Youtube Link: https://youtu.be/z8jZJ1tAOY4

Introduction

Audio visualization is a sophisticated technique that converts sound signals into dynamic visual representations. This project employs an Arduino-controlled array of six RGB LED lights to visually represent audio frequencies processed in MATLAB. Each LED corresponds to a distinct frequency band, enabling real-time audio-to-light transformation.

General Overview

A standardized, high-quality audio file is provided as input to the system for frequency spectrum visualization. MATLAB is utilized to read the file, decompose the audio into six frequency bands, and establish serial communication with Arduino to facilitate real-time data transfer.

The transmission process from MATLAB to Arduino consists of the following steps:

1. **Establishing a Serial Connection** – Configuring a serial port in MATLAB to communicate with Arduino.
2. **Data Transmission** – Sending computed brightness levels to Arduino.
3. **Processing on Arduino** – Receiving and interpreting data to adjust LED color and brightness accordingly.

Baud Rate and Synchronization

The baud rate dictates the speed of serial communication. A high baud rate of **115200** is selected to minimize latency while ensuring robust synchronization to prevent data loss.

- **Potential Synchronization Issues**: If MATLAB transmits data before Arduino is ready, initial values may be lost or corrupted.
- **Solution**: MATLAB waits for a **START** signal from Arduino before initiating data transmission, ensuring proper synchronization.

(Figure 2: Synchronization Process)

Splitting Audio into Frequency Bands

To accurately map LED brightness to different audio components, we divide the frequency spectrum into six bands corresponding to sub-bass, low frequencies (kick drums), mids (vocals), and highs (cymbals). Gaussian smoothing with a **100ms window** reduces rapid fluctuations.

- **Frequency Band Definitions**:
    - **High-Pass Frequencies (HPF)**: [10, 70, 250, 1000, 3000, 6000] Hz
    - **Low-Pass Frequencies (LPF)**: [70, 250, 1000, 3000, 6000, fs/2-1] Hz

Each band is processed using **bandpass filtering** and **Gaussian smoothing** to enhance stability.

Processing and Downsampling

- **Computational Optimization**: The signal is **downsampled by a factor of 200**, significantly reducing computational load while preserving key frequency components.
- **Dynamic Normalization**: To maintain perceptual consistency, upper and lower amplitude percentiles are dynamically scaled using a **five-second moving window**.
- **Soft Clipping Function**: This technique ensures smooth LED transitions and avoids saturation effects by applying a cubic polynomial to the normalized values.

Visualization and Synchronization

- **Color Mapping**: Each frequency band is assigned a distinct color to enhance visual contrast.
- **Adaptive Delay Mechanism**: To maintain alignment between audio playback and LED brightness updates, a **real-time correction system** dynamically adjusts delays.

Arduino Integration: Real-Time LED Control

To ensure smooth and responsive LED behavior, the Arduino implementation consists of:

1. **Defining Constants and Initializing PWM Pins**:
    - **Six frequency bands** are mapped to designated **PWM-capable pins**: `{3,5,6,9,10,11}`.
2. **Serial Communication and Processing**:
    - Arduino waits for MATLAB to send brightness data.
    - The received values are parsed and applied to the corresponding **PWM outputs**.
3. **PWM-Based LED Control**:
    - The parsed brightness values are used to modulate the intensity of LEDs in real time, ensuring smooth and synchronized illumination.

Calibration Methodology

To ensure the accuracy and efficacy of the visualization system, four distinct calibration audio samples were utilized:

1. **Full Frequency Transient (1Hz)**:
    - Evaluates the **single-band build** to confirm the system maintains stable speeds.
    - Ensures the effectiveness of **variable delay implementation**.
2. **Spectral Sweeps**:
    - Introduces a **gradual frequency increase** to validate multi-band configurations.

    - Ensures proper **crossover accuracy** and a full **0-255 dynamic range** representation.
3. **Sine Sweep (10Hz - 20kHz)**:
    - Assesses **band transition smoothness** and verifies correct spectral segmentation.
4. **Drum Sample Test**:
    - Evaluates the system's ability to **synchronize rhythmically** and maintain consistency across frequency bands.

Optimization Insights from Calibration Tests

Initial implementations relied on **Root Mean Square (RMS)** averaging to determine amplitude. However, results indicated that:

- The RMS approach led to **rapid oscillations** between 0 and a fixed maximum value.
- A **Gaussian convolution-based smoothing function** proved superior, producing **more stable and visually distinct outputs**.

Conclusion

This project successfully integrates MATLAB and Arduino to produce real-time audio visualizations. The updated implementation ensures accurate frequency segmentation, efficient computational performance, and enhanced synchronization protocols. Future enhancements will explore:

- **Extending RGB channel support** for additional LED colors.
- **Wireless communication** via Bluetooth or Wi-Fi.
- **Further refinements in adaptive filtering techniques** to optimize color transitions and system responsiveness.