Logo

Description automatically generated

Documentație Proiect PMP

Music Reactive LED Strip

Student: Boloș Andrei-Nicolae

Grupa 3023**6**

Cuprins

[Introducere 3](#_Toc122516076)

[Bibliografie 3](#_Toc122516077)

[Soluții și Implementare 3](#_Toc122516078)

[Testare 4](#_Toc122516079)

[Exemple Demonstrative. 4](#_Toc122516080)

[Concluzii 5](#_Toc122516081)

# Introducere

Acest proiect este o tentativă de a reprezenta vizual undele sonore receptate de un senzor de sunet (muzica), în funcție de frecvența sunetelor, intenitatea lor și de variația unui potențiometru.

Neavând cunoștințele necesare pentru îndeplinirea acestui proiect, am folosit inspirația datorată vizionării unor videoclipuri de pe TouTube și accesarea unor site-uri utile pentru dezvoltarea de idei.

În cele din urmă, am descoperit o soluție asemănătoare cu ceea ce doream eu să realizez pe site-ul de proiecte de la Arduino.

# Bibliografie

<https://www.hackster.io/buzzandy/music-reactive-led-strip-5645ed>

<https://create.arduino.cc/projecthub/buzzandy/music-reactive-led-strip-5645ed?ref=search&ref_id=music%20reactive%20led&offset=0>

https://www.youtube.com/watch?v=lHRquzNed8s&t=2s&ab\_channel=AndyC



Această implementare mi-a stârnit curiozitatea deoarece are funcționalitatea de a analiza frecvența (octavele) sunetului determinat de senzorul de sunet, pe când alte solutii determinau doar dacă sunetul trece de un anumit prag stabilit de volum fără a putea fi filtrat după frecvență.

# Soluții și Implementare

Soluția folosește librăriile FastLED și FHT. Banda LED poate fi privită ca un vector sau ca o matrice, ceea ce dă efect de serpentine, precum în imaginea de mai sus.

În ***setup()*** se pornește un moritor serial, se adaugă banda LED la care se setează culoarea de bază pe **negru** și se fac setările necesare pentru registrul Adc Control and Status Register (ADCSRA).

În ***loop()*** se declară și se inițializează variabile utilizate ulterior. Se formează *input*-ul dat de microfon și este utilizat la apelul funcțiilor din librăria FHT. Se formează *input*-ul dat de potențiometru și este stocat intr-o variabilă. Acesta are rol în stabilirea intensității luminoase de iețire. Se setează vectorul de zgomot variat de volum. Se parcurge vectorul de octave (fecvențe) și se calculează intensitatea sunetului *(j)* pentru fiecare octavă *(i)*. Aceasta este adusă in intervalul pozitiv. Se poate modifica pe baza unui vector de Equalizer pentru a altera sensibilitatea la anumite octave, după preferințe. Dacă această valoare trece de un anumit prag stabilit, se verifică care octavă a generat această valoare și se incrementează o variabilă *beat* pentru alterarea efectului vizual. Se salvează continutul din *j* intr-o variabilă *(prev\_j).*

Sesetează culoarea și intensitatea luminoasă a LED-urilor în funcție de aceste valori și de poziția acestora în matrice/vector. Această poziție este determinată de o funcție auxiliară.

Connections:

* MIC – IN 🡪 pin A0
* Potentiometer – IN 🡪 pin A1
* LED – OUT 🡪 pin D5

A close-up of a device

Description automatically generated

# Testare

Testarea intensivă a avut loc la determinarea valorilor de prag la citirea intensităților sonore, la stabilirea poziției fiecărui LED corespondent cu sunetul pe care il reprezintă, la determinarea valorilor de amplificare ale *Equalizer*-ului și la valorile înregistrate de variabila *beat*. Toate aceste valori au fost modificate în funcție de preferința și calitatea rezultatul final: *Efectul Vizual.*

Soluția este deschisă la modificări deoarece se pot altera atât valori în software (valoare de prag la *beat*, corespondența dintre octava i și intensitatea sonoră j, amplasarea LED-urilor), cât și in hardware (potențiometru intensitate luminoasă, sensibilitate senzor de sunet)

# Exemple Demonstrative.

A picture containing light, indoor, lit, dark

Description automatically generatedA picture containing green, light

Description automatically generatedA picture containing light, traffic, night, dark

Description automatically generated

# Concluzii

Proiectul a fost realizat cu succes și efectul vizual este pe placul meu.

Ar putea fi aduse îmbunătățiri la partea de hardware încât să fie un display mai estetic, o încapsulare într-un suport pentru a putea fi expus mai frumos. În suport ar putea fi inclus și microfonul. O funcționalitate adițională ar putea fi implementarea unui modul Bluetooth astfel încât să nu mai fie nevoie conectarea cu cablu la calculator, doar să fie conectată plăcuța la alimentare.

// PMP Arduino Music Reactive LED Strip Project

#include "FastLED.h"

#include <FastLED.h>

#define OCTAVE 1 // Group buckets into octaves (use the log output function LOG\_OUT 1)

#define OCT\_NORM 0 // Don't normalise octave intensities by number of bins

#define FHT\_N 256 // set to 256 point fht

#include <FHT.h> // include the library

//int noise[] = {204,188,68,73,150,98,88,68}; // noise level determined by playing pink noise and seeing levels

//int noise[] = {204,190,108,85,65,65,55,60}; // noise for mega adk

//int noise[] = {204,195,100,90,85,80,75,75}; // noise for NANO

float noise[] = {204,198,100,85,85,80,80,80};

float noise\_fact[] = {15, 7, 1.5, 1, 1.2, 1.4, 1.7,3}; // BASE noise level determined by playing pink noise and seeing levels

float noise\_fact\_adj[] = {15, 7, 1.5, 1, 1.2, 1.4, 1.7,3}; // Amplified with master\_volume

float eq[] = {150, 160, 150, 120, 90, 90, 95, 100}; //equalizer - boost <- normal \* EQ value[i]

#define LED\_PIN 5 //digital output LEDS

#define LED\_TYPE WS2811

#define COLOR\_ORDER GRB

// Matrix dimensions

const uint8\_t kMatrixWidth = 11;

const uint8\_t kMatrixHeight = 27;

#define NUM\_LEDS (kMatrixWidth \* kMatrixHeight)

//#define NUM\_LEDS 297

int counter2=0;

CRGB leds[NUM\_LEDS]; //the led strip

//////////////////////

// SETUP //

//////////////////////

void setup() {

Serial.begin(115200); // to see the values we get from analogRead, we open a terminal

delay(1000);

FastLED.addLeds<LED\_TYPE, LED\_PIN, COLOR\_ORDER>(leds, NUM\_LEDS).setCorrection( TypicalLEDStrip );

FastLED.setBrightness (200);

fill\_solid(leds, NUM\_LEDS, CRGB::Black);

FastLED.show();

// TIMSK0 = 0; // turn off timer0 for lower jitter

ADCSRA = 0xe5; // set the adc to free running mode

ADMUX = 0x40; // use adc0 for MIC

DIDR0 = 0x01; // turn off the digital input for adc0

}

/////////////////////

// Loop //

/////////////////////

void loop() {

int prev\_j[8];

int beat=0;

int prev\_oct\_j;

int counter=0;

int prev\_beat=0;

int led\_index=0;

int saturation=0;

int saturation\_prev=0;

int brightness=0;

int brightness\_prev=0;

//bool eq\_on = false;

bool eq\_on = true;

while (1) { // reduces jitter

cli(); // UDRE interrupt slows this way down on arduino1.0

for (int i = 0 ; i < FHT\_N ; i++) { // save 256 samples

while (!(ADCSRA & 0x10)); // wait for adc to be ready

ADCSRA = 0xf5; // restart adc

byte m = ADCL; // fetch adc data

byte j = ADCH; // concatenate the lower byte to the higher byte

int k = (j << 8) | m; // form into an int

k -= 0x0200; // form into a signed int

k <<= 6; // form into a 16b signed int

fht\_input[i] = k; // put real-recorded data into bins

}

fht\_window(); // window the data for better frequency response

fht\_reorder(); // reorder the data before doing the fht

fht\_run(); // process the data in the fht

fht\_mag\_octave(); // take the output of the fht fht\_mag\_log()

// every 50th loop, adjust the volume accourding to the value on A2 (Pot)

if (counter >= 50) {

ADMUX = 0x40 | (1 & 0x07); // set admux to look at Analogpin A1 - Master Volume

while (!(ADCSRA & 0x10)); // wait for adc to be ready

ADCSRA = 0xf5; // restart adc

delay(10);

while (!(ADCSRA & 0x10)); // wait for adc to be ready

ADCSRA = 0xf5; // restart adc

byte m = ADCL; // fetch adc data

byte j = ADCH; // concatenate the lower byte to the higher byte

int k = (j << 8) | m; // form into an int

float master\_volume=(k+0.1)/1000 +.5; // so the value will be between ~0.5 and 1.5

// Serial.println (master\_volume);

// amplify with volume/intensity from potentiometer for each octave(freq)

for (int i=1; i<8; i++) {

noise\_fact\_adj[i]=noise\_fact[i]\*master\_volume;

}

ADMUX = 0x40 | (0 & 0x07); // set admux back to look at A0 analog pin to read the microphone input

counter = 0;

}

sei();

counter++;

// End of Fourier Transform code - output is stored in fht\_oct\_out[i].

// i=0-7 frequency (octave) bins (don't use 0 or 1), fht\_oct\_out[1]= amplitude of frequency for bin 1

// for loop

// a) removes background noise average and takes absolute value

// b) low / high pass filter as still very noisy

// c) maps amplitude of octave to a colour between blue and red

// d) sets pixel colour to amplitude of each frequency (octave)

for (int i = 2; i < 8; i++) { // goes through each octave. skip the first 1, which is not useful

float j; // the output from the

j = (fht\_oct\_out[i] - noise[i]); // take the pink noise average level out, take the asbolute value to avoid negative numbers

if (j<10) {j=0;}

if(eq\_on){

j = j \* noise\_fact\_adj[i] \* eq[i] /100;} // multiply by Equalizer

else{

j = j \* noise\_fact\_adj[i];}

if (j<10) {j=0;}

else {

if(eq\_on){

j = j \* noise\_fact\_adj[i] \* eq[i] /100;} // multiply by Equalizer

else{

j = j \* noise\_fact\_adj[i];}

Serial.print("j: ");

Serial.print(j);

Serial.print(", i:");

Serial.print(i);

Serial.print("| ");

if (j>=140) { // for higher volume intensity

// Serial.print("i: ");

// Serial.print(i);

// Serial.print(", ");

if(j>=200) Serial.print(" ---o<#@@@@@@@@#>o--- ");

if (i>=7) {beat+=2;} // higher octave

else

if ((i==2) && j>=200) {beat+=7;} // lower octave and high volume intensity

else

if(i==2 || i==3) {beat+=2;} // lower octave and low volume intensity

else {beat+=1;}

}

j=j/30;

j=j\*30; // (force it to more discrete values)

}

prev\_j[i]=j;

// this fills in 11 LED's with interpolated values between each of the 8 OCT values

if (i>=2) {

led\_index=2\*i-3;

prev\_oct\_j=(j+prev\_j[i-1])/2;

saturation=constrain(j+30, 0,255);

saturation\_prev=constrain(prev\_oct\_j+30, 0,255);

brightness=constrain(j, 0,255);

brightness\_prev=constrain(prev\_oct\_j, 0,255);

if (brightness==255) {

saturation=100;

brightness=200;

}

if (brightness\_prev==255) {

saturation\_prev=100;

brightness\_prev=200;

}

for (uint8\_t y=0;y<kMatrixHeight;y++){ // sets each led's color using the mapping function

leds[XY(led\_index-1,y)] = CHSV(j+y\*30,saturation, brightness);

if (i>2){

prev\_oct\_j=(j+prev\_j[i-1])/2;

leds[ XY(led\_index-2,y)]=CHSV(prev\_oct\_j+y\*30,saturation\_prev, brightness\_prev);

}

}

}

}

if (beat>=6) { // when a high beat is encounered

// fill\_solid(leds, NUM\_LEDS, CRGB::Grey);

// FastLED.setBrightness(150);

// fill\_solid(leds, NUM\_LEDS, CRGB::White);

// FastLED.setBrightness(200);

// fill\_solid(leds, NUM\_LEDS, CRGB(random(25,255),random(25,255),random(25,255))); // set all leds to a random color

fill\_solid(leds, NUM\_LEDS, CRGB::Green);

FastLED.setBrightness(250);

}

else {

if (prev\_beat!=beat) {

FastLED.setBrightness(40+beat\*beat\*5);

// FastLED.setBrightness(200);

prev\_beat=beat;

}

}

FastLED.show();

if (beat) {

counter2+=((beat+4)/2-2);

if (counter2<0) {counter2=100;}

if (beat>3 && beat<7) {

FastLED.delay (20);

}

beat=0;

}

Serial.println();

}

} //end loop

/// function for layout setup

// Param for different pixel layouts

const bool kMatrixSerpentineLayout = true;

// Set 'kMatrixSerpentineLayout' to false if your pixels are laid out all running the same way

// Set 'kMatrixSerpentineLayout' to true if your pixels are laid out back-and-forth

uint16\_t XY( uint8\_t x, uint8\_t y)// mapping function

{

uint16\_t p;

if( kMatrixSerpentineLayout == false) {

p = (y \* kMatrixWidth) + x;

}

if( kMatrixSerpentineLayout == true) {

if( y & 0x01) {

// Odd rows run backwards

uint8\_t reverseX = (kMatrixWidth - 1) - x;

p = (y \* kMatrixWidth) + reverseX;

} else {

// Even rows run forwards

p = (y \* kMatrixWidth) + x;

}

}

p = (p+counter2) % NUM\_LEDS;

return p;

}