5 Menüführung

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

5. Menüführung

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. 5_program_menu.simu
    2. 5_program_menu.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_program_menu.simu`
    2. Laden Sie `5_program_menu.hex` als firmware auf den 328 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:
```

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

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

```
Program_Menu.c

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

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 Variablen `vorteiler` und `hundertstel` sind für die Umrechnung des Interrupts auf längere Perioden wichtig.
6. In `counter` wird die eigentliche, auf- bzw. absteigende Zahl gespeichert.

   Aufgabe

   Welchen Wertebereich hat `int`?

7. `timertick`, `takt10ms`, `takt100ms` sind Bit-Botschaften (auch Flag genannt). Diese Boolewerte geben bescheid, ob die Interrupt Service Routine aufgerufen wurde (timertick), oder ob 10ms oder 100ms abgelaufen ist.

8. Wird die Taste S1 gedrückt, so wird `sw1_neu` gesetzt. `sw1_alt` entspricht dem vorherigen Wert. Gleiches gibt es für die anderen Taster.

```
(Hauptebene):
                +-----------
-----+        +------------
---+
                |-
Experiment 5 -|         |
Main Level    |
                |  Program
Menu  |         | P1  P2  P3
P4 |
                +-----------
-----+        +------------
---+

                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
============================
============================
======

// Festlegung der
```

9. Die Makros wurden bereits erklärt

10. Die Funktionsprototypen zeigen wieder die kommenden Unterprogramme an

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

1. Zunächst werden zwei Initialisierungsroutinen aufgerufen (siehe weiter unten)
2. Dann werden die “Timer/Counter Control Register” des Timers 0 `TCCR0A` und `TCCR0B` gesetzt. Im verwendeten “Normal Mode” zählt der ein Timer (=Zählerbaustein) im Microprozessor hoch. Die entspricht etwa dem `a=a+1` im C Code, nur, dass der Microprozessor dafür keinen Code ausführen muss. Das Register `TCCR0B` gibt mit dem Prescaler an, dass das Hochzählen um ein nur alle 8 Prozessortakte erfolgen soll. Der verwendete Timer 0 ist ein 8-Bit Timer. Er zählt also von 0 bis 255, läuft dann über und beginnt wieder bei 0.
3. `TIMSK0` ist die “Timer Interrupt MaSK” des Timers 0. Damit kann angegeben werden, ob und wenn ja, welcher Interrupt ausgelöst werden soll. Timer kann damit so konfiguriert werden, dass er keinen Interrupt auslöst, oder einen Interrupt bei einem bestimmten Wert auslöst, oder einen Interrupt beim Überlauf auslöst.
   Mit dem Bit `TOIE0` wird der Interrupt bei Überlauf aktiviert (vgl. ATmegaX8 Datenblatt (Kap. 15.9.6) oder ATmega328 Datasheet (Kap. 14.9.6)).
4. erst mit dem Befehl `sei()` wird die Bearbeitung von Interrupts aktiv
5. in der Endlosschleife sind zwei if-Befehle zu finden, welche über Flags prüfen, ob $10ms$ oder $100ms$ abgelaufen sind. Wenn ja, wird als erstes das Flag zurückgesetzt und dann die gewünschte Unterfunktion aufgerufen.
6. Die Abfrage der Tasten soll entprellt geschehen. Das ist durch das Abtasten / Einlesen des Signals alle $10ms$ möglich.
7. Für die Textanzeige ist eine keine ruckelfreie Darstellung notwendig. Damit kann für die Darstellung der Wert von $30 Hz$ unterschritten werden, über dem ein Bild als

```
Quarzfrequenz
#ifndef F_CPU
// optional definieren
#define F_CPU 12288000UL
// MiniMEXLE mit 12,288 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-
Anzeige


// 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
60    // 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
```

flüssig animiert war genommen wird. Eine Anzeige alle $100ms$ ist also ausreichend

Interrupt Routine

1. Mit dem Befehl `ISR()` wird eine Interrupt Service Routine angelegt. Das verwendete `TIMER0_OVF_vect` spezifiziert den gewünschten Interrupt, hier den OVerFlow Interrupt für TIMER0.
2. Der Überlauf-Interrupt durch den Timer0 wird erst bei Überlauf des 8-Bit Wert ausgeführt. Das entspricht alle $t_{ISR}=\frac{256 \cdot Prescaler}{f_{Quarz}} = \frac{256 \cdot 8}{12'288'000 Hz} = 0,16\bar{6}ms$.
3. Als erstes wird beim Ausführen die boole-Variable `tick` gesetzt. Diese gibt an: ISR wurde aufgerufen.
4. Die Variable `vorteiler` ist auch ein Zähler, welcher mit jedem Aufruf von ISR heruntergezählt wird. Mit `vorteiler = VORTEILER_WERT` als Ausgangswert (Zeile 65) zählt `vorteiler` von 60 herunter. Da ISR alle $0,16\bar{6}ms$ aufgerufen wird, wird `vorteiler` alle $60\cdot0,16\bar{6}ms=10ms$ gleich 0.
5. Wenn `vorteiler` 0 erreicht wird die Variable wieder auf den Startwert zurückgesetzt und der das Flag für das Erreichen der $10ms$ gesetzt. Um auch $10\cdot10ms$ abzählen zu können, muss nach $10ms$ `hundertstel` auch herunter gezählt werden.
6. Erreicht `hundertstel` den Wert 0, so wird auch diese Variable auf 0 und ebenso das Flag für das erreichen von $100ms$ zurückgesetzt
7. Mit dieser Methode erzeugt der Interrupt nur 3 Flags, die anderweitig ausgelesen werden können, z.B. in `main()`. Die ISR bleibt also sehr schlank. Wäre in der ISR() viel Code auszuführen, so würde der Prozessor zwischen zwei Interrupts kaum noch Zeit haben, um sich dem unterbrochenen Programm zu widmen.

Taster initialisieren

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

Zaehlfunktion

```
#define OFF_TIME  100
// "Aus-Zeit" in Inkrementen
zu 100 ms

#define MIN_PER       59
// minimale Periodendauer in
"Timerticks"
#define MAX_PER       255
// maximale Periodendauer in
"Timerticks"
#define WAIT_TIME 2000
// Wartezeit zwischen
Flanken in ms

#define NULL  0x30
// ASCII-Zeichen '0'
#define EINS  0x31
// ASCII-Zeichen '1'


// Variable
unsigned char vorteiler   =
VORTEILER_WERT;    //
Zaehlvariable Vorteiler
unsigned char hundertstel
= 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
```

1. Zunächst werden die einzelnen Tastenstellungen mittels verUNDen einer Bitmaske für den jeweiligen Taster aus PINB in die Variable ausgelesen.

2. Für die Reaktion auf einen Tastendruck gibt es nun zwei Varianten:
   a. immer wenn erkannt wird, dass die Taste gedrückt ist (der Schalter geschlossen ist), wird reagiert. \\b. nur beim Wechsel von 'Taster nicht gedrückt' zu 'Taster gedrückt' (Flanke von 0 auf 1) wird reagiert.
   Das Zurücksetzen auf 0 soll immer ausgelöst werden; entsprechend wird hier Variante a. gewählt. Der Zähler soll nur zu dem Zeitpunkt Herauf-/Herunterzählen, wenn der Schalter gerade geschlossen wurde; entsprechend wird hier Variante b. gewählt.

3. Im Falle das Heraufzählens, ist ein Überlauf bei 10000 vorhanden. Im Falle des Herunterzählens, gibt es einen Unterlauf für werte kleiner als 0 - dann wird auf 9999 gesprungen.

4. Zum Ende dieser Funktion müssen die Schalterstellungen in die Variablen `sw1_alt` bis `sw3_alt` gespeichert werden. Damit kann beim nächsten Aufruf die Flankendetektion stattfinden.

Anzeige Zaehler

1. Zur Ausgabe des Zählerwerts wird eine Hilfsvariable angelegt und auf eine Position unten rechts auf dem Display gesprungen
2. Um den Wert 3456 auszugeben, wird dieser Schritt für Schritt im Display aufgebaut. Für die Tausenderstelle wird zunächst der Wert $3456/1000$ ohne Nachkommastellen ausgerechnet. Für die Anzeige muss dieser Wert in einen ASCII-Wert umgewandelt werden. Dazu muss `0x30` addiert werden.
3. Für die Hunderterstelle von 3456 muss nun vom Tausender-Rest 456 wieder die höchste Stelle ausgegeben werden. Der Tausender-

```
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_slop = 0;
// Flankenspeicher fuer
Taste 1
bool sw2_slop = 0;
// Flankenspeicher fuer
Taste 2
bool sw3_slop = 0;
// Flankenspeicher fuer
Taste 3
bool sw4_slop = 0;
// Flankenspeicher fuer
Taste 4


// Funktionsprototypen
void initTaster(void);
// Taster initialisieren
void initTimer0(void);
// Timer 0 initialisieren
(Soundgenerierung)
void initDisplay(void);
// Initialisierung des
Displays
void readButton(void);
// Tasten einlesen
void mainMenu(void);
// Hauptmenu bearbeiten
void mainDisplay(void);
// Anzeige des Hauptmenus

void blinkingLED(void);
// Teilprogramm 1: Blinkende
LED
void
blinkingLED_Display(void);
// Anzeige zu Teilprogramm 1

void sound(void);
// Teilprogramm 2:
Soundgenerierung
void sound_Display(void);
// Anzeige zu Teilprogramm 2
```

Rest kann über die Modulo-Funktion (im Code mittels %) ermittelt werden. Für Zehner- und Einerwert kann aus dem Hunderter-Rest direkt Division durch 10 ohne Rest und gerade dieser Rest verwendet werden

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

1. Hier wird wieder die Startanzeige mit dem Namen des Programms generiert

```
void logicFunctions(void);
// Teilprogramm 3: Logische
Funktionen
void logic_Display(void);
// Anzeige zu Teilprogramm 3

void counterProg(void);
// Teilprogramm 4: Zaehler
void counter_Display(void);
// Anzeige zu Teilprogramm 4

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

int main()
{
    initTaster();
// Taster initialisieren
    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
    {
        switch(modus)
// Programmverteiler:
Variable "modus"
        {
        case 0:
// Modus 0: Hauptmenu
            mainMenu();
            break;
```

```
        case 1:
// Modus 1: Blinkende LED
            blinkingLED();
// Programm laeuft bis zum
Abbruch
            modus = 0;
// danach auf Hauptmenu
zurueckschalten
            mainDisplay();
            break;

        case 2:
// Modus 2: Soundgenerierung
            sound();
// Programm laeuft bis zum
Abbruch
            modus = 0;
// danach auf Hauptmenu
zurueckschalten
            mainDisplay();
            break;

        case 3:
// Modus 3: Logische
Funktionen
logicFunctions();        //
Programm laeuft bis zum
Abbruch
            modus = 0;
// danach auf Hauptmenu
zurueckschalten
            mainDisplay();
            break;

        case 4:
// Modus 4: Up-Down-Counter
            counterProg();
// Programm laeuft bis zum
Abbruch
            modus = 0;
// danach auf Hauptmenu
zurueckschalten
            mainDisplay();
            break;
        }
    }
    return 0;
}


// 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

        }
    }
}

// Taster initialisieren
============================
============================
void initTaster(void)
```

```
{
    DDRB = DDRB & 0xE1;
// Port B auf Eingabe
schalten
    PORTB |= 0x1E;
// Pullup-Rs eingeschaltet
    _delay_us(10);
// Wartezeit Umstellung
Hardware-Signal
}

// Funktion Tasten einlesen
============================
=======================
void readButton(void)
{
    // Einlesen der 4
Tastensignale
    sw1_neu = (PINB & (1 <<
PB1));
    sw2_neu = (PINB & (1 <<
PB2));
    sw3_neu = (PINB & (1 <<
PB3));
    sw4_neu = (PINB & (1 <<
PB4));

    // 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
((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

    mainDisplay();
}


// Anzeige Hauptmenu
============================
============================
==
void mainDisplay()
{
    lcd_gotoxy(0,0);
// 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) auf dem
MiniMEXLE-Board 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 blinkingLED()
{

    blinkingLED_Display();
// Initialisierung Display
    SET_BIT(DDRB, DDB0);
// Port B, Pin 2 (LED3) auf
Ausgang schalten

    while(!sw1_slope)
// unendliche Schleife
    {
        SET_BIT(PORTB,PB0);
// Port B, Pin 2 auf LOW:
LED einschalten
        lcd_gotoxy(1,15);
        lcd_putc(EINS);
// Anzeige LED-Wert "1" auf
Display

        _delay_ms(1000);

        CLR_BIT(PORTB, PB0);
// Port B, Pin 2 auf HIGH:
LED ausschalten
        lcd_gotoxy(1,15);
        lcd_putc(NULL);
// Anzeige LED-Wert "0" auf
Display

        _delay_ms(1000);

    }
```

```
// 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 blinkingLED_Display()
// Start der Funktion
{
    lcd_init();
// Initialisierungsroutine
aus der lcd_lib

    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 des
MiniMEXLE-Boards (Buzzer)
                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 sound()
{
    unsigned char temp = 0;
// lokale Variable

    sound_Display();
// Anzeige zum Programm

        // Ports
initialisieren

    DDRB |= (1<<DDB0);
// 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 + NULL);
// Hunderter als ASCII
ausgeben
            temp = temp%100;
// Rest = Zehner, Einer
            lcd_putc(temp/10
+ NULL);    // Zehner als
ASCII ausgeben
            lcd_putc(temp%10
+ NULL);    // 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_TIME);    //
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 + NULL);
// Hunderter als ASCII
ausgeben
            temp = temp%100;
// Rest = Zehner, Einer
            lcd_putc(temp/10
+ NULL);    // Zehner als
ASCII ausgeben
            lcd_putc(temp%10
+ NULL);    // 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_TIME);    //
```

```
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 sound_Display()
// 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 des MiniMEXLE Boards 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 logicFunctions()
```

```
{
    unsigned char temp = 0;
// lokale Variable

    logic_Display();
// Anzeige initialisieren

    while(!sw1_slope)
// Solange keine Flanke auf
SW1: Warteschleife
    {
        if
((!sw3_alt)&&(!sw4_alt))
temp=EINS; // Ergebnis der
UND-Verknuepfung
        else temp=NULL;
        lcd_gotoxy(0,6);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben

        if
((!sw3_alt)||(!sw4_alt))
temp=EINS; // Ergebnis der
ODER-Verknuepfung
        else temp=NULL;
        lcd_gotoxy(0,15);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben

        if (sw3_alt)
temp=EINS;     // Ergebnis
der Negation
        else temp=NULL;
        lcd_gotoxy(1,4);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben

        if
((!sw3_alt)^(!sw4_alt))
temp=EINS;     // Ergebnis
der XOR-Verknuepfung
        else temp=NULL;
        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 logic_Display()
{
    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 counterProg()
{
    int temp;
// lokale Variable

    counter_Display();
// 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+NULL); //
Tausender ausgeben

        temp = temp%1000;
// Rest = Hunderter, Zehner,
Einer
lcd_putc(temp/100+NULL); //
Hunderter ausgeben

        temp = temp%100;
// Rest = Zehner. Einer
lcd_putc(temp/10+NULL);
// Zehner ausgeben
lcd_putc(temp%10+NULL);
// 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 counter_Display()
{
    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
}

// Hauptmenu
```

```
============================
============================
==========
void mainMenu()
{
    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 nacheinander 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. Erweiterung der des Zählers:
    1. Bauen Sie den Zähler so um, dass er jede Sekunde um 1 nach oben zählt.
    2. Ändern Sie die Funktionsweise der Tasten S2 und S3 so, dass diese die Zählrichtung angeben.
2. Variation der Eingabe
    1. Fügen Sie einen weiteren Schalter S4 hinzu.
    2. Mit diesem Schalter soll nun die Stelle (Einer, Zehner, Hunderter, Tausender) ausgewählt werden, die geändert werden soll. Die Funktion soll der in folgender hex-Datei entsprechen: 4_up-down-counter_mit_stellenvorgabe.hex
    3. Tipp 1 Ändern Sie das herauf-/herunterzählen in counterCounting so, dass eine Variable addiert bzw. subtrahiert wird. Überprüfen Sie am besten bereits diese Änderung ohne weitere Funktionalitäten.
    4. Tipp 2 Wie muss die neue Variable bei Tastendruck auf S4 geändert werden? Wann muss die neue Variable wieder zurückgesetzt werden?