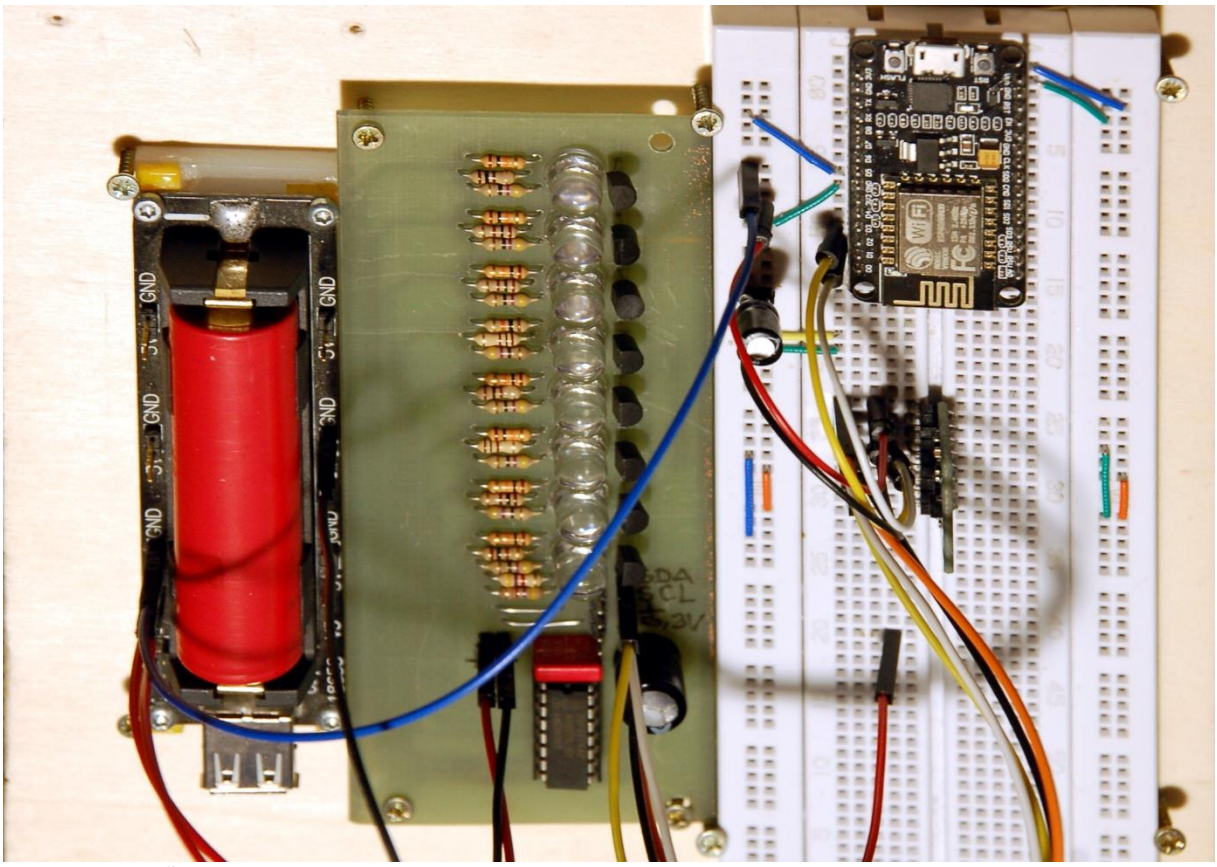

Gesamter Aufbau

Diesen Beitrag gibt es auch als PDF-Dokument zum Download.

Um eine besondere Form, bei der Übermittlung von Weihnachtsgrüßen geht es in diesem Beitrag. Zum Einsatz kommen acht LEDs, die von einem ESP32/ESP8266 über einen Porterweiterungsbaustein PCF8574(A) angesteuert werden. Die LEDs werden in einer Linie auf einer Lochrasterplatine zusammen mit dem PCF8574 platziert.

Die benötigten Bitmuster für die Buchstaben der Nachricht bringen die LEDs zum Leuchten. Dafür sorgt der Controller. Und damit der weiß, wann er mit dem Job anfangen soll, Muster für Muster zum PCF8574 zu übertragen, spendieren wir ihm noch ein Accelerator-Modul (Beschleunigungssensor).

Durch die Bewegung der Schaltung entstehen dann, weil die Muster an wechselnden Orten auftreten, quasi freischwebend die Buchstaben der Nachricht im Raum, der am besten abgedunkelt sein sollte. Wie das Zusammenspiel funktioniert, das erzähle ich Ihnen im heutigen Beitrag aus der Reihe

MicroPython auf dem ESP32 und ESP8266

heute

Weihnachtsgrüße per Schütteltext.

Die Aufgabe der Schaltung und des MicroPython-Programms, das wir dazu entwerfen werden, habe ich schon beschrieben. Wie lässt sich der Plan umsetzen und warum habe ich die Bauteile ausgewählt? Untersuchen wir die Hintergründe.

Ich beginne mit der Definition eines Buchstabens. Für die Festlegung des Musters habe ich mit Hilfe eines Zeichenprogramms (zum Beispiel Libre Office Draw) ein Raster erstellt und ausgedruckt. Eine Seite mit 8 mal 7 – Blöcken können Sie hier herunterladen. Schauen wir uns an, wie der Buchstabe „A“ codiert wird.

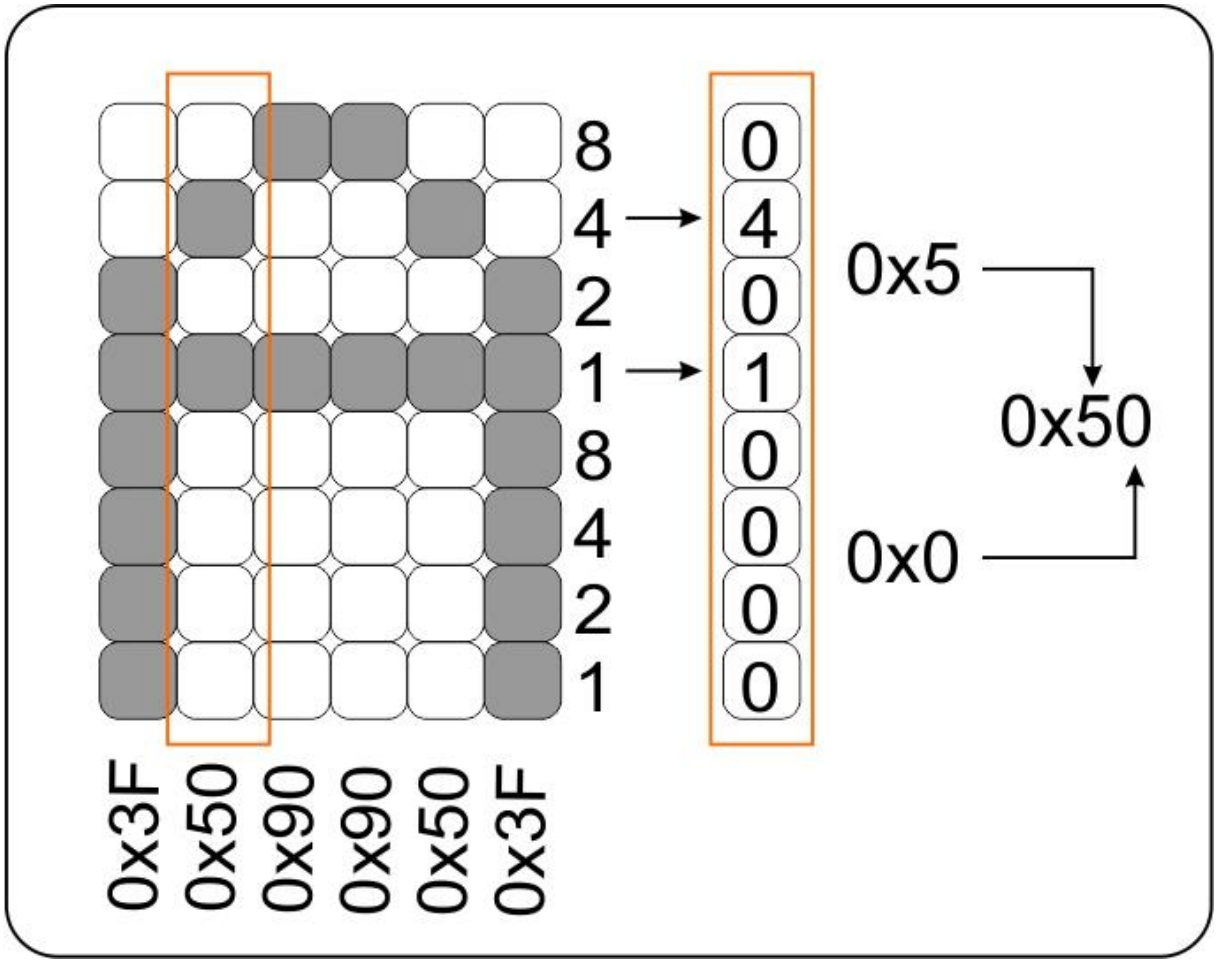

Abbildung 1: Buchstabenmatrix

Mit einem Stift wurden auf dem Ausdruck die Felder markiert, die später aufleuchten sollen. Wie die Zeichen aussehen sollen, legen Sie also selbst fest. Auch die Breite, ursprünglich auf sieben Spalten festgelegt, kann verändert werden. Aus einem Zeichensatz mit fester Breite wird so ein Proportional-Zeichensatz. Das „I“ ist dann zum Beispiel schmäler als ein „W“. Das Programm wird das berücksichtigen und zwar auf ganz simple Weise.

So wie in der Abbildung 1 die Spaltenmuster räumlich aufeinander folgen, folgen die Lichtmuster der LED-Zeile zeitlich und damit natürlich ebenfalls räumlich versetzt, wenn man den gesamten Aufbau seitlich bewegt. Wir machen uns später dazu noch weitere Gedanken. Abbildung2 zeigt eine schwankende Hin- und Her- Bewegung.

Abbildung 2: Freischwebender Grußtext

Beginnen wir mit der Besprechung der Hardware.

Hardware

Welcher Controller eingesetzt wird, steht zur freien Auswahl. Für die Programmentwicklung habe ich immer gerne ein Modul mit Flashtaste, weil ich dadurch die Möglichkeit erhalte, ein Programm geordnet verlassen zu können. Wenn der Umfang des Projekts es erlaubt, setzte ich auch gerne Controller-Boards ein, die auf dem Breadboard noch jeweils eine Kontaktreihe für Jumperkabel freilassen. Deswegen habe ich bei der Entwicklung der Schaltung hier einen ESP8266 Amica verwendet, der hat nämlich außerdem eine Flashtaste an Bord.

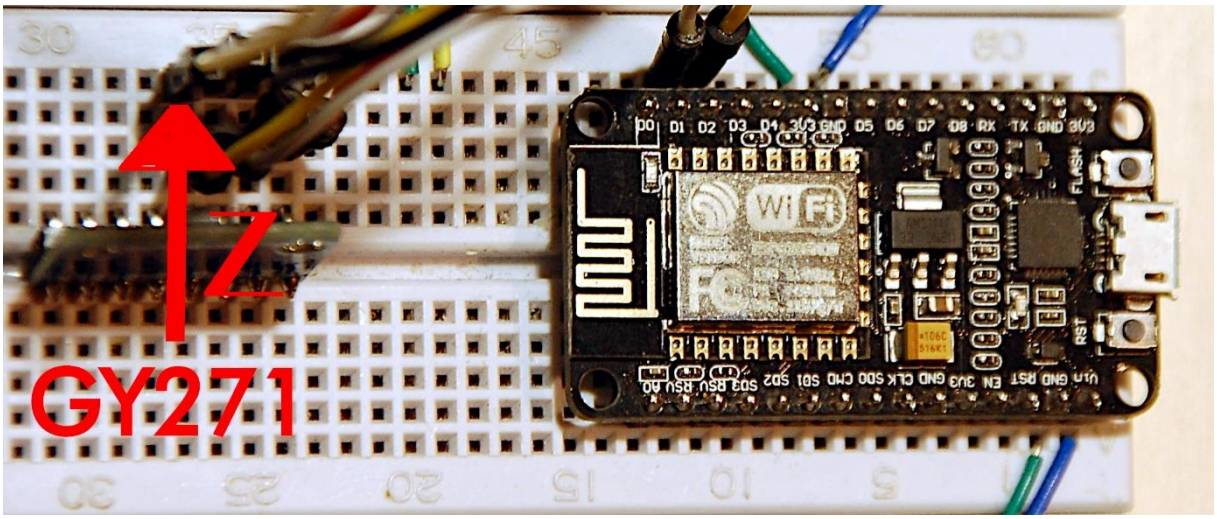

Abbildung 3: ESP8266 und GY-271

2	D1 Mini NodeMcu mit ESP8266-12F WLAN Modul oder D1 Mini V3 NodeMCU mit ESP8266-12F oder NodeMCU Lua Amica Modul V2 ESP8266 ESP-12F WIFI oder NodeMCU Lua Lolin V3 Module ESP8266 ESP-12F WIFI oder ESP32 Dev Kit C unverlötet oder ESP32 Dev Kit C V4 unverlötet oder ESP32 NodeMCU Module WLAN WiFi Development Board mit CP2102 oder NodeMCU-ESP-32S-Kit oder ESP32 Lolin LOLIN32 WiFi Bluetooth Dev Kit
1	GY-521 MPU-6050 3-Achsen-Gyroskop und Beschleunigungssensor
8	LEDs 3mm Ø z. B. aus LED Leuchtdioden Sortiment Kit, 350 Stück, 3mm & 5mm, 5 Farben oder Jumbo-LEDs 8mm Ø *** *****
8	Widerstände z. B. aus Widerstände Resistor Kit 525 Stück Widerstand Sortiment****
8	Widerstände 47Ω*****
8	Widerstände 10kΩ*****
8	PNP-Transistoren BC558 o. ä. *****
1	IC-Fassung 16-polig
1	Elektrolytkondensator 470µF / 16V z. B. aus ElKo Sortiment Elektrolytkondensator *****
1	PCF8574-Modul oder PCF8574-IC *
1	PCB Board Set Lochrasterplatte **
1	MB-102 Breadboard Steckbrett mit 830 Kontakten
diverse	Jumper Wire Kabel 3 x 40 STK. je 20 cm M2M/ F2M / F2F evtl. auch 65Stk. Jumper Wire Kabel Steckbrücken für Breadboard
optional	Logic Analyzer

* Für diese Variante gibt es Platinen-Layouts zum Download
** Die Mini-Ausführung der Schaltung kann anhand des Layouts auch leicht auf einer Lochrasterplatine verdrahtet werden.
*** Nur für das große Platinenlayout verwendbar.
**** Die Widerstandswerte richten sich nach der verwendeten LED-Farbe (siehe Text)
***** Für die Bestückung des großen Layouts

Die Schaltungen

Die Schaltung kann in verschiedenen Varianten aufgebaut werden. Das beginnt bei der Auswahl der Platinengröße. Es gibt ein Layout für Jumbo-LEDs mit 8mm Durchmesser und eine Minivariante mit 3mm-LEDs. Die kleine Variante kann auch auf den verlinkten Lochrasterplatinen, dem Layout folgend, bestückt und verdrahtet werden. Wie Sie selbst PCBs über das Bügel- oder Lichtpaus-Verfahren herstellen können ist hier beschrieben. Die PDF-Datei mit den Layouts laden Sie über den Link herunter. Die ausgedruckten Vorlagen werden mit der Druckseite direkt auf die Platine gelegt. Das Duo kommt dann unter das Bügeleisen oder auf die UV-Licht-Box.

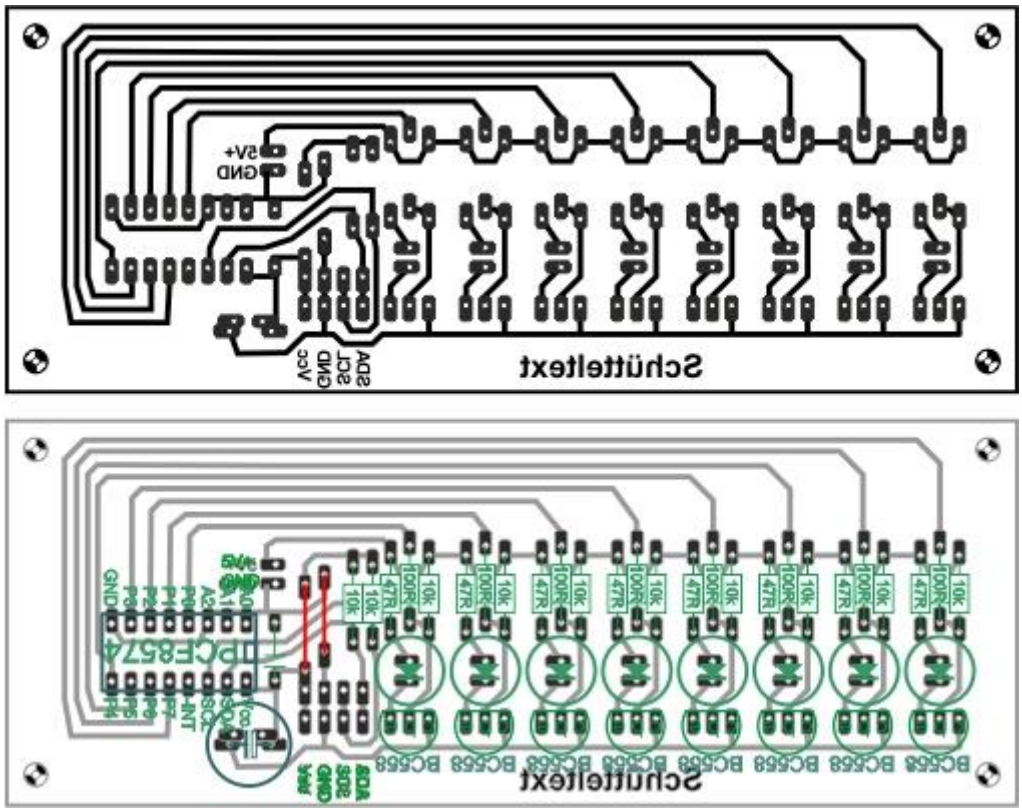

Abbildung 4: Schaltung mit Jumbo-LEDs

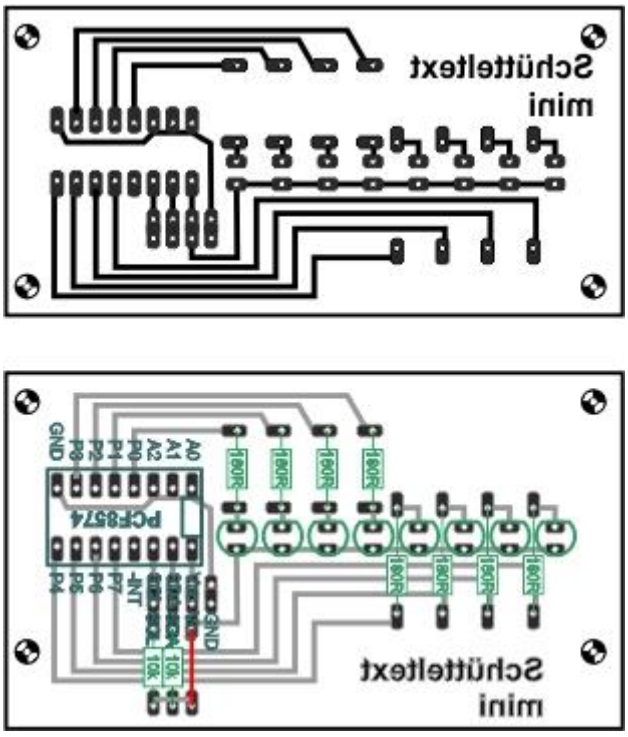

Abbildung 5: Schaltung Mini-Ausführung

Hier ist der Schaltplan für die etwas komplexere Schaltung mit den Jumbo-LEDs. Um größere Helligkeit zu erreichen, sind die LEDs an eine 5V-Quelle angeschlossen, die auch den Controller, einen ESP8266 D1 mini, mitversorgt. Der Plan zeigt auch den Anschluss des GY-521-Moduls und eines PCF8574-Moduls. Bei der Verdrahtung der Miniversion folgen Sie bitte einfach dem Layout in Abbildung 5. Die Kathode einer

LED ist übrigens immer der kürzere Anschluss (kurz wie Kathode). Außerdem ist der Sockelring des Gehäuses auf der Kathodenseite abgeflacht.

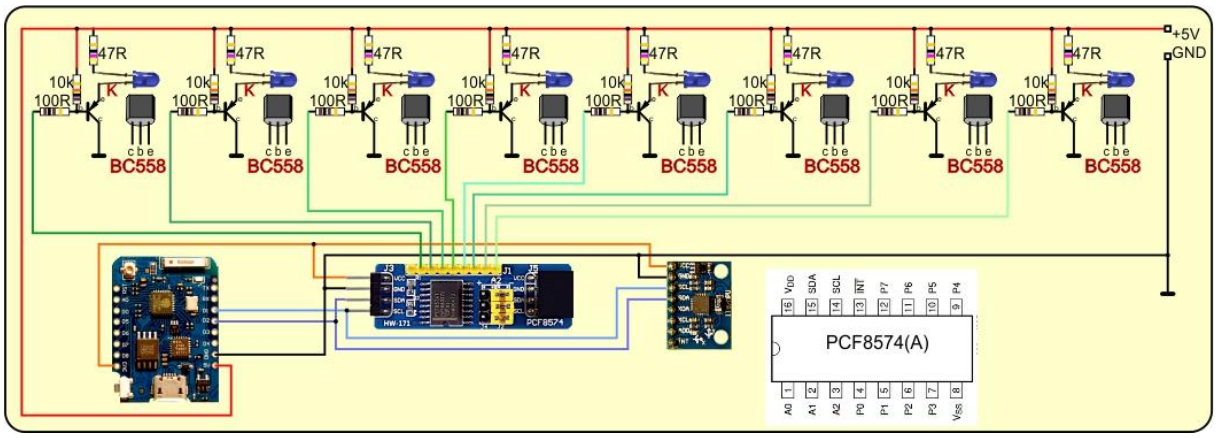

Abbildung 6: LED-Zeile – Schaltplan mit PCF8574-Modul

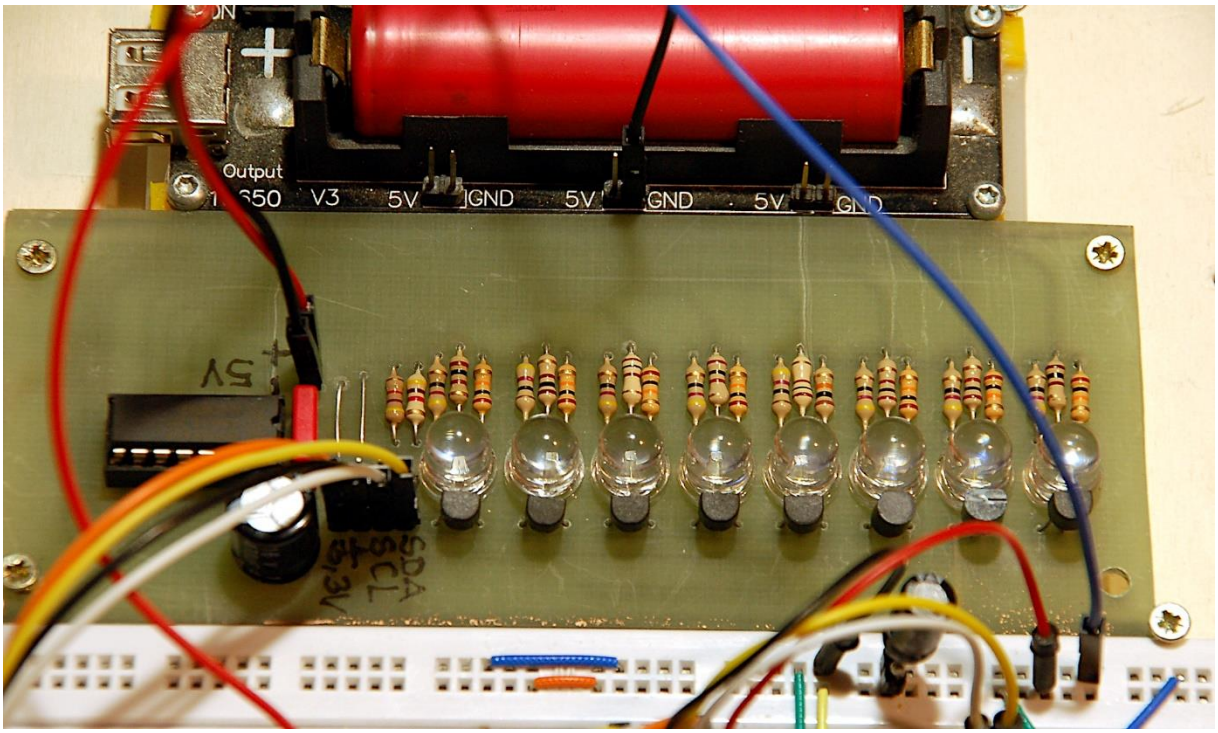

Abbildung 7: LED-Zeile maxi

Was leistet das PCF8574-Modul? Nun, es sorgt dafür, dass alle LEDs der Spalte **gleichzeitig** aufleuchten. Würde ich die LEDs über separate GPIOs schalten, müsste das zwangsweise zeitversetzt passieren, was zu einer Verformung der Buchstaben führen würde. Mit dem Porterweiterungsbaustein, der sein Bitmuster für die Ausgänge über den I2C-Bus erhält und erst danach parallel umsetzt, werden alle relevanten Positionen simultan geschaltet. An einem AT MEGA328 (Arduino) gibt es Anweisungen, die alle erreichbaren Bits eines Ports gemeinsam ansteuern. Die ESPs haben diese Möglichkeit leider nicht. Daher der Umweg über den PCF8574.

Wird statt dem **PCF8574-Modul** wie in Abbildung 6 dargestellt, ein **PCF8574-IC** verwendet, müssen wir berücksichtigen, dass die I2C-Bus-Leitungen mit je einem Pullup-Widerstand von 4,7kΩ bis 10k auf Vcc gezogen werden müssen. In I2C-

Modulen sind diese in der Regel bereits eingebaut. Zusammen mit dem GY521-Modul am Bus sollte es also auch ohne die beiden Pullups funktionieren, es reicht nämlich, wenn an einer Stelle die Widerstände eingefügt sind. Die INT-Leitung wird in diesem Projekt nicht verwendet.

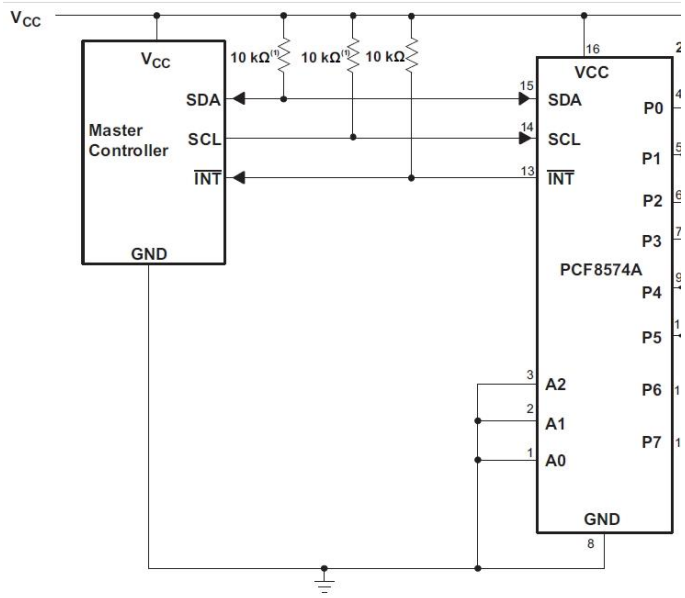

Abbildung 8: SDA-SCL-Beschaltung

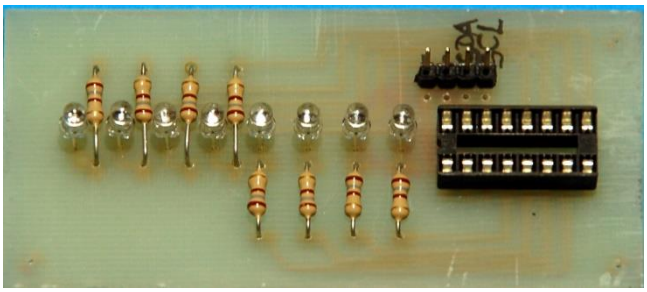

Abbildung 9: LED-Zeile mini (mit gelben 3mm-LEDs)

Die Ausgänge des PCF8574 sind durch eine Gegentakt-CMOS-Stufe realisiert. Der untere Transistor zieht den Ausgang auf GND-Potenzial, wenn wir eine 0 an diese Stufe senden. Eine 1 legt den Ausgang über die 100µA-Konstantstromquelle auf Vcc-Potenzial. Das heißt aber auch, dass eine Stromsenke am Ausgang Px maximal 100µA aus der Leitung ziehen kann. Das ist zu wenig, um damit eine nachfolgende Transistorstufe anzusteuern und reicht schon gar nicht, um damit eine LED zu betreiben.

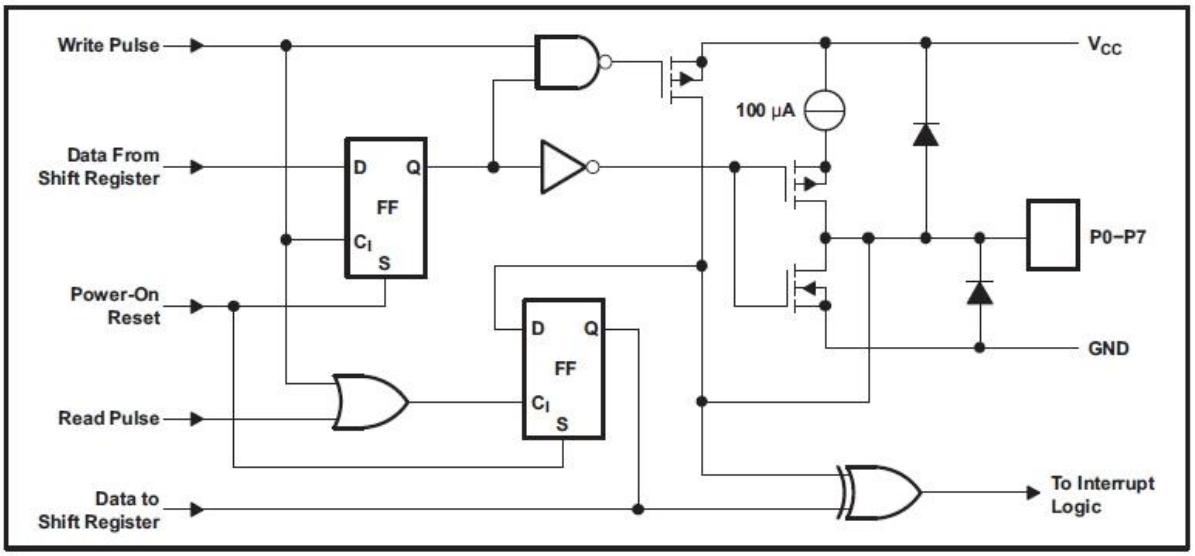

Abbildung 10: Innenleben des PCF8574

Aus diesem Grund sind die Transistoren in Abbildung 5 vom Typ PNP. Damit der BC558 durchschaltet, muss seine Basis auf GND-Potenzial gezogen werden. Das kann der Ausgang des PCF8574 leisten und er kann auch den Kollektor der LEDs in der Mini-Schaltung auf Masse ziehen und dabei bis maximal 25mA aufnehmen, die von der LED geliefert werden. In der vorliegenden Schaltung erlauben die 47Ohm-Widerstände einen Strom von ca 11 mA. Der BC558-Basisstrom liefert zusammen mit den 100 Ohm- und 10kΩ-Widerständen ca. 2,7mA. Beides ist also im grünen Bereich und kann vom PCF8574-Ausgang aufgenommen und nach GND abgeführt werden.

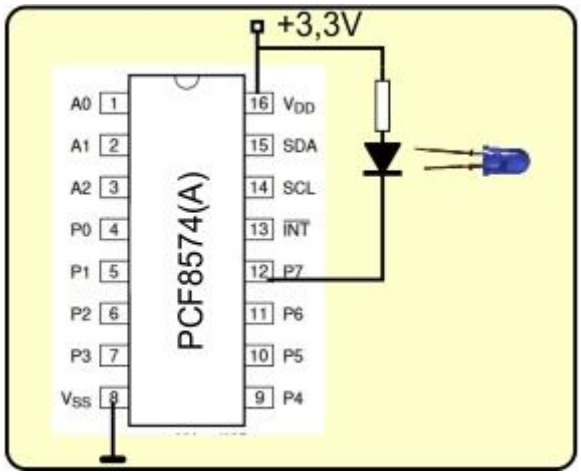

Abbildung 11: LED direkt am PCF8574

Nun ergibt sich daraus das Problem, dass in den Character-Definitionen eine 1 für eine leuchtende LED steht, der PCF8574 aber eine 0 an dem entsprechenden Ausgang liefern muss. Der Controller muss also den Bytewert vor der Ausgabe an den PCF8574 bitweise negieren. Das passiert im Programm durch Exoderieren mit 0xFF – alle Probleme gelöst – FAST!

Da ist noch die Sache mit den Strombegrenzungswiderständen, die mit den LEDs in Reihe geschaltet sind und je nach LED-Farbe angepasst werden müssen. Die Vorwärts- oder Durchlassspannung der LED bestimmt zusammen mit der angepeilten Stromstärke letztlich den Wert des Reihenwiderstands. Die Vorwärtsspannung hängt von der Farbe der LED ab und ist dem Datenblatt zu entnehmen. Ganz grob ergibt sich folgender Zusammenhang.

Farbe	**Uv**
rot	1,8V
gelb	1,9V
grün	2,1V
blau	2,8V
violett	3,2V
weiß	3,2V

Am Widerstand muss so viel Spannung abfallen, dass von der Betriebsspannung Vcc an der LED genau die Vorwärtsspannung überbleibt. Es Rechenbeispiel zeigt, wie das geht.

Vcc = 3,3V
LED-Farbe: gelb
U_V = 1,9V
U_R = 3,3V – 1,9V = 1,4V
Ziel-Stromstärke I_Z = 8mA = 0,008A
Widerstandsformel: R = U / I = 1,4V / 0,008A = 175 V/A = 175 Ω

In der E12-er Reihe sind die entsprechenden Werte 180Ω oder, wenn's a Bisserl mehr an Strom sein darf, 150Ω, das ergibt gute 9mA.

Die Software

Fürs Flashen und die Programmierung des ESP32:

Thonny oder
µPyCraft

Verwendete Firmware für den ESP32:

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

Verwendete Firmware für den ESP8266:

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

Die MicroPython-Programme zum Projekt:

mpu6050.py: Treiber für das GY-521-Modul

shakingtext.py: Demosoftware

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 18.06.2022) 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.

Das Schüttelprogramm

Für den Fall, dass Sie die Schaltung erst einmal ausprobieren wollen, empfiehlt sich der Aufbau auf einem Breadboard. Das oben verlinkte bietet reichlich Platz für alle Bauteile und wir können mit der Besprechung des erfrischend kleinen Programms beginnen.

Wie eingangs schon erwähnt, arbeitet das Programm automatisch auf beiden Controller-Familien. Ich habe hier ein ESP8266 – Amica-Board eingesetzt, weil ich dadurch mit nur einem Breadboard auskommen kann. Weil ich die PCBs für die LEDs aber bereits für Testzwecke hergestellt hatte, stecken auf dem Breadboard nur der ESP8266 D1 mini und das GY-271-Modul. Batterie, Breadboard und LED-PCB habe ich auf eine Sperrholzplatte geschraubt, damit ich alle Teile zusammen auch gut schütteln kann und nichts fliegen geht. Für die Schüttelversuche sollte man außerdem den Aufbau vom USB abtrennen, damit bei den heftigen Bewegungen die Buchse am Controller nicht beschädigt wird. Das Programm wird unter dem Namen **main.py**, wie oben beschrieben, in den Flash des ESP8266 hochgeladen. Nach einem Reset startet der Controller dann autonom auch ohne USB-Verbindung.

Das Importgeschäft für unser Programm fällt schon mal sehr übersichtlich aus. In der letzten Zeile importieren wir aus dem Modul **mpu6050** die Klasse **ACCEL**. Die Datei **mpu6050.py** muss sich dafür im Flash des Controllers befinden. Die anderen Klassen und Methoden sind bereits im Kern von MicroPython enthalten.

```
from machine import SoftI2C,Pin
from time import sleep_ms
import sys
from mpu6050 import ACCEL
```

Dann fällt die Auswahl des Controllertyps an. Die Textkonstante **sys.platform** informiert uns über den Controllertyp. Entsprechend den Standardvorgaben instanziieren wir ein I2C-Bus-Objekt. Die für den Bus, laut Datenblatt des PCF8574, vorgegebene Transferfrequenz von 100000kHz bremst den MPU6050 aus, der mit 400000kHz arbeiten könnte.

```
if sys.platform == "esp8266":
    i2c=SoftI2C(scl=Pin(5),sda=Pin(4), freq=100000)
elif sys.platform == "esp32":
    i2c=SoftI2C(scl=Pin(22),sda=Pin(21) ), freq=100000)
else:
    raise RuntimeError("Unknown Port")
```

Das I2C-Objekt übergeben wir dem Konstruktor der MPU6050-Klasse und erhalten eine solche Instanz zurück. Dann steuern wir zur flüssigen Unterhaltung zwischen dem Controller und der Peripherie bei, indem wir die Geräteadressen von PCF8574 und MPU6050 deklarieren.

```
mpu=ACCEL(i2c)

HWADRpcf=0x38
HWADRmpu=0x68
```

Das Tasten-Objekt für den kontrollierten Programmabbruch und ein Ausgang zum Messen der Frequenz für die Spaltenübertragung der Zeichen werden festgelegt. Als Taste nehme ich die Flash-Taste des ESP8266-Moduls.

```
taste=Pin(0,Pin.IN,Pin.PULL_UP)
trigger=Pin(14,Pin.OUT, value=0)
```

Die Zeichen-Matrix legen wir in Form eines Dictionarys fest. In den geschweiften Klammern werden Schlüssel-Wert-Paare, durch Kommas getrennt, aufgelistet. Der Key ist ein ASCII-Zeichen, nach einem Doppelpunkt folgt der zugewiesene Wert in Form eines Tupels. Wie die Feldwerte der Matrix ermittelt werden, habe ich oben schon beschrieben. Die letzte Spalte, die letzte Hexadezimalzahl, ist stets 0x00, damit die Zeichen nicht unmittelbar zusammenkleben wie die Kletten. Fügen Sie gerne noch weitere Trennspalten hinzu, ganz nach Ihrem Gefühl.

```
zeichen={
    "F":(0xff,0x90,0x90,0x90,0x80,0x80,0x00),
    "R":(0xff,0x90,0x90,0x90,0x9c,0x63,0x00),
    "O":(0x7e,0x81,0x81,0x81,0x81,0x7e,0x00),
    "H":(0xff,0x10,0x10,0x10,0x10,0xff,0x00),
    "A":(0x3f,0x70,0x90,0x90,0x70,0x3f,0x00),
    "E":(0xff,0x91,0x91,0x91,0x81,0x81,0x00),
    "S":(0x61,0x91,0x91,0x91,0x91,0x8e,0x00),
    "T":(0x80,0x80,0xff,0x80,0x80,0x80,0x00),
    "W":(0xfe,0x01,0x1e,0x01,0x06,0xf8,0x00),
    "I":(0x81,0xff,0x81,0x00),
    "N":(0xff,0x40,0x30,0x0c,0x02,0xff,0x00),
    "C":(0x7e,0x81,0x81,0x81,0x81,0x42,0x00),
    "*":(0x4a,0x2c,0xf8,0x1e,0x34,0x52,0x00),
    "P":(0xff,0x90,0x90,0x90,0x90,0x60,0x00),
    "U":(0xfe,0x01,0x01,0x01,0x01,0xfe,0x00),
    "J":(0x84,0x82,0x81,0x81,0x82,0xfc,0x00),
    " ":(0x00,0x00,0x00,0x00,0x00,0x00,0x00),
    }
```

Es folgen Definitionen für Funktionen zum Zeichentransfer, **writeReg**() schickt ein Byte an ein Peripherie-Device auf dem I2C-Bus. Zu übergeben sind die Hardware- oder Geräteadresse, die Registeradresse und das Datenbyte.

```
def writeReg(hwadr, adr, dat):
    i2c.writeto(hwadr,bytearray([adr,dat]))
```

Wichtig ist die Umwandlung von Registeradresse und Bytewert in ein Bytearray. Das ist notwendig, weil die Methode **writeto**() der i2c-Instanz keine Ganzzahlen, sondern nur Objekte akzeptiert, die das sogenannte **Bufferprotokoll** unterstützen, das sind Objekte vom Typ **bytes**- oder **bytearray**. Wir bauen die beiden Ganzzahlen dazu in eine Liste ein, die wir anschließend in ein Bytearray umwandeln lassen, indem wir die Liste dem Konstruktor der Klasse **bytearray** übergeben.

Beachten Sie bitte, dass der folgende Befehl ein ganz anderes Ergebnis bringt.

>>> bytearray(2,3)
bytearray(b'\x00\x00')

Hier wird ein Bytearray mit zwei Feldern mit dem Defaultwert 0x00 erzeugt und die 3 wird einfach kommentarlos vom Konstruktor gefressen, wohl bekomm's!

Der PCF8574 ist ein ganz genügsames Kerlchen, das nicht einmal Register besitzt. Schreibanweisungen gehen direkt in den Ausgangspuffer. Das heißt es muss, oder besser darf keine Registeradresse gesendet werden. **pcfWrite**() berücksichtigt das.

```
def pcfWrite(hwadr, dat):
    i2c.writeto(hwadr,bytearray([dat]))
```

Die Hardwareadresse des Chips wird dennoch mit übergeben, denn es können bis zu acht PCF8574 auf dem Bus liegen. Möglich wird das durch die drei Adress-Pins A0, A1 und A2. Für unser Projekt liegen alle drei Pins auf GND-Potenzial. Damit wird der PCF8574 mit seiner Basisadresse angesprochen. Aber selbst die kann variieren, denn es gibt verschiedene Ausführungen dieses Bausteins. Am besten wird es sein, wenn Sie den PCF8574 alleine auf den Bus legen und mit **i2c.scan**() seine Adresse ermitteln. Das geht in REPLso.

>>>from machine import SoftI2C,Pin
>>> i2c=SoftI2C(scl=Pin(5),sda=Pin(4), freq=100000)
>>> hex(i2c.scan()[0])
'0x38'

Mein **PCF8574AP** hat demnach die Hardware-Adresse **0x38**, in den Datenblättern findet man für den PCF8574 die 0x40 und für den PCF8574A die 0x70. Das Mysterium wird komplett, weil uns **i2c.scan**() die **7-Bit-Adresse** liefert und im **Datenblatt** aber die **8-BitAdresse** mit dem R/W-Bit angegeben ist. Auch für **i2c.writeto**() und **i2c.readfrom**() wird die 7-Biteadresse benötigt, weil MicroPython das R/W-Bit, je nach Operation automatisch selbst hinzufügt. Die Werte aus den Datenblättern müssen also um ein Bit nach rechts geschoben werden.

>>> hex(0x40 >> 1)
'0x20'
>>> hex(0x70 >> 1)
'0x38'

Der Scan hat den PCF8574AP also korrekt als A-Typ geortet.

Um ein Byte vom Bus zu lesen, müssen für einen Baustein, der Register benutzt, Hardware-Adresse und Register-Adresse übergeben werden. Beides wird an den externen Chip gesandt.

readfrom() liest daraufhin ein **bytes**-Objekt der Länge 1 ein. Durch Indizieren mit [0] erhalten wir den Dezimalwert dieses ersten Bytes.

```
def readReg(hwadr, adr):
    i2c.writeto(hwadr,bytearray([adr]))
    return self.i2c.readfrom(hwadr,1)[0]
```

Es folgt eine Funktion, welche die Bit-Matrix für ein Zeichen an den PCF8574 senden kann.

```
def showChar(char,delay=5):
    c=(0x10,0x10,0x10,0x10,0x10,0x10,0x00)
    if char in zeichen:
        c=zeichen[char]
    for val in c:
        trigger.value(1)
        val ^= 0xFF
        pcfWrite(HWADRpcf,val)
        sleep_ms(delay)
        trigger.value(0)
```

showChar() nimmt das ASCII-Zeichen und optional im Schlüsselwortparameter **delay** die Zeit der Anzeige in Millisekunden. Falls ein nichtdefiniertes Zeichen übergeben wurde, setzen wir in **c** das Muster für einen Bindestrich, "-". Wird das Zeichen als Schlüssel im Dictionary **zeichen** gefunden, belegen wir **c** mit dessen Muster.

In **c** ist jetzt in jedem Fall eine Zeichen-Matrix. In der for-Schleife tasten wir die Spalten ab. Unser Trigger-Ausgang geht auf 1, wir exoderieren den gepflückten Wert in **val** mit 0xFF und senden das Ergebnis an den PCF8574. Nach **delay** Millisekunden geht der Trigger-Pin auf 0. Die Schleife sendet so viele Werte, wie im Tupel in **c** angegeben sind. Dadurch können beliebig breite Muster dargestellt werden. Den Trigger-Ausgang können wir mit einem DSO (Digitales Speicher Oszilloskop) oder dem Logic Analyzer abtasten, um die tatsächliche Zeichen-Folge-Frequenz zu bestimmen.

Der nächste Schritt bringt die Darstellung von kurzen Texten. Die Umsetzung erfolgt analog zu **showChar**() durch eine Schleife. Der übergebene String in **s** wird durch die for-Schleife geparst und die vorgefundenen Zeichen wandern zum PCF8574.

```
def showString(s, delay=5):
    for char in s:
        showChar(char, delay)
```

Eine meiner shortest main loops ever bildet den Abschluss des Programms.

```
while 1:
    while abs(mpu.getValue("z")) < 2000:
        pass

    showString("FROHES FEST",3)

    if taste.value() == 0:
        showChar(" ",10)
        trigger.value(0)
        sys.exit()
```

Die innere while-Schleife wartet darauf, dass der MPU6050 ein Beschleunigungssignal meldet, das den Grenzwert von 2000 übersteigt, sowohl in positive, wie auch in negative Achsenrichtung. Welche Achse das bei Ihnen ist, hängt davon ab, wie Sie das GY-271-Modul eingebaut haben. Orientieren Sie sich dazu einfach über den Aufdruck auf dem Break Out Board. Für seitliche Bewegungen ist das bei mir die Z-Achse, denn das Modul steckt senkrecht zum Breadboard.

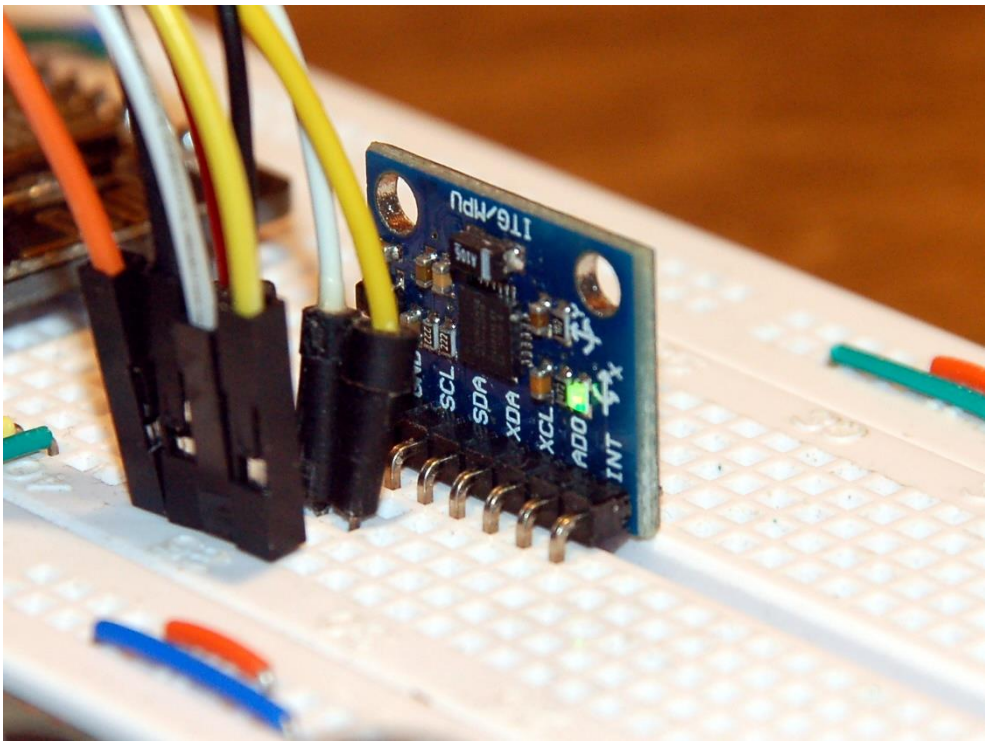

Abbildung 12: Z-Achse zeigt in Bewegungsrichtung

Dann folgt die Ausgabe des Strings und danach die Tastenabfrage. Ist die in diesem Moment gedrückt, dann machen wir die LEDs aus, stellen den Trigger-Ausgang auf 0 und verlassen das Programm.

Eine Überlegung zum Schluss

Wenn eine Spalte räumlich neben der vorigen angezeigt werden soll, muss eine Mindestgeschwindigkeit eingehalten werden. Für die 8mm-LEDs ist das in meinem

Ansatz mit 3ms Leuchtdauer: 8mm / 3ms = 2,67m/s, für die 3mm-LED nur 1m/s. Für die Anzeige des Strings "FROHES FEST" mit 11 • 7 = 77 Spalten ergeben sich schwebende Texte mit 77 • 8mm = 616mm beziehungsweise 231mm. Einen Eindruck davon, wie das aussieht, vermittelt die Abbildung 2.

Eine Aufgabe für „Jugend forscht“

In Abbildung 2 wird der Text in umgekehrter Buchstabenfolge angezeigt. Finden Sie eine Möglichkeit das zu ändern? Es ist im Prinzip recht einfach. Was liefert der MPU6050 für Werte, wenn die Richtung der Beschleunigung auf die Schaltung umgekehrt wird? Wie müssen Sie darauf im Programm reagieren? Letzten Endes fällt die Änderung dann doch recht umfangreich aus, weil nicht nur die Richtungsänderung der Bewegung, sondern auch das Parsen des Textes und die Darstellung der Spaltenmuster angepasst werden müssen. Viel Erfolg beim Programmieren von "Schütteltext 2.0"!

Und natürlich wünsche ich Ihnen eine schöne Adventszeit!