6 Würfel und Zufall

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

6 Würfel und Zufall

Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie man Zufallswerte in einem deterministischen System einfach erstellen kann.

Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. 6._wuerfel_und_zufall.sim1
    2. 6._wuerfel_und_zufall.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 `6. Wuerfel und Zufall.sim1`
    2. Laden Sie `6. Wuerfel und Zufall.hex` als firmware auf den 88 Chip
    3. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    4. Als nächstes ist im Display ein Menu zu sehen. Durch Tastendruck kann ein Würfel gestartet (Druch auf Taste 1) oder gestoppt (4) werden. Die Augenzahl des Würfels ist in der Mitte der unteren Zeile dargestellt.
2. Das Programm zu diesem Hexfile soll nun erstellt werden

III. Eingabe in Microchip Studio

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

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

Experiment 6:   MEXLEcast Elektronischer Wuerfel auf dem MEXLE
=============================================================
=================

Dateiname:
MEXLEcast_de.c

Autoren:        Peter
```

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

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

Version:         1.2 vom
30.04.2020

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:        Es wird ein
elektronischer Wuerfel mit
Anzeige auf dem Display
                 realisiert.
Mit zwei Tasten S1 = Start
und S4 = Stop wird der
                 Wuerfel
gesteuert. Der Wuerfel wird
mit 10ms-Takt gezaehlt. Die
                 Anzeige
erfolgt als Ziffer im 100ms-
Takt.

Displayanzeige: Start (fuer
2s):         Betrieb:
                 +----------
-----+      +--------------
-+
                 |-
Experiment 6 -|
|Electronic Cast |
                 |Electronic
Cast |       |Start  1   Stop
|
                 +----------
-----+      +--------------
-+

Tastenfunktion: S1: Start
```

```
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. Alle Variablen außer `castBit` entsprechen denen der letzten Programme. `castBit` ist ein “Merker”, bzw. Flipflop welches das Würfeln aktiviert oder deaktiviert
6. Bei den Funktionsprototypen sind einige bekannte Unterprogramme vorhanden. Details werden weiter unten erklärt.

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

1. Das Hauptprogramm ähnelt sehr stark dem Up/Down Counter.
2. Zunächst werden zwei Initialisierungsroutinen aufgerufen (siehe weiter unten)
3. Dann werden wieder die “Timer/Counter

```
(Set-Funktion Flip-Flop)
                S4: Stop
(Reset-Funktion Flip-Flop)

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

// 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
```

Control Register" des Timers 0 TCCR0A und TCCR0B gesetzt. Der 8-Bit Timer und auch hier im "Normal Mode" zum hochzählen genutzt. Auch hier gibt das Register TCCR0B den Prescaler an.

4. Auch hier wird über die "Timer Interrupt MaSK" `TIMSK0` durch das Bit `TOIE0` ("Timer Overflow Interrupt Enable") der Interrupt bei Überlauf aktiviert.
5. Mit dem Befehl `sei()` wird die Bearbeitung von Interrupts aktiv

6. In der Endlosschleife sind auf der ersten Ebene wieder nur If-Abfragen zu den Flags `cycle10msActive` und `cycle100msActive` zu finden.

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

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

```
Interrupt Routine
=========================
```

1. Mit dem Befehl `ISR()` wird eine Interrupt Service Routine für den OVerFlow Interrupt für TIMER0 angelegt.
2. Der Überlauf-Interrupt durch den Timer0 wird erst bei Überlauf des 8-Bit Wert ausgeführt. Auch hier ergibt sich durch den Prescaler und Modus (TCCR0A und TCCR0B) eine Periode von $T_{\rm ISR}= 0,16\bar{6}~\rm ms$.
3. Die Ermittlung von `timertick`, `softwarePrescaler`, `cycle10msActive`, `cycle10msCount` und `cycle100msActive` ist hier wieder gleich dem im Up/Down Counter.

```
#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)

// Konstanten
#define PRESCALER_VAL
90      // Faktor für
softwarePrescaler
(Timerticks 0,111 ms)
#define CYCLE10MS_MAX
10      // Faktor von
cycle1ms zu cycle10ms

#define INPUT_PIN_MASK
0b00001111
#define ASC_NULL
0x30

// Variable
unsigned char
softwarePrescaler =
PRESCALER_VAL;   //
Zaehlvariable Vorteiler
unsigned char cycle10msCount
=  CYCLE10MS_MAX; //
Zaehlvariable 10ms Raster

unsigned char castVar = 1;
// Variable für Wuerfel-
Zaehler

bool cycle10msActive  = 0;
// Bit-Botschaft alle 10ms
bool cycle100msActive = 0;
// Bit-Botschaft alle 100ms

bool button1          = 0;
// Bitspeicher fuer Taste 1
bool button4          = 0;
// Bitspeicher fuer Taste 4
bool castBit = 0;
// Flip-Flop-Bit fuer
Start/Stop

uint8_t buttonState    =
0b00001111;        //
Bitspeicher fuer Tasten
```

4. Eine große Änderung ist, dass bereits im Interrupt alle 10ms die Unterfunktion `readButton()` aufgerufen wird.

`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 die Bedienung der Schalter nur das Flag `castBit` setzen. Falls dieses gesetzt ist, wird die Variable `castVar` hochgezählt. Falls diese 6 überschreitet wird sie auf 1 zurückgesetzt.

`Anzeigefunktion Wuerfel ========================`

1. Hierüber wird die Augenzahl in der zweiten Zeile an Position 7 ausgegeben.

```
// Funktionsprototypen
void timerInt0(void);
// Init Zeitbasis mit Timer
0
void castCounting(void);
// Zaehlfunktion Wuerfel
void castDisplay(void);
// Anzeige Wuerfel
void initDisplay(void);
// Initialisierung Display

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

    // Hauptprogrammschleife

    while(1)
// unendliche Warteschleife
    {
        if (cycle10msActive)
// alle 10ms:
        {
            cycle10msActive
= 0;    //      Botschaft
"10ms" loeschen
            castCounting();
//      Tasten abfragen,
Wuerfel zaehlen
        }
```

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.

```
        if
(cycle100msActive)       //
alle 100ms:
        {
            cycle100msActive
= 0;   // Botschaft "100ms"
loeschen
            castDisplay();
// Wuerfelwert ausgeben
        }
    }
    return 0;
}

// Interrupt-Routine
============================
============================
==
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:
softwarePrescaler
cycle10msCount

   Ausgangsvariable:
cycle10msActive
cycle100msActive
*/
{
    --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
        }
    }
}


// Wuerfelfunktion
============================
============================
====
void castCounting(void)
{
    DDRC = DDRC
&~INPUT_PIN_MASK;    // Port
C 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 C0..C3, 0b1 ist hier
offener Schalter
    DDRC |= INPUT_PIN_MASK;
// Port C auf Ausgabe
schalten

    // Einlesen der
Tastensignale
    button1 = (buttonState &
(1 << PC0));
    button4 = (buttonState &
(1 << PC3));
    // Auswertung der Tasten

    if (button1==0)
// solange Taste 1
gedrueckt:
        castBit = 1;
```

```
//    Flip-Flop "Wuerfeln"
Setzen

    if (button4==0)
// solange Taste 4
gedrueckt:
        castBit = 0;
//    Flip-Flop "Wuerfeln"
Ruecksetzen

    if (castBit)
// Solange Flip-Flop
"Wuerfeln" gesetzt
    {
        castVar++;
//    Wurfelwert hochzaehlen
im Takt 10ms
        if (castVar>6)
//    groesser als 6?
            castVar=1;
//       => auf 1 setzen
    }
}

// Anzeigefunktion Wuerfel
============================
========================
void castDisplay(void)
{
    lcd_gotoxy(1,7);
// Cursor auf
Ausgabeposition
lcd_putc(castVar+ASC_NULL);
// ASCII-Wert des
Wuerfelzaehlers ausgeben
}

// 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
```

```
6 -"); // Ausgabe Festtext:
16 Zeichen

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

    _delay_ms(2000);
// Wartezeit nach
Initialisierung

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

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

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. Darstellung des Augenwerts ändern
    1. Laden Sie folgende Dateien herunter: 6_mexle_cast_extern.sim1, 6a_mexlecast_extern.hex
    2. Simulieren Sie die Schaltung mit der beigefügten hex-Datei
    3. Ändern Sie das bisherige Programm so, dass Sie das gleiche Ergebnis erhalten, bzw. zumindest die Ansteuerung der LEDs.