7 Uhr und Zeitraster

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

7 Uhr und Zeitraster

Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie man aus den auf Interupt-basierten Zeitrastern langsamere Raster umsetzt.

Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. 7._uhr_und_zeitraster.sim1
    2. 7._uhr_und_zeitraster.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 `7._uhr_und_zeitraster.sim1`
    2. Laden Sie `7._uhr_und_zeitraster.hex` als firmware auf den 88 Chip
    3. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    4. Als nächstes ist im Display eine Uhr mit dem Format HH:MM:SS Menu zu sehen
    5. Die Tasten 2 und 3 ermöglichen das Einstellen der Stunde und Minute. Werden die Minuten hochgezählt, so werden die Sekunden auf 0 gesetzt.

III. Eingabe in Atmel Studio

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

Experiment 7:   7_mexleclock
mit Stunden-, Minuten- und
Sekunden-Anzeige
=============
============================
============================
=

Dateiname:
7_MexleClock.c
```

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

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

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

Version:        1.3 vom
22.10.2022

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:       Digitaluhr
mit Anzeige von Stunden,
Minuten und Sekunden. Eine
                einfache
Stellfunktion ist mit den
Tasten S2 und S3 realisiert.

Displayanzeige: Start (fuer
2s):        Betrieb:
                +-----------
-----+      +--------------
-+
                |-
Experiment 7 -|      |===
00:00:00 ===|
                | Digital
Clock  |      |   Std Min
|
                +-----------
-----+      +--------------
-+

Tastenfunktion: S2: Std
(zaehlt Stunden bei Flanke
aufwaerts. Überlauf bei 24)
                S3: Min
(zaehlt Minuten bei Flanke
aufwaerts. Überlauf bei 60)
```

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.
   Zusätzlich wird der Ausdruck ASC_NULL durch den Hexadezimalwert einer '0' in ASCII, also 0x30 und ASC_COLON durch den ASCI-Wert eines Doppelpunkts, also 0x3A, ersetzt

5. Bei den Variablen entsprechen einige denen der letzten Programme.
6. Für die Uhr werden Stunden, Minuten, Sekunden und Zehntelsekunden mit Anfangswerten deklariert.

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

```
(setzt Sekunden beim
Druecken zurueck auf 00)

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)

Module:         1)
Taktgenerator
                2) Zaehler
fuer Uhr (Takt: 1 s)
                3)
Anzeigetreiber   (Takt: 100
ms)
                4)
Stellfunktion    (Takt: 10
ms)

    Modul 1:    Das Modul
"Taktgenerator" erzeugt den
Takt von 1s fuer die Uhr.
                Zusaetzliche
Takte: 10 ms fuer
Stellfunktion
100 ms fuer Anzeige.

                Verwendung
von Hardware-Timer 0 und T0
Overflow-Interrupt.
                Frequenzen:
Quarzfrequenz
12,288 MHz.
Timer-Vorteiler     /  8
=>   1,536 MHz
Hardware-Timer      /256
=>   6 kHz / 166 µs
Software-Vorteiler  / 60
=> 100 Hz  /  10 ms
Hundertstel-Zaehler / 10
=>  10 Hz  / 100 ms
Zehntel-Zaehler     / 10
=>   1 Hz  /   1 s

    Modul 2:    Das Modul
```

Hauptprogramm ==========

1. Das Hauptprogramm ähnelt sehr stark dem Up/Down Counter. Entsprechend werden die Zeilen 143-157 hier nicht weiter erklärt.

2. In der Endlosschleife sind auf der ersten Ebene wieder nur If-Abfragen zu den Flags `takt10ms` und `takt100ms` zu finden.

    1. Alle $10~\rm ms$ (bzw. wenn das entsprechende Flag gesetzt wird) wird das Flag zurückgesetzt und das Unterprogramm `uhrStellen()` aufgerufen

    2. Alle $100~\rm ms$ (bzw. wenn das entsprechende Flag gesetzt wird) wird das Flag zurückgesetzt und das Unterprogramm `uhrAnzeigen()` aufgerufen

    3. Alle $1~\rm s$ (bzw. wenn das entsprechende Flag gesetzt wird) wird das Flag zurückgesetzt und das Unterprogramm `uhrZaehlen()` aufgerufen

Interrupt Routine ==========

1. Auch die Interrupt Routine ist dem Programm Up/Down Counter entlehnt und wird hier nicht weiter erklärt.

```
"Zaehler fuer Uhr" wird
durch den Takt 1s
aufgerufen.
                Sekunden,
Minuten und Stunden werden
als Binaerzahlen gezaehlt
                Sekunden und
Minuten zaehlen 00..59, die
Stunden 00..23.
                Ein "Tick"
auf dem Lautsprecher wird
jede Sekunde ausgegeben.

    Modul 3:    Das Modul
"Anzeigetreiber" startet
alle 100 ms. Es gibt die
Hintergrundinformationen und
die aktuelle Uhrzeit aus.
                Darstellung
auf der Anzeige (mittig in
Zeile 1):  [23:59:59]
    Modul 4:    Das Modul
"Stellfunktion" ist an den
10 ms-Takt gekoppelt.
                Es dient
1. zum Einlesen und
Entprellen der Stelltasten
- Auswertung der fallenden
Flanke 1=> 0
2. zum Ausfuehren der
Stellfunktion:
- S2 zaehlt die Stunden
aufwaerts
- S3 zaehlt die Minuten
aufwaerts
- solange Taste S3
gedrueckt: Sekunden = 00
(einfache Synchronisierung
der Uhr!)
                Beim Stellen
kein Uebertrag von den
Minuten auf die Stunden.
    Die Kopplung der Module
wird ueber global definierte
Variable realisiert:

    1-Bit-Variable:
takt10ms:   Taktgenerator =>
Stellfunktion
takt100ms:  Taktgenerator =>
Anzeigetreiber
```

`Stellfunktion ==========`

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

2. Neu hier ist, dass die Bedienung der Schalter die Variablen für Stunden, Minuten um eins hochsetzen, bzw. bei Überlauf wider zurück auf 0 setzen. Zusätzlich wird bei eine Änderung des Minuten-Werts der Sekunden-Wert auf 0 gesetzt.

`Anzeigefunktion Uhr ========================`

1. Hierüber wird die Uhrzeit in der ersten Zeile im Format hh:mm:ss ausgegeben.
2. Ähnlich zum Counter werden die zweistelligen

```
takt1s:     Taktgenerator =>
Zaehler fuer Uhr

    8-Bit-Variable:
sekunden    Stellfunktion =>
Zaehler => Anzeige
minuten
stunden

============================
============================
====================*/

// Deklarationen
============================
============================
======

// 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 <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(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
```

Werte mit Division durch 10 und dessen Rest in zwei einzelne Ziffern gewandelt

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.

Zaehlfunktion Uhr

1. Die Zähl-Funktion `uhrZaehlen()` ist ganz ähnlich aufgebaut zur Interrupt-Service-Routine
2. Zunächst wird ein Schaltwechsel am Ausgang mit dem Lautsprecher ausgegeben, um einen Knackton zu erzeugen
3. Dann werden die Sekunden hochgezählt
4. ist das Maximum erreicht, so wird der Sekunden-Wert zurückgesetzt und der Minuten-Wert um eins hochgezählt.
5. ebenso wird beim Maximum des Minuten-Serts

```
(toggle)

// Konstanten
#define PRESCALER_VAL
90      // Faktor Vorteiler
= 90
#define CYCLE10MS_MAX
10      // Faktor
Hundertstel = 10
#define CYCLE100MS_MAX
10      // Faktor Zehntel =
10

#define SPEAK_PORT
PORTD   // Port-Adresse fuer
Lautsprecher
#define SPEAK_BIT
5       // Port-Bit fuer
Lautsprecher

#define ASC_NULL
0x30        // Das Zeichen
'0' in ASCII
#define ASC_COLON
0x3A        // Das Zeichen
':' in ASCII
#define INPUT_PIN_MASK
0b00001111

// Variable
unsigned char
softwarePrescaler =
PRESCALER_VAL;    //
Zaehlvariable Vorteiler
unsigned char cycle10msCount
= CYCLE10MS_MAX;    //
Zaehlvariable Hundertstel
unsigned char
cycle100msCount   =
CYCLE100MS_MAX;   //
Zaehlvariable Zehntel
unsigned char seconds
= 56;               //
Variable Sekunden
unsigned char minutes
= 34;               //
Variable Minuten
unsigned char hours
= 12;               //
Variable Stunden
```

dieser zurückgesetzt und der Stundenwert hochgezählt.
6. beim Maximum des Stunden-Werts wird dieser wieder auf Null gesetzt

```
bool timertick;
// Bit-Botschaft alle
0,166ms (bei Timer-
Interrupt)
bool cycle10msActive;
// Bit-Botschaft alle 10ms
bool cycle100msActive;
// Bit-Botschaft alle 100ms
bool cycle1sActive;
// Bit-Botschaft alle 1s

bool button2_new = 1;
// Bitspeicher fuer Taste 2
bool button3_new = 1;
// Bitspeicher fuer Taste 3
bool button2_old = 1;
// alter Wert von Taste 2
bool button3_old = 1;
// alter Wert von Taste 3

uint8_t buttonState    =
0b00001111;        //
Bitspeicher fuer Tasten

// Funktionsprototypen
void initDisplay(void);
// Init Anzeige
void setTime(void);
// Stellfunktion
void showTime(void);
// Anzeigefunktion
void refreshTime(void);
// Uhrfunktion

// Hauptprogramm
============================
============================
======
int main()
{
    // Initialisierung
    initDisplay();
// Initialisierung LCD-
Anzeige

    TCCR0A = 0;
// Timer 0 auf "Normal Mode"
schalten
    SET_BIT(TCCR0B, CS01);
// mit Prescaler /8
betreiben
```

```
    SET_BIT(TIMSK0, TOIE0);
// Overflow-Interrupt
aktivieren

    SET_BIT(DDRD,
SPEAK_BIT);   // Speaker-Bit
auf Ausgabe

    sei();
// generell Interrupts
einschalten

    // Hauptprogrammschleife

    while(1)
// unendliche Warteschleife
mit Aufruf der
// Funktionen abhaengig von
Taktbotschaften
    {
        if (cycle10msActive)
// alle 10ms:
        {
            cycle10msActive
= 0;       //      Botschaft
"10ms" loeschen
            setTime();
//      Tasten abfragen, Uhr
stellen
        }
        if
(cycle100msActive)
// alle 100ms:
        {
            cycle100msActive
= 0;      //      Botschaft
"100ms" loeschen
            showTime();
//      Uhrzeit auf Anzeige
ausgeben
        }
        if (cycle1sActive)
// alle Sekunden:
        {
            cycle1sActive =
0;         //      Botschaft
"1s" loeschen
            refreshTime();
//      Uhr weiterzaehlen
        }
    }
```

```
    return 0;
}

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

ISR (TIMER0_OVF_vect)
/*  In der Interrupt-Routine
sind die Softwareteiler
realisiert, die die Takt-
    botschaften (10ms,
100ms, 1s) fuer die gesamte
Uhr erzeugen. Die Interrupts
    werden von Timer 0
ausgeloest (Interrupt Nr. 1)

*/
{
    timertick = 1;
// Botschaft 0,166ms senden
    --softwarePrescaler;
// Vorteiler dekrementieren
    if
(softwarePrescaler==0)
// wenn 0 erreicht: 10ms
abgelaufen
    {
        softwarePrescaler =
PRESCALER_VAL; //
Vorteiler auf Startwert
        cycle10msActive = 1;
//    Botschaft 10ms senden
        --cycle10msCount;
//    Hunderstelzaehler
dekrementieren

        if
(cycle10msCount==0)
// wenn 0 erreicht: 100ms
abgelaufen
        {
            cycle10msCount =
CYCLE10MS_MAX; // Teiler auf
Startwert
            cycle100msActive
= 1;          //
Botschaft 100ms senden
            --
cycle100msCount;
```

```
//    Zehntelzaehler
dekrementieren

            if
(cycle100msCount==0)
// wenn 0 erreicht: 1s
abgelaufen
            {
cycle100msCount =
CYCLE100MS_MAX; //    Teiler
auf Startwert
cycle1sActive = 1;
//    Botschaft 1s senden
            }
        }
    }
}
// Stellfunktion
============================
============================
======
void setTime(void)
/*  Die Stellfunktion der
Uhr wird alle 10ms
aufgerufen. Dadurch wir eine
    Entprellung der
Tastensignale realisiert.
Das Stellen wir bei einer
    fallenden Flanke des
jeweiligen Tastensignals
durchgefuehrt. Darum
    muss fuer einen weiteren
Stellschritt die Taste
erneut betaetigt werden.

    Eine Flanke wird durch
(alter Wert == 1) UND
(aktueller Wert == 0)
erkannt.

    Mit der Taste S2 werden
die Stunden aufwaerts
gestellt.
    Mit der Taste S3 werden
die Minuten aufwaerts
gestellt (kein Uebertrag)
    Solange Taste S3
gedrueckt ist werden die
Sekunden auf 00 gehalten

    Veraenderte Variable:
```

```
stunden
minuten
sekunden

    Speicher fuer Bits:
sw2Alt
sw3Alt
*/
{
    DDRC = DDRC
&~INPUT_PIN_MASK;    // Port
B auf Eingabe schalten
    PORTC |=
INPUT_PIN_MASK;    //
Pullup-Rs eingeschaltet
    _delay_us(1);
// Wartezeit Umstellung
Hardware-Signal
    buttonState    = (PINC &
INPUT_PIN_MASK) ;
// Hole den Schalterstatus
von B1..B4, 0b1 ist hier
offener SChalter
    DDRC |= INPUT_PIN_MASK;
// Port B auf Ausgabe
schalten

    // Einlesen der
Tastensignale
    button2_new =
(buttonState & (1 << PC1));
    button3_new =
(buttonState & (1 << PC2));

    if
((button2_new==0)&(button2_o
ld==1))  // wenn Taste 2
eben gedrueckt wurde:
    {
        hours++;
//    Stunden hochzaehlen,
Ueberlauf bei 23
        if (hours==24)
            hours = 00;
    }
    if
((button3_new==0)&(button3_o
ld==1))  // wenn Taste 3
eben gedrueckt wurde:
    {
        minutes++;
```

```
//    Minuten hochzaehlen,
Ueberlauf bei 59
        if (minutes==60)
            minutes = 00;
    }
    if (button3_new==0)
// solange Taste 3
gedrueckt:
        seconds = 00;
//    Sekunden auf 00 setzen

    button2_old =
button2_new;              //
aktuelle Tastenwerte
speichern
    button3_old =
button3_new;              //
in Variable fuer alte Werte
}

// Anzeigefunktion Uhr
============================
============================
void showTime(void)
/*  Die Umrechnung der
binaeren Zaehlwerte auf BCD
ist folgendermaßen geloest:
    Zehner: einfache
Integer-Teilung (/10)
    Einer:  Modulo-
Ermittlung (%10), d.h. Rest
bei der Teilung durch 10
*/
{
    lcd_gotoxy(0,4);
// Cursor auf Start der
Zeitausgabe setzen
    lcd_putc(ASC_NULL +
hours/10);    // Stunden
Zehner als ASCII ausgeben
    lcd_putc(ASC_NULL +
hours%10);    // Stunden
Einer als ASCII ausgeben
    lcd_putc(ASC_COLON);
// Doppelpunkt ausgeben
    lcd_putc(ASC_NULL +
minutes/10);    // Minuten
als ASCII ausgeben
    lcd_putc(ASC_NULL +
minutes%10);    //
    lcd_putc(ASC_COLON);
```

```
// Doppelpunkt ausgeben
    lcd_putc(ASC_NULL +
seconds/10);   // Sekunden
als ASCII ausgeben
    lcd_putc(ASC_NULL +
seconds%10);   //
}

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

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr(" Digital
Clock  "); // Ausgabe
Festtext: 16 Zeichen

    _delay_ms(2000);
// Wartezeit nach
Initialisierung

    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("=== 00:00:00
==="); // Ausgabe Festtext:
16 Zeichen

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("   Std Min
"); // Ausgabe Festtext: 16
Zeichen
}
// Ende der Funktion
```

```
// Zaehlfunktion Uhr
============================
============================
==
void refreshTime (void)
// wird jede Sekunde
gestartet
/*  Die Uhr wird im
Sekundentakt gezaehlt. Bei
jedem Aufruf wird auch ein
    "Tick" auf dem
Lautsprecher ausgegeben.
Ueberlaeufe der Sekunden
zaehlen
    die Minuten, die
Ueberlaeufe der Minuten die
Stunden hoch.

    Veraenderte Variable:
sekunden
minuten
stunden
*/
{
    TGL_BIT (SPEAK_PORT,
SPEAK_BIT); // "Tick" auf
Lautsprecher ausgeben
// durch Invertierung des
Portbits

    seconds++;
// Sekunden hochzaehlen
    if (seconds==60)
// bei Überlauf:
    {
        seconds = 0;
//  Sekunden auf 00 setzen
        minutes++;
//  Minuten hochzaehlen
        if (minutes==60)
//  bei Ueberlauf:
        {
            minutes = 0;
//  Minuten auf 00 setzen
            hours++;
//  Stunden hochzaehlen
            if (hours==24)
//  bei Ueberlauf:
                hours = 0;
//  Stunden auf 00 setzen
        }
```

```
    }
}
```

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:

Aufgabe

1. Bauen Sie einen “ewigen Kalender” ein
    1. Es sollen nicht nur Stunde, Minute, Sekunde dargestellt werden und veränderbar sein, sondern auch Tag, Monat und Jahr im Format dd:mm:yy
    2. Achten Sie auf die Schaltjahrermittlung
2. Erweitern Sie den Kalender auf vierstellige Jahresangabe.
3. Überlegen Sie sich, wie man Ihre Programme testen kann.
4. BONUS: Wie kann der Wochentag bestimmt werden?