundgenerierung)
void initDisplay(void);
// Initialisierung des
Displays

void readButton(void);
```

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

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

2. Neu hier ist, dass über `if ( (sw1_neu==0) & (sw1_alt==1) )` die positive Flanke (=aufsteigende Flanke) erkannt wird und dies im Flag `sw1_slope` gespeichert wird.

`Initialisierung Display-Anzeige ==========================`

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

```
// Tasten einlesen
void
getChoiceInMainMenu(void);
// Hauptmenu bearbeiten
void showMainDisplay(void);
// Anzeige des Hauptmenus
void doBlinkingLed(void);
// Teilprogramm 1: Blinkende
LED
void
showBlinkingLedDisplay(void)
; // Anzeige zu Teilprogramm
1
void doSound(void);
// Teilprogramm 2:
Soundgenerierung
void showSoundDisplay(void);
// Anzeige zu Teilprogramm 2
void doLogicFunctions(void);
// Teilprogramm 3: Logische
Funktionen
void showLogicDisplay(void);
// Anzeige zu Teilprogramm 3
void doCounterProg(void);
// Teilprogramm 4: Zaehler
void
showCounterDisplay(void);
// Anzeige zu Teilprogramm 4
// Hauptprogramm
============================
============================
======
int main()
{
    initDisplay();
// Initialisierung LCD-
Anzeige
    TCCR2A = 0;
// Timer 2 auf "Normal
Mode": Basistakt
    TCCR2B |= (1<<CS01);
// mit Prescaler /8
betreiben
    TIMSK2 |= (1<<TOIE2);
// Overflow-Interrupt
aktivieren
    sei();
// generell Interrupts
einschalten
    while(1)
// unendliche Schleife
```

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` 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.

```
    {
        switch(modus)
// Programmverteiler:
Variable "modus"
        {
        case 0:
// Modus 0: Hauptmenu
showMainDisplay();
getChoiceInMainMenu();
            break;
        case 1:
// Modus 1: Blinkende LED
            doBlinkingLed();
// Programm laeuft bis zum
Abbruch
            modus = 0;
// danach auf Hauptmenu
zurueckschalten
            break;
        case 2:
// Modus 2: Soundgenerierung
            doSound();
// Programm laeuft bis zum
Abbruch
            modus = 0;
// danach auf Hauptmenu
zurueckschalten
            break;
        case 3:
// Modus 3: Logische
Funktionen
doLogicFunctions();
// Programm laeuft bis zum
Abbruch
            modus = 0;
// danach auf Hauptmenu
zurueckschalten
            break;
        case 4:
// Modus 4: Up-Down-Counter
            doCounterProg();
// Programm laeuft bis zum
Abbruch
            modus = 0;
// danach auf Hauptmenu
zurueckschalten
            break;
        }
    }
    return 0;
}
```

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

Hier ist das Programm Sound und Timer etwas angepasst eingefügt.

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.

```
// Interrupt-Routine
============================
============================
==
ISR(TIMER2_OVF_vect)
// In der Interrupt-Routine
sind die Softwareteiler
realisiert, durch die Takt-
// botschaften (10ms, 100ms)
erzeugt werden. Die
Interrupts werden von Timer
2
// ausgeloest.
{
    timertick = 1;
// Botschaft 0,166ms senden
    --vorteiler;
// Vorteiler dekrementieren
    if (vorteiler==0)
// wenn 0 erreicht: 10ms
abgelaufen
    {
        vorteiler =
VORTEILER_WERT; //
Vorteiler auf Startwert
        takt10ms = 1;
//    Botschaft 10ms senden
        readButton();
        --hundertstel;
//    Hunderstelzaehler
dekrementieren
    if (hundertstel==0)
// wenn 0 erreicht: 100ms
abgelaufen
        {
            hundertstel =
HUNDERTSTEL_WERT; // Teiler
auf Startwert
            takt100ms = 1;
//    Botschaft 100ms senden
        }
    }
}

// Funktion Tasten einlesen
============================
======================
void readButton(void)
{
    //   Bitposition im
Register:
```

7. Das Heraufzählen der Frequenz gleich dem Herunterzählen, bis auf die Werte der for-Schleife.

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

Hier ist das Programm Logische Funktionen etwas angepasst eingefügt.

```
    //            __76543210
    DDRC = DDRC &
0b11110000;        //
Zunaechst Port B auf Eingabe
schalten
    PORTC =
0b00001111;        //
Pullup-Rs eingeschaltet
    _delay_us(1);
// Umschalten der Hardware-
Signale abwarten

    // Einlesen der 4
Tastensignale
    sw1_neu = (PINC & (1 <<
PC0));
    sw2_neu = (PINC & (1 <<
PC1));
    sw3_neu = (PINC & (1 <<
PC2));
    sw4_neu = (PINC & (1 <<
PC3));
    DDRC = DDRC |
0b00001111;       // Am Ende
Port B wieder auf Ausgabe
schalten

    // Auswerten der Flanken
beim Druecken
    if
((sw1_neu==0)&(sw1_alt==1))
// wenn Taste 1 soeben
gedrueckt wurde:
        sw1_slope = 1;
//    Flankenbit Taste 1
setzen
    if
((sw2_neu==0)&(sw2_alt==1))
// wenn Taste 2 eben
gedrueckt wurde:
        sw2_slope = 1;
//    Flankenbit Taste 2
setzen
    if
((sw3_neu==0)&(sw3_alt==1))
// wenn Taste 3 eben
gedrueckt wurde:
        sw3_slope = 1;
//    Flankenbit Taste 3
setzen
    if
```

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)`

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

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

```
((sw4_neu==0)&(sw4_alt==1))
// wenn Taste 4 eben
gedrueckt wurde:
        sw4_slope = 1;
//    Flankenbit Taste 4
setzen
    // Zwischenspeichern
aktuelle Tastenwerte
    sw1_alt = sw1_neu;
// aktuelle Tastenwerte
speichern
    sw2_alt = sw2_neu;
//    in Variable fuer alte
Werte
    sw3_alt = sw3_neu;
    sw4_alt = sw4_neu;
}
// 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
5 -"); // Ausgabe Festtext:
16 Zeichen
    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("  Program
Menu  "); // Ausgabe
Festtext: 16 Zeichen
    _delay_ms(2000);
// Wartezeit nach
Initialisierung
    showMainDisplay();
}
// Anzeige Hauptmenu
============================
============================
==
void showMainDisplay()
{
    lcd_gotoxy(0,0);
```

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

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

1. Je nach gedrückter Taste wird hier die Variable

```
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("   Main
Level   "); // Ausgabe
Festtext: 16 Zeichen
    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr(" P1  P2  P3
P4 "); // Ausgabe Festtext:
16 Zeichen
}
// Ende der Funktion
/* Teilprogramm 1: Blinkende
LED
============================
==================
Funktion:      Die gelbe LED
(LED 3) in der Schaltung
blinkt mit einer
Periodendauer von 2 Sekunden
(1 s ein, 1 s aus). Auf dem
LCD-
                Display wird
rechts unten der Wert der
LED ("1" oder "0") als
                Zahl
dargestellt. Abbruch mit
Taste S1 nach voller
Periode.
Displayanzeige: +-----------
-----+
                |P1:
Blinking LED|
                |Home
1|
                +-----------
-----+
Tastenfunktion: S1 Flanke:
zurueck zur
Hauptprogrammebene
============================
============================
=================== */
void doBlinkingLed()
{
showBlinkingLedDisplay();
// Initialisierung Display
    SET_BIT(DDRB, DDB2);
// Port B, Pin 0 (LED3) auf
Ausgang schalten
```

modus gesetzt

```
    while(!sw1_slope)
// unendliche Schleife
    {
        SET_BIT(PORTB,PB2);
// Port B, Pin 0 auf LOW:
LED einschalten
        lcd_gotoxy(1,15);
        lcd_putc(ASC_ONE);
// Anzeige LED-Wert "1" auf
Display
        _delay_ms(WAIT_LED);
        CLR_BIT(PORTB, PB2);
// Port B, Pin 0 auf HIGH:
LED ausschalten
        lcd_gotoxy(1,15);
        lcd_putc(ASC_ZERO);
// Anzeige LED-Wert "0" auf
Display
        _delay_ms(WAIT_LED);
    }
// Ende der Warteschleife
    sw1_slope = 0;
// Alle Flankenbits loeschen
    sw2_slope = 0;
    sw3_slope = 0;
    sw4_slope = 0;
}
// zurück zur Hauptschleife
// Anzeige zu Teilprogramm 1
void
showBlinkingLedDisplay()
// Start der Funktion
{
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("P1: Blinking
LED"); // Ausgabe Festtext:
16 Zeichen
    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("Home
"); // Ausgabe Festtext: 16
Zeichen
}
// Ende der Funktion
/* Teilprogramm 2:
Soundgenerierung
============================
===============
```

```
Funktion:      Auf dem
kleinen Lautsprecher
(Buzzer) in der Schaltung
               wird ein
sirenenartiger Sound
ausgegeben. Zwischen den
auf-
               und
absteigenden Tönen bleibt
die Frequenz kurz stabil.
               Die Frequenz
wird mit dem Timer 0 (im
CTC-Mode) erzeugt und
               direkt über
den Output-Compare-Pin im
Toggle-Mode ausgegeben.
               Die
jeweilige Periodendauer wird
dreistellig in Timerticks
               auf der
Anzeige rechts unten
dargestellt.
Displayanzeige: +----------
-----+
               |P2: Create
Sound|
               |Home
123|
               +----------
-----+
Tastenfunktion: S1 Flanke:
zurueck zur
Hauptprogrammebene nach
Ablauf des
               gesamten
Sound-Zyklus
============================
============================
=================== */
void doSound()
{
    unsigned char temp = 0;
// lokale Variable
    showSoundDisplay();
// Anzeige zum Programm
        // Ports
initialisieren
    DDRB |= (1<<DDB2);
// Port B, Pin 0 (zur LED)
auf Ausgang
    DDRD |= (1<<DDD5);
```

```
// Port D, Pin 5 (zum
Buzzer) auf Ausgang
    initTimer0();
// Timer 0 fuer
Soundgenerierung
    while(!sw1_slope)
// Solange keine Flanke auf
SW1: Warteschleife
    {
        for (OCR0A=MAX_PER;
OCR0A>=MIN_PER; OCR0A--) //
Frequenz erhoehen
        {
            temp = OCR0A;
// Anzeige des aktuellen
Periodenzaehlers
lcd_gotoxy(1,13);
lcd_putc(temp/100 +
ASC_ZERO);  // Hunderter als
ASCII ausgeben
            temp = temp%100;
// Rest = Zehner, Einer
            lcd_putc(temp/10
+ ASC_ZERO);   // Zehner als
ASCII ausgeben
            lcd_putc(temp%10
+ ASC_ZERO);   // Einer als
ASCII ausgeben
            _delay_ms(100);
// in Schritten von 100 ms
            if(sw1_slope)
// Schleifenabbruch, wenn
Taster S1 gedrückt wird
            {
                TCCR0A = 0;
// Timer 0 stoppen: Sound
ausschalten
                sw1_slope =
0;              // alle
Flankenbits loeschen
                sw2_slope =
0;
                sw3_slope =
0;
                sw4_slope =
0;
                return;
            }
        }
        _delay_ms(WAIT_SND);
// Wartezeit hohe Frequenz
```

```
        for (OCR0A=MIN_PER;
OCR0A<MAX_PER; OCR0A++)  //
Frequenz absenken
        {
            temp = OCR0A;
// Anzeige des aktuellen
Periodenzaehlers
lcd_gotoxy(1,13);
lcd_putc(temp/100 +
ASC_ZERO);  // Hunderter als
ASCII ausgeben
            temp = temp%100;
// Rest = Zehner, Einer
            lcd_putc(temp/10
+ ASC_ZERO);   // Zehner als
ASCII ausgeben
            lcd_putc(temp%10
+ ASC_ZERO);   // Einer als
ASCII ausgeben
            _delay_ms(100);
// in Schritten von 100 ms
            if(sw1_slope)
// Schleifenabbruch, wenn
Taster S1 gedrückt wird
            {
                TCCR0A = 0;
// Timer 0 stoppen: Sound
ausschalten
                sw1_slope =
0;              // alle
Flankenbits loeschen
                sw2_slope =
0;
                sw3_slope =
0;
                sw4_slope =
0;
                return;
            }
        }
        _delay_ms(WAIT_SND);
// Wartezeit niedrige
Frequenz
    }
// Ende der unendlichen
Schleife
    // Nach Erkennen der
Flanke von SW1
    TCCR0A = 0;
// Timer 0 stoppen: Sound
ausschalten
```

```
    sw1_slope = 0;
// alle Flankenbits loeschen
    sw2_slope = 0;
    sw3_slope = 0;
    sw4_slope = 0;
}
// zurück zur Hauptschleife
// Intialisierung des Timers
0 fuer Sounderzeugung
void initTimer0()
{
    TCCR0A = (1<<WGM01)
|(1<<COM0B0);   // CTC Mode
waehlen
    TCCR0B = (1<<CS01 |
1<<CS00);       // Timer-
Vorteiler /64
    OCR0A = MAX_PER;
// Start mit tiefstem Ton
}
// Anzeige zu Teilprogramm 2
void showSoundDisplay()
// Start der Funktion
{
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("P2: Create
Sound"); // Ausgabe
Festtext: 16 Zeichen
    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("Home
"); // Ausgabe Festtext: 16
Zeichen
}
// Ende der Funktion
/* Teilprogramm 3: Logische
Funktionen
============================
============
Funktion:      Auf dem
Display werden Ergebnisse
von
                logischen
Verknuepfungen (UND, ODER,
NOT, XOR) dargestellt.
                Die
logischen Eingangssignale
werden von den Tasten S3 und
```

```
S4
                eingelesen.
Displayanzeige: Start
Nach 2 s:
                +-----------
-----+      +--------------
-+
                |P3: Logic
Funct.|      |S3&S4=0
S3+S4=0|
                |Home
|      |/S3=0  S3xorS4=0|
                +-----------
-----+      +--------------
-+
Tastenfunktion: S1 Flanke:
zurueck zur
Hauptprogrammebene
                S3:
Logischer Eingang (ohne
Entprellung)
                S4:
Logischer Eingang (ohne
Entprellung)
============================
============================
==================== */
void doLogicFunctions()
{
    unsigned char temp = 0;
// lokale Variable
    showLogicDisplay();
// Anzeige initialisieren
    while(!sw1_slope)
// Solange keine Flanke auf
SW1: Warteschleife
    {
        if
((!sw3_alt)&&(!sw4_alt))
temp=ASC_ONE; // Ergebnis
der UND-Verknuepfung
        else temp=ASC_ZERO;
        lcd_gotoxy(0,6);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben
        if
((!sw3_alt)||(!sw4_alt))
temp=ASC_ONE; // Ergebnis
der ODER-Verknuepfung
        else temp=ASC_ZERO;
```

```
        lcd_gotoxy(0,15);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben
        if (sw3_alt)
temp=ASC_ONE;  // Ergebnis
der Negation
        else temp=ASC_ZERO;
        lcd_gotoxy(1,4);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben
        if
((!sw3_alt)^(!sw4_alt))
temp=ASC_ONE;    // Ergebnis
der XOR-Verknuepfung
        else temp=ASC_ZERO;
        lcd_gotoxy(1,15);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben
        _delay_ms(100);
// Wartezeit 100 ms vor
neuer Auswertung
    }
    sw1_slope = 0;
// alle Flankenbits loeschen
    sw2_slope = 0;
    sw3_slope = 0;
    sw4_slope = 0;
}
// zurück zur Hauptschleife
// Anzeige zu Teilprogramm 3
void showLogicDisplay()
{
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("P3: Logic
Funct."); // Ausgabe
Festtext: 16 Zeichen
    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("Home
"); // Ausgabe Festtext: 16
Zeichen
    _delay_ms(2000);
// Wartezeit 2 s
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
```

```
Zeichen
    lcd_putstr("S3&S4=0
S3+S4=0"); // Ausgabe
Festtext: 16 Zeichen
    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("/S3=0
S3xorS4=0"); // Ausgabe
Festtext: 16 Zeichen
}
/* Teilprogramm 4: Up-Down-
Counter
============================
================
Funktion:      Es wird ein
4-stelliger Dezimal-Zaehler
(0000..9999) mit
                Anzeige und
Ueber-/ Unterlauf
realisiert. Das Aufwaerts-
und
Abwaertszaehlen wird mit
zwei Tasten (S3: +) (S4: -)
gesteuert.
                Es werden
die Flanken beim Druecken
der Tasten ausgewertet.
                Die Taste S2
dient zum Ruecksetzen des
Zaehlers auf 0000.
Displayanzeige: +-----------
-----+
                |P4: Counter
0000|
                |Home RES  +
- |
                +-----------
-----+
Tastenfunktion: S1 Flanke:
zurueck zur
Hauptprogrammebene
                S2 Reset
Counter (ohne Entprellung)
                S3 Flanke:
Counter++ (mit Entprellung)
                S4 Flanke:
Counter-- (mit Entprellung)
============================
============================
==================== */
```

```
void doCounterProg()
{
    int temp;
// lokale Variable
    showCounterDisplay();
// Anzeige initialisieren
        // Auswertung der
Tasten
    while(!sw1_slope)
// Solange keine Flanke auf
SW1: Warteschleife
    {
        if (sw2_alt==0)
// solange Taste 1
gedrueckt:
            counter = 0000;
//    Counter auf 0000
setzen
        if (sw3_slope)
// wenn Taste 2 eben
gedrueckt wurde:
        {
            sw3_slope = 0;
//    Flankenbit loeschen
            counter++;
//    Counter hochzaehlen,
Überlauf bei 9999
            if
(counter==10000)
                counter =
0000; //       auf 0000
setzen
        }
        if (sw4_slope)
// wenn Taste 3 eben
gedrueckt wurde:
        {
            sw4_slope = 0;
//    Flankenbit loeschen
            counter--;
//    Counter
herunterzaehlen, Unterlauf
bei 0
            if
(counter<0000)
                counter =
9999; //       auf 9999
setzen
        }
        _delay_ms(100);
// Auswertung alle 100 ms
```

```
            // Anzeige der
Werte
        lcd_gotoxy(0,12);
        temp = counter;
lcd_putc(temp/1000+ASC_ZERO)
; // Tausender ausgeben
        temp = temp%1000;
// Rest = Hunderter, Zehner,
Einer
lcd_putc(temp/100+ASC_ZERO);
// Hunderter ausgeben
        temp = temp%100;
// Rest = Zehner. Einer
lcd_putc(temp/10+ASC_ZERO);
// Zehner ausgeben
lcd_putc(temp%10+ASC_ZERO);
// Einer ausgeben
    }
    sw1_slope = 0;
// alle Flankenbits loeschen
    sw2_slope = 0;
    sw3_slope = 0;
    sw4_slope = 0;
}
// zurück zur Hauptschleife
// Anzeige zu Teilprogramm 4
void showCounterDisplay()
{
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("P4: Counter
0000"); // Ausgabe Festtext:
16 Zeichen
    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("Home RES  +
- "); // Ausgabe Festtext:
16 Zeichen
}
// Auswahl im Hauptmenu
ermitteln
============================
============================
void getChoiceInMainMenu()
{
    if (sw1_slope)
// Wenn Flanke auf Taste 1
    {
        sw1_slope=0;
```

```
//    Flankenbit loeschen
        modus=1;
//    neuer Modus 1
    }
    if (sw2_slope)
// Wenn Flanke auf Taste 2
    {
        sw2_slope=0;
//    Flankenbit loeschen
        modus=2;
//    neuer Modus 2
    }
    if (sw3_slope)
// Wenn Flanke auf Taste 3
    {
        sw3_slope=0;
//    Flankenbit loeschen
        modus=3;
//    neuer Modus 3
    }
    if (sw4_slope)
// Wenn Flanke auf Taste 4
    {
        sw4_slope=0;
//    Flankenbit loeschen
        modus=4;
//    neuer Modus 4
    }
}
```

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

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 in zwei Schritten optimiert: Erster Schritt war 5_program_menu_opt.c . Aus funktionaler Sicht sind alle Programme gleich. Kompilieren Sie diesen Code und überprüfen Sie die Speicherauslastung.
    1. Wie funktioniert die optimierte Funktionen `void getPressedButton()`?

    2. Für was wird der Array `DisplayText` verwendet?
3. Für zweite Version wurde gänzlich auf Delays im Millisekundenbereich verzichtet: 5_program_menu_opt_v2.rar
    1. Analysieren Sie die `main.c`. Was macht die Unterfunktion `doCycle10ms`? Was der Array `runSubFuncPointer`?
    2. Alle Unterfunktionen wurden in separate Dateien ausgelagert. `BlinkingLed.c` ist hierbei wieder die einfachste Funktion. Analysieren Sie, wie diese funktioniert.