TU Berlin Fakultät IV
Institut für Energie und Automatisiertungstechnik
Fachgebiet Elektronische Mess- und Diagnosetechnik
Praktikum Messdatenverarbeitung

Praktikum Messdatenverarbeitung
Termin 1

Özgü Dogan (326 048)
Timo Lausen (325 411)
Boris Henckell (325 779)

3. Mai 2012

Gruppe: G1 Fr 08-10

Betreuer: Jürgen Funk

Inhaltsverzeichnis

1 Vorbereitungsaufgaben

1.1 Quellcode

```
// Vorbereitungsaufgabe Termin 2
//    zg    Dogan (326048)
// Timo Lausen (325411)
// Boris Henckell (325779)


#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>

ISR(ADC_vect)
{
uint16_t ADUWERT = ADC;
PORTC &= ~(1<<PC5);      //anschalten der Roten LED
if (ADUWERT<=340){
    PORTC |= (1<<PC1);   //Orangene LED ausgeschaltet
    PORTC |= (1<<PC4);   //Gr ne LED ausgeschaltet
    }
else if (ADUWERT<=682){
    PORTC |= (1<<PC1);   //Orangene LED ausgeschaltet
    PORTC &= ~(1<<PC4);  //Gr ne LED anschalten
    }
else {
    PORTC &= ~(1<<PC1);  //Orangene LED anschalten
    PORTC &= ~(1<<PC4);  //Gr ne LED anschalten
    }

}

int
main (void)
{

    // ADU anschalten

    ADCSRA |= (1<<ADEN);    // shiften 1 in ADCSRA um ADEN, damit ADU angeht

    // ADCH und ADCL Register (je 8bit)
    // 10bit rechts auslesen - ganz normal
    // links adjusten
    // ADMUX |= (1<<ADLAR);    // shiften 1 in ADMUX um ADLAR

    ADMUX  |= (1<<REFS1) | (1<<REFS0);       //Voltage reference 2.56V (s. Dat
    ADCSRA |= (1<<ADPS0) | (1<<ADPS2);       //F_adu = (F_clk)/32 (s. Datenbla

    //  Als Eingang soll der Kanal ADC0 im Single-Ended-Modus genutzt werden.
    //  Daf r wird im ADCSRB Register MUX 4:0 zu null gesetzt. Der 5. Bit (MU
    //  (s. Datenblatt S.290, 26.8.2)

    //  F r den free running modus m ssen die ADTS2,1,0 Bits im ADCSRB Regi
```

```
    //  ADCSRB &= ~(1<<ADTS2) & ~(1<<ADTS1) & ~(1<<ADTS0);
(s. Datenblatt S.296, Tabelle 26-6)

    sei();                                  //aktiviert Interrupts allgemein (

    //alternativ Interrupts global aktivieren: //(s. Datenblatt S.14, 7.4.1)
    //SREG |= (1<<I);




    ADCSRA |= (1<<ADIE);                    //aktiviert das Interrupt im ADC (

    while ( 1 ) {

    }
    return 0 ;
}
```

1.2 Abtastrate des ADU

Der ADU arbeitet bei dieser Einstellung mit einer Abtastrate von $\frac{f_{clk}}{32}$. Es lassen sich auch Abtastraten von $\frac{f_{clk}}{2}$, $\frac{f_{clk}}{4}$, $\frac{f_{clk}}{8}$, $\frac{f_{clk}}{16}$, $\frac{f_{clk}}{64}$ und $\frac{f_{clk}}{128}$ einstellen.
Für eine Umsetzung benötigt der ADU-jedoch 13 solcher Takte und daher ist die effective Abtstrate $f_{sample} = \frac{f_{clk}}{13 \cdot 32}$

2 Versuch

Im Praxisteil soll nun der Analog-Digital-Umwandler (ADU) zur Erfassung von Messwerten genutzt werden. Dafür soll es möglich sein die Samplerate und die Dauer der Messung einzustellen. Der Free-Running-Mode aus dem Theorieteil ist hierfür nicht geeignet.
Statt dessen soll der ADU von einem Timer getriggert werden. Dieser kann frei konfiguriert werden und (nahezu) beliebige Abtastraten zur Verfügung stellen. Um die Messdaten zu speichern und auszulesen muss eine Verbindung zu einem PC bestehen. Die nötige Software hierfür ist vorgegeben.

2.1 Vorgehensweise

Das Programm wird in mehrere Funktionen aufgeteilt und einzeln implementiert.

Function	Anwendung
adcInit()	initialisiert den ADU
adcStart()	startet eine neue Messung
adcIsRunning()	gibt an ob gerade eine Messung vorgenommen wird

Zusätzlich sind noch folgende Interruptroutinen zu programmieren.

Interruptvektor	Funktion
ADC	ließt einen Abtastwert aus und speichert ihn

2.2 Implementierung: adcInit()

Die Funktion adcInit() soll den ADU initialisieren. Als Referenzspannung soll die μC interne $2,56V$ Bandgapreferenz genutzt werden. Der ADU wird wie in den Vorbereitungsaufgaben mit einem 32tel des CPU-Taktes betrieben. Gemessen wird an Kanal 0.
Um beliebige Abtastraten einstellen zu können, soll eine AD-Umsetzung diesmal von einem Timer getriggert werden. In dieser Funktion wird auch das AD-Wandlung-komplett-Interrupt aktiviert.

2.3 Implementierung: adcStart()

Diese Funktion soll eine neue Messung starten. Dafür werden (indirekt) die Samplerate und die Anzahl der benötigten Messwerte übergeben. Die Anzahl der Messwerte wird in einer globalen Variable gespeichert und somit auch anderen Funktionen zur Verfuegung gestellt.Außerdem kann optional noch auf die steigende oder fallende Flanke getriggert werden. Hierfür kann auch noch ein Triggerlevel übergeben werden.
Die Samplerate wird nicht direkt übergeben. Der Timer zählt die Impulse des CPU-Taktes. Auf diese Weise lässt sich die Zeit messen. Ist eine gewisse Zeit überschritten, wird abgetastet und der Timer wird resettet. Dies ist der Clear-on-Compare-Modus (CTC) des Timers. Die Samplerate wurde nun vorher in die Anzahl von Impulsen, die der Timer zwischen zwei Triggerimpulsen warten soll, umgerechnet. Die Anzahl an Impulsen ist der RATECODE und berechnet sich wie folgt.

$$RATECODE = \frac{F_{CPU}}{SampleRate} - 1 \tag{1}$$

Der Wert RATECODE wird dann an das Output-Compare-Register A und B übergeben. Dies ist nötig, da der ADU nur vom Output-Compare-Register B getriggert, der Timer im CTC-Mode aber nur vom Output-Compare-Register A resettet werden kann.

2.4 Interruptroutinen

Die Interrupt-Routine ADC wird aufgerufen, sobald der ADU eine Wandlung beendet hat. Dann wird der Messwert ausgelesen, ein Offset von 512 abgezogen und gespeichert. Dadurch ergibt sich dann ein Wertebereich von -512 bis +511. Zusätzlich wird bei jedem Aufruf von ADC die globale Variable mit der Anzahl der benötigten Samples um eins reduziert. Ist die Anzahl der Samples auf 0, so ist die Messung abgeschlossen und es werden keine weiteren Messwerte aufgenommen.

2.5 Triggermodus und Triggerlevel

Manchmal will man die Messung bei einem gewissen Initialwert starten. Einfach nur zu hoffen zufällig genau zum richtigen Zeitpunkt zu starten, ist keine gute Idee. In adcStart() kann festgelegt werden, auf welche Signalverläufe die Messung getriggert werden soll.
Um steigende oder fallende Flanke auf dem richtigen Level zu erkennen, wird in der ADC-Routine zunächst der vorletzte Messwert gespeichert. Soll die Messung auf die steigende

Flanke erfolgen, wird überprüft, ob der aktuelle Messwert größer als der Triggerlevel ist. Ist dies der Fall wird überprüft, ob der vorletzte Messwert kleiner war als der Triggerlevel. Wenn ja, wurde eine steigende Flanke detektiert. Erst ab diesem Zeitpunkt werden die Messwerte des ADU tatsächlich gespeichert. Das Triggern auf die Fallende flanke funktioniert äquivalent zur steigenden Flanke.

2.6 Implementierung: adcIsRunning()

Diese Funktion gibt zurück, ob gerade eine Messung erfolgt oder nicht. Dazu wird einfach die Anzahl der benötigten Samples überprüft. Werden noch Samples benötigt, ist noch eine Messung in Gang.

3 Listing

Listing 1: Quellcode

```
#include "adc.h"
#include "serial.h"
#include "filter.h"

//------------------globale Variablen--------------------

uint8_t Samples = 0; //Anzahl noch zu erstellender Samples

uint8_t triggerSet = 0; //gibt an ob bereits getriggert wurde

trigger_t triggerEvent = NONE; //gibt an ob auf fallende oder steigende Flanke ge

int lastValue = 0; //das letzte gemessene Sample

int triggerValue = 0; //auf diesen Wert soll getriggert werden

//------------------Interrupt-Routinen--------------------

//################################################
//#Wenn dieser Interrupt ausgeloest wird, loescht#
//#er das Output-Compare-Match-Flag A. Der Timer #
//#kann nun erneut starten.                      #
//################################################
ISR(TIMER1_COMPA_vect) {
}

//################################################
//#Wenn dieser Interrupt ausgeloest wird, loescht#
//#er das Output-Compare-Match-Flag B. Der ADU   #
//#kann nun erneut getriggert werden.            #
//################################################
ISR(TIMER1_COMPB_vect) {
}

//################################################
//#Diese Routine liesst den ADU aus, zieht einen #
```

```
//# Offset ab und speichert den Wert.            #
//################################################
ISR(ADC_vect) {

        int result = ADC; //liesst ADU aus
        result = result - 512; //zieht Offset ab

        if(triggerSet==0){
                if(triggerEvent==RISING){
                        if(lastValue<triggerValue){
                                if(result>=triggerValue){triggerSet=1;}
                        }
                } else {
                        if(lastValue>triggerValue){
                                if(result<=triggerValue){triggerSet=1;}
                        }
                }
        }

        if(triggerSet==1){
                if(Samples>0){ //ueberprueft ob weiter Samples benoetigt werden
                        filterWrite2Buf(result); //speichert Sample
                        Samples--; //Anzahl benoetigter Samples decrementieren
                }
        }

        lastValue = result; //speichert letzten Messwert
}

//----------------------Funktionen----------------------

//################################################
//# Diese Funktion initialisiert den ADU.        #
//# Verwendet werden die interne Referenz und der #
//# Autotriggermodus.                            #
//################################################
void adcInit() {

        ADMUX |= (1<<REFS1) | (1<<REFS0); //interne 2,56V Referenz
        ADCSRA |= (1<<ADIE) | (1<<ADPS2) | (1<<ADPS0); //ADU-Enable und FCPU:32
        ADCSRA |= (1<<ADATE); //Autotrigger mode

        ADCSRB |= (1<<ADTS2) | (1<<ADTS0); //Wandlung auf Comparematch Timer 1B
        ADCSRA |= (1<<ADEN); //ADU starten

        ADCSRA |= (1<<ADSC); //Dummywandlung
}

//################################################
//# Hier wird eine Messung gestartet. Die Anzahl  #
//# der Samples und die Samplerate werden hier    #
//# festgelegt.                                  #
//################################################
void adcStart(uint16_t sampleRateCode, uint32_t sampleCount, trigger_t triggerMode
```

```
        triggerEvent = triggerMode; //setzt Triggermodus
        triggerValue = triggerLevel; //setzt Triggerlevel

        if(triggerMode==NONE){ //im Fall von Triggermode = NONE wird der Triggerin
                triggerSet = 1;
        } else {
                triggerSet = 0;
        }

        TCCR1B |= (1<<WGM12); //CTC auf OutputcompareA
        OCR1A = sampleRateCode; //dient dem CTC zum resetten
        OCR1B = sampleRateCode; //loesst den Autotrigger aus

        TCNT1 = 0x0000; //Timer resetten
        TCCR1B |= (1<<CS20); //Timer1 mit CPU-Takt
        Samples = sampleCount;//Sampleanzahl resetten

}

//################################################
//#Diese Funktion gibt an ob gerade eine Messung #
//#durchgefuehrt wird.                           #
//################################################
uint8_t adcIsRunning() {

        if (Samples == 0){return 0;} //Messung laeuft noch
        return 1; //Messung beendet
}
```