ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)
Кафедра комп'ютерних систем та технологій
(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«Розробка аналізатора спектру звукового сигналу на платформі Arduino»

«Development of an audio signal spectrum analyzer on the Arduino platform»

Виконав: здобувач заочної форми навчання
спеціальності 123 – Комп'ютерна інженерія
(код, назва спеціальності)
Освітня програма ОП - Комп'ютерна інженерія
(назва)
Хороших Владислав Володимирович
(прізвище, ім'я, по-батькові здобувача)

Керівник к.ф.-м.н., доцент Шугайло Ю.Б. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)
Рецензент д.т.н., проф. Гунченко Ю.О.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ ___ від ____.____. 20___ р.

Завідувач(ка) кафедри
_____ Гунченко Юрій
(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.20___ р.

Оцінка_____/_____/_____
(за національною шкалою/шкалою ECTS/ бали)
Голова ЕК
_____ Кобозєва Алла
(підпис) (прізвище, ім'я)

Одеса 2024

АНОТАЦІЯ

Дана дипломна робота присвячена розробці звукового аналізатора спектру на основі платформи Arduino. Звуковий аналізатор спектру є важливим інструментом у галузях аудіоінженерії, музичної продукції та наукових досліджень, оскільки дозволяє візуалізувати частотний склад звукових сигналів та їх амплітуду в режимі реального часу.

Основною метою роботи було створення функціонального та зручного у використанні пристрою, який забезпечує точну обробку та відображення звукових сигналів. Для досягнення цієї мети було проведено аналіз існуючих рішень та обрано платформу Arduino за її доступність і гнучкість.

У ході роботи було розроблено апаратну частину аналізатора. Програмне забезпечення, розроблене для пристрою, використовує алгоритми швидкого перетворення Хартлі (ШПХ) для аналізу частотного спектру звукових сигналів.

В результаті виконаної роботи створено функціональний звуковий аналізатор спектру, який охоплює весь чутний діапазон частот і забезпечує чітке та яскраве відображення частотних смуг. Пристрій може бути використаний у різних аудіосистемах та умовах, надаючи зручний інструмент для візуалізації та аналізу звукових сигналів.

Ключові слова: звуковий аналізатор спектру, Arduino, швидке перетворення Хартлі, частотний спектр, аудіоінженерія, світлодіодна матриця, візуалізація звуку.

ABSTRACT

This thesis is dedicated to the development of a sound spectrum analyzer based on the Arduino platform. A sound spectrum analyzer is an important tool in the fields of audio engineering, music production, and scientific research, as it allows for the visualization of the frequency composition of sound signals and their amplitude in real-time.

The main goal of the work was to create a functional and user-friendly device that provides accurate processing and display of sound signals. To achieve this goal, an analysis of existing solutions was conducted, and the Arduino platform was chosen for its accessibility and flexibility.

During the course of the work, the hardware part of the analyzer was developed. The software designed for the device uses Hartley Fast Transform (HFT) algorithms to analyze the frequency spectrum of sound signals.

As a result of the work performed, a functional sound spectrum analyzer was created, covering the entire audible frequency range and providing clear and vivid display of frequency bands. The device can be used in various audio systems and conditions, providing a convenient tool for the visualization and analysis of sound signals.

Keywords: sound spectrum analyzer, Arduino, Hartley Fast Transform, frequency spectrum, audio engineering, LED matrix, sound visualization.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АЦП (ADC) – аналогово-цифровий перетворювач (Analog-to-Digital Converter)

ДПФ (DFT) – Дискретне перетворення Фур’є (Discrete Fourier Transform)

ДПХ (DHT) – Дискретне перетворення Хартлі (Discrete Hartley Transform)

ШПФ (FFT) – швидке перетворення Фур'є (Fast Fourier Transform)

ШПХ (FHT) – швидке перетворення Хартлі (Fast Hartley Transform)

ЦОС (DSP) – цифрова обробка сигналів (Digital Signal Processing)

LED – світлодіод (Light Emitting Diode)

dB – децибел

SPI – послідовний периферійний інтерфейс (Serial Peripheral Interface)

I2C – міжінтегральна шина (Inter-Integrated Circuit)

LCD – рідкокристалічний дисплей (Liquid Crystal Display)

PWM – широтно-імпульсна модуляція (Pulse Width Modulation)

UART – універсальний асинхронний приймач-передавач (Universal Asynchronous Receiver-Transmitter)

ВСТУП

Звуковий аналізатор спектру є важливим інструментом в області аудіоінженерії, музичної продукції та наукових досліджень. Аналіз спектру звукових сигналів дозволяє візуалізувати та розуміти структуру звуку, визначати частотні компоненти та їх амплітуду, що важливо для оптимізації якості звуку та діагностики аудіосистем.

В сучасних умовах цифрова обробка звуку стала невід'ємною частиною багатьох галузей, включаючи музичне виробництво, телекомунікації, акустичну інженерію та медичну діагностику. Зокрема, використання аналізаторів спектру дозволяє ефективно проводити аналіз звукових сигналів, що важливо для професійного аудіообладнання, розробки музичних додатків та інших аудіотехнологій.

Метою цієї дипломної роботи є розробка звукового аналізатора спектру на основі платформи Arduino, здатного в режимі реального часу візуалізувати частотний спектр звукових сигналів. Для досягнення цієї мети необхідно забезпечити високу точність обробки звукових сигналів, плавне відображення спектральних смуг і зручність у використанні пристрою.

Для досягнення цієї мети необхідно виконати наступні завдання:

- Провести огляд та аналіз існуючих рішень для аналізу спектру звукових сигналів.
- Визначити переваги та недоліки різних підходів та технологій.
- Обрати оптимальну платформу для реалізації проекту.
- Вибрати необхідні апаратні та програмні компоненти для створення аналізатора спектру.
- Створити схему пристрою на основі вибраних компонентів.
- Розробити програму для зчитування та обробки звукових сигналів.
- Реалізувати алгоритми спектрального аналізу
- Провести тестування роботи пристрою та програмного забезпечення.
- Виявити та усунути можливі недоліки.

– Оптимізувати роботу аналізатора для забезпечення максимальної точності та ефективності.

Ці завдання дозволяють забезпечити комплексний підхід до розробки звукового аналізатора спектру, що охоплює як технічні, так і теоретичні аспекти проекту.

1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналізатори спектру та їх використання

Аналізатор спектра – прилад для спостереження й вимірювання відносного розподілу енергії електричних (електромагнітних) коливань у смузі частот.

Традиційно в цифровому звукозапису аудіодоріжка представляється у вигляді осцилограми звукової хвилі, що відображає форму (waveform), тобто залежність амплітуди звуку від часу (див. рис. 1.1). Таке представлення достатньо наочно: осцилограма дозволяє побачити основні події у звуці, такі як зміни гучності, паузи між частинами твору й найчастіше навіть окремі ноти в сольному записі інструмента. Але одночасне звучання декількох інструментів на осцилограмі "змішується" і візуальний аналіз сигналу стає скрутним. Проте, наше вухо без труднощів розрізняє окремі інструменти в невеликому ансамблі.

Рисунок 1.1 – Осцилограма звуку

Перші звукові аналізатори спектра розділяли сигнал на частотні смуги за допомогою набору аналогових фільтрів. Дисплей такого аналізатора (див. рис. 1.2) показує рівень сигналу в множині частотних смуг, відповідних до фільтрів [1].

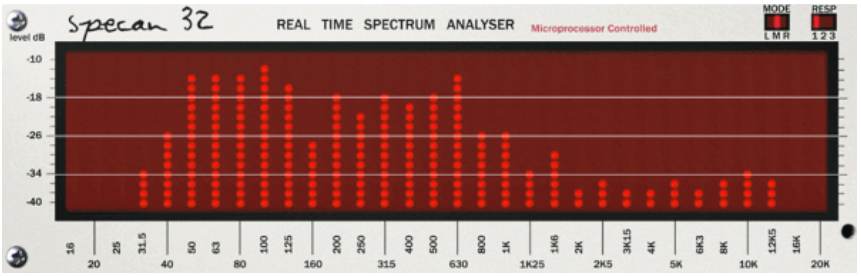

Рисунок 1.2 – Третьоктавний аналізатор Specan32, що емулює відомий прилад KlarkTeknik DN60

На рис. 1.3 наведено приклад частотних характеристик смугових фільтрів у аналізаторі. Такий аналізатор називається третьоктавним, так як у кожній октаві частотного діапазону є три смуги. Видно, що частотні характеристики смугових фільтрів перекриваються; їхня крутість залежить від порядку використовуваних фільтрів.

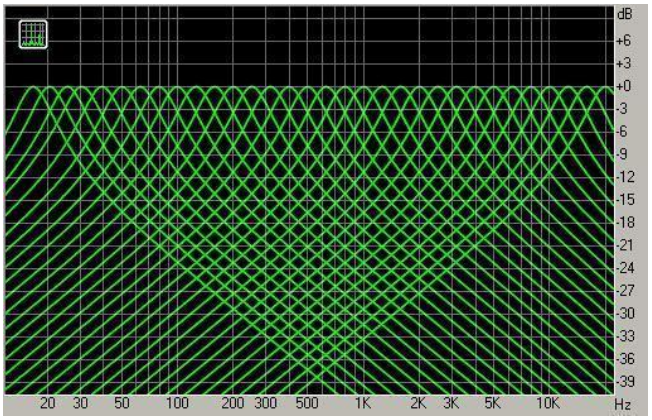

Рисунок 1.3 – Частотні характеристики фільтрів третьоктавного спектроаналізатора

Важливою властивістю спектроаналізатора є балістика – інерційність вимірників рівня в частотних смугах. Вона може регулюватися завданням швидкості наростання (атаки) і спаду рівня. Типовий час атаки й спаду в такому аналізаторі – порядку 200 і 1500 мс.

Смугові спектроаналізатори часто застосовуються для настроювання АЧХ (амплітудно-частотної характеристики) акустичних систем на концертних майданчиках. Якщо на вхід такому аналізатору подати рожевий шум (що має однакову потужність у кожній октаві), то дисплей покаже горизонтальну лінію, з можливою поправкою на варіацію шуму в часі. Якщо рожевий шум, проходячи через звукопідсилювальну систему залу, спотворився, то зміни його спектра будуть показні на аналізаторі. При цьому аналізатор, як і наше вухо, буде малочутливий до вузьких провалів АЧХ (менш 1/3 октави).

1.2 Цифрова обробка та аналіз звуку

Сучасний розвиток електроніки та мікроконтролерної техніки дозволяє будувати аналізатори спектру звукового сигналу без використання великої кількості аналогових смугових фільтрів. Замість цього можна використати математичну обробку аналізуємого сигналу, попередньо перевівши його в цифрову форму. Для цього необхідно розглянути деякі параметри аналого-цифрового перетворення.

Усі величини у фізичному світі носять аналоговий характер. Це стосується й звукових сигналів. Щоб автоматизувати процес вимірювання аналогових величин, і покласти це завдання на електронні прилади, створений спеціальний пристрій, названий аналого-цифровим перетворювачем (АЦП). Цей пристрій дозволяє перетворювати аналоговий сигнал у цифровий код, придатний для використання в ЕОМ.

Насамперед розберемося з тим, що таке цифровий сигнал, як він утворюється із аналогового й звідки власне береться аналоговий сигнал. Останній максимально просто можна визначити як коливання напруги, що виникають через коливання мембрани в мікрофоні.

Типова осцилограма звукового сигналу була показано на рис. 1.1. Для того щоб зрозуміти, як улаштований процес перетворення аналогового сигналу в цифровий, потрібно намалювати осцилограму звуку на

міліметровому папері (див. рис. 1.4). Для кожної вертикальної лінії знайдемо точку перетинання з осцилограмою і найближче ціле значення по вертикальній шкалі — набір таких значень і буде найпростішим записом цифрового сигналу [2].

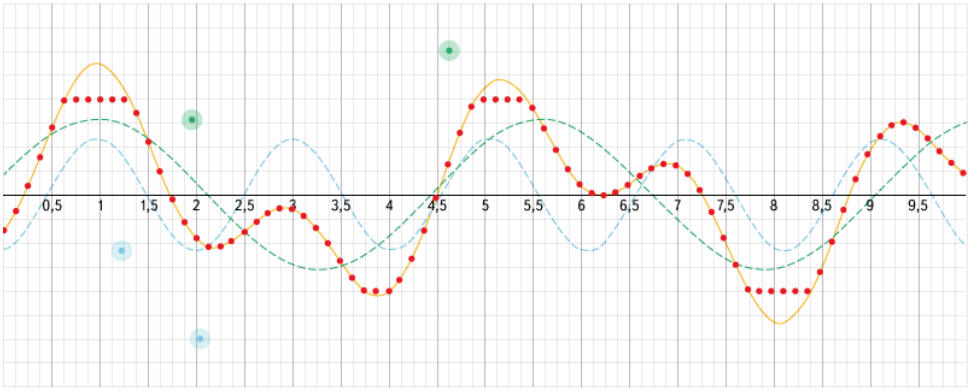

Рисунок 1.4 – Приклад складання хвиль та оцифрування сигналу [3]

Дискретизація

Як відомо, цифровий сигнал — це набір значень рівня сигналу, записаний через задані проміжки часу. Процес перетворення безперервного аналогового сигналу в цифровий сигнал називається дискретизацією за часом і за рівнем (див. рис. 1.5). Є дві основні характеристики цифрового сигналу — частота дискретизації й глибина дискретизації за рівнем [4].

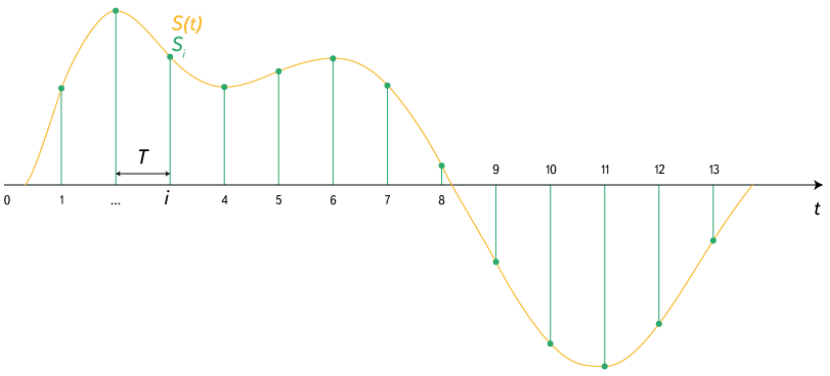

Рисунок 1.5 – Дискретизація сигналу.

Частота дискретизації вказує на те, з якими інтервалами за часом йдуть дані про рівні сигналу. Існує теорема Котельникова (у західній літературі її згадують як теорему Найквіста — Шеннона, хоча зустрічається й назва Котельникова — Шеннона), яка затверджує: для можливості точної відбудови аналогового сигналу з дискретного потрібно, щоб частота дискретизації була мінімум у два рази вище, чим максимальна частота в аналоговому сигналі. Якщо брати зразковий діапазон сприйманих людиною частот звуку 20 Гц — 20 кГц, то оптимальна частота дискретизації (частота Найквіста) повинна бути в районі 40 кГц. У стандартних аудіо-CD вона становить 44.1 кГц

Глибина дискретизації за рівнем визначає розрядність числа, що описує рівень сигналу. Ця характеристика накладає обмеження на точність запису рівня сигналу та його мінімальне значення. Слід зазначити, що дана характеристика не має відношення до гучності – вона відображає точність запису сигналу. Стандартна глибина дискретизації на audio-CD – 16 біт. При цьому якщо не використовувати спеціальну студійну апаратуру, різницю в звучанні більшість перестає помічати вже в районі 10–12 біт. Однак велика глибина дискретизації дозволяє уникнути шумів під час подальшої обробки звуку.

Джерела шумів

У цифровому звуку можна виділити три основні джерела шумів: джиттер, шум подрібнення та аліасінг.

Джиттер — це випадкові відхилення сигналу, як правило, що виникають через нестабільність частоти генератора, що задає, або різної швидкості поширення різних частотних складових одного сигналу. Ця проблема виникає на стадії оцифрування.

Шум дроблення безпосередньо пов'язаний із глибиною дискретизації. Так як при оцифровці сигналу його реальні значення округляються з певною точністю, виникають слабкі шуми, пов'язані з її втратою. Ці шуми можуть з'являтися не тільки на стадії оцифрування, але й у процесі цифрової обробки

(наприклад, якщо спочатку рівень сигналу сильно знижується, а потім знову підвищується).

Аліасінг. При оцифровці можлива ситуація, коли в цифровому сигналі можуть з'явитися частотні складові, яких не було в оригінальному сигналі. Ця помилка отримала назву Aliasing. Цей ефект безпосередньо пов'язаний із частотою дискретизації, а точніше — із частотою Найквіста [5]. Найпростіше зрозуміти, як це відбувається, розглянувши рисунок 1.6.

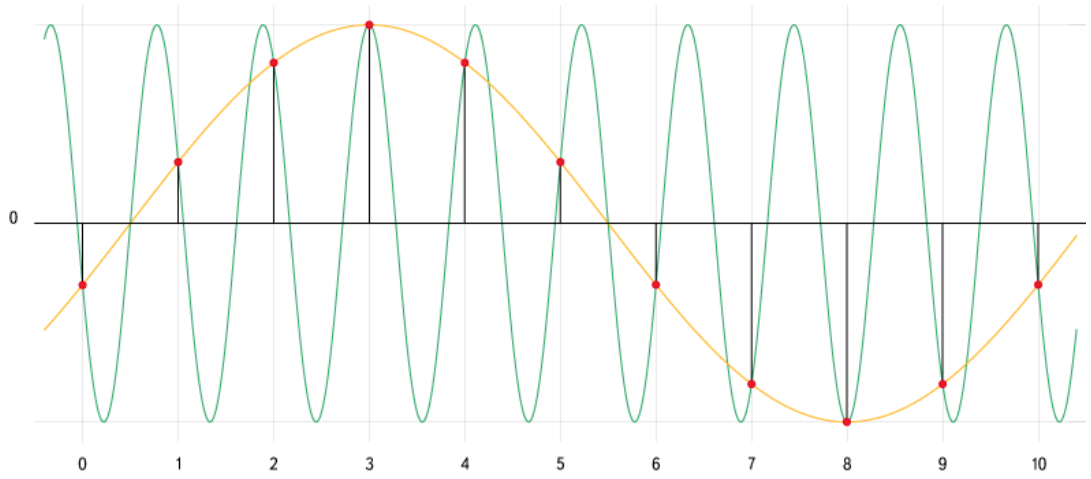

Рисунок 1.6 – Аліасінг. Два різних синусоїдальних сигнали, що не відрізняються при оцифровці: високочастотний з частотою (зелений) на низькочастотний (жовтий).

Зеленим показана частотна складова, частота якої вища за частоту Найквіста. При оцифровуванні такої частотної складової не вдається записати достатньо даних для її коректного опису. В результаті при відтворенні виходить зовсім інший сигнал — жовта крива.

Рівень сигналу

Для початку варто відразу зрозуміти, що коли йдеться про цифровий сигнал, можна говорити тільки про відносний рівень сигналу. Абсолютний залежить в першу чергу від апаратури, що відтворює звук, і прямо пропорційний відносному. При розрахунках відносних рівнів сигналу

прийнято використовувати децибели. При цьому за точку відліку береться сигнал із максимально можливою амплітудою при заданій глибині дискретизації. Цей рівень вказується як 0 dBFS (dB – децибел, FS = Full Scale – повна шкала). Нижчі рівні сигналу вказуються як -1 dBFS, -2 dBFS і т.д. Цілком очевидно, що вищих рівнів просто не буває (ми спочатку беремо максимально можливий рівень).

Спочатку буває важко розібратися про те, як співвідносяться децибели і реальний рівень сигналу. Насправді, все просто. Кожні ~6 dB (точніше 20log(2) ~ 6.02 dB) вказують зміну рівня сигналу вдвічі. Тобто, коли ми говоримо про сигнал з рівнем -12 dBFS, розуміємо, що це сигнал, рівень якого в чотири рази менший за максимальний, а -18 dBFS — у вісім, і так далі. Якщо подивитися на визначення децибелу, то в ньому вказується значення 10log(a/a0) — тоді звідки береться 20? Вся справа в тому, що децибел — це логарифм відношення двох однойменних енергетичних величин, помножений на 10. Амплітуда не є енергетичною величиною, отже її потрібно перевести в відповідну величину. Потужність, яку переносять хвилі з різними амплітудами, пропорційна квадрату амплітуди. Отже, для амплітуди (якщо всі інші умови, крім амплітуди прийняти незмінними), формулу можна записати як $10\log(a^2/a_0^2) => 20\log(a/a_0)$. Логарифм у разі береться десятковий, тоді як більшість бібліотек під функцією з назвою log має на увазі натуральний логарифм.

При різній глибині дискретизації рівень сигналу за цією шкалою не буде змінюватися. Сигнал із рівнем -6 dBFS залишиться сигналом із рівнем -6 dBFS. Але все ж таки одна характеристика зміниться — динамічний діапазон. Динамічний діапазон сигналу – це різниця між його мінімальним та максимальним значенням. Він розраховується за формулою n*20log(2), де n — глибина дискретизації (для грубих оцінок можна скористатися простішою формулою: n*6). Для 16 біт це ~96.33 dB, для 24 біт ~144.49 dB. Це означає, що найбільший перепад рівня, який можна описати з 24-бітною глибиною дискретизації (144.49 dB), на 48.16 dB більше ніж найбільший перепад рівня з

16-бітною глибиною (96.33 dB). Плюс до того — шум дроблення при 24 бітах на 48 dB тихіше.

Сприйняття

Коли говоримо про сприйнятті звуку людиною, слід спочатку розібратися, як люди сприймають звук. Зрозуміло, що ми чуємо за допомогою вух. Звукові хвилі взаємодіють з барабанною перетинкою, зміщуючи її. Вібрації передаються у внутрішнє вухо, де їх уловлюють рецептори. Те, наскільки зміщується барабанна перетинка залежить від такої характеристики, як звуковий тиск. При цьому гучність, що сприймається, залежить від звукового тиску не безпосередньо, а логарифмічно. Тому при зміні гучності прийнято використовувати відносну шкалу SPL (Sound Pressure Level), значення якої вказуються у тих же децибелах.

Одиницею абсолютної шкали гучності є сон [6]. Гучність в 1 сон — це гучність безперервного чистого синусоїдального тону частотою 1 кГц, що створює звуковий тиск 2 мПа.

Рівень гучності звуку — відносна величина, що вимірюється у фонах і чисельно дорівнює рівню звукового тиску (в децибелах), створюваного синусоїдальним тоном частотою 1 кГц такої ж гучності. Для інших частот використовують виправлення з таблиці або спеціального графіку — кривих рівних гучностей (рис.1.7), що являє собою стандартизоване (ISO 226) сімейство кривих, називаних також ізофонами.

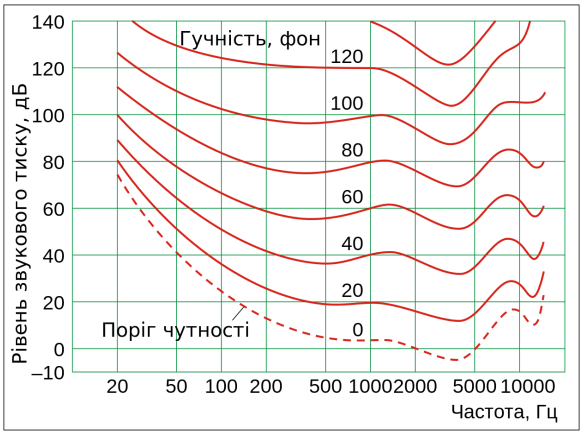

Рисунок 1.7– Криві рівних гучностей.

Суб'єктивне відчуття гучності залежить не тільки від інтенсивності, але і від тривалості звуку. Встановлено, що слуховий апарат людини сприймає гучність інтегровано з вікном у 600–1000 мс. Наприклад, звук однакової інтенсивності сприйматиметься гучнішим по мірі зростання його тривалості до 20, 50, 100, 200 мс і досягне сталого значення при тривалості ~1000 мс. Для довших звуків кожен момент його сприйняття оцінюватиметься середнім значенням останніх 600–1000 мс звучання.

Гучність

Найпростішим прикладом обробки звуку є зміна його гучності. При цьому відбувається просто збільшення рівня сигналу на деяке фіксоване значення. Однак навіть у такій простій справі як регулювання гучності є один підводний камінь. Як зазначалося раніше, гучність, що сприймається, залежить від логарифму звукового тиску, а це означає, що використання лінійної шкали гучності виявляється не дуже ефективним. При лінійній шкалі гучності виникає відразу дві проблеми — для відчутної зміни гучності, коли регулятор знаходиться вище середини шкали доводиться досить далеко його зрушувати, при цьому ближче до низу шкали зсув менше, ніж на товщину волосся, може змінити гучність в два рази. Для вирішення цієї проблеми використовується логарифмічна шкала гучності. При цьому по всій її довжині пересування регулятора на фіксовану відстань змінює гучність в однакову кількість разів. У професійній записувальній та обробній апаратурі, як правило, використовується саме логарифмічна шкала гучності.

1.3 Математична обробка сигналу

Аналізатор спектра дозволяє визначити амплітуду й частоту спектральних компонентів, що входять до складу аналізованого процесу.

Найважливішою його характеристикою є роздільна здатність: найменший інтервал по частоті між двома спектральними лініями, які ще розділяються аналізатором спектра.

Цифрові аналізатори можна побудувати двома способами. У першому випадку це звичайний аналізатор послідовного типу, в якому вимірювальна інформація, отримана методом сканування смуги частот за допомогою гетеродину, оцифровується за допомогою АЦП і обробляється цифровим методом. У другому випадку реалізується цифровий еквівалент паралельного типу як аналізатора з використанням дискретного перетворення Фур'є (ДПФ), який обчислює спектр за допомогою алгоритмів. У порівнянні з послідовними, цифрові паралельні ДПФ-аналізатори мають певні переваги: більш високу роздільну здатність і швидкість роботи, можливість аналізу імпульсних і одноразових сигналів. Вони здатні обчислювати не тільки амплітудний, а й фазовий спектри, а також одночасно подавати сигнали у часовій та частотній областях. На жаль, паралельні ДПФ-аналізатори через обмежені можливості аналого-цифрових перетворювачів (АЦП) працюють тільки на відносно низьких частотах.

Перетворення Фур'є

Перетворення Фур'є – це математичний апарат розкладання сигналів на синусоїдальні коливання. Наприклад, якщо сигнал x(t) безперервний і нескінченний за часом, то його можна подати у вигляді інтеграла Фур'є:

$$x(t) = \int_0^\infty X_\omega \cos(\omega t + \varphi_\omega)\, d\omega \qquad (1.3.1)$$

Інтеграл Фур'є збирає сигнал x(t) з нескінченної множини синусоїдальних складових різних частот ω, що мають амплітуди Xω і фази φω.

Насправді нас більше цікавить аналіз кінцевих за часом звуків. Оскільки музика не є статичним сигналом, її спектр змінюється у часі. Тому

при спектральному аналізі нас цікавлять окремі короткі фрагменти сигналу. Для аналізу таких фрагментів цифрового аудіосигналу існує дискретне перетворення Фур'є:

$$x(n) = \sum_{k=0}^{N/2} X_k \cos \frac{2\pi k(n+\varphi_k)}{N} \qquad (1.3.2)$$

Тут N відліків дискретного сигналу x(n) на інтервалі часу від 0 до N–1 синтезуються як сума кінцевого числа синусоїдальних коливань з амплітудами X_k та фазами φ_k. Частоти цих синусоїд дорівнюють kF/N, де F – частота дискретизації сигналу, а N – число відліків вихідного сигналу x(n) на аналізованому інтервалі. Набір коефіцієнтів X_k називається амплітудним спектром сигналу. Як видно з формули, частоти синусоїд, на які розкладається сигнал, рівномірно розподілені від 0 (постійна складова) до F/2 максимально можливої частоти в цифровому сигналі. Таке лінійне розташування частот відрізняється від розподілу смуг третьоктавного аналізатора.

FFT (Fast Fourier transform) — алгоритм швидкого обчислення дискретного перетворення Фур'є. Завдяки йому стало можливим аналізувати спектр звукових сигналів у реальному часі.

Розглянемо роботу типового FFT-аналізатора [1]. На вхід надходить цифровий аудіосигнал. Аналізатор вибирає з сигналу послідовні інтервали («вікна»), на яких обчислюватиметься спектр, і обраховує FFT у кожному вікні для отримання амплітудного спектру X_k. Обчислений спектр відображається у вигляді графіка залежності амплітуди від частоти (рис. 1.8). Аналогічно смуговим аналізаторам зазвичай використовується логарифмічний масштаб по осях частот і амплітуд. Але через лінійне розташування смуг FFT по частоті спектр може виглядати недостатньо

детальним на нижніх частотах або надмірно осцилюючим на верхніх частотах.

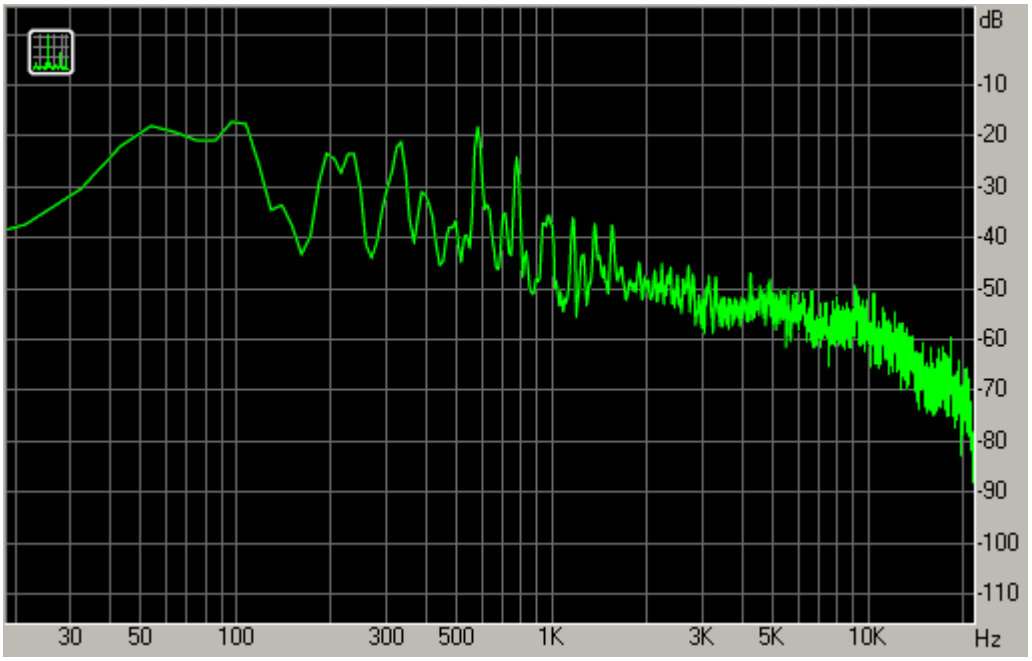

Рисунок 1.8 – Дисплей FFT-аналізатора

Якщо розглядати FFT як набір фільтрів, то, на відміну від смугових фільтрів третьоктавного аналізатора, FFT фільтри будуть мати однакову ширину в герцах, а не в октавах. Тому рожевий шум на FFT-аналізаторі буде вже не горизонтальною лінією, а похилою зі спадом 3 дБ/окт. Горизонтальною лінією на FFT-аналізаторі буде білий шум – він містить рівну енергію у рівних лінійних частотних інтервалах.

Параметр N – число аналізованих відліків сигналу – має вирішальне значення вигляду спектра. Чим більше N, тим щільніше сітка частот, якими FFT розкладає сигнал, і більше деталей по частоті видно на спектрі. Для досягнення вищої частотної роздільної здатності доводиться аналізувати довші ділянки сигналу. Якщо сигнал у межах вікна FFT змінює свої властивості, спектр буде відображати деяку усереднену інформацію про сигнали з усього інтервалу вікна.

Коли потрібно проаналізувати швидкі зміни сигналу, довжину вікна N вибирають маленькою. І тут роздільна здатність аналізу за часом збільшується, а, по частоті – зменшується. Таким чином, роздільна здатність аналізу за частотою обернено пропорційно роздільні здатності за часом. Цей факт називається співвідношенням невизначеностей.

Дискретне перетворення Хартлі

Перетворення Хартлі – інтегральне перетворення, тісно пов'язане з перетворенням Фур'є, але на відміну від останнього перетворює одні дійсні функції в інші дійсні функції.

Перетворення Хартлі є одним із багатьох відомих типів перетворень Фур'є. Перетворення Хартлі може бути зворотнім.

Пряме перетворення:

$$H(\omega) = \{\mathcal{H}f\}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)\,\mathrm{cas}(\omega t)\,\mathrm{d}t\,, \tag{1.3.3}$$

$$\mathrm{cas}(t) = \cos(t) + \sin(t) = \sqrt{2}\sin(t + \frac{\pi}{4}) = \sqrt{2}\cos(t - \frac{\pi}{4}) \tag{1.3.4}$$

Де ω може бути кутовою частотою, а cas(t) є косинус-синус, або ядро Хартлі. Інженерно кажучи, це перетворення переносить сигнал (функцію) з часової області в спектральну область Хартлі (частотну область).

Зворотне перетворення Хартлі:

$$f = \{\mathcal{H}\{\mathcal{H}f\}\} \tag{1.3.5}$$

Зворотне перетворення виходить за принципом інволюції.

Це перетворення відрізняється від класичного перетворення Фур'є виборам ядра перетворення. Однак ці два перетворення тісно пов’язані.

Дискретне перетворення Хартлі (ДПХ) – це пов’язане з Фур’є перетворення дискретних періодичних даних, подібне до дискретного перетворення Фур’є (ДПФ), з аналогічним застосуванням в обробці сигналів і пов’язаних областях. Його головна відмінність від ДПФ полягає в тому, що він перетворює дійсні вхідні дані в дійсні виходи без внутрішньої участі комплексних чисел. Подібно до того, як DFT є дискретним аналогом безперервного перетворення Фур’є (ПФ), DHT є дискретним аналогом безперервного перетворення Хартлі (ПХ).

Дискретне перетворення Хартлі (DHT) і Швидке перетворення Хартлі (FHT) — це два різних методи обчислення перетворення Хартлі для дискретних сигналів. Ось їхні основні відмінності:

Алгоритмічний підхід:

– DHT використовує прямий підхід до обчислення перетворення Хартлі. Його обчислення відбувається за допомогою формул, схожих на ті, які використовуються в дискретному перетворенні Фур'є, але замість синусів і косинусів використовуються інші базисні функції.
– FHT як і у випадку Швидкого перетворення Фур'є (FFT), FHT використовується для швидкого обчислення DHT. Це досягається за допомогою рекурсивного поділу сигналу на підсигнали та застосування швидкісного алгоритму для кожного з них.

Швидкодія:

– DHT має складність O(N2), де N — кількість вхідних точок. Це може бути дуже повільним для обробки великих сигналів.

- FHT має складність O(N logN), що робить його значно швидшим для великих даних.

Застосування:

- DHT зазвичай використовується для невеликої кількості точок або там, де потрібна точність.
- FHT широко використовується в цифровій обробці сигналів, особливо для великих даних, таких як аудіо або відео.

Реалізація:

- DHT може бути реалізований безпосередньо відповідно до математичної формули.
- FHT існують різні оптимізовані алгоритми, що реалізують FHT з ефективністю O(N log N).

Отже, хоча обидва методи забезпечують обчислення перетворення Хартлі для дискретних сигналів, FHT є переважним вибором завдяки своїй швидкості та ефективності, особливо для великих даних.

Витрати на обчислення швидкого перетворення Хартлі становлять приблизно на 36% менше, ніж витрати на обчислення швидкого перетворення Фур'є [7]. Це пов'язано з відсутністю комплексних чисел при обчисленні. Що дозволяє використовувати менш швидкодіючі мікроконтролери, тим самим знизити витрати при розробці модемів та їх функціональних вузлів. Точність швидкого перетворення Хартлі, як правило, трохи вища, ніж у швидкого перетворення Фур'є [8].

2. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

У світі випускається велика кількість різноманітних аналізаторів спектра, які працюють у широкому діапазоні частот. Навіть якщо обмежитися пристроями, призначеними для роботи в звуковому діапазоні, вибір буде дуже різноманітним: від конструкторів для аматорської творчості до професійного дорогого обладнання.

У відносно недорогому сегменті ринку, незважаючи на велику кількість пропозицій, важко знайти пристрій, який можна було б назвати повноцінним аналізатором спектра. Більшість таких пропозицій представляють собою «просунуті» світломузики, з яких практично неможливо отримати об'єктивні дані.

Розглянемо деякі з аналізаторів спектра, які пропонуються промисловістю та розробляються самостійно.

TM10P

Це світлодіодний 5-смуговий аналізатор спектра (рис. 2.1). Модуль TM10P керується мікроконтролером STM32F030F4P6, який виконує обробку звукового сигналу, що надходить на АЦП (аналогово-цифровий перетворювач). Звуковий діапазон умовно поділений на п'ять смуг. Кожна смуга формується за допомогою швидкого перетворення Фур'є (ШПФ), після чого обчислюється середній рівень сигналу в кожній смузі та відображається на світлодіодних стовпчиках [9].

Модуль TM10P пропонує зручний і наочний спосіб аналізу звукового спектра, що робить його корисним інструментом як для аматорських, так і для професійних аудіопроєктів. Можливості модуля дозволяють візуалізувати звуковий сигнал у режимі реального часу, забезпечуючи точне уявлення частотних характеристик аудіосигналу. Такий аналізатор спектра може знайти застосування у різних галузях, включаючи налаштування аудіосистем, звукове інженерство та освітні проекти.

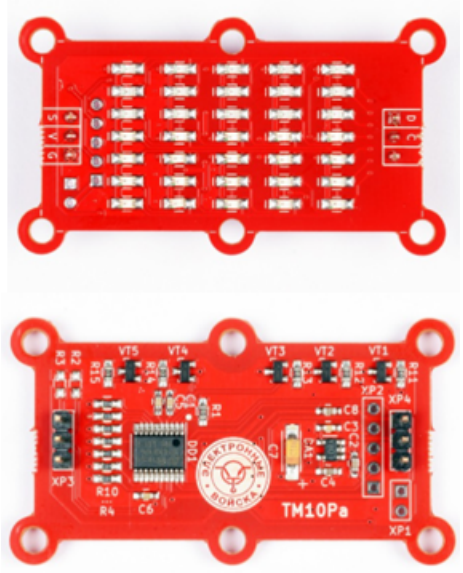

Рисунок 2.1. – Аналізатор спектру TM10PA

За замовчуванням смуги відповідають наступним діапазонам частот:

– від 625 Гц до 937.5 Гц;
– від 1250 Гц до 1562.5 Гц;
– від 1875 Гц до 3125 Гц;
– від 3437.5 Гц до 6250 Гц;
– від 6562.5 Гц до 19 687.5 Гц.

У разі потреби налаштування смуг можна змінити в програмному коді.

Частота дискретизації пристрою становить 40 кГц, кількість точок у перетворенні Фур'є дорівнює 128. Частотна роздільна здатність 40 (кГц) / 128 = 312,5 (Гц). Частота, яку потрібно відобразити на індикаторі, визначається як 312,5 (Гц) * N; де N — номер гармоніки від 1 до 128/2 = 64.

Недоліки цього пристрою — це висока ціна та мала кількість смуг.

LINK1 AK1616 16 Level LED indicator Audio Music spectrum

16-сегментний дисплей для відображення звукового спектру розміром 176*50 мм, розмір світлодіодного поля 167*41 мм (рисунок 2.1).

Рисунок 2.2 – Зовнішній вигляд модуля LINK1 AK1616

Основні характеристики [10]:

- Модуль зеленого кольору;
- Джерело постійного струму: 5 В/600 мА (максимальна яскравість);
- Роздільна здатність: 16 х 16;
- Підтримує частоту розгортки: 32 Гц–16 кГц;
- Регулювання швидкості падіння світлової смуги;
- Час утримання та швидкість падіння піку можна регулювати окремо;
- Яскравість свічення: можна регулювати (15 рівнів);
- Вхідні ланцюги аудіосигналу оптимізовані для коректного відображення при слабкому сигналі;
- Додано функцію для усунення впливу шуму для більш реалістичного відображення музичного спектру.

Переваги цього пристрою: великий функціонал. Недоліки: висока вартість (близько 2000 грн.), неможливість налаштування частот смуг, відсутність фіксації пікових сигналів.

MasterKit MP1205

Цифровий аналізатор спектру звукового сигналу MasterKit MP1205 (рис 2.3) забезпечує відображення амплітуд частотних складових сигналу, що

подається на його вхід, і може використовуватися як у складі музичних систем, так і окремо від них [11].

Рисунок 2.3 – Цифровий аналізатор спектру звукового сигналу MP1205

Технічні характеристики:

- Робоча напруга: 7,5 ... 12 В;
- Потужність не більше: 1,5 Вт;
- Кількість смуг: 10;
- Висота смуг, у пікселях: 7;
- Смуга пропускання: 40 ... 16000 Гц;
- Напруга вхідного сигналу: 0 ... 5 В;
- Режими відображення: лінія, крапка, піки;
- Габаритні розміри друкованої плати: 48x120x1,5 мм

Опис роботи

Вхідний аудіо-сигнал надходить на вхід пристрою, де відбувається його аналогова фільтрація та цифрова обробка. Результат обробки відображається на світлодіодній матриці 10x7.

Переваги: діапазон частот, вибір режимів відображення.

Недоліки: невеликий малоінформативний дисплей – невелика кількість смуг, мала кількість рівнів, що відображаються, висока вартість (понад 1500 грн.)

Geektone LED2015

Аналізатор спектру та еквалайзер LED2015 з DSP (Music Spectrum Level Equalizer) [12] об’єднує в собі два пристрої: еквалайзер на основі DSP та, власне спектроаналізатор. Саме останній буде нас цікавити (рис.2.4).

Рисунок 2.4 – Панель аналізатора спектру Geektone LED2015

Технічні характеристики:

- Живлення: DC 5В (вхід TypeC) / 7-18В
- Частотний діапазон: 20 Гц-20 кГц
- Чутливість: > -40 дБ
- Канали: 2-канальне стерео
- Еквалайзер: 15 смуг ±30 дБ
- Розмір: 250 х 60 х 25 мм
- Розмір дисплея: 200 х 30 мм
- Вага: 180 г.

Основні функції:

- стерео аудіовхід або мікрофонний датчик;
- 10 режимів + одно/двоканальне відображення;
- вбудований годинник;
- швидкість/аналогове посилення/цифрове посилення;
- цифрова обробка звуку DSP;
- автоматичне регулювання посилення DSP.

Для керування та налаштувань є енкодер на повний оборот 30 клацань плюс вбудована кнопка. А також трипозиційний джойстик. Всі налаштування

можна умовно розділити на «Спектрометр/Еквалайзер» та «Інші налаштування».

Розглянемо переважно налаштування, що стосуються спектрометра:

- GAIN — Налаштування підсилення, діапазон: 0–19, значення за замовчуванням: 6;
- CHANEL — Кількість каналів. 0: моно; 1: стерео;
- SPEED — Налаштування швидкості. 0-4 від швидкого до повільного, впливає швидкість падіння плаваючої точки чи швидкість горизонтального переміщення тощо. буд. (залежно від режиму MODE), значення за замовчуванням: 1;
- MIC — вбудований мікрофон. 0: вимкнений; 1: увімкнено;
- DEFLT — Відновлення заводських установок.
- MODE — Режим відображення
- MODE 0: Звичайний спектр з піком вгорі
- MODE 1: Звичайний спектр, без піку вгорі
- MODE 2: Водоспадна діаграма з частотою осі X і часом по осі Y
- MODE 3: Аналогічно 0, але показує лише пікову точку
- MODE 4: Діаграма форми сигналу, бінауральне мікшування, вісь X – час, вісь Y – амплітуда
- MODE 5: аналогічно 1, але орієнтиром є центральна горизонтальна лінія
- MODE 6: Смуга рівня однакової ширини, від середини до обох сторін у разі двох каналів
- MODE 7: Індикатор рівня збільшення ширини (так написано в інструкції)
- MODE 8: смуга рівня однакової ширини
- MODE 9: режим годинника.

Справжній цілком нормальний спектроаналізатор з адекватним відображенням. Через лінійний вхід розпізнавання трохи точніше (напевно,

принаймні немає впливу навколишнього шуму). Закінчений готовий до використання пристрій.

Можна використовувати взагалі як самостійний пристрій із аналізом спектра через вбудований мікрофон.

Недоліки: висока вартість (понад 2000 грн.), неможливість налаштування частот смуг, відкрита конструкція – потребує доробки корпусу.

Розглянуті набори для побудови аналізатора спектра звукового сигналу є одними з найкращих, і навіть мають істотні недоліки. Така ситуація призвела до того, що багато хто намагається створити подібні пристрої, що відповідають їхнім вимогам. У мережі Інтернет можна знайти багато конструкцій, розроблених ентузіастами. За великим рахунком, всі вони можуть бути розділені на дві категорії:

- проекти, побудовані на дорогих компонентах, але забезпечують достатній функціонал;
- бюджетні розробки, що мають не високу собівартість, але мають ті чи інші недоліки (вузька смуга частот, що відображаються, відсутність вибору режиму відображення і т.п.

Нижче представлений один із популярних DIY проектів.

DIY FFT Audio Spectrum Analyzer

Даний аналізатор спектру створений на базі Arduino Nano без будь-яких спеціальних прийомів та оптимізації [13]. В результаті, як видно зі шкали (рис. 2.5) максимальна частота не перевищує 4 кГц.

Рисунок 2.5 – DIY звуковий аналізатор спектру.

На рисунку 2.6 представлена схема пристрою, що складається з:

– Arduino Nano;
– LCD дисплей з роздільною здатністю 128 на 64 пікселі (ST7920 128x64 LCD);
– два резистори (10кОм);
– потенціометр (10кОм);
– конденсатор (1 мкФ).

Зі схеми видно, що воно є максимально спрощеним.

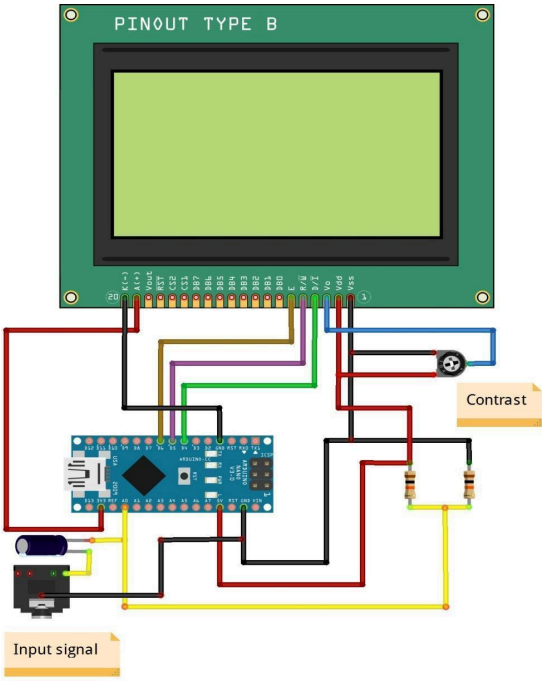

Рисунок 2.6 – Схема DIY FFT Audio Spectrum Analyzer

При аналізі синусоїдального вхідного сигналу можна побачити несучу. Якщо подати на вхід прямокутний сигнал, то на спектральному аналізаторі чітко буде видно основний сигнал, а також три непарні гармоніки x3, x5 та x7.

Це не професійний інструмент, оскільки він має низьку роздільну здатність і частотний діапазон, але може бути відмінним навчальним посібником.

Спектральний аналіз звукового сигналу є дуже корисним в професійний сфері як для оцінки акустичних особливостей приміщень, так і для редагування звукових записів (корегування співвідношення частот, виявлення та прибирання зайвих звуків та т.і.). Але професійне обладнання є доволі дорогим і тому не завжди доступним для невеликих студій або залів.

Дешеві зразки аналізаторів в більшості мають суттєві недоліки, інколи такі, що взагалі не можуть бути застосовані для об'єктивного аналізу.

3. АНАЛІЗ ОБРАНИХ ТЕХНОЛОГІЙ ТА ЇХ ВИКОРИСТАННЯ

3.1 Засоби розробки платформи Arduino

Для розробки проекту аналізатору спектра звукового сигналу було вирішено застосувати платформу Arduino, оскільки ця платформа є найбільш доступною й включає в себе велику кількість плат, що побудовані на різних мікроконтролерах.

Мова програмування Arduino

Мова програмування пристроїв Arduino заснована на C/C++ і використовує бібліотеку AVR Libc, що дозволяє застосовувати будь-які її функції. Вона проста в освоєнні, і на сьогодні Arduino є, мабуть, найзручнішим способом програмування пристроїв на мікроконтролерах.

Основні особливості мови Arduino включають набір бібліотек, що надають функції (на кшталт pinMode) та об'єкти (на кшталт Serial). Під час компіляції програми середовище розробки створює тимчасовий файл .cpp, який включає в себе код користувача і додаткові рядки, необхідні для компіляції. Потім цей файл обробляється компілятором і лінкером з потрібними параметрами.

Приклади бібліотек Arduino:

- FastLED бібліотека для керування адресними світлодіодами.
- FFT бібліотека для швидкого перетворення Фур'є.
- SPI бібліотека для передачі даних через інтерфейс SPI.
- Max72xxPanel бібліотека для керування світлодіодною матрицею.

Програми, написані для Arduino, називаються скетчами (або начерками) і зберігаються у файлах з розширенням .ino. Ці файли перед компіляцією обробляються препроцесором Arduino. Також можливо створювати та підключати до проекту стандартні файли C++.

Функцію main(), обов'язкову в C++, препроцесор Arduino створює самостійно, вставляючи необхідні дії. Програміст повинен написати дві

обов'язкові для Arduino функції: setup() і loop(). Функція setup() викликається один раз при старті, а loop() виконується в нескінченному циклі.

В текст скетчу програміст не зобов'язаний вставляти заголовки стандартних бібліотек, ці заголовки додає препроцесор Arduino відповідно до конфігурації проекту. Однак користувацькі бібліотеки потрібно вказувати вручну.

Arduino IDE має унікальний механізм додавання бібліотек. Бібліотеки у вигляді вихідних текстів на стандартному C++ додаються у спеціальну папку в робочому каталозі IDE. Назва бібліотеки з'являється у списку бібліотек в меню IDE. Програміст обирає необхідні бібліотеки, і вони додаються до списку для компіляції.

Arduino IDE не пропонує налаштувань компілятора і мінімізує інші налаштування, що спрощує початок роботи для новачків і зменшує ризик виникнення проблем.

Середовище розробки Arduino IDE

Інтегроване середовище розробки Arduino (IDE) – це кросплатформний додаток (для Windows, macOS, Linux), написаний на мові програмування Java. Він використовується для написання та завантаження програм на плати Arduino, а також на плати інших виробників за допомогою сторонніх ядер.

Вихідний код для IDE випущений під загальнодоступною ліцензією GNU версії 2.0 Arduino IDE підтримує мови C і C ++, використовуючи спеціальні правила структурування коду. Arduino IDE надає бібліотеку програмного забезпечення з проекту Wiring, яка надає безліч загальних процедур введення і виведення. Для написаного користувачем коду потрібні тільки дві основні функції — для запуску ескізу і основного циклу програми, які компілюються і зв'язуються з заглушкою програми main() в циклічну виконавчу програму за допомогою ланцюжка інструментів GNU, також включеною в дистрибутив IDE. В Arduino IDE використовується програма avrdude для перетворення коду, в текстовий файл в шістнадцятковому

кодуванні, який завантажується в плату Arduino програмою-завантажувачем що вбудовано в плати.

Елементи інтерфейсу Arduino IDE:

- Verify — перевірка програмного коду на помилки та компіляція;
- Upload — компіляція коду та завантаження його на пристрій Arduino;
- New — створення нового скетчу;
- Open — відкриття меню доступу до всіх скетчів;
- Save — збереження скетчу;
- Serial Monitor — відкриття монітору послідовної шини;
- Ім'я скетчу — відображення назви поточного скетчу.

Додаткові команди знаходяться в меню File, Edit, Sketch, Tools та Help. У цих меню активні тільки ті пункти, які можна застосувати до поточного елементу або фрагменту коду.

Arduino може використовуватися як для створення автономних об'єктів автоматики, так і для підключення до програмного забезпечення на комп’ютері через стандартні дротові та бездротові інтерфейси.

Arduino IDE є потужним та гнучким інструментом, який надає всі необхідні засоби для розробки та налагодження програм для мікроконтролерів, роблячи цей процес доступним навіть для новачків.

3.2 Апаратні засоби платформи Arduino

Arduino — це компактна плата, що містить у собі компоненти, серед яких основний — мікроконтролер сімейства AVR ATmega різних конфігурацій. Мікроконтролер є центральною обчислювальною системою платформи, для якого створюється програмне забезпечення. Воно дозволяє мікроконтролеру взаємодіяти з зовнішнім світом через спеціальні порти вводу/виводу даних. Основна мета продукту — забезпечити просте підключення та використання пристрою за принципом "plug-and-play". Керування мікроконтролером здійснюється користувачем через програмний код. Кожна плата Arduino містить мікроконтролер, контакти якого зручно

розведені по краях плати та підписані. Таблиця 3.1 містить основні характеристики найбільш поширених плат Arduino. Інформація взята з офіційного сайту розробників сімейства плат та технічної документації мікроконтролерів [14].

Таблиця 3.1 – Технічні характеристики програмованих плат Arduino

Характеристика	Arduino Uno R3	Arduino Nano	Arduino Mega 2560	Arduino Leonardo	Arduino Due
Мікроконтролер	ATmega328P	ATmega328P	ATmega2560	ATmega32u4	ATSAM3X8E
Напруга живлення	5В	5В	5В	5В	3.3В
Напруга входу	7-12В	7-12В	7-12В	7-12В	7-12В
Цифрові входи/виходи	14 (з них 6 ШІМ)	14 (з них 6 ШІМ)	54 (з них 15 ШІМ)	20 (з них 7 ШІМ)	54 (з них 12 ШІМ)
Аналогові входи	6	8	16	12	12
Flash пам'ять	32 KB	32 KB	256 KB	32 KB	512 KB
SRAM	2 KB	2 KB	8 KB	2.5 KB	96 KB
EEPROM	1 KB	1 KB	4 KB	1 KB	-
Тактова частота	16 МГц	16 МГц	16 МГц	16 МГц	84 МГц
USB	USB-B	Mini-USB	USB-B	Micro-USB	Native USB, Programing USB
Порти UART	1	1	4	1	4
Порти SPI	1	1	1	1	1
Порти I^2C	1	1	1	1	1

Терміни, що використовуються у таблиці, включають:

– EEPROM (Electrically Erasable Programmable Read-Only Memory) — електрично стирається енергонезалежна пам'ять
– Flash-пам'ять — енергонезалежна пам'ять, яка допускає багаторазову перезапис всього вмісту.

– SRAM (Static Random Access Memory) — статична енергозалежна пам'ять з довільним доступом.
– PWM (Pulse-Width Modulation) — широтно-імпульсна модуляція
– UART (Universal Asynchronous Receiver/Transmitter) — універсальний асинхронний приймач/передавач.

АЦП та аналогові входи Arduino

Аналого-цифровий перетворювач (АЦП, англ. Analog-todigital conventer, ADC) — це пристрій, що перетворює вхідний аналоговий сигнал в дискретний код (цифровий сигнал).

У мікроконтролері ATmega328P є 10-розрядний АЦП, який підключений до порту C (див. рис. 3.1) [15]. Вихідна напруга конвертується в 10-розрядне двійкове значення. Мінімальне значення відповідає 0, а максимальне — опорні напрузі. Результат перетворення можна розрахувати за наступною формулою:

$$ADC = \frac{V_{in} \times 1024}{V_{ref}}.$$

(3.2.1)

де: ADC – це значення, отримане від АЦП (в діапазоні від 0 до 1023 для 10-розрядного АЦП);

V_{in} — це вхідна напруга, яку потрібно виміряти;

V_{ref} — це опорна напруга АЦП.

Для ATmega328P вона зазвичай дорівнює напрузі живлення Vcc (наприклад, 5 В), якщо використовується внутрішнє джерело опорної напруги.

Ця формула передбачає, що опорна напруга співпадає з напругою живлення (вибране внутрішнє джерело опорної напруги). Якщо використовується зовнішнє джерело опорної напруги, необхідно замінити V_{ref} на відповідне значення напруги.

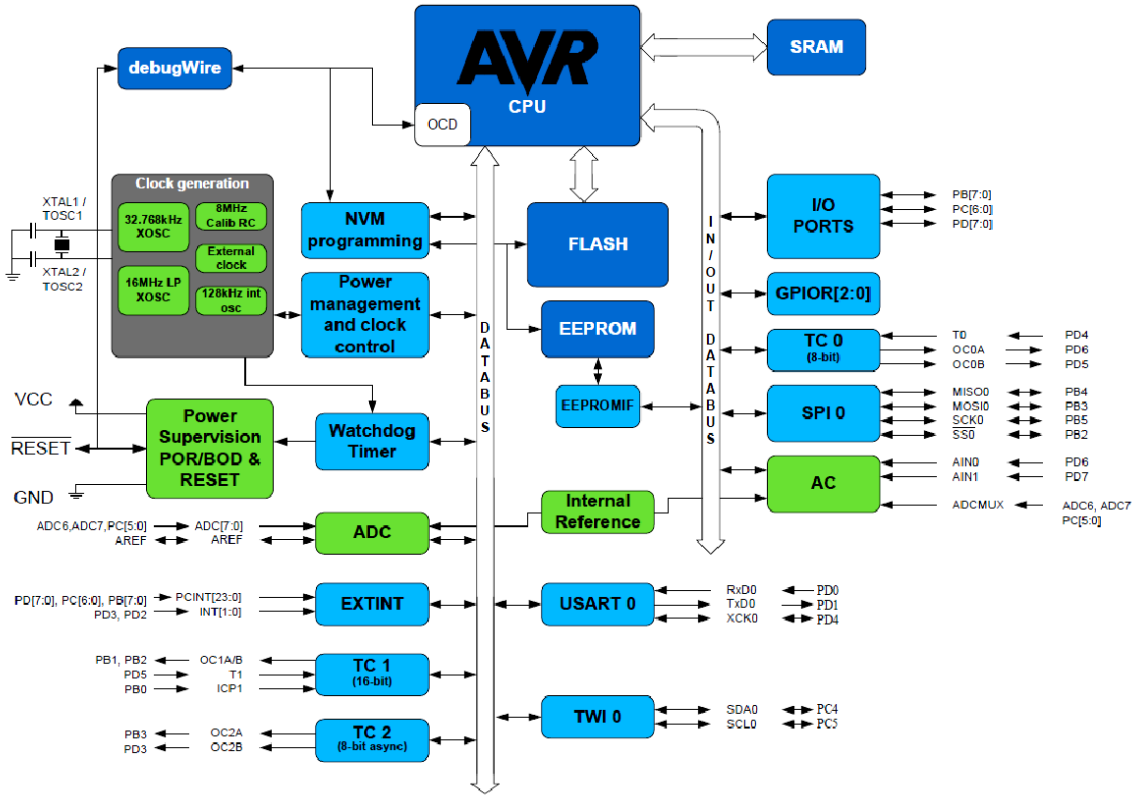

Рисунок 3.1 – Блок-схема мікроконтролера AVR.

Для коректного проведення перетворення, АЦП потребує еталонного значення напруги, з яким буде порівнюватися вхідний аналоговий сигнал. Це еталонне значення відоме як джерело опорної напруги (ДОН). Мікроконтролер Atmega 328P дозволяє використовувати в якості ДОН:

– напругу живлення мікроконтролера 5 В,
– внутрішнє джерело опорної напруги на 1,1 В (на Atmega8 2,56 В),
– напругу на виводі AREF (зовнішнє ДОН, референсна напруга).

За замовчуванням, у Arduino, в якості ДОН використовується напруга живлення МК — 5 В. Для використання інших джерел опорної напруги у мікроконтролера є додатковий вхід AREF. Перед проведенням перетворення аналогового сигналу, при використанні зовнішнього ДОН, необхідно викликати функцію analogReference().

Сигнал, що надходить на вхід АЦП, повинен знаходитися в межах встановленого діапазону 0 (GND)...ДОН. 10-бітний означає, що встановлений діапазон буде розбитий на 2^{10}=1024 значення, і вхідний сигнал буде

оцифровано в відповідне цифрове значення з діапазону 0...1023.

Так, процес оцифрування аналогового сигналу дійсно полягає у послідовному виборі найближчої напруги з відомим (референсним) значенням до вхідної напруги. В АЦП цей процес відбувається шляхом порівняння аналогового сигналу зі значеннями напруги на внутрішньому або зовнішньому джерелі опорної напруги. АЦП визначає, яке з цих значень найбільш близьке до вхідної напруги, і видає відповідне цифрове значення, яке представляє цю напругу в цифровому форматі.

Цей процес можна уявити як розділення вхідного діапазону на рівні проміжки, і вибір того проміжку, до якого належить вхідна напруга. Чим більше бітів у АЦП, тим більша точність оцифрування, оскільки діапазон значень, на які поділяється вхідний діапазон, стає дрібнішим, що дозволяє отримати більш деталізоване представлення вхідної напруги у цифровій формі.

Так, в один момент часу АЦП може оцифровувати сигнал лише з одного аналогового входу через мультиплексор, який відповідає за вибір сигналу. У мікроконтролері Atmega 328P є 6 аналогових входів (у корпусі DIP), які всі належать до порту C.

Запуск перетворення може бути здійснений кількома способами:

- у ручному режимі — одиночне перетворення;
- у автоматичному режимі — за сигналами з різних джерел.

У ручному режимі користувач може ініціювати перетворення за допомогою відповідних команд мікроконтролера. У автоматичному режимі перетворення може бути ініційоване автоматично, наприклад, за допомогою таймерів або інших подій у системі.

Функція analogRead() на Arduino — це простий та зручний спосіб зчитування аналогового вхідного сигналу з піну. Функція ініціює аналогово-цифрове перетворення на вказаному піні та повертає отримане значення, яке знаходиться в діапазоні від 0 до 1023 для 10-бітного АЦП на більшості плат Arduino.

Ось короткий огляд того, як працює analogRead() та її вплив на частоту дискретизації:

- виклик функції -викликається analogRead(pin), де pin — це номер аналогового піну, з якого будуть зчитуватися дані;
- час перетворення процес аналогово-цифрового перетворення (АЦП) займає близько 100 мікросекунд. Цей час визначається тактовими циклами, необхідними для процесу перетворення;
- теоретична частота дискретизації - враховуючи, що кожен виклик analogRead() займає 100 мікросекунд, максимальна теоретична частота дискретизації ($F_{дискр}$) може бути розрахована наступним чином:

$$F_{\text{дискр}} = \frac{1}{\text{Час перетворення}} = \frac{1}{100 \times 10^{-6} \text{секунд}} = 10000 \text{ зр/сек} \qquad (3.2.2)$$

Однак на практиці, через накладні витрати функції та інші фактори, ефективна частота дискретизації трохи нижча, зазвичай близько 9600 Гц [16]. Такої частоти для аналізу звукового сигналу, який може сягати 20 кГц, явно недостатньо. Але в Atmega 328P існують регістри, які дозволяють змінювати налаштування роботи АЦП.

Прискорення роботи аналогового входу.

Щоб прискорити функцію analogRead() на Arduino, можна змінити передмасштабний коефіцієнт АЦП (ADC). Тактовий сигнал АЦП отримується з системного тактового сигналу (16 МГц для більшості плат Arduino), поділеного на передмасштабний коефіцієнт, який за замовчуванням дорівнює 128. Це призводить до тактового сигналу АЦП 125 кГц. Оскільки кожне перетворення АЦП займає 13 тактових циклів, стандартна частота дискретизації приблизно становить 9600 Гц. В таблиці 3.2 наведені значення регістрів керування АЦП та відповідні частоти.

Таблиця 3.2. – Визначення передмасштабного коефіцієнта для АЦП

Передмасштабний коефіцієнт	ADPS2	ADPS1	ADPS0	Тактова частота (MHz)	Частота дискретизації (KHz)
2	0	0	1	8	615
4	0	1	0	4	307
8	0	1	1	2	153
16	1	0	0	1	76.8
32	1	0	1	0.5	38.4
64	1	1	0	0.25	19.2
128	1	1	1	0.125	9.6

Зменшивши передмасштабний коефіцієнт до 16, тактовий сигнал АЦП збільшиться до 1 МГц (16 МГц / 16). Відповідно, частота дискретизації зросте до приблизно 76,8 кГц (1 МГц / 13).

Ось як можна змінити передмасштабний коефіцієнт у функції setup():

```
void setup() {
  // Встановити передмасштабний коефіцієнт АЦП на 16
  ADCSRA &= ~(1 << ADPS2); // Скинути біти ADPS2
  ADCSRA |= (1 << ADPS1) | (1 << ADPS0); // Встановити біти ADPS1 і ADPS0

  Serial.begin(9600); // Ініціалізувати серійний зв'язок на 9600 біт/с
}

void loop() {
  int sensorValue = analogRead(A0); // Зчитати аналоговий вхід на піні A0
  Serial.println(sensorValue);       // Вивести значення на серійний монітор
  delay(1);                          // Невелика затримка для серійного зв'язку
}
```

ADCSRA — це регістр управління і статусу АЦП.

ADPS2, ADPS1 і ADPS0 — це біти в регістрі ADCSRA, які встановлюють передмасштабний коефіцієнт АЦП.

Скидання біта ADPS2 та встановлення бітів ADPS1 і ADPS0 налаштовують передмасштабний коефіцієнт на 16.

Важливі зауваження:

- точність та шум. Вищі тактові частоти АЦП можуть зменшити точність перетворення та збільшити рівень шуму. Потрібно переконатися, що підвищена частота дискретизації не впливає негативно на точність, необхідну для застосування;
- якість сигналу. Потрібно переконайтеся, що сигнал, який зчитується, може обробляти підвищену частоту без значного впливу аліасингу або шуму;
- споживання енергії та нагрівання. Вищі швидкості можуть збільшити споживання енергії та потенційно нагрівання мікроконтролера.

Дотримуючись зазначеного методу, можна ефективно прискорити функцію analogRead() і досягти вищої частоти дискретизації для застосування.

Використання режиму безперервного перетворення АЦП на Arduino (або подібному мікроконтролері) може значно покращити ефективність та точність проекту, дозволяючи АЦП працювати незалежно від основної програми. Ось як можна налаштувати АЦП в режимі безперервного перетворення для безперервного відбору проб та обробки результатів за допомогою переривань.

Покрокове налаштування:

- Налаштування передмасштабного коефіцієнта АЦП (описане вище).
- Налаштування режиму безперервного перетворення — увімкнути АЦП у режимі безперервного перетворення, щоб він постійно виконував перетворення.
- Увімкнення переривань АЦП — налаштувати АЦП для виклику переривання після кожного перетворення, дозволяючи програмі обробляти результати без очікування.
- Обробка переривання — додати обробник переривань для обробки результатів АЦП після завершення кожного перетворення.

Ось приклад, як це можна реалізувати в програмі для Arduino:

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile uint16_t adcResult;

void setup() {
  ADCSRA |= (1 << ADPS2); // Встановити передмасштабний
коефіцієнт на 16
  ADMUX |= (1 << REFS0); // Встановити опорну напругу на AVcc
  ADCSRA |= (1 << ADATE); // Увімкнути режим безперервного
перетворення
  ADCSRA |= (1 << ADEN) | (1 << ADIE); // Увімкнути АЦП та
переривання АЦП
  ADCSRA |= (1 << ADSC); // Запустити перше перетворення
  sei(); // Увімкнути глобальні переривання
}

ISR(ADC_vect) {
  adcResult = ADC; // Прочитати результат АЦП
}

void loop() {
  // Основний код може виконуватися тут без блокування функцією
analogRead()
  // можна отримати доступ до останнього результату АЦП через
змінну adcResult
}
```

Переваги.

- Не блокуюче зчитування — основний цикл може виконувати інші завдання, поки АЦП працює незалежно.
- Зменшення джитеру — АЦП працює з постійною швидкістю, зменшуючи варіації часу відбору проб.
- Використовуючи цей підхід, ви можете покращити ефективність та точність процесу збору даних у вашому проекті на базі Arduino.

3.3 Засоби для відображення спектрограми

Дисплей LCD1602

Рідкокристалічний дисплей (Liquid Crystal Display) скорочено LCD побудований на технології рідких кристалів. При проектуванні електронних

пристроїв, потрібно недорогий пристрій для відображення інформації та другий не менш важливий фактор, наявність готових бібліотек для Arduino.

З усіх доступних LCD дисплеїв на ринку, що найчастіше використовується є LCD 1602, який може відображати ASCII символу в 2 рядки (16 знаків в 1 рядку) кожен символ у вигляді матриці 5x7 пікселів (рис. 3.2). Існує безліч різновидів даного РК модуля, він може бути 1, 2, 4-х рядковий з різним числом символів на рядку.

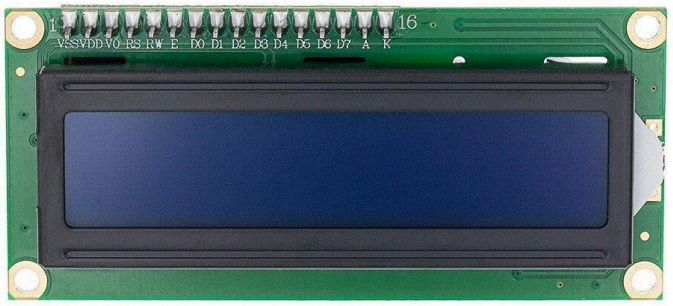

Рисунок 3.2 – Символьний дисплей LCD 1602

LCD 1602 є електронним модулем заснований на драйвері HD44780 від Hitachi. LCD1602 має 16 контактів і може працювати в 4-бітному режимі (з використанням лише 4 лінії даних) або 8-бітному режимі (з використанням усіх 8 рядків даних) [17].

Технічні характеристики LCD 1602A:

- Напруга живлення: 4,7 ... 5,5 В
- Струм споживання: 1,2 мА
- Розмір символів: 3 х 5,23 мм
- Розмір екрану: 64 х 15 мм
- Колір підсвічування екрану: синій
- Колір символів: білий
- Розміри модуля: 80 х 36 х 11 мм

Такий дисплей можна використовувати для відображення спектрограм з

використанням псевдографічних символів (рис. 3.3), але невеликі розміри та якість зображення бажає кращого.

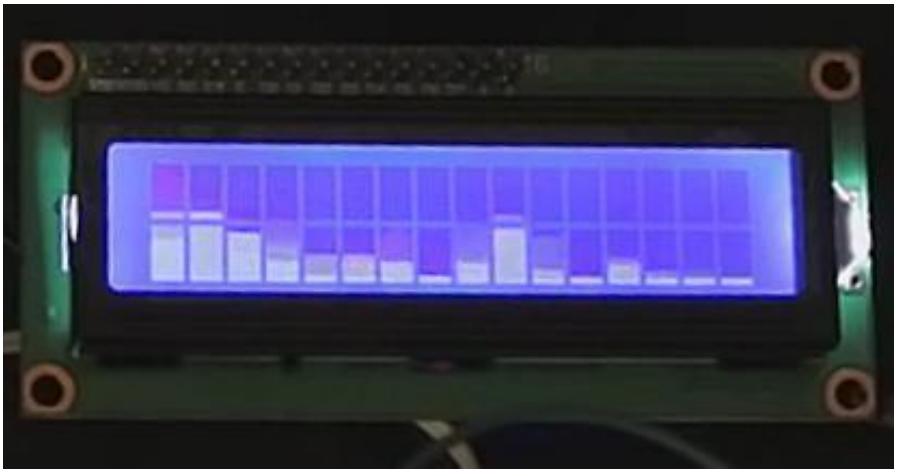

Рисунок 3.3 – Відображення спектрограми на LCD 1602

Світлодіодні матриці

Світлодіодні матриці є популярним компонентом для проектів з використанням Arduino, оскільки вони дозволяють відображати текст, зображення та анімацію. Залежно від розміру та складності, світлодіодні матриці можуть містити від кількох одиниць до сотень світлодіодів.

Світлодіодні матриці бувають:

- однокольорові — найпростіший тип, що використовує світлодіоди одного кольору;
- RGB матриці — дозволяють відображати повний спектр кольорів завдяки використанню червоних, зелених і синіх світлодіодів;
- адресовані світлодіодні матриці — містять світлодіоди, кожен з яких може бути індивідуально керований. Зазвичай базуються на чипах WS2812 або подібних.

Світлодіодні матриці бувають різних розмірів, наприклад, 8x8, 16x16, 32x8 і так далі. Розмір матриці визначається кількістю світлодіодів по горизонталі та вертикалі.

Модуль на чотири матриці поєднує на одній платі відразу чотири світлодіодні матриці 1088AS з драйверами MAX7219 в SMD корпусі [18]. Для управління світлодіодними матрицями використовується інтерфейс SPI, тобто використовується лише три лінії даних.

Кілька таких модулів можна включити послідовно, причому розміри плати відповідають розмірам матриці, то модулі можна об'єднувати в двомірні панелі (рис. 3.4).

Технічні характеристики:

- Тип пристрою: світлодіодна матриця;
- Кількість секцій: 4;
- Кількість точок: 256 (32x8);
- Діаметр точки: 3.7 мм;
- Колір світіння: червоний;
- Інтерфейс: MAX7219 (SPI);
- Робоча напруга: 5 В;
- Тип матриці: загальний катод;
- Розміри 128 x 32 x 15 мм;
- Підтримка каскадного підключення кількох модулів.

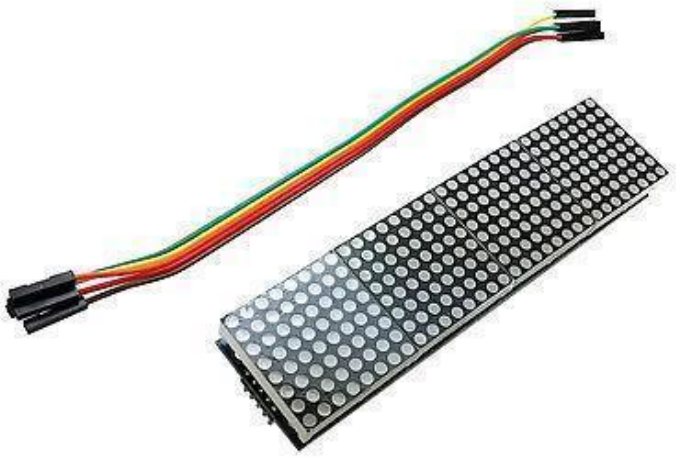

Рисунок 3.4. Світлодіодна матриця Max7219

Відображення спектрограми на такій матриці є більш інформативним: можна вивести до 32 частотних смуг (як у професійних аналізаторах спектру). Але кількість рівнів сигналу, що відображаються, не є достатнім — всього 8 (див. рис. 3.5).

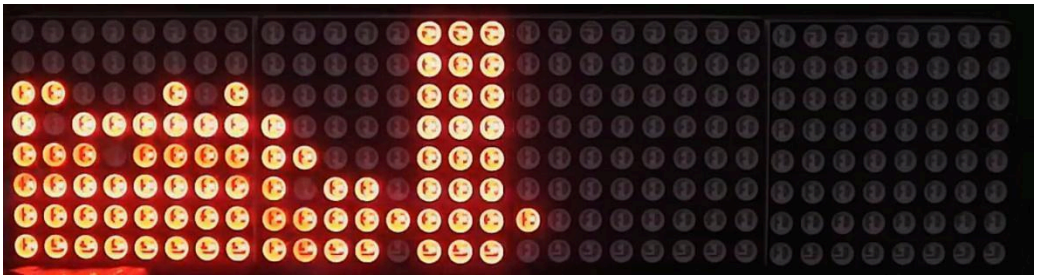

Рисунок 3.5 – Відображення аудіоспектру на світлодіодні матриці 8x32

Світлодіодна матриця на адресних світлодіодах

Вже давно не проблема придбати так звані розумні світлодіоди, або їх похідні — світлодіодні стрічки, матриці, екрани тощо. Розумні світлодіоди відрізняються від звичайних наявністю вбудованого контролера.

Кожен світлодіод оснащений вбудованим ШІМ-контролером та схемою адресації, що дозволяє керувати кожним світлодіодом незалежно. Керування здійснюється через однопровідну послідовну шину [19].

Індивідуальне керування кожним світлодіодом у стрічці або матриці дозволяє створювати красиві ефекти, відображати кольоровий текст, анімацію або піктограми, де можна засвітити будь-який "піксель". RGB світлодіоди підтримують 256 градацій яскравості кожного кольору (8 біт), що у сумі для трьох кольорів дає 256 * 256 * 256 = 16,7 мільйонів, тобто 24-бітну кольорову глибину. Таким чином, виходить повноцінний 24-бітний дисплей наднизької роздільної здатності (рис. 3.6).

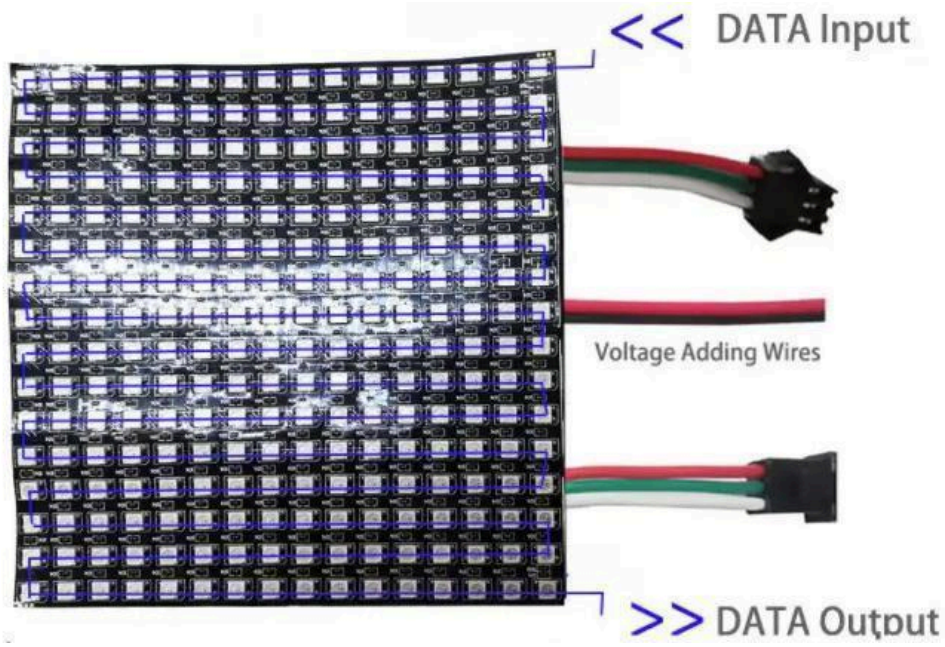

Рисунок 3.6 – Матриця на адресних RGB світлодіодах розміром 16x16

Алгоритм керування послідовністю світлодіодів WS2812 дуже простий: перший світлодіод приймає та записує у свою внутрішню пам'ять перші 24 біти інформації про яскравість кожного з трьох вбудованих у нього діодів, а всі інші біти послідовності передає наступному світлодіоду. Другий світлодіод отримає свої 24 біти та передасть залишок початкового сигналу третьому і так далі. Після передачі даних усім світлодіодам необхідно зробити паузу тривалістю 50 мкс, щоб стан світлодіодів оновився.

Технічні характеристики:

- Кількість світлодіодів: 256 шт.
- Колір світіння: RGB
- Ступінь захисту IP: 20
- Напруга: 5 В
- Мінімальна робоча температура: -25 град.
- Максимальна робоча температура: 60 град.
- Довжина: 160 мм
- Ширина: 160 мм

– Висота: 2.13 мм
– Термін служби: 50000
– Вага: 50 гр
– Колір плати (PCB): чорний

Для підключення на світлодіодної матриці розташовані три групи контактів (див. рис. 3.7).

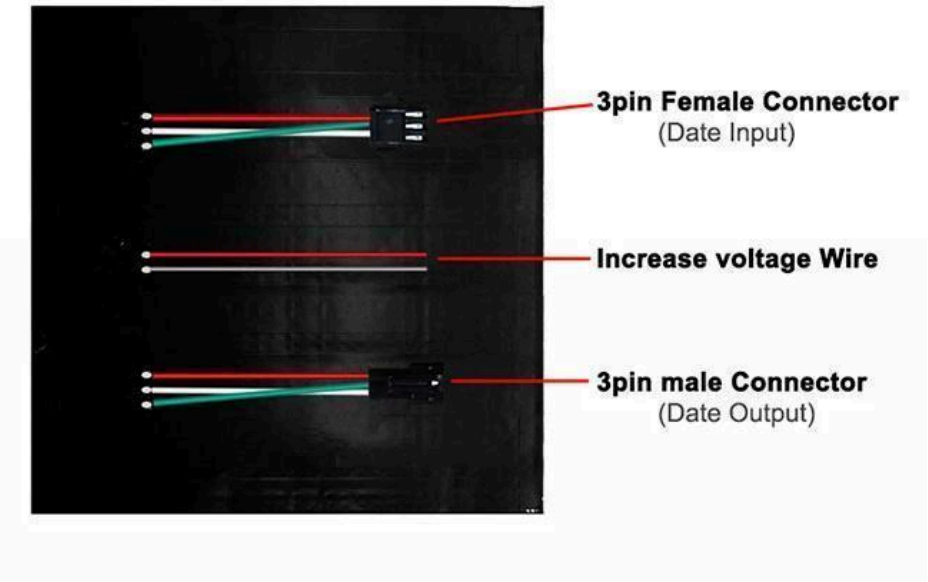

Рисунок 3.7 – Розташування контактів для підключення Smart LED матриці LT WS2812B 16x16

Перша група контактів використовується для з'єднання матриці з мікроконтролером:

– сигнальний (DI) — цифровий вхід матриці. Під'єднується до будь-якого цифрового виходу мікроконтролера;
– живлення (V) — живлення модуля. Під'єднується до живлення мікроконтролера;
– земля (G) — земля модуля. З'єднується із землею мікроконтролера.

Якщо є потреба з'єднати кілька матриць у ланцюжок (гірлянду), використовується друга група контактів:

- сигнальний (DO) — цифровий вихід матриці. З'єднується із цифровим входом наступної матриці;
- живлення (V) — живлення лінії;
- земля (G) — земля.

Під час використання гірлянди з матриць потрібно переконатися в працездатності схеми живлення. Не можна подати живлення всій гірлянди тільки через першу матрицю — доріжки занадто тонкі для великого струму. Потрібно під'єднати живлення до кожної матриці окремо, через третю групу контактів:

- живлення (V) — живлення лінії;
- земля (G) — земля.

Використання матриці на адресованих світлодіодах розміром 16x16 є найкращим варіантом для відображення спектрограми. Така матриця забезпечує достатню кількість рівнів (16) на частоту, дозволяє відображати до 16 частотних смуг і використовувати різні кольори для візуалізації, наприклад, пікових сигналів. Крім того, за потреби можна з’єднати кілька матриць у «гірлянду», збільшивши кількість рівнів і частотних «стовпчиків».

Основними недоліками таких матриць є висока ціна та значне споживання струму.

4. РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Розробка схеми пристрою

В розробляємому аналізаторі аудіо спектра буде використовуватися лише плата Arduino Nano, без застосування будь-яких додаткових мікросхем.

Напруга, що знімається з лінійного виходу звукового пристрою або виходу на навушники досить низька — у межах 1 В. Arduino Nano за замовчуванням оцифровує напруги від 0 до 5 вольтів, верхній поріг в 5 вольт потрібно понизити для більшої точності оцифрування на не високій гучності. Зробити це можна шляхом зниження опорної напруги АЦП. На практиці це реалізується в такий спосіб: з'єднуємо вхід опорної напруги AREF с виходом 3,3 вольта через дільник напруги на резисторах як на схемі (рис. 4.1).

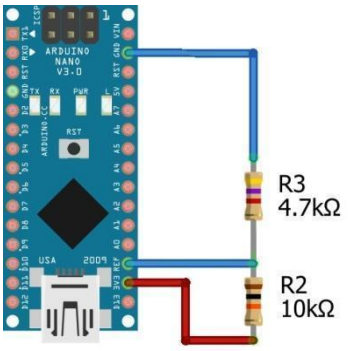

Рисунок 4.1 – Схема підключення зовнішнього джерела опорної напруги АЦП

Тепер максимальна напруга для оцифрування складе близько 1,2 вольта. Це дасть можливість працювати з таким слабким аудіо сигналом, як вихід на навушники або лінійний аудіо-вихід.

У якості обладнання для відображення спектрограм буде використовуватися RGB-матриця на адресних світлодіодах розміром 16x16. Потрібно врахувати, що при високій яскравості світіння, матриця споживає достатньо великий струм і вимагає окремого джерела живлення 5В, що забезпечує струм не менше 3А.

Аудіосигнал бажано подавати на аналоговий вхід мікроконтролера через конденсатор, щоб унеможливити попадання постійної складової, яка може бути присутня у вхідному сигналі.

Повна схема аналізатора спектра показу на рисунку 4.2.

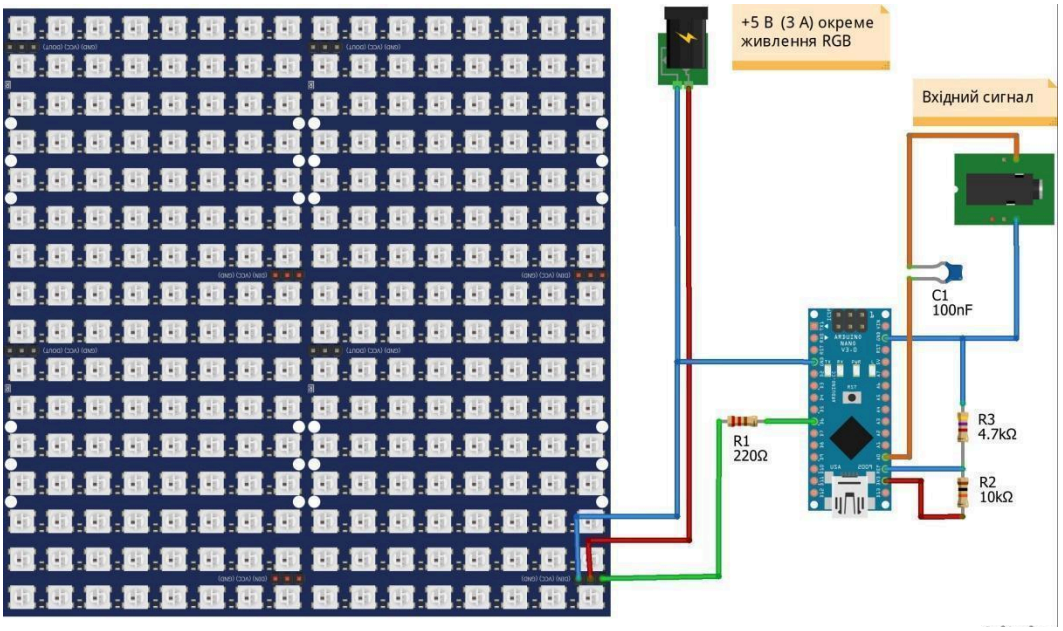

Рисунок 4.2 – Схема звукового аналізатора спектру

4.2 Отримання та формування частотного спектру

Налаштування роботи АЦП

Як уже згадувалося раніше, частота дискретизації аналогових входів за замовчуванням становить близько 9600 герців і по теоремі Найквіста можна оцифрувати частоту у два рази менше, тобто всього близько 5 кГц. У той же час, діапазон звукових частот простирається до 20 кГц і встановленої частоти дискретизації явно не достатньо для нормальної роботи аналізатора спектру.

Використовуючи дані з таблиці 3.2, можна підібрати й встановити у void setup() передмасштабний коефіцієнт так, щоб частота дискретизації складала 38,4 кілогерц:

```
sbi(ADCSRA, ADPS2);
cbi(ADCSRA, ADPS1);
sbi(ADCSRA, ADPS0);
```

Це дасть можливість оцифрувати увесь звуковий діапазон. При цьому на один стовпчик при середній гучності припадає глибина близько 6 біт, а точніше 60 рівнів виміру гучності. Це дає можливість виводити значення на матрицю з висотою стовпчика у 64 точки.

Тестування роботи бібліотеки ШПХ

Оскільки в проекті використовується бюджетна модель Arduino Nano, яка не має високої швидкодії, то застосування ШПФ може не дати бажаного результату. Тому буде використано швидке перетворення Хартлі. Для платформи Arduino існує готова бібліотека для оцифрування звуку. Її розробник також пропонує варіант швидкого перетворення Хартлі.

Ця бібліотека дозволяє отримати спектр звуку у вигляді набору чисел (рис. 4.3).

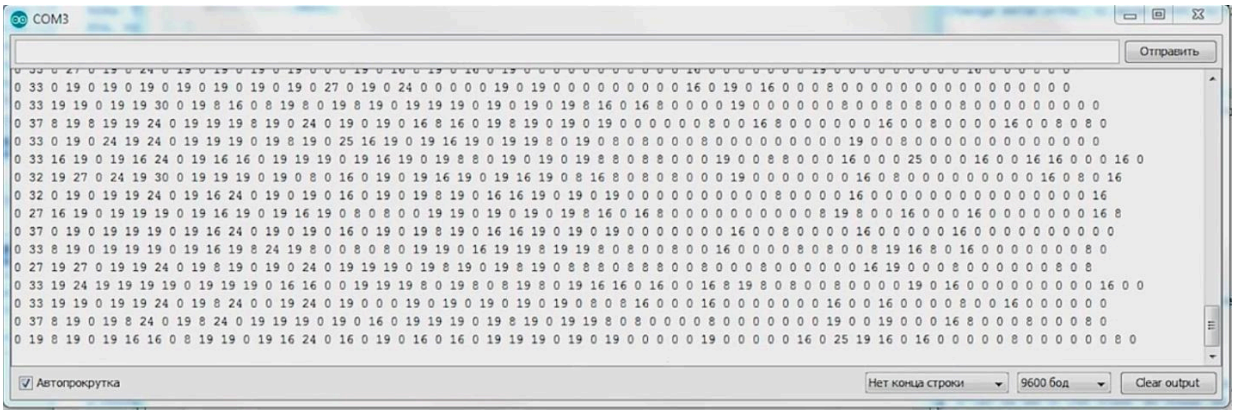

Рисунок 4.3 – Числові дані, отримані в реальному часі за допомогою ШПХ

Чим більше число, тим більше вага певної частоти в спектрі. Розкладання в спектр іде лінійно з деяким кроком по частоті. Продуктивності Arduino вистачає на 128 смуговий спектр.

Вибір та формування частотних смуг

Як правило, частота в аналізаторах росте нелінійно. Спочатку докладно виводиться низькочастотна область, у якій грає більшість музичних інструментів, потім уже відображаються високі частоти.

Оскільки, швидке перетворення Хартлі дає лінійну розбивку по смугах частот, то проміжки між стовпчиками потрібно встановлювати вручну так, щоб приблизно до середини були низькі частоти, а потім уже все інше:

```
byte posoffset[17] = {2, 3, 4, 6, 8, 10, 12, 14, 16, 20, 25, 30,
35, 60, 80, 100, 120}; // для 16 смуг
```

Щоб переконатися в правильності роботи з розкладання спектра можна використати генератор звукових частот. Плавно змінюючи вихідну частоту, відслідковують, які індикаторні стовпчики на неї реагують. Якщо дозволяє генератор, то можна задати декілька частот одночасно. У цьому випадку на спектроаналізаторі вони повинні відобразитися у вигляді максимумів.

Оскільки розкладання в спектр відбувається лінійно на 128 смуг, то вибірка окремих смуг (16 шт.) приводить до того, що інші смуги просто відкидаються й ні як не відображаються аналізатором. При прогоні всього звукового діапазону за допомогою генератора це добре видно.

Якщо, приміром, подивитися, як відбувається виділення частот в мікросхемі графічного еквалайзера MSGEQ7 [20], те діаграма частот виглядає як на рисунку 4.4.

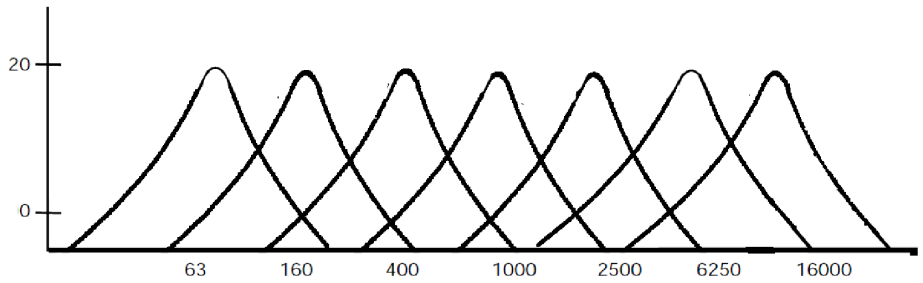

Рисунок 4.4 – Діаграма розподілу частот у мікросхемі MSGEQ7

Тобто на одну смугу, що відображається, береться досить широкий діапазон частот.

Щоб вирішити цю проблему в розроблювальному спектроаналізаторі довелося додати окремий алгоритм. Робота якого полягає в тому, що він буде брати величину сусідніх (не показуваних) частотних смуг і підсумовувати їх з деяким коефіцієнтом в одну смугу, у наслідку вийде так само, як в MSGEQ7. При цьому можна вибрати будь-яку кількість смуг, що відображаються.

```
byte linesbetween;
if (pos > 0 && pos < WIDTH) {
  linesbetween = posoffset[pos] - posoffset[pos - 1];
  for (byte i = 0; i < linesbetween; i++) {   // від попередньої
смуги до поточної
    poslevel += (float) ((float)i / linesbetween) *
fht_log_out[posoffset[pos] - linesbetween + i];
  }
  linesbetween = posoffset[pos + 1] - posoffset[pos];
  for (byte i = 0; i < linesbetween; i++) {   // від попередньої
смуги до поточної
    poslevel += (float) ((float)i / linesbetween) *
fht_log_out[posoffset[pos] + linesbetween - i];
  }
}
```

4.3 Розробка додаткових опцій відображення спектрограми

Регулювання рівня сигналу

Ще одна проблема, яку довелося вирішувати — це робота при різному рівні гучності. Чим тихіше музика на вході, тем нижче стовпчики. Можна регулювати гучність, додавши в схему потенціометр, але настроювати його вручну не завжди зручно. Більш кращий варіант — зробити адаптивне настроювання висоти стовпчиків по самому довгому з них.

```
// ручне настроювання рівня гучності
if (MANUAL_GAIN) gain = map(analogread(POT_PIN), 0, 1023, 0,
150);
// авто настроювання рівня гучності
if (AUTO_GAIN) {
```

```
  if (millis() - gaintimer > AGT) { // кожні 10 мс за
замовчуванням
    maxvalue_f = maxvalue * k + maxvalue_f * (1 - k);
    // якщо максимальне значення більше порога, взяти його як
максимум для відображення
    if (maxvalue_f > LOW_PASS) gain = (float) MAX_COEF *
maxvalue_f;
    // якщо ні, ті взяти поріг більше, щоб шуми взагалі не
проходили
        else gain = 100;           //100 за замовчуванням
        gaintimer = millis();
      }
    }
```

Тепер усе відображається відмінно при будь-якому рівні гучності — система підлаштовується сама.

Керування плавністю відображення

У більшості спектроаналізаторів індикаторні стовпчики «ростуть» і зменшуються. У розроблювальному спектроаналізаторі вони просто спалахують і гаснуть, що робить загальну картину дуже різкою й не зручною для сприйняття. Для усунення цього довелося додати запізнювальний фільтр, який ураховує попереднє значення сигналу. Завдяки цьому відображення стало плавним.

```
// фільтрація довжини стовпчиків, для їхнього плавного руху
      poslevel = poslevel * SMOOTH + poslevel_old[pos] * (1 -
SMOOTH);
      poslevel_old[pos] = poslevel;
```

Запізнювання, а відповідно, і плавність можна задати за допомогою значення змінної SMOOTH у початкових налаштуваннях програми.

Відображення точок максимумів

Найбільш просунуті спектроаналізатори можуть відображати точки максимумів по кожній смузі частот. Така інформація буває корисною при аналізі недостатньо якісних фонограм. Для реалізації цієї функції потрібно зробити пошук максимумів і задати часові проміжки, на яких ці максимуми

визначаються. Також встановити час, в продовж якого будуть відображатися й «падати» максимуми.

```
      if (poslevel > 0 && poslevel > maxlevel[pos]) { // якщо
для цієї смуги є максимум, який більше попереднього
        maxlevel[pos] = poslevel;         // запам'ятати його
        timelevel[pos] = millis();        // запам'ятати час
      }

// якщо крапка максимуму вище нуля (або дорівнює йому) -
увімкнути піксель
      if (maxlevel[pos] >= 0 && MAX_DOTS) {
if (SNAKE)
        if (pos % 2 != 0)                 // якщо парний рядок
          if (REVERSE) leds[pos * HEIGHT + HEIGHT -
maxlevel[pos] - 1] = MAX_COLOR;   // заповнюємо у
зворотному/прямому порядку
          else leds[pos * HEIGHT + maxlevel[pos]] = MAX_COLOR;
        else                      // якщо непарний
          if (REVERSE) leds[pos * HEIGHT + maxlevel[pos]] =
MAX_COLOR;      // заповнюємо в прямому/зворотному порядку
          else leds[pos * HEIGHT + HEIGHT - maxlevel[pos] - 1] =
MAX_COLOR;
else
          if (pos % 2 != 0)               // якщо парний рядок
          if (REVERSE) leds[pos * HEIGHT + maxlevel[pos]] =
MAX_COLOR;        // заповнюємо в прямому/зворотному порядку
          else leds[pos * HEIGHT + HEIGHT - maxlevel[pos] - 1] =
MAX_COLOR;
        else                              // якщо непарний
          if (REVERSE) leds[pos * HEIGHT + maxlevel[pos]] =
MAX_COLOR;      // заповнюємо в прямому/зворотному порядку
          else leds[pos * HEIGHT + HEIGHT - maxlevel[pos] - 1] =
MAX_COLOR;
      }

      if (fallflag) {             // якщо падає на крок
        if ((long)millis() - timelevel[pos] > FALL_PAUSE) {
// якщо максимум тримався на своїй висоті довше FALL_PAUSE
          if (maxlevel[pos] >= 0) maxlevel[pos]--; // зменшити
висоту крапки на 1
```

У результаті отримуємо звуковий аналізатор спектра у всьому чутному діапазоні частот із плавною анімацією й автоматичним настроюванням під рівень вхідного сигналу, плавним відображення частотних смуг і їх максимумів.

Оскільки для відображення спектра використовується RGB матриця на адресних світлодіодах, то кожний стовпчик розбитий на чотири частини, кожна має свій колір (див рис.4.5). Кольори можна настроїти, повний список доступних кольорів можна знайти у файлі бібліотеки. Окремо можна настроїти колір точок максимуму (рис.4.6).

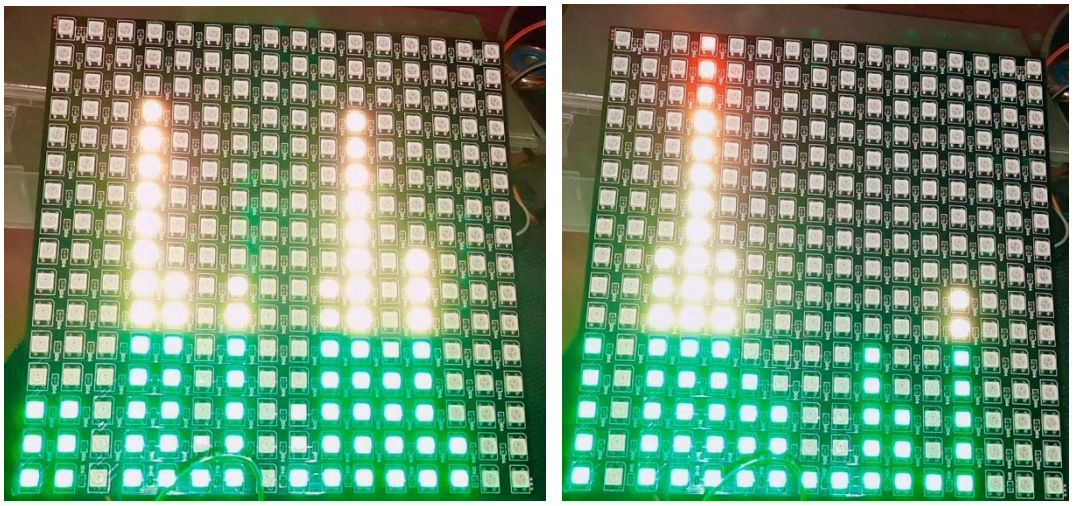

Рисунок 4.5 – відображення спектрограм без точок максимумів

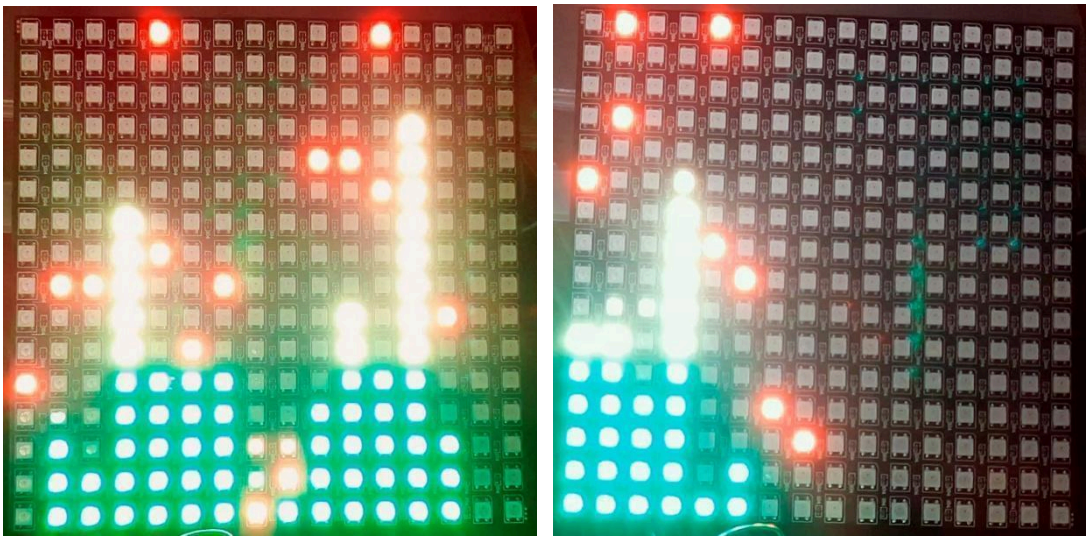

Рисунок 4.6 – відображення спектрограм з точками максимуму.

ВИСНОВКИ

У результаті виконання дипломної роботи досягнуто наступних результатів.

- Проведено огляд існуючих рішень для аналізу спектру звукових сигналів.
- Обрано платформу Arduino для реалізації проекту завдяки її широким можливостям для роботи з апаратними та програмними компонентами.
- Створено схему пристрою на основі платформи Arduino.
- Використано АЦП Arduino для обробки звукових сигналів.
- Впроваджено LED-матрицю для відображення спектрограм з використанням адресних світлодіодів, що дозволило застосовувати різні кольори в залежності від рівню сигналу.
- Розроблено програмне забезпечення для зчитування та обробки звукових сигналів.
- Реалізовано алгоритми для отримання та формування частотного спектру за допомогою бібліотеки швидкого перетворення Хартлі (ШПХ).
- Впроваджено функції регулювання рівня сигналу, плавного відображення частотних смуг та точок максимумів.
- Проведено тестування роботи пристрою та програмного забезпечення.
- Оптимізовано роботу АЦП для прискорення зчитування даних та покращення точності відображення спектру.

Загалом, було створено функціональний звуковий аналізатор спектру, який охоплює весь чутний діапазон частот. Пристрій забезпечує плавну анімацію та автоматичне налаштування під рівень вхідного сигналу, що робить його зручним для використання в різних аудіосистемах. Завдяки використанню адресних світлодіодів, забезпечується чітке та яскраве відображення частотних смуг, що дозволяє легко візуалізувати звукові сигнали.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Спектроанализатор – что мы на нем видим? [Електронний ресурс] – Режим доступу: https://prosound.ixbt.com/education/spektr-analys.shtml;
2. Теория звука. Что нужно знать о звуке, чтобы с ним работать. [Електронний ресурс] – Режим доступу: https://habr.com/ru/company/yandex/blog/270765/;
3. Desmos Studio Графічний калькулятор [Електронний ресурс] – Режим доступу: www.desmos.com/calculator/aojmanpjrl;
4. Дискретизація [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Sampling_(signal_processing);
5. Аліасинг [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Аліасинг
6. Гучність звуку [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Гучність_звуку
7. Comparison of computational costs of Hartley transform and Fourier transform. 2016 13th International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE). DOI: 10.1109/APEIE.2016.7802196
8. Слюсар, В. И. Метод неортогональной частотной дискретной модуляции на основе преобразования Хартли с квадратурной амплитудной модуляцией частотных несущих /В. И. Слюсар, В. Г. Смоляр // Системы обработки информации. – 2008. – Вып. 2. – С. 102–104.
9. TM10PA, Светодиодный 5-ти полосный анализатор спектра [Електронний ресурс] – Режим доступу: https://www.chipdip.kz/product/tm10pa
10. LINK1 AK1616 16 Level LED indicator Audio Music spectrum VU Meter Stereo Amplifier Board Adjustable light Speed Board [Електронний ресурс] – Режим доступу: https://www.aliexpress.com/item/32892564364.html
11. Цифровий аналізатор спектру звукового сигналу MasterKit MP1205) [Електронний ресурс] – Режим доступу:

https://electronoff.ua/good/cifrovoj-analizator-spektra-zvukovogo-signala-mp1205.php

12. Анализатор спектра и эквалайзер GEEKTONE LED2015 с DSP (Music Spectrum Level Equalizer). [Електронний ресурс] – Режим доступу: https://mysku.club/blog/china-stores/98888.html
13. DIY FFT Audio Spectrum Analyzer [Електронний ресурс] – Режим доступу: https://create.arduino.cc/projecthub/mircemk/diy-fft-audio-spectrum-analyzer-ca2926?ref=user&ref_id=168805&offset=0
14. Arduino DOCS Hardware [Електронний ресурс] – Режим доступу: https://docs.arduino.cc/hardware/
15. Microcontroller ATmega328-328P. Datasheet [Електронний ресурс] – Режим доступу: https://cdn.sparkfun.com/assets/c/a/8/e/4/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf
16. Fast sampling from analog input [Електронний ресурс] – Режим доступу: http://yaab-Arduino.blogspot.com/2015/02/fast-sampling-from-analog-input.html
17. LCD 1602A V2.0 синий фон белые символы с подсветкой [Електронний ресурс] – Режим доступу: https://3v3.com.ua/product_5128.html
18. Модуль на чотири матриці 1088AS із драйверами MAX7219 (SMD) [Електронний ресурс] – Режим доступу: https://www.mini-tech.com.ua/4-matrix-8x8-max7219%20
19. Світлодіодна матриця 16х16 см на адресных светодиодах - WS2812B 256 діода [Електронний ресурс] – Режим доступу: https://ledtechnics.dp.ua/ua/p1465476261-adresnaya-smart-svetodiodnaya.html
20. Seven Band Graphic Equalizer Data Sheet MSGEQ7 [Електронний ресурс] –Режим доступу: https://www.sparkfun.com/datasheets/Components/General/MSGEQ7.pdf

ДОДАТОК А

```
// Матриця (виставлена: початок ліворуч знизу, стрічка нагору
з'єднання зиґзаґ)
#define WIDTH 16 // ширина матриці (число діодів) кількість смуг
#define HEIGHT 16 // висота матриці (число діодів) висота смуг
#define REVERSE 1 // напрямок стовпчиків знизу - 1, зверху 0
#define SNAKE 1  // схема матриці
#define BRIGHTNESS 35 // яскравість (0 - 255) впливає на
сприйняття
#define SCHEM 1 // 0 - схема з мікрофоном без конденсатора, 1 -
схема з мікрофоном з конденсатором  або лінійний вхід
#define OFFSET 1.22 // постійна складова на вході AUDIO_IN при
підключенні max9814 напряму, не потрібно,якщо з лінійного входу
#define REFERENCE 2.06 // напруга aref
// Кольори висоти смуг спектра.
//кольори за замовчуванням
#define COLOR1 0x007D1C  //CRGB::Green
#define COLOR2 0x007D1C  //CRGB::Yellow
#define COLOR3 0x007D1C  //CRGB::Orange
#define COLOR4 CRGB::Red
// колір точок максимуму
#define MAX_COLOR CRGB::Orange
/*Кольори з бібліотеки
   CRGB::Red
   CRGB::Orange
   CRGB::Yellow
   CRGB::Green
   CRGB::Aqua
   CRGB::Blue
   CRGB::Purple
   CRGB::Pink - може відображатися як білий
   0xff4500 - більш природній жовтогарячий, в hex
   0x007D1C - колір люмінесцентного індикатору, в hex
 */
//Висота кольорів смуг
#define LEV1 5
#define LEV2 7
#define LEV3 8
// сигнал
#define INPUT_GAIN 1.4 // коефіцієнт підсилення вхідного сигналу
1,5
#define LOW_PASS 34    // нижній поріг чутливості шумів (немає
стрибків при відсутності звуку) 30 за замовчуванням
#define MAX_COEF 1.1   // коефіцієнт пікових значень
#define NORMALIZE 0    // нормалізувати піки (стовпчики низьких
і високих частот будуть однакової довжини при однаковій
гучності) (1 ввімкнути, 0 вимкнути за замовчуванням)
// анімація
#define SMOOTH 0.3     // плавність руху стовпчиків, менше –
```

Лістинг А1, лист 1

```
плавніше, більше - різкіше (0 - 1)0,3
#define DELAY 4         // затримка між відновленнями матриці
(періодичність основного циклу), мілісекунди 4
// гучність
#define DEF_GAIN 50     // максимальний поріг за замовчуванням (
при MANUAL_GAIN або AUTO_GAIN ігнорується)
#define MANUAL_GAIN 0   // ручне настроювання потенціометром на
гучність (1 ввімкнути, 0 вимкнути)
#define AUTO_GAIN 1     // автоналаштування по гучності
(експериментальна функція) (1 ввімкнути, 0 вимкнути)
#define AGT 30          // таймер відліку для виміру рівня
автоналаштування гучності, за замовчуванням 10 мс (більше
значення дає менше сплесків перешкод у тиші й більш заповнену
картину смуг)
// точки максимуму
#define MAX_DOTS 1      // увімкнути/виключити відображення точок
максимуму (1 ввімкнути, 0 вимкнути)
#define FALL_DELAY 50   // швидкість падіння точок максимуму
(затримка, мілісекунди) 60
#define FALL_PAUSE 200 // пауза перед падінням точок максимуму,
мілісекунди 700
// масив тонів, від 80 Гц до 16 кГц
byte posoffset[17] = {2, 3, 4, 6, 8, 10, 12, 14, 16, 20, 25, 30,
35, 60, 80, 100, 120}; // для 16 смуг
#define POTENT 1    // 1 - використовуємо резистор, 0 -
використовується внутрішнє джерело опорної напруги 1.1 В
#define AUDIO_IN 2 // пін, куди підключений звук
#define DIN_PIN 12 // пін Din стрічки ( через резистор 220 Ом!)
#define POT_PIN 7   // пін потенціометра настроювання (якщо
потрібний MANUAL_GAIN)
#if (SCHEM)
 #define OFFSET 0
#endif
#define NUM_LEDS WIDTH * HEIGHT
#define FHT_N 256       // ширина спектра x2
#define LOG_OUT 1
#include <FHT.h>        // перетворення Хартлі
#include <Fastled.h>
CRGB leds[NUM_LEDS];
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
int gain = DEF_GAIN;    // посилення за замовчуванням
unsigned long gaintimer, falltimer;
byte maxvalue;
float k = 0.05, maxvalue_f = 0.0;
int maxlevel[HEIGHT];
byte poslevel_old[WIDTH];
unsigned long timelevel[WIDTH], maindelay;
boolean fallflag;
void setup() {
  sbi(ADCSRA, ADPS2);
```

Лістинг А1, лист 2

```
  cbi(ADCSRA, ADPS1);
  sbi(ADCSRA, ADPS0);
  if (POTENT) analogreference(EXTERNAL);
  else        analogreference(INTERNAL);
  Serial.begin(9600);
  Fastled.setbrightness(BRIGHTNESS);
  Fastled.addleds<WS2812B, DIN_PIN, GRB>(leds, NUM_LEDS);
}
void loop() {
if (millis() - maindelay > DELAY) {
    maindelay = millis();
    analyzeaudio();
    for (int i = 0; i < 128; i ++) {
      if (fht_log_out[i] < LOW_PASS) fht_log_out[i] = 0;
      fht_log_out[i] = (float)fht_log_out[i] * INPUT_GAIN;
      if (NORMALIZE) fht_log_out[i] = (float)fht_log_out[i] /
((float)1 + (float)i / 128);
    }
    maxvalue = 0;
    Fastled.clear();
    for (byte pos = 0; pos < WIDTH; pos++)
      int poslevel = fht_log_out[posoffset[pos]];
      byte linesbetween;
      if (pos > 0 && pos < WIDTH) {
        linesbetween = posoffset[pos] - posoffset[pos - 1];
        for (byte i = 0; i < linesbetween; i++) {
          poslevel += (float) ((float)i / linesbetween) *
fht_log_out[posoffset[pos] - linesbetween + i];
        }
        linesbetween = posoffset[pos + 1] - posoffset[pos];
        for (byte i = 0; i < linesbetween; i++) {
          poslevel += (float) ((float)i / linesbetween) *
fht_log_out[posoffset[pos] + linesbetween - i];
        }
      }
      if (poslevel > maxvalue) maxvalue = poslevel;
      poslevel = poslevel * SMOOTH + poslevel_old[pos] * (1 -
SMOOTH);
      poslevel_old[pos] = poslevel;
      if (MAX_DOTS) {poslevel = map(poslevel, LOW_PASS, gain, 0,
HEIGHT-1);
                     poslevel = constrain(poslevel, 0, HEIGHT-1);
        }
      else {poslevel = map(poslevel, LOW_PASS, gain, 0, HEIGHT);
           poslevel = constrain(poslevel, 0, HEIGHT);
      }
      if (poslevel > 0)
        for (int j = 0; j < poslevel; j++) {
          uint32_t color;
          if (j < LEV1) color = COLOR1;           //5
          else if (j < LEV2) color = COLOR2;      //10
```

Лістинг А1, лист 3

```
          else if (j < LEV3) color = COLOR3;      //13
          else if (j < HEIGHT-1) color = COLOR4;  //15
if (SNAKE)
          if (pos % 2 != 0)
            if (REVERSE) leds[pos * HEIGHT + HEIGHT - j - 1] =
color;
            else leds[pos * HEIGHT + j] = color;

          else
            if (REVERSE) leds[pos * HEIGHT + j] = color;
            else leds[pos * HEIGHT + HEIGHT - j - 1] = color;
else
          if (pos % 2 != 0)
            if (REVERSE)leds[pos * HEIGHT + j] = color;
            else leds[pos * HEIGHT + HEIGHT - j - 1] = color;
          else
            if (REVERSE) leds[pos * HEIGHT + j] = color;
            else leds[pos * HEIGHT + HEIGHT - j - 1] = color;
        }
      }
      if (poslevel > 0 && poslevel > maxlevel[pos]) {
        maxlevel[pos] = poslevel;
        timelevel[pos] = millis();
      }
      if (maxlevel[pos] >= 0 && MAX_DOTS) {
if (SNAKE)
        if (pos % 2 != 0)
          if (REVERSE) leds[pos * HEIGHT + HEIGHT -
maxlevel[pos] - 1] = MAX_COLOR;
          else leds[pos * HEIGHT + maxlevel[pos]] = MAX_COLOR;
        else
          if (REVERSE) leds[pos * HEIGHT + maxlevel[pos]] =
MAX_COLOR;
          else leds[pos * HEIGHT + HEIGHT - maxlevel[pos] - 1] =
MAX_COLOR;
else
          if (pos % 2 != 0)
          if (REVERSE) leds[pos * HEIGHT + maxlevel[pos]] =
MAX_COLOR;
          else leds[pos * HEIGHT + HEIGHT - maxlevel[pos] - 1] =
MAX_COLOR;
        else
          if (REVERSE) leds[pos * HEIGHT + maxlevel[pos]] =
MAX_COLOR;
          else leds[pos * HEIGHT + HEIGHT - maxlevel[pos] - 1] =
MAX_COLOR;
      }
      if (fallflag) {
        if ((long)millis() - timelevel[pos] > FALL_PAUSE) {
          if (maxlevel[pos] >= 0) maxlevel[pos]--;
        }
```

Лістинг А1, лист 4

```
      }
    }
    Fastled.show();
    fallflag = 0;
    if (millis() - falltimer > FALL_DELAY) {
      fallflag = 1;
      falltimer = millis();
    }
    if (MANUAL_GAIN) gain = map(analogread(POT_PIN), 0, 1023, 0,
150);
    if (AUTO_GAIN) {
      if (millis() - gaintimer > AGT) {
        maxvalue_f = maxvalue * k + maxvalue_f * (1 - k);
        if (maxvalue_f > LOW_PASS) gain = (float) MAX_COEF *
maxvalue_f;
        else gain = 100;
        gaintimer = millis();
      }
    }
  }
}
void analyzeaudio() {
  for (int i = 0 ; i < FHT_N ; i++) {
    int sample = ((float)analogread(AUDIO_IN) - (OFFSET /
(REFERENCE / 1024)));
    fht_input[i] = sample;
  }
  fht_window();
  fht_reorder();
  fht_run();
  fht_mag_log();
}
/*void analyzeaudio() {
  for (int i = 0 ; i < FHT_N ; i++) {
    int sample = analogread(AUDIO_IN);
    fht_input[i] = sample;
  }
  fht_window();
  fht_reorder();
  fht_run();
  fht_mag_log();
}
```

Лістинг А1, лист 5