4 Up-Down Counter

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

4. Up-Down Counter

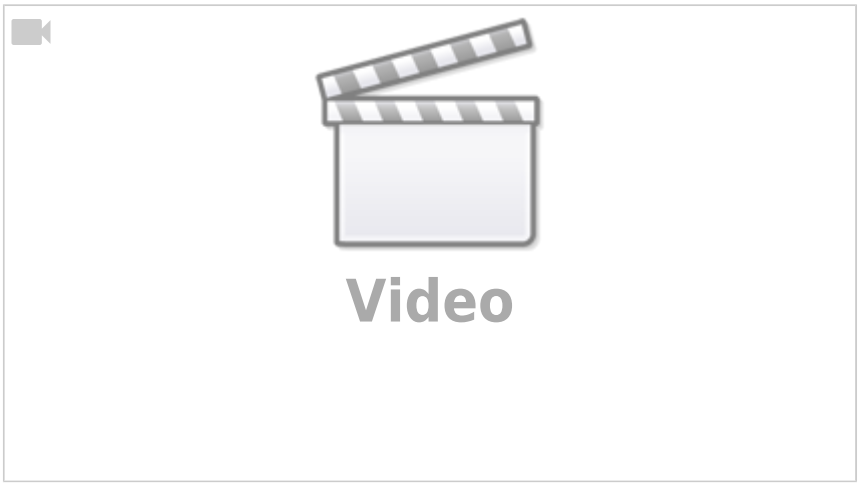
Interrupts - was tun bei Unterbrechungen?

Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie eine Interrupt genutzt wird

Video



Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. 4_up_down_counter.simu
    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 `3_logic_functions.simu`
    2. Laden Sie `3_logic_functions.hex` als firmware auf den 328 Chip
2. Betrachtung der neuen Komponenten: In der Simulation sind nun neben dem Microcontroller, der LED und dem Display Hd44780 zwei Schalter als neue Komponenten zu sehen, welche mit S1 und S2 bezeichnet sind. Diese werden in diesen Beispiel zur Eingabe genutzt.
3. Betrachtung des Programmablaufs
    1. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    2. Als nächstes ist ein Displaybild zu sehen, in dem verschiedene logische Formeln mit Ergebnissen abgebildet sind:
        1. AND-Verknüpfung: $S1\&S2$,
        2. OR-Verknüpfung: $S1+S2$,
        3. NOT-Verknüpfung: $/S1$,
        4. XOR-Verknüpfung: $S1 xor S2$
    3. Werden die Tasten S1 und S2 gedrückt, so werden die Ergebnisse aktualisiert.
4. Das Programm zu diesem Hexfile soll nun erstellt werden

III. Eingabe in Atmel 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.
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 muss. Das Register TCCR0B gibt mit dem Prescaler an, dass das Hochzählen um ein nur

```
======

// Festlegung der
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

// Konstanten
#define VORTEILER_WERT
60    // Faktor Vorteiler =
60 (Timerticks)
#define HUNDERTSTEL_WERT
10    // Faktor Hundertstel
= 10 (1/100 s)
#define ZEHNTEL_WERT
10    // Faktor Zehntel = 10
(1/10 s)

// Variable
unsigned char vorteiler
= VORTEILER_WERT;    //
Zaehlvariable Vorteiler
unsigned char hundertstel
= HUNDERTSTEL_WERT; //
Zaehlvariable Hundertstel

int counter = 0000;
// Variable fuer Zaehler

bool timertick;
```

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 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

```
// Bit-Botschaft alle
0,111ms (bei 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 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(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

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

// Hauptprogramm
============================
============================
======
int main()
{
    initTaster();
// Taster initialisieren
```

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.

`Funktionen ========================`

1. In `initDisplay` wird wieder zunächst das Display initialisiert und die Startanzeige mit dem Namen des Programms angezeigt. Nach 2 Sekunden werden dann die booleschen Funktionen auf dem Display dargestellt. Dort sind die Ergebnisse für nicht gedrückte Schalter vorgegeben.

2. Funktion `initTaster`
    1. Durch die Änderung des Datenrichtungs-Register (DDR) wird die Richtung der Anschlüsse vorgegeben. Diese beiden Anschlüsse sind in der Simulide Umgebung an die Schalter S1 und S2 angeschlossen. Durch die UND-Verknüpfung mit der Maske `0b11111001` werden die Anschlüsse B0 und B3..B7 nicht geändert, sondern nur die Anschlüsse B1 und B2 auf Eingang gesetzt.
    2. Mit der Zuweisung von PORTB wurde

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

bisher bei Ausgängen der Ausgabewert vorgegeben. Bei Eingängen wird über diese Zuweisung ein Pullup-Widerstand dazugeschalten. Damit ergibt sich aus dem äußeren Schalter und dem internen Widerstand ein Spannungsteiler. Bei leitfähigem Schalter gibt der Spannungsteiler $0V$ (=logisch 0) zum Microcontroller aus, bei offenem Schalter $5V$ (=logisch 1).

3. Funktion `readButtons` liest aus dem Register `PINB` das dem Schalter entsprechende Bit aus. Als Eselsbrücke: P**I**N steht für Input, P**O**RT für Output.

```
============================
==
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 0,166ms senden
    --vorteiler;
// Vorteiler dekrementieren
    if (vorteiler==0)
// wenn 0 erreicht: 10ms
abgelaufen
    {
        vorteiler =
VORTEILER_WERT;    //
Vorteiler auf Startwert
        takt10ms = 1;
//    Botschaft 10ms senden
        --hundertstel;
//    Hunderstelzähler
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
}

// Zaehlfunktion
============================
============================
======
void counterCounting(void)
{
    // Einlesen der 3
Tastensignale
    sw1_neu = (PINB & (1 <<
PB1));    // aktuelle Werte
der Tasten 1-3 lesen
    sw2_neu = (PINB & (1 <<
PB2));
    sw3_neu = (PINB & (1 <<
PB3));

    // 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 = 0000;
    }
    if
```

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

// 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+0x30);
// Ausgabe Tausender als
ASCII-Wert

    temp = temp%1000;
// Divisionrest = Hunderter
+ Zehner + Einer
    lcd_putc(temp/100+0x30);
// Ausgabe Hunderter als
ASCII-Wert

    temp = temp%100;
// Divisionsrest = Zehner +
Einer
```

```
    lcd_putc(temp/10+0x30);
// Ausgabe Zehner als ASCII-
Wert
    lcd_putc(temp%10+0x30);
// 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 Schalteranzahl:
    1. Fügen Sie zwei weitere Tasten mit Verbindung zu Masse und jeweils den Eingängen B3 und B4 ein - analog zu den vorhandenen Schaltern.
    2. Klicken Sie rechts auf den Schalter und wählen Sie im Kontextmenu `Properties`. links sollten nun die Eigenschaften des Schalters sichtbar sein. geben Sie als `id` S3 bzw. S4 ein und wählen Sie bei `Show id true`. Der Name des Schalters sollte nun sichtbar sein.
    3. Ändern Sie den Code so, dass diese Schalter eingelesen werden können. Dazu sollten die Funktionen `initTaster`, `readButton` und `main` angepasst werden.
    4. Als ersten Test sollten die booleschen Funktionen statt den Schaltern S1 und S2 die Schalter S3 und S4 als Eingangswerte haben. Testen Sie diese Änderung.
    5. Im nächsten Programm sollen alle Schalter S1...S4 die Eingangswerte darstellen. Es sollen nun alle alle Eingagen per Schalter S1...S4in die verschiedenen booleschen Funktionen eingehen. Also bei z.B. aus $S1\&S2$ wird $S1\&S2\&S3\&S4$. Überlegen Sie sich wie bei XOR vorzugehen ist.