6 Würfel und Zufall

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

6. Würfel und Zufall 2
Ziele 2
Übung 2
..... 2

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_mexle_cast.simu
    2. 6_mexle_cast.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_mexle_cast.simu`
    2. Laden Sie `6_mexle_cast.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. Durch Tastendruck ein Würfel gestartet (Druch auf Taste 1) oder gestoppt (4) werden kann. 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 Atmel Studio

/*

Experiment 6: MEXLEcast Elektronischer Wuerfel auf MiniMEXLE

Dateiname: MEXLEcast_de.c

Autoren: Peter 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 (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 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

Makros #define SET_BIT(PORT, BIT) [1]) Port-Bit Setzen #define CLR_BIT(PORT, BIT) [2]) *Port-Bit Loeschen #define TGL_BIT(PORT, BIT) [3])* Port-Bit Toggeln

Konstanten #define VORTEILER_WERT 60 Faktor Vorteiler (Timerticks 0,111 ms) #define HUNDERTSTEL_WERT 10 *Faktor Hunderstel (1/100 s)* Variable unsigned char vorteiler = VORTEILER_WERT; *Zaehlvariable Vorteiler unsigned char hundertstel = HUNDERTSTEL_WERT;* Zaehlvariable Hunderstel

unsigned char castVar = 1; *Variable für Wuerfel-Zaehler bool timertick;* Bit-Botschaft alle 0,111ms (Timer-Interrupt) bool takt10ms; *Bit-Botschaft alle 10ms bool takt100ms;* Bit-Botschaft alle 100ms

bool sw1; *Bitspeicher fuer Taste 1 bool sw4;* Bitspeicher fuer Taste 4 bool castBit = 0; *Flip-Flop-Bit fuer Start/Stop* Funktionsprototypen void init_Taster(void); *Taster initialisieren 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
init_Taster();              // Taster initialisieren
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 (takt10ms)            // alle 10ms:
    {
        takt10ms = 0;        //        Botschaft "10ms" loeschen
        castCounting();        //        Tasten abfragen, Wuerfel
zaehlen
```

```
    }
    if (takt100ms)            // alle 100ms:
    {
        takt100ms = 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 2 (Interrupt 1) Verwendete Variable: vorteiler hunderstel Ausgangsvariable: takt10ms takt100ms */ { timertick = 1; Botschaft 0,111ms senden

1. -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 init_Taster(void) { DDRB = DDRB & 0xE1; Port B auf Eingabe schalten

```
PORTB |= 0x1E;                   // Pullup-Rs eingeschaltet
_delay_us(10);                   // Wartezeit Umstellung Hardware-Signal
```

```
}
```

Wuerfelfunktion

void castCounting(void) { Einlesen der Tastensignale

```
sw1 = (PINB & (1 << PB1));
sw4 = (PINB & (1 << PB4));

// Auswertung der Tasten
```

```
if (sw1==0)                 // solange Taste 1 gedrueckt:
    castBit = 1;            //       Flip-Flop "Wuerfeln" Setzen
```

```
if (sw4==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+0x30);     // 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 </sxh> </WRAP>*

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

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

</WRAP> <-- --> IV. Ausführung in Simulide # - Geben Sie die oben dargestellten Codezeilen ein und kompilieren Sie den Code. - 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: - 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). - Der oben gezeigte Code wurde optimiert: 5_program_menu_opt.c . Aus funktionaler Sicht sind die beiden Programme gleich. Kompilieren Sie diesen Code und überprüfen Sie die Speicherauslastung. - Wie funktioniert die optimierte Funktionen `void getChoiceInMainMenu()`? - Für was wird der Array `DisplayText` verwendet? <--

[1)]

```
Deklarationen
==================================
```

PORT) |= (1 « (BIT [2)]

PORT) &= ~(1 « (BIT [3)]

PORT) ^= (1 « (BIT

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.

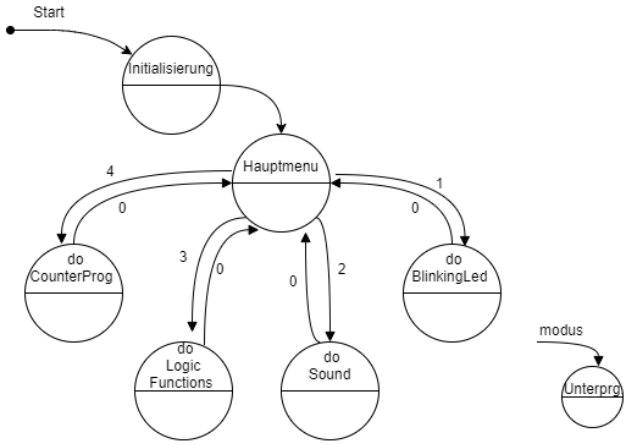
6. Wird die Taste S1 gedrückt, so wird `sw1_neu` gesetzt. `sw1_alt` entspricht dem vorherigen Wert. Gleiches gibt es für die anderen Taster.

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 wir 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