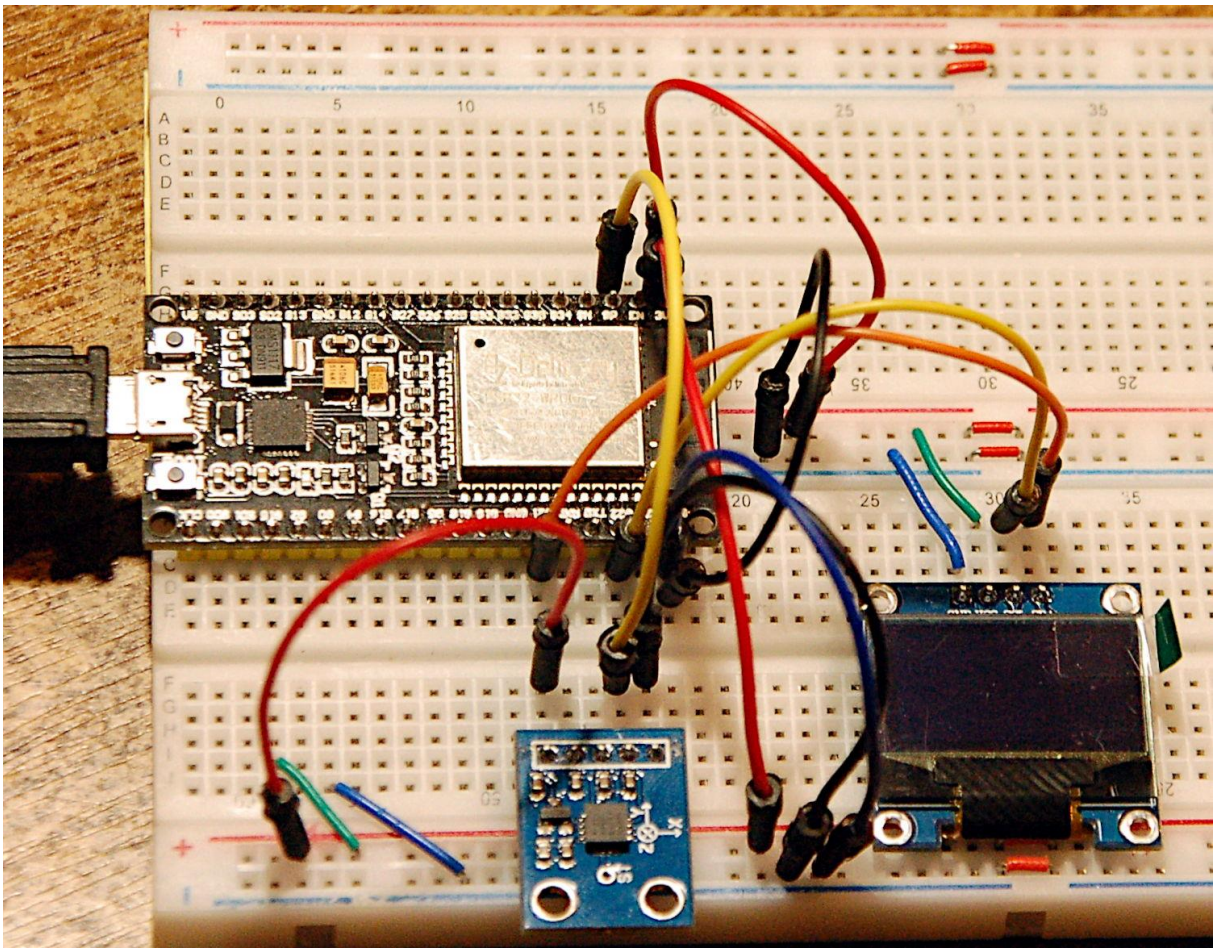

Erdbeben-Sensor - Aufbau

Dieser Beitrag ist für Anfänger gut geeignet. Es gibt ihn auch als PDF-Dokument.

Um Erschütterungen, oder besser Schwingungen, des Untergrundes zu erfassen gibt es zwei Arten von Sensoren. Die einfachere Art ist ein Rüttelkontakt. Das kann eine kleine Kugel in einem Röhrchen sein, die durch einen Stoß eine Verbindung zwischen zwei Kontakten herstellt oder öffnet. Eine andere Bauart verwendet eine kleine Blattfeder an deren Ende eine Schwungmasse sitzt. Erfährt das Bauteil einen Stoß, dann gerät die Feder in Schwingungen und schließt den Kontakt zu einer zweiten Feder oder dem Gehäuse. Solche Rüttel-Sensoren können nur "an" und "aus", sie kennen keine Zwischentöne und sagen nichts über die Größenordnung der Stöße aus.

Die andere Art von Sensor verwende ich hier. Es ist ein Beschleunigungssensor oder Accelerometer. Dessen Funktion ist vergleichbar mit dem eben beschriebenen Federkontakt. Nur bestehen hier die Federn aus hauchdünnen Piezo-Plättchen. Werden die durch die Schwerkraft oder durch Erschütterungen verbogen, dann geben sie eine Spannung ab, die zur Verbiegungsamplitude, also auch zur wirkenden Kraft proportional ist. Damit kann die Heftigkeit eines Stoßes festgestellt werden.

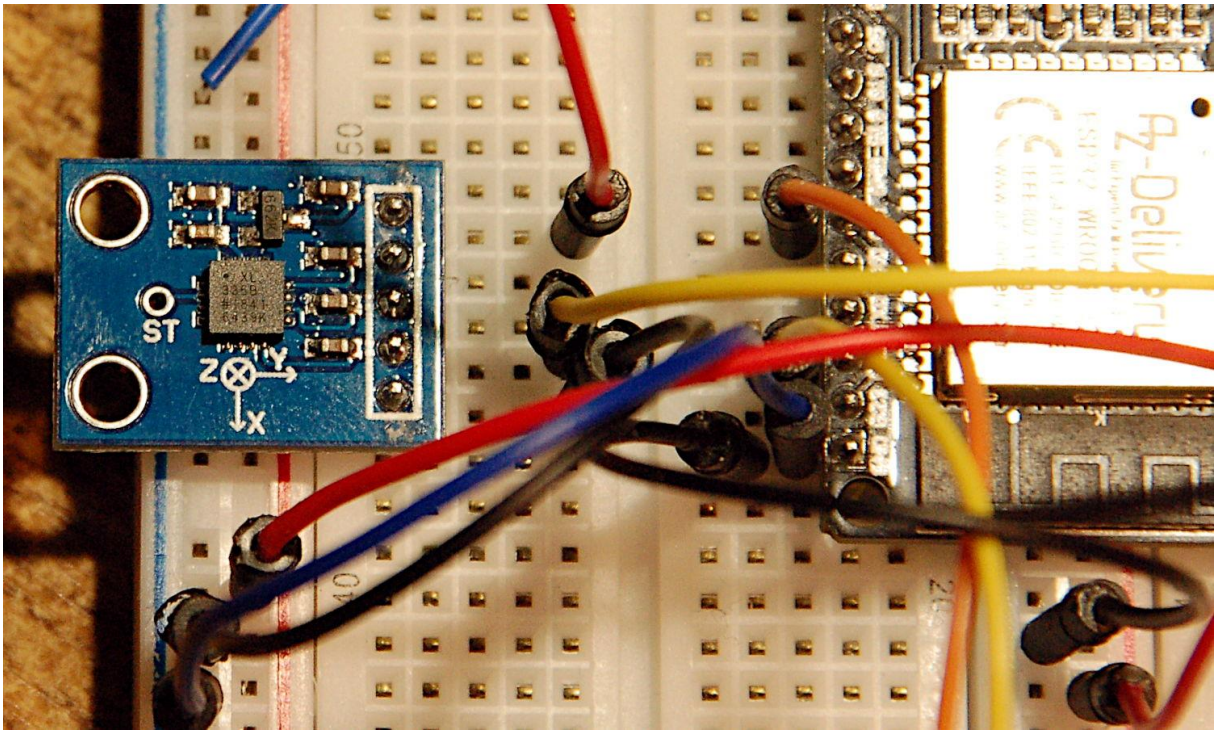

Abbildung 1: XL335B - 3-Achsen-Accelerometer mit analogen Ausgängen

Aber auch in diesem Fall gibt es zwei Typen von Sensoren, die sich vor allem in der Art der Datenübertragung zum Controller unterscheiden. Die einen werden über den I2C-Bus angesteuert und abgefragt, die anderen geben die Spannung von den Piezoplättchen, natürlich über einen integrierten Verstärker, an Analogausgängen ab. So ein Bauteil ist der ADXL335, der hier in dem BOB (Break Out Board) GY-61 zusammen mit einem Spannungsregler verbaut ist. Ich werde im Folgenden beschreiben, wie man aus dem GY-61, einem OLED-Display und einem ESP32 ein Seismometer oder einen Seismographen bauen kann. Außerdem zeige ich eine weitere Möglichkeit, wie man dauerhaft Daten im Flash des ESP32 ablegen und abrufen kann und es gibt eine Darstellung der Funktionsweise eines Accelerometers. Folgen Sie mir auf eine neue Tour zum Thema

MicroPython auf dem ESP32 und ESP8266

heute

Bodenerschütterungen messen und darstellen

Hardware

1	ESP32 Dev Kit C unverlötet oder ESP32 NodeMCU Module WLAN WiFi Development Board oder NodeMCU-ESP-32S-Kit
1	GY-61 ADXL335 Beschleunigungssensor 3-Axis Neigungswinkel Modul
1	0,96 Zoll OLED SSD1306 Display I2C 128 x 64 Pixel
1	Breadboard Kit - 3x Jumper Wire m2m/f2m/f2f + 3er Set MB102 Breadbord kompatibel mit Arduino und Raspberry Pi - 1x Set
1	Mehrgang rotary Potentiometer mit Schutzwiderstand 3590S 10K Ohm
1	LED
1	Widerstand 270 Ohm
diverse	Jumperkabel
	Digital-Voltmeter (DVM) für die Kalibrierung

Der Aufbau gestaltet sich sehr einfach, das OLED-Display wird am I2C-Bus angeschlossen. Den Chefposten übernimmt ein ESP32. Für dessen Einsatz bedarf es zweier Breadboards, die mit einer Stromschiene in der Mitte verbunden werden, damit man mit den Abständen der Pinreihen hinkommt und am Rand auch noch Kabel gesteckt werden können.

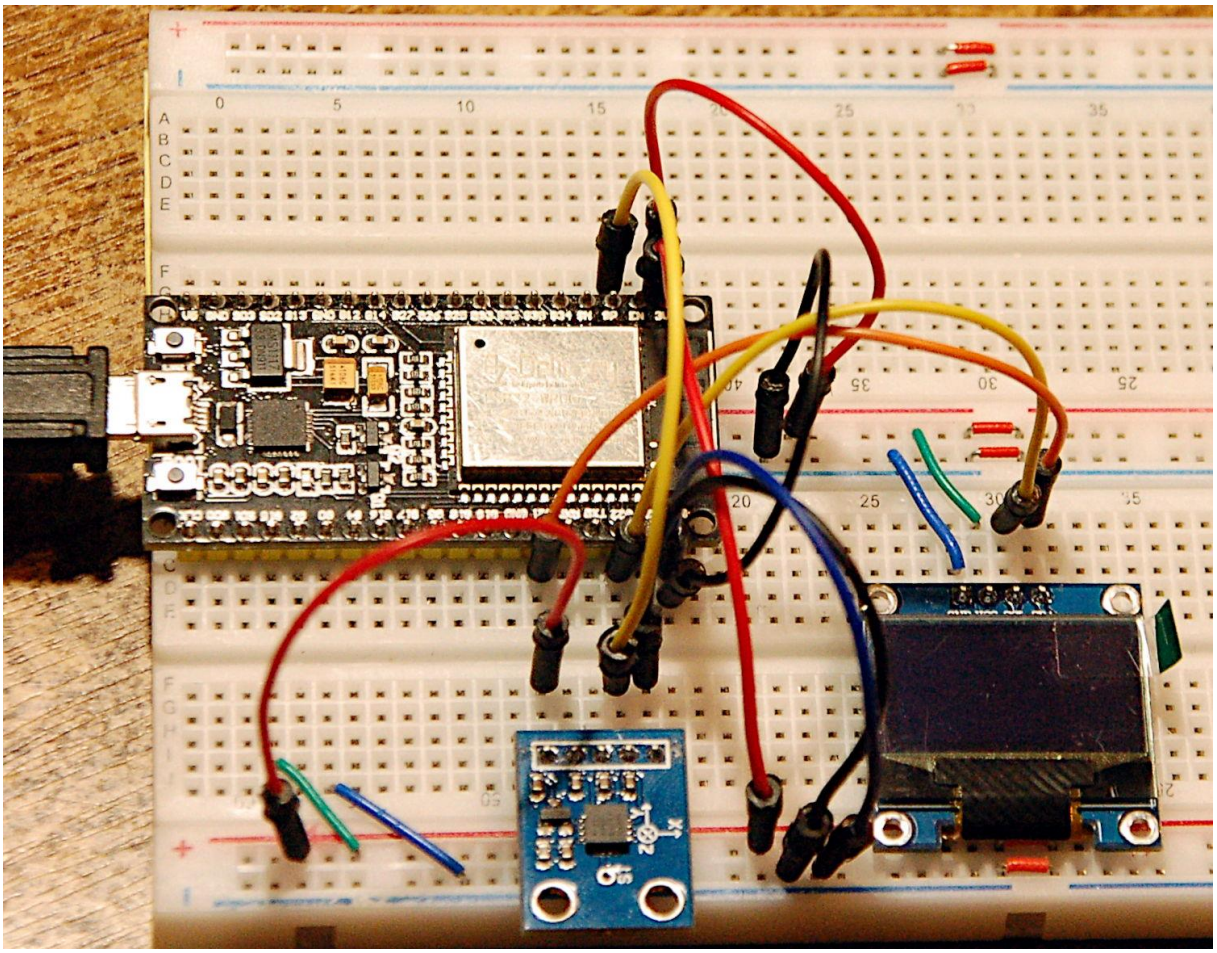

Abbildung 2: Erdbeben-Sensor - Aufbau

Für den ESP32 habe ich mich deshalb entschieden, weil der mit einem 12-Bit ADC aufwarten kann und darüber hinaus eine Auswahl des Spannungsbereichs bietet. Für den weiteren Ausbau des Projekts kommt uns die reichliche Auswahl an GPIO-Pins

zu Gute. Grundsätzlich wäre für den Umfang dieses Projekts auch ein ESP8266 geeignet. Das Display hat eine Größe von 128x64 Pixeln und dient unter anderem zur grafischen Darstellung der Messwerte.

Die Software

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft

Verwendete Firmware für den ESP32:

MicropythonFirmware
v1.19.1 (2022-06-18) .bin

Die MicroPython-Programme zum Projekt:

ssd1306.py Hardwaretreiber für das OLED-Display
oled.py API für OLED-Displays
earthquake.py Betriebsprogramm

MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine ausführliche Anleitung (english version). Darin gibt es auch eine Beschreibung, wie die Micropython-Firmware (Stand 05.02.2022) auf den ESP-Chip gebrannt wird.

MicroPython ist eine Interpretersprache. Der Hauptunterschied zur Arduino-IDE, wo Sie stets und ausschließlich ganze Programme flashen, ist der, dass Sie die MicroPython-Firmware nur einmal zu Beginn auf den ESP32 flashen müssen, damit der Controller MicroPython-Anweisungen versteht. Sie können dazu Thonny, µPyCraft oder esptool.py benutzen. Für Thonny habe ich den Vorgang hier beschrieben.

Sobald die Firmware geflasht ist, können Sie sich zwanglos mit Ihrem Controller im Zwiegespräch unterhalten, einzelne Befehle testen und sofort die Antwort sehen, ohne vorher ein ganzes Programm kompilieren und übertragen zu müssen. Genau das stört mich nämlich an der Arduino-IDE. Man spart einfach enorm Zeit, wenn man einfache Tests der Syntax und der Hardware bis hin zum Ausprobieren und Verfeinern von Funktionen und ganzen Programmteilen über die Kommandozeile vorab prüfen kann, bevor man ein Programm daraus strickt. Zu diesem Zweck erstelle ich auch gerne immer wieder kleine Testprogramme. Als eine Art Makro fassen sie wiederkehrende Befehle zusammen. Aus solchen Programmfragmenten entwickeln sich dann mitunter ganze Anwendungen.

Autostart

Soll das Programm autonom mit dem Einschalten des Controllers starten, kopieren Sie den Programmtext in eine neu angelegte Blankodatei. Speichern Sie diese Datei unter boot.py im Workspace ab und laden Sie sie zum ESP-Chip hoch. Beim nächsten Reset oder Einschalten startet das Programm automatisch.

Programme testen

Manuell werden Programme aus dem aktuellen Editorfenster in der Thonny-IDE über die Taste F5 gestartet. Das geht schneller als der Mausklick auf den Startbutton, oder über das Menü **Run**. Lediglich die im Programm verwendeten Module müssen sich im Flash des ESP32 befinden.

Zwischendurch doch mal wieder Arduino-IDE?

Sollten Sie den Controller später wieder zusammen mit der Arduino-IDE verwenden wollen, flashen Sie das Programm einfach in gewohnter Weise. Allerdings hat der ESP32/ESP8266 dann vergessen, dass er jemals MicroPython gesprochen hat. Umgekehrt kann jeder Espressif-Chip, der ein kompiliertes Programm aus der Arduino-IDE oder die AT-Firmware oder LUA oder … enthält, problemlos mit der MicroPython-Firmware versehen werden. Der Vorgang ist immer so, wie hier beschrieben.

Wirkungsweise des ADXL335

Zur Erfassung betrachte ich erst einmal nur die vertikale Richtung. Beim Sensor ist das die z-Achse. Sie zeigt nach oben, wenn man den Sensor flach auf den Tisch legt. Das entsprechende Piezoelement wird dann durch die Gewichtskraft F_g auf die Masse nach unten verbogen. Drehe ich das BOB in die vertikale Ausrichtung, senkrecht zur Tischfläche, findet keine Biegung statt, weil die Schwerkraft in Richtung des Plättchens wirkt. Drehe ich das BOB um weitere 90°, so dass es auf dem Kopf steht, dann wird das Plättchen in die Gegenrichtung gebogen. Der Messverstärker stellt für die drei Fälle am Ausgang Spannungen ein, die der Ungleichungskette in Abbildung 3 entsprechen.

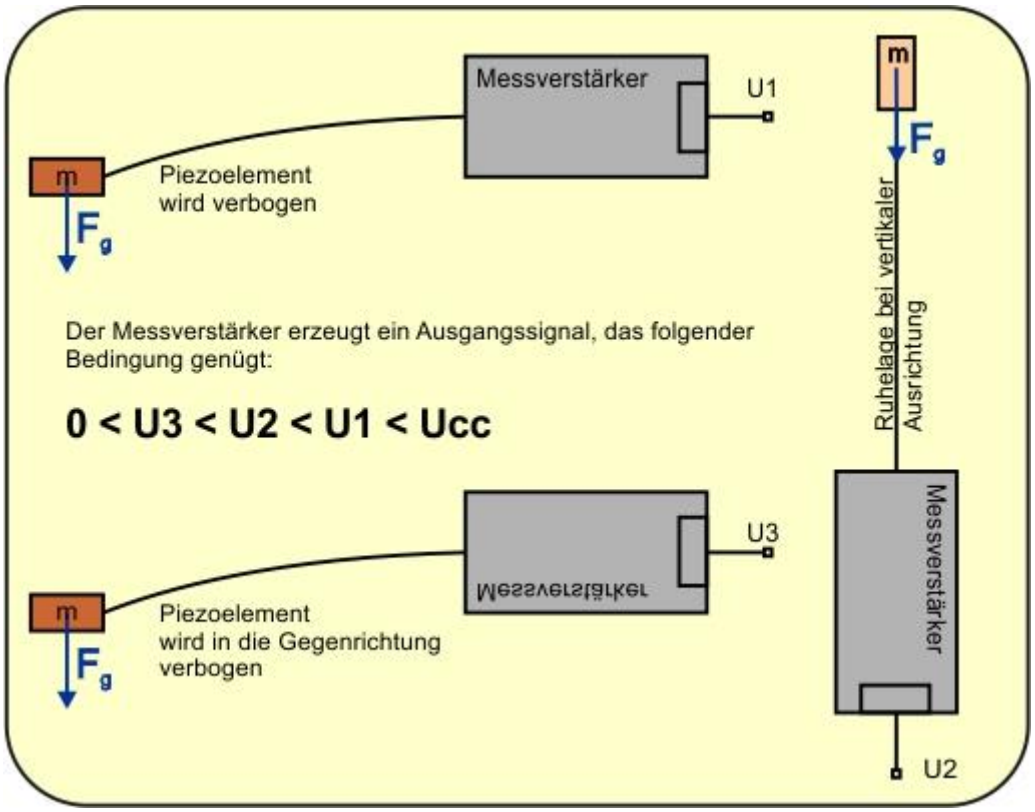

Abbildung 3: Arbeitsprinzip eines Piezoelements im ADXL335

Bleiben wir bei der ersten Ausrichtung parallel zur Tischfläche. Die Ausgangsspannung U1 ist für unser Vorhaben die Bezugsgröße. Der ADC des ESP32 liefert in diesem Fall einen Wert von um die 2368 counts. Klopfe ich jetzt mit dem Finger auf die Tischplatte, dann versetze ich sie in Schwingungen, sie wird sich schnell hintereinander etwas heben und senken. Sie können sich das veranschaulichen, wenn Sie das eine Ende eines Lineals mit der einen Hand am Tisch festklemmen und mit der anderen Hand das überstehende Ende kurz antippen. Das ist übrigens auch gleich ein Modell für die Funktionsweise der Piezoelemente.

Jetzt kommt eine weitere Sache ins Spiel, die Trägheit der Masse. Das ist die Eigenschaft von Körpern, sich einer Bewegungszustandsänderung zu widersetzen. Daraus resultieren Kräfte F_t, deren Richtung der Richtung der Bewegung genau entgegengesetzt ist.

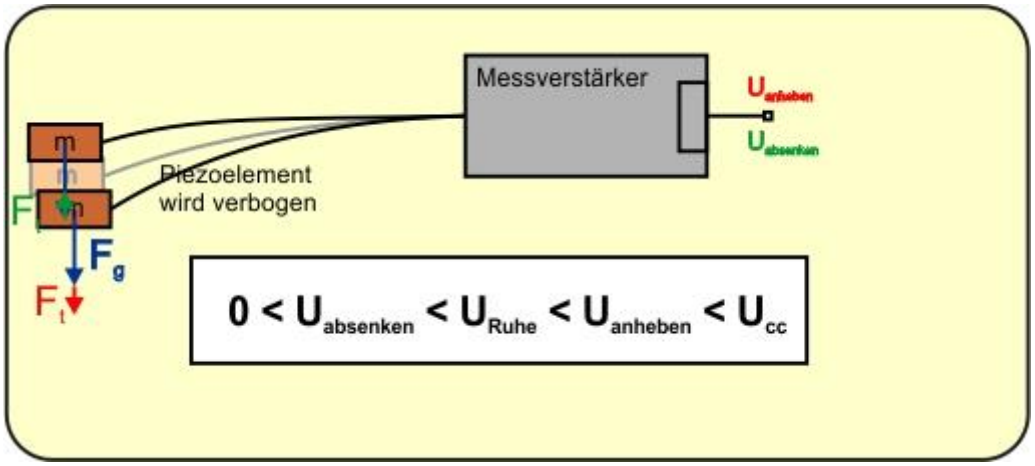

Abbildung 4: Kräftebilanz und Spannungen beim Schwingen

Hebt die schwingende Tischplatte das BOB an, bewirkt die Trägheit der Masse m am Piezoarm, die an ihrem Ort bleiben möchte, eine stärkere Biegung des Piezoelements nach unten, es entsteht am Ausgang des Messverstärkers eine höhere Spannung $U_{anheben}$. Im Gegenzug verringert sich aus dem gleichen Grund die Biegung des Elements, wenn das BOB mit der Tischplatte nach unten geht. Die Spannung am Ausgang sinkt.

Die Schwingungen erfolgen um die Ruhelage und bringen am ADC eine Differenz der Zählerpunkte (counts) von 18 (leichtes Klopfen) bis 242 (Faustschlag). Letzteres entspricht, wie wir später noch sehen werden ca. 0,5g. Die Erdbeschleunigung g = 9,81m/s² bewirkt über die Newtonsche Formel $F_g = m \cdot g$ die Verbiegung des Piezoelements. Die Massenträgheit führt durch eine Negativbeschleunigung a, bezüglich der Bewegungsrichtung, zur Trägheitskraft $F_t = m \cdot a$, welche die Gewichtskraft F_g überlagert. Beim Anheben werden die Kräfte addiert, beim Absenken verringert sich die Gewichtskraft um den Betrag von F_t.

Die Schaltung

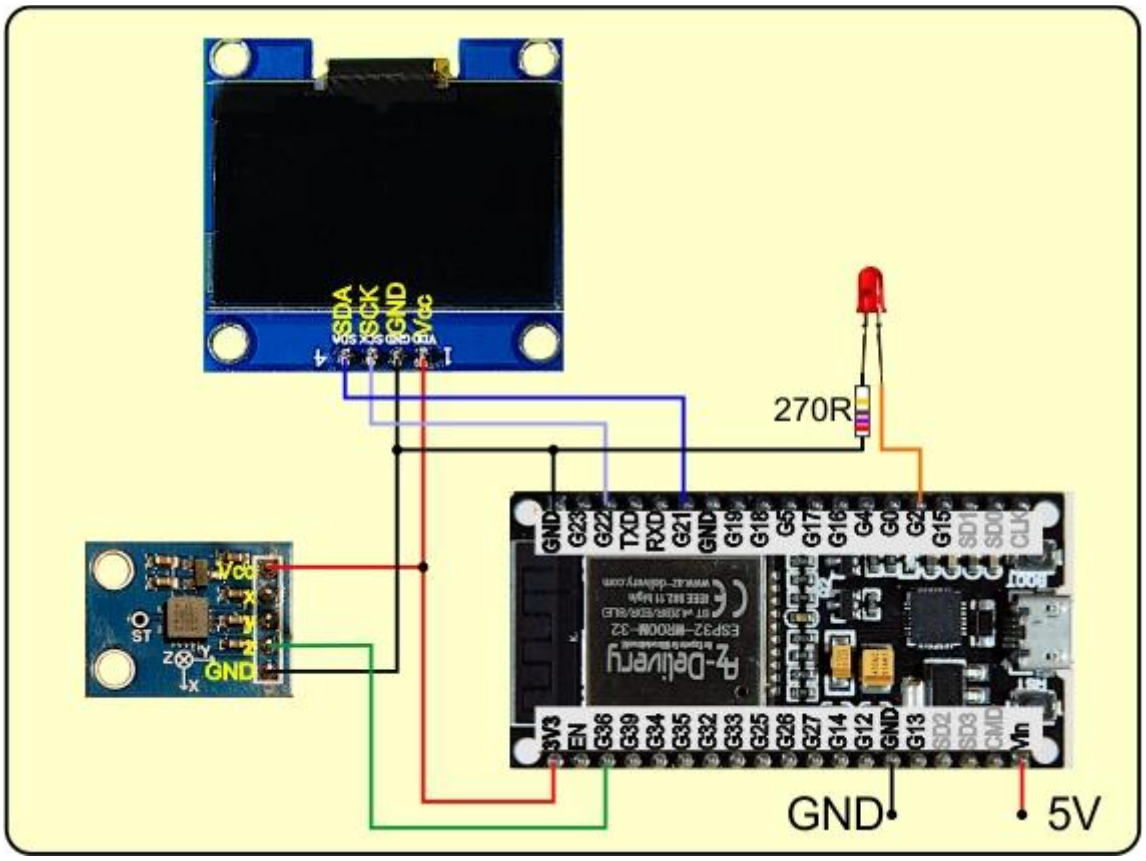

Abbildung 5: Seismometer - Schaltung

Die Beschaltung der drei Baugruppen ist denkbar einfach. Die Spannungsversorgung sollte für einen Langzeitbetrieb aus einem 5V-Steckernetzteil erfolgen. Während der Entwicklung versorgt der PC die Schaltung über den USB-Bus.

Das Programm

Man kann die Aufgaben des Programms in drei Bereiche aufteilen: erfassen, darstellen und dem PC die Daten zur weiteren Auswertung bereitstellen.

Die Erfassung geschieht in der Hauptschleife und startet automatisch durch das Auftreten einer Bodenschwingung, die mindestens den doppelten Pegel des Grundrauschens erreichen. Die Messdauer beträgt dann eine halbe Sekunde. Während dieser Zeit wird eine Liste aufgebaut, deren Werte nach Ablauf des Timers in eine Datei im Flash des ESP32 geschrieben werden. Die Benennung der Dateien geschieht automatisch über einen Zähler. Aber schauen wir uns das einfach mal der Reihe nach an.

```
from machine import Pin, ADC, SoftI2C
from time import sleep,ticks_ms
from sys import exit
from oled import OLED
from esp32 import NVS
```

Wir brauchen Pin-Objekte, den ADC und die I2C-Schnittstelle, ferner **sleep** und **ticks_ms** für Verzögerungen. Mit **exit**() bauen wir einen geordneten Ausstieg aus dem Programm. Die Klasse OLED liefert die API für das Display. NVS ist das Akronym für Non Volatile Storage = nicht flüchtige Speicherung. Gemeint ist eine Speichermöglichkeit für 32-Bit Integerwerte und sogenannte Blobs, das sind bytes-Objekte, im Flash-Speicher. Um darauf zugreifen zu können, erzeugen wir NVS-Objekt, dem wir den Bezeichner eines Namensraums geben. Jeder Namespace kann Bezeichner-Wertpaare aufnehmen und dauerhaft vorhalten. Das testen wir doch gleich einmal.

>>> from esp32 import NVS # Klasse importieren
>>> nvs=NVS("test") # Namensraum **test** erstellen
>>> blob=b"Das ist ein Test" # bytes-Objekt erzeugen
>>> blob # Kontrolle
b'Das ist ein Test'
>>> nvs.set_blob("text",blob) # Name ist **text**, Wert ist das bytes-Objekt **blob**
>>> container=bytearray(30) # zum Auslesen brauchen wir ein bytes-Array
>>> nvs.get_blob("text",container) # *Blob in das Array holen, 16 Bytes sind es*
16
>>> print(container[:16]) # den Inhalt wiedergeben
bytearray(b'Das ist ein Test')
>>> nvs.set_i32("zahl",1234567)
>>> nvs.get_i32("zahl")
1234567
>>> nvs.commit() # Werte in den Flash schreiben

Jetzt können Sie ausschalten – nach dem nächsten Booten sind die Werte noch da.

>>> from esp32 import NVS # Klasse importieren
>>> nvs=NVS("test") # Namensraum test referenzieren
>>> nvs.get_i32("zahl")
1234567
>>> nvs.erase_key("zahl") # den Schlüssel **zahl** löschen

>>> nvs.get_i32("zahl")
Traceback (most recent call last):
* File "<stdin>", line 1, in <module>*
OSError: (-4354, 'ESP_ERR_NVS_NOT_FOUND')

```
nvs=NVS("Quake")
```

Wir erzeugen ein I2C-Objekt. Und übergeben es an die OLED-Instanz d.

```
i2c=SoftI2C(scl=Pin(22),sda=Pin(21),freq=400000)

dheight=64
d=OLED(i2c,heightw=dheight)
d.writeAt("EARTHQUAKE 1.0",0,0)
```

An GPIO36 deklarieren wir die ADC-Instanz adc und stellen den, am ADXL335 gemessenen Spannungsbereich ein sowie die Auflösung auf 12 Bit. Der höchste Zählwert ist dann 4095.

```
adcPinNumber=36
adcPin=Pin(adcPinNumber)
adc=ADC(adcPin)
adc.atten(ADC.ATTN_11DB) # 150 - 2450 mV
adc.width(ADC.WIDTH_12BIT)
# 0...4095; LSB = 3149mV/4095cnt=0,769mV/cnt
# @313mV/g  (313mV/g)/0,769mV/cnt = 408cnt/g
# LSBg = 1/408cnt/g = 2,45mg/cnt
lsbC=3149/4095 # mV / cnt
lsbG=lsbC/313  # g / mV
```

Mit Hilfe eines DVM und des 10-Gang-Potentiometers bestimme ich den Spannungswert, für den ich 4095 counts (count = Zählwert-Einheit) erhalte. Das Voltmeter liegt an GPIO36 und GND. Damit kann ich das LSB (=Spannungswert für 1 cnt) des ADCs berechnen.

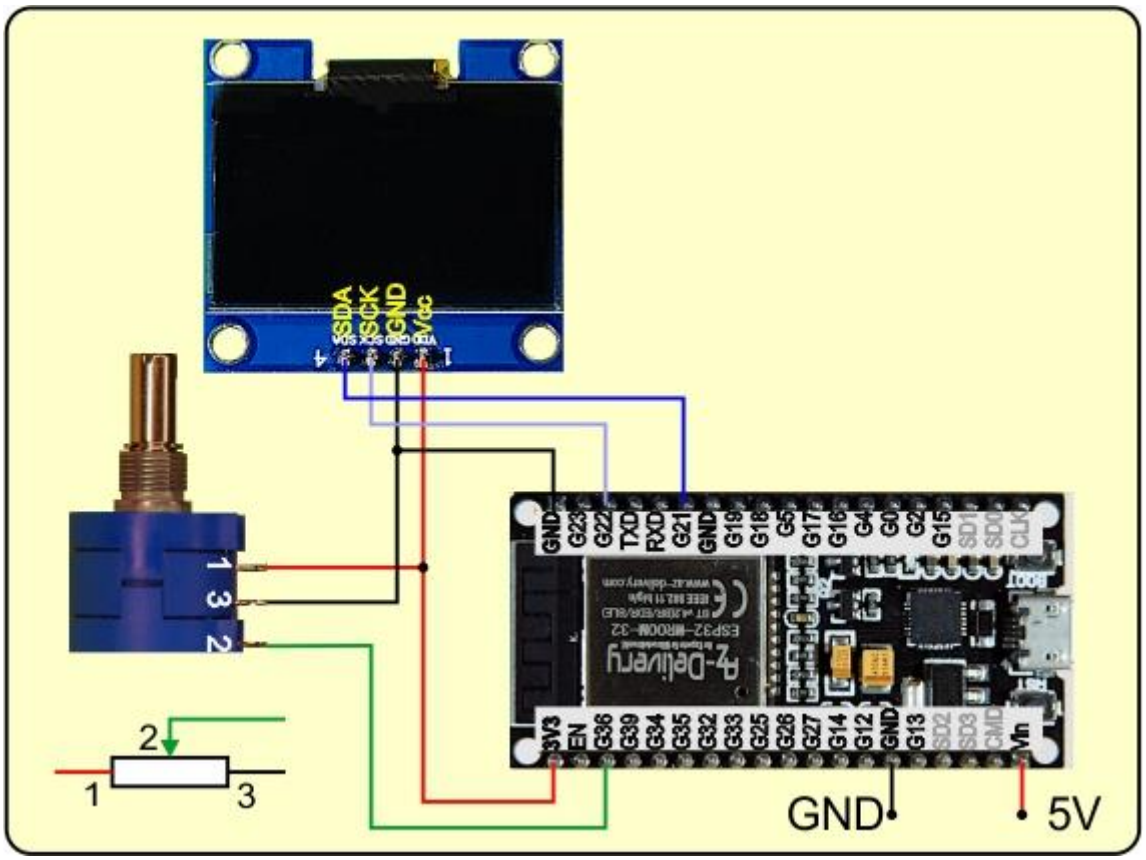

Abbildung 6: Eichen des ADC

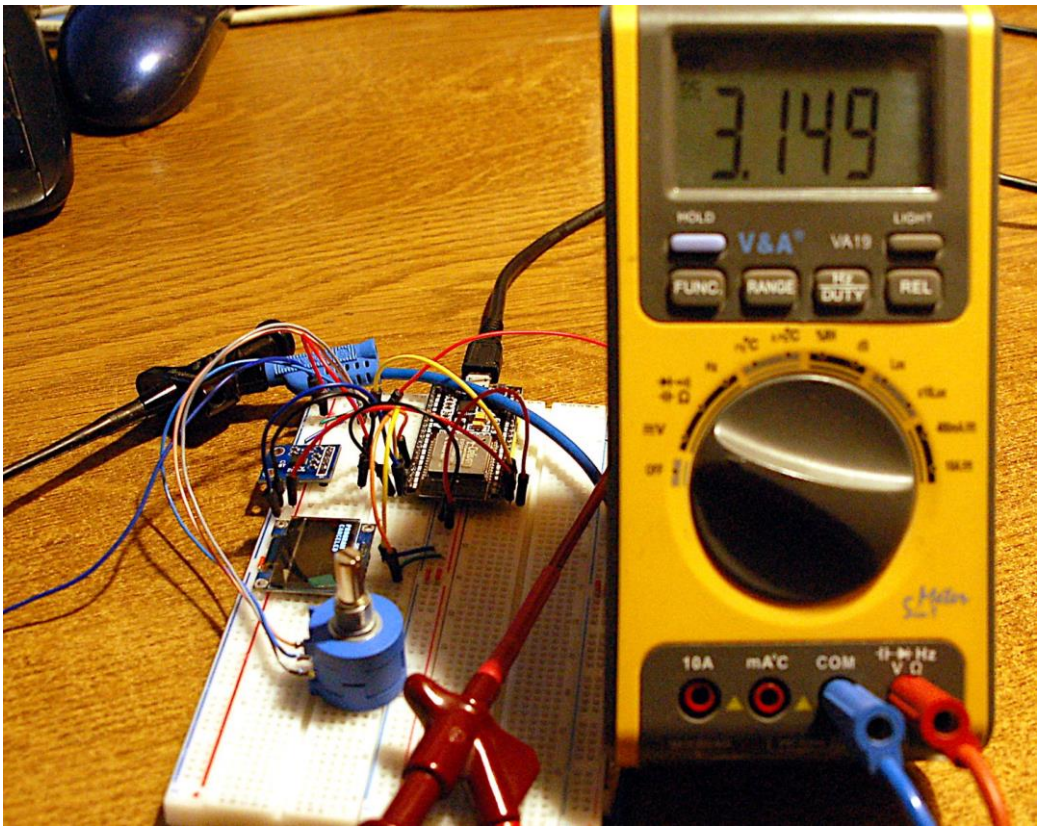

Abbildung 7: Erdbeben-sensor - ADC-Eichung

Am ADXL335 liegt am z-Ausgang eine Spannung von 0,313V an, wenn dieser parallel zur Tischfläche liegt. Er erfasst dann die Erdbeschleunigung g. Teile ich

diese Spannung durch das LSB vom ADC, dann erhalte ich die Anzahl cnt für 1g: 408 cnt/g. Der Kehrwert davon ist das LSB der Erfassung der Beschleunigung LSBg=2,45 Milli-g/cnt. Damit kann ich direkt die ADC-Werte in Beschleunigungswerte umrechnen.

Die Erdbeschleunigung interessiert mich nur am Rande, weil ich sie zur Berechnung der Beschleunigungswerte durch Bewegungen des Untergrundes brauche. Auch da begnüge ich mich mit dem ADC-Wert. Um das Rauschen (Abweichungen durch zufällige Messfehler von ADXL335 und ESP32-ADC) zu verringern, bestimme ich den Mittelwert von 1000 Einzelmessungen. s0 lege ich im NVS-Namespace **Quark** unter dem Schlüssel **s0** ab.

```
s0=0
su=4095
so=0
n=1000
m=[]
for i in range(n):
    s=adc.read()
    m.append(s)
    s0+=s
    su=min(su,s)
    so=max(so,s)
s0=int(s0/n)
nvs.set_i32("s0",s0)
nvs.commit()
```

Gleichzeitig habe ich den größten und kleinsten Messwert ermittelt. Daraus berechne ich die größte Abweichung nach unten und oben. Die Werte wandern mit **print**() ins Terminalfenster.

```
dsu=int(s0-su+1)
dso=int(so-s0+1)
dsc=max(dsu,dso)*2
ds = dsc*lsbC*lsbG
print ("s0= {}; su={}; so= {}; dsu={}; dso=
{}".format(s0,su,so,dsu,dso))
print ("Rauschen: {} cnts= {:0.2f} g".format(dsc,ds))
```

Die Abbruchtaste ist die Flash-Taste an GPIO0, die LED an GPIO2 sagt uns, wann eine Messung läuft.

```
bussy=Pin(2,Pin.OUT,value=0)
taste=Pin(0,Pin.IN,Pin.PULL_UP)
```

Zur Verteilung der Arbeit definiere ich ein paar Funktionen. Das schafft Übersicht durch Modularisierung.

```
def getAdc(n):
    s=0
    for i in range(n):
        s+=adc.read()
    return int(s/n)
```

Auch die Messergebnisse werden durch Mittelwertbildung ein wenig geglättet. Die Funktion **getS0**() holt mir den s0-Wert aus dem NVS-Namespace Quake.

```
def getS0():
    return nvs.get_i32("s0")
```

In der Hauptschleife wird jedes Quake-Ereignis in durchnummerierten Dateien festgehalten. **readData**() liest diese Dateien aus und speichert die Werte in der Liste **s**. Die Nummer der Datei übergebe ich beim Aufruf an den Parameter **n**.

```
def readData(n):
    global s
    s=[]
    name="data"+str(n)+".csv"
    with open(name,"r") as f:
        for line in f:
            zeile=line.strip()
            if  zeile.find(";") != -1:
                nr,s1=zeile.split(";")
                val=int(s1)
                s.append(val)
```

Damit die Änderungen an der Liste **s** aus der Funktion nach draußen durchdringen, deklariere ich s als **global**. Die Liste wird geleert und der Dateiname zusammengesetzt. Die Datei öffne ich mit der **with**-Anweisung zum Lesen, dadurch brauche ich mich am Ende nicht um das Schließen der Datei zu kümmern.

Die for-Schleife iteriert über den gesamten Inhalt und liefert mir in **line** den Text einer Zeile. Der Zeilenvorschub, "\n" = ASCII 10, wird abgezwickt. Dann suche ich nach einem ";". Ist das Trennzeichen enthalten, spalte ich die Zeile in Laufindex und Wert. Den Text wandle ich in eine Ganzzahl und hänge diese als neues Listenelement an s an.

Für die Darstellung in EXCEL kann ich den Inhalt der Liste s durch **convert**() in trickreicher Weise vorbereiten. Über den Parameter **v** übergebe ich einen Teil von s oder die ganze Liste. Ich hole **s0** und öffne eine Datei **data.csv** zum (Über-) Schreiben. Der String S0 erhält die Form ";2376". In der for-Schleife setze ich Zeilen der Form "23;2482;2376\r" zusammen und schreibe sie in die Datei. Windows braucht am Zeilenende statt des Line feeds \n einen Wagenrücklauf \r. Die so erzeugte Datei data.csv (csv = character separated Values) kann ich mit Rechtsklick vom ESP32 zum PC ins Arbeitsverzeichnis von Thonny hochladen und dort in Excel öffnen. Genaueres dazu später.

```
def convert(v):
    s0=getS0()
    with open("data.csv","w") as f:
        print("Elemente:",len(v))
        S0=";"+str(s0)
        for n in range(len(v)):
            line=str(n)+";"+str(v[n])+S0+"\r"
            f.write(line)
```

Die Funktion, die ich am häufigsten wiederverwende, ist **TimeOut**(). Die Closure erzeugt mir einen nichtblockierenden Softwaretimer. Die Ablaufzeit über gebe ich als Wert in Millisekunden. Die zurückgegebene Referenz auf die Funktion **compare**() weise ich einfach einem Bezeichner zu. So kann ich an verschiedenen Stellen im Programm den Timerzustand prüfen – noch nicht abgelaufen: **False**, beendet: **True**.

```
def TimeOut(t):
    start=ticks_ms()
    def compare():
        return int(ticks_ms()-start) >= t
    return compare
```

Zur Darstellung der Werte im Display kann man die Funktion **grafik**() aufrufen. Wie bei **convert**() übergebe ich als Argument eine Liste oder ein Slice (Scheibe) davon. S[:50] bringt die Werte von Platz 0 bis 49, s[45:132] holt die Werte von Platz 45 bis 131. Die Variable **laenge** merkt sich die Anzahl der Elemente, **ml** kriegt die halbe Displayhöhe ab. **min**() und **max**() ermitteln den größten und kleinsten Wert. Die beiden brauche ich für die Skalierung, damit jeder Datenpunkt ins Display passt. Mit **s0** und **max**() wird die größte Abweichung berechnet, die in die Berechnung des Skalierungsfaktors **yFaktor** eingeht. Die erste for-Schleife muss auf zwei Fälle reagieren können – mehr als 64 Werte oder bis zu 64 Werten in der Liste. Das klärt die konditionale Zuordnung an **ziel**. Bei maximal 64 Werten reicht ein Durchlauf.

Display löschen, Mittenlinie ziehen, der erste x-Wert ist 0. Der ganzzahlige Anteil der skalierten Differenz aus Messwert s1 und Ruhewert s0 wird von der halben Anzeigenhöhe subtrahiert, weil positive y-Werte oberhalb der Mittenlinie liegen müssen. Die linke obere Ecke des Displays hat die Koordinaten 0|0, die untere 0|63.

Die innere for-Schleife greift sich nun bis zu 62 Werte aus der Liste, beginnend bei der i-ten Position, welche die äußere Schleife vorgibt. Der x-Wert für den Endpunkt der Linie von x1|y1 aus, ergibt sich als Differenz von j und i. Der Faktor 2 dehnt die x-Achse, weil die Kurvenlinien sonst zu dicht aufeinander folgen. Das erklärt, warum bei einem Display mit 128 Pixel Breite der Index i nur bis maximal 64 Punkte vor Listenende läuft. Linie zeichnen und x1|y1 auf x2|y2 setzen. Bislang haben wir nur in den Puffer gearbeitet. **show**() schickt den Inhalt zur Anzeige.

Das Durchschieben der Werte kann ganz schön lange dauern. Das hängt davon ab, wie lang die Scandauer gewählt wird. Mit der Flash-Taste kann man den Vorgang deshalb abbrechen.

Damit stehen wir auch schon kurz vor der Hauptschleife, Dateienzähler auf 0.

Wir holen einen Wert vom ADC, berechnen die Abweichung vom Grundwert **s0**, der Absolutwert davon kommt nach **ds**. Liegt der Wert außerhalb des Grundrauschens, wird ein Scan getriggert. Den **trigger**-Wert schicken wir ans Terminal und ans Display. Dann wird ein Dateiname erzeugt und damit eine Datei zum Schreiben geöffnet.

```
n=0
while 1:
    s1=getAdc(3)#adc.read()
    trigger=s1-s0
    ds=abs(s1-s0)
    if ds > dsc :
        bussy.on()
        print("{}. Trigger: {}".format(n,ds))
        d.writeAt("{}. Trig {}".format(n,trigger),0,4)
        name="data"+str(n)+".csv"
        sock.sendto((str(n)+"\r").encode(),receiver)
        with open(name,"w") as f:
```

Wir erzeugen eine leere Liste s und stellen den Timer auf eine halbe Sekunde. **over** verweist jetzt auf die Funktion **compare**(), die wir durch **over**() aufrufen können. Weshalb das möglich ist, obwohl ja die Funktion **TimeOut**() mit der Rückgabe der Referenz auf **compare**() bereits beendet ist, das können Sie hier nachlesen.

```
            s=[]
            over=TimeOut(500)
```

Solange **over**() den Wert **False** zurückgibt, sind die 500 Millisekunden noch nicht abgelaufen. Weil unser Timer den Programmablauf nicht blockiert, können wir in der Zwischenzeit einen Haufen Dinge erledigen. Wir hängen den Messwert in die Liste, holen einen neuen, verschlafen eine Millisekunde und prüfen den Status der Flash-Taste. Ist sie in diesem Moment gedrückt, verlassen wir die Schleife

```
            while not over():
                s.append(s1)
                s1=getAdc(5)
                sleep(0.001)
                if taste.value()==0: break
```

Die Anzahl der Werte wird bestimmt und uns im Terminal mitgeteilt, dann schreiben wir die Liste in die Datei und erhöhen den Zähler für den nächsten Trigger.

```
            aw=len(s)
            print("***** bitte warten {} Werte *****".\
                  format(aw))
            for i in range(aw):
                f.write("{};{}\n".format(i,s[i]))
            print("========== FERTIG ================")
            n+=1
    if taste.value()==0:
        d.clearAll()
        d.writeAt("PROGRAM",0,0)
        d.writeAt("CANCELED",0,1)
        sys.exit()
```

Mit der Flash-Taste kann man das Programm geordnet verlassen.

Die Auswertung mit Excel

Mit **convert**() kann man eine Datei herstellen, die ohne Probleme direkt in Excel geöffnet werden kann. Im Dialog Datei öffnen wählen Sie den Typ **csv**. Navigieren Sie zum Arbeitsverzeichnis und öffnen Sie die Datei data.csv, die Sie mit **convert**() hergestellt haben.

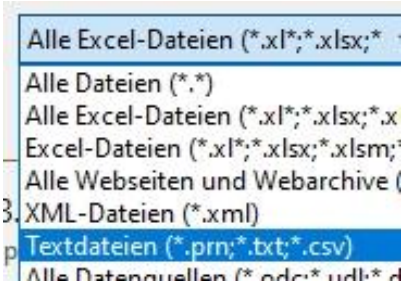

Abbildung 8: Dateityp csv

Markieren Sie die drei Spalten A,B und C. Durch die konstanten s0-Werte in der Spalte C bekommen wir auf einfache Weise die Null-Linie. Sie entspricht der Mittenlinie im Display. Öffnen Sie jetzt das Menü **Einfügen**.

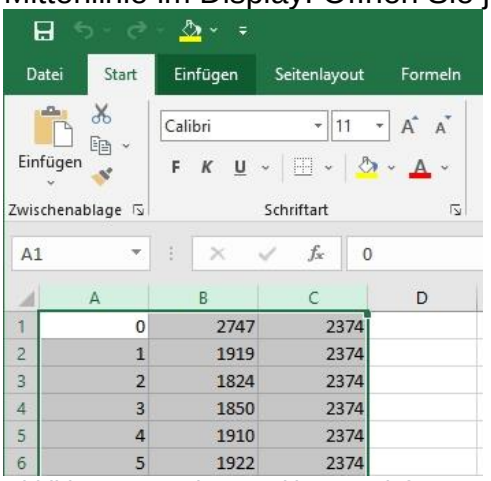

Abbildung 9: Spalten markieren - Einfügen

Klappen Sie das Menü **Punkt-xy-Diagramme** auf und klicken Sie auf **Punkte mit interpolierten Linien**

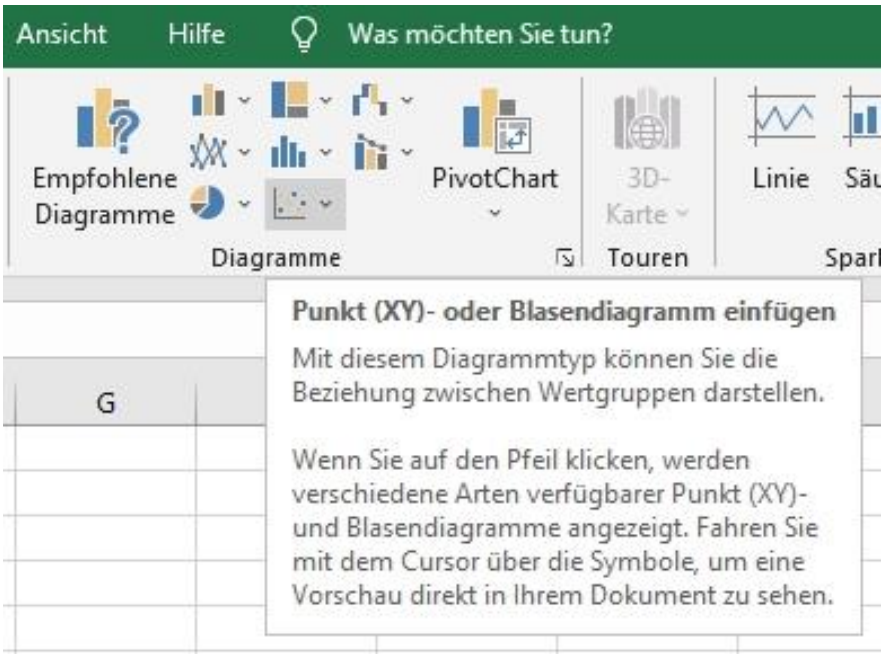

Abbildung 10: Punkt-xy-Diagramme

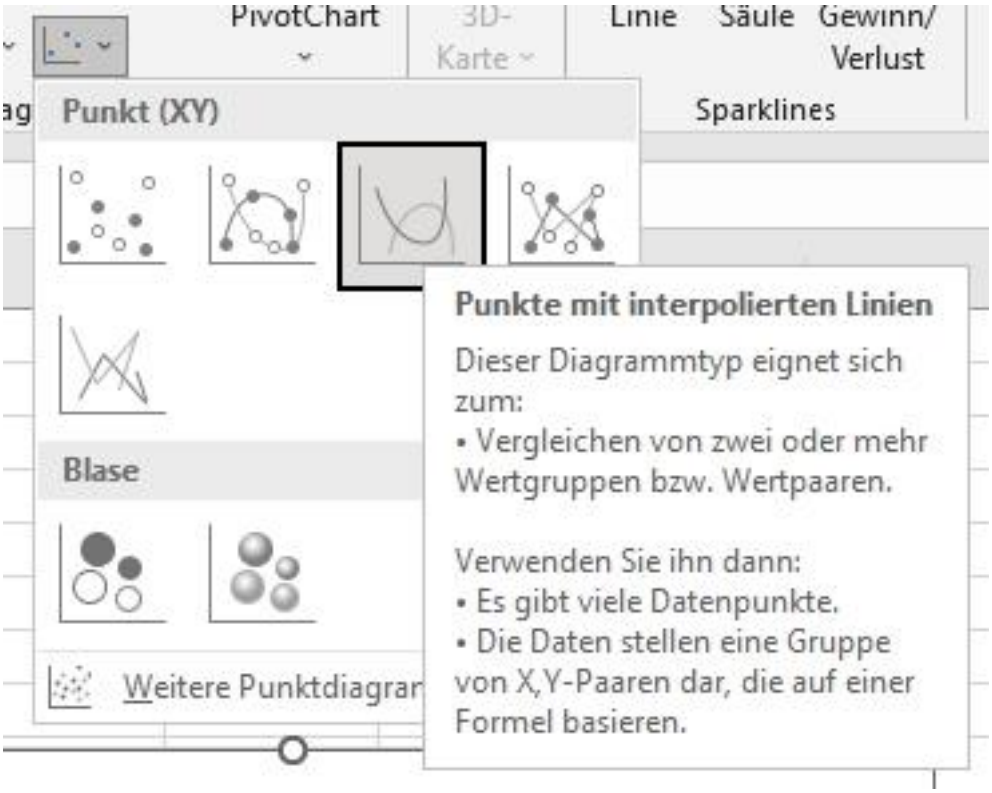

Abbildung 11: Punkte mit interpolierten Linien

Fertig!

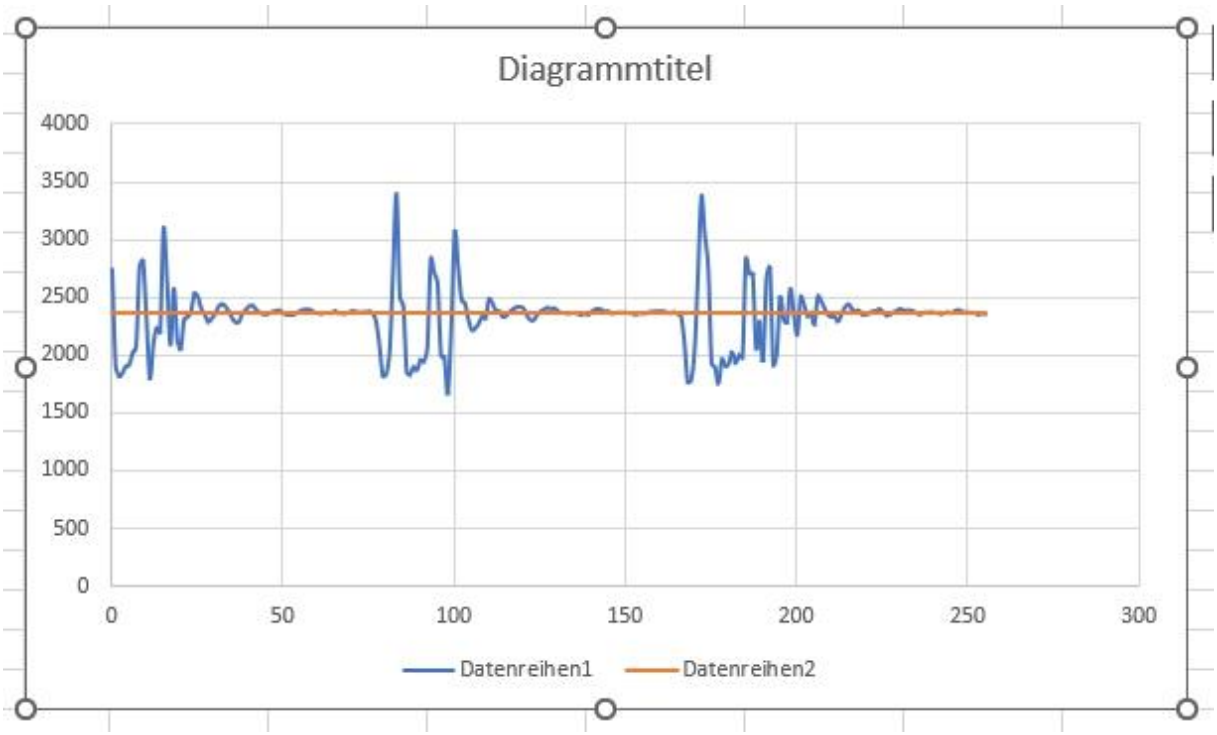

Abbildung 12: Fertiges Diagramm

Den Start der Aufzeichnung können Sie zoomen, mit einem Rechtsklick auf das Diagramm – Daten auswählen.

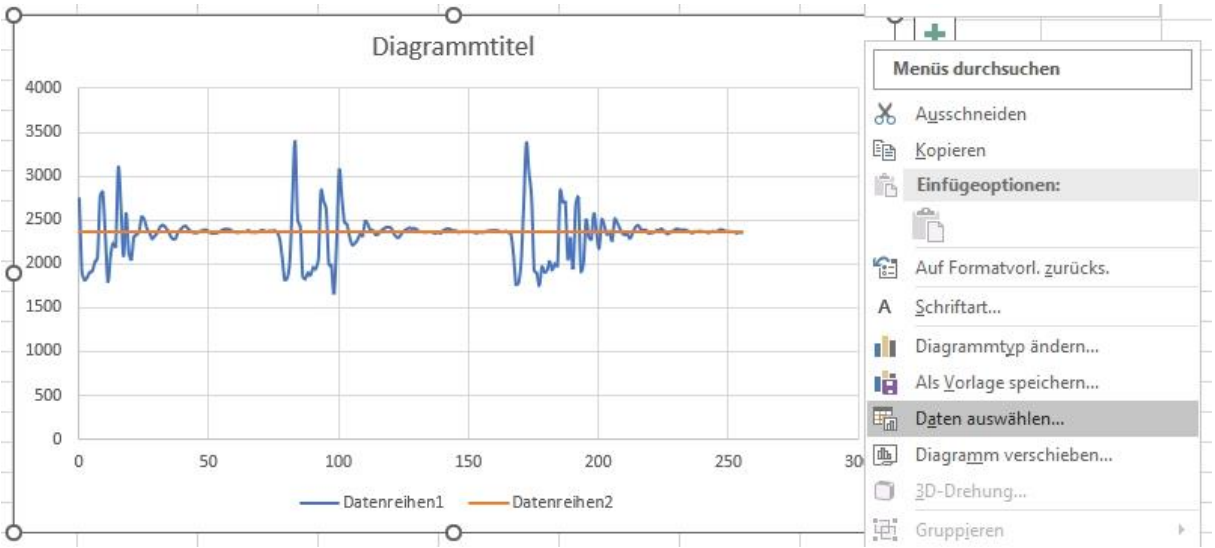

Abbildung 13: Daten auswählen

Ersetzen Sie den Wert hinter C$ durch einen kleineren - OK

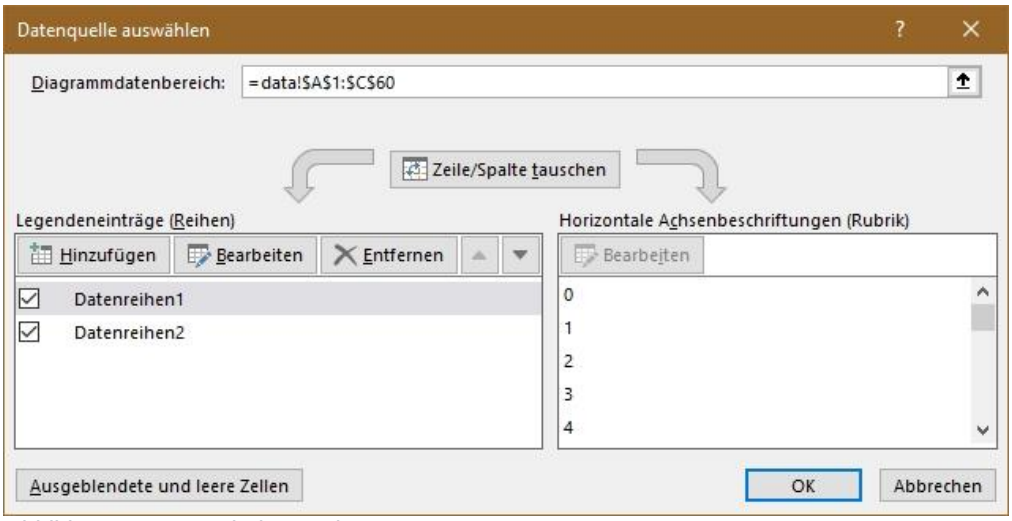

Abbildung 14; Ausschnitt anzeigen

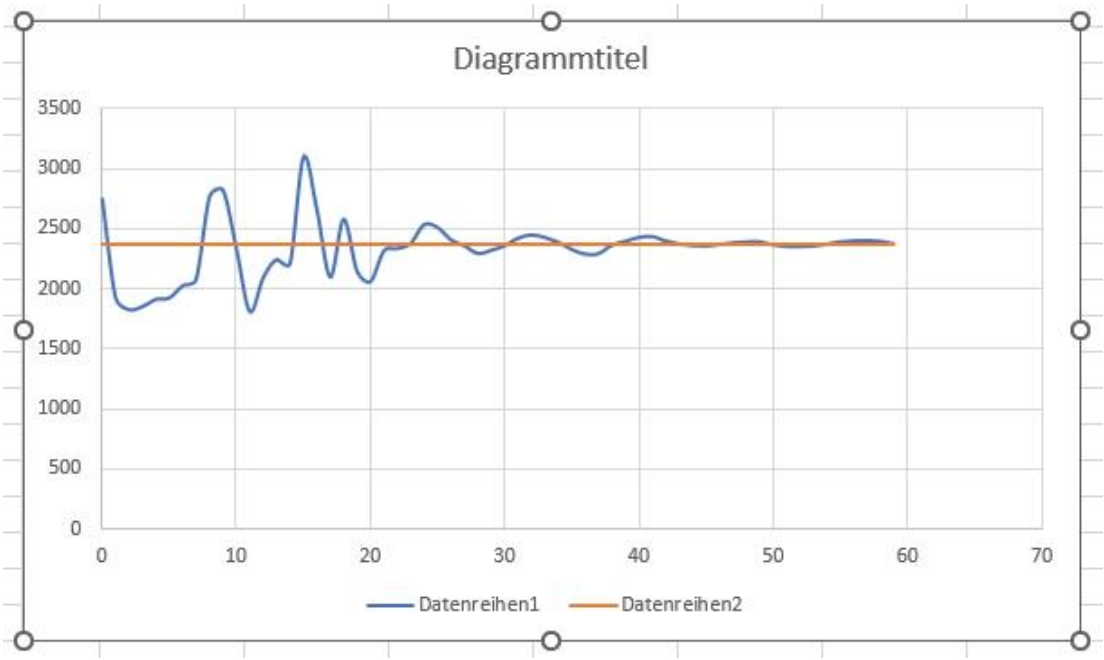

Abbildung 15: Zoom auf den Startbereich

Damit sind wir für heute am Ende angekommen. Was kommt als Nächstes?

Wie wäre es mit einem Funkkontakt vom ESP32 zum PC? Oder würde es Ihnen gefallen, wenn Sie mit dem Joystick durch die Messkurven auf dem Display wandern könnten? Praktisch wäre doch sicher auch eine Aufzeichnung der Messwerte auf einer SD-Speicherkarte. Wenn dann auch noch nachträglich feststellbar wäre, wann das auslösende Ereignis stattgefunden hat, wäre das auch nicht zu verachten, oder? Weil wir bislang nur die Erfassung der vertikalen z-Richtung betrachtet haben, sollten wir vielleicht auch die x- und y-Achse mit einbeziehen, denn Erdbebenwellen können sich auch logitudinal oder transversal in der Ebene ausbreiten.

Ich mache mir schon mal ein paar Gedanken darüber.

Bis dann.