Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

К защите допустить:

И.О. Заведующего кафедрой информатики

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**ВИЗУАЛИЗАТОР МУЗЫКИ НА ОСНОВЕ FHT И ARDUINO**

БГУИР КП 1–40 04 01 015 ПЗ

Студент А. С. Николайчик

Руководитель А. Н. Марков

Нормоконтролёр А. А. Калиновская

Минск 2024

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc184035503)

[1 Архитектура вычислительной системы 6](#_Toc184035504)

[1.1 Структура и архитектура вычислительной системы 6](#_Toc184035505)

[1.2 История, версии и достоинства 9](#_Toc184035506)

[1.3 Обоснование выбора вычислительной системы 10](#_Toc184035507)

[1.4 Анализ выбранной вычислительной системы 12](#_Toc184035508)

[2 Платформа программного обеспечения 14](#_Toc184035509)

[2.1 Структура и архитектура платформы 14](#_Toc184035510)

[2.2 История, версии и достоинства 15](#_Toc184035511)

[2.3 Обоснование выбора платформы 16](#_Toc184035512)

[2.4 Анализ операционной системы для написания программы 17](#_Toc184035513)

[3 Теоретическое обоснование разработки программного продукта 19](#_Toc184035514)

[3.1 Обоснование необходимости разработки 19](#_Toc184035515)

[3.2 Технологии программирования, используемые для решения](#_Toc184035516)

[поставленных задач 20](#_Toc184035517)

[3.3 Связь архитектуры вычислительной системы с разрабатываемым](#_Toc184035518)

[разрабатываемым программным обеспечением 20](#_Toc184035519)

[4 Проектирование функциональных возможностей программы 23](#_Toc184035520)

[4.1 Обоснования и описание функций программного обеспечения 23](#_Toc184035522)

[4.2 Описание функциональной схемы программы 25](#_Toc184035524)

[5 Архитектура разрабатываемой программы 27](#_Toc184035525)

[5.1 Cхема подключения 27](#_Toc184035526)

[5.2 Общая структура программы 27](#_Toc184035527)

[5.3 Описание блок-схемы алгоритма программы 28](#_Toc184035528)

[5.4 Тестирование и обработка ошибок 31](#_Toc184035529)

[Заключение 33](#_Toc184035530)

[Список литературных источников 34](#_Toc184035531)

[Приложение А (обязательное) Справка о проверке на заимствования 37](#_Toc184035533)

[Приложение Б (обязательное) Листинг программного кода 37](#_Toc184035533)

[Приложение В (обязательное) Функциональная схема алгоритма,](#_Toc184035534)

[реализующего программное средство 40](#_Toc184035536)

[Приложение Г (обязательное) Блок схема алгоритма, реализующего программное средство 41](#_Toc184035537)

[Приложение Д (обязательное) Графический интерфейс пользователя 42](#_Toc184035538)

[Приложение Е (обязательное) Ведомость документов курсового проекта 43](#_Toc184035539)

# ВВЕДЕНИЕ

В современных условиях развития цифровых технологий все больше внимания уделяется системам обработки и визуализации данных. Визуализация аудиосигналов, как одна из таких областей, позволяет не только наглядно представлять звуковые колебания, но и анализировать их спектральные характеристики. Использование микроконтроллерных платформ, таких как *Arduino*, в сочетании с алгоритмами обработки сигналов, например, быстрым преобразованием Хартли (*FHT*), открывает широкие возможности для создания компактных и эффективных устройств.

Актуальность темы обусловлена растущей потребностью в доступных инструментах для анализа звуковых сигналов и визуализации их характеристик. Такие устройства находят применение в образовательных целях, для изучения основ акустики и электроники, а также в музыкальной индустрии для мониторинга и настройки звуков.

Целью является разработка визуализатора музыки на основе платформы *Arduino Uno* с применением алгоритма *FHT* для анализа звуковых сигналов. Визуализатор позволяет преобразовывать аудиосигналы, поступающие с микрофона, в их частотное представление, которое затем отображается на светодиодном матричном дисплее. Это позволяет пользователю наблюдать амплитуду различных частотных диапазонов в реальном времени.

Необходимо решить ряд задач. Во-первых, исследовать методы цифровой обработки сигналов с использованием алгоритма *FHT*. Во-вторых, разработать программное обеспечение на языке *C++* для микроконтроллера *Arduino*, которое обеспечит анализ и обработку поступающих аудиосигналов. Важной задачей также является выбор и интеграция компонентов аппаратной части, таких как микрофон и светодиодный матричный дисплей, с целью построения полноценного устройства. После разработки и интеграции компонентов предстоит выполнить отладку и тестирование устройства для обеспечения его корректной работы.

Визуализатор, созданный в рамках этой работы, может служить учебным инструментом для понимания принципов работы с аудиосигналами, их обработки и визуализации. Такой проект дает возможность применить полученные знания в области электроники, программирования и обработки сигналов. Дополнительно визуализатор может быть расширен и модифицирован, например, путем добавления фильтров, изменения алгоритмов отображения и интеграции с другими устройствами.

# 1 **АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ**

## 1.1 Структура и архитектура вычислительной системы

Для успешного создания визуализатора музыки на базе *Arduino* важно не только определить необходимые компоненты, но и понять, как они взаимодействуют между собой на уровне структуры и архитектуры системы. Далее рассмотрена структура вычислительной системы, включающая все основные элементы, а также их роль в процессе обработки и визуализации аудиосигналов.

Структура визуализатора музыки состоит из следующих компонентов, которые взаимодействуют для выполнения основных функций системы:

1 Плата *Arduino Uno R3*.

Функция устройства: плата *Arduino Uno R3* (рисунок 1.1)служит центральным процессорным элементом системы, который отвечает за обработку входящих аудиосигналов, управление визуальными компонентами.

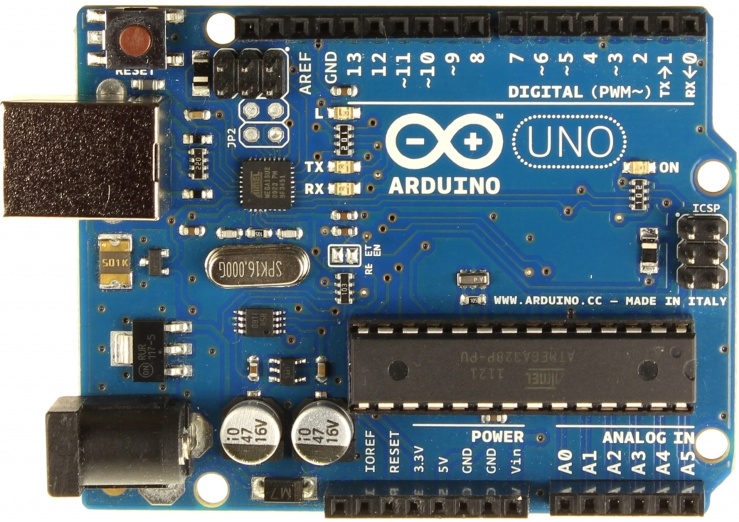


Рисунок 1.1 – Внешний вид *Arduino Uno R3*

Технические характеристики: *Arduino Uno R3* контроллер построен на *ATmega328*. Платформа имеет 14 цифровых вход/выходов (6 из которых могут использоваться как выходы ШИМ), 6 аналоговых входов, кварцевый генератор 16 МГц, разъем *USB*, силовой разъем, разъем *ICSP* и кнопку перезагрузки. Для работы необходимо подключить платформу к компьютеру посредством кабеля *USB*, либо подать питание при помощи адаптера *AC/DC* или батареи.

Для платы *Arduino Uno R3* микроконтроллером, обеспечивающим связь с компьютером по *USB*, является *ATmega16U2*. Этот микроконтроллер заменил *FTDI*-чипы, которые использовались в предыдущих моделях, и предоставляет более гибкие возможности. *ATmega16U2* прошивается с использованием встроенного загрузчика и может быть перепрограммирован, чтобы выполнять функции, отличные от *USB-to-Serial* преобразователя, что делает его универсальным. [1]

Подключение и работа: обработка аудиосигнала осуществляется с использованием алгоритма *FHT* (*Fast Hartley Transform*), написанного на языке *C++*, который преобразует аудиосигналы из временной области в частотную для их анализа. Плата управляет выводом обработанных данных на светодиодный дисплей, обеспечивая синхронизацию между звуковым спектром и визуальными эффектами.

2 Два микрофонных модуля с усилителем *MAX9814*.

Функция устройства: микрофонный модуль (рисунок 1.2) предназначен для считывания звуковых волн из окружающей среды и преобразования их в электрические сигналы, пригодные для дальнейшей обработки.

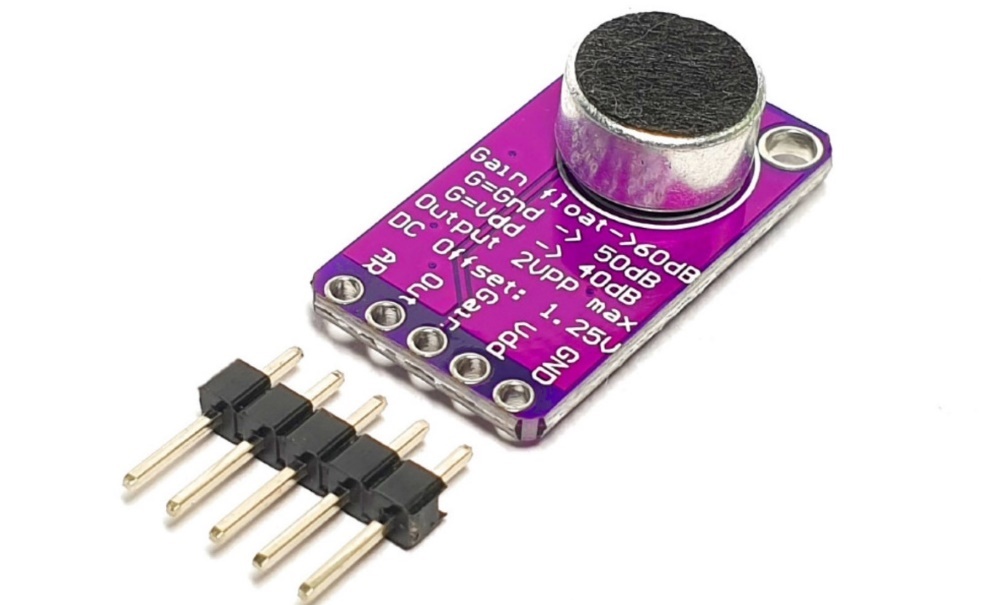


Рисунок 1.2 – Микрофонный модуль с усилителем *MAX9814*

Технические характеристики: диапазон частот от 20 Гц до 20 кГц, что соответствует слышимому диапазону человеческого уха. Встроенная система автоматической регулировки усиления (*AGC – Automatic Gain Control*), которая оптимизирует сигнал, минимизируя шумы и улучшая качество записи. [2]

Подключение и работа: подключается к аналоговому входу микроконтроллера, обеспечивая передачу аудиосигнала для дальнейшего анализа с помощью алгоритма *FHT* (*Fast Hartley Transform* – быстрое преобразование Хартли). Усиленный сигнал обрабатывается микроконтроллером для создания визуального отображения на дисплее.

3 Светодиодный матричный дисплей *MAX7219* *32x8*.

Функция устройства: дисплей (рисунок 1.3) отображает изменения амплитуды различных частот в реальном времени, позволяя наблюдать динамику звука. Каждый столбец светодиодов отображает уровень амплитуды для конкретного частотного диапазона, создавая эффект спектрограммы. Это позволяет пользователю визуально оценить звуковой сигнал и его характеристики, включая распределение частот и их интенсивность. Такое отображение полезно для анализа аудиосигналов, настройки музыкальных инструментов или наблюдения за изменениями звука в окружающей среде. Визуализация помогает быстро понять, какие частоты доминируют в сигнале и как они изменяются со временем.

Технические характеристики: светодиодный матричный дисплей *32x8* (32 светодиода по горизонтали и 8 по вертикали). Управляется через интерфейс *SPI* (*Serial Peripheral Interface*), что обеспечивает высокую скорость передачи данных и синхронизацию с микроконтроллером.

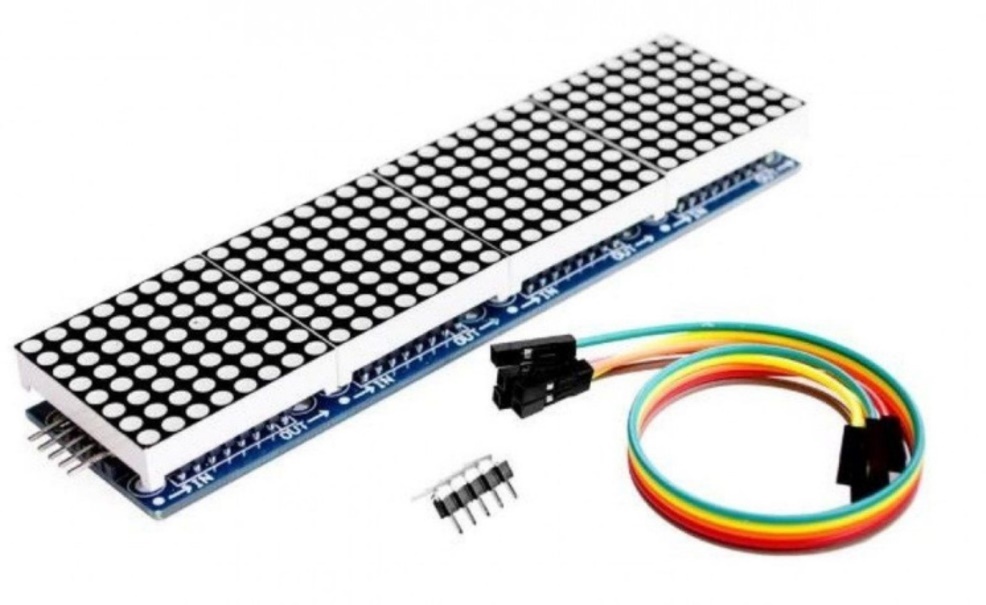


Рисунок 1.3 – Светодиодный матричный дисплей *MAX7219* *32x8*

Подключение и работа: дисплей подключается к микроконтроллеру через интерфейс *SPI*, что требует подключения следующих выводов: *VCC* (питание, обычно 5 В), *GND* (земля), *DIN* (вход данных для передачи информации), *CLK* (тактовый сигнал для синхронизации передачи данных) и *CS* (*Chip Select*, сигнал выбора микросхемы для активации дисплея). Дисплей обновляется в реальном времени, обеспечивая актуальное представление аудиосигнала, что позволяет пользователю видеть, как меняются частоты и их интенсивность. [3]

Визуализатор музыки на основе *Arduino* может быть отнесён к архитектуре микроконтроллеров, использующей принципы *RISC* (*Reduced Instruction Set Computer*). Микроконтроллер *ATmega328*, на котором основана плата *Arduino Nano*, реализует упрощённый набор инструкций, что позволяет эффективно выполнять программы с минимальными затратами ресурсов. Кроме того, данная система функционирует по фон-Неймановской архитектуре, где данные и инструкции хранятся в едином адресном пространстве. Это сочетание архитектур обеспечивает оптимальную обработку аудиосигналов, позволяя визуализировать спектр в реальном времени с высокой эффективностью.

## 1.2 История, версии и достоинства

*Arduino*– это платформа, которая произвела революцию в области электронного прототипирования и доступности технологий. Она была представлена в 2005 году как доступная платформа для прототипирования электронных устройств. С момента своего появления она прошла значительную эволюцию, адаптируясь к потребностям пользователей и технологическим изменениям:

1 В 2005 году был выпуск первой версии платы *Arduino*, основанной на микроконтроллере *ATmega8*. Это стало началом нового подхода к обучению и разработке интерактивных устройств.

2 В 2006 году с появлением *Arduino IDE* – интегрированной среды разработки – процесс программирования значительно упростился.

3 Выпуск *Arduino Uno* был в 2008 году. Плата на базе *ATmega328* обеспечила улучшенные функциональные возможности и удобство подключения через *USB*.

4 В 2010 году появилось *Arduino Mega*, обеспечивающая больше входов-выходов и оперативной памяти, что сделало её подходящей для сложных проектов, таких как управление несколькими датчиками и исполнительными механизмами.

5 Запуск *Arduino Due* – первой платы с 32-битным микроконтроллером– был в 2011 году, что расширило возможности обработки данных и вычислительной мощности.

6 В 2015 году платформа  *Arduino* официально переходит к открытой аппаратной архитектуре, позволяя сторонним разработчикам создавать совместимые устройства и платы, что способствовало расширению экосистемы *Arduino*.

7 Выпуск новых моделей был в 2019 году, таких как *Arduino Nano Every*, что свидетельствует о продолжающемся развитии платформы и её адаптации к современным требованиям. [4]

С течением времени *Arduino* развивалась, предлагая пользователям различные версии, каждая из которых обладает уникальными характеристиками и возможностями:

1 *Arduino Uno* была выпущена в 2008 году  и использует микроконтроллер *ATmega328*. Она предлагает 14 цифровых входов-выходов и 6 аналоговых входов, что делает её идеальной для простых проектов, таких как управление светодиодами.

2 *Arduino Nano* также появилась в 2008 году. Эта компактная версия с тем же микроконтроллером *ATmega328* подходит для проектов с ограниченным пространством.

3 *Arduino Mega*  была представлена  в 2010 году  и оснащена микроконтроллером *ATmega2560*. Она предлагает 54 цифровых входа-выхода и 16 аналоговых входов, что делает её подходящей для сложных проектов, таких как робототехника.

4 *Arduino Due* вышла в 2011 году и основана на 32-битном *ARM Cortex-M3*. Эта плата обеспечивает высокую вычислительную мощность для приложений, требующих интенсивной обработки данных.

5 *Arduino Leonardo* была представлена в  2012 году и использует микроконтроллер *ATmega32U4*. Она поддерживает *USB* и *HID*, что позволяет взаимодействовать с компьютером.

6 Серия *Arduino MKR* была запущена в 2016 году и ориентирована на *IoT*. Эти платы предлагают беспроводные протоколы, такие как *Wi-Fi* и *LoRa*, и подходят для интернет-приложений.

7 *Arduino  Nano Every* появилась  в 2019 году и оснащена микроконтроллером *ATmega4809*. Эта плата обладает улучшенными характеристиками и поддержкой *USB-C*, что делает её идеальной для современных проектов, требующих малых размеров и высокой производительности. [5]

*Arduino* предлагает ряд значительных преимуществ, способствующих её популярности в разработке электроники. Во-первых, среда *Arduino IDE* основана на *AVRGCC*, что позволяет пользователям изучать *C++* и при необходимости заменять высокоуровневые команды на аналогичные *C++* функции. Во-вторых, возможность питания, программирования и обмена данными с *Arduino* через один *USB*-кабель упрощает процесс работы с устройством. Кроме того, использование стандартных библиотек позволяет быстро создавать простые проекты, такие как считывание сигналов с кнопок или управление дисплеями, без необходимости глубокого погружения в программирование. Наконец, качественно реализованные последовательные и *SPI* интерфейсы обеспечивают надежную связь между компонентами. Эти факторы делают *Arduino* универсальным инструментом, подходящим как для новичков, так и для опытных разработчиков.

## 1.3 Обоснование выбора вычислительной системы

Выбор платформы *Arduino* для разработки визуализатора музыки обусловлен её доступностью, простотой использования и широкими функциональными возможностями. *Arduino* предоставляет интуитивно понятную среду разработки, что делает её оптимальным решением для реализации проектов в области электроники и программирования. Обширная библиотека готовых модулей и примеров проектов значительно ускоряет процесс разработки, позволяя сосредоточиться на функциональности системы и алгоритмах обработки сигналов, а не на решении технических вопросов, связанных с низкоуровневым программированием.

Конкретный выбор модели *Arduino Uno R3* обусловлен несколькими ключевыми факторами. Во-первых, стандартный размер и удобный форм-фактор этой платы делают её подходящей для широкого спектра проектов, включая прототипирование и разработку встроенных систем. Во-вторых, *Arduino Uno R3* оснащена 8-битным микроконтроллером *ATmega328P*, работающим на тактовой частоте 16 МГц. Это обеспечивает достаточную вычислительную мощность для обработки аудиосигналов и выполнения алгоритма *FHT*. Таким образом, плата позволяет эффективно анализировать спектр звуковых сигналов и визуализировать их на дисплее. [6]

*Arduino Uno R3* также обладает 2 КБ *SRAM* и 32 КБ флеш-памяти, что предоставляет достаточно ресурсов для хранения программы и временных данных, необходимых для обработки сигналов. Её возможности делают эту плату универсальным и надежным выбором для реализации проектов, связанных с обработкой данных и управления периферийными устройствами.

Кроме того, наличие нескольких аналоговых входов упрощает подключение микрофонных модулей, что является важным аспектом для реализации визуализатора музыки. Плата также поддерживает интерфейсы связи, такие как *I2C* и *SPI*, что обеспечивает надежную интеграцию с другими компонентами системы. Наконец, возможность программирования и обмена данными через *USB* значительно упрощает процесс разработки и тестирования, позволяя проводить быструю отладку и верификацию функционирования системы. Таким образом, *Arduino Uno R3* представляет собой оптимальный выбор для создания визуализатора музыки, обеспечивая необходимую вычислительную мощность и функциональность в компактном формате. [7]

При выборе микрофонных модулей для проекта был предпочтён *MAX9814*.

Во-первых, обеспечивается высокое качество звука благодаря встроенному усилителю, что позволяет эффективно захватывать даже тихие звуки. Это особенно важно для анализа частот с низкой амплитудой, так как данные, получаемые для обработки, будут максимально достоверными.

Во-вторых, широкий динамический диапазон модуля, охватывающий частоты от 20 Гц до 20 кГц, соответствует тому, что может слышать человек. Этот диапазон позволяет наиболее точно представлять частоты и их амплитуды, что является ключевым моментом для успешной визуализации аудиосигнала. [8]

Таким образом, выбор микрофонного модуля *MAX9814* обеспечит наиболее точные и надежные данные для алгоритма быстрого преобразования Хартли (*FHT*), что позволит создать качественное визуальное представление аудиосигнала.

Светодиодный матричный дисплей *MAX7219 32x8*, выбранный для проекта, играет ключевую роль в визуализации обработанных аудиосигналов. Он обеспечивает возможность отображения амплитуд различных частотных диапазонов в реальном времени, что позволяет пользователю наблюдать за динамикой аудиосигнала. Дисплей управляется через интерфейс *SPI*, что обеспечивает высокую скорость передачи данных и синхронизацию с микроконтроллером, что критично для своевременного отображения информации. [9]

Комбинация *Arduino Uno R3*, микрофонных модулей *MAX9814* и матричного дисплея *MAX7219* представляет собой оптимальный выбор для создания визуализатора музыки, обеспечивая необходимую вычислительную мощность, функциональность и эффективность в компактном формате.

## 1.4 Анализ выбранной вычислительной системы

В процессе разработки визуализатора аудиосигнала с использованием алгоритма быстрого преобразования Хартли (*FHT*) была выбрана вычислительная система, состоящая из микроконтроллера *Arduino Uno R3* и соответствующих периферийных устройств. Этот выбор оправдан как с точки зрения аппаратного обеспечения, так и с точки зрения удобства написания программы.

1 Микроконтроллер *Arduino Uno R3*.

Производительность. *Arduino Uno R3* на базе микроконтроллера *ATmega328P* с тактовой частотой 16 МГц обеспечивает достаточную вычислительную мощность для реализации алгоритма *FHT*. Эффективность работы программы зависит от способности микроконтроллера обрабатывать аудиосигналы в реальном времени, что вполне достижимо благодаря доступным ресурсам этой платформы.

Память. *Arduino Uno R3* предоставляет необходимые ресурсы для хранения программы и временных данных, а именно 2 КБ *SRAM* и 32 КБ флеш-памяти. Этот объем памяти позволяет эффективно реализовать алгоритмы обработки, включая *FHT*, и сохранять промежуточные результаты, что критически важно для точного анализа аудиосигналов.

Гибкость и доступность библиотек. *Arduino Uno R3* поддерживает множество библиотек, которые значительно облегчают процесс разработки. Наличие библиотек для работы с *FHT*, а также интерфейсов *I2C* и *SPI* позволяет ускорить написание кода. Это освобождает разработчика от необходимости заниматься низкоуровневой реализацией и позволяет сосредоточиться на основной логике программы. [10]

2 Периферийные устройства.

Модули микрофонов *MAX9814*. Использование микрофонного модуля *MAX9814* обеспечивает высокую чувствительность и точность захвата аудиосигналов. Для программы важно, чтобы аудиосигнал был получен и обработан с минимальными задержками. Модуль передает аналоговый сигнал, который затем конвертируется в цифровой формат для анализа.

Светодиодный матричный дисплей *MAX7219*. Дисплей используется для визуализации данных, полученных от алгоритма *FHT*. Работа с дисплеем осуществляется через *SPI*, что обеспечивает быструю передачу данных. Программа управляет дисплеем, отображая амплитуду различных частот в реальном времени, что позволяет пользователю наблюдать за изменениями звука.

3 Программное обеспечение.

Язык программирования *C++*. *C++* является основным языком программирования для разработки на платформе *Arduino*. Он предоставляет возможности для низкоуровневого управления ресурсами микроконтроллера и эффективного выполнения алгоритмов обработки данных.

Библиотеки для *FHT*. Использование готовых библиотек для быстрого преобразования Хартли позволяет значительно упростить реализацию алгоритма. Эти библиотеки предоставляют функции для обработки аудиосигналов и позволяют сосредоточиться на логике взаимодействия с другими компонентами системы.

Среда разработки *Arduino IDE*. Для написания и отладки программы используется *Arduino IDE*, которая предлагает интуитивно понятный интерфейс и встроенные инструменты для разработки. В частности, средства отладки, такие как *Serial Monitor*, позволяют отслеживать состояние программы и анализировать получаемые данные на различных этапах обработки. Среда также поддерживает интеграцию с множеством библиотек и плат расширения. [11]

Использование специализированных библиотек для обработки сигналов, таких как *FHT*, и удобной среды разработки *Arduino IDE* значительно упрощает создание сложных проектов на базе микроконтроллеров. Эти инструменты позволяют сосредоточиться на логике и функционале приложения, минимизируя затраты времени на разработку и отладку низкоуровневого кода. Благодаря интеграции готовых решений и интуитивной среде программирования ускоряется процесс разработки и повышается качество конечного продукта.

# 2 **ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## 2.1 Структура и архитектура платформы

Платформа программного обеспечения для разработки визуализатора аудиосигнала основана на среде разработки *Arduino IDE* и включает в себя ряд компонентов, обеспечивающих эффективное взаимодействие с микроконтроллером *Arduino Uno R3.*

1 Организация платформы.

*Arduino IDE* – это интегрированная среда разработки (*IDE*), предназначенная для написания, компиляции и загрузки программного обеспечения на микроконтроллеры *Arduino*. *Arduino IDE* предлагает пользователям интуитивно понятный интерфейс, позволяя легко писать и редактировать код на языке *C++*. Кроме того, среда поддерживает синтаксическую подсветку, что улучшает читаемость кода и помогает разработчикам избегать ошибок. Встроенные инструменты, такие как редактор последовательного порта, позволяют отслеживать работу программы в реальном времени и анализировать данные, передаваемые между микроконтроллером и внешними устройствами. [12]

2 Структура программных компонентов.

*Arduino* предлагает широкий выбор библиотек, которые расширяют функциональность микроконтроллера.

Библиотека *FHT*. Предоставляет функции для быстрого преобразования Хартли, что позволяет эффективно анализировать аудиосигналы и преобразовывать их в частотную область.

Библиотеки для работы с *SPI* и *I2C*. Эти библиотеки облегчают взаимодействие с периферийными устройствами, такими как светодиодные дисплеи и сенсоры, через соответствующие интерфейсы связи.

Основной скетч (код). Содержит логику визуализатора, включая инициализацию устройств, считывание данных с микрофона, обработку аудиосигнала и отображение результатов на дисплее. [13]

Функции и модули. Код организован в виде отдельных функций и модулей, что позволяет улучшить читаемость и удобство поддержки. Например, существуют отдельные функции для работы с микрофоном, обработки *FHT* и вывода данных на дисплей. Функция *setup()* выполняется один раз при запуске программы и используется для инициализации всех необходимых компонентов, таких как микрофон и дисплей. Функция *loop()* выполняется непрерывно после завершения *setup()*, и в ней реализован основной алгоритм работы визуализатора, включая считывание данных, обработку сигналов и обновление дисплея.

3 Взаимодействие компонентов.

Считывание и обработка данных. Программа начинается с инициализации микрофонного модуля, который захватывает аудиосигналы. Эти данные затем обрабатываются с использованием алгоритма *FHT*, реализованного через соответствующую библиотеку.

Визуализация данных. Результаты обработки отображаются на светодиодном матричном дисплее, управляемом через библиотеку *SPI*. Дисплей обновляется в реальном времени, показывая изменения амплитуды различных частотных диапазонов.

## 2.2 История, версии и достоинства

*Arduino IDE (Integrated Development Environment)* была создана как часть проекта *Arduino*, который начал своё развитие в 2005 году в Италии. Целью проекта было создание доступной и простой в использовании платформы, позволяющей легко программировать микроконтроллеры без глубоких знаний в области электроники и программирования.

Основатели проекта – Массимо Банци, Дэвид Куартьерс и их команда –хотели создать недорогую и легкую в освоении платформу, которая бы способствовала обучению студентов по специальностям дизайна и инженерии.   
 С самого начала проект *Arduino* включал в себя как аппаратную часть (микроконтроллерные платы), так и программное обеспечение – *Arduino IDE*. *IDE* была создана на базе существующего проекта *Wiring* (платформа для создания электронных устройств). Программное обеспечение, как и сам *Arduino*, было с открытым исходным кодом (*open source*), что позволило разработчикам со всего мира вносить изменения и улучшения в данный программный продукт.   
 Первый официальный релиз (2006) *Arduino IDE* включал минимальный набор инструментов для разработки. Среда разработки была ориентирована на пользователей с минимальными знаниями программирования и предлагала простой интерфейс. Она включала текстовый редактор для написания кода, компилятор для его преобразования в машинный код и загрузчик, который позволял записывать программный код в микроконтроллер через *USB*-интерфейс.

В течение 2008-20013 *IDE* постепенно развивалась и становилась всё более функциональной. Были добавлены: поддержка новых микроконтроллеров (в том числе *Arduino Mega*, *Leonardo* и *Due*), интеграция сторонних библиотек, что позволило пользователям легко подключать различные модули, датчики и экраны, обновления и улучшения интерфейса. [14]

К 2015 году *Arduino IDE* получила поддержку множества аппаратных платформ, включая платы на основе *ARM*, таких как *Arduino Due*, и другие системы на чипе (*SoC*). Эта поддержка значительно расширила сферу применения *Arduino*, сделав её популярной не только среди хобби-разработчиков, но и среди профессионалов.

1 *Arduino IDE 1.0*(2011).                                                                        
 Первая стабильная версия *Arduino IDE*. Основной функционал включал текстовый редактор, компилятор и загрузчик программ на микроконтроллеры. Поддерживались базовые платы, такие как *Arduino Uno* и другие модели.

2 *Arduino IDE* *1.x* (2011-2020).

Эта версия получила регулярные обновления, расширяя поддержку новых плат, таких как *Arduino Mega* и *Arduino Due*. В ней также появилась возможность подключения сторонних библиотек, что сделало работу с микроконтроллерами более гибкой и разнообразной.

3 *Arduino IDE 2.0*(2020).   
 *Arduino IDE* *2.0* представила новый, современный интерфейс, поддерживающий автодополнение кода и встроенный отладчик. Эти функции улучшили процесс написания и отладки программ, а также ускорили компиляцию и загрузку данных на микроконтроллеры.

4 *Arduino IDE* *2.x* (2020-настоящее время).

Текущая версия 2.x продолжает развиваться, внедряя новые функции и расширяя поддержку дополнительных платформ, таких как *ESP32* и *ARM*. Идея модульности и поддержка внешних разработчиков сделали её ещё более гибкой и эффективной для создания сложных проектов.

*Arduino IDE* характеризуется высокой простотой использования, что обеспечивает доступность платформы даже для начинающих разработчиков. Она функционирует на различных операционных системах, включая *Windows*, *macOS* и *Linux*.

Среда разработки поддерживает обширный набор встроенных и сторонних библиотек, что значительно упрощает интеграцию различных модулей и сенсоров в проекты. Это позволяет разработчикам сосредоточиться на логике своих приложений, минимизируя время, затрачиваемое на написание кода для базовых операций. [15]

*Arduino IDE* совместима с разнообразными микроконтроллерами и платами, включая *Arduino Nano, Uno, Mega* и *Leonardo*. Кроме того, *IDE* поддерживает возможность программирования через *API*, позволяя интегрировать её с другими инструментами и языками программирования.

Встроенные инструменты для отладки, такие как *Serial Monitor* и функции профилирования, повышают эффективность разработки, позволяя отслеживать выполнение программ и анализировать данные в реальном времени. Также стоит отметить, что *Arduino IDE* предоставляет возможность использования различных режимов компиляции и загрузки.

## 2.3 Обоснование выбора платформы

*Arduino IDE* была выбрана в качестве платформы для разработки визуализатора аудиосигнала по нескольким ключевым причинам, связанным с её функциональными возможностями и преимуществами.

1 Программирование на *C++.* Основным языком программирования в *Arduino IDE* является *C++*, что обеспечивает низкоуровневый доступ к ресурсам микроконтроллера. Это позволяет эффективно реализовывать алгоритмы обработки данных, такие как преобразование аудиосигналов. Понятный синтаксис *C++* и наличие стандартных библиотек упрощают написание и чтение кода, что особенно важно для разработки сложных приложений.

2 Инструменты для отладки. *Arduino IDE* включает в себя встроенные средства отладки, такие как *Serial Monitor*, которые позволяют отслеживать выполнение программ и анализировать данные в реальном времени. Это значительно облегчает выявление ошибок и оптимизацию кода, позволяя разработчику получать мгновенную обратную связь о состоянии системы.

3 Кроссплатформенность. *Arduino IDE* работает   на различных операционных системах, включая *Windows*, *macOS* и *Linux*. Это делает её доступной для широкого круга пользователей и позволяет разработчикам выбирать наиболее удобную для них среду для работы.

4 Библиотеки для обработки аудиосигналов. В *Arduino IDE* доступны множество встроенных и сторонних библиотек, включая библиотеки для быстрого преобразования Хартли (*FHT*). Эти библиотеки значительно упрощают реализацию сложных алгоритмов обработки аудиосигналов, позволяя разработчику сосредоточиться на логике работы приложения.

5 Поддержка микроконтроллера *Arduino Uno. Arduino IDE* полностью совместима с различными микроконтроллерами, включая *Arduino Uno*. Это дает возможность использовать все преимущества этой компактной и мощной платы, что особенно важно для проекта визуализатора аудиосигнала, где требуется экономия пространства и высокая производительность. [16]

Таким образом, *Arduino IDE* предлагает целый ряд возможностей и инструментов, которые делают её идеальным выбором для разработки проекта по визуализации аудиосигналов, обеспечивая эффективную интеграцию с необходимыми компонентами и удобство в написании кода.

## 2.4 Анализ операционной системы для написания программы

Для разработки программы визуализатора аудиосигнала была выбрана операционная система *Windows*. Этот выбор обусловлен её высокой совместимостью с *Arduino IDE*, а также наличием необходимых драйверов для работы с различными моделями плат *Arduino*, включая *Arduino Uno*. *Arduino Uno* может использовать различные чипы для преобразования *USB*-сигнала в последовательный интерфейс *UART*, что требует установки соответствующих драйверов.

Для *Arduino Uno R3* чаще используется чип ***ATmega16U2***, который программируется как *USB-to-UART* преобразователь и не требует сторонних драйверов, поскольку он распознаётся операционной системой автоматически.

Преобразователь *USB-to-UART* на основе микроконтроллера *ATmega16U2* обеспечивает связь основного микроконтроллера *ATmega328P* с компьютером. Через этот преобразователь микроконтроллер общается с компьютером по интерфейсу *UART*, используя сигналы *RX* и *TX*, которые параллельно выведены на контакты 0 и 1 платы *Arduino Uno R3*. Важно учитывать, что во время прошивки и отладки программы не рекомендуется использовать эти пины для других целей, так как это может привести к конфликтам в передаче данных и помешать успешной загрузке кода на микроконтроллер. [17]

*Windows* играет важную роль в управлении портами ввода-вывода, обеспечивая передачу данных между программным обеспечением и аппаратными компонентами. Через *Arduino IDE* *Windows* позволяет взаимодействовать с внутренними регистрами микроконтроллера *ATmega328PB*, который является мозгом *Arduino Uno R3*. Например, при работе с *GPIO* (*General Purpose Input/Output)*, цифровыми и аналоговыми выводами, *Arduino IDE* отправляет команды на микроконтроллер, которые изменяют состояние регистров:

–  *DDRx* регистр отвечает за направление пинов (на ввод или вывод);

– *PORTx* регистр используется для управления состоянием выводов (например, включение или отключение светодиода);

– *PINx* регистр используется для считывания состояние пина, если он настроен как входной.

*Windows* также управляет подключением к внешним периферийным устройствам, таким как микрофоны и дисплеи. Например, микроконтроллер, работающий с микрофоном, может собирать данные об амплитуде звукового сигнала с помощью аналогового входа *ADC* (*Analog-to-Digital Converter*), который преобразует аналоговые сигналы в цифровые. Эти данные затем могут быть обработаны в программе, а результаты визуализированы на дисплее через цифровые пины, подключенные к матрице светодиодов. [18]

В итоге, *Windows* через драйверы и *COM*-порты предоставляет надежную инфраструктуру для передачи данных и управления аппаратными компонентами, обеспечивая стабильное взаимодействие между программным кодом и аппаратной частью системы.

# 3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ

**ПРОГРАММНОГО ПРОДУКТА**

## 3.1 Обоснование необходимости разработки

Разработка программного обеспечения для музыкального визуализатора на основе платформы *Arduino* и алгоритма быстрого преобразования Хартли (*FHT*) обоснована необходимостью создания эффективного и компактного решения для визуализации аудиосигналов. Визуализаторы звука позволяют переводить аудиоконтент в визуальные образы, что обогащает восприятие музыки и делает его более увлекательным и впечатляющим.

Разработанный код включает оптимизированные алгоритмы обработки сигналов, которые выполняют спектральный анализ в реальном времени. Это позволяет отображать амплитудные колебания звука с высокой точностью.

1 Фильтрация сигналов. Код включает алгоритмы фильтрации, которые обеспечивают отсечение частот с амплитудой ниже заданного порога, определяемого в процессе калибровки. Для каждой частоты амплитуда порога фиксируется, что позволяет исключить незначительные шумы и обеспечить плавную визуализацию столбиков на матрице.

2 Оптимизация  работы с аппаратными компонентами. Применение библиотек *FHT* и *Adafruit GFX* позволяет достичь быстрого отклика и корректной отрисовки на светодиодной панели.

3 Проверка состояния системы. Программа включает механизм проверки состояния системы, который при каждом считывании данных с аналогового входа проверяет наличие подключенных микрофонов. Это позволяет своевременно обнаружить обрыв соединения или отсутствие сигнала и, при необходимости, уведомить пользователя о неисправности для устранения проблемы.

Кроме того, в разработке были учтены принципы эффективного использования памяти и процессорных ресурсов *Arduino*. Все операции с массивами и вычисления производятся так, чтобы избежать перегрузки микроконтроллера и поддерживать стабильную работу устройства при постоянной визуализации аудиосигнала.

Таким образом, разработка данного программного обеспечения была необходима для создания легко настраиваемого визуализатора звука с поддержкой реального времени, который может быть интегрирован в устройства с низким энергопотреблением и низкими вычислительными ресурсами, такими как микроконтроллеры *Arduino*.

## 3.2 Технологии программирования, используемые для решения

## поставленных задач

Для реализации проекта визуализатора аудиосигнала с использованием *Arduino Uno R3* были применены современные технологии программирования и специализированные библиотеки, обеспечивающие эффективную обработку данных и управление периферийными устройствами.

Основной используемый язык программирования  *C++*, адаптированный для платформы *Arduino.* Данный язык предоставляет низкоуровневый доступ к аппаратным ресурсам и позволяет оптимизировать код для выполнения операций с высокой скоростью. Это особенно важно при разработке приложений, работающих в режиме реального времени, таких как визуализаторы аудиосигнала.

1 *Преобразование Хартли (FHT).* Для анализа частотного  спектра аудиосигнала использована библиотека *FHT*, которая выполняет дискретное преобразование Хартли. Этот метод эффективен для микроконтроллеров, так как требует меньше вычислительных ресурсов по сравнению с другими преобразованиями (например, *FFT*). Он обеспечивает разбиение аудиосигнала на частотные компоненты, что позволяет визуализировать амплитуды различных частот.

2 *Библиотека для управления светодиодной матрицей.* Для управления светодиодной матрицей была использована библиотека *Max72xxPanel* совместно с *Adafruit\_GFX*. Библиотека *Max72xxPanel* предоставляет простой интерфейс для управления матрицей светодиодов на базе чипов *MAX7219/7221*, а *Adafruit\_GFX* отвечает за рисование графических примитивов, включая линии и пиксели, что позволяет отображать частотные столбики и максимальные пики. [19]

3 *Оптимизация алгоритмов.* В проекте применены методы фильтрации и анализа аудиосигналов для создания эффективной и плавной визуализации. Во время калибровки для каждой частоты определяется амплитуда, ниже которой сигнал будет игнорироваться, что позволяет исключать шумы и незначительные колебания. В ходе работы программа проверяет состояние микрофонов, исключая сигналы от неисправных или неработающих датчиков. Алгоритмы обработки оптимизированы для минимальной нагрузки на процессор, что обеспечивает стабильную работу устройства и точную визуализацию сигналов на матрице.

## 3.3 Связь архитектуры вычислительной системы с

## разрабатываемым программным обеспечением

Взаимодействие программного обеспечения с архитектурой вычислительной системы определяет ключевые аспекты функционирования всего проекта. В данном случае программное обеспечение разработано с учетом особенностей микроконтроллера *Arduino Uno R3*, микрофонного модуля с усилителем *MAX9814* и светодиодного матричного дисплея *MAX7219*. Все эти компоненты работают совместно, чтобы обеспечить анализ аудиосигнала и его визуализацию.

1 *Arduino Uno R3*. Микроконтроллер *ATmega328P*, который используется на плате *Arduino Uno R3*, имеет ограниченные ресурсы, такие как оперативная память и вычислительная мощность. Это требует оптимизированного программного кода, чтобы обеспечить обработку сигналов и управление периферийными устройствами.

2 *Микрофонные модули с усилителем MAX9814*. Микрофонные модули с усилителем *MAX9814* используются для захвата аудиосигналов с двух аналоговых входов *AUDIO\_IN* и *AUDIO\_IN\_2*. В проекте реализована проверка состояния микрофонов для предотвращения ошибок при их отсутствии или неисправности. Если уровень сигнала слишком низкий на одном из входов, система считает микрофон неисправным и выводит соответствующее сообщение. Для каждого из микрофонов используется подсчёт стабильных низких значений, чтобы определить, работает ли микрофон. Считываемые данные с аналоговых входов затем усредняются и передаются в массив *fht\_input[]* для дальнейшей обработки с использованием алгоритма преобразования Хартли.

3 *Светодиодный матричный дисплей MAX7219*. Дисплей управляется с использованием интерфейса *SPI* и специализированной библиотеки *Max72xxPanel*. Это позволяет отправлять данные на матрицу быстро и с минимальной нагрузкой на процессор. Программное обеспечение реализует цикл обновления визуализации, где значения амплитуд столбцов, рассчитанных после преобразования Хартли, передаются на матрицу. Для обеспечения синхронизации используется команда *matrix.write()*, которая обновляет дисплей после обработки данных. Каждый столбец матрицы отображает амплитуды, рассчитанные на основе 4 соседних значений массива *fht\_log\_out[]*. Эти 4 значения группируются, и их суммарный уровень амплитуды используется для определения высоты столбика. Определяется порог шума с помощью заранее полученных калибровочных значений (массив *calibrationValues[]*). Если амплитуда сигнала превышает порог шума, то её значение используется для расчета высоты столбика на дисплее. Уровень амплитуды усредняется и преобразуется в значение от 0 до 7 с помощью функции *map()*, чтобы соответствовать высоте столбика на матрице (матрица имеет 8 строк).

Для работы с быстрым преобразованием Хартли используется библиотека *FHT*, которая выполняет преобразование с логарифмической шкалой (*fht\_mag\_log()*), чтобы наглядно отобразить частотный спектр. Применение функций *fht\_window()* и *fht\_reorder()* обеспечивает предварительную обработку данных, снижая шумы и улучшая точность анализа.

Модуль *MAX7219* упрощает процесс управления светодиодной матрицей, позволяя программному обеспечению фокусироваться на обработке данных. Управление осуществляется через библиотеку *SPI*, что делает обмен данными быстрым и эффективным. Настройка яркости и обновление отображения производится вызовами функций библиотеки *Max72xxPanel*.

Чтобы повысить точность считывания сигнала с микрофонного модуля, используется внешний источник опорного напряжения, подключенный к пину *AREF* платы *Arduino Uno R3*. Этот пин соединен с выходом 3.3 В платы через делитель напряжения. Такое решение позволяет понизить диапазон измерений до 3.3 В, что делает АЦП более чувствительным к небольшим изменениям входного сигнала.

Пин *AREF* соединяется перемычкой с выходом 3.3 В платы. Между выводами *3V3* и *AREF* используется делитель напряжения (например, резисторы 10–20 кОм и 10 кОм), чтобы ограничить ток, проходящий по цепи. [20]

Стандартный предделитель частоты для АЦП в *Arduino* установлен на значение 128, что приводит к частоте выборки около 9.6 кГц (при рабочей частоте процессора 16 МГц). Однако, поскольку в коде происходит два последовательных чтения с разных аналоговых входов, частота выборки фактически уменьшается в два раза, что снижает её до примерно 4.8 кГц. Согласно теореме Котельникова, для корректной оцифровки сигналов без потерь, максимальная частота сигнала, которую можно безопасно оцифровать, составляет 2.3 кГц. Частоты выше этой величины приводят к эффекту наложения, или алиасингу, при котором высокочастотные компоненты сигнала могут быть неверно интерпретированы как низкочастотные.

Интеграция ключевых компонентов проекта, включая микроконтроллер *Arduino Uno R3*, микрофонные модули с усилителем *MAX9814* и светодиодный матричный дисплей *MAX7219*, позволяет эффективно реализовать систему аудиоанализа и визуализации, обеспечивая точное захватывание звуковых сигналов и их наглядное отображение в реальном времени. Использование оптимизированного программного кода позволяет эффективно обрабатывать аудиосигналы и управлять дисплеем при ограниченных вычислительных ресурсах микроконтроллера. Микрофоны захватывают аудиосигналы и обеспечивают проверку работоспособности, а данные усредняются для повышения точности. Визуализация частотного спектра на дисплее происходит с помощью библиотеки *Max72xxPanel*, что позволяет отображать амплитуды звуковых волн с минимальной нагрузкой на процессор. Проект демонстрирует успешную интеграцию аппаратных и программных средств для создания эффективной системы аудиоанализа и визуализации.

# 4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ

# ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

## 4.1 Обоснования и описание функций программного

## обеспечения

Программа представляет собой аудиовизуализатор, который считывает входной аудиосигнал, проводит его спектральный анализ и визуализирует данные в виде динамических столбиков, отражающих амплитуду различных частотных составляющих.

Основная цель программы  преобразование звукового сигнала в визуальную форму, позволяющую наблюдать изменение спектра звука в реальном времени. Это достигается с помощью преобразования Хартли (*FHT*), которое используется для разложения аудиосигнала на спектральные компоненты. Отображение происходит на основе значений, полученных после анализа, с применением различных параметров для фильтрации шума, усиления сигнала.

1 Калибровка.

Калибровка микрофонов осуществляется в начале работы устройства с целью определения уровня шума окружающей среды и установки индивидуальных пороговых значений для каждой частоты, которые будут использоваться для фильтрации слабых сигналов.

В процессе калибровки производится сбор данных в течение 10 циклов. В каждом цикле с двух микрофонов (*AUDIO\_IN* и *AUDIO\_IN\_2*) считываются значения с помощью функции *analogRead*, а затем эти данные передаются в массив *fht\_input[]* для дальнейшей обработки с использованием алгоритма преобразования Хартли. После обработки сигналов с помощью функций преобразования Хартли (например, *fht\_run()* и *fht\_mag\_log()*), для каждой частоты вычисляются амплитуды, которые суммируются и усредняются в массив *calibrationValues[]*.

Таким образом, для каждой частоты сохраняется усреднённое значение амплитуды, которое будет использоваться в дальнейшем для сравнения с реальными данными и определения уровня сигнала относительно шума.

После завершения процесса калибровки на консоль выводится усреднённый порог для каждой частоты, что позволяет пользователю видеть уровень шума и убедиться, что калибровка была выполнена правильно. Задержка в 3 секунды перед началом сбора данных гарантирует тишину в момент калибровки, исключая влияние посторонних шумов.

 2 Считывание звукового сигнала.

Звуковой сигнал поступает на аналоговые входы микроконтроллера, определенные макросами *#define AUDIO\_IN* и *#define AUDIO\_IN\_2*. В функции *analyzeAudio()* происходит считывание данных с этих входов в количестве, заданном параметром *FHT\_N*, равным 256.

Во время считывания сигнала с микрофонов выполняется проверка их работоспособности. Для каждого из микрофонов (*AUDIO\_IN* и *AUDIO\_IN\_2*) с помощью функции *checkAudio()* анализируются значения сигналов. Если сигнал с микрофона остается на уровне низких значений в течение нескольких выборок, то микрофон считается неисправным. Это определяется с помощью счетчиков *mic1\_stable\_low\_count* и *mic2\_stable\_low\_count*, которые увеличиваются, если значения с микрофонов находятся ниже определенного порога. Когда количество таких низких значений превышает определенный лимит (в данном случае 5), выводится сообщение о том, что микрофон неисправен.

В случае восстановления сигнала с микрофона (когда значения сигнала снова становятся нормальными), система автоматически восстанавливает работу с этим микрофоном, и выводится сообщение о восстановлении.

Для каждого из микрофонов, если сигнал находится в допустимых пределах, данные усредняются и записываются в массив *fht\_input[]*. Если же один из микрофонов не передает сигнал или его значение слишком низкое, данные с этого микрофона не используются, и усредняется только сигнал работающего микрофона. Если же отключены оба микрофона, то в массив *fht\_input[]* будут записан 0, то есть в итоге на матрицу ничего не выведется.

Таким образом, функция *checkAudio()* не только обрабатывает и усредняет данные, но и проверяет состояние микрофонов, обеспечивая надежность и корректность работы системы.

 3 Обработка данных.

*Применение оконной функции.* После заполнения массива значениями входного сигнала вызывается функция *fht\_window()*. Она применяет оконную функцию для сглаживания границ выборки, уменьшая искажения на концах и обеспечивая более точное частотное преобразование.

*Перестановка данных.* Для корректного выполнения спектрального преобразования вызывается функция *fht\_reorder()*, которая переупорядочивает элементы массива.

*Преобразование Хартли.* Основная операция преобразования спектра выполняется функцией *fht\_run()*. Она реализует дискретное преобразование Хартли, которое позволяет получить спектральные компоненты входного сигнала.

*Логарифмическая шкала.* После выполнения преобразования результат переводится в логарифмическую шкалу при помощи функции *fht\_mag\_log()*. Это делается для улучшения восприятия данных человеком, так как логарифмическая шкала лучше отображает широкий диапазон амплитуд.

*Фильтрация и усиление сигнала.* После выполнения преобразования каждый элемент массива *fht\_log\_out[]* фильтруется. Значения, меньше определенного порога, не учитываются в дальнейшем для вывода на матрицу.

*Группировка данных*. В процессе обработки данных для визуализации на матрице используется массив *fht\_log\_out*, который содержит амплитуды сигнала для разных частот. Для получения значения, необходимого для отображения одного столбика матрицы, из массива *fht\_log\_out* берутся 4 соседних значения. Эти значения соответствуют амплитудам звука на различных частотах, которые сгруппированы для одного столбика на матрице.

Каждое значение амплитуды для столбика получается путем вычисления средней величины из 4 последовательных частот. Этот процесс повторяется для каждого столбика на матрице, таким образом создавая визуализацию спектра звука, где каждый столбик отражает уровень громкости на определенных частотах, сгруппированных в блоки по 4 частоты.

3 Вывод данных.

*Отображение на матрице.* Прежде чем вывести новые данные, матрица очищается, что предотвращает наложение новых столбиков на старые. Затем программа рассчитывает высоту каждого столбика для всех сегментов матрицы. Полученные значения нормализуются, что обеспечивает гармоничный вид визуализации, а также преобразуются с учетом заданных параметров, таких как яркость и предельные значения высоты с использованием функций *map()* и *constrain()*.

*Отрисовка столбиков.* После вычисления высоты столбиков, программа проверяет, если значение высоты (*posLevel)* больше нуля, то на матрице отрисовывается линия с помощью функции *matrix.drawLine()*, начиная с координаты (*pos*, 7) и заканчивая точкой (*pos*, 7 - *posLevel)*. Таким образом, каждый столбик визуально представляет собой амплитуду определенной частоты в текущем спектре.

*Обновление матрицы.* После всех вычислений и отрисовки новых данных вызов *matrix.write()* отправляет данные на матрицу, обновляя её изображение. Это позволяет зрителю видеть обновленную визуализацию звукового спектра в реальном времени.

*Синхронизация данных.* Синхронизация обновлений отображения на матрице осуществляется за счет механизма контроля времени, основанного на функции *micros()*. Эта функция позволяет контролировать частоту обновлений, что помогает избежать перегрузки матрицы визуальной информацией. В коде реализована проверка времени, прошедшего с последнего обновления, с использованием условия *if (micros() % 2000 < 1000)*. Это означает, что данные обновляются каждые 2 миллисекунды, но в пределах каждой задержки только одно обновление происходит.

## 4.2 Описание функциональной схемы программы

Функциональная схема алгоритма, реализующего программное средство, представлена в графическом приложении В.

1 *Настройка интерфейсов и пинов для всех устройств*. На данном этапе происходит настройка микроконтроллера для взаимодействия с периферийными устройствами. Определяются пины для подключения микрофонов и матрицы, настраиваются интерфейсы SPI для управления светодиодной матрицей.

2 *Калибровка шумовых параметров частот*. Система выполняет калибровку, определяя базовый уровень шума для каждой частоты. Это необходимо для фильтрации фоновых шумов в дальнейшем. Результаты калибровки сохраняются в массиве, чтобы использовать их при обработке данных.

3 *Обработка входных аудиосигналов*. На этапе обработки входных аудиосигналов система анализирует данные с двух микрофонов. Для фильтрации низких значений, которые могут быть вызваны шумом или неисправностью, система отслеживает стабильность низких значений. Если сигнал на микрофоне *A0* или *A4* остается ниже определенного порога в течение пяти последовательных измерений, то система считает микрофон неисправным и выводит соответствующее уведомление. После восстановления нормальных уровней сигнала система выводит сообщение о восстановлении работы микрофона. В случае, если микрофон остается неисправным, его данные не участвуют в дальнейшей обработке, и система использует только данные с исправного микрофона. После проверки и фильтрации данных, система вычисляет среднее значение между показаниями исправных микрофонов и передает результаты для дальнейшего анализа.

4 *Частотный анализ звукового спектра*. Выполняется преобразование аудиосигнала с помощью библиотеки *FHT*. Алгоритм проходит через этапы применения оконной функции, перестановки данных и выполнения преобразования Хартли. Полученные спектральные данные переводятся в логарифмическую шкалу и фильтруются относительно шумовых порогов.

5 *Визуализация данных на матрице*. Спектральные данные группируются для формирования столбиков матрицы. Каждый столбик отображает уровень амплитуды для группы частот. Высота столбика рассчитывается на основе усредненных данных, нормализованных и отфильтрованных для комфортного восприятия.

Алгоритм циклически обновляет матрицу с заданной частотой, обеспечивая синхронную и плавную визуализацию звукового спектра в реальном времени.

Таким образом, были рассмотрены ключевые функции программного обеспечения, обеспечивающие эффективную работу системы.

# 5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

## 5.1 Cхема подключения

1 *Подключение опорного напряжения.* Для увеличения точности измерений подключается перемычка между выводом *3,3V* и контактом *AREF* на *Arduino Uno R3* через резистор 5,1 кОм. Это позволяет установить опорное напряжение в 3,3 В, что важно для более корректного считывания слабого сигнала с микрофонного модуля.

2 *Подключение двух микрофонных модулей.*

– *VDD* (питание): подключается к контакту *5V Arduino Uno R3*;

– *OUT* (сигнальный выход): подключается к аналоговому входу *A0*(*A4*);

– *GND* (земля): соединяется с *GND Arduino Uno R3*.

Подключение микрофонного модуля с опорным напряжением 3,3 В улучшает точность считывания звукового сигнала, что особенно важно при работе с небольшими уровнями напряжения от микрофона.

3 *Подключение светодиодной матрицы.*

– *DIN* (данные): подключается к цифровому выводу *D11 Arduino Uno R3*;

– *CS* (выбор чипа): подключается к цифровому выводу *D10*;

– *CLK* (тактовый сигнал): подключается к цифровому выводу;

– *GND* (земля): соединяется с *GND Arduino Uno R3* и микрофона;

– *VCC* (питание): подключается к контакту *5V Arduino Uno R3*.

Все компоненты объединяются в единую схему, что обеспечивает подачу и считывание сигнала с микрофона, его обработку на микроконтроллере, а затем отображение результатов на светодиодной матрице.

## 5.2 Общая структура программы

Исходный код описанного ниже программного обеспечения находится в приложении Б.

Программа состоит из нескольких основных блоков, каждый из которых выполняет определенные функции для реализации системы аудиовизуализации. Включает в себя инициализацию устройства, обработку входных аудиосигналов, фильтрацию и анализ данных, а также вывод информации на светодиодную матрицу.

Программа начинается с инициализации пинов *Arduino Uno R3*. Пины *A0* и *A4* используются для считывания входных сигналов от двух микрофонов, а цифровые пины 10, 11 и 13 управляют работой матрицы. Для повышения точности считывания аналогового сигнала подключение организовано с использованием внешнего опорного напряжения через делитель с *AREF*, соединенным с выходом 3,3 В.

После инициализации начинается этап калибровки. На этом этапе система измеряет фоновый шум с каждого микрофона, подключенного к входам A0 и A4. Эти значения используются для установления порога, который поможет фильтровать слабые сигналы и уменьшить влияние фонового шума на дальнейший анализ.

Далее, функция *analyzeAudio()* проводит анализ спектра аудиосигнала. Для этого используется преобразование Хартли (*FHT*). Программа сначала считывает данные с микрофонов, проверяет их на наличие неисправностей с помощью функции *checkAudio()* (если уровень сигнала на микрофоне слишком низкий, он исключается из анализа). Если оба микрофона исправны, их данные усредняются для улучшения точности. Затем применяется оконная функция, выполняется перестановка данных и расчет спектра с использованием логарифмической шкалы.

 В основной части цикла программы *loop()* происходит расчет и отображение амплитуд частот на светодиодной матрице. Для каждого столбца матрицы, соответствующего определенному диапазону частот, из массива, содержащего амплитудные значения частот, выбираются четыре подряд идущих значения. Эти значения представляют собой амплитуды для узкого диапазона частот. Далее для каждого столбца рассчитывается их среднее значение, которое затем используется для определения высоты столбика на матрице. Это значение подвергается дополнительной нормализации: оно масштабируется в диапазон от 0 до 7 с помощью функции *map*, что позволяет точно отобразить амплитуду в пределах ограничений матрицы. Если результат больше нуля, то рисуется вертикальная линия, высота которой соответствует среднему уровню амплитуды выбранных частот.

Важным аспектом работы программы является периодичность выполнения цикла, которая контролируется условием *if (micros() % 2000 < 1000)*. Это условие позволяет программе выполнять анализ аудиосигнала и обновление матрицы с интервалом в 2 миллисекунды. Это обеспечивает выполнение цикла с заданной частотой, гарантируя, что обработка данных и вывод на матрицу происходят с определенной регулярностью и в заданные интервалы времени.

Таким образом, каждый столбец матрицы визуализирует информацию о звуковом сигнале, соответствующую определенному диапазону частот, с учетом фильтрации и усреднения значений для более стабильного отображения.

## 5.3 Описание блок-схемы алгоритма программы

Далее рассмотрена блок-схема алгоритма программного обеспечения (приложение Г).

*Инициализация системы*. На данном этапе производится настройка микроконтроллера, установка всех необходимых параметров работы и подготовка элементов для последующего взаимодействия с системой. В первую очередь, определяются параметры работы для преобразования Хартли (*FHT*), устанавливая количество выборок в 256. Это количество выборок напрямую влияет на точность и разрешение при анализе сигналов в частотной области, позволяя более детально и точно интерпретировать полученные данные.

Одновременно с этим настраивается логарифмическая шкала для обработки и отображения выходных данных, с установкой значения *LOG\_OUT 1*. Это необходимо для того, чтобы представить амплитуды сигналов в удобном для восприятия виде, сокращая диапазон значений и акцентируя внимание на важных сигналах.

В качестве источника питания используется внешнее напряжение 3,3 В, обеспечивающее стабильную работу системы и всех её компонентов. Аналоговые пины *A0* и *A4* назначаются для подключения первого и второго микрофонов соответственно. Эти микрофоны будут считывать аудио сигналы, и на этом этапе также задаются пороговые значения для определения их работоспособности.

Особое внимание уделяется настройке порога для определения неисправных микрофонов. В данном случае, если микрофон не подаёт сигнал, или сигнал слишком слабый (ниже заданного порога), система будет считать его неисправным после пяти последовательных низких значений. Это позволяет системе точно распознавать неисправные микрофоны и избежать ошибок в обработке сигнала.

Кроме того, создаётся массив для хранения калибровочных значений, которые будут использоваться для фильтрации шумов и дальнейшей корректной работы системы. Эти значения играют ключевую роль в настройке системы на реальное окружение и оптимизацию анализа.

Для визуализации данных используется матрица *MAX7219*, которая подключается к микроконтроллеру через пин *CS (D10)* и управляется четырьмя модулями.

Наконец, открывается последовательный порт для вывода диагностических данных и отладочной информации. Это необходимо для мониторинга работы системы в реальном времени, что даёт возможность оперативно отслеживать её состояние.

*Калибровка системы.* Этап калибровки начинается с того, что система выводит сообщение о начале процесса, предупреждая пользователя, чтобы сохранялась тишина. Затем создается массив для хранения калибровочных значений, в который будут записываться данные о шумовых порогах для каждой частоты. В цикле калибровки выполняется несколько измерений, в ходе которых анализируется аудио сигнал с помощью функции *analyzeAudio()*. В каждой итерации цикла данные о сигналах, полученные из массива *fht\_log\_out*, аккумулируются, чтобы получить более точные значения. Это повторяется несколько раз (по умолчанию 10 циклов), и после завершения циклов данные усредняются. Каждый элемент массива *calibrationValues* делится на количество циклов, чтобы исключить случайные отклонения и получить точные пороговые значения для фильтрации шума. Эти значения выводятся в монитор порта для ознакомления пользователя с результатами калибровки. После завершения всех циклов и подсчёта средних значений выводится сообщение о завершении калибровки, и система готова к дальнейшей работе.

*Периодичность выполнения.* Основной цикл выполняется каждые 2 миллисекунды. Условие *micros() % 2000 < 1000* обеспечивает выполнение цикла дважды в каждый двухмиллисекундный интервал. Это позволяет контролировать периодичность обработки данных с точностью до 1 миллисекунды.

*Ввод данных.* Если интервал прошел, программа запускает процесс сбора новых данных. Производится 256 выборок аудиосигнала с каждого из аналоговых портов, к которым подключены микрофоны. Это происходит в функции *analyzeAudio()*, после каждой выборки данные передаются в функцию *checkAudio()*.

*Проверка состояния микрофонов и расчет среднего значения*. На этом этапе происходит проверка состояния микрофонов. Если уровень сигнала с микрофона падает ниже установленного порога, то увеличивается счетчик стабильных низких значений. Если этот счетчик достигает 5, то микрофон считается неисправным, и выводится сообщение о его отключении. Когда уровень сигнала восстанавливается, счетчик сбрасывается, и микрофон снова считается работоспособным. Далее производится расчет среднего значения сигналов с двух микрофонов. Если оба сигнала достаточно высокие, то их усредняют, иначе используется значение, которое превышает порог. В случае, если оба сигнала слишком низкие, возвращается значение 0.

*Преобразование входного сигнала и получение спектра.* После того как данные с микрофонов были считаны и обработаны в функции *checkAudio()*, они передаются в массив *fht\_input*, который используется для дальнейшего преобразования Хартли. Для подготовки данных к быстрому преобразованию Хартли (*FHT*) используется оконная функция, сглаживающая края массива и обеспечивающая более точный спектральный анализ. Затем проводится перестановка данных, после чего выполняется само преобразование Хартли для вычисления спектра частот. Результаты преобразования преобразуются в логарифмическую шкалу, что позволяет получить удобный для дальнейшей обработки массив спектральных значений.

*Анализ спектра (128 полос).* На этом этапе начинается анализ полученных спектральных значений. Просматривается 128 полос спектра, полученных в результате применения преобразования Хартли к считанным аудиосигналам.

*Вычисление средней амплитуды для каждой группы часто.* Для каждого столбца матрицы берется 4 соседние частоты, и их амплитуды из массива *fht\_log\_out* сравниваются с пороговыми значениями из массива *calibrationValues*. Если амплитуда сигнала выше порога, она добавляется в общий накопленный сигнал. Затем из этой суммы вычисляется среднее значение амплитуды для 4 частот.

*Визуализация амплитудных значений на светодиодной матрице.* Значение амплитуды преобразуется в диапазон от 0 до 7 с использованием функции отображения. Дополнительно, значение ограничивается, чтобы гарантировать, что оно остается в заданном диапазоне.

Если итоговое значение амплитуды больше нуля, то рисуется вертикальная линия на матрице. Это визуализирует амплитуду звука в виде столбиков, где высота столбика пропорциональна уровню амплитуды.

***Изменение смещения для следующего столбца***. После обработки каждого столбца смещение на 4 частоты увеличивается (переменная *posOffset*), чтобы для следующего столбца использовались другие группы частот.

*Финализация.* После завершения обработки данных и их отображения на матрице цикл повторяется с заданной периодичностью, обновляя вывод и анализируя новые входные данные.

## 5.4 Тестирование и обработка ошибок

Результаты калибровки и сообщения о процессе работы системы можно наблюдать в мониторе порта, скриншоты которых приведены в приложении Д.

При разработке визуализатора на базе микроконтроллера *Arduino Uno R3* была реализована система тестирования и обработки ошибок для обеспечения стабильной работы устройства. Данная система направлена на выявление неисправностей и предотвращение сбоев при анализе звуковых сигналов и их визуализации на матричном дисплее.

1 Калибровка системы.

В начале работы устройства выполняется процедура калибровки. На этапе калибровки система настраивается для работы в условиях фоновых и внешних шумов, определяя порог шума для каждой из 128 частотных полос спектра отдельно:

– сбор выборки сигналов с микрофонов (*AUDIO\_IN* и *AUDIO\_IN\_2*);

– определение среднего уровня шума для каждой из 128 полос спектра;

– установка пороговых знаечний фильтрации для дальнейшего отсечения аудиосигналов ниже заданного порога.

По завершении калибровки на дисплей выводится информация о пороговых значениях для каждой частоты. Эти значения помогают системе точно настроиться на реальное рабочее окружение и эффективно фильтровать шум.

2 Проверка работоспособности микрофонов.

Для каждого микрофона выполняется мониторинг стабильности сигнала. Если уровень сигнала на протяжении заданного количества итераций (*STABLE\_CHECK\_COUNT*) остаётся низким, устройство определяет, что микрофон может быть отключён или неисправен:

– при обнаружении неисправности система выводит сообщение об ошибке;

– если работоспособность микрофона восстанавливается, устройство уведомляет об это.

Если система обнаруживает, что сигнал с одного из микрофонов отсутствует или находится на критически низком уровне (ниже минимального порога на протяжении заданного времени), она автоматически переключается на использование данных только со второго микрофона. В этом случае процедура вычисления среднего значения сигналов отключается, чтобы избежать искажения данных из-за неисправного источника. Вместо этого система использует данные исключительно от исправного микрофона для анализа и визуализации. Такой подход обеспечивает стабильную работу устройства, даже если один из микрофонов вышел из строя или временно не функционирует. При восстановлении сигнала с неисправного микрофона система возвращается к нормальному режиму работы, учитывая данные от обоих источников. Это решение позволяет повысить устойчивость устройства к аппаратным сбоям и гарантировать непрерывную работу визуализатора.

Кроме того, если оба микрофона оказываются нерабочими (то есть оба сигнала находятся на критически низком уровне), система устанавливает среднее значение сигнала в 0, и визуализация не происходит.

Реализованная система тестирования и обработки ошибок значительно повысила надёжность работы визуализатора. Она позволяет своевременно обнаруживать неисправности, минимизировать их влияние на работу устройства и обеспечить стабильную работу.

# ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом, посвященным разработке визуализатора музыки на основе *FHT* и *Arduino*, была достигнута основная цель — создание системы, которая способна анализировать аудиосигнал и отображать результаты на светодиодной матрице в виде динамических графиков частотного спектра.

Используя преобразование Хартли (*FHT*), удалось эффективно выделить ключевые частотные компоненты звукового сигнала, что позволило создать реалистичную визуализацию, отражающую динамику музыки. Важной частью проекта стало использование библиотеки *Max72xxPanel* для управления светодиодными матрицами, что обеспечило гибкость и точность в отображении результатов. Также в процессе разработки были учтены параметры, такие как усиление входного сигнала и фильтрация шума, для получения стабильной и качественной работы устройства.

Особое внимание было уделено оптимизации работы системы с использованием двух микрофонов для более точного и реалистичного восприятия окружающей аудиосреды. Также была добавлена возможность калибровки системы, что позволяет адаптировать устройство под различные условия использования и улучшить его производительность.

Результатом работы стал компактный и эффективный визуализатор музыки, который может быть использован не только в образовательных целях, но и в различных развлекательных или демонстрационных приложениях. В дальнейшем проект можно расширить, добавив больше настроек для пользователя, улучшив интерфейс и повысив точность анализа спектра, а также интегрируя поддержку более сложных алгоритмов обработки сигнала.

Использование платформы *Arduino* и алгоритмов преобразования Хартли является мощным инструментом для создания интерактивных систем, позволяющих визуализировать сложные данные в реальном времени. Такой подход может быть использован не только для музыкальных приложений, но и для более широких сфер, таких как обработка и визуализация других типов сигналов, например, для научных или инженерных исследований.

Данный проект продемонстрировал возможности интеграции простых аппаратных решений с современными алгоритмами обработки сигнала для создания доступных и интерактивных систем, которые могут быть использованы как в образовательных целях, так и в реальных практических приложениях.

# СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Arduino Uno [Электронный ресурс]. – Режим доступа: https://arduino.ru/Hardware/ArduinoBoardUno. – Дата доступа: 09.09.2024

[2] MAX9814  Microphone  Amplifier  with  AGC  and   Low-Noise Microphone Bias [Электронный ресурс] : Datasheet / Maxim Integrated. – Режим доступа : MAX9814.pdf.

[3] MAX7219 8x8 Dot Matrix Display Module [Электронный ресурс] : Datasheet / Handson Technology. – Режим доступа : MAX7219-8x8.pdf.

[4] Барретт С. Ф. Arduino: искусственный интеллект и машинное обучение / пер. с англ. Ю. В. Ревича. – М.: ДМК Пресс, 2024. – 242 с.

[5] Блум Дж. Изучаем Arduino: инструменты и методы технического волшебства: Пер. с англ. - СПб.: БХВ-Петербурr, 2015. - 336 с.: ил.

[6] Петин В. А. Проекты с использованием Arduino. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2015. – 464 с.: ил. – (Электроника)

[7] AnalogRead Программирование Ардуино [Электронный ресурс]. – Режим доступа: <https://doc.arduino.ua/ru/prog/AnalogRead>. – Дата доступа: 09.09.2024

[8] Getting Started with Arduino IDE 2 [Электронный ресурс]. – Режим доступа: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/. – Дата доступа: 09.09.2024

[9] Software [Электронный ресурс]. – Режим доступа: <https://www.arduino.cc/en/software>. – Дата доступа: 09.09.2024

[10] Scherz, P. Audio Electronics / P. Scherz // Practical  Electronics  for Inventors. – 1989. – №4. – С. 21–25.

[11] Брейсуэлл, Р. Преобразование Хартли. Теория и приложения / пер. с англ. Ю. В. Ревича. – М.: Мир, 1990. – 175 с.

[12] Программирование  Ардуино [Электронный ресурс]. – Режим доступа: [https://arduino.ru/Reference?ysclid=m3uo8nh23u496956708](https://arduino.ru/Reference?ysclid=m3uo8nh23u496956708" \t "_blank).

[13] Монк, С. Программируем Arduino. Профессиональная работа  со скетчами / С. Монк – СПб.: Питер, 2017. – 251 с.

[14] Arduino Docs [Электронный ресурс]. – Режим доступа: https://docs.arduino.cc/. – Дата доступа: 30.09.2024

[15] Reliable Frequency Measurment Using Autocorrelation - Projects Discussion and Showcase / Science and Measurement - Arduino Forum [Электронный ресурс]. – Режим доступа: <https://forum.arduino.cc/t/reliable-frequency-measurment-using-autocorrelation/190147>. – Дата доступа: 30.09.2024

[16] Соммер У. С61 Программирование микроконтроллерных плат Arduino/Freeduino. – СПб.: БХВПетербург, 2012. – 256 с.: ил. — (Электроника)

[16] Arduino & MAX9814 – Getting Started! [Электронный ресурс]. – Режим доступа: https://www.electroschematics.com/arduino-max9814-getting-started/. – Дата доступа: 01.11.2024

[17] Guide to 3V3 and 5V Power Supplies Differences [Электронный ресурс]. – Режим доступа: <https://docs.arduino.cc/learn/microcontrollers/5v-3v3/>. – Дата доступа: 01.11.2024

[18] Ревич Ю. В. Занимательная электроника. - 5-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2018. -672 с.: ил. - (Электроника)

[19] About the AREF pin – Arduino Help Center [Электронный ресурс]. – Режим доступа: <https://support.arduino.cc/hc/en-us/articles/360018922239-About-the-AREF-pin>. – Дата доступа: 01.11.2024

[20] Serial.println() [Электронный ресурс]. – Режим доступа: https://arduino.ru/Reference/Serial/Println. – Дата доступа: 01.11.2024

# ПРИЛОЖЕНИЕ А (обязательное) Справка о проверке на заимствования

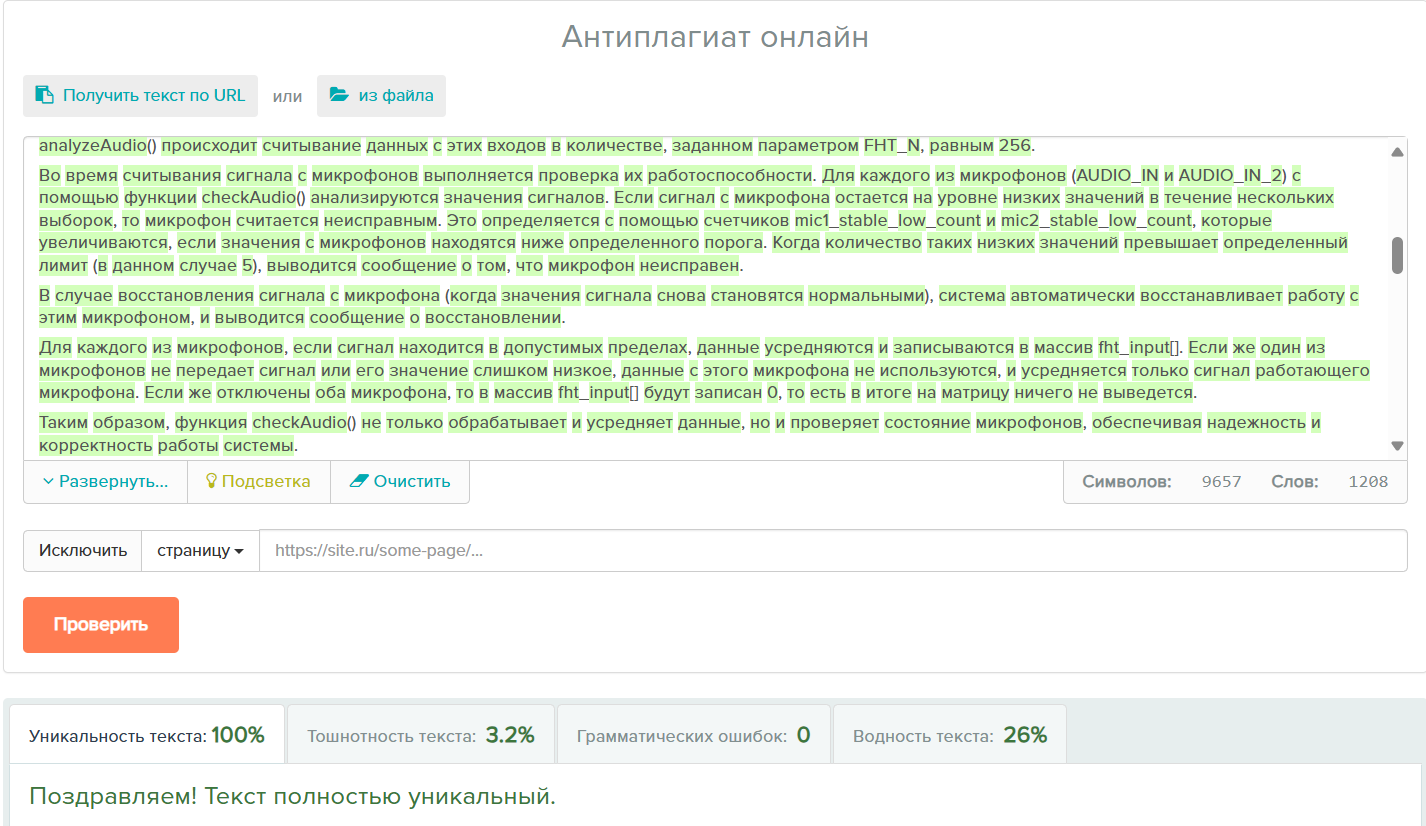


Рисунок 1 – Справка о проверке на заимствования

# ПРИЛОЖЕНИЕ Б (обязательное) Листинг программного кода

#define FHT\_N 256

#define LOG\_OUT 1

#define AUDIO\_IN 0

#define AUDIO\_IN\_2 4

#define STABLE\_CHECK\_COUNT 5

#define THRESHOLD 65

#include <FHT.h>

#include <SPI.h>

#include <Adafruit\_GFX.h>

#include <Max72xxPanel.h>

Max72xxPanel matrix = Max72xxPanel(10, 4, 1);

// Массив для хранения калибровочных значений

unsigned int calibrationValues[FHT\_N / 2];

byte maxLevel = 80;

void setup() {

  analogReference(EXTERNAL);

  Serial.begin(9600);

  Serial.println("Привет, user!");

  matrix.setIntensity(1);

  for (byte i = 0; i < 4; i++) {

    matrix.setRotation(i, 1);

  }

  matrix.fillScreen(LOW);

  matrix.write();

  delay(50);

  calibrateNoise();

}

void loop() {

  if (micros() % 2000 < 1000) {

    analyzeAudio();

    matrix.fillScreen(LOW);

    delay(2);

    byte posOffset = 4;

    for (byte pos = 0; pos < 32; pos++) {

      int posLevel = 0;

      for (byte i = posOffset - 4; i < posOffset; i++) {

        int noiseThreshold = calibrationValues[i];

        int signalLevel = fht\_log\_out[i];

        if (!(signalLevel <= noiseThreshold)) {

          posLevel += signalLevel;

        }

      }

      posLevel /= 4;

      if (posLevel > maxLevel) maxLevel = posLevel;

      posLevel = map(posLevel, 0, maxLevel, 0, 7);

      posLevel = constrain(posLevel, 0, 7);

      if (posLevel > 0) matrix.drawLine(pos, 7, pos, 7 - posLevel, HIGH);

      posOffset += 4;

    }

    matrix.write();

  }

}

void calibrateNoise() {

  Serial.println("Калибровка: пожалуйста, сохраняйте тишину...");

  delay(3000);

  const int calibrationCycles = 10;

  for (int i = 0; i < FHT\_N / 2; i++) {

    calibrationValues[i] = 0;

  }

  for (int cycle = 0; cycle < calibrationCycles; cycle++) {

    analyzeAudio();

    for (int i = 0; i < FHT\_N / 2; i++) {

      calibrationValues[i] += fht\_log\_out[i];

    }

  }

  for (int i = 0; i < FHT\_N / 2; i++) {

    calibrationValues[i] /= calibrationCycles;

    calibrationValues[i] += 20;

    Serial.print("Частота ");

    Serial.print(i);

    Serial.print(": усреднённый порог ");

    Serial.println(calibrationValues[i]);

    delay(200);

  }

  Serial.println("Калибровка завершена.");

}

void analyzeAudio() {

  for (int i = 0; i < FHT\_N; i++) {

    int sample1 = analogRead(AUDIO\_IN);

    int sample2 = analogRead(AUDIO\_IN\_2);

    int averageSample = checkAudio(sample1, sample2);

fht\_input[i] = averageSample;

  }

  fht\_window();

  fht\_reorder();

  fht\_run();

  fht\_mag\_log();

}

int mic1\_stable\_low\_count = 0;

int mic2\_stable\_low\_count = 0;

bool mic1\_ok = true;

bool mic2\_ok = true;

int checkAudio(int sample1, int sample2){

  if (sample1 <= THRESHOLD) {

    mic1\_stable\_low\_count++;

  }

  else {

    mic1\_stable\_low\_count = 0;

  }

  if (mic1\_stable\_low\_count >= CHECK\_COUNT && mic1\_ok) {

    Serial.println("Микрофон на A0 отключён или не работает.");

    mic1\_ok = false;

  }

  else if (mic1\_stable\_low\_count == 0 && !mic1\_ok) {

    Serial.println("Микрофон на A0 восстановил работу.");

    mic1\_ok = true;

  }

  if (sample2 <= THRESHOLD) {

    mic2\_stable\_low\_count++;

  }

  else {

    mic2\_stable\_low\_count = 0;

  }

  if (mic2\_stable\_low\_count >= CHECK\_COUNT && mic2\_ok) {

    Serial.println("Микрофон на A4 отключён или не работает.");

    mic2\_ok = false;

  }

  else if (mic2\_stable\_low\_count == 0 && !mic2\_ok) {

    Serial.println("Микрофон на A4 восстановил работу.");

    mic2\_ok = true;

  }

  int averageSample;

  if (sample1 > THRESHOLD && sample2 > THRESHOLD) {

    averageSample = (sample1 + sample2) / 2;

  }

  else if (sample1 > THRESHOLD) {

    averageSample = sample1;

  }

  else if (sample2 > THRESHOLD) {

    averageSample = sample2;

  }

  else {

    averageSample = 0;

  }

  return averageSample;

}

# ПРИЛОЖЕНИЕ В

# (обязательное)

# Функциональная схема алгоритма, реализующего программное средство

# ПРИЛОЖЕНИЕ Г (обязательное) Блок схема алгоритма, реализующего программное средство

# ПРИЛОЖЕНИЕ Д (обязательное) Графический интерфейс пользователя

# ПРИЛОЖЕНИЕ Е

# (обязательное) Ведомость документов курсового проекта