Der Geiguino - Ein Geigerzähler auf Zählrohrbasis kombiniert mit Arduino UNO zur Auswertung und Anzeige

Bernd Laquai, 6.9.2013

Es ist zwar nicht die stromsparendste Lösung einen Arduino dazu zu benutzen um einen auf einem Geiger-Müller-Zählrohr basierendes, portables Kern-Strahlungsmessgerät zu bauen aber mit Sicherheit die einfachste, wenn man eine flexible Nachverarbeitung des Sensorsignals und eine digitale Anzeige anvisiert. Dennoch kann man trotz LCD-Anzeige mit Backlight unter Verwendung der ganz normalen Arduino UNO Platine einen Stromverbrauch von weniger als 70mA bei 5V erreichen. Die Tatsache, dass man mit dem Arduino eine tolle Entwicklungsumgebung und Zugriff auf eine große Zahl an fertigen Programmbibliotheken bekommt und dazu eine riesen Auswahl an zusätzlichen fertig bestückten und getesteten Modulen für alle möglichen Zusatzfunktionen hat, rechtfertigt dieses Vorgehen. Dazu gehören vor allem LCD-Anzeigen, SD-Karten Daten-Logger und Kommunikationsmodule.

Trotzdem ist es sinnvoll, eine stromsparende Hochspannungserzeugung und einen stromsparenden Zählrohrverstärker zu verwenden, um die Stromversorgung nicht noch zusätzlich dadurch zu belasten. Aus diesem Grund soll hier die auf dem LT3468 beruhende low-power Hochspannungsquelle und die stromsparende Komparatorschaltung benutzt werden um vom Arduino und einem Open-Source Geiger-Müller Zählrohr- Zähler zu dem als Geiguino bezeichneten Gerät zu kommen. Beides ist auf der opengeiger.de Webseite im Detail beschrieben.

Die Abbildungen 1 und 2 zeigen noch einmal die beiden Funktionsblöcke für die Hochspannungsversorgung und den Zählrohrverstärker. Wobei beide Blöcke natürlich auch an ein anderes als das verbreitete Geiger-Müller Endfenster Zählrohr ZP1400 angepasst werden können, indem eine andere Größe der Hochspannung, des Anodenvorwiderstands und des Messwiderstands entsprechend des Datenblatt des Herstellers gewählt wird.

Hinsichtlich der Stromversorgung muss allerdings beachtet werden, dass der Sperrwandler relativ starke Stromspitzen erzeugen kann, vor allem beim Hochlaufen nach dem Einschalten, solange die Hochspannungskapazität noch nicht auf die volle Endspannung von 400V geladen ist. Deswegen muss die eigentliche Spannungsquelle, welche wegen des Arduino 5V Spannung erzeugen sollte, auch entsprechend Strom liefern können, wenn es nötig ist.

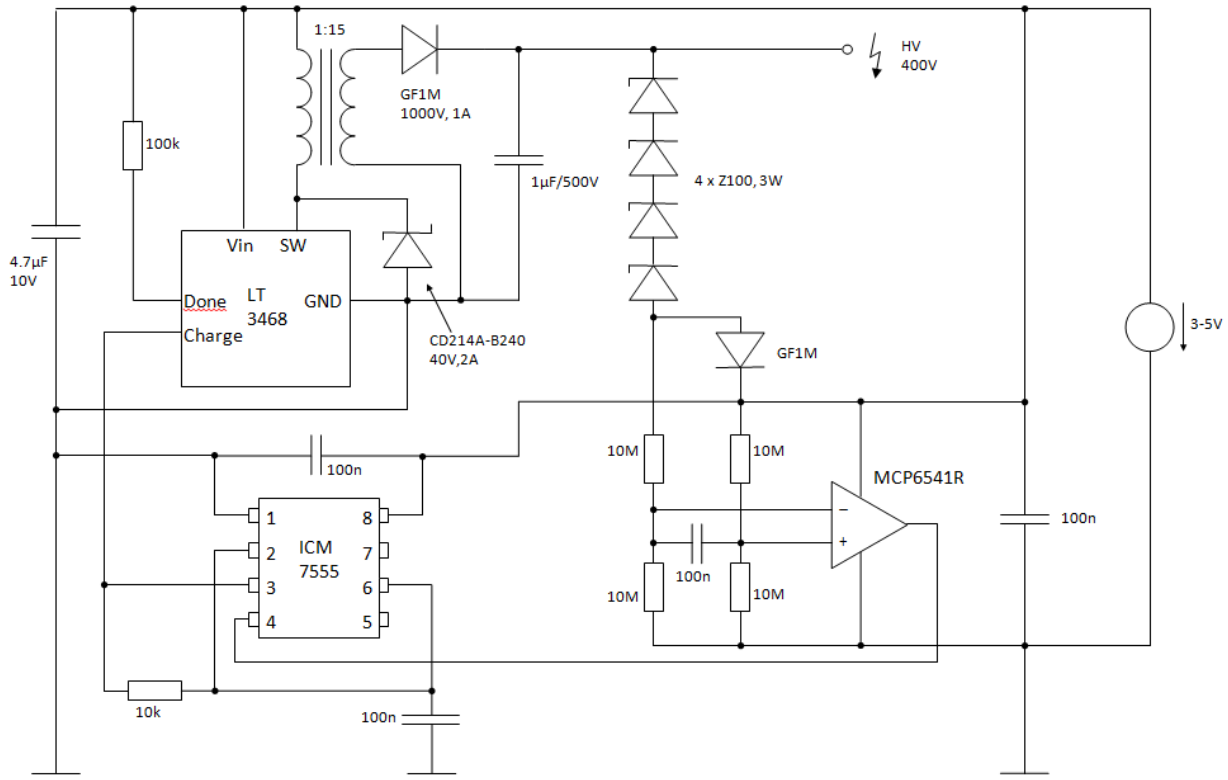

Abb. 1: Geregelte Hochspannungserzeugung mit einer Sperrwandlerschaltung und dem Photo-Flash IC LT3468

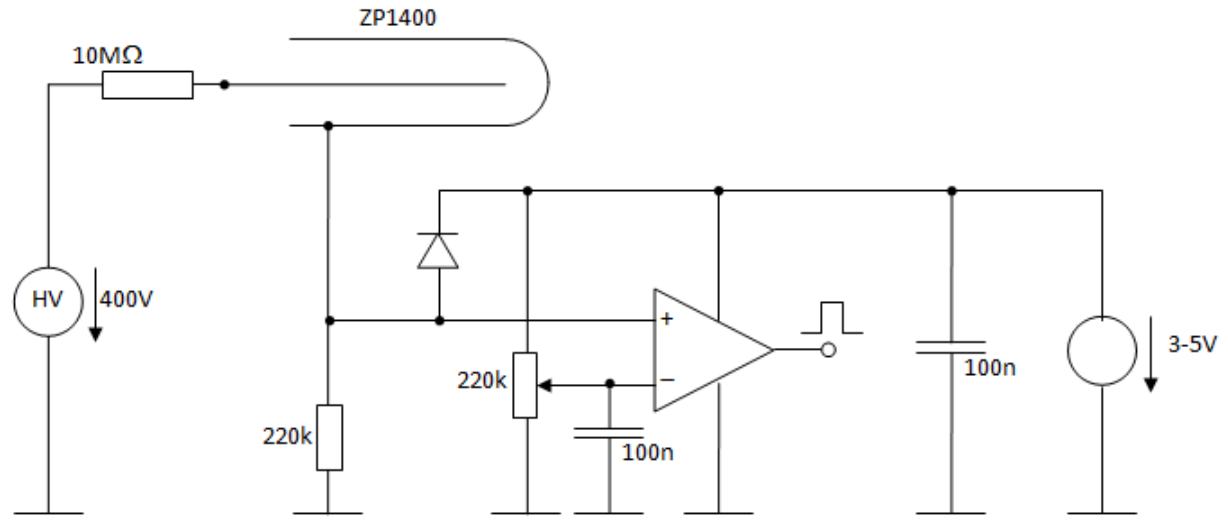

Abb. 2: EInfacher und stromsparender Zählrohrverstärker

Aus diesem Grund sollte der Arduino und die Hochspannungsversorgung mit Zählrohrverstärker nicht über den On-Board Linearregler betrieben werden, sondern direkt aus einer USB Spannungsquelle, die auch mindestens 1A liefern kann. Dazu bietet sich ein getaktetes USB Netzteil an, ein (guter) USB-Adapter für die 12V Autosteckdose oder eine USB Power-Bank auf der Basis von zwei Lithium-Ionen Akkus, wie sie als Notstromquelle für Smartphones und andere USB-versorgte Elektronikgeräte angeboten werden. Zusätzlich sollte man noch zur Vorsicht eine große Stützkapazität (Elko > 100uF) direkt am Spannungsanschluss der Hochspannungsquelle anbringen, welche die größten Spannungsschwankungen abpuffert und damit vom Zählrohrverstärker bzw. dem Arduino fernhält.

Als digitale Anzeige der Messdaten bietet sich ein stromsparendes LCD-Display an. Meist sind solche LCD Display mit einem Parallelbus nach dem Hitachi-Standard ausgestattet. Der direkte Anschluss eines solchen Displays würde aber viele Digitalpins auf der Arduino-Platine belegen. Deswegen wurde hier ein LCD-Display gewählt, welches einen I2C-Bus Anschluss hat. Die meisten I2C-Bus Displays haben aber in Wahrheit dennoch einen Controller mit Parallelbus, sie verwenden jedoch eine kleine Zusatzplatine auf der ein PCF8574 Baustein sitzt (Remote 8-bit I/O expander for I2C-bus), der den Parallelbus in den I2C-Bus übersetzt. Die passende Bibliothek zu den I2C-Bus Displays mit dem Namen „LiquidCrystal_I2C“ bekommt man von Fransisco Malpartida unter:

https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads

Das hier eingesetzte zweizeilige I2C-Bus LCD Display stammte von dem Hersteller YwRobot mit der Bezeichnung Arduino LCM1602 IIC V1. Das Modul hat einen Stromversorgungsanschluss und benötigt sonst nur noch die zwei Datenleitungen SDA und SCL des I2C-Buses zum Anschluss an den Arduino.

Insgesamt ergibt sich so für den Geiguino V1 das folgende Blockschaltbild:

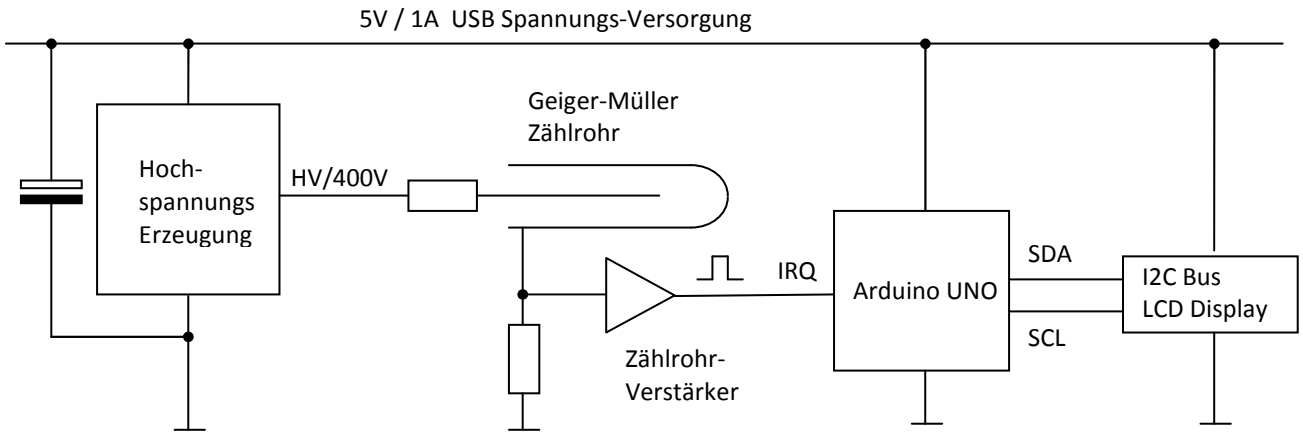

Abb. 3: Blockschaltbild des Geiguino V1

An dem I2C-Bus LCD Display sind die beiden Bussignale SDA und SCL neben den Vdd und Vss Versorgungsanschlüsssen auf dem Interface Piggy-Back Board herausgeführt. Wenn auf dem Arduino die

„LiquidCrystal_I2C“ Library zusammen mit der I2C Library „Wire“ verwendet wird, dann müssen auf dem Arduino UNO für das SDA Signal der Analog Pin A4 und für das SCL Signal der Analog Pin A5 verwendet werden.

Da der Zählrohr-Verstärker die Zählimpulse zu völlig unvorhersehbaren Zeitpunkten liefert, bietet es sich an, zur Registrierung der Zählimpulse die Interrupt-Logik des Arduino zu verwenden. Auf dem Arduino UNO wird hierzu der Digital Pin D2 verwendet. Es ist ebenfalls wichtig, dass der Zählrohr-Verstärker die Zählimpulse mit positiven Signalen übergibt, das legt nahe, die Interrupt-Logik mit der steigenden Flanke eines Pulses (rising edge) zu triggern.

Ein einfaches Listing für den Betrieb des Geiguino ist in Tab. 1: gegeben. Dieses Programm gibt lediglich die Zahl der in jeweils 10 Sekunden gezählten Impulse in der zweiten Zeile des LCD Displays wiederholt aus. Andere Arten der Ausgabe bleiben der Kreativität des Einzelnen überlassen.

```
#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 maxTime 10000

volatile unsigned int cnt = 0;
unsigned long time;
unsigned long oldTime = 0;
unsigned int dt;

//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("Geiguino");

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

// Print at cursor location
  lcd.print("Count:");
  attachInterrupt(0, count, RISING);
}

void loop()
{
  time = millis();
  dt = time - oldTime;
  if (dt > maxTime) {
    lcd.setCursor (7,1);
    lcd.print("      ");
    lcd.setCursor (7,1);
    lcd.print(cnt);
    cnt=0;
    oldTime = millis();
  }
}

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

Tab. 1: Einfaches Listing zur Ausgabe der Zahl der Zählimpulse innerhalb von 10 Sekunden

Die Initialisierung und Ansteuerung des LCD-Displays ist aus Anleitungen, die sich im Internet finden lassen entnommen. Die I2C-Bus LCD-Bibliothek “LiquidCrystal_I2C” wird mit dem include Statement “#include <LiquidCrystal_I2C.h>” eingebunden. Damit der Compiler darauf zugreifen kann, muss die Bibliothek zunächst in das Arduino Entwicklungssystem importiert werden. Unter dem entsprechenden Menupunkt (sketch->Import Library…) kann man dazu das heruntergeladene zip File direkt angeben, der Rest geht automatisch.

Für die Implementierung der Zählraten-Erfassung wurde ein ganz einfaches Konzept eingesetzt, welches lediglich die Zahl der Impulse in einem vorgegebenen Zeitintervall zählt. Dabei löst ein Zählimpuls des Zählrohrs einen Interrupt-Request aus, der, wenn er auftritt, eine Programmverzweigung aus der normal abgearbeiteten Endlosschleife bewirkt. Damit das funktioniert, muss mit dem Statement

„attachInterrupt(0, count, RISING)“ erklärt werden, dass die Interrupt Service Routine count heißt und dass der Interrupt auf dem Kanal 0 (digital Pin 2 beim UNO) mit der rising edge ausgelöst wird. Solange kein Zählimpuls auftritt, läuft das Programm in einer Endlosschleife, in der in regelmäßigen Zeitabschnitten (hier alle 10000msec) der Zählerstand, welcher in der Variablen cnt gespeichert ist, ausgegeben wird. Der Ablauf eines Zeitintervalls wird aus der Zeitdifferenz zum letzten Ausgabe-Zeitpunkt bestimmt. Sobald also die Zeitdifferenz (dt) zwischen dem Zeitpunkt der letzten Ausgabe (oldTime) und der momentanen Zeit (time) größer ist als der Wert für das Zeitintervall (maxTime) wird eine neue Ausgabe gemacht, ansonsten passiert nichts. Nach einer erfolgten Ausgabe wird der Zähler zurückgesetzt und der Ausgabezeitpunkt in oldTime neu abgespeichert.

Tritt ein Zählimpuls auf, verzweigt der Programmfluss in das Unterprogramm count(), in dem lediglich der Zähler inkrementiert wird. Danach kehrt die Programmausführung sofort wieder in die Endlosschleife zurück.

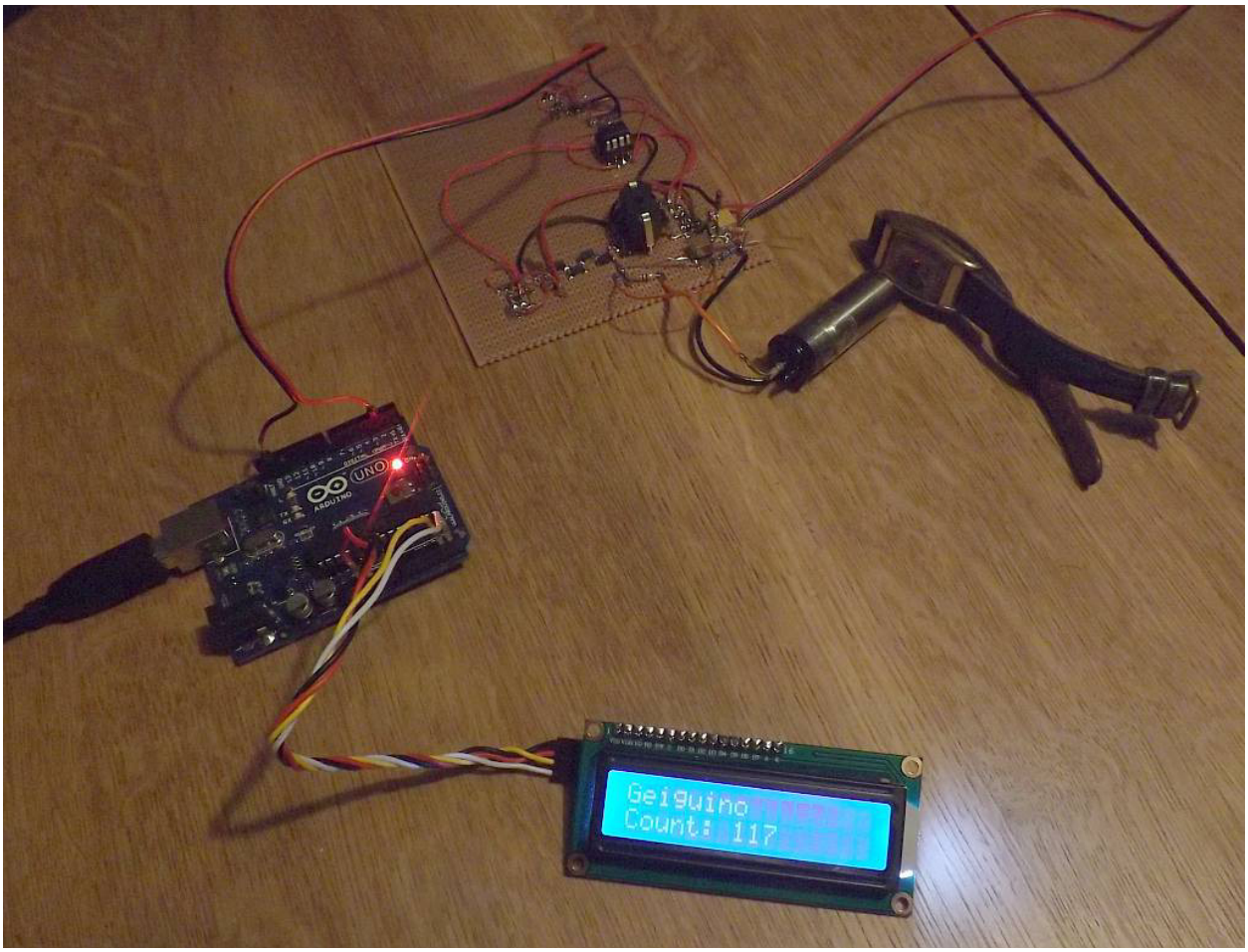

Abb. 4: Prototyp des Geiguino_V1 aufgebaut mit einer Lochrasterplatine