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 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

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

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

```
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.

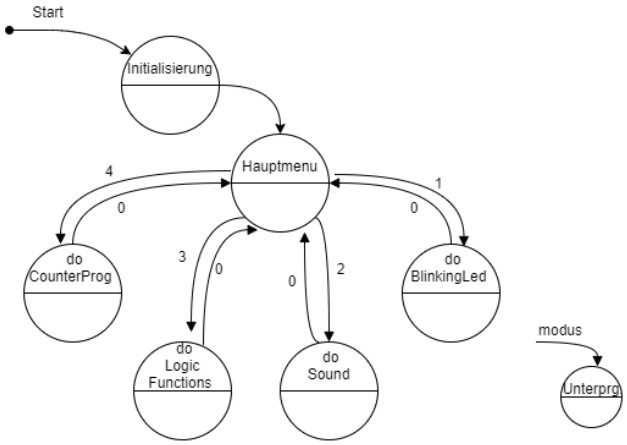
6. Wird die Taste S1 gedrückt, so wird `sw1_neu` gesetzt. `sw1_alt` entspricht dem vorherigen Wert. Gleiches gibt es für die anderen Taster.

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.

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

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:



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 $T_{ISR}= 0,16\bar{6}ms$.
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.

Taster initialisieren

1. Das Einstellen des Data Direction Registers

und der Pullups wurde bereits in vorherigen Programmen erklärt.

```
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
=========================
```

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.

```
/* 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.

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.

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.

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 `modus` gesetzt

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.