ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

К У Р С О В П Р О Е К Т

Тема: Music visualizer w/ WS2812B

Ученик:
Илиян Антов

СОФИЯ
2019

СЪДЪРЖАНИЕ

1. ОПИСАНИЕ НА ПРОЕКТА

Целта на проекта е да бъде разработен визуализатор на музика, базиран на микроконтролер Arduino Uno R3 и лента със светодиодни пиксели WS2812B. Режимите на визуализиране трябва да бъдат задавани чрез компютърно приложение. Управлението на лентата трябва да става от цифров пин на микроконтролера.

Реализираното компютърно приложение е базирано на библиотеката Windows Forms. Връзката с микроконтролера е осъществена посредством серийна комуникация през USB порт.

Визуализацията се извършва на базата на ритъма на музиката. Компютърното приложение следи за резки скоци в басовите честоти в реално време. При засичане на такива, то изпраща сигнали към микроконтролера, който ги обработва и визуализира на лентата.

Засичането на скоци в басовите честоти е базирано на анализ на Фурие. Приложението записва в реално време данните от дадено звуково устройство, пресмята енергията спрямо честотата чрез т.нар. FFT анализ (Fast Fourier Transform) и сравнява моментната енергия на басовите честоти със средната от предварително дефиниран период от време. При достатъчно голям скок, към микроконтролера бива изпратен сигнал за визуализация.

Реализирани са 2 вида визуализация на тактовете. Първият вид е от тип “Бягаща светлина”. При всеки получен сигнал от компютърното приложение, в началото на лентата светват съответен брой пиксели спрямо големината на скока. Същевременно всички светлини се преместват към далечния край на лентата през предварително дефиниран период от време, докато не достигнат края ѝ.

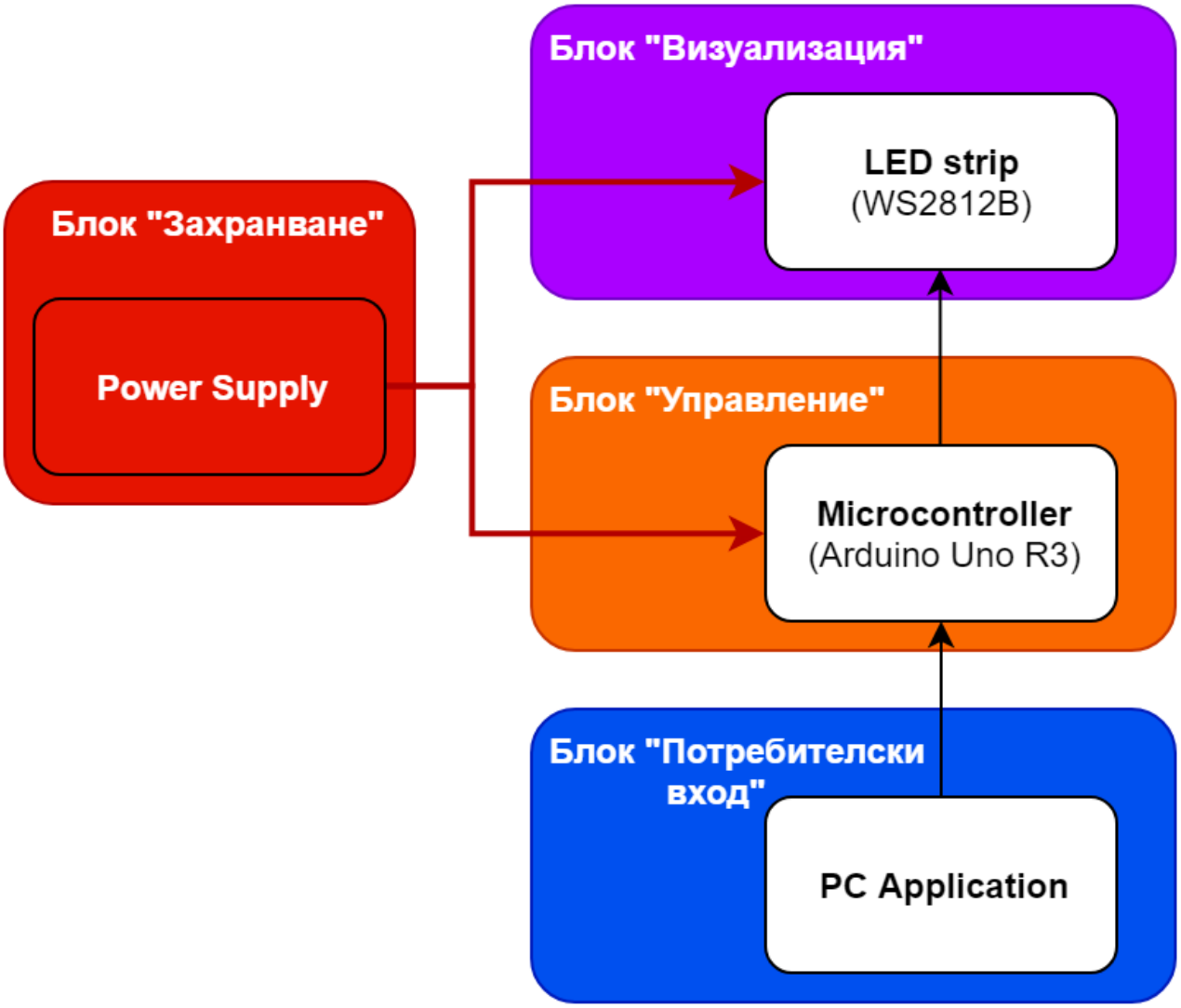
При втория вид визуализация, цялата лента свети, а тактовете само променят яркостта ѝ. При получаване на сигнал от компютърното приложение, яркостта на цялата лента се увеличава, след което плавно се връща към стартовата, по този начин отразявайки скока в басовите честоти.

Реализирана е също функционалност за персонализиране на визуализацията. Добавени за бутони за промяна на скоростта на бягащата светлина. Добавени са слайдер за промяна на чувствителността на алгоритъма за засичане на тактове и слайдер за промяна на скоростта на изменение на цветовата гама на лентата.

2. ПРОЕКТИРАНЕ НА БЛОКОВАТА СХЕМА НА УСТРОЙСТВОТО

2.1. Блокова схема на устройстовто

На Фиг. 2.1. е показана блоковата схема на проектираното устройство.



Фиг. 2.1. Блокова схема на устройството

2.2. Описание на блоковете

Блок “Потребителски вход” - Служи за обработване на потребителския вход на приложението, записване и анализиране на звука, който трябва да бъде визуализиран, както и изпращане на необходимите сигнали на блок “Управление” при въведена от потребителя команда или при засечен такт в режим на визуализация.

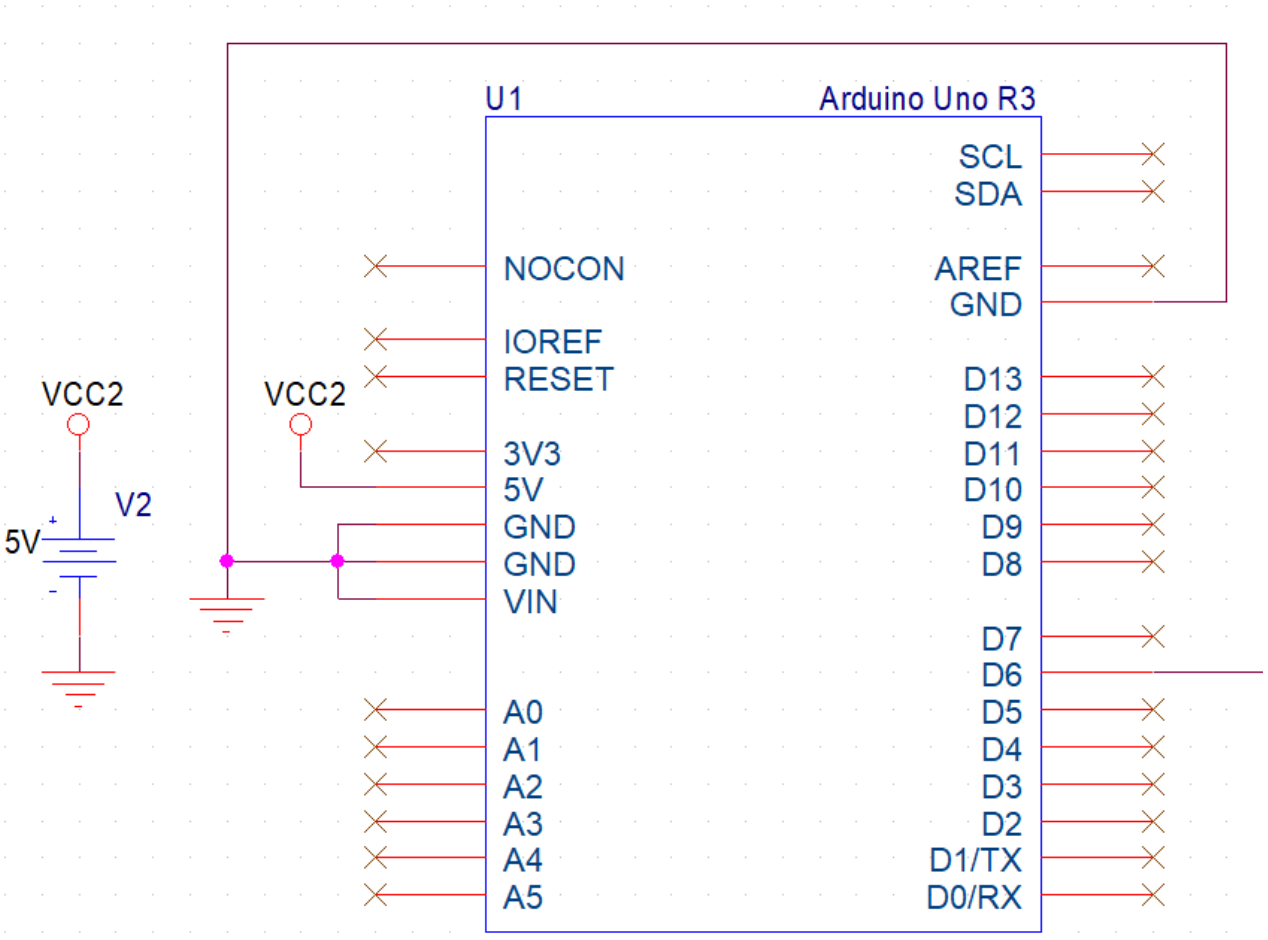
Блок “Управление” - Служи за обработване на получената от блок “Потребителски вход” информация, определяне на режима на работа на визуализацията и изпращане на нужните сигнали към блок “Визуализация”

Блок “Визуализация” – Служи за визуализиране на получените от блок “Управление” сигнали.

Блок “Захранване” – Служи за осигуряването на захранване на всички останали блокове.

3. ПРОЕКТИРАНЕ НА УСТРОЙСТВОТО

3.1. Принципна електрическа схема на Блок “Управление”

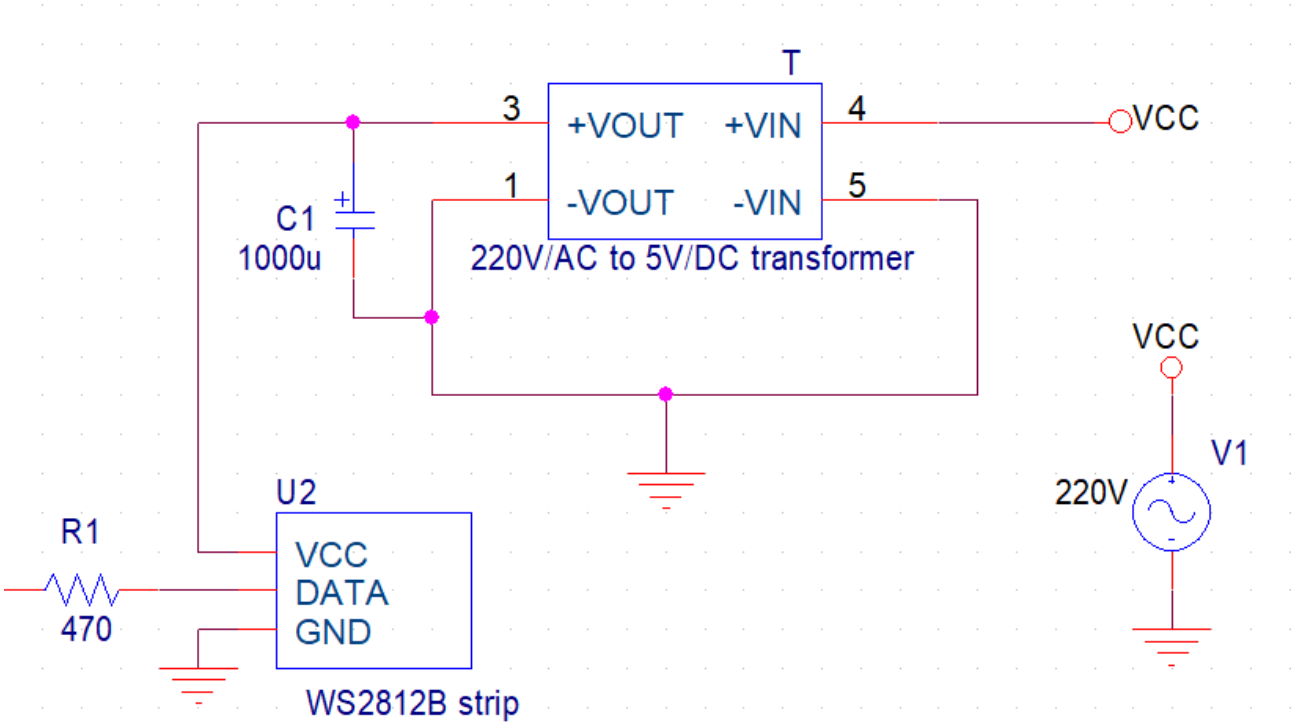


Фиг. 3.1. Принципна електрическа схема на блок “Управление”

На фиг. 3.1. е показана принципната електрическа схема на блок “Управление”. Блокът е базиран на микроконтролера Arduino Uno R3. Захранва се чрез наличния на контролера USB-B порт, свързан към USB-A порта на компютър. Чрез тази връзка се осъществява и серийна комуникация между двете устройства.

Основната задача на блока е да извършва необходимите изчисления и да управлява блок “Визуализация”. Той приема информация от блок “Потребителски вход” и изпълнява въведените от потребителя команди. При команда за начало на визуализацията, блокът започва да изпраща нужните сигнали към блок “Визуализация”.

3.2. Принципна електрическа схема на Блок “Визуализация”



Фиг. 3.2. Принципна електрическа схема на блок “Визуализация”

На фиг. 3.2. е показана принципната електрическа схема на блок “Визуализация”. Основен компонент на блока е лента от LED пиксели WS2812B. Захранва се чрез 220V/AC - 5V/DC адаптер, свързан към контакт. Земите на лентата и микроконтролера от блок “Управление” са обединени. DATA пинът на лентата е свързан към цифров пин на микроконтролера през 470Ω резистор. Към двата изходни терминала на захранващия адаптер е свързан голям електролитен кондензатор, който има за цел да филтрира шумовете и да предпази лентата от пикове на напрежение при включване на захранването.

Използван е адаптер с максимален ток 10A. При изчисляване на нужния ток за захранване на цялата лента са използвани формулуи 3-1 и 3-2. За изчисление на максималния ток през един RGB светодиоден пиксел е използвана формула 3-1. Полученият максимален ток е (20mA * 3 =) 60mA.

$$I_{RGB(max)} = I_{LED(max)} * 3$$

3-1. Формула за изчисление на максималния ток през един RGB LED пиксел

За изчисление на консумацията на цялата лента е използвана формула 3-2. Използваната лента съдържа 300 светодиода. Полученият максимален ток е (60mA * 300 =) 18 000mA = 18A.

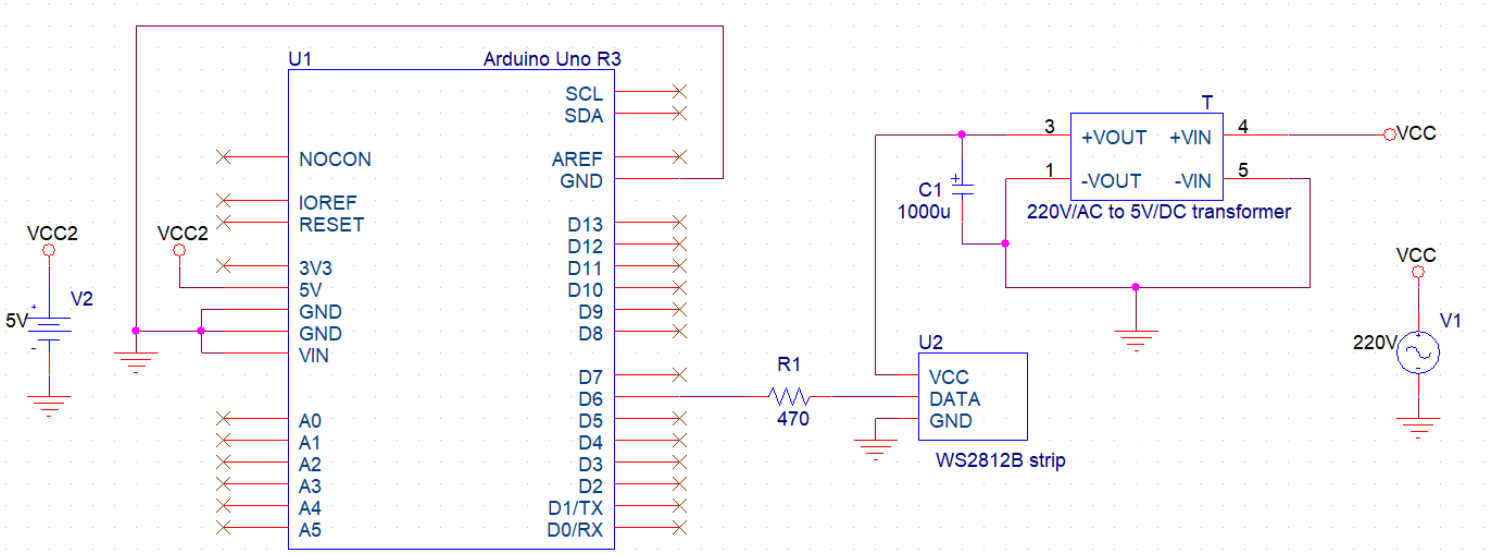
$$I_{max} = I_{RGB(max)} * pixel\ count$$

3-2. Формула за изчисление на максималния ток през цялата лента

Достигането до I_{max} консумация предполага цялата светодиодна лента да свети в бял цвят и със 100% яркост. За целите на устройството е преценено, че лентата никога няма да консумира повече от 50% от максималния ток. Така се стига до извода, че избраният адаптер трябва да поддържа (50% * 18A =) 9A максимален ток на консумация. Следователно изходните му параметри трябва да бъдат 5V/9A или по-добри. Избран е адаптер с параметри 5V/10A.

3.3. Пълна принципна електрическа схема на устройството

На Фиг. 3.3. е показана пълната принципната електрическа схема на проектираното устройство.



Фиг. 3.3. Пълна принципна електрическа схема на устройството

Списък на всички съставни части, използвани в устройството, е приложен в т. 3.4.

3.4. Списък съставни части

[1] Микроконтролер - Arduino Uno R3
- Каталожна информация:
https://www.farnell.com/datasheets/1682209.pdf

[2] LED лента - WS2812B, 60 leds/m, 5m, 300 leds, Black PCB, IP65
- Каталожна информация:
https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf

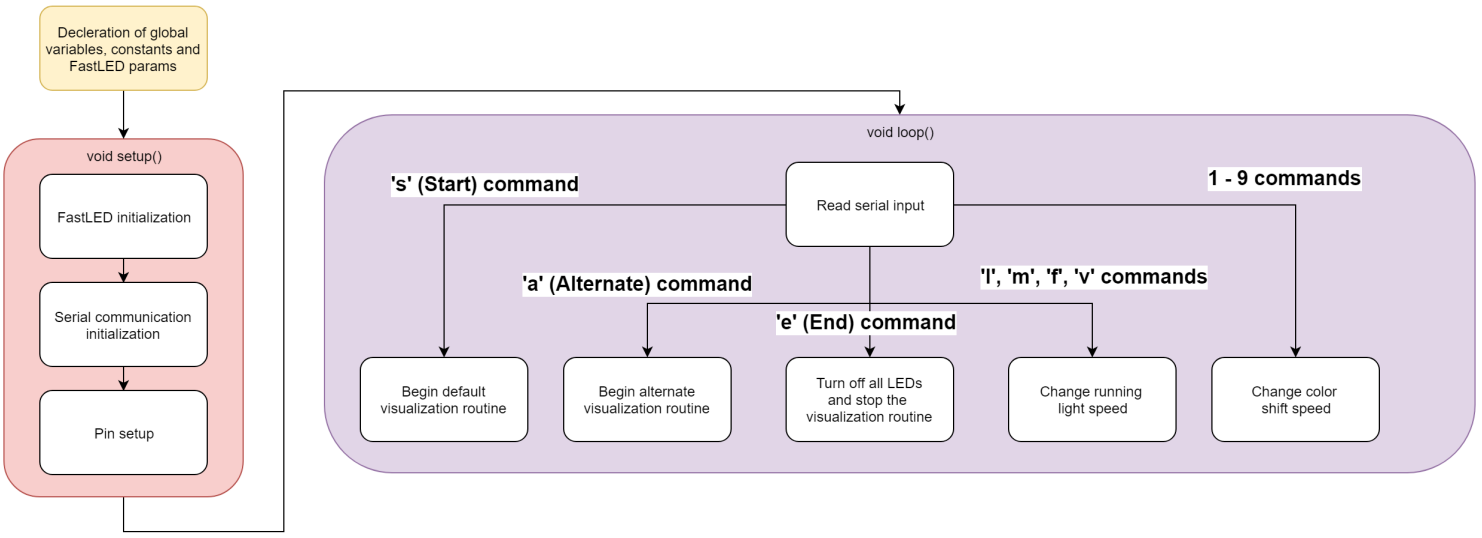
[3] AC/DC адаптер - AC 110-240V / DC 5V, 10A
- Каталожна информация:
https://www.aliexpress.com/item/32810906485.html?spm=a2g0s.12269583.0.0.23d6522dKwRfND

[4] Електролитен кондензатор - 1000uF, 16V
- Каталожна информация:
https://store.comet.bg/download-file.php?id=10105

[5] Резистор - 470Ω, 1/4W
- Каталожна информация:
https://store.comet.bg/download-file.php?id=413

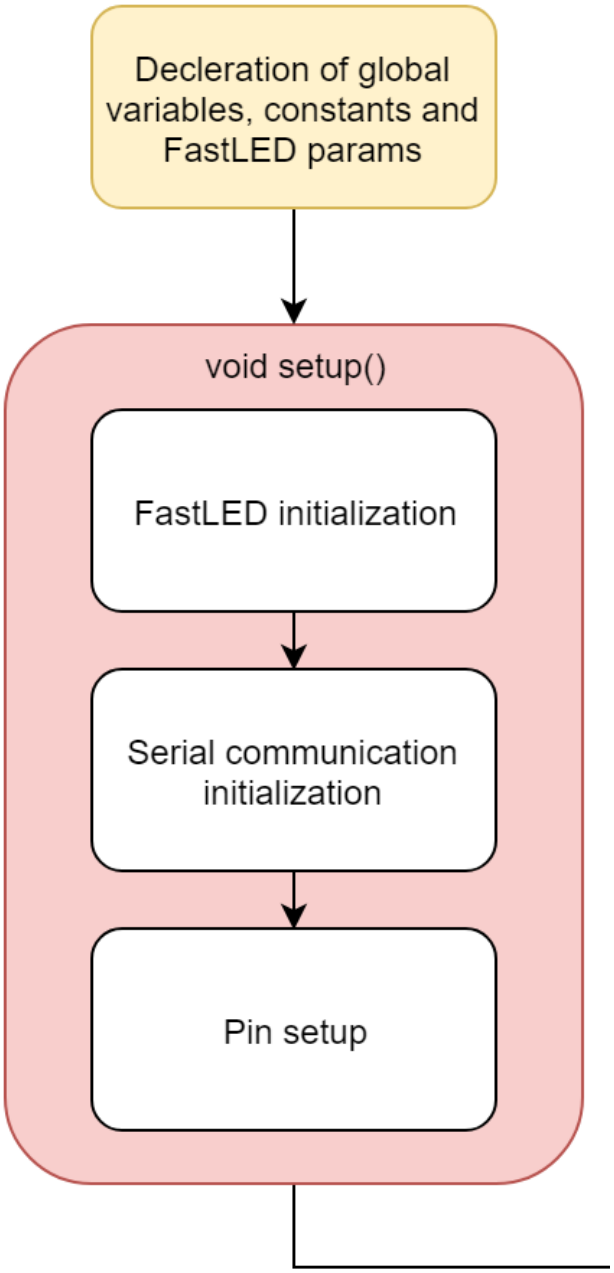
4. ПРОГРАМЕН КОД НА УСТРОЙСТВОТО

4.1. Блокова схема на кода на микроконтролера



Фиг. 4.1. Блокова схема на кода на микроконтролера

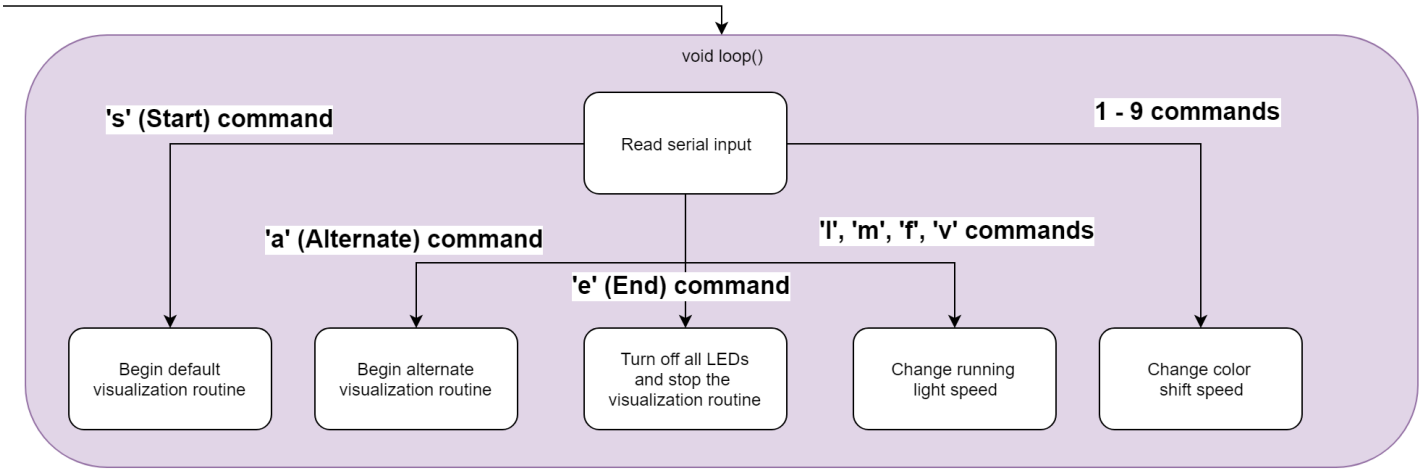
4.2. Блокова схема на кода за инициализация на микроконтролера



Фиг. 4.2. Блокова схема на кода за инициализация (void setup())

На Фиг. 4.2. е показана блоковата схема на кода за инициализация на устройството. През първият етап от инициализацията се извършва деклариране на глобални променливи, константи и параметри на FastLED библиотеката. Втория етап е изпълнението на setup() метода, като в него се инициализира библиотеката FastLED, инициализира се серийната комуникация и се декларират режимите на работа на използваните пинове. В края на инициализацията всички светодиоди се изключват.

4.3. Блокова схема на основния цикъл на микроконтролера

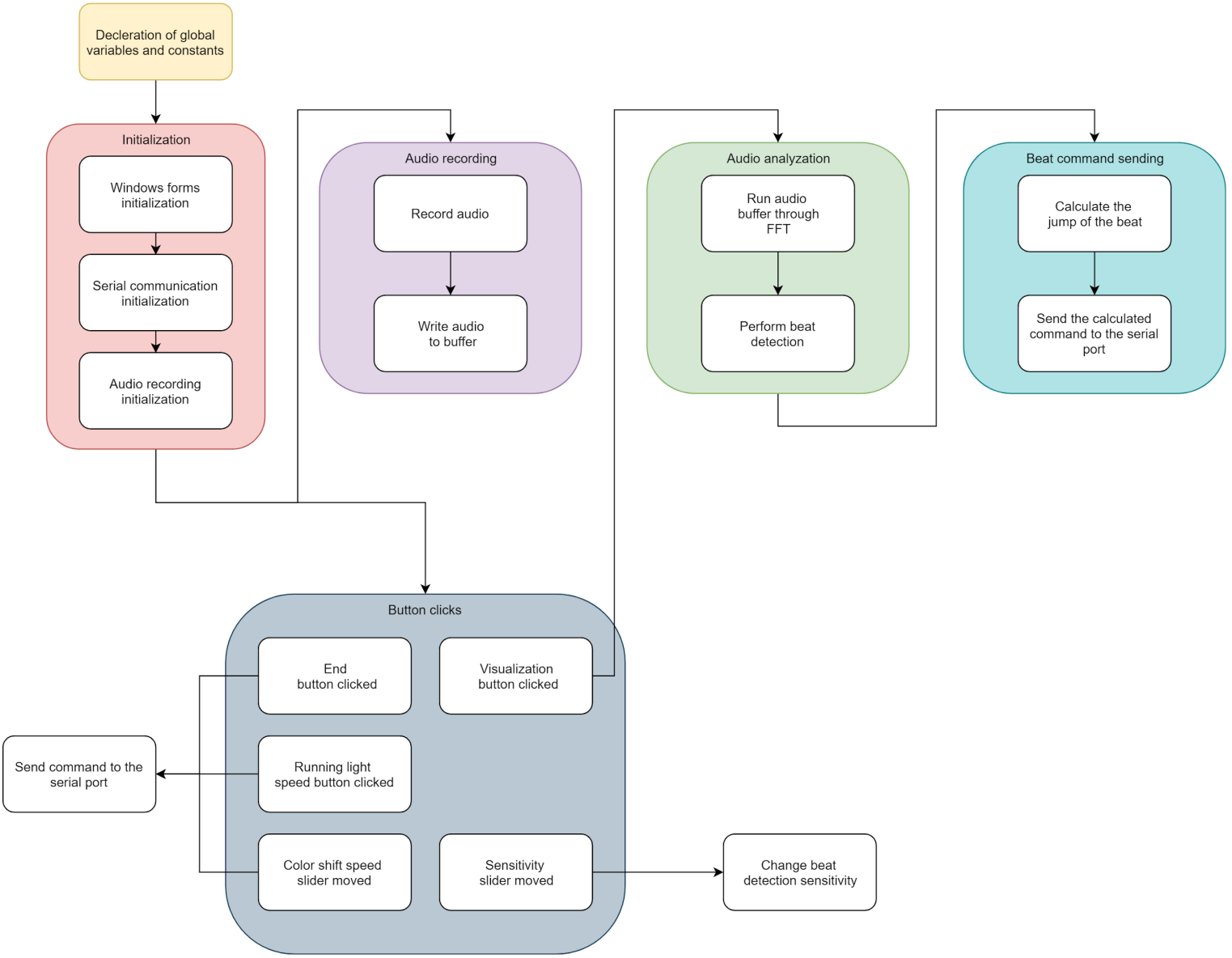


Фиг. 4.3. Блокова схема на основния цикъл (void loop())

На Фиг. 4.3. е показана блоковата схема на основния цикъл. В него се извършва изчакване за получаване на информация от компютърното приложение. При наличие на данни на серийния порт, входът бива прочетен и записан. Възможните следващи стъпки са 6:

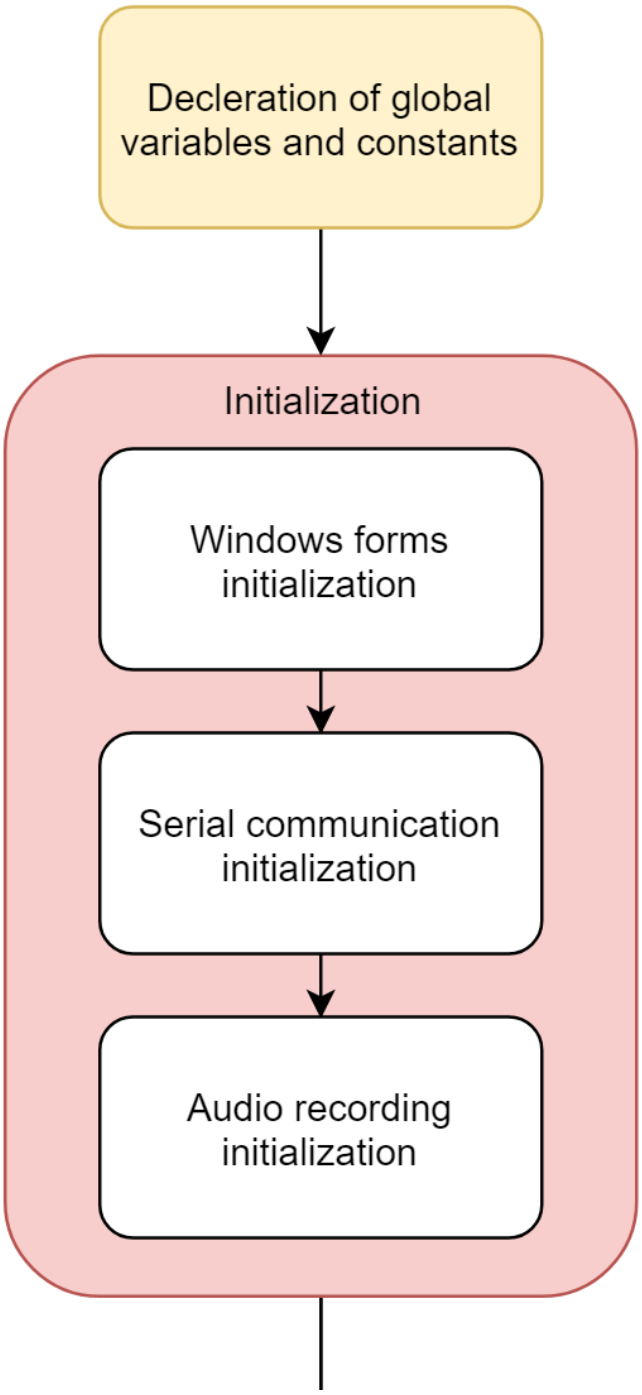
1) Ако пристигналата команда е ‘s’, т.е. Start, се инициализира визуализацията по подразбиране
2) Ако пристигналата команда е ‘a’, т.е. Alternate, се инициализира другият вид визуализация
3) Ако пристигналата команда е ‘l’, ‘m’, ‘f’ или ‘v’, се променя скоростта на преместване на бягащата светлина от визуализацията по подразбиране
4) Ако пристигналата команда е число между 1 и 9, се променя скоростта на изменение на цветовата гама на светодиодите
5) Ако пристигналата команда е ‘e’, т.е. End, се спират всички светодиоди и се прекратява визуализацията
6) Ако пристигналата команда е невалидна, тя бива игнорирана

4.4. Блокова схема на кода на Windows Forms приложението



Фиг. 4.4. Блокова схема на кода на компютърното приложение

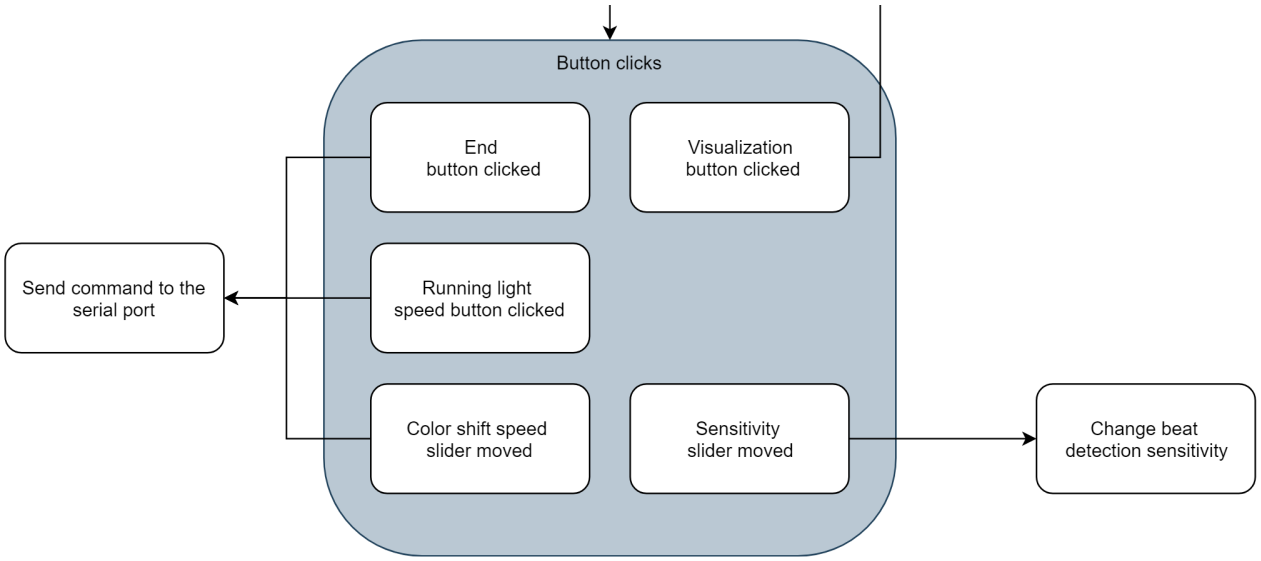
4.5. Блокова схема на кода за инициализация на Windows Forms приложението



Фиг. 4.5. Блокова схема на кода за инициализация

На Фиг. 4.5. е показана блоковата схема на кода за инициализация на компютърното приложение. През първия етап от инициализацията се извършва деклариране на глобални променливи и константи. Втория етап от изпълнението включва инициализиране на Windows Forms библиотеката, инициализация на серийната комуникация и начало на записването на звук от избраното устройство.

4.6. Блокова схема на кода за четене на бутони на Windows Forms приложението

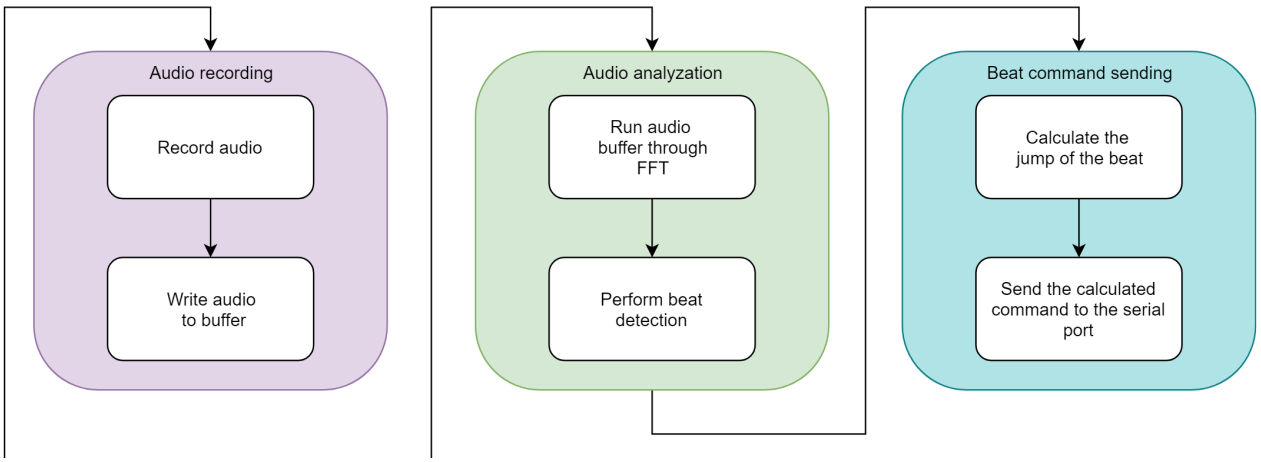


Фиг. 4.6. Блокова схема на кода за четене на бутони

На Фиг. 4.6. е показана блоковата схема на кода за четене на бутони. При натискане на бутон или преместване на слайдер са възможни 3 последващи следствия:

1) При натискане на един от бутоните за визуализация, към микроконтролера се изпраща сигнал за начало и започва анализирането на звука в буфера.
2) При преместване на слайдера за чувствителност на алгоритъма за засичане на тактове, локално се променя стойността на променливата, определяща чувствителността
3) При натискане на бутона за край, бутоните за скорост на бягащата светлина или преместване на слайдера за скорост на промяна на цветовата гама на лентата, съответните команди се изпращат към микроконтролера, който ги обработва и изпълнява.

4.7. Блокова схема на кода за засичане на тактове на Windows Forms приложението



Фиг. 4.7. Блокова схема на кода за засичане на тактове

На Фиг. 4.7. е показана блоковата схема на кода за засичане на тактове. Когато приложението бъде стартирано, започва записване на звук от избраното устройство в буфер.
При натискане на един от бутоните за визуализация, започва анализирането на звука в буфера.
Първата стъпка на алгоритъма е прекарването на буфера през анализ на Фурие, който да преобразува информацията във функция на силата на звука спрямо честотата. По този начин става възможно анализирането на ниските честоти.
След тази стъпка, получената информация се анализира чрез алгоритъм за засичане на тактове. При наличие на такъв, се преминава към следващата стъпка- пресмятане на големината на отскока. Последният етап е изпращането на изчислената информация към серийния порт, т.е. към микроконтролера.

4.8. Приложение 1 - Програмен код на микроконтролера

https://github.com/IliyanAntov/MusicVisualizer

```
#include <FastLED.h>
#define NUM_LEDS 300
#define DATA_PIN 6
#define LED_TYPE WS2811
#define COLOR_ORDER GRB

#define BRIGHTNESS 5
#define MAX_BRIGHTNESS 125
#define FRAMES_PER_SECOND 100

char input; // s -> Start default visualization
            // a -> Start alternate visualization
            // e -> Turn off all LEDs / Stop visualization
            // l, m, f, v -> LED travel speed ([l]ow,[m]edium, [f]ast, [v]ery fast)

int beatStrength; // Recieved by the PC every time there is a beat.

int shiftAmount = 2; // How many spaces a single LED moves every cycle
int colorShiftSpeed = 1; // Determines how fast the color changes (factor)
float colorShiftMaxDelay = 50; // Maximum delay before the color changes (in ms)
float colorShiftDelay = (colorShiftMaxDelay / colorShiftSpeed); // Total delay for every color change (in ms)

CRGB colors = {0, 0, 255}; // FastLED color struct for calculations (Starting color => blue)
int colorCounter = 0; // Color change helper variable

float lastChange = millis(); // Used for the non-blocking delay in the ShiftColors() function

int brightnessFactor = 0; // (Alternate visualization) Determines how much brighter the LEDs need to be

CRGB leds[NUM_LEDS]; // Array of all LEDs
```

Фиг. 4.8. Деклариране на променливи и константи

```
34  void setup() {
35
36      // Safety delay
37      delay(2000);
38
39      // FastLED initial setup
40      FastLED.addLeds<LED_TYPE,DATA_PIN,COLOR_ORDER>(leds, NUM_LEDS).setCorrection(TypicalLEDStrip);
41      FastLED.setMaxPowerInVoltsAndMilliamps(5, 8000);
42      FastLED.setBrightness(BRIGHTNESS);
43
44      // Serial communication intialization
45      Serial.begin(115200);
46
47      // Pin setup
48      pinMode(LED_BUILTIN, OUTPUT);
49      pinMode(DATA_PIN, OUTPUT);
50
51      // Making sure all LEDs are stopped
52      TurnOffLeds();
53  }
54
```

Фиг. 4.9. setup() метод. Инициализация на FastLED библиотеката, серийната комуникация и използваните пинове.

```
55  void loop() {
56
57      // Waiting for user input
58      if(Serial.available() > 0){
59          input = Serial.read();
60      }
61
62      // If 's' is recieved, begin default visualization
63      if(input == 's'){
64          DefaultVisualization();
65      }
66
67      // If 'a' is recieved, begin alternate visualization
68      else if(input == 'a'){
69          AlternateVisualization();
70      }
71
72      // If 'e' is recieved, turn off all LEDs and end visualization
73      else if(input == 'e'){
74          TurnOffLeds();
75      }
76
77      // Set shift speed to [l]ow
78      else if (input == 'l'){
79          shiftAmount = 1;
80      }
81
82      // Set shift speed to [m]edium
83      else if (input == 'm'){
84          shiftAmount = 2;
85      }
86
87      // Set shift speed to [f]ast
88      else if (input == 'f'){
89          shiftAmount = 3;
90      }
91
92      // Set shift speed to [v]ery fast
93      else if (input == 'v'){
94          shiftAmount = 4;
95      }
96
97      // Change the color shift speed
98      else if (input - '0' >= 1 && input - '0' <= 9){
99          colorShiftSpeed = input - '0';
100         colorShiftDelay = colorShiftMaxDelay / colorShiftSpeed;
101     }
102 }
```

Фиг. 4.10. loop() метод. Основен цикъл.

```
106  void ReadInput(){ // Reads the user input from the PC application
107
108      // Setup the default value
109      input = '0';
110
111      // Wait for serial input
112      if(Serial.available() > 0){
113
114          // Write to the input variable
115          input = Serial.read();
116      }
117
118      // Turn off all LEDs if the [e]nd command or invalid input is recieved
119      if(input == 'e' || (input - '0' > 9 || input - '0' < 0)){
120          TurnOffLeds();
121      }
122
123      // Change the color shift delay if a number is recieved
124      else if (input - '0' >= 1 && input - '0' <= 9) {
125          colorShiftSpeed = input - '0';
126          colorShiftDelay = (colorShiftMaxDelay / colorShiftSpeed);
127      }
128  }
```

Фиг. 4.11. Функция за прочитане на информация от серийния порт

```
void DefaultVisualization(){ // Launches the default visualization routine

    // Turn on the status LED
    digitalWrite(LED_BUILTIN, HIGH);

    do{
        // Read the user input
        ReadInput();

        // Convert the user input to an integer
        beatStrength = input - '0';

        // Shift all LEDs and light up the recieved amount
        ShiftLeds(beatStrength);

        // Shift the colors
        ShiftColors();

    }while(input != 'e'); // Stop the visualization when [e]nd flag is recieved
}

void ShiftLeds(int ledsToLight){ // Shifts LEDs down the strip (Used for default visualization)

    // If input is recieved (= a beat), light up that many LEDs
    if(ledsToLight > 0){
        for(int i = 0; i < ledsToLight; i++){
            leds[i] = colors;
        }
    }

    // If no input is recieved, turn off the first [shiftAmount] LEDs
    else{
        for(int i = 0; i < shiftAmount; i++){
            leds[i] = CRGB(0, 0, 0);
        }
    }

    // Shift every LED [shiftAmount] spaces down the strip
    for(int i = NUM_LEDS-1; i >= shiftAmount; i--){
        leds[i] = leds[i-shiftAmount];
    }

    // Update the strip
    FastLED.show();
    FastLED.delay(1000/FRAMES_PER_SECOND);
}

```

Фиг. 4.12. Функции за визуализацията по подразбиране

```
179  void AlternateVisualization(){ // Launches the alternate visualization routine
180
181      // Turn on the status LED
182      digitalWrite(LED_BUILTIN, HIGH);
183
184      do{
185          // Read the user input
186          ReadInput();
187
188          // Turn the brightness down every iteration until it is at the default level
189          if (brightnessFactor > 0){
190              brightnessFactor -= 1;
191          }
192
193          // Convert the user input to an integer
194          int beatStrength = input - '0';
195
196          // Check if the brightness can be turned up
197          if(brightnessFactor + (beatStrength*3) + BRIGHTNESS <= MAX_BRIGHTNESS){
198
199              // Increase the brightness factor
200              brightnessFactor += (beatStrength*3);
201          }
202
203          // Set the new brightness
204          FastLED.setBrightness(BRIGHTNESS + brightnessFactor);
205
206          // Light up all leds with the new brightness
207          LightAllLeds();
208
209          // Shift the colors
210          ShiftColors();
211
212      }while(input != 'e');
213  }
214
215  void LightAllLeds(){ // Lights up every LED on the strip (Used for alternate visualization)
216
217      // Setup every LED with the current color
218      for(int i = 0; i < NUM_LEDS; i++){
219          leds[i] = colors;
220      }
221
222      // Update the strip
223      FastLED.show();
224      FastLED.delay(1000/FRAMES_PER_SECOND);
225  }
226
```

Фиг. 4.13. Функции за другия вид визуализация

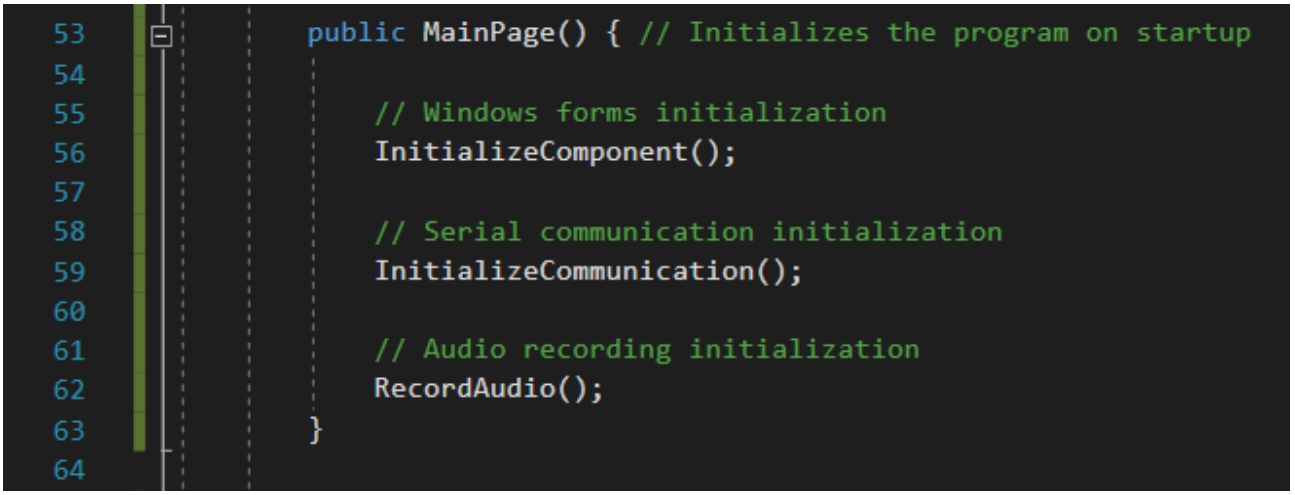
```
228 void ShiftColors(){ // Shifts the colors of the strip after a set amount of time
229
230     // Only update the colors if the given time has passed
231     if(millis() - lastChange > colorShiftDelay){
232
233         if(colorCounter < 255){ // To pink
234             colors.red++;
235         }
236
237         else if (colorCounter >= 255 && colorCounter < 510){ // To white
238             colors.green++;
239         }
240
241         else if (colorCounter >= 510 && colorCounter < 765){ // To yellow
242             colors.blue--;
243         }
244
245         else if (colorCounter >= 765 && colorCounter < 1020){  // To cyan
246             colors.blue++;
247             colors.red--;
248         }
249
250         else if (colorCounter >= 1020 && colorCounter < 1275){  // Back to blue
251             colors.green--;
252         }
253
254         else {
255             colorCounter = 0;
256             colorCounter--;
257         }
258
259         colorCounter++;
260         lastChange = millis();
261     }
262 }
263
264 void TurnOffLeds(){ // Turns off every LED on the strip
265
266     // Turn off the status LED
267     digitalWrite(LED_BUILTIN, LOW);
268
269     // Turn off every LED
270     for(int i = 0; i < NUM_LEDS ; i++) {
271         leds[i] = CRGB(0,0,0);
272     }
273
274     // Update the strip
275     FastLED.show();
276 }
```

Фиг. 4.14. Функции за промяна на цветовете и за изключване на всички светодиоди

4.9. Приложение 2 - Програмен код на компютърното приложение

```
namespace WindowsControlApplication {
    public partial class MainPage : Form {

        public BufferedWaveProvider bwp;

        private int soundCardSampleRate = 48000; // Sound card sample rate

        private int bufferSize = (int)Math.Pow(2, 11); // Sound buffer size

        private int audioDevice = 1; // Audio device to record from
        private int audioChannels = 1; // Audio channels

        private Queue<double> bassHistory = new Queue<double>(); // Used for keeping a history of the bass range for better beat detection
        private Queue<double> lowMidHistory = new Queue<double>(); // Used for keeping a history of the low midrange
        private int historyMaxSize; // Used for calculating bassHistory and lowMidHistory sizes

        private double historySizeInSeconds = 1; // Determines the history queue's size in seconds

        private double bassMultiplier = 3; // Determines how high the jump has to be to detect a bass (in times)

        private double bassAverage = 0; // Used for storing the average of the values stored in bassHistory
        private double lowMidAverage = 0; // Used for storing the average of the low midrange values stored in lowMidHistory

        private int variance = 0; // (TODO) Used for altering the jump requirement for detecting a beat based on the variance of the song

        private int bassLowFreq = 30; // Determines the lowest frequency of the bass range (in Hz)
        private int bassHighFreq = 130; // Determines the highest frequency of the bass range (in Hz)

        int lowMidLowFreq = 250; // Determines the lowest frequency of the low midrange (in Hz)
        int lowMidHighFreq = 500; // Determines the highest frequency of the low midrange (in Hz)

        int hzPerItem; // Used for calculating what the frequency of each item in the buffer is (The difference in Hz between 2 items in the buffer)
        int bassItemsToSkip; // Stores the number of items that need to be skipped from the buffer to reach the bass range
        int bassItemsToTake; // Stores the number of items that need to be taken from the buffer to get the desired bass range values
        int lowMidItemsToSkip; // Stores the number of items that need to be skipped from the buffer to reach the low midrange
        int lowMidItemsToTake; // Stores the number of items that need to be taken from the buffer to get the desired low midrange values

        private SerialPort port = new SerialPort("COM5", 115200); // Defines the port at which the Arduino is plugged in
```

Фиг. 4.15. Деклариране на променливи и константи



Фиг. 4.16. Инициализация на приложението

```
private void RecordAudio() { // Initializes the audio recording

    // Makes an instance of the WaveIn class using the given audio device, sample rate and audio channels
    WaveIn wi = new WaveIn();
    wi.DeviceNumber = audioDevice;
    wi.WaveFormat = new NAudio.Wave.WaveFormat(soundCardSampleRate, audioChannels);

    // Calculates the milliseconds needed to record 1 byte with the given sound card sample rate
    double msPerByte = (1 / (double)soundCardSampleRate) * 1000;

    // Sets up the buffer milliseconds of the WaveIn class to fill the desired buffer
    wi.BufferMilliseconds =  (int)(bufferSize * msPerByte);

    // Wries avaliable data to the buffer
    wi.DataAvailable += new EventHandler<WaveInEventArgs>(AudioDataAvailable);

    // Makes an instance of the BufferedWaveProvider class using the given buffer size and wave format
    bwp = new BufferedWaveProvider(wi.WaveFormat);
    bwp.BufferLength = bufferSize;

    // Sets up the buffer to discard old items when new data is avaliable
    bwp.DiscardOnBufferOverflow = true;

    // Starts the audio recording from the desired audio device
    try {
        wi.StartRecording();
    }
    catch {
        Console.WriteLine("Error reading audio input");
    }

    // Calculates the difference in Hz between 2 items in the buffer
    hzPerItem = (soundCardSampleRate / 2) / (bufferSize / 2);

    // Calculates the maximum items to be stored in history
    historyMaxSize = (int)((soundCardSampleRate / (bufferSize / 2)) * historySizeInSeconds);

    // Calculates the number of items that need to be skipped from the buffer to reach the bass range
    bassItemsToSkip = bassLowFreq / hzPerItem;

    // Calculates the number of items that need to be taken from the buffer to get the desired bass range values
    bassItemsToTake = (bassHighFreq - bassLowFreq) / hzPerItem;

    // Calculates the number of items that need to be skipped from the buffer to reach the low midrange
    lowMidItemsToSkip = lowMidLowFreq / hzPerItem;

    // Calculates the number of items that need to be taken from the buffer to get the desired low midrange values
    lowMidItemsToTake = (lowMidHighFreq - lowMidLowFreq) / hzPerItem;
}
```

Фиг. 4.17. Метод за записване на звук в буфер

```
121     private void InitializeCommunication() { // Initializes the serial communication with the microcontroller
122
123         // Opens the serial port
124         port.Open();
125     }
126
127     private void StartDefaultButton_Click(object sender, EventArgs e) { // Executes when the "Start default" button is pressed
128
129         // Writes the default visualization start flag to the serial port
130         port.Write("s");
131
132         // Enables the update timer, allowing beat detection to begin
133         UpdateTimer.Enabled = true;
134     }
135
136     private void StartAlternateButton_Click(object sender, EventArgs e) { // Executes when the "Start alternate" button is pressed
137
138         // Writes the alternate visualization start flag to the serial port
139         port.Write("a");
140
141         // Enables the update timer, allowing beat detection to begin
142         UpdateTimer.Enabled = true;
143     }
144
145     private void StopButton_Click(object sender, EventArgs e) { // Executes when the "Stop" button is pressed
146
147         // Writes the stop flag to the serial port
148         port.Write("e");
149
150         // Disables the update timer, thus stopping beat detection
151         UpdateTimer.Enabled = false;
152
153     }
154
```

Фиг. 4.18. Методи за инициализация на серийната комуникация, начало и край на визуализацията

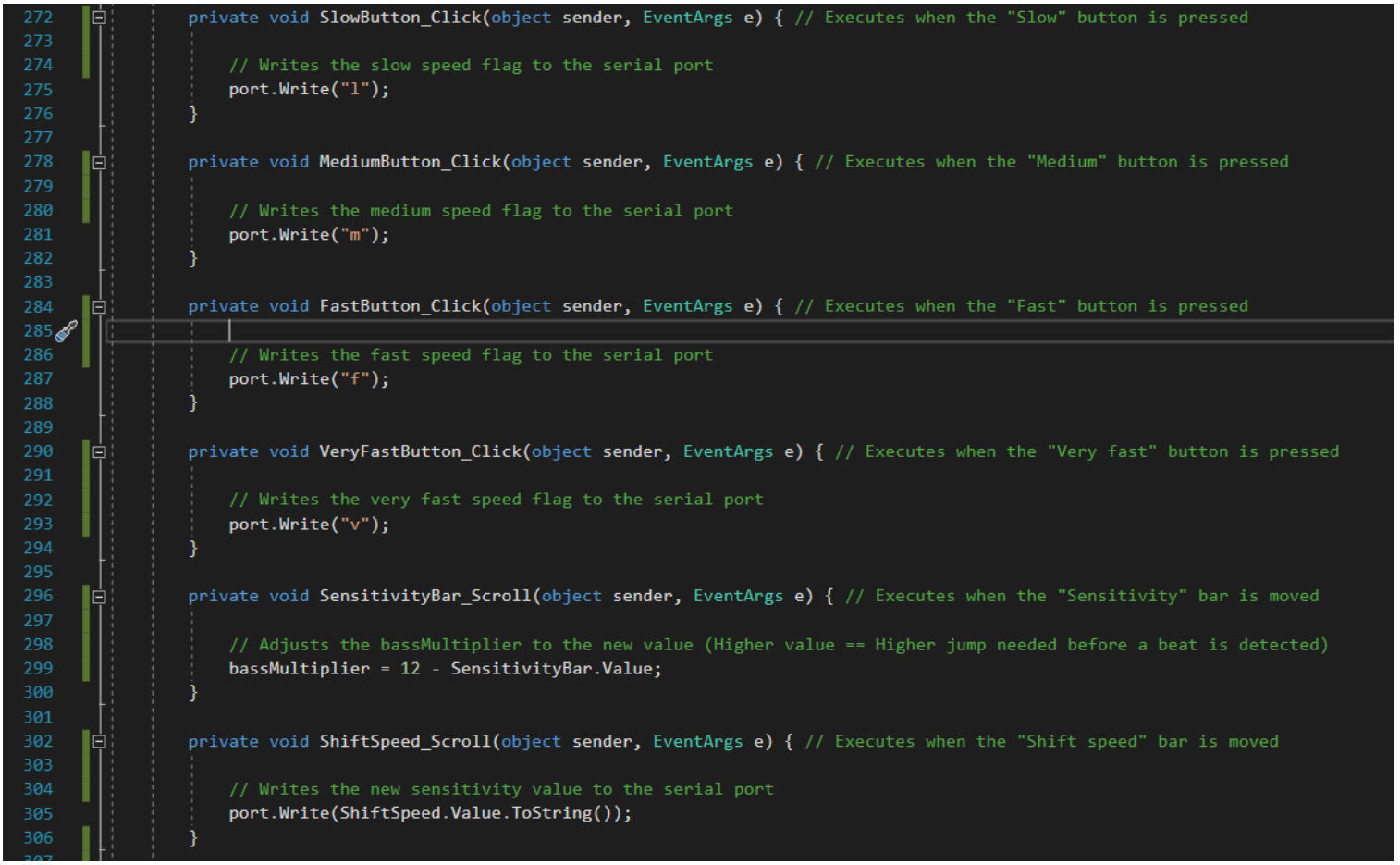
```
155     private void UpdateTimer_Tick(object sender, EventArgs e) { // Executes every time the update timer ticks
156
157         // Analyzes the incoming audio and detects beats
158         AnalyzeAudio();
159     }
160
161     private void AnalyzeAudio() { // Analyzes the incoming audio and detects beats
162
163         // Reads audio bytes from the buffer and stores them in a local array
164         var audioBytes = new byte[bufferSize];
165         bwp.Read(audioBytes, 0, bufferSize);
166
167         // Calculates the compressed array size
168         int compressedArraySize = audioBytes.Length / 2;
169
170         // Initializes the 3 arrays that are going to be used for calculations
171         double[] compressedArray = new double[compressedArraySize];
172         double[] fft = new double[compressedArraySize];
173         double[] fftReal = new double[compressedArraySize / 2];
174
175         // Populates the compressed array with data
176         for (int i = 0; i < compressedArraySize; i++) {
177
178             // Converts 2 consecutive bytes to 1 Int16 for easier calculation
179             Int16 val = BitConverter.ToInt16(audioBytes, i * 2);
180
181             // Converts the value in the Int16 to percent (+/- 100%)
182             compressedArray[i] = (double)(val) / Math.Pow(2, 16) * 200.0;
183         }
184
185         // Calculates the Fast Fourier Transform
186         fft = FFT(compressedArray);
187
188         // Copies the real part of the FFT to the fftReal array
189         Array.Copy(fft, fftReal, fftReal.Length);
190
191         // Raises the values to the second power so that low values will be closer to 0 and high values will be higher, thus making the difference bigger and easier to work with
192         fftReal = fftReal.Select(x => (x * x)).ToArray();
193
194         // Calculates the current bass value from the last FFT
195         double currentBassValues = fftReal.Skip(bassItemsToSkip).Take(bassItemsToTake).Average();
196
```

```
197             // Calculates the current low midrange value from the last FFT
198             double currentLowMidValues = fftReal.Skip(lowMidItemsToSkip).Take(lowMidItemsToTake).Average();
199
200             // Detects beats by comparing the current bass value to the history average
201             if (currentBassValues > (bassMultiplier * bassAverage) && currentBassValues >= 1) {
202
203                 // Calculates the beat strength using the current bass value and the average and sends the result to the serial port
204                 CalculateNumberOfLeds(currentBassValues, bassAverage);
205             }
206
207             // Removes the oldest entry in each queue if it is full
208             if (bassHistory.Count() >= historyMaxSize) {
209                 bassHistory.Dequeue();
210                 lowMidHistory.Dequeue();
211             }
212
213             // Saves the current bass value to the bass history
214             bassHistory.Enqueue(currentBassValues);
215
216             // Saves the current low midrange value to the low midrange history
217             lowMidHistory.Enqueue(currentLowMidValues);
218
219             // Calculates the bass average from history
220             bassAverage = bassHistory.Average();
221
222             // Calculates the low midrange average from history
223             lowMidAverage = lowMidHistory.Average();
224         }
225
```

Фиг. 4.19. Метод за анализиране на звука.

```
226     private void CalculateNumberOfLeds(double currentBassValues, double bassAverage) { // Calculates the beat strength and sends it to the serial port
227
228         // Calculates how much stronger the beat was from the averge
229         double jump = currentBassValues / bassAverage;
230
231         // Low jump
232         if (jump < 10) {
233             port.Write("2");
234         }
235
236         // Medium jump
237         else if (jump >= 10 && jump <= 100) {
238             port.Write("4");
239         }
240         else {
241
242             // High jump
243             port.Write("8");
244         }
245     }
246
247     public double[] FFT(double[] data) { // Performs a FFT on the given data
248
249         // Used to store the result
250         double[] fft = new double[data.Length];
251
252         // Used to store the complex of the FFT
253         System.Numerics.Complex[] fftComplex = new System.Numerics.Complex[data.Length];
254
255         // Calculates the complex for every item in the array
256         for (int i = 0; i < data.Length; i++) {
257             fftComplex[i] = new System.Numerics.Complex(data[i], 0.0);
258         }
259
260         // Performs a FFT
261         Accord.Math.FourierTransform.FFT(fftComplex, Accord.Math.FourierTransform.Direction.Forward);
262
263         // Copies the magnitude of every item to the resulting array
264         for (int i = 0; i < data.Length; i++) {
265             fft[i] = fftComplex[i].Magnitude;
266         }
267
268         // Returns the result of the FFT
269         return fft;
270     }
271
```

Фиг. 4.20. Метод за изчисление на скока на такта и метод за FFT



```
272     private void SlowButton_Click(object sender, EventArgs e) { // Executes when the "Slow" button is pressed
273
274         // Writes the slow speed flag to the serial port
275         port.Write("l");
276     }
277
278     private void MediumButton_Click(object sender, EventArgs e) { // Executes when the "Medium" button is pressed
279
280         // Writes the medium speed flag to the serial port
281         port.Write("m");
282     }
283
284     private void FastButton_Click(object sender, EventArgs e) { // Executes when the "Fast" button is pressed
285
286         // Writes the fast speed flag to the serial port
287         port.Write("f");
288     }
289
290     private void VeryFastButton_Click(object sender, EventArgs e) { // Executes when the "Very fast" button is pressed
291
292         // Writes the very fast speed flag to the serial port
293         port.Write("v");
294     }
295
296     private void SensitivityBar_Scroll(object sender, EventArgs e) { // Executes when the "Sensitivity" bar is moved
297
298         // Adjusts the bassMultiplier to the new value (Higher value == Higher jump needed before a beat is detected)
299         bassMultiplier = 12 - SensitivityBar.Value;
300     }
301
302     private void ShiftSpeed_Scroll(object sender, EventArgs e) { // Executes when the "Shift speed" bar is moved
303
304         // Writes the new sensitivity value to the serial port
305         port.Write(ShiftSpeed.Value.ToString());
306     }
```

Фиг. 4.21. Методи за промяна на скоростта на бягащата светлина, метод за промяна на чувствителността на алгоритъма за засичане на тактове и метод за промяна на скоростта на изменение на цветовата гама на лентата

5. ЗАКЛЮЧЕНИЕ

Всички поставени изисквания към устройството са реализирани успешно.

Разработено е компютърно приложение, чрез което е възможно записване и анализиране на звук в реално време с цел засичане на тактове. Приложението позволява изпращането на управляващи команди към микроконтролера за промяна и персонализиране на режимите на визуализация. При получаване на команда за начало на визуализацията, приложението започва да изпраща команди към микроконтролера при всеки засечен такт на музиката.

Разработен е и програмен код за управление на микроконтролера на устройството Arduino Uno R3. Той позволява четенето на входни данни от серийния порт. Микроконтролерът изпълнява получените команди и изпраща нужните за визуализация сигнали към лентата LED пиксели WS2812B. При получаване на управляващи команди, микроконтролерът ги интерпретира и изпълнява, за да бъде правилно отразен потребителският вход.

ИЗПОЛЗВАНА ЛИТЕРАТУРА

1) https://github.com/swharden/Csharp-Data-Visualization/tree/master/projects/18-09-19_microphone_FFT_revisited
2) https://www.youtube.com/watch?v=q9cRZuosrOs
3) https://stackoverflow.com/questions/37148997/trying-to-understand-buffers-with-regard-to-naudio-in-c-sharp?rq=1
4) https://www.parallelcube.com/2018/03/30/beat-detection-algorithm/
5) https://stackoverflow.com/questions/79445/beats-per-minute-from-real-time-audio-input
6) https://www.teachmeaudio.com/mixing/techniques/audio-spectrum/
7) https://github.com/FastLED/FastLED/wiki/Basic-usage
8) https://www.youtube.com/watch?v=lU1GVVU9gLU
9) https://github.com/DevonCrawford/LED-Music-Visualizer