

Multiband-Spektrometer

Programmierprojekt des SS18
unter
Thorsten Wagener

Inhaltsverzeichnis

1. Die Projektgruppe	3
2. Die Aufgabe	3
3. Die Idee	3
4. Das Ziel	3
5. Kosten- / Materialplanung	3
6. Die Probleme & Lösungen	4
7. Code	5
8. Schaltplan	8
9. Fotos	9

1. Die Projektgruppe

Jan R.
David P.
Maria S.
Michelle Janine S.

2. Die Aufgabe

In Rahmen des Faches „Programmieren“ wurde uns eine Aufgabe mit folgenden Bedingungen gestellt:

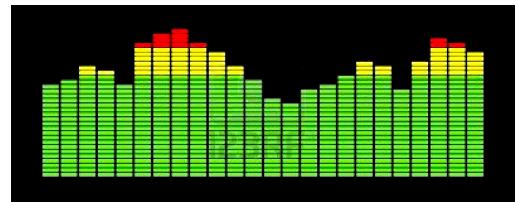
- Benutzung Raspberry Pi
- Entwicklung eines Codes in Python

3. Die Idee

Nach einigen Überlegungen sind wir zur Entscheidung gekommen, ein Spektrometer zu bauen. Dieses soll weniger als Messinstrument dienen und mehr als eine kleine visuelle Hilfe bei der Musikproduktion oder einfach als nette Visualisierung für den Home-Hifi-Gebrauch.

4. Das Ziel

Das ursprüngliche Ziel war ein über ein Raspberry Pi umgesetztes Multiband-Spektrometer mit ca. 20 Bändern. Dieses wird auf einem kleinen Monitor dargestellt, der eine kleine Benutzeroberfläche zum Anpassen einiger Einstellungen wie Farbe, Auflösung oder RMS/Peak-Metering bietet. Die Lautstärkeachse soll logarithmisch dargestellt werden, die Frequenzachse nach eigener Beurteilung gewichtet. Dies geschieht ebenfalls auf Basis einer logarithmischen Darstellung.



5. Kosten- / Materialplanung

ZEITPLAN

01.06 - 31.06: Projektplanung, Materialaquisition, Aufgabenverteilung
 01.07 - 31.07: Beginn Programmierung, Informieren über das Thema Audio-Analyse
 01.08 - 31.08: Programmierung, Tests, Interface
 01.09 - 15.10: Finalisierung, Protokoll

KOMPLIKATIONEN

Der ursprüngliche Plan involvierte noch ein Display. Da dieses nicht von der HAW bestellt wurde, würde der Plan geändert, siehe 6. Probleme & Lösungen.

DE-FACTO KOSTENÜBERSICHT

Teil	Kosten	Menge	Gesamt	
Raspberry Pi	0	1	0,00 €	-
USB-Audio-Adapter	15,00 €	1	15,00 €	https://www.reichelt.de/Soundkarten/ST-IC
8x8 WS2812B Matrix	0	2	0,00 €	-

6. Die Probleme & Lösungen

DAS DISPLAY

Problem:

Da die HAW das ausgesuchte Display nicht bestellt hatte, muss eine Alternative her.

Lösung:

Anstatt des Displays wurden zwei Diamedex 8x8-WS2812 LED-Matrizen verwendet. Damit wird zwar die Umsetzung eines UI nicht mehr möglich, stilistisch passt der Look jedoch. Damit verändert sich jedoch auch ein großer Teil der Programmierung, da jetzt die Bildinformation seriell über PWM ausgegeben wird und nicht per HDMI.

DIE ANSTEUERUNG DER MATRIX

Problem:

Die Pixelmatrix ist eigentlich eine lange LED-kette, die Schlangenlinienförmig auf einer PCB sitzt. Diese muss nun korrekt angesteuert werden.

Lösung:

Die Helligkeits-Information für jede LED wird zunächst in einer 3D-Matrix (x-Position, y-Position, r/g/b) hinterlegt und dann mit aufsteigender Reihenfolge ausgelesen, Spalte für Spalte von links nach rechts. Da die LEDs in Schlangenlinien verbunden sind, wird jede zweite Spalte rückwärts ausgelesen.

DIE PEGELDIFFERENZ

Problem:

Der Raspberry gibt bei seinen IO-Pins nur eine Spannung von 3,3V aus. Die WS2812-LEDs benötigen jedoch eine Eingangs-Spannung von 5V. Daher muss eine Lösung her, die den Pegel bei einem PWM-Signal auf 80kHz wandelt.

Lösung:

Es wurde ein 74HCT-Doppelinverter-IC verwendet, um das Signal mithilfe von zwei gekoppelten Invertern hochzustufen. Ein Test mit dem Oszilloskop zeigte eine relativ gute Flankensteilheit, die beispielsweise ein 5V-Pegelwandler-IC nicht bieten konnte.

DIE LATENZ

Problem:

Das Spektrometer funktioniert nur bei Buffergrößen über 2000. So ist es zwar benutzbar aber noch nicht perfekt.

Lösung:

Die zu große Buffergröße liegt weder an einem restlichen Teil des Codes noch an einer Überlastung des RasPi. Daher ist die einzige mögliche Lösung ein Wechsel des Audiotreibers. Dies wäre zu viel Aufwand gewesen, daher ist dieses das einzige bis dato ungelöste Problem.

AUTOSTART

Problem:

Damit das Gerät auch transportiert oder nach einem Crash neu gestartet werden kann, muss der RasPi nach erhalten einer Stromversorgung ganz alleine das Programm ausführen, damit keine weiteren Peripherie-Geräte zum Start notwendig sind.

Lösung:

Der Datei /etc/rc.local wurde die Zeile

sudo python /Desktop/spectrum/spectrum-16band.py hinzugefügt.

Damit wird bei jedem Systemstart automatisch der Code auf Root-Level ausgeführt.

DER AUDIOTREIBER

Problem:

Der PWM-Pin funktioniert bei aktivem RasPi Audiotreiber nicht, da beide PWM benötigen.

Lösung:

Der Audiotreiber muss deaktiviert werden. Doch sowohl ein Blacklisting der snd_bcm2835 als auch ein generelles Deaktivieren des Audiotreibers führte zu Fehlern im pyaudio-Code (duh). Dennoch ermöglichte dieser Vorgang ein permanentes Umschalten auf USB-Audio anstatt des Internen Ausgangs, was das Problem ebenfalls behob.

7. Code

```
#System-Import
import sys
import numpy as np #fuer fft-analyse
from struct import unpack #zum interpretieren der daten
import time
import argparse
import random
import math
import pyaudio #fuer audiointerfacing
from neopixel import *

#Haupt-Matrix anlegen, in der der displayinhalt aufgebaut wird
#2 dimensionen fuer die pixel, 3. dimension fuer die farbe
c, w, h = 3, 8, 16;
Matrix = [[[0 for x in range(w)] for x in range(w)] for y in range(h)]

# LED strip config
LED_COUNT      = 128
LED_PIN         = 18          #hier wird IO-Pin 18 definiert
LED_FREQ_HZ    = 800000
LED_DMA         = 10
LED_BRIGHTNESS = 50
LED_INVERT      = False
LED_CHANNEL     = 0

#Definitionen
no_channels = 1
sample_rate = 44100
device = 2
chunk = 2822 #vielfaches von 8! 3072
movement=0

#Listen generieren
switched = [0,2,4,6,8,10,12,14] # umgedrehte reihen, da die pixelmatrix
schlangenlinienfoermig ist
spectralvalues = [1,2,3,4,5,6,7,8,7,6,5,4,3,2,1,0]
matrix = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
power = []
weighting = [3,3,4,4,6,8,8,12,16,16,35,35,64,64,64,64] #weighting der
spektralbaender

#ZITAT: Power Array Index Funktion
def piff(val):
    return int(2*chunk*val/sample_rate)
#ZITAT Ende

#Audio Setup pyaudio mit dem USB Interface
p = pyaudio.PyAudio()
stream = p.open(
    format = pyaudio.paInt16,
    channels = no_channels,
    rate = sample_rate,
    input = True,
    frames_per_buffer = chunk,
    input_device_index = device)

def calculate_levels(data, chunk,sample_rate):
    #ZITAT: Umwandlung des Streams in eine Matrix
    global matrix
```

```

#ascii-daten in numpy array stecken
data = unpack("%dh"%(len(data)/2),data)
data = np.array(data, dtype='h')
#FFT anwenden
fourier=np.fft.rfft(data)
#Array zuschneiden
fourier=np.delete(fourier,len(fourier)-1)
#amplitude pro band berechnen
power = np.abs(fourier)
#ZITAT Ende
#einzelne baender berechnen
matrix[0]= np.mean(power[piff(0) :piff(17):1])
matrix[1]= np.mean(power[piff(17) :piff(28):1])
matrix[2]= np.mean(power[piff(42) :piff(69):1])
matrix[3]= np.mean(power[piff(69) :piff(110):1])
matrix[4]= np.mean(power[piff(110) :piff(180):1])
matrix[5]= np.mean(power[piff(180) :piff(290):1])
matrix[6]= np.mean(power[piff(290) :piff(480):1])
matrix[7]= np.mean(power[piff(480):piff(750):1])
matrix[8]= np.mean(power[piff(750) :piff(1100):1])
matrix[9]= np.mean(power[piff(1100) :piff(2000):1])
matrix[10]= np.mean(power[piff(2000) :piff(3300):1])
matrix[11]= np.mean(power[piff(3300) :piff(5500):1])
matrix[12]= np.mean(power[piff(5500) :piff(7000):1])
matrix[13]= np.mean(power[piff(7000) :piff(8000):1])
matrix[14]= np.mean(power[piff(8000) :piff(10000):1])
matrix[15]= np.mean(power[piff(10000) :piff(20000):1])
#aufraeumen
matrix=np.divide(np.multiply(matrix,weighting),100)
matrix=np.log10(matrix)
matrix=matrix-3
matrix=matrix*3
#Werte auf 0-8 beschraenken und zurueckgeben
matrix=matrix.clip(0,8)
return matrix

#MAIN LOOP
if __name__ == '__main__':
    #Argumente
    parser = argparse.ArgumentParser()
    parser.add_argument('-c', '--clear', action='store_true', help='display clear')
    args = parser.parse_args()

    #NeoPixel initialisieren
    strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA,
LED_INVERT, LED_BRIGHTNESS, LED_CHANNEL)
    strip.begin()

    print ('Ctrl-C zum beenden')

    #hauptschleife
    try:
        while True:

            #Audio-Stream in Chunks kopieren
            data = stream.read(chunk)

            #Matrix-Audiolevel berechnen lassen
            spectralvalues=calculate_levels(data, chunk,sample_rate)

```

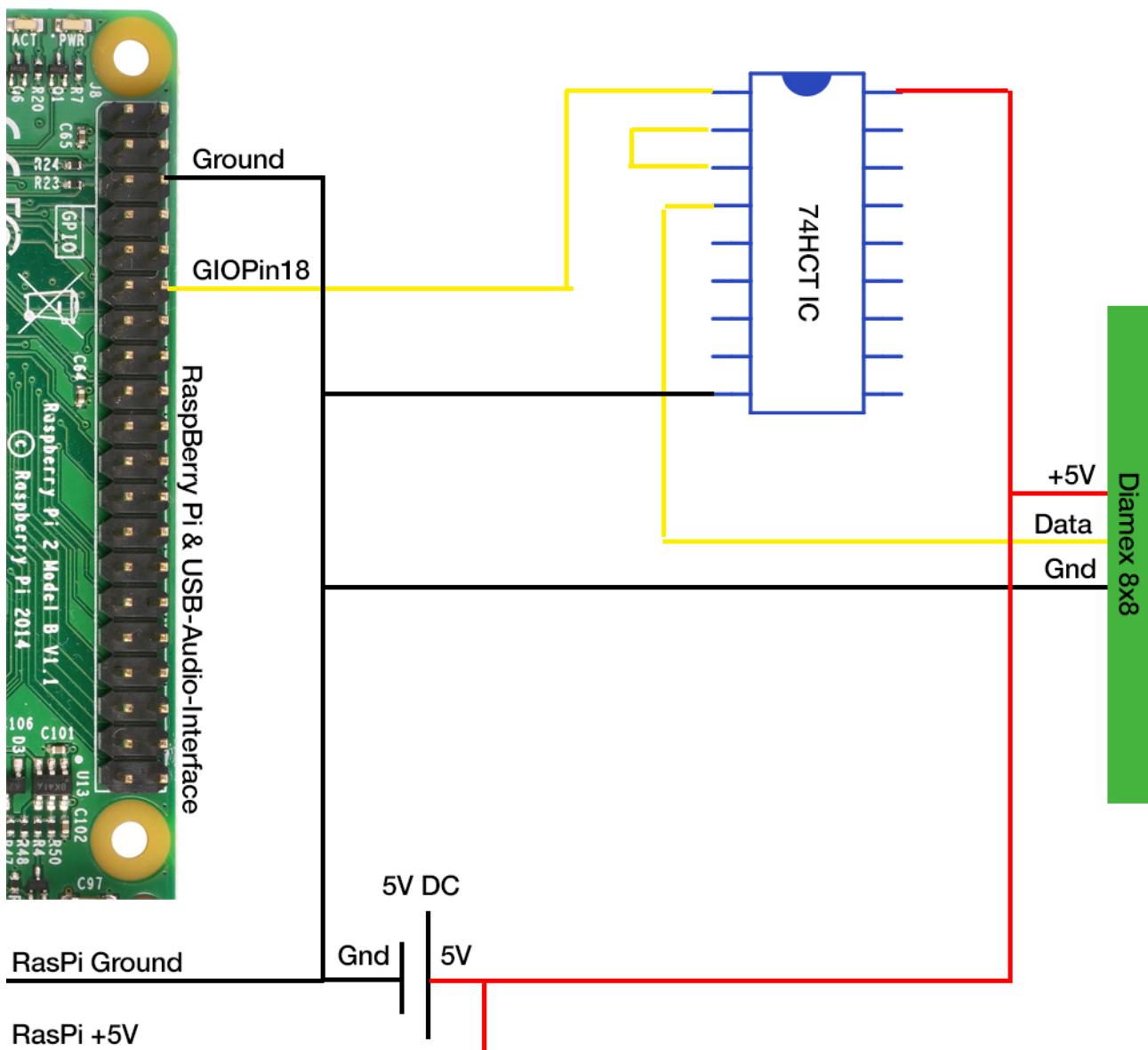
```
#Matrix auswerten
for x in range (0,16):
    for y in range (0,8):
        #volle pixel zeigen den wert
        if (spectralvalues[x]-y)>1:
            Matrix[x][y][2] = 255
            Matrix[x][y][1] = int(y*255/7)
            #Matrix[x][y][1] = 0
        #rest wird in der helligkeit des obersten pixels dargestellt
        elif (spectralvalues[x]-y)<1 and (spectralvalues[x]-y)>0:
            Matrix[x][y][2] = int((spectralvalues[x]-y)*255)
            Matrix[x][y][1] = int((spectralvalues[x]-y)*255)
        else:
            Matrix[x][y][0] = 0
            Matrix[x][y][1] = 0
            Matrix[x][y][2] = 0

#matrix auf den strip uebertragen
for x in range (0,16):
    for y in range (0,8):
        #leds sind in schlangenlinien, daher jede zweite reihe invers
        if x in switched:
            p = 8*x + y
        else:
            p = 8*x + (7-y)
        strip.setPixelColorRGB(p,Matrix[x][y][0],Matrix[x][y]
[1],Matrix[x][y][2])

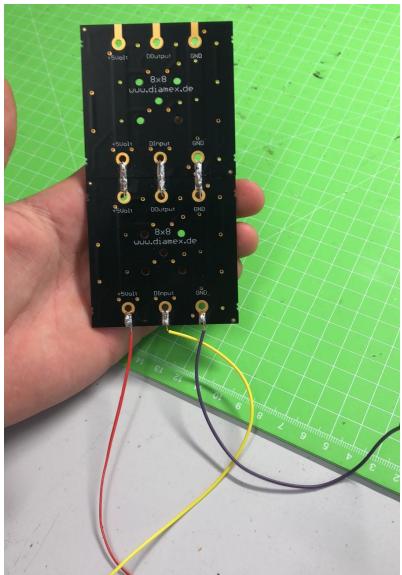
    strip.show()

except KeyboardInterrupt:
    if args.clear:
        for x in range (0,16):
            for y in range (0,8):
                strip.setPixelColorRGB(p,0,0,0)
```

8. Schaltplan



9. Fotos



Die verlötzten LED-Panels

```

examples
pi@raspberrypi:~/rpis_ws281x/python/exa
File Edit Tabs Help
set pixel 60 to R: 0 G: 136 B: 145
set pixel 68 to R: 0 G: 145 B: 145
set pixel 76 to R: 0 G: 160 B: 145
set pixel 84 to R: 0 G: 175 B: 145
set pixel 92 to R: 0 G: 190 B: 145
set pixel 100 to R: 0 G: 205 B: 145
set pixel 108 to R: 0 G: 220 B: 145
set pixel 116 to R: 0 G: 235 B: 145
set pixel 124 to R: 0 G: 250 B: 145
set pixel 5 to R: 0 G: 25 B: 175
set pixel 13 to R: 0 G: 40 B: 175
set pixel 21 to R: 0 G: 55 B: 175
set pixel 29 to R: 0 G: 70 B: 175
set pixel 37 to R: 0 G: 85 B: 175
set pixel 45 to R: 0 G: 100 B: 175
set pixel 53 to R: 0 G: 115 B: 175
set pixel 61 to R: 0 G: 130 B: 175
set pixel 69 to R: 0 G: 145 B: 175
set pixel 77 to R: 0 G: 160 B: 175
set pixel 85 to R: 0 G: 175 B: 175
set pixel 93 to R: 0 G: 190 B: 175
set pixel 101 to R: 0 G: 205 B: 175
set pixel 109 to R: 0 G: 220 B: 175
set pixel 117 to R: 0 G: 235 B: 175
set pixel 125 to R: 0 G: 250 B: 175
set pixel 6 to R: 0 G: 25 B: 265
set pixel 14 to R: 0 G: 40 B: 265
set pixel 22 to R: 0 G: 55 B: 265
set pixel 30 to R: 0 G: 70 B: 265
set pixel 38 to R: 0 G: 85 B: 265
set pixel 46 to R: 0 G: 100 B: 265
set pixel 54 to R: 0 G: 115 B: 265
set pixel 62 to R: 0 G: 130 B: 265

```

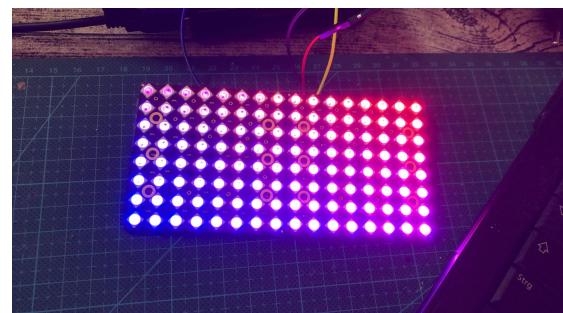
Debug-Ausgaben

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																		
2																		
3																		
4	<u>X-achse</u>	<u>Y-achse</u>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	7	0	7	8	23	24	39	40	55	56	71	72	87	88	103	104	119	120
6	6	1	6	9	22	25	38	41	54	57	70	73	86	89	102	105	118	121
7	5	2	5	10	21	26	37	42	53	58	69	74	85	90	101	106	117	122
8	4	3	4	11	20	27	36	43	52	59	68	75	84	91	100	107	116	123
9	3	4	3	12	19	28	35	44	51	60	67	76	83	92	99	108	115	124
10	2	5	2	13	18	29	34	45	50	61	66	77	82	93	98	109	114	125
11	1	6	1	14	17	30	33	46	49	62	65	78	81	94	97	110	113	126
12	0	7	0	15	16	31	32	47	48	63	64	79	80	95	96	111	112	127
13																		

Die LED-Matrix durchnummeriert.



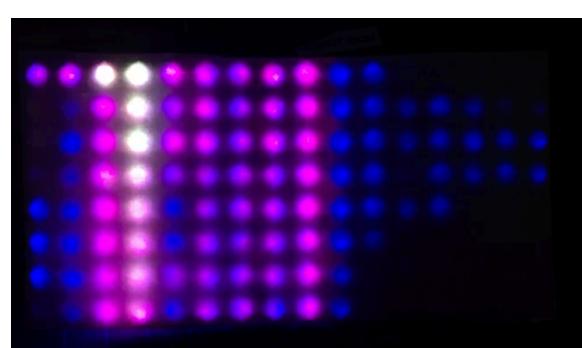
Das Gehäuse



LED-Test



Der Multibandanalyser



Der Wasserfallanalyser