

Utilizarea RPi ca analizor spectral audio

1. Sumar	2
2. Descrierea detaliata a proiectului	2
3. Implementare	7
4. Lista de componente/subansamble/soft-uri	10
5. Concluzii	10
6. Bibliografie	11

1. Sumar

Un analizor spectral audio este un instrument care permite analiza spectrului frecventelor intr-un semnal audio. Acest lucru poate fi util pentru a determina componentele frecventelor dintr-un semnal audio, cum ar fi voci, instrumente sau sunete de fundal.

Există diverse exemple ce justifică folosirea unui astfel de sistem. Muzicienii pot folosi un analizor spectral audio pentru a analiza înregistrările audio ale melodiei lor pentru a identifica probleme de calitate sau pentru a ajusta mixajul. Inginerii audio pot folosi acest instrument pentru a evalua calitatea semnalelor audio în diferite medii de înregistrare. Cercetatorii pot folosi un analizor spectral audio pentru a studia caracteristicile auditive ale diferitelor specii.

În continuare voi detalia cum se poate implementa un astfel de dispozitiv hardware folosind un micro-computer Raspberry Pi.

2. Descrierea detaliată a proiectului

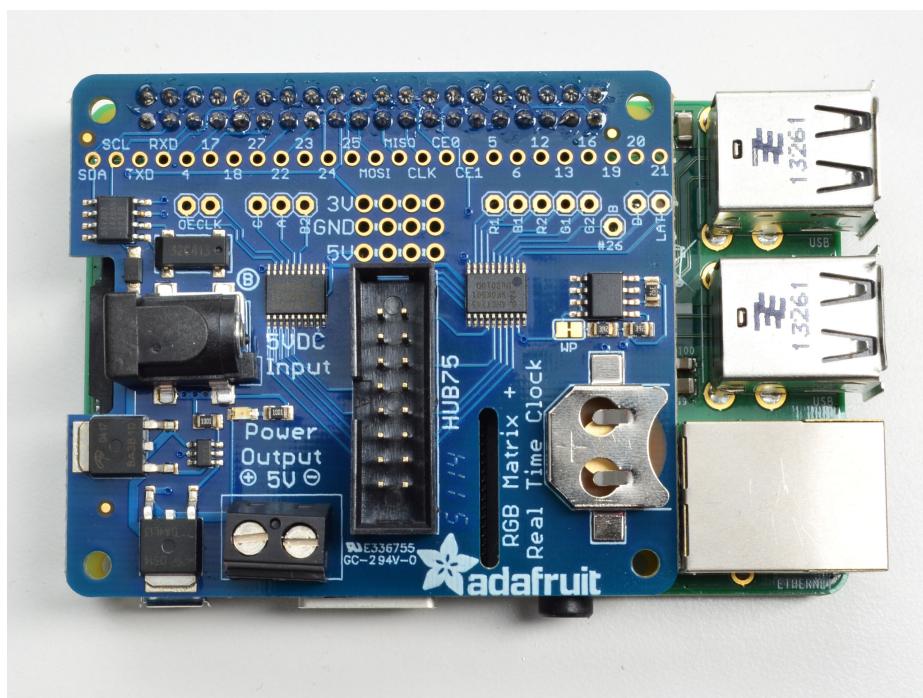
Principiul de baza care sta la realizarea unui analizor spectral audio este analiza Fourier. Avem nevoie să convertim semnalul audio care este periodic în domeniul timpului în domeniul frecvenței. Transformarea Fourier poate fi utilizată pentru a analiza aceste semnale și a extrage frecvența respectiva lor. În special ne vom folosi de algoritmul cunoscut drept Fast Fourier Transformation (FFT) deoarece aplicația noastră rulează în timp real și avem nevoie de un grad de eficiență și optimizare ridicat. Obținând datele transformate în noul domeniu discret dorim să le logaritmăm pentru a rămâne într-un interval concordant cu nevoile noastre.

Având aceste date este necesar să ne conectăm la un panou bidimensional de diode emisitoare de lumina (LED-uri) RGB într-o configurație cunoscută, iar pentru fiecare semnal în parte să asignăm o culoare corespunzătoare cu nivelul amplitudinii semnalului. Asadar vom transmite coloanei de LED-uri x informație ca ele să se aprindă în funcție de valorile semnalului transformat retinut matrice la elementul matrice[x].

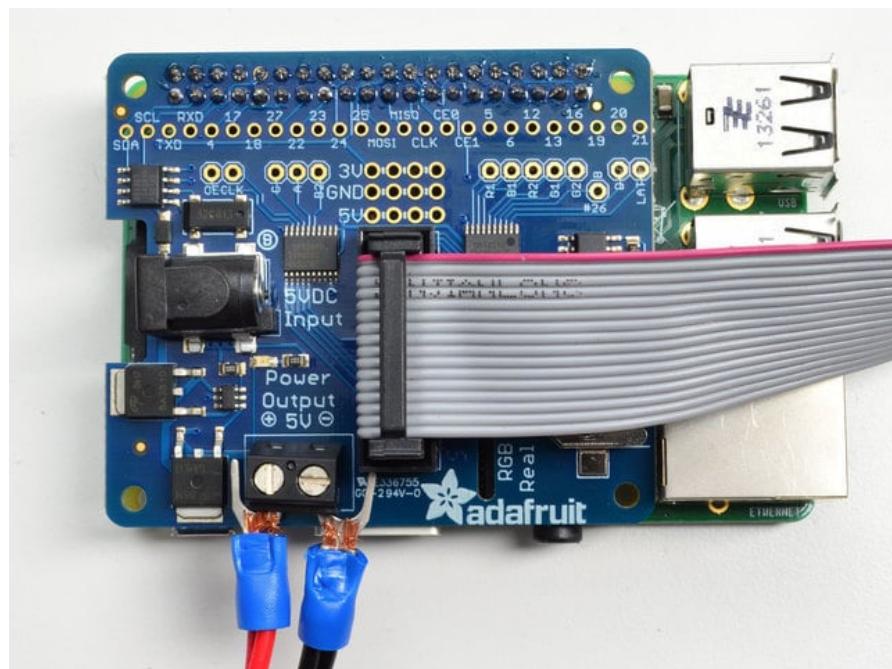
Acet lucru trebuie repetat in secente de lungime cunoscuta pentru intreaga durata a semnalului, adica `durata_totala / lungime_secventa` pentru fisiere aflete local pe harddisk, sau continuu pana la intreruperea programului de catre utilizator in cazul preluarii semnalului din microfon.

Pentru comoditate vom folosi un modul Hardware Attached on Top + Real Time Clock (HAT + RTC) care ne va da posibilitatea de a adresa LED-urile matricei ($16 \times 32 = 512$) individual prin multiplexare fara a fi nevoie de a ne ingrijora despre acest lucru la nivel de cod. De asemenea ne ofera capabilitati de a tine cont de timp in mod exact daca am atasat acestui modul o baterie de tip CR1220, dar acest lucru nu este necesar pentru capabilitatile pe care dorim sa le folosim in cadrul acestui proiect.

In primul rand modulul HAT se conecteaza la ambele coloane de 20 de pini ai micro-computer-ului Raspberry Pi folosind conectorul inclus.



Apoi se realizeaza conexiunea dintre matricea de LED-uri RGB si modulul HAT pentru controlul de curent utilizand conectorul inclus de tip MOLEX in matrice respectiv conectorul de tip spade in HAT. Inainte ca matricea si Raspberry Pi-ul sa poata fi conectate la curent trebuie sa se mai realizeze conexiunea pentru a transmite date catre matrice folosind pinii IDC ai modulului si un cablu data.



In final, HAT-ul si Raspberry Pi-ul pot fi conectate la sursele respective de acumulare.

Scriptul principal de executie:

```

import alsaaudio as aa
import wave
import pyaudio
from struct import unpack
import numpy as np
from rgbmatrix import RGBMatrix, RGBMatrixOptions, graphics
import sys

def calculate_levels(data, chunk):
    # aplică transformarea FTT pe toate datele matricii la momentul actual
    data = unpack("%dh" % (len(data) / 2), data)
    data = np.array(data, dtype="h")
    fourier = np.fft.rfft(data)
    fourier = np.delete(fourier, len(fourier) - 1)
    power = np.log10(np.abs(fourier)) ** 2
    power = np.reshape(power, (32, chunk / 32))
    matrix = np.int_(np.average(power, axis=1))
    return matrix

def display_loop():
    Dmatrix.Clear()
    for y in range(32):
        
```

```

for x in range(matrix[y]):
    x *= 2
    # default alb
    r, g, b = 255, 255, 255
    if x < 33:
        # verde
        r, g, b = 0, 132, 80
    elif x < 50:
        # galben
        r, g, b = 239, 183, 0
    else:
        # rosu
        r, g, b = 184, 29, 19
    Dmatrix.SetPixel(y, x, r, g, b)
    Dmatrix.SetPixel(y, x - 1, r, g, b)
return None

def spectrum_fisier(path):
    wavfile = wave.open(path)
    sample_rate = wavfile.getframerate()
    # obiectul de output folosit pentru a reda sunetul pentru a fi auzit in timp real
    output = aa.PCM(aa.PCM_PLAYBACK, aa.PCM_NORMAL)
    output.setchannels(2)
    output.setrate(sample_rate)
    output.setformat(aa.PCM_FORMAT_S16_LE)
    output.setperiodsize(chunk)

    data = wavfile.readframes(chunk)
    while data != "":
        output.write(data)
        matrix = calculate_levels(data, chunk)
        display_loop(matrix)
        data = wavfile.readframes(chunk)
    return None

def spectrum_microfon():
    no_channels = 1
    sample_rate = 44100
    device = 2
    p = pyaudio.PyAudio()
    stream = p.open(
        format=pyaudio.paInt16,

```

362 CTI

Licu Mihai-George

```
channels=no_channels,
rate=sample_rate,
input=True,
frames_per_buffer=chunk,
input_device_index=device,
)

# loop infinit care citeste audio data de la microfon si updateaza matricea cu
noile informatii
while 1:
    try:
        data = stream.read(chunk)
        matrix = calculate_levels(data, chunk)
        Dmatrix.Clear()
        display_loop(matrix)
    except KeyboardInterrupt:
        print("Se opreste...")
        stream.stop_stream()
        stream.close()
        p.terminate()
        sys.exit(1)
    except Exception as e:
        print(e)
        print("Eroare...")
        stream.stop_stream()
        stream.close()
        p.terminate()
        sys.exit(1)

    return None

# interfata pentru a comunica cu matricea de leduri si a updatea pixelii acestia
options = RGBMatrixOptions()
options.rows = 16
options.cols = 32
options.chain_length = 1
options.parallel = 1
options.hardware_mapping = "adafruit-hat"
Dmatrix = RGBMatrix(options=options)
Dmatrix.Clear()

# reprezinta dimensiunea portiei pe care o citim din bufferul audio
```

```

chunk = 16 * 32

# reprezentarea doi dimensională a matricei de LEDuri
matrix = [0 for _ in range(32)]


if __name__ == "__main__":
    if len(sys.argv) == 2:
        if sys.argv[1] == "mic":
            print("CTRL+C pentru a termina executia")
            spectrum_microfon()
        else:
            spectrum_fisier(sys.argv[1])
    else:
        print(f"Eroare utilizare. Utilizare corecta: {sys.argv[0]} <optiune> unde
optiune = mic SAU calea spre fisierul audio dorit.")

```

3. Implementare

Deoarece este nevoie de un număr mare de operații complexe într-o succesiune rapidă am ales modelul Raspberry Pi 4 B varianta cu 4GB pentru a ma asigura că există destulă putere computatională pentru a conduce toate cele 512 LED-uri fără probleme.

Matricea trebuia să fie o marime destul de mare pentru a reprezenta o gama de frecvențe largă dar totuși am considerat că nu este nevoie de foarte multe diode pentru nivelul de înalțime al fiecarui semnal, de aceea am ales o matrice de 16x32. De asemenea, ca matricea să fie compatibilă cu modulul HAT trebuie să fie o varianta cu LED-urile legate în catod comun și de tip pinout HUB75 pentru ca LED-urile să fie adresate corect. Aceasta matrice de asemenea are nevoie de alimentare separată deoarece Pi-ul nu o poate alimenta din cauza curentilor mari. Alimentarea fiind făcută prin intermediul modulului HAT.

Am ales acest modul HAT+RTC deoarece oferă o comoditate ridicată pentru lucrul cu matrici de LED-uri, realizează multiplexarea automat fără a fi nevoie de modificări în cod sau legături manuale, convertește automat tensiunea de logica de 3.3V a Raspberry Pi-ului la 5.0V pentru a conduce matricea fără probleme sau intreruperi, și este complet plug-and-play nefiind nevoie de reconfigurări.

Pentru a putea analiza spectrograma sunetului provenit din microfon avem nevoie ca microfonul folosit sa fie conectat la Pi prin interfata USB. Microfoanele cu conexiune analogica nu pot fi utilizate.

Pentru functionarea corecta a scriptului este nevoie de urmatoarele biblioteci:

- alsaaudio: este folosit pentru a reda sunetul procesat cu scopul de a fi auzit de utilizator in timp real
- wave: necesar pentru a accesa si a citi informatiile din fisierul audio dorit pentru procesare
- unpack: este folosit pentru a converti informatie binara din format audio in numere in virgula mobila pe care se pot executa operatii aritmetice
- numpy: biblioteca matematica folosita pentru a usura calcului, de asemenea aduce implementari gata de a fi folosite pentru algoritmul FFT
- pyaudio:
- rgbmatrix: controlarea ledurilor

Pentru a instala aceste biblioteci dorim sa rulam comanda:

```
pip install alsaaudio numpy pyaudio
```

Biblioteca rgbmatrix este nevoie sa fie instalata in urmatorul mod:

```
curl -O  
https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/rgb-matrix.sh  
sudo bash rgb-matrix.sh
```

Ruland script-ul dorim sa selectam optiunile HAT+RTC, CONVENIENCE si nu avem nevoie de modul RTC.

```
This script installs software for the Adafruit  
RGB Matrix Bonnet or HAT for Raspberry Pi.  
Steps include:  
- Update package index files (apt-get update)  
- Install prerequisite software  
- Install RGB matrix driver software and examples  
- Configure boot options
```

362 CTI

Licu Mihai-George

Run time ~15 minutes. Some options require reboot.

EXISTING INSTALLATION, IF ANY, WILL BE OVERWRITTEN.

CONTINUE? [y/n] y

Select interface board type:

1. Adafruit RGB Matrix Bonnet
2. Adafruit RGB Matrix HAT + RTC

SELECT 1-2: 2

Install realtime clock support? [y/n] n

Now you must choose between QUALITY and CONVENIENCE.

QUALITY: best output from the LED matrix requires commandeering hardware normally used for sound, plus some soldering. If you choose this option, there will be NO sound from the audio jack or HDMI (USB audio adapters will work and sound best anyway), AND you must SOLDER a wire between GPIO4 and GPIO18 on the Bonnet or HAT board.

CONVENIENCE: sound works normally, no extra soldering. Images on the LED matrix are not quite as steady, but maybe OK for most uses. If eager to get started, use 'CONVENIENCE' for now, you can make the change and reinstall using this script later!

What is thy bidding?

1. Quality (disables sound, requires soldering)
2. Convenience (sound on, no soldering)

SELECT 1-2: 2

Interface board type: Adafruit RGB Matrix HAT + RTC

Install RTC support: NO

Optimize: Convenience (sound on, no soldering)

CONTINUE? [y/n]

Acum biblioteca este instalata si putem rula un demo din ea pentru a verifica conectivitatea corecta.

```
sudo ./demo -D0 --led-rows=16 --led-cols=32 --hardware-mapping=adafruit-hat
```

Exemple de rulare a scriptului

```
./script.py mic
./script.py fisier_audio.wav
```

4. Lista de componente/subansamble/soft-uri

Nume	Caracteristici	Adresa de achizitie	Pret Unitar	Numar Bucati	Pret Total
Raspberry Pi 4 B	4GB RAM, 4 nuclee x64, 1.5 GHz	link	301,98	1	301,98
Alimentator	USB Tip C/MicroUSB 5V 3A	link	37,99	2	75,98
Matrice LED RGB	LEDuri 16x32, 5V putere intrare regulata, 170g, densitate pixeli 6mm	link	115,51	1	115.51
RGB Matrix HAT + RTC	65mm x 57mm x 2mm	link	175	1	175

Total general: 668.47

5. Concluzii

In concluzie, acest sistem ajunge sa coste mai mult decat optiunie de cost scazut care sunt deja disponibile pe piata, dar spre deosebire de acestea, ofera avantajul de a putea fi customizat si adaptat in functie de nevoile specifice ale utilizatorului, avand un potential mai mare de a se

dezvolta si de a evolu, si fiind mult mai flexibil si util in a face fata diferitelor cerinte. HAT-ul din sistem poate conduce o matrice de LED-uri pana la 64x64 fara a necesita nici o alta modificare. Utilizand modulul RTC putem folosi exact acelasi sistem pentru a configura o multitudine de diferite aplicatii, de exemplu un ceas cu alarma digital, afisarea notificarilor in timp real, capabilitati de cronometrizare, etc.

Astfel, acest sistem este suficient pentru a realiza un analizator spectral audio de inalta calitate intr-un mod cat mai usor si accesibil si ofera posibilitati pentru cei care doresc o solutie personalizata si flexibila pentru diverse aplicatii utilizand o matrice LED.

6. Bibliografie

[https://www.hackster.io/gatoninja236/raspberry-pi-audio-spectrum-di splay-1791fa#things](https://www.hackster.io/gatoninja236/raspberry-pi-audio-spectrum-display-1791fa#things)

[https://learn.adafruit.com/adafruit-rgb-matrix-plus-real-time-clock-hat- for-raspberry-pi](https://learn.adafruit.com/adafruit-rgb-matrix-plus-real-time-clock-hat-for-raspberry-pi)

<https://www.rototron.info/raspberry-pi-spectrum-analyzer/>

[https://www.tomshardware.com/news/raspberry-pi-pico-audio-spectr um-visualizer](https://www.tomshardware.com/news/raspberry-pi-pico-audio-spectrum-visualizer)

[https://www.instructables.com/Raspberry-Pi-Spectrum-Analyzer-with- RGB-LED-Strip-/](https://www.instructables.com/Raspberry-Pi-Spectrum-Analyzer-with-RGB-LED-Strip-/)

https://en.wikipedia.org/wiki/Fast_Fourier_transform