Zeitintervall-Vorgabe versus Zählimpuls-Vorgabe für die Datenauswertung mit dem Arduino bei zählenden Strahlungs-Detektoren für Radioaktivität

Bernd Laquai, 2. Oktober 2015

Bekanntlich unterliegt der radioaktive Zerfall einer Statistik die, zumindest wenn man Zeiträume betrachtet, in die nicht mehr als 50 Zerfallsakte fallen, mit einer Poisson-Verteilung beschrieben und berechnet werden kann. So ist die Aussage dieser Statistik, dass die Wahrscheinlichkeit, mit der in einem Messintervall T genau k Zerfälle stattfinden durch:

$P(k, \lambda) = \lambda^k/k!*\exp(-\lambda)$

angegeben werden kann.

Wenn ein zählender Detektor eine zeitunabhängige Empfindlichkeit hat und nur einen festen Teil der Zerfallsakte in einer Probe sieht (begrenzte Effizienz), dann gilt diese Statistik immer noch genauso für die Zählpulse des Detektors. Interessant ist an dieser Verteilung, dass sie nur von einem Parameter λ beschrieben wird, der den Mittelwert angibt. Die Streuung, und das ist eine Besonderheit, ist einfach die Wurzel aus dem Mittelwert, also:

$\mu = \lambda$ und $\sigma = \sqrt{(\lambda)}$

Wichtig dabei ist, dass man diesen statistischen Mittelwert λ der Poisson-Verteilung nicht mit der Zerfallskonstanten λ aus der Kernphysik verwechselt. Das statistische λ bestimmt bei konstanter Aktivität die Zählrate R im Zeitintervall T:

$R=\lambda/T$

Bei einer Probe, die im Mittel die Rate R (üblicherweise angegeben in cpm) erzeugt, kann man sagen, dass im Intervall T im Mittel $\lambda=R*T$ Zählpulse registriert werden. Damit ist dann aber auch Streuung und damit die Messunsicherheit:

$\sigma = \sqrt{(R/T)}$

also proportional zur Wurzel aus dem Quotient von gemessener Rate und Messintervall. Diese Streuung begrenzt also in erster Linie die bei der Vorgabe des Messintervalls T erreichbare Genauigkeit. Die Genauigkeit nimmt also nur mit der Wurzel aus der Dauer des Zählintervalls zu.

Wartet man bei einem zählenden Detektor auf N Pulse und misst die Zeit um die Rate R zu berechnen, dann ist die Situation meist einfacher. Dann ergibt sich eine mittlere Messzeit T so, dass gilt R=N/T. Dann aber ergibt sich die Streuung zu:

$\sigma = R/\sqrt{(N)}$

Prinzipiell hat die Vorgabe der Zählpulse den Vorteil, dass sich die Messzeit automatisch an die Rate anpasst und die Genauigkeit einfach abgeschätzt werden kann. Beispielsweise ist bei 100 Zählpulsen die Streuung einfach 10% von der gemessenen Rate. Wenn sich die Rate beispielsweise von einem niedrigen Wert, der mit einem langen Messintervall gemessen wird, plötzlich auf einen hohen Wert

ändert, dann bekommt man das recht schnell mit, denn dann ist die vorgegebene Anzahl N an Zählpulsen bei der hohen Aktivität sehr schnell erreicht, während man bei einer Zeitintervall-Vorgabe das komplette lange Messintervall abwarten muss um mitzubekommen, das sich etwas getan hat.

Trotzdem gibt es Fälle, da ist die Zeitintervall-Vorgabe der bessere Weg. Das ist dann der Fall wenn sich die mittlere Anzahl der Zählpulse stets über einem statistischen sinnvollen Wert λ bewegt und damit eine ausreichende Genauigkeit gesichert ist. Dann ist ein konstanter Update der berechneten Zählrate in einem festen Takt oft die bevorzugte Lösung.

Die Zeitintervallvorgabe lässt sich für die Implementierung auf dem Arduino sehr leicht aus dem Programm für die Zählimpulsvorgabe ableiten.

```
#define MAXCNT 10
#define CalFactor 1

volatile int counter = 0;
unsigned long oldTime = 0;

void setup() {
  Serial.begin(9600);
  attachInterrupt(0, count, RISING);
}

void loop() {
  unsigned long time;
  unsigned long dt;
  float rate;

  if (counter == MAXCNT) {
    time = millis();
    dt = time-oldTime;
    rate = (float)MAXCNT*1.0*1000.0/(float)dt/CalFactor;
    Serial.println(rate);
    oldTime = time;
    counter = 0;
  }
}

void count()
{
  counter++;
}
```

Listing 1: Einfacher Zähler mit Zählimpuls-Vorgabe und Ausgabe der Rate auf dem Serial Monitor

Die Zählimpulsvorgabe ist in Listing 1 dadurch realisiert, dass in der Schleife loop() mit der Anweisung if (counter == MAXCNT) abgefragt wird, ob die Anzahl an Zählimpulsen MAXCNT erreicht wurde. Wenn ja, wird mit dem inneren Konstrukt der Schleife die vergangene Zeit bestimmt und daraus die Rate berechnet und auf dem Serial Monitor ausgegeben.

Das Listing 2 verwendet dagegen eine Zeitintervall-Vorgabe. Zudem wird eine Ausgabe auf einem seriellen LCD-Display gemacht, so dass der Arduino mit dem Display als universelles Zählgerät in ein Gehäuse gebaut werden kann. Dieses Zählgerät betimmt dann die mittlere Rate der am Interrupt Eingang (Digital Pin 2) statistisch einreffender digitaler Pulse in Pulse Pro Minute (counts per minute, cpm) . Als Zeitvorgabe sind in diesem Beispiel eine Zeitintervall von 10 Sekunden vorgesehen (#define tInterv 10000). In diesem Programm wird in der Schleife loop() mit if (dt >= tInterv) abgefragt, ob das Zeitintervall schon erreicht ist. Wenn das der Fall ist wird im inneren Konstrukt der Schleife die rate berechnet und auf dem LCD Display ausgegeben.

```
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
// Define variables

#define I2C_ADDR    0x27  // Define I2C Address where the PCF8574A is
#define BACKLIGHT_PIN     3
#define En_pin  2
#define Rw_pin  1
#define Rs_pin  0
#define D4_pin  4
#define D5_pin  5
#define D6_pin  6
#define D7_pin  7


#define tInterv 10000
// counts in cpm

volatile int counter = 0;
unsigned long oldTime = 0;

//Initialise the LCD
LiquidCrystal_I2C   lcd(I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);

void setup() {

// Define LCD as 16 column x 2 rows
  lcd.begin (16,2);

// Switch on the backlight
  lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);
  lcd.setBacklight(HIGH);
```

```
// Goto first column (0 not 1!), first line (0 not 1!),
  lcd.setCursor ( 0, 0 );

// Print at cursor location
  lcd.print("opengeiger.de");

// Go to first column (0 not 1!), second line (which is 1 not 2!)
  lcd.setCursor ( 0, 1 );

// Print at cursor location
  lcd.print("CPM:");

  Serial.begin(9600);
  attachInterrupt(0, count, RISING);
}

void loop() {
  unsigned long time;
  unsigned long dt;
  //float rate;
  int rate;

  time = millis();
  dt = time-oldTime;
  if (dt >= tInterv) {
    rate = counter*60000/ tInterv;

    lcd.setCursor ( 5, 1 );
    lcd.print("          ");
    lcd.setCursor ( 5, 1 );
    lcd.print(rate);


    oldTime = time;
    counter = 0;
  }
}

void count()
{
  counter++;
}
```

Listing 2: Einfacher Zähler mit Zeitintervall-Vorgabe und Ausgabe der Rate auf einem seriellen LCD-Display.

Man kann die korrekte Funktion des Zählgeräts sehr einfach mit einem Pulsgenerator testen, der allerdings seine Pulse streng periodisch anliefert. Gibt man beispielsweise Rechteckpulse mit 3.6kHz auf

den Interrupteingang, dann entspricht das 3600 Pulsen pro Sekunde also 60 pro Minute, so dass das Gerät nun auch 60 cpm anzeigen muss, wenn es korrekt funktioniert.

Dieses Gerät lässt sich nun nach Belieben an Radioaktivitäts-Detektoren anschließen, die einen digitalen Zählimpulsausgang besitzen. Der digitalen Zählimpulsausgang wird meist durch einen Komparator in der Detektorschaltung erreicht. Um eine Zerstörung des Arduino Interrupteingangs zu verhindern ist aber wichtig, dass der Pegel der digitalen Zählpulse zur Spannungsversorgung des verwendeten Arduino passt. Viele Arduino typen arbeiten mit 5V manche mit 3.3V Spannungsversorgung. Man kann aber eine Schutzschaltung anbringen, die eine Pegel-Angleichung vornimmt. Die einfachste Variante ist die Verwendung einer Zenerdiode mit 5.1V mit einem Vorwiderstand (für einen 5V Arduino). Die etwas sauberere Lösung ist die Verwendung einer schnellen Clamping Diode, die mit der Versorgung des Arduino verbunden wird und sich damit automatisch an die jeweilige Versorgung des verwendeten Arduino Typen anpasst. Damit kann dann die Spannung am Interrupt-Eingang nie größer werden als die Versorgungsspannung. Der Rest der Komparatorausgangsspannung fällt an dem Vorwiderstand (z.B. 1k) ab.

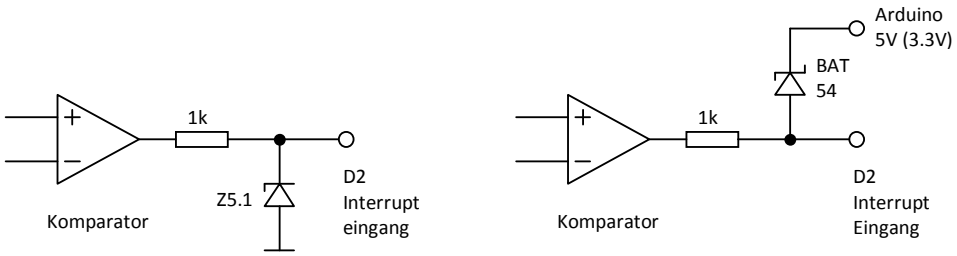

Abb. 1: Zwei mögliche Pegel-Anpass-Schaltungen für den Arduino Interrupt-Eingang des Zählgeräts

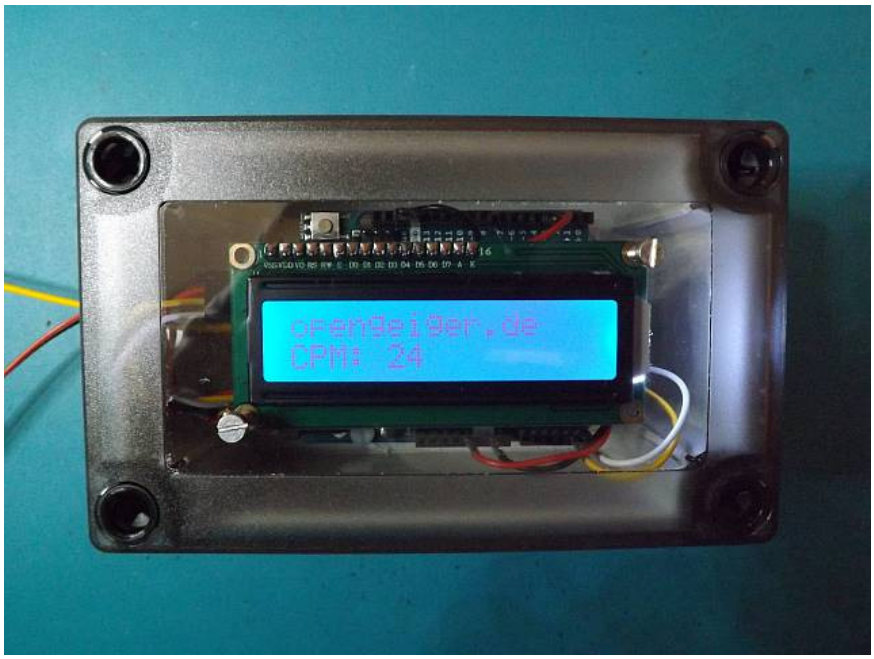

Abb. 2: Pulsraten-Zähler mit Arduino Uno und LCD-Anzeige und Zählintervallvorgabe (10 Sekunden)