Engel blau auf rot

Dieser Beitrag ist auch als PDF-Datei verfügbar.

Als einfaches Projekt für die Weihnachtszeit bieten sich kleine Beleuchtungsprojekte an, die Stimmung machen. Dazu gehört eine ansprechende Mechanik und natürlich eine lichtspendende Hardware, die mittels eines Controllers und dem entsprechenden Programm gesteuert werden kann.

Für die Mechanik genügt hier ein DIN-A4-Blatt festes, weißes Papier oder besser Transparentpapier. Zur Beleuchtung verwenden wir einen Neopixelring und einen ESP32 oder einen ESP8266. Was wir damit anstellen können, verrate ich in der heutigen Folge aus der Reihe

MicroPython auf dem (ESP32) ESP8266 und Raspberry Pi Pico

heute

Stimmungsvolle Engellampe

Aufgrund seiner schnuckeligen Abmessungen ist ein ESP8266 D1 mini sehr gut geeignet. Zur Beleuchtung kann man einen 8-er oder 12-er Neopixelring nehmen. Beides platzieren wir auf einem Breadboard. Natürlich ist auch die Verwendung eines ESP32 oder eine Raspberry Pi Pico denkbar. An der Verdrahtung und am Programm ändert sich nichts.

Die Hardware

Ich verwende in diesem Beitrag einen ESP8266 D1 mini und einen 12-er Neopixelring.

1	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 oder NodeMCU Lua Amica Modul V2 ESP8266 ESP-12F WIFI Wifi Development Board mit CP2102 oder D1 Mini NodeMcu mit ESP8266-12F WLAN Modul kompatibel mit Arduino oder D1 Mini V3 NodeMCU mit ESP8266-12F
1	RGB LED Ring 8 bit WS2812 5050 + integrierter Treiber
1	Battery Expansion Shield 18650 V3 inkl. USB Kabel
1	Li-Po-Akkumulator vom Typ 18650
1	MB-102 Breadboard Steckbrett mit 830 Kontakten
1	Jumper Wire Kabel 3 x 40 STK. je 20 cm M2M/ F2M / F2F
optional	Logic Analyzer
1	Weißes Papier 160 bis 250 g/m² oder Transparentpapier Format DIN-A4

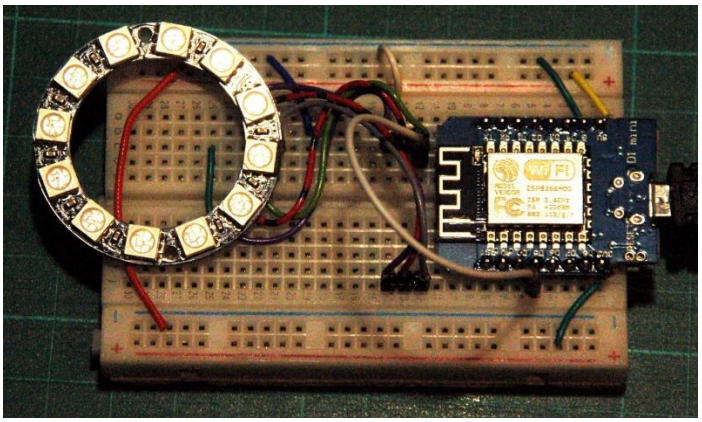

Abbildung 1: Lampen Hardware

Folgende Verbindungen müssen hergestellt werden:

Ring Vcc an ESP8266 5V
Ring GND an ESP8266 GND
Ring In an D5 = ESP8266 GPIO14

Die Software

Fürs Flashen und die Programmierung des ESP32:

Thonny oder
µPyCraft

Signalverfolgung:

Saleae Logic 2

Verwendete Firmware für einen ESP32:

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

Die MicroPython-Programme zum Projekt:

angel_light.py erstes Demoprogramm
soft_angel.py Betriebsprogramm der Weihnachtsbeleuchtung

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. Wie Sie den **Raspberry Pi Pico** einsatzbereit kriegen, finden Sie hier.

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 main.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.

Der Neopixelring

Neopixel-LEDs vom Typ WS2812 enthalten drei einzelne LEDs, die rotes, grünes oder blaues Licht abgeben. Angesteuert werden sie von einem Chip, der seine Anweisungen über eine Art Bussystem erhält, das mit 800kHz vom ESP32/ESP8266 getaktet wird.

Beim I2C-Bus oder beim SPI-Bus erreichen die Signale vom Controller, zum Beispiel einem ESP32, alle Slaves am Bus in gleicher Weise, alle sehen alles. Bei den WS2812-Bausteinen ist das anders. Jeder Baustein hat einen Dateneingang und einen Datenausgang. Mehrere Bausteine können kaskadiert werden, indem man den Dateneingang jedes weiteren Bausteins an den Datenausgang des Vorgängers

anschließt. Der erste Baustein in der Kette bekommt vom ESP32 eine Pulskette von drei Bytes für R, G und B zugespielt, die er selbst futtert, also vom gesamten Pulszug entfernt. Alle nachfolgenden Impulse werden durchgewunken und am Datenausgang abgegeben. Jeder Baustein in der Kette holt sich auf dieselbe Weise seinen Anteil und gibt den Rest weiter. So wird es möglich, dass jeder Baustein in der Kette individuell angesteuert werden kann. Die Intensität jeder Farbe kann in 256 Stufen variiert werden, je nach dem Wert des empfangenen Bytes für die einzelne Farbe. Wir geben den Farbcode in Form einer Liste oder eines Tupels mit drei Bytes für jeden WS2812 an.

Starten wir mit einem Versuch. Wir verwenden den Aufbau mit den oben angegebenen Verbindungen. Zuerst werden die Klassen **Pin** und **NeoPixel** importiert. Ich erzeuge ein GPIO-Pin-Objekt als Ausgang und instanziiere damit ein Neopixel-Objekt mit zwölf Bausteinen. Die NeoPixel-Instanz **neo** enthält ein Bytearray **buf** und die Methode **write**(), mit welcher der Inhalt des Bytearrays **buf** zum Neopixelring übertragen wird. Eingaben in REPL sind fett formatiert, Antworten vom ESP8266 sind kursiv dargestellt.

```
>>> from machine import Pin
>>> from neopixel import NeoPixel
>>> np=Pin(14,Pin.OUT)
>>> neo=NeoPixel(np,12)
>>> neo[0]=(0xe0,0x07,0x3c)
>>> neo[1]=(0xf0,0xf0,0xf0)
>>> neo.write()
>>> neo.buf
```

bytearray(b'\x07\xe0<\xf0\xf0\xf0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')

Mit **neo**[0] spreche ich die ersten drei Elemente des Arrays an und übergebe die Werte für rot, grün und blau, 0xe0, 0x07 und 0x3c. Intern macht das die private Funktion __**setitem**__(). In den Buffer werden die Werte in veränderter Reihenfolge eingetragen. Wie wir gleich am Kurvenzug sehen werden, den ich mit Hilfe des Logic Analyzers aufgezeichnet habe. Die Werte werden so gesendet, wie sie im Puffer stehen. Einer 0 entspricht ein schmaler Impuls von ca. 500ns Breite gefolgt von einer Pause von ca. 750ns. Die 1 wird durch einen Puls von 750ns und einer Pause von 500ns gesendet. Es fällt auf, dass die ersten beiden Bytes in vertauschter Reihenfolge gesendet werden. Das entspricht der Information aus dem Datenblatt und kann auch wie folgt verifiziert werden.

```
>>> neo.ORDER
(1, 0, 2, 3)
```

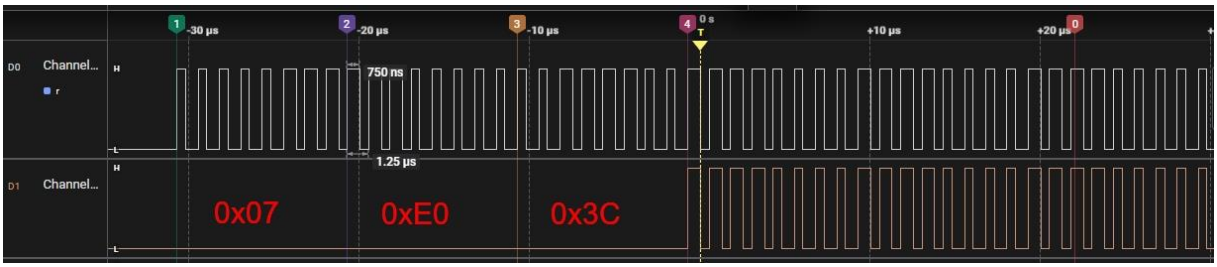

Abbildung 2: Pulsfolge für RGB = 0xE0, 0x07, 0x3C, 0xf0, 0xf0, 0xf0

Am Ausgang des ersten WS2812B fehlen die drei Bytes 0xe0, 0x07 und 0x3c, die dieser Baustein gefressen hat. Stattdessen kommt der Code 0xf0, 0xf0, 0xf0 zeitversetzt heraus, der Code für den zweiten Baustein. Der Eingang für Kanal 1 des Logic Analyzers war für diese Messung am Eingang der ersten LED, Kanal 2 am Ausgang derselben angeschlossen.

Jetzt wissen wir wie das Känguru beim WS2812B läuft und können uns später dem Programm für die Lampe zuwenden. Zuerst werfen wir aber einen Blick auf den Engel, den wir beleuchten wollen.

Der Engel für die Lampe

Das Lampengehäuse besteht aus einem Scherenschnitt, der zu einem Kegelstumpf gebogen wird.

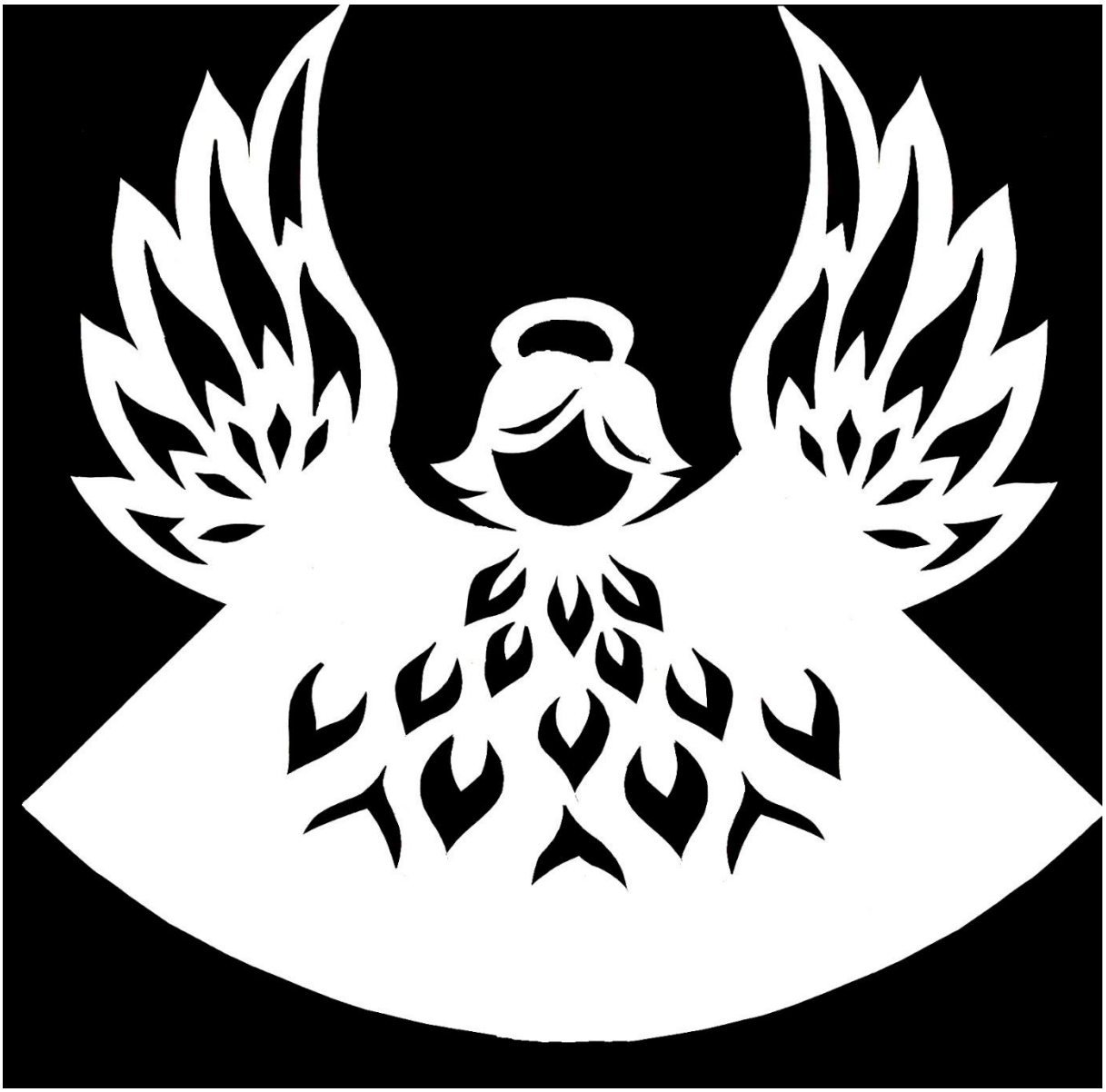

Abbildung 3: Engelvorlage fertig

Die Vorlage kann als PDF-Datei heruntergeladen werden. Die schwarzen Flächen in Abbildung 2 werden mit einem Cutter oder besser einem Design Messer herausgeschnitten. Wenn das erledigt ist, rollen wir das Papier auf, stecken die Flügel zusammen und verkleben die freie Rückseite mit der Klebelasche.

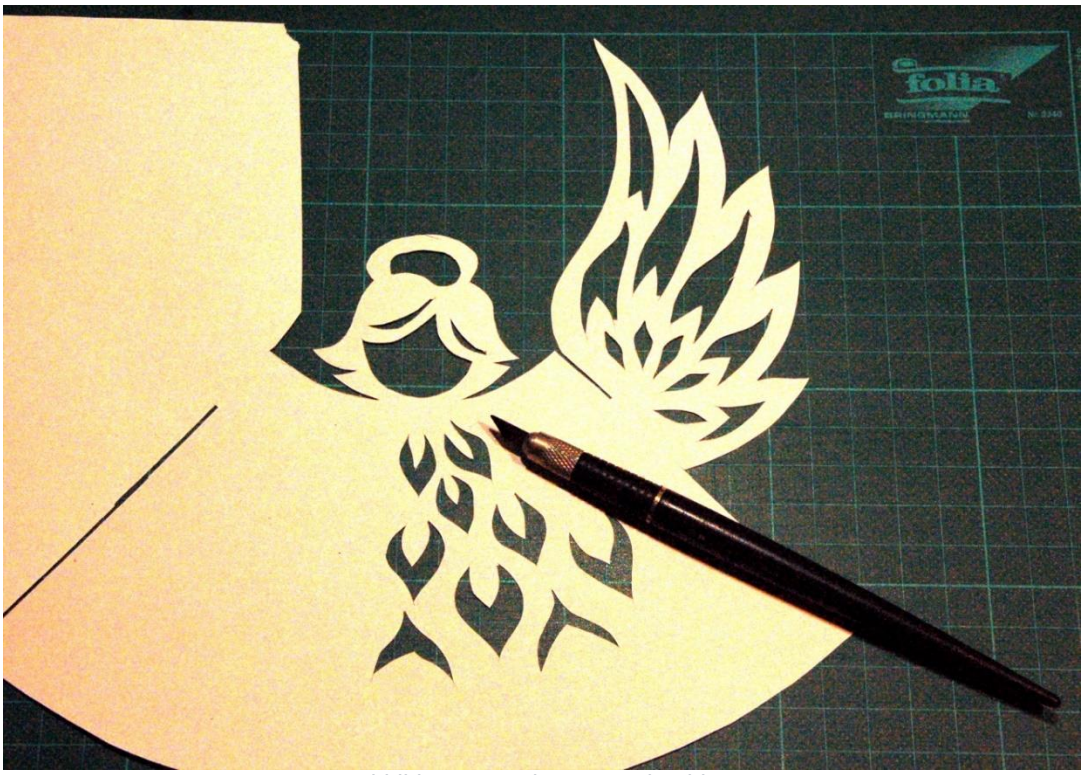

Abbildung 4: Vorlage ausschneiden

Abbildung 5: Engel in blau

Das Programm

Aller Anfang ist nicht schwer

Das erste Demoprogramm schaltet die Grundfarben durch, die wir mit dem Ring erreichen können.

```
# angel_light.py
#
# Pintranslator fuer ESP8266-Boards
# LUA-Pins     D0 D1 D2 D3 D4 D5 D6 D7 D8
# ESP8266 Pins 16  5  4  0  2 14 12 13 15
#                 SC SD
# used                        NE

from machine import Pin
from time import sleep
import neopixel

# An Pin GPIO14 liegt ein 12-er Neopixelring
n=neopixel.NeoPixel(Pin(14),12)

rt =(255,0,0)
gn =(0,255,0)
bl =(0,0,255)
ge =(255,255,0)
cy =(0,255,255)
pu =(255,0,255)
ws =(255,255,255)
sw =(0,0,0)

colors =(rt,gn,bl,ge,cy,pu,ws,sw)
for col in colors:
    color=col
    for i in range(12):
        # fill Buffer
        n[i] = color
    # Update the strip.
    n.write()
    sleep(3)
```

Wir holen die benötigten Module an Bord und instanziieren ein Neopixel-Objekt an Pin D5 = GPIO14. Ich habe hier einen 12-er -Ring verwendet.

Dann definiere ich die Tupel für die Grundfarben. In der äußeren Schleife lassen wir **col** die Elemente des Tupels **colors** durchlaufen, das sind die eben definierten Farbcodes.

In der inneren Schleife weisen wir den Farbcode an alle Buffer-Elemente des Rings zu. Wir schicken den Bufferinhalt an die Ringelemente und warten jeweils drei Sekunden.

Weiche Übergänge sind entspannender

Sanfte Übergänge sind für die „staade Zeit“ sicher besser angebracht als das Farben-Stakkato des letzten Prográmmchens. Also machen wir uns an die Arbeit, das Verhalten der Lampe zu glätten. Die Einträge in **colors** sind nach wie vor die Meilensteine. Um Übergänge zu schaffen, führen wir ein paar weitere Variablen ein. Mit **steps** legen wir die Anzahl der Schritte zwischen den Landmarks fest. Idealerweise wählen wir dafür eine Zweierpotenz. Größere Werte bescheren uns weichere Übergänge. Aus der Schrittanzahl berechnen wir die Schrittweite **step** = 256 // **steps**. Das ergibt stets eine ganze Zahl. Die Verweildauer für jeden Farbcode legen wir in **delay** ab. **steps** und **delay** definieren in etwa die Periodendauer für einen ganzen Durchlauf zu **delay** * **steps** Millisekunden.

Wir vergleichen immer die Komponenten des ersten Farbcodes mit denen des darauffolgenden. Auf den letzten Farbcode folgt im Tupel **colors** der Code mit dem Index 0. Wir brauchen für die Adressierung der Codes also einen Ringzähler. Den konstruieren wir mit Hilfe des Modulo-Operators. Wir benutzen die Anzahl der Farbeinträge in **colors** als Divisor einer Ganzzahldivision. Der Modulo-Operator **%** liefert den dann Teilungsrest, der zwischen 0 und **len(colors)** liegt.

Den Übergang am Ende des Tupels **color** zeigt folgendes Beispiel. Eingaben in REPL sind fett formatiert, Antworten vom ESP8266 sind kursiv dargestellt.

>>> colors =(rt,ge,gn,cy,cy,bl,pu,pu,ws,sw)
>>> len(colors)
10
>>> i = 8
>>> i += 1
>>> i
9
>>> i %= 10
>>> i
9
>>> i += 1
>>> i
10
>>> i %= 10
>>> i
0

Die **augmented assignments** (erweiterte Zuweisungen) i += 1 und i %= 10 stehen als Kurzformen für i = i + 1 und i = i % 10.

In einer Schleife, in der j von 0 bis **steps** – 1 läuft, berechnen wir aus den Komponenten der beiden Farbcodes **col1** und **col2** die aktuellen Werte. Dabei müssen wir drei Fälle unterscheiden.

a) Die Farbkomponenten der beiden Codes sind gleich.
In diesem Fall übernehmen wir den Komponentenwert in den aktuellen Farbcode.

b) Die Komponente des ersten Codes ist kleiner als dieselbe Komponente des Folgecodes.
In diesem Fall berechnen wir den Produktwert aus dem Schleifenindex j und der Schrittweite **step** und addieren ihn zum entsprechenden Wert der Komponente des ersten Farbcodes **col1**.

c) Die Komponente des ersten Codes ist größer als dieselbe Komponente des Folgecodes.
In diesem Fall berechnen wir den Produktwert aus dem Schleifenindex j und der Schrittweite **step** und subtrahieren ihn vom entsprechenden Wert der Komponente des ersten Farbcodes **col1**.

Weil die Komponenten neu zusammengestellt werden, können wir als Container kein Tupel verwenden, dieser Datentyp ist immutable. Das bedeutet, dass die Elemente nicht verändert werden können.

>>> color = (0,0,0)
>>> color[1] = 255
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item assignment

Auch das Erweitern eines Tupels mit **append**() ist nicht möglich.

>>> color = ()
>>> color
()
>>> color.append(255)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'

Aus diesen Gründen müssen wir zum Zusammensetzen des aktuellen Farbcodes in **color** eine Liste verwenden. Der Rest ist nun einfach. In einer weiteren for-Schleife übertragen wir den Farbcode in den Buffer des Neopixel-Rings und schicken den Buffer zum Ring. Danach legen wir den Controller für **delay** Millisekunden schlafen.

Zwei Tricks habe ich noch auf Lager.

Debugging:
Während der Programmentwicklung ist es oft hilfreich, an neuralgischen Stellen den Inhalt von Variablen in REPL auszugeben, was in der Produktionsphase nicht nötig oder gar unerwünscht ist. Letztlich beeinflussen solche Ausgaben auch das Zeitverhalten des Programms. Damit wir beim Debuggen nicht ständig die print-Zeilen einfügen und wieder löschen müssen, führen wir das Flag **debug** ein. Die Ausgabe soll nur erfolgen, wenn **debug True** ist.

```
if debug: print(i,k,name1,name2)
```

Sauberer Programmabbruch mit Strg + C

Wenn das Programm mit **Strg + C** an einer nicht vorhersehbaren Stelle unterbrochen wird, leuchten die LEDs weiter. Wir können dann natürlich jedes Mal den Buffer händisch mit (0,0,0)-Tupeln füllen und zum Ring schicken. Bequemer ist es, wenn wir den Keyboard Interrupt mit einer try-except-Konstruktion abfangen. Dann können wir die LEDs vom Programm abschalten lassen, eine Meldung ausgeben und das Programm mit **exit**() verlassen.

Hier kommt nun das gesamte Programm, dessen Elemente wir soeben besprochen haben.

```
# soft_angel.py
#
# Pintranslator fuer ESP8266-Boards
#LUA-Pins     D0 D1 D2 D3 D4 D5 D6 D7 D8 RX TX
#ESP8266 Pins 16  5  4  0  2 14 12 13 15  3  1
#                 SC SD
# Achtung      hi       hi hi         lo hi hi
# used                        NE

from machine import Pin
from time import sleep_ms
import neopixel
from sys import exit

# An Pin GPIO14 liegt ein 12-er Neopixelring
noc = 12
n=neopixel.NeoPixel(Pin(14),noc)

rt =(255,0,0)
gn =(0,255,0)
bl =(0,0,255)
ge =(255,255,0)
cy =(0,255,255)
pu =(255,0,255)
ws =(255,255,255)
sw =(0,0,0)

debug = False
colors =(rt,ge,gn,cy,cy,bl,pu,pu,ws,sw)
colnames = ("rt","ge","gn","cy","cy","bl","pu","pu","ws","sw")
steps = 64
step = 256 // steps
delay = 25 # ms

while 1:
    try:
        for i in range(len(colors)):
            col1=colors[i]
            name1=colnames[i]
            k=i+1
            k = k % len(colors)
            col2 = colors[k]
```

```
            name2=colnames[k]
            if debug: print(i,k,name1,name2)

            for j in range(steps):
                color=[0,0,0]

                for c in range(3):
                    if col1[c] == col2[c]:
                        color[c] = col1[c]
                    elif col1[c] < col2[c]:
                        color[c] = col1[c] + j * step
                    else :
                        color[c] = col1[c] - j * step
                if debug: print(color,col1,col2)
                for i in range(12):
                    # fill Buffer
                    n[i] = color
                # Update the strip.
                n.write()
                sleep_ms(delay)
    except KeyboardInterrupt:
        n.buf = bytearray( (0,0,0) * noc)
        n.write()
        print("Programm abgebrochen")
        exit()
```

Für einen autonomen Betrieb müssen wir das Programm unter dem Namen **main.py** in den Flash des Controllers hochladen. Beim nächsten Neustart startet das Programm dann selbständig. Ein Betrieb ist entweder mit Netzteil oder mit dem Batterieadapter und der Li-Ion-Zelle möglich. In letzterem Fall kann man bequem mit dem Schalter am Adapter den Aufbau an- und ausschalten.

Ausblick

In den nächsten Folgen verwandeln wir die weihnachtliche Engelleuchte in eine Ganzjahres-Tischlampe, die mit Sensortasten gesteuert werden kann. Dabei kommen ESP32, ESP8266 und Raspberry Pi Pico (W) in unterschiedlicher Umgebung zum Einsatz. Außerdem gibt es wieder diverse Tipps und Tricks zu MicroPython.

Eine besinnliche, staade Zeit und bis bald!