Eigenbau einer H*(10)-fähigen Sonde zur Messung der Gamma-Ortsdosisleistung für Citizen Science Anwendungen in der Radioökologie

Bernd Laquai, 6.2.2024

Das Projekt Openstreetmap kann sicherlich als Paradebeispiel für einen gesellschaftlich äußerst wertvollen Beitrag von Citizen Science angesehen werden. Ein wesentlicher Erfolgsfaktor bei diesem Projekt rührt daher, dass niemand besser die kartenrelevanten Details, wie z.B. den Straßen- und Wegeverläufe, so schnell und zuverlässig liefern kann, wie die lokale Bevölkerung vor Ort. Im Gegensatz dazu muss eine Vermessungsbehörde die kartenrelevanten Veränderungen mit hohem Personalaufwand in großen Regionen erfassen und altes Kartenmaterial entsprechend aktualisieren, was zwangsläufig dazu führt, dass amtliche Karten der Zeit meist stark hinterherhinken. Der Erfolg von Openstreetmap beruht aber auch auf der Verfügbarkeit von Mobiltelefonen und speziellen Wandernavigationsgeräten mit hochgenauen Empfängern für Satelliten-Navigationssysteme wie GPS, GLONASS und Galileo, die sehr einfach und unkompliziert von normalen Bürgern bedient werden können. Dadurch ergeben sich Geoinformations-Daten mit hoher Präzision, welche den amtlich erfassten Daten in der Qualität nur um wenig nachstehen.

Bei der Erfassung von radioökologischen Daten wie beispielsweise der Ortsdosisleistung (ODL) ist die Herausforderung an die Behörden hinsichtlich der zeitnahen Erfassung ähnlich herausfordernd wie bei der geodätischen Datenerfassung. Dies hat sich insbesondere nach der Nuklearkatastrophe in Fukushima-Daiichi 2011 sehr deutlich gezeigt. Weder der Betreiber des havarierten Kernkraftwerks noch die Behörden waren in der Lage in angemessener Zeit die notwendigen Daten über die radioaktiven Kontaminationen der Umwelt zu erfassen und die betroffene Bevölkerung korrekt und ausreichend zu informieren. Diese Arbeit wurde schließlich von Bürgern selbst geleistet, die, mit einfachen Geigerzählern und GPS-Geräten ausgestattet, Straßen abgefahren und Wege abgelaufen sind um die Daten in die IoT-Plattform Pachube (später safecast.org) einzuspeisen und sie dort in elektronischen Karten für jedermann sichtbar machten. Diese Karten werden bis heute noch aktuell gehalten (https://map.safecast.org/). Dies zeigt recht deutlich, dass Citizen Science im Bereich der Radioökologie durchaus sehr sinnvoll sein kann. Dazu muss aber nicht nur entsprechendes Wissen in der Bevölkerung vorhanden sein, wie solche Messungen mit einer der Verantwortung entsprechenden Datenqualität durchzuführen sind, sondern auch die geeigneten Messgeräte.

Mit dieser Erkenntnis im Hintergrund wurde von der Europäischen Metrologie Vereinigung EURAMET im Rahmen des European Metrology Program for Innovation and Research (EMPIR) das Projekt 16ENV04 “Preparedness” gestartet, in dem untersucht werden sollte, in welcher Form man nach Eintreten eines nuklearen Notfalls zusätzlich Daten verwerten könnte, welche von Bürgern mit typischen Consumer-Geräten erfasst werden, um die Daten der behördlich betriebenen Messnetze zu ergänzen /1/. Neben der Tatsache, dass Strahlungsmessungen von unerfahrenen Bürgern häufiger mit Fehlern behaftet sein können, als beim Mapping von Wegen und Straßen für Openstreetmap, stellt auch die Verwendung von Consumer-Messgeräten mit unterschiedlichen Eigenschaften und unterschiedlicher Qualität, eine deutlich größere Herausforderung dar. Dies gilt insbesondere dann, wenn es um quantitative Aussagen zu den radioökologischen Auswirkungen für die Bevölkerung geht und nicht nur um das pure Feststellen einer signifikanten Boden-Kontamination.

Aus diesem Grund untersuchten zwei nationale Metrologie-Behörden, die deutsche PTB und die britische NPL, sowie etliche europäische Institute für Nukleartechnik die Fähigkeiten von Consumer Messgeräten, die im Citizen Science Bereich gerne als mobile Messgeräte für Strahlungsmessungen verwendet werden und solche, welche häufig in Citizen Science Messnetzwerken zur Erfassung von radioaktiver Strahlung eingebunden werden. Diese Messgeräte wurden hinsichtlich ihrer Eignung für die im Projekt definierte „Preparedness" bewertet /2/. Während Eigenschaften wie „Detektor Hintergrundrate", „Linearität", „Empfindlichkeit auf sekundäre kosmische Strahlung", „Antwort auf kleine Änderungen der Dosisleistung" und „Stabilität der Messwerte unter verschiedenen Klimabedingungen", bei den meisten der untersuchten Geräte als noch akzeptabel angesehen wurden, kam das Experten Team zu der Erkenntnis, dass die Energieabhängigkeit der unkompensierten Detektoren die Nutzung für quantitative Messungen weitestgehend verhindert. Im Wesentlichen wird den Consumer-Geräten also nur die Detektion einer Kontamination und die Möglichkeit einer qualitativen Abbildung einer Ausbreitungsfahne auf Kartenmaterial zugetraut.

Ein gezielter Eigenbau eines Strahlungsmessgeräts mit einem Schwerpunkt auf der Energiekompensation könnte aber genau diesen Makel von Consumer-Geräten vermeiden. Daher soll im Folgenden ein Selbstbau-Messgerät für die Messung der Gamma-Ortsdosisleistung beschrieben werden, welches ein energiekompensiertes Geiger-Müller-Zählrohr verwendet, wie es auch in den ODL-Sonden des Bundesamts für Strahlenschutz eingesetzt wird. Ein solcher Selbstbau ist um Größenordnungen kostengünstiger als der Kauf eines professionellen Messgeräts, das ebenfalls ein energiekompensiertes Zählrohr verwendet und kann dazu beitragen, dass Citizen Science Communities nicht an den Kosten professioneller Geräte scheitern. Mit Hilfe von zusätzlichen Bildungs- und Kalibrierangeboten könnten die Behörden und Forschungseinrichtungen die ehrenamtliche Betätigung der Bevölkerung im Bereich der Radioökologie fördern und dann schließlich auch auf Daten mit hoher Qualität zugreifen, ganz ähnlich wie die Behörden auch Openstreetmap-Daten nutzen.

Der Selbstbau einer geeigneten Sonde lohnt sich nicht nur wegen des dabei erworbenen Verständnisses, sondern auch rein ökonomisch, selbst dann, wenn ein neues energiekompensiertes Zählrohr, wie es in den BfS ODL-Sonden verwendet wird, in der Größenordnung von 500Euro liegt. Der Kaufpreis entsprechender professioneller Geräte liegt nämlich um ein Vielfaches höher. Eine noch kostengünstigere Alternative ergibt sich durch den Kauf eines gebrauchten energiekompensierten Zählrohrs, wobei man natürlich das Risiko eingeht, dass es in irgendeiner Form defekt sein kann, was nicht immer einfach zu erkennen ist. Eine umfangreiche Funktionsprüfung und eine geeignete Kalibration sind daher zwingend nötig, wenn man mit einem solchen Messgerät Daten mit hoher Qualität erzeugen will.

Das energiekompensierte Zählrohr, das heute in den ODL-Sonden des BfS verbaut wird, ist das Zählrohr vom Type 70031A der Dresdener Firma VacuTec. Dieses Zählrohr ist speziell auf die neue Ortsdosisgröße H*(10) energiekompensiert und berücksichtigt damit auch die neue Messmethode der Ortsdosis mit Hilfe des IRCU-Kugelphantoms. Es unterscheidet sich damit vom Zählrohrtyp 70031E, welches zwar auch energiekompensiert ist, jedoch in Bezug auf die alte Größe für die Ortsdosis Hx, welche aus der Ionendosis mit Hilfe von Strahlungs-Wichtungsfaktoren berechnet wurde.

Abb. 1: Das energiekompensierte VacuTec Zählrohr 70031A im Größenvergleich zu einem normalen Schraubenzieher

Für das VacuTec 70031A Zählrohr sind auch die spezifizierten Daten des Herstellers im Internet zu finden. Es ist mit einer Dosisempfindlichkeit von 14cps/(uSv/h) bzw. 840cpm/(uSv/h) und einer Eigen-Nullrate von 40cpm für Cs137 vom Hersteller spezifiziert. Die entscheidende Größe in Bezug auf die Datenqualität der Messdaten ist das Ansprechvermögen als Funktion der Energie. Das Ansprechvermögen beschreibt wie stark die Zählrate von der Energie abhängt und wird dabei auf die Zählrate bei Verwendung einer Cs137-Quelle normiert. Idealerweise wäre es wünschenswert, wenn die Zählrate eines Zählrohrs ganz unabhängig von der Energie der Strahlung streng proportional zur Gamma-Ortsdosisleistung wäre. Bei normalen Geiger-Müller Zählrohren ist das jedoch nur der Fall, wenn sich die das Radionuklid (bzw. das Gemisch von Radionukliden) von Messung zu Messung nicht ändert. Wenn daher ein normales Geiger-Müller-Zählrohr mit der Dosisleistung auf eine Cs137 Quelle kalibriert wird, dann ist diese Kalibrierung auch nur für Cs137 als Strahlungsquelle gültig und ein Messergebnis wäre dann beispielsweise auch nur für Cs137 Kontaminationen korrekt. Bei natürlichem Uran als Strahlungsquelle würde sich eine nicht unerhebliche Abweichung ergeben, da die Zählrate für das im Unat. enthaltene Pb214 beispielsweise eine andere ist, als die für das Cs137.

Bei einem ideal energiekompensierten Zählrohr wäre es dagegen völlig egal, für welches Radionuklid das Zählrohr mit der Dosisleistung kalibriert ist, es würde unabhängig von den als Quelle vorliegenden Radionukliden stets dieselbe Zählrate ergeben, so dass die Proportionalität zwischen Zählrate und Dosisleistung immer dieselbe ist. Ganz perfekt lässt sich eine solche ideale Energiekompensation mit Hilfe der Filterschichten, die um das Zählrohr gewickelt werden, aber nicht erreichen. Deswegen wird der nutzbare Energiebereich meist entsprechend eingeschränkt (beim 70031A ist das 35kev-2.6MeV). Aber auch in diesem eingeschränkten Energiebereich ist das Ansprechverhalten über der Energie nicht ganz konstant, sondern es verbleibt eine gewisse Restwelligkeit, die dann aber als vernachlässigbar angesehen wird. Der Dosisleistungsbereich, für den das 70031A Zählrohr dabei einsetzbar ist, ist ebenfalls beschränkt und liegt bei 50nSv/h bis 3mSv/h.

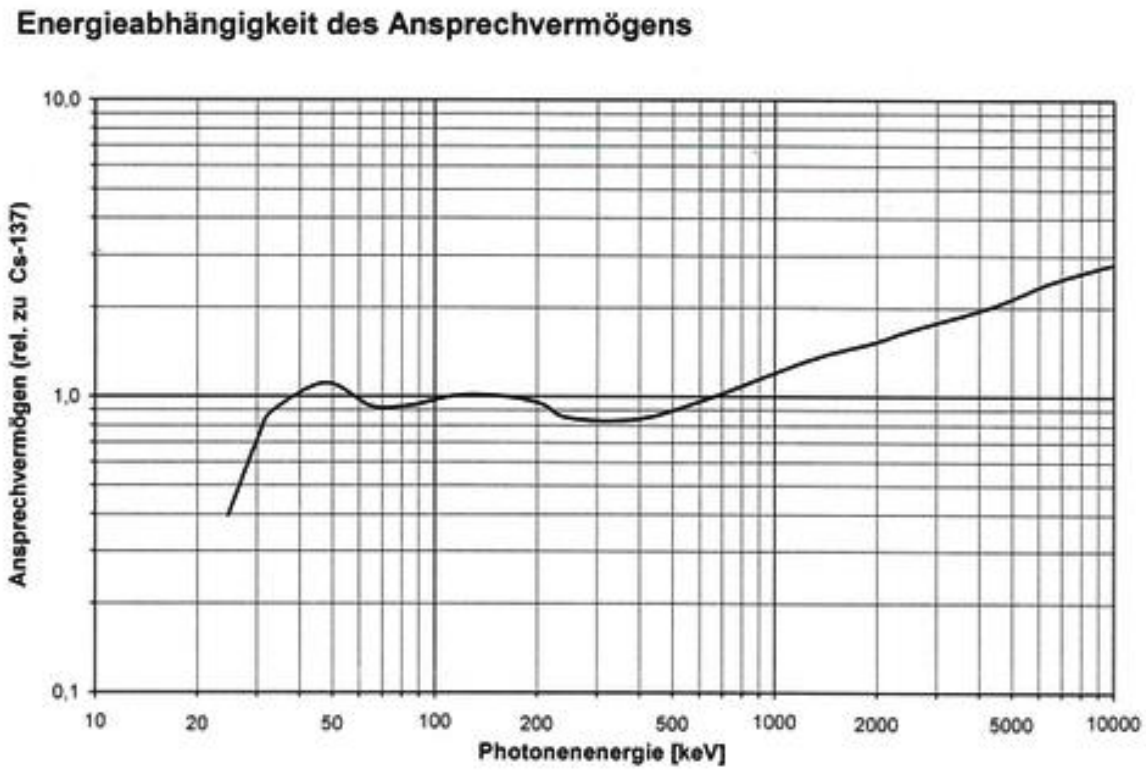

Abb. 2 Energieabhängigkeit des Ansprechvermögens beim 70031A Zählrohr von VacuTec

Ziel ist es nun, für dieses Zählrohr eine bezüglich der Kenndaten möglichst transparente Zähl- und Auswerteelektronik zu entwickeln, so dass damit ein Messgerät entsteht, welches ausschließlich durch die Eigenschaften des Zählrohrs bestimmt wird. Dies ist durch Verwendung eines passenden Zählrohr-Treibermoduls, und einem Mikrocontroller überraschend einfach möglich. Das Zählrohr-Treibermodul, erzeugt dabei eine geeignete Hochspannung, führt die Signalauskopplung durch und wandelt die analogen Zählimpulse in digitale Zählimpulse um, ohne die Totzeit des Zählrohrs zu erhöhen. Zudem muss sichergestellt sein, dass die zählende Elektronik in dem anvisierten Dosisleistungsmessbereich auch für die höheren Dosisleistungen nicht in die Sättigung gerät. Dies ist aber bei der Verwendung eines einfachen, modernen Mikrocontrollers, welcher die zählende Messung per Software durchführt, bis etwa 100uSv/h (1000cps) kein besonderes Problem.

Als Zählrohr-Treibermodul soll hier das GA-500 Modul von Theremino Verwendung finden, das klein und sehr kostengünstig ist und über Ebay aus Italien bezogen werden kann. Als Mikrocontroller dient ein einfacher und kostengünstiger Arduino-Uno Mikrocontroller an den ein zweizeiliges LCD-Display mit 16 Zeichen und I2C-Bus Interface der Firma Seeedstudio zur Anzeige der Messwerte angeschlossen wird. Um schließlich zu dem Konzept der OM01-Gamma-ODL Sonde zu gelangen, wurde diese Grundschaltung zusätzlich noch um ein Datalogging-Shield mit Real-Time-Clock der Firma Adafruit erweitert, so dass die Messdaten nicht nur angezeigt, sondern auch auf eine SD-Karte gespeichert werden können. Optional kann noch ein Piezo-Lautsprecher angeschlossen werden um die

Zählimpulse akustisch zu signalisieren. Der Piezo-Lautsprecher sollte jedoch keine eigene Elektronik enthalten, die bereits selbst versucht ein Ton zu erzeugen.

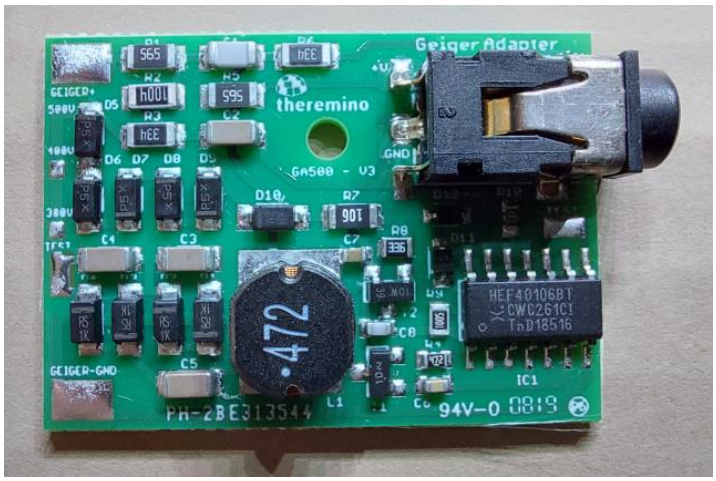

Abb. 3: Theremino Zählrohr-Treibermodul GA-500

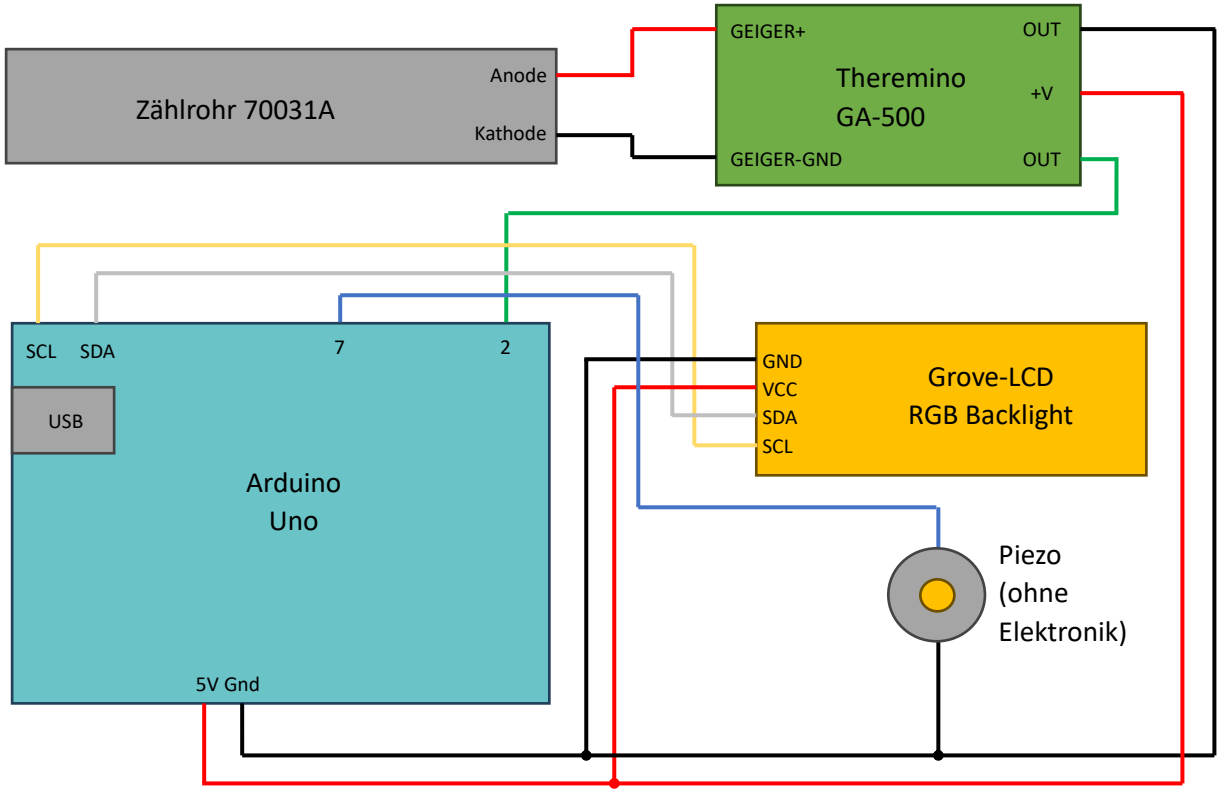

Abb. 4: Schematische Beschreibung der Verschaltung der verschiedenen Module zur einer einfachen Grundschaltung für Zählratenmessungen bestehend aus Zählrohr, Zählrohr-Treibermodul, Zähl- und Auswertemodul, LCD-Anzeige und akustischer Signalisierung

Weitere wichtige Daten des 70031A Zählrohrs sind nun die Arbeitsspannung von 500V und der Anodenwiderstand von 5.1MΩ. Das Theremino Treibermodul ist auf die Spannung von 500V

voreingestellt und hat auf der Platine bereits einen 5.6MΩ Anodenwiderstand installiert. Die kleine Differenz zu den 5.1MΩ für den Anodenwiderstand im Datenblatt des 70031A Zählrohrs kann ignoriert werden. Wichtig jedoch ist, dass das Treibermodul nahe am Zählrohr montiert wird, so dass das Anodenkabel (rot) und das Kathodenkabel ohne es zu verlängern am Treibermodul angelötet werden kann. Die Anoden- und Kathodenzuleitung sollten nicht verlängert werden, damit keine zusätzliche Kapazität entsteht, welche Ladung speichern könnte, da dies sowohl die Lebensdauer wie auch die Signalqualität beeinträchtigen könnte. Die übrige Verschaltung ist unkritisch und kann Abb. 4 entnommen werden.

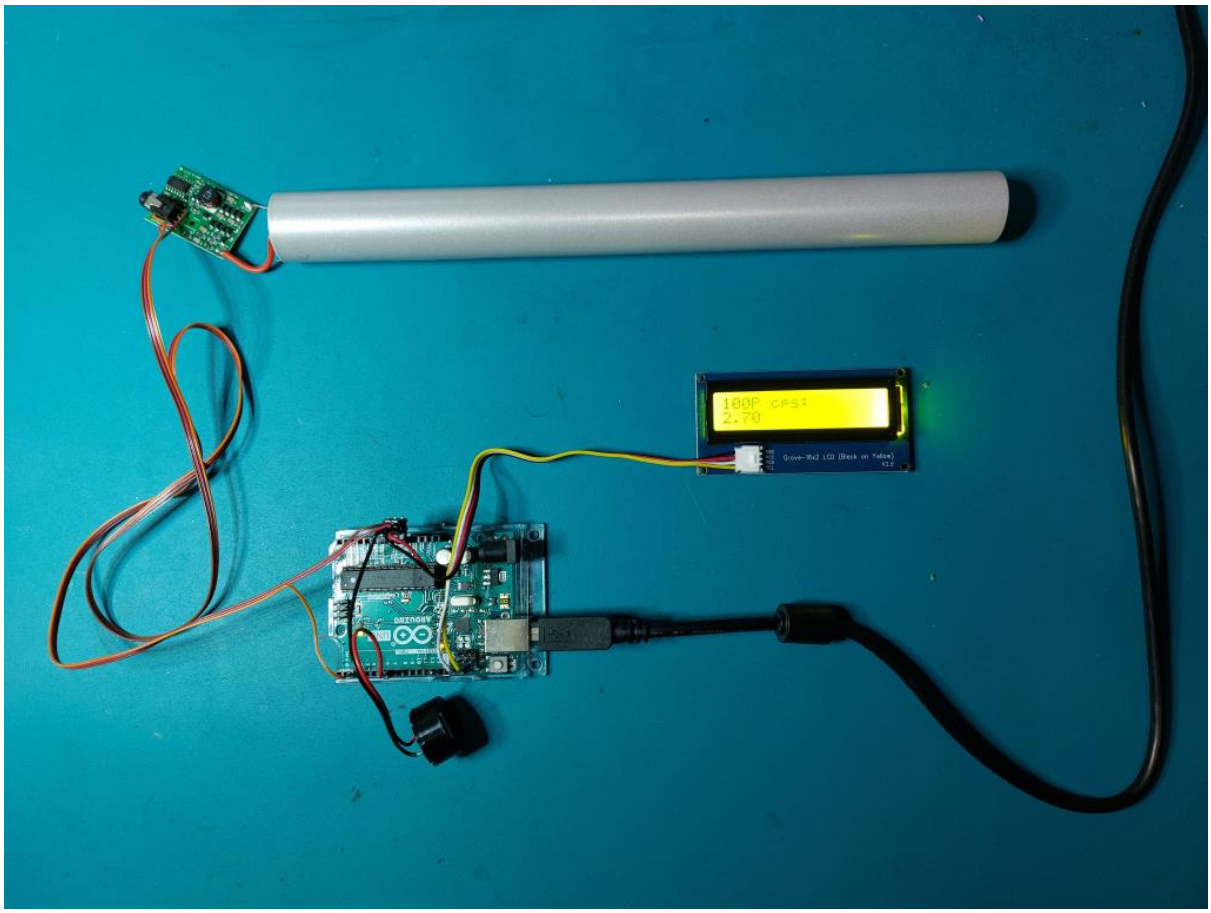

Abb. 5: „Fliegender Aufbau“ der Grundschaltung

Ein minimaler Sketch (Software-Code für den Arduino), mit dem die Grundschaltung noch ohne das Display und den Piezo-Lautsprecher in Betrieb genommen werden kann, ist folgender:

```
volatile int counter = 0;
unsigned long oldTime = 0;
int maxcnt = 100;

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

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

void loop()
{
  unsigned long time;
  unsigned long dt;
  float rate;
  if (counter >= maxcnt) {
    time = millis();
    dt = time-oldTime;
    rate = (float)maxcnt*1000.0/(float)dt;
    Serial.println(rate);
    oldTime = time;
    counter = 0;
  }
}
```

Listing 1: Minimal-Sketch für die Impuls-Zählung und Zählraten Auswertung mit dem Arduino Uno

Der Zähler für die Zählimpulse wird als globale Variable mit dem Namen counter vereinbart. Die Variable oldTime speichert die Zeit nach einer Auswertung, damit die Zählimpulsrate aus der Zeitdifferenz zwischen der neuen Zeit und der gespeicherten alten Zeit berechnet werden kann. Diese Zeitdifferenz dt stellt das Messzeitintervall dar. Die Anzahl der Impulse auf die gewartet wird, bevor eine Zählimpulsrate berechnet wird, wird in maxcnt gespeichert und wird mit 100 initialisiert. Der Code verwendet also eine Impulsvorwahl, die dafür sorgt, dass der statistische Fehler von vorneherein bekannt ist. Für maxcnt=100 beträgt der statistische Fehler Wurzel(100) = 10%. Wer ungeduldig ist, kann den Wert für maxcnt auf 10 setzen, dann werden die Ergebnisse schneller erzeugt, allerdings mit mehr Streuung, da es sich bei der berechneten Zählrate ja um eine Schätzung aus den zufällig eintreffenden Zähl-Impulsen handelt.

Der Ausgang OUT des Theremino Zählrohr-Treibermoduls ist mit dem Interrupt-Eingang 0 am Pin 2 des Arduino-Uno verbunden. Das bedeutet, dass immer dann, wenn ein digitaler Zählimpuls vom Treibermodul erzeugt wird, der Mikrocontroller seine normale Programmabarbeitung unterbricht und in eine Interrupt-Service-Routine verzweigt. Als Interrupt-Service-Routine, die für diesen Fall abgearbeitet wird, wird count() vereinbart. In der Interrupt-Service-Routine sollte der Mikrocontroller möglichst wenig zu tun haben um möglichst wenig zusätzliche Totzeit zu erzeugen. In diesem Beispiel wird nur die globale Zählimpuls-Zählervariable counter hochgezählt, die mit null initialisiert wurde.

In der Setup-Routine, die beim Programmstart nur einmal ausgeführt wird, werden typischerweise die die notwendigen Initialisierungen gemacht. Hier wird der serielle Monitor für eine Übertragung auf den Serial Monitor mit 9600 Zeichen pro Sekunde festgelegt. Danach wird vereinbart, dass für den Fall, dass ein Zählimpuls über den Interrupteingang 0 eintrifft, mit der fallenden Flanke in die Interrupt-Service-Routine count() verzweigt werden soll (das Zählrohr-Treibermodul erzeugt Zählimpulse von 1 nach nach 0).

Für die normale Programmausführung wird die Routine loop() verwendet, die wiederholt ausgeführt wird. Die meiste Zeit aber tut sich in der loop() Routine nichts, weil nämlich die Anweisung if (counter >= maxcnt) den Wert falsch ergibt und der Zählimpulszähler auf weitere Zählimpulse wartet. Erst wenn maxcnt Zählimpulse eingetroffen sind, dann beginnt eine Auswertung. Dazu wird zuerst mit millis() die aktuelle Prozessorzeit geholt und die Zeitdifferenz dt zur letzten Auswertung berechnet, wobei auf den

Wert der Prozessorzeit zugegriffen wurde, der am Ende von loop() in der globalen Variablen oldTime als alte Zeit abgespeichert wurde. Aus der Anzahl maxcnt an Zählimpulsen, und der dafür benötigten Zeitdifferenz dt, berechnet der Mikrocontroller nun die Zählrate im Format einer Kommazahl und speichert sie in die Variable mit dem Namen rate. Diese Zählrate überträgt er an den seriellen Monitor. Dann merkt er sich die Prozessorzeit als alte Zeit und setzt den Impulszähler für die nächste Auswertung auf 0 zurück, die nach weiterem Eintreffen von maxcnt Zählimpulsen beim Schleifendurchlauf der loop()-Routine durchgeführt wird. Man kann nun die laufend neu berechneten Zählraten aus wiederholten Messungen auf dem Serial Monitor beobachten. Die Zeit für eine Messung kann man in etwa aus der Empfindlichkeit des Zählrohrs mit 840cpm/(uSv/h) und der Anzahl, der Zählimpulse maxcnt abschätzen. Bei einer Ortsdosisleistung von 0.1uSv/h erzeugt das Zählrohr etwa 84 Zählimpulse pro Minute. Daher benötigt eine Messung dann in etwa 100Impulse/84cpm=1.1Minuten. Das heißt es müsste unter diesen Bedingungen alle 1.1Minuten eine Zählrate auf dem Bildschirm entstehen.

Dieser Sketch kann nun um die Ansteuerung des Grove I2C LCD-Display der Firma Seeedstudio und den Piezo- Lautsprecher erweitert werden:

```
#include <Wire.h>
#include "rgb_lcd.h"


volatile int counter = 0;
unsigned long oldTime = 0;
int maxcnt = 100;

rgb_lcd lcd;

void count()
{
  counter++;
  digitalWrite(7,HIGH);
  delay(5);
  digitalWrite(7,LOW);
}

void setup()
{
  pinMode(7, OUTPUT);
  digitalWrite(7,LOW);

  lcd.begin(16, 2);
  lcd.print(maxcnt);
  lcd.print("P ");
  lcd.print("cps:");
  lcd.setCursor(0, 1);
  lcd.print(counter);
  attachInterrupt(0, count, FALLING);
}

void loop()
{
  unsigned long time;
  unsigned long dt;
  float rate;
  if (counter >= maxcnt) {
    detachInterrupt(0);
    time = millis();
    dt = time-oldTime;
    rate = (float)maxcnt*1000.0/(float)dt;
```

```
    lcd.setCursor(0, 1);
    lcd.print("        ");
    lcd.setCursor(0, 1);
    lcd.print(rate);
    oldTime = time;
    counter = 0;
    attachInterrupt(0, count, FALLING);
  }
}
```

Listing 2: Erweiterung um die LCD-Anzeige und den Piezo-Lautsprecher

Für die Ansteuerung des LCD-Displays mit I2C Interface wird die Arduino-interne Bibliothek wire.h verwendet. Zusätzlich muss noch die LCD-Display Bibliothek des Herstellers heruntergeladen und in den Arduino Bibliotheks-Ordner kopiert werden. Beide Bibliotheken müssten mit jeweils einer #include Anweisung in den Sketch eingebunden werden.

Die count-Routine wird dahingehend verändert, dass nach dem Hochzählen der Zähler-Variablen der Piezo-Ausgang angeschaltet und nach 5 Millisekunden wieder abgeschaltet wird. Diese Zeit ist erforderlich um einen Knacks im Piezo-Lautsprecher hörbar zu machen. Die delay(5) Anweisung erzeugt allerdings eine zusätzliche 5 Millisekunden Totzeit, in der keine Zählimpulse gezählt werden können. Man sollte diese Anweisungen daher nur bei niedrigen Impulsraten verwenden. Ansonsten sollte man diese Zeilen für die akustische Signalisierung auskommentieren, insbesondere wenn man bei einer Messung auch mit höheren Zählraten rechnen muss und ein genaues Ergebnis erreichen will.

In der Setup-Routine muss für die Verwendung des Piezo-Lautsprechers der Digital-Pin 7, als Ausgangspin initialisiert werden. Danach erfolgt nun anstelle der Initialisierung des Serial Monitors die Initialisierung des LCD-Displays. Dazu wird mit der lcd.begin() Methode das Display initialisiert und der Wert von maxcnt und dem Zeichen ,,P" für Pulse ausgegeben, gefolgt von dem Text cps: . Damit weiß man, dass nun das Ergebnis einer Zählraten-Berechnung aus maxcnt Zählimpulsen folgt. Das Display zählt seine Zeilen und Zeichen immer beginnend bei 0. „lcd.setCursor(0, 1);“ setzt daher den Schreibcursor auf das erste Zeichen in der zweiten Zeile. Danach wird in der Setup-Routine nur der Impuls-Zählerstand ausgegeben, der beim ersten Durchlauf in der Setup-Routine mit Null initialisiert wurde.

Eine weitere Änderung ist, dass der if-Block if (counter >= maxcnt) in der Loop-Routine jetzt mit der Anweisung „detachInterrupt(0);“ beginnt. Das bedeutet, dass das Interrupt-Handling ausgesetzt wird, bis der If-Block mit attachInterrupt(0, count, FALLING); } endet, und das Interrupt-Handling erneut aktiviert wird. Hintergrund ist, dass ja auch während eine Berechnung in der Loop-Routine gemacht wird, neue Zählimpulse eintreffen können, die je nach Prozessortyp in einer Warteschleife abgespeichert und nach dem Abarbeiten der Interrupt-Service-Routine einen erneuten Interrupt auslösen können, obwohl die Zeitmessung bereits beendet ist. Daher ist es gute Programmier-Praxis, das Interrupt-handling in der Auswerte-Routine, solange keine Zeitmessung mehr läuft, zu deaktivieren, unabhängig davon wie viele Interrupts vom Prozessor in einer Warteschlange tatsächlich gespeichert werden können.

In der Loop-Routine wird jetzt die Ausgabe auf den Serial Monitor durch eine Ausgabe auf das LCD-Display ersetzt. Der Inhalt der ersten Zeile des Displays bleibt in der Loop-Routine jedoch unangetastet. Allerdings muss der vorige Wert der Zählrate in der zweiten Zeile des Displays zuerst gelöscht werden, bevor ein neuer Wert geschrieben werden kann, weil die String-Länge je nach Anzahl der Vorkomma-Stellen variieren kann. Die Ausgabe von 8 Leerzeichen löscht daher den vorigen Anzeigewert in dieser Zeile. Dann wird der Cursor wieder auf den Beginn der zweiten Zeile gesetzt und der aktuelle Zählratenwert wird ausgegeben.

Mit dieser Software hat man bereits ein mobiles Messgerät, wenn man den Arduino von einer externen Power-Bank (5V über USB) oder einem Modellbau-Akku her versorgt (11.1V über Stromversorgungsbuchse). Das Messgerät spiegelt auch die Genauigkeit des Zählrohrs wider, sofern man eine entsprechende Kalibrierung durchführt. Allerdings sind das Ablesen und Aufschreiben von wiederholten Messungen doch noch etwas mühsam, so dass sich eine weitere Erweiterung um eine Datalogging-Funktionalität mit einer Daten-Speicherung auf SD-Karte anbietet, die dann erst nach der Messung am PC ausgewertet wird. Dafür gibt es von der Firma Adafruit ein Datalogging-Shield, das auf den Arduino Uno aufgesteckt werden kann und welches die Speicherung der Messdaten auf SD-Karte erlaubt. Gleichzeitig verfügt dieses Datalogging-Shield über eine integrierte Echtzeit-Uhr (Real Time Clock, RTC), so dass die gemessenen Zählraten-Daten zusammenmit dem Datum und der aktuellen Uhrzeit abgespeichert werden können und später direkt in ein Tabellenkalkulationsprogramm eingelesenen werden können.

Die Erweiterung um die Echtzeit-Uhr und um die Datalogging Funktion führt schließlich zum Konzept für die OM01 Gamma-ODL Messsonde, einem Messgerät, das ähnlich wie die BfS ODL-Sonde Messdaten aufzeichnen kann. Allerdings erfolgt die Aufzeichnung lokal auf die SD-Karte, während die BfS ODL-Sonden ihre Daten live zu einem Server übertragen, welcher die Daten in einer Datenbank auf einer Festplatte des Servers in einem Data Center für das Post-Processing durch das BfS abspeichert.

Die Anordnung der Komponenten für die OM01-Sonde sieht ganz ähnlich aus, wie bei der Grundschaltung, nur dass jetzt die Komponenten an das Datalogging Shield angeschlossen werden müssen. Die Anschluss-Positionen sind aber dieselben wie bei der Grundschaltung am Arduino, da das auf den Arduino Uno aufgesteckte Datalogging-Shield die Arduino Pins direkt nach oben durchreicht. Für den Betrieb des Datalogging Shield ist neben der SD-Karte noch eine Lithium Stützbatterie mit 3V vom Typ CR1220 nötig, die in den Batteriehalter eingeschoben wird. Sie verhindert, dass die Echtzeit-Uhr die Uhrzeit beim Ausschalten wieder vergisst. Daher muss man die Uhrzeit dann nur einmal nach dem Einsetzen stellen und erst nach längerer Zeit wieder Nachjustieren, wenn die Uhr nicht so ganz präzise ist, wie beispielsweise eine gut getrimmte Quarzuhr.

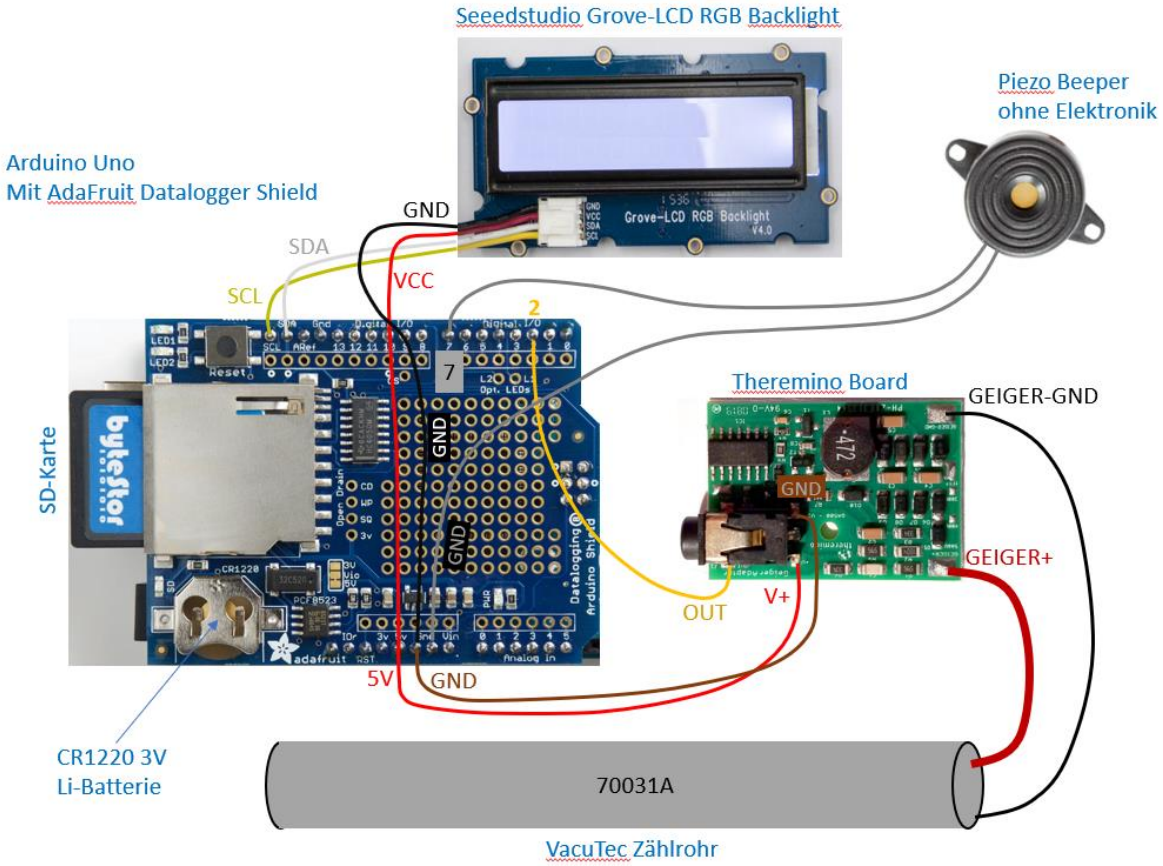

Abb. 6: Hardware-Verschaltung der Komponenten für die Gamma-ODL Sonde OM01

Es gibt nun mehrere Möglichkeiten die Komponenten der OM01 Gamma-ODL Sonde mechanisch anzuordnen. In jedem Fall aber sollte beachtet werden, dass ein Gehäuse die Eigenschaften des Zählrohrs durch eine entsprechende Dämpfung leicht beeinträchtigen kann. Daher sollte die Sonde, wenn es geht, ohne jedes Gehäuse aufgebaut werden. Eine Möglichkeit ist es das Zählrohr mit Kabelbindern und einer elastischen Schaumstoff-Unterlage auf einer dünnen Sperrholz-Platte zu fixieren. Die Schaumstoff-Unterlage dient als Schutz gegen das Verdrehen und Verschieben des Zählrohrs und hält es nach dem Festzurren der Kabelbinder auf Grund seiner Elastizität sicher fest. Die Sperrholz-Grundplatte kann so weit ausgedehnt werden, dass auch das Zählrohrtreiber-Modul, das LCD-Display und der Arduino ebenfalls befestigt werden können. Um die Sperrholz-Grundplatte von der Rückseite her zusätzlich zu versteifen, sollte noch ein dünnes L-Profil mit etwa 1cm Kantenlänge auf das Sperrholz aufgeleimt werden.

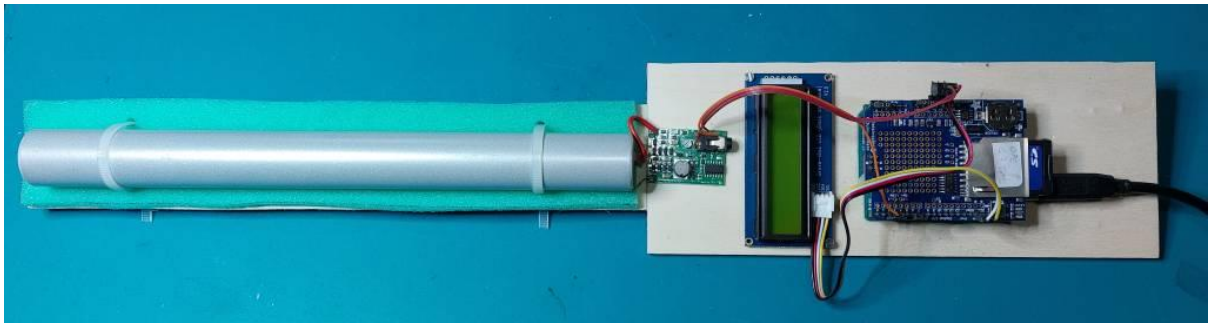

Abb. 7: Befestigung der Komponenten der OM01-Sonde auf der Sperrholz-Grundplatte

Die Grundplatte kann schließlich am unteren Ende an einem Fotostativ befestigt werden, das für einen sicheren Stand in 1 Meter Höhe über dem Boden sorgt (Standardhöhe für Gamma-ODL Messungen).

Die Stromversorgung erfolgt zweckmäßigerweise über eine USB-Power-Bank.

Abb. 8: Befestigung der Grundplatte derOM01-Sonde auf Gonadenhöhe an einem Fotostativ

Der Arduino-Code für die OM01-Sonde kann dem Folgenden Listing 3 entnommen werden.:

```
#define MAXCNT 100
#define PIEZO 7
#define DEBUG 0

#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include "RTClib.h"
#include "rgb_lcd.h"


volatile int counter = 0;
unsigned long oldTime = 0;
//RTC_DS1307 rtc;
RTC_PCF8523 rtc;
char fileName[15] = "datalog.txt";
File myFile;

rgb_lcd lcd;

void count()
{
  counter++;
  /*
  digitalWrite(PIEZO,HIGH);
  delay(5);
  digitalWrite(PIEZO,LOW);
  */
}

void setup()
{
  pinMode(PIEZO, OUTPUT);
  digitalWrite(PIEZO,LOW);

  lcd.begin(16, 2);
  lcd.print(MAXCNT);
  lcd.print("P ");
  lcd.print("cps:");
  lcd.setCursor(0, 1);
  lcd.print(counter);

  Wire.begin();
  pinMode(10, OUTPUT); // SD Card CS
  Serial.begin(9600);


  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }
  //if (! rtc.isrunning()) {    //DS1307
  if (! rtc.initialized()) {
    Serial.println("RTC is NOT running!");
  }
  //rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

  if (!SD.begin(10)) {
    Serial.println("SDcard not ready");
  }
  else
```

```
    Serial.println("SDcard ok");

  if (!SD.exists(fileName)) {
    myFile = SD.open(fileName, FILE_WRITE);
    myFile.println("###");
    myFile.flush();
  }
  else {
    myFile = SD.open(fileName, FILE_WRITE);
    myFile.println("-----");
    myFile.flush();
  }
  Serial.println("Logging started ...");

  attachInterrupt(0, count, FALLING);
}

void loop()
{
  unsigned long time;
  unsigned long dt;
  float rate;
  if (counter >= MAXCNT) {
    detachInterrupt(0);
    time = millis();
    dt = time-oldTime;
    rate = (float)MAXCNT*1000.0/(float)dt;
    lcd.setCursor(0, 1);
    lcd.print("        ");
    lcd.setCursor(0, 1);
    lcd.print(rate);
    oldTime = time;
    counter = 0;

    //logging
    DateTime now = rtc.now();

    if(DEBUG) {
      Serial.print(now.day());
      Serial.print('.');
      Serial.print(now.month());
      Serial.print('.');
      Serial.print(now.year());
      Serial.print(' ');
      Serial.print(now.hour());
      Serial.print(':');
      Serial.print(now.minute());
      Serial.print(':');
      Serial.print(now.second());
      Serial.print(' ');
      Serial.print(rate);
      Serial.println();
  }

    myFile.print(now.day(), DEC);
    myFile.print('.');
    myFile.print(now.month(), DEC);
    myFile.print('.');
    myFile.print(now.year(), DEC);
    myFile.print(' ');
    myFile.print(now.hour(), DEC);
    myFile.print(':');
    myFile.print(now.minute(), DEC);
```

```
    myFile.print(':');
    myFile.print(now.second(), DEC);
    myFile.print('\t');
    myFile.print(rate);

    myFile.println();
    myFile.flush();
    attachInterrupt(0, count, FALLING);
  }
}
```

Listing 3: Der vollständige Arduino Code für die OM1-Sonde

Der Code beinhaltet gegenüber dem vorigen Listing 2 im Wesentlichen die Erweiterung um die Speicherung der Daten auf dem Adafruit Datalogging-Shield. Es wurden aber noch einige Kleinigkeiten zusätzlich geändert. MAXCNT wird hier per Compileranweisung #define vereinbart und nicht in einer Variablen gespeichert. Das bedeutet, dass Compiler jedes Auftreten von MAXCNT im Code durch 100 ersetzen wird, was wieder die Anzahl der Impulse auf die gewartet wird darstellt, bevor eine Zählimpulsrate berechnet wird. Diese Compileranweisung verbraucht im Gegensatz zur Variablendeklaration keinen Speicherplatz. Wenn man will, kann man wieder einen Piezo-Lautsprecher auf Pin 7 als Ticker anschließen, das ist allerdings später im Code dann auskommentiert, um die zusätzliche Totzeit zu vermeiden. Diese Zeilen sollte man nur fürs Debugging aktivieren, sonst wird man dadurch bei hohen Zählraten Zählimpulse verlieren und die Messung wird ungenau. Die DEBUG Anweisung steht auf Null und verhindert, dass Ergebnisse auch auf einen Computermonitor geschrieben werden, weil das ebenfalls Zeit kostet. Da das im Debugging-Fall aber nur während der Auswertung gemacht werden würde, wäre es für die Qualität der Messung keine Einschränkung. Setzt man DEBUG auf 1, kann man auch im Serial Monitor der Arduino IDE mitlesen, was auf die SD-Karte geschrieben wird.

Danach folgen die Anweisungen zum Einbinden der Bibliotheken. Wire ist die Arduino-Bibliothek für den I2C-Bus, der vom Display verwendet wird, SPI und SD sind auch Arduino-Bibliotheken für die SD-Karte, die über einen SPI-Bus mit dem Prozessor kommuniziert. RTClib stammt von Adafruit und ist die Bibliothek für die Realtime-Clock. Die Bibliothek rgb_lcd schließlich stammt von Seedstudio und ist der Softwaretreiber fürs LCD-Display. RTClib und rgb_lcd muss man sich wie bei der Grundschaltung bei den Herstellern herunterladen und explizit im Arduino Bibliotheken-Verzeichnis stehen haben, damit der Compiler sie dort auch finden kann.

Als globale Variable wird hier wieder die Variable counter vereinbart, die als Zähler für die Zählimpulse dient. Die Variable oldTime speichert die Zeit nach einer Auswertung, damit die Zählimpulsrate aus der Zeitdifferenz zwischen der neuen Zeit und der gespeicherten alten Zeit berechnet werden kann. Die Variable RTC_PCF8523 wird für die Echtzeit-Uhr benötigt und ist vom Datentyp rtc, das ist von der Real-Time Bibliothek so vorgegeben. Ältere Shields verwenden statt des PCF8523 ICs den IC vom Typ DS1307 von Dallas, deswegen sind die entsprechenden Anweisungen noch als Kommentar enthalten. Beim Schreiben des Datalog-Files auf die SD Karten wird stets der Name ,,datalog.txt" für den Logfile verwendet, das ist hier als Character-Feld mit 15 Zeichen vereinbart. Die Datei selbst verwendet myFile als Variable vom Datentyp file.

Die Count-Routine ist wieder die Routine, die am zeitkritischsten ist. Bei höheren Zählimpulsraten sollte hier nur der Zähler für die Zählimpulse mit counter++ erhöht werden. Sonst sollte in dieser Routine nichts passieren, was Zeit kostet. Jede weitere Instruktion erzeugt eine zusätzliche Totzeit. Nun taktet der Prozessor beim Uno zwar mit 16 MHz, aber im auskommentierten Code für den Piezo Click

steht beispielsweise delay(5) als Wartezeit-Anweisung, was dann eine Wartezeit von 5ms erzwingt um die Membran des Piezo-Tickers nicht zu schnell zu bewegen, sonst hört man ihn nämlich nicht. Und in 5ms, da können bei hoher Zählrate durchaus weitere Zählimpulse eintreffen. Wenn man also ein Ticken auch bei höheren Dosisleistungen noch ohne Einfluss auf das Messergebnis erzeugen will, sollte man das besser mit Hardware tun und nicht mit Software in der Zählroutine. Der Code für den Piezo-Lautsprecher ist hier nur für eine Debugging-Option als Kommentar enthalten. Man wird das bei Zählraten unter 100cps im Messergebnis kaum merken, darüber aber schon.

In der Setup-Routine, die bei Programmstart nur einmal ausgeführt wird, werden nun alle Initialisierungen gemacht, auch solche, die notwendig sind um das Datalogging und die Echtzeituhr zu verwenden. Zuerst wird der IO-Pin für den Piezo-Lautsprecher auf Output geschaltet und auf 0V gelegt, sonst ist dieser per Default ein Input. Dann wird mit der lcd.begin() Methode das Display initialisiert und der Wert von MAXCNT und dem Zeichen ,,P" als Akronym für „Pulse“ ausgegeben, gefolgt von dem Text „cps: „. Damit weiß man, dass nun eine Zählrate folgt. Bei dem Code für die OM01-Sonde wird auch wieder der vorige Zählraten-Wert auf dem LCD-Display durch die Ausgabe von Leerzeichen gelöscht.

Dann wird die SD-Karte initialisiert, die auf dem PCB des Adafruit Shield am Arduino Pin 10 angeschlossen ist. Falls man etwas auf dem Bildschirm ausgeben will, wird mit Serial.begin(9600) die Arduino UART aktiviert, welche die ASCII-Zeichen mit 9600Zeichen/s über den COM-Port des Computers zum Serial Monitor überträgt. So kann man sich die Messdaten auch mit dem Serial Monitor in der Arduino IDE anschauen, aber auch mit jedem anderen ASCII Terminal Programm.

Danach wird die Echtzeit-Uhr initialisiert und auf die Computer-Uhr synchronisiert, aber nur, wenn man die Zeile:

rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

nicht auskommentiert hat. Das Kommentar muss man einmal entfernen um die Uhr zu stellen und dann nochmal in kommentierter Form übersetzen, damit der Prozessor von da ab die Uhrendaten der Echtzeit-Uhr verwendet. Auch hier ist der Code für den älteren DS1307 Chip noch als Kommentar im Code enthalten.

Danach wird die SD-Karte initialisiert. Dabei wird zuerst abgefragt, ob die SD-Karte eingesetzt ist, oder ob sie sonst irgendwelche Probleme hat, wie beispielsweise, dass der Schreibschutz an der SD-Karte aktiviert ist. Erst dann wird mit SD.open() der File-Identifier zum Schreiben auf datalog.txt erzeugt und drei # in die allererste Zeile geschrieben, damit man später sieht, dass der Beginn des Logvorgangs auch auf der Karte den Beginn der Datenaufzeichnung darstellt. Falls es schon einen datalog.txt File gibt, wird ,,-----,, als Trennzeile in den Logfile geschrieben. Daran erkennt man später, dass ab dieser Position in den Daten eine neue Messdatensequenz geloggt wurde. Alte Daten werden nicht weggelöscht, sondern es werden neue Daten an einen bestehenden Logfile angehängt. Löschen muss man den Datalog-File auf der SD-Karte mit dem Computer, nach einer Daten-Sicherung mit eventuellem Post-Processing.

Nachdem man Daten auf die SD-Karte geschrieben hat, muss man den Schreibpuffer auch immer wieder leeren, damit jedes Mal alle Daten auf die Karte übertragen werden. Dies wird mit der Anweisung myFile.flush() erreicht.

Schließlich wird noch das Interrupt-Handling für den Interrupt-Eingang 0 mit der fallenden Flanke der Zählimpulse scharf geschaltet. Ab hier löst dann ein am Pin 2 eintreffender Zählimpuls wieder Interrupt und damit das Pausieren des normalen Programmablaufs aus (loop-Routine) und verzweigt temporär

in die count() Routine um den Zählimpuls-Zähler hoch zu zählen und um danach wieder mit der normalen Programmausführung fortzufahren.

Nach Ausführen der Setup-Routine wird die Routine loop() periodisch wiederholt, bis ein Reset eintrifft oder der Versorgungsstrom abgestellt wird. Loop() stellt daher den normalen Programmablauf dar, solange kein Interrupt ausgelöst wird.

Auch bei dieser Code Variante passiert in der loop() Routine nicht viel, weil wieder die Anweisung if (counter >= MAXCNT) die meiste Zeit über den Wert „falsch“ ergibt und der Zählimpulszähler auf weitere Zählimpulse wartet. Erst wenn MAXCNT Zählimpulse eingetroffen sind, beginnt wieder die Auswertung. Für einen Auswertung wird zunächst auch wieder das Interrupt-Handling abgeschaltet, damit die Warteschlange für Interrupts nicht überlaufen kann. Dann holt sich der Mikrocontroller mit millis() die aktuelle Prozessorzeit und berechnet die Zeitdifferenz zur letzten Auswertung, wobei er sich am Ende von loop() die Prozessorzeit als alte Zeit gemerkt hat. Aus der Anzahl MAXCNT an Zählimpulsen, für die er das Zeitintervall vermessen hat, und der Zeitdifferenz dt dafür, berechnet er nun wieder die Zählrate als Kommazahl. Diese Zählrate schreibt er aufs Display, wobei er dort vorher alte Daten durch Leerzeichen ersetzt, um die Zeile zu löschen. Dann merkt er sich die Prozessorzeit als alte Zeit und setzt den Impulszähler auf 0 zurück für die nächste Auswertung in einem erneuten Schleifendurchlauf der loop()-Routine. Danach erfolgt nun die Ausgabe von Datum und Zeit aus der Real-Time Clock und der Zählrate auf die SD-Karte und optional auf den Serial Monitor. Zum Schluss wird das Interrupt-Handling wieder scharf geschaltet und eine neue Messung kann mit erneuten Zähl-Durchläufen in der Loop-Routine beginnen.

Die Umrechnung der auf der SD-Karte gespeicherten Zählraten-Daten macht man zweckmäßigerweise in einem Post-Processing Schritt auf dem PC mit Hilfe von Kalibierdaten. Ein Kalibierkonzept für die OM01-Sonde wird in /6/ vorgestellt.

Abb. 9a, b: Einsatz der OM01-Sonde in Wittichen, Schwarzwald a) auf einem Waldweg, der über die ehemalige Halde der Grube Sophia führt (Zählrate 747cpm) und b) Messung mit der OM01 Sonde auf der Schmiedestollenhalde (Zählrate 1344cpm)

Zum Test der Zähl- und Auswerteelektronik der OM01-Sonde empfiehlt es sich, einen auf einer zufälligen Poisson-Statistik beruhenden Zählimpulsgenerators zu verwenden. Dies ist in /5/ beschrieben.

Literatur

/1/ Metrology for mobile detection of ionising radiation following a nuclear or radiological incident Short Name: Preparedness, Project Number: 16ENV04
https://www.euramet.org/research-innovation/search-research-projects/details/project/metrology-for-mobile-detection-of-ionising-radiation-following-a-nuclear-or-radiological-incident

/2/ Viacheslav Morosh et al.; Investigation into the performance of dose rate measurement instruments used in non-governmental networks
https://www.sciencedirect.com/science/article/pii/S1350448721000950?via%3Dihub

/3/ VacuTec Geiger-Müller Zählrohre
https://www.vacutec-gmbh.de/index.php?id=39&L=18&L=18

/4/ Theremino: “The geigeradapter” - adapter between Geiger tubes and PIN standard
https://www.theremino.com/en/technical/schematics

/5/ Bernd Laquai; Ein Zählimpulsgenerator mit Poisson-Statistik zur Prüfung der Auswerteelektronik von Geigerzählern und Szintillationszählern
http://opengeiger.de/PoissonTestgen.pdf

/6/ Messung der Umgebungs-Äquivalentdosisleistung H*(10) am Gamma-ODL Referenzpunkt in der Kapelle im Höhenpark Killesberg in Stuttgart
http://www.opengeiger.de/MessungHstern10KapelleKillesberg.pdf

/7/ Bernd Laquai; Gamma-ODL Referenzpunkt „Kapelle im Höhenpark Killesberg in Stuttgart"
http://opengeiger.de/RefpunktKapelleKillesberg.pdf

/8/ U. Stöhlker et al.; The German Dose Rate Monitoring Network and Implemented Data Harmonization, Radiation Protection Dosimetry (2019), Vol. 183, No. 4, pp. 405–417, doi:10.1093/rpd/ncy154