4 Up-Down Counter

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

4 Up-Down Counter

Interrupts - was tun bei Unterbrechungen?

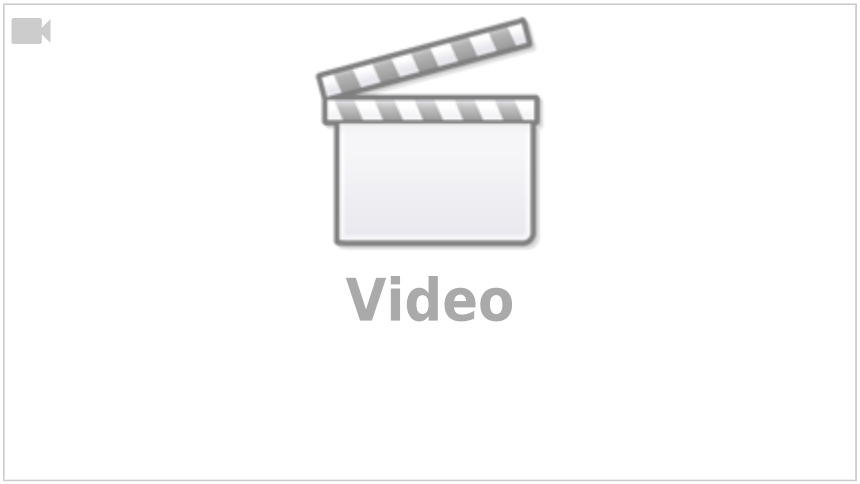
Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie eine Interrupt genutzt wird

Video

Im Video sind nur die ersten 30 Minuten relevant. Danach werden verschiedene Sleep-Modes betrachtet - diese sind für uns nicht relevant.



Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. 4._up_down_counter.sim1
    2. 4._up_down_counter.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 `4_up_down_counter.simu`
    2. Laden Sie `4_up-down-counter.hex` als firmware auf den Atemga 88 Chip
    3. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    4. Als nächstes ist ein Displaybild zu sehen, in dem die Wirkung der verschiedenen Schalter in der zweiten Zeile zu sehen ist:
        1. Schalter 1 setzt den Zähler auf `0000` zurück.
        2. Schalter 2 zählt den Zähler um 1 herauf.
        3. Schalter 3 zählt den Zähler um 1 herab.
2. Prüfen Sie wann genau der Zähler auf- bzw. abzählt? Geschieht dies beim Schließen oder öffnen des Schalters?
3. Das Programm zu diesem Hexfile soll nun erstellt werden

III. Eingabe in Microchip Studio

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

Experiment 4:   Up-Down-
Counter
=============
===============

Dateiname:      Up-Down-
Counter_de.c

Autoren:        Peter
Blinzinger
                Marc
Neumeister
                Prof. G.
Gruhler (Hochschule
Heilbronn)
                D.
Chilachava    (Georgische
Technische Universitaet)

Version:        1.2 vom
01.05.2020

Hardware:       MEXLE2020
Ver. 1.0 oder höher
                AVR-USB-
PROGI Ver. 2.0

Software:
Entwicklungsumgebung: AVR
Studio 7.0
                C-
Compiler:AVR/GENU C Compiler
5.4.0

Funktion:       Es wird ein
4-stelliger Dezimal-Zaehler
(0000..9999) mit
                Anzeige und
Ueber-/ Unterlauf
realisiert. Das Aufwaerts-
und
Abwaertszaehlen wird mit
zwei Tasten (S2: +) (S3: -)
```

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

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. Bei den Header-Dateien wird zusätzlich die`interrupt.h` inkludiert. Damit können “Interrupt Service Routinen” - also

```
gesteuert.
                Es werden
die Flanken beim Druecken
der Tasten ausgewertet.
                Die Taste S1
dient zum Ruecksetzen des
Zaehlers auf 0000.

Displayanzeige: Start (fuer
2s):        Betrieb:
                +-----------
-----+      +--------------
-+
                |-
Experiment 4 -|
|Up/Down-Counter |
                |Up/Down-
Counter |      |RES  +   -
0000|
                +-----------
-----+      +--------------
-+

Tastenfunktion: S1: Reset
Counter   (ohne Entprellung)
                S2: (+)
Aufwaerts   (mit
Entprellung)
                S3: (-)
Abwaerts    (mit
Entprellung)

Jumperstellung: keine
Auswirkung

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

Unterprogramme für Unterbrechungen - definiert werden.

3. Als Konstanten werden `VORTEILER_WERT`, `HUNDERTSTEL_WERT` und `ZEHNTEL_WERT` definiert. Diese sind notwendig, um von der Periode des Interrupts auf die Hunderstelsekunde und Zentelsekunde zu kommen (siehe `ISR (TIMER0_OVF_vect)`)
4. Auch die Variablen `vorteiler` und `hundertstel` sind für die Umrechnung des Interrupts auf längere Perioden wichtig.
5. In `counter` wird die eigentliche, auf- bzw. absteigende Zahl gespeichert.

Aufgabe

Welchen Wertebereich hat `int`?

6. `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.

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

8. Die Makros wurden bereits erklärt

9. 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

```
// 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-
Anzeige

// Konstanten
#define ASC_ZERO
0x30// ASCII-Zeichen '0'
#define VORTEILER_WERT
90  // Faktor Vorteiler = 90
(Timerticks)
#define TAKT10MS_WERT
10  // Faktor Taks10ms  = 10
(1/100 s)

// Variable
unsigned char vorteiler
= VORTEILER_WERT;//
Zaehlvariable Vorteiler
unsigned char
takt10msZaehler   =
TAKT10MS_WERT; //
Zaehlvariable im 10ms Raster

int counter = 0;
// Variable fuer Zaehler

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

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 ATmega88 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 $10~\rm ms$ oder $100 ~\rm ms$ 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 $10 ~\rm ms$ möglich.
7. Für die Textanzeige ist eine keine ruckelfreie Darstellung notwendig. Damit kann für die Darstellung der Wert von $30 ~\rm Hz$ unterschritten werden, über dem ein Bild als flüssig animiert wahrgenommen wird. Eine Anzeige alle $100 ~\rm ms$ 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 einer Periode von $T_{\rm ISR}=\frac{256 \cdot \rm Prescaler}{f_{\rm

```
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 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

// 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)
#define GET_BIT(BYTE, BIT)
((BYTE) &   (1 << (BIT))) //
Bit Zustand in Byte einlesen

// Funktionsprototypen
void initTaster(void);
// Taster initialisieren
void initDisplay(void);
// Initialisierung des
Displays
void counterCounting(void);
// Zaehlfunktion
void counterDisplay(void);
// Anzeigefunktion

// Hauptprogramm
============================
============================
======
int main()
{
```

Quarz}} = \frac{256 \cdot 8}{18'432'000 ~\rm Hz} = 0,1\bar{1}~\rm ms$.

3. Als erstes wird beim Ausführen die boole-Variable `Timertick` 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 90 herunter. Da die ISR alle $0,1\bar{1}~\rm ms$ aufgerufen wird, wird `vorteiler` alle $90\cdot0,1\bar{1}~\rm ms=10~\rm ms$ gleich 0.
5. Wenn `vorteiler` 0 erreicht wird die Variable wieder auf den Startwert zurückgesetzt und der das Flag für das Erreichen der $10~\rm ms$ gesetzt. Um auch $10\cdot10~\rm ms$ abzählen zu können, muss nach $10~\rm ms$ `takt10msZaehler` auch herunter gezählt werden.
6. Erreicht `takt10msZaehler` den Wert 0, so wird auch diese Variable auf 0 und ebenso das Flag für das erreichen von $100~\rm ms$ 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.

`Zaehlfunktion ==============`

1. Zunächst werden die einzelnen Tastenstellungen mittels verUNDen einer Bitmaske für den jeweiligen Taster aus `PINC` 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'

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

    TCCR0A = 0;
// Timer 0 auf "Normal Mode"
schalten
    TCCR0B |= (1<<CS01);
// mit Prescaler /8
betreiben
    TIMSK0 |= (1<<TOIE0);
// Overflow-Interrupt
aktivieren

    sei();
// generell Interrupts
einschalten

    while(1)
// unendliche Warteschleife
mit Aufruf der
// Funktionen abhaengig von
Taktbotschaften
    {
        if (takt10ms)
// alle 10ms:
        {
            takt10ms = 0;
//      Botschaft "10ms"
loeschen
counterCounting();  //
Tasten abfragen,
Zaehlfunktion

        }
        if (takt100ms)
// alle 100ms:
        {
            takt100ms = 0;
//      Botschaft "100ms"
loeschen
counterDisplay();   //
Zaehlerstand auf Anzeige
ausgeben
        }
    }
    return 0;
}

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

(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-Rest kann über die Modulo-Funktion (im Code mittels %) ermittelt werden. Für Zehner- und

```
============================
==
ISR (TIMER0_OVF_vect)
/* In der Interrupt-Routine
sind die Softwareteiler für
die Taktbotschaften
   (10ms, 100ms) realisiert.
Die Interrupts stammen von
Timer 0 (Interrupt 1)

   Verwendete Variable:
vorteiler
hunderstel

   Ausgangsvariable:
takt10ms
takt100ms
*/
{
    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
        --takt10msZaehler;
//    Hunderstelzähler
dekrementieren

        if
(takt10msZaehler==0)     //
wenn 0 erreicht: 100ms
abgelaufen
        {
            takt10msZaehler
= TAKT10MS_WERT; // Teiler
auf Startwert
            takt100ms = 1;
//    Botschaft 100ms senden
        }
    }
}
```

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

```
// Zaehlfunktion
============================
============================
======
void counterCounting(void)
{
    //   Bitposition im
Register:
    //            __76543210
    DDRC = DDRC &
0b11111000;       //
Zunaechst Port B auf Eingabe
schalten
    PORTC =
0b00000111;       // Pullup-
Rs eingeschaltet
    _delay_us(1);
// Umschalten der Hardware-
Signale abwarten

    // Einlesen der 3
Tastensignale
    sw1_neu = GET_BIT(PINC,
PC0) ;// aktuelle Werte der
Tasten 1-3 lesen
    sw2_neu = GET_BIT(PINC,
PC1) ;
    sw3_neu = GET_BIT(PINC,
PC2) ;

    // Auswertung der 3
Tasten

    if (sw1_neu==0)
// solange Taste 1
gedrueckt:
        counter = 0000;
//    Counter auf 0000
setzen

    if
((sw2_neu==0)&(sw2_alt==1))
// wenn Taste 2 eben
gedrueckt wurde:
    {
        counter++;
//    Counter hochzaehlen,
Ueberlauf >9999
        if (counter==10000)
            counter = 0;
    }
```

```
    if
((sw3_neu==0)&(sw3_alt==1))
// wenn Taste 3 eben
gedrueckt wurde:
    {
        counter--;
//    Counter herabzaehlen,
Unterlauf <0000
        if (counter < 0)
            counter = 9999;
//    auf 9999 setzen
    }

    // Zwischenspeichern
aktuelle Tastenwerte

    sw1_alt = sw1_neu;
// aktuelle Tastenwerte
umspeichern
    sw2_alt = sw2_neu;
//    in Variable für alte
Werte
    sw3_alt = sw3_neu;

    DDRC = DDRC |
0b00000111;       // Am Ende
Port B wieder auf Ausgabe
schalten
}

// Anzeige Zaehler
============================
============================
====
void counterDisplay(void)
{
    int temp;
// lokale temporaere
Variable
    lcd_gotoxy(1,12);
// Cursor auf
Ausgabeposition im Display
    temp = counter;
lcd_putc(temp/1000+ASC_ZERO)
;   // Ausgabe Tausender als
ASCII-Wert

    temp = temp%1000;
// Divisionrest = Hunderter
+ Zehner + Einer
lcd_putc(temp/100+ASC_ZERO);
```

```
// Ausgabe Hunderter als
ASCII-Wert

    temp = temp%100;
// Divisionsrest = Zehner +
Einer
lcd_putc(temp/10+ASC_ZERO);
// Ausgabe Zehner als ASCII-
Wert
lcd_putc(temp%10+ASC_ZERO);
// Ausgabe Einer als ASCII-
Wert
}

// 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
4 -");   // Ausgabe
Festtext: 16 Zeichen

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("Up/Down-
Counter ");   // Ausgabe
Festtext: 16 Zeichen

    _delay_ms(2000);
// Wartezeit nach
Initialisierung

    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("Up/Down-
Counter "); // Ausgabe
Festtext: 16 Zeichen

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
```

```
Zeichen
    lcd_putstr("RES  +   -
0000"); // Ausgabe Festtext:
16 Zeichen
}
// Ende der Funktion
```

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?

Weiterführendes

- Diese Falstad Schaltung skizziert die Struktur des Timer/Counters