Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Структурная и функциональная организация ЭВМ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

Музыкальный пад

БГУИР КП 1-40 02 01 211 ПЗ

Выполнил:
студент группы 050502 Крачковский А.В

Проверил:
инженер-электроник кафеды ЭВМ Дмитриев А.С

Минск 2023

Содержание

Введение

Темой данного курсового проекта является разработка музыкального пада (пад — дощечка, платформа).

Данное устройство предназначено для последовательного вывода звуков, чаще всего коротких и ритмичных (называемых «сэмплами»), для создания музыки.

Такое устройство имеет достаточно широкое применение в мире: в общественном транспорте мы часто слышим как воспроизводят заранее заготовленные фразы, будь то следующая остановка или какая-либо пропаганда. В мире музыки такие устройства стали популярны после появления жанра даб-степ (электронная музыка), где композиторы могли менять (искривлять) звучание мелодий. В обоих случаях главным фактором такого устройства была чистота звучания, поэтому в данном курсовом проекте будет идти большой упор именно на качество звуков.

Разработка курсового проекта будет происходить поэтапно. Вначале нужно определиться с элементами устройства, учитывая их стоимость, размеры и работоспособность. Потом нужно спроектировать устройство и разработать для него программное обеспечение. Завершающий этап проекта – тестирование устройства на правильность работы.

1 Обзор литературы по теме проекта

1.1 Обзор предметной области

Есть ряд функций которыми должно выполнять музыкальный пад, а именно: проигрывание звуков, их хранение, выбор звука который должен проиграться в данный момент и светоиндикация. Для решения этих задач в состав устройство должно входить:

- Микроконтроллер
- Аудио адаптер
- Орган управления
- Читатель карт памяти
- Цифро-аналоговый преобразователь
- Светодиодная лента

1.2 Микроконтроллеры

Основным элементов устройства является контроллер. Данный компонент управляет работой всех элементов устройства, и отвечает за их взаимодействие. На рынке представлены различные модели микроконтроллеров, которые различаются ценой, размерами, выполняемыми задачами и многим другим.

Сравнение наиболее подходящий микроконтроллеров можно увидеть в таблице 1.1

Таблица 1.1 – Сравнение параметров микроконтроллеров

Параметры	Arduino Uno	ESP-32	Raspberry PI 2
Микроконтроллер	ATmega328	CP2102	RP2040
Входное напряжение (рабочее), В	5-20	2.3-3.6	1.8-5.5
Выходное напряжение, В	3.3, 5	3.3	3.3, 5
Цифровые входы\выходы	14 шт	15 шт	15 шт
Аналоговые входы\выходы	6 шт	4 шт	6 шт
Flash-память, кб	32	64	128
ОЗУ, кб	2	8	129
Тактовая частота, МГц	16	80	72
Разрядность, бит	8	12	12
Интерфейс	I2C	I2C,I2S	I2C
Размер, мм	69 x 53	50 x 30	101.6 x 86

Для получения более подробной информации о рассмотренных микроконтроллерах использовались источники [1, 2, 3].

1.3 Аудио адаптер

Аудио адаптер используется для того чтобы соединить данное устройство с внешними динамики для вывода звуков, сравнение таких адаптеров можно увидеть в таблице 1.2.

Таблица 1.2 – Сравнение параметров аудио адаптеров

Параметры	NYS 215	ST214-C	NYS 232
Количество контактов	3	5	3
диаметр	6.5	3.5	6.35
Рабочее напряжение, В	5	5	5
моно\стерео	стерео	стерео	стерео

1.4 Орган управления

Чтобы выбрать какой именно звук нужно проиграть требуется каким-либо способом дать эту команду, для этого будет использоваться матричная клавиатура, но подходящей клавиатуры, по размеру, найти не удалось, поэтому будем делать её самостоятельно, для этого понадобиться найти тактовые клавиши. Сравнение подходящих клавиш можно увидеть в таблице 1.3.

Таблица 1.3 – Сравнительная характеристика клавиш

Параметры	TLS12-ASP	TC-12ET	IT-1102K
Сопротивление контакта, Ом	<0,1	<0,1	<0.1
Число циклов	100000	100000	100000
Входное напряжение, В	12	12	12
Рабочий ток, А	0.05	0.05	0.05

1.5 Светодиодные ленты

Светодиодная лента будем отличным вариантом для индикации взаимодействия пользователя с устройством, но обычная светодиодная лента не подойдёт по причине того, что тогда не будет возможности включить только один конкретный светодиод. Из этого следуют что в данном случае подойдёт адресная лента с возможностью конфигурации определённых светодиодов. Сравнение подходящих светодиодных лент для данного курсового проекта можно увидеть в таблице 1.4.

Таблица 1.4 – Сравнительная характеристика светодиодных лент

Параметры	SMD 550 LED, WS2812B	LP SMD 3528, Standart	Class Premium 2835
Потребляемый ток, mA	1920	800	1170
Цвет свечения	RGB	Тёплый белый	Белый
Угол излучения	120	140	120
Сила света, мкд	200	250	300
Рабочее напряжение, В	5	12	24
Кол-во светодиодов	150	100	98
Цвет линзы	прозрачный	тёплый белый	тёплый белый

1.6 Считыватель карт памяти

Считыватель карт памяти, также известное как карт-ридер, является устройством, предназначенным для считывания информации, хранящейся на картах памяти. В нашем случае, звуковые файлы будут храниться на картах памяти, и поэтому для доступа к этим файлам потребуется установка и использование устройства для чтения карт.

Считыватель карт памяти должен будет работать с microSD картами размером от 2 до 32 гигабайт. Установка считывателя карт памяти позволит легко переносить звуковые файлы с карты памяти на компьютер или другие устройства, что обеспечит удобство доступа и управления звуковыми файлами.

В таблице 1.5 можно увидеть сравнения считывателей карт памяти.

Таблица 1.5 – Сравнительная характеристика считывателей карт памяти

Параметр	SDCMF-10915WOT1	WR-CRD SD4.0 SMT PushPush Card Detect	SD MEM CD CONN R/A SMT PUSHPUSH
Входное напряжение. В	30	50	50
Диапазон рабочих температур	-25..+85	-25..+85	-25..+85
Поддержка карт памяти, Гб	<=2	<=8	<=8
Кол-во контактов	6	9	9

1.6 Цифро-аналоговый преобразователь

Цифро-аналоговый преобразователь (ЦАП) - это устройство, которое преобразует цифровой аудиосигнал, представленный в виде цифрового кода, в аналоговый формат. Это необходимо для того, чтобы звук мог быть воспроизведен на аналоговых устройствах вывода, таких как акустические системы, наушники или аудиоусилители.

В нашем случае, ЦАП будет использоваться для преобразования поступающего цифрового кода звука в аналоговый формат, чтобы он мог быть передан через устройство вывода и воспроизведен. Это позволяет преобразовывать цифровые звуковые файлы или потоки в аналоговый звук, который может быть услышан человеческим ухом.

Передача данных от ЦАПа будет осуществляться по I2S-каналу. I2S (Inter-Integrated Circuit Sound) - это интерфейсная шина, специально разработанная для передачи аудиоданных между устройствами. Он предоставляет отдельные линии для синхронизации (лайн-фрейм), передачи данных (лайн-дата) и тактового сигнала (лайн-битовый клок). Использование I2S-канала обеспечивает стабильную и надежную передачу аудиоданных без помех, что является важным для высококачественного воспроизведения звука.

Сравнительная характеристика ЦАП-ов показана в таблице 1.6

Таблица 1.6 – Сравнение цифро-аналоговых преобразователей

Параметр	PCM5102A	AD1933WBSTZ	AD5322BRMZ
Цифровой интерфейс	I2S	I2S, SPI	SPI, DPS
Напряжение питания, В	3,3 или 5	3,3	5,5
Разрядность	32	24	12

2 Разработка структуры устройства

2.1 Постановка задачи

Структура устройства всегда разрабатывается на основе функций, которые оно должно выполнять. Это очень важный этап проектирования, так как принципиальная и функциональная электрические схемы будут строится на базе структурной.

Единственная цель устройства – это вывод звука, по заранее записанным «сэмплам» на карту памяти.

2.2 Определение компонентов структурной схемы

- Микроконтроллер – ключевой компонент в схеме, выполняет функции обработки нажатий на клавиатуре, определение звукового файла и отправка его на цифро-аналоговый преобразователь.
- Модуль питания – источник электроэнергии, который подаёт питание на устройство
- Считыватель карт памяти – устройство в которое вставляется карта памяти и выступающее как хранилище звуков.
- Цифро-аналоговый преобразователь – устройство преобразующее цифровой сигнал в аналоговый.
- Аудио адаптер – гнездо позволяющее подсоединить динамики.
- Орган управления – матричная клавиатура из тактовых клавиш.
- Светодиодная лента – служит для индикации готовности устройства и обозначения нажатой клавиши.

2.3 Взаимодействие устройств

Для реализации всей функциональности схемы и обеспечения достаточного количества входов/выходов было решено добавить дополнительный микроконтроллер, который будет выполнять определенные задачи и работать в многопроцессорном режиме. В результате возникает следующая последовательность взаимодействия:

1. После подачи питания микроконтроллер 1 подключается к шине I2C в качестве ведомого устройства и ожидает подтверждения настройки от микроконтроллера 2. Пока происходит ожидание, микроконтроллер 1 воспроизводит заранее настроенный шаблон световой индикации на светодиодной ленте.

2. В это время микроконтроллер 2 настраивает интерфейс I2S, подключается к шине I2C в качестве ведущего устройства и ожидает подключения карты памяти к считывателю карты памяти.

3. После подключения карты памяти микроконтроллер 2 отправляет сигнал о готовности к получению инструкций и переключает режим чтения

I2C с ведущего на ведомый. Затем микроконтроллер 1 оповещает пользователя, подавая тройной сигнал на все светодиоды ленты.

4. При нажатии клавиши на клавиатуре микроконтроллер 1 определяет нажатую клавишу, подает сигнал на соответствующий светодиод и передает информацию о нажатой клавише через шину I2C. Микроконтроллер 2, получив информацию о нажатой клавише, считывает соответствующий файл с помощью считывателя карты памяти и отправляет его на ЦАП.

5. Преобразованный аналоговый сигнал поступает на аудиоадаптер, откуда он передается на внешний динамик для воспроизведения звука.

Таким образом, оба микроконтроллера работают в сотрудничестве, выполняя различные задачи: микроконтроллер 1 управляет светодиодной индикацией и определением нажатых клавиш, а микроконтроллер 2 управляет процессом чтения файлов с карты памяти и передачей данных на ЦАП для воспроизведения звука. Такое разделение функций между двумя микроконтроллерами позволяет распределить нагрузку и обеспечить более эффективное выполнение всей схемы.

3 Обоснование выбранных узлов, элементов функциональной схемы

3.1 Обоснование выбора микроконтроллеров

Микроконтроллер ESP-32 был выбран как основной из-за ряда своих возможностей, а именно: поддержка I2S интерфейса передачи данных (которого нету в Arduino UNO) и наличие аналоговых входов\выходов (которых нету в Raspberry PI 2), но в ESP-32 не достаточно выводов для подсоединения всех элементов схемы. Хорошей альтернативой мог бы стать микроконтроллер Arduin UNO, но в данном микроконтроллере отсутствует I2S. Поэтому было принято решение использовать связку микроконтроллеров ESP-32 и Arduino UNO, наладив между ними общение по I2C интерфейсу. Сравнение микроконтроллеров можно увидеть в таблице 1.1. Внешний вид и обозначение выводов можно увидеть на рисунке 3.1.

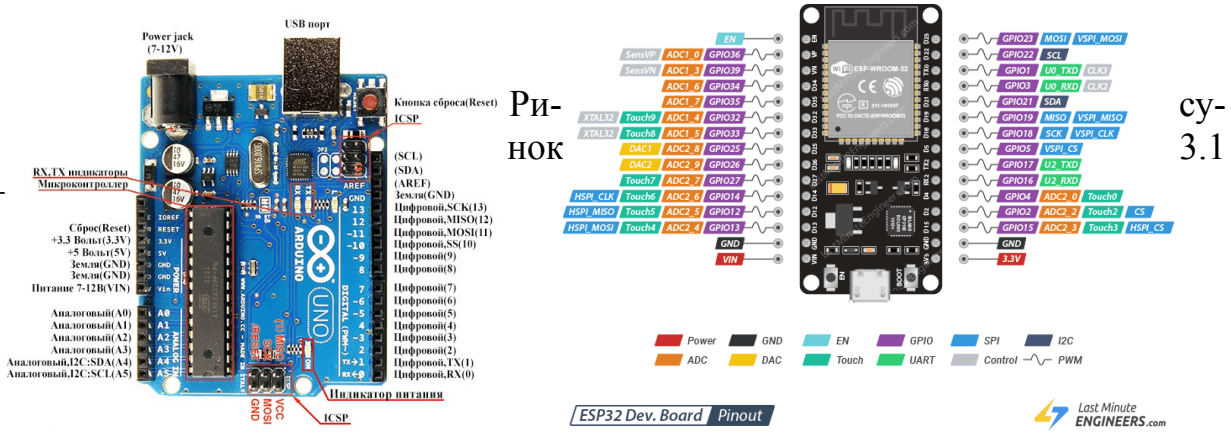

Рисунок 3.1 – Выбранные микроконтроллеры

3.2 Обоснование выбора тактовых клавиш

Так как особой разницы между клавишами нет, был выбран самый доступный вариант, а именно клавиши из старой клавиатуры. Сравнение тактовых клавиш можно увидеть в таблице 1.3, саму клавишу можно увидеть на рисунке 3.2.

Рисунок 3.2 – Gateron Switch

3.3 Обоснование выбора светодиодной ленты

Поскольку светодиоды нужны просто для индикации нажатия клавиши, то был выбран самый просто вариант адресной светодиодной ленты без сильного свечения (SMD 550 LED). В данном курсовом проекте рассматривались конкретно адресные светодиодные ленты так как при нажатии клавиши необходимо подавать сигнал на конкретный светодиод. Помимо этого плюсом выбранной ленты стала хорошая доступность. Сравнение светодиодных лент можно увидеть в таблице 1.4, а саму ленту на рисунке 3.3.

Рисунок 3.3 – Светодиодная лента SMD 550 LED

3.4 Обоснование выбора аудио адаптера

Идея установить именно аудио адаптер, а не встроенные динамики обусловлена тем, что пользователю не будет привязан к определённым динамикам и при желании сможет даже подсоединить наушники чтобы не мешать окружающим. Поэтому был выбран ST214-C аудио адаптер, который больше всего подходил к уже купленным наушникам из-за своего диаметра гнезда. Сравнение аудио адаптеров можно увидеть в таблице 1.2, а сам аудио адаптер на рисунке 3.4.

Рисунок 3.4 – аудио адаптер ST214-C

3.5 Обоснование выбора цифро-аналогового преобразователя

Главное условие для цифро-аналогового преобразователя была работа именно на I2S интерфейсе, который был создан специально для передачи аудио. Хоть с помощью интерфейса I2C тоже можно передавать звуки и в каком-то смысле было бы даже проще использовать конкретно этот интерфейс, но по нему будут происходить очень сильные искривления звука.

Поэтому был выбран PCM5102A который поддерживает I2S (об этом интерфейсе речь пойдёт в разделе 4) интерфейс передачи данных и имеет разрядность более подходящую для устройства.

Сравнение цифро-аналоговых преобразователей можно увидеть в таблице 1.6, а его выводы на рисунке 3.5.

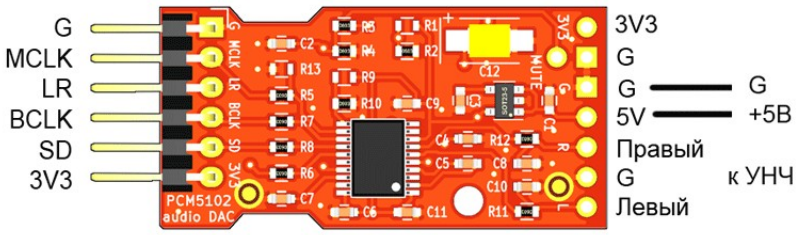

Рисунок 3.5 – цифро-аналоговый преобразователь PCM5102A

3.6 Обоснование выбора читателя карт памяти

Был выбран SDCMF-10915WOT1 потому что он оказался самым доступным по цене. Сравнение считывателей карт памяти можно увидеть в таблице 1.5, сам считыватель карт памяти можно увидеть на рисунке 3.6.

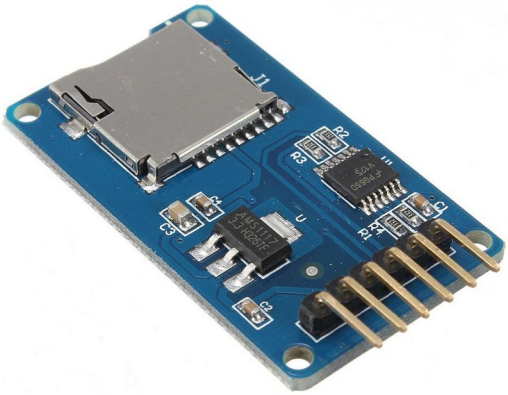

Рисунок 3.6 – Считыватель карт памяти

4 Разработка принципиальной электрической схему устройства

4.1 Светодиодная лента

В данном курсовом проекте используется светодиодная лента подключенная к соответствующему выводу микроконтроллера.

Для ограничения тока ленты используется резистор номиналом, рассчитываемым по следующей формуле:

$$R=\frac{U_n-U_d}{I_{np}}$$

Где U_n – напряжения питания, U_d – напряжение, падающее на светодиоде, I_{np} – прямой ток светодиода.

Светодиоды со следующими параметрами: $I_{np}=13.3.U_d=1$ Получаем:

$$R=\frac{5-1}{13.3\cdot10^{-3}}=300Ом$$

Следовательно чтобы лента работала исправно, её требуется подключать через резистор номиналом не меньше 300 Ом.

4.2 Микроконтроллеры

Информация об выбранных микроконтроллерах (Arduino Uno, ESP-32) представлена в пункте 3.1 раздела 3.

Микроконтроллер Arduino Uno отвечает за определение и обработку нажатий на клавиатуре. Матричная клавиатура разделена на четыре столбца и четыре линии которые соединены с цифровыми входами D2-D9 соответственно. D11 отвечает за соединение со светодиодной лентой. Помимо этого используются 2 аналоговых вывода (A4, A5) для связи с ESP-32 по интерфейсу I2C. Arduino Uno питается от напряжения 5В.

Поскольку мы уже не раз затрагивали тему интерфейса I2C, стоит подробнее рассмотреть данный интерфейс.

Шина I2C синхронная, состоит из двух линий: данных (SDA) и тактирования (SCL). Есть ведущий (master) и ведомые (slave). Инициатором обмена всегда выступает ведущий (в нашем случае Arduino UNO), обмен между двумя ведомыми невозможен. Всего на одной двухпроводной шине может быть до 127 устройств.

Такты на линии SCL генерирует master. Линией SDA могут управлять как мастер, так и ведомый в зависимости от направления передачи. Единицей обмена информации является пакет, обрамленный уникальными условиями на шине, именуемыми стартовым и стоповым условиями. Мастер в начале каждого пакета передает один байт, где указывает адрес ведомого и направление передачи последующих данных. Данные передаются 8-битными словами. После каждого слова передается один бит подтверждения приема приемной стороной. Пример схемотехники шины I2C можно увидеть на рисунке 4.1.

Процедура обмена начинается с того, что ведущий формирует **состояние СТАРТ**: при ВЫСОКОМ уровне на линии SCL он генерирует переход сигнала линии SDA из ВЫСОКОГО состояния в НИЗКОЕ. Этот переход воспринимается всеми устройствами, подключенными к шине, как признак начала процедуры обмена. Генерация синхросигнала – это всегда обязанность ведущего; каждый ведущий генерирует свой собственный сигнал синхронизации при пересылке данных по шине.

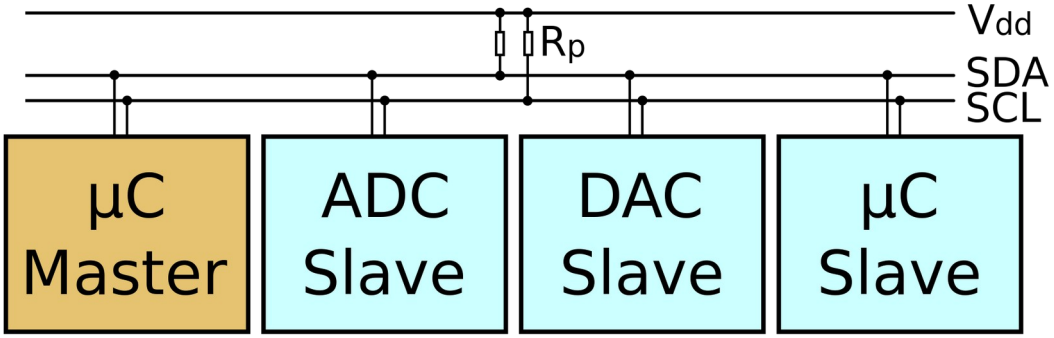

Рисунок 4.1 – схемотехника шины I2C.

При передаче посылок по шине I2C каждый ведущий генерирует свой синхросигнал на линии SCL. После формирования состояния СТАРТ ведущий опускает состояние линии SCL в НИЗКОЕ состояние и выставляет на линию SDA старший бит первого байта сообщения. Количество байт в сообщении не ограничено. Спецификация шины I2C разрешает изменения на линии SDA только при НИЗКОМ уровне сигнала на линии SCL. Данные действительны и должны оставаться стабильными только во время ВЫСОКОГО состояния синхроимпульса. Для подтверждения приёма байта от ведущего-передатчика ведомым-приёмником в спецификации протокола обмена по шине I2C вводится специальный бит подтверждения, выставляемый на шину SDA после приёма 8 бит данных.

Процедура обмена завершается тем, что ведущий формирует **состояние СТОП –** переход состояния линии SDA из НИЗКОГО состояния в ВЫСОКОЕ при ВЫСОКОМ состоянии линии SCL. Состояния СТАРТ и СТОП всегда вырабатываются ведущим. Считается, что шина занята после фиксации состояния СТАРТ. Шина считается освободившейся через некоторое время после фиксации состояния СТОП.

Таким образом, передача 8 бит данных от передатчика к приёмнику завершаются дополнительным циклом (формированием 9-го тактового импульса линии SCL), при котором приёмник выставляет низкий уровень сигнала на линии SDA, как признак успешного приёма байта.

Подтверждение при передаче данных обязательно, кроме случаев окончания передачи ведомой стороной. Соответствующий импульс синхронизации генерируется ведущим. Передатчик отпускает (переводит в ВЫСОКОЕ состояние) линию SDA на время синхроимпульса подтверждения. Приёмник должен удерживать линию SDA в течение ВЫСОКОГО состояния синхроимпульса подтверждения в стабильном НИЗКОМ состоянии. После этого ведущий может выдать состояние СТОП для прерывания пересылки данных.

Также в проекте используется Serial Peripheral Interface (SPI). SPI является синхронным последовательным интерфейсом связи, используемым для коротких расстояний между устройствами. Он часто используется во встроенных системах и других приложениях, которые требуют обмена данными между устройствами.

Для работы SPI необходимы четыре сигнала: сигнал тактовой частоты (SCLK), сигнал выбора ведомого устройства (SS), сигнал передачи данных от мастера к ведомому устройству (MOSI) и сигнал передачи данных от ведомого устройства к мастеру (MISO). Сигнал тактовой частоты генерируется мастер-устройством и используется для синхронизации коммуникации между мастером и ведомым устройством. Сигнал выбора ведомого устройства используется для выбора конкретного ведомого устройства, с которым мастер-устройство хочет общаться. Сигнал MOSI передает данные от мастера к ведомому устройству, в то время как сигнал MISO передает данные от ведомого устройства к мастеру.

SPI является протоколом полного дуплекса, что означает, что данные могут передаваться и приниматься одновременно. Он также может поддерживать несколько ведомых устройств, каждое с собственным сигналом выбора ведомого устройства.

SPI относительно простой и быстрый, что делает его популярным выбором для связи между микроконтроллерами, датчиками и другими встроенными устройствами. Он часто используется в приложениях, таких как сбор данных, взаимодействие с датчиками и коммуникация между периферийными устройствами.

Микроконтроллер ESP-32 соединён почти со всем элементами схемы:

- С цифро-аналоговым преобразователем D27-D25, общение происходит по интерфейсу I2S.
- С считывателем карт памяти на цифровых входах D5, D18, D9.
- С микроконтроллером Arduino Uno на аналоговых входах D22, D23.

Питается ESP-32 от напряжение 5В.

Так же более подробно рассмотрим интерфейс I2S.

I2S – стандарт интерфейса электрической последовательной шины, использующийся для соединения цифровых аудиоустройств. Шина I2S передает по разным линиям сигналы синхронизации и сигналы данных, что приводит к снижению фазового дрожания, типичного для систем связи, восстанавливающих сигналы синхронизации из целого потока.

Последовательный приёмник четко фиксирует уровни сигналов, которые тактируются синхронизатором. Далее эти данные побитно поступают по линии принимаемых данных в регистр сдвига, который синхронизирует данные от синхронизатора. После того, как регистр сдвига заполнится, он переписывается в буфер регистра приёма. Размер разрядности регистра сдвига задается длиной слова. После того, как буфер регистра регистрирует принятые данные, может произойти расширение знака. Так как заведомо используется формат с фиксированной запятой, надо расширить знак до 32-х разрядов, получив 12 разрядов данных. Старший разряд полученных данных распростра-

няется во все старшие разряды регистра, чтобы было правильное число, представленное в дробном формате.

Данные из регистра считываются в схему сужения знака, которая отбрасывает лишние разряды, сужает знак, выделяет передаваемые данные и записывает их в буфер передачи. Она сужает знак с разных частей и число записывается в соответствующие разряды передатчика. Как только регистр сдвига освободится, одновременно записываются биты сдвига и начинают выдвигать по одному разряду, начиная со старшего, передавая данные в передатчик, которые передает данные в выходную цепь.

4.3 Цифро-аналоговый преобразователь

Информацию об выбранном цифро-аналоговом преобразователе (далее ЦАП) можно увидеть в пункте 3.5 в разделе 3.

ЦАП использует выводы LR, BCLK, SD для связи с микроконтроллером ESP-32 для получения звуков и пересылает их на аудио адаптер с помощью выводов R,G,L. Питается ЦАП от 5В.

Принцип работы ЦАП заключается в суммировании аналоговых сигналов (ток или напряжение). Суммирование производится с коэффициентами, равными нулю или единице в зависимости от значения соответствующего разряда кода.

4.4 Считыватель карт памяти

Информацию об выбранном считывателе карт памяти можно увидеть в пункте 3.6 в разделе 3.

Считыватель карт памяти связан с ESP-32 выводами MISO, MOSI, CS, SCK. Питается от 3.3В.

Для форматирования карты памяти использовалась программа Gparted. FAT32 файловая системы была выбрана так как является очень эффективной при использовании на картах памяти, потому что она позволяет достаточно быстро получать доступ к файлам в этой файловой системе, но при этом ограничивает размер файлов до четырех гигабайт, в нашем случае это никак не повлияет.

4.5 Расчет мощности электрической схемы

Потребляемая мощность разрабатываемого устройства равна сумме мощностей, потребляемых его элементами. Расчет мощности элементов схемы устройства представлены в таблице 4.1

Таблица 4.1 – Расчёт мощности элементов схемы устройства

Блок	U, В	I, мА	Кол-во	P, мВт
Микроконтроллер Arduino Uno	5	22	1	110
Микроконтроллер ESP-32	5	20	1	100
Цифроаналоговый преобразователь	5	28	1	140
Считыватель карт памяти	3,3	40	1	132
Светодиодная лента(светодиоды)	5	13	16	1600

Суммарная мощность, мВт	1522

Таким образом потребляемая мощность будет равна:

$$P = 5 \cdot 22 + 5 \cdot 20 + 5 \cdot 28 + 3.3 \cdot 40 + 5 \cdot 13 \cdot 16 = 1522\ мВт$$

Учитывая поправочный коэффициент в 30%, максимальная потребляемая мощность составляет 1978 мВт

Рассчитаем потребляемый ток:

$$I = \frac{P}{U} = \frac{1978 \cdot 10^{-3}}{5} = 0.4\ A$$

4.6 Разработка устройства

Для того чтобы сделать печатные платы, устройство требуется разработать, для этого использовалась программа Easy EDA. Была выбрана именно эта программа так как она интуитивна понятная, имеет множество обучающих видеороликов и доступна на платформе Linux. Так же огромным плюсом является то, что в данном приложении есть возможность добавлять собственные элементы, если таких не оказалось в официальной библиотеке, что не раз помогло в сборке данного устройства.

После этого требуется преобразовать схему для печати на плате, для этого в программе есть редактор печатных плат. Можно попробовать переложить основную работу по проведению дорожек на программу, так как там поддерживается функция автопрокладки, но поскольку мы будем делать одностороннюю печать, дорожки придется проводить вручную. В местах где не удаётся свободно провести дорожки требуется сделать перемычки который будут соединять дорожки не примыкая к другим. Готовую печатную схему можно увидеть на рисунке 4.2.

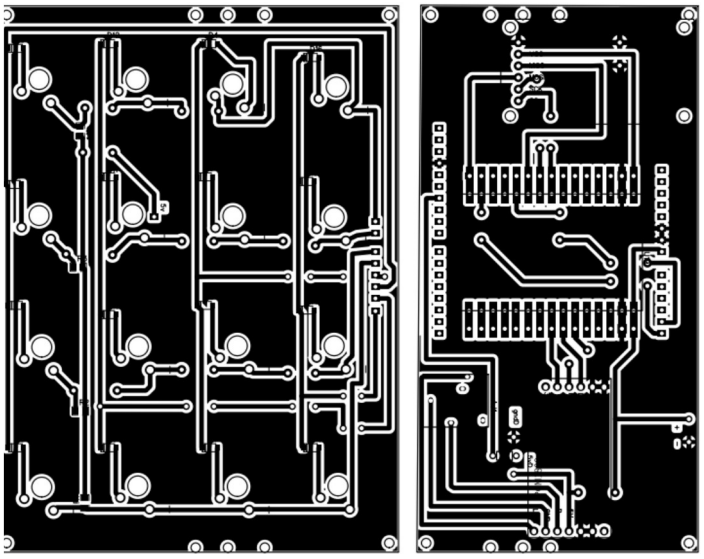

Рисунок 4.2 – Печатная схема микропроцессорного устройства синхронизированного вывода звуков

5 Разработка программного обеспечения

5.1 Требование к разработке программного обеспечения

Разработанное микроконтроллерное устройство работает следующим образом:

При подключении питания Arduino UNO должно подключиться к шине I2C и во время ожидания ответа переключить светодиоды в режим ожидания. Параллельно ESP-32 должно обнаружить карту памяти и отправить сигнал о завершении подготовки Arduino UNO.

Далее при нажатии на клавиши загорается светодиод, сигнализирующий об нажатии. Сигнал приходит на микроконтроллер Arduino Uno, которая связана с микроконтроллером ESP-32 по интерфейсу I2C. В данном случае master-ом (главным передающим) является Arduino.

Проанализировав сигнал, Arduino отправляет нужные указания и на ESP-32 происходит чтение нужного файла и соответствующая его отправка на цифро-аналоговый преобразователь. Преобразованный сигнал отправляется на аудио адаптер, где подсоединённая гарнитура должна проиграть звук.

Программирование микроконтроллеров осуществлялось в программе Arduino IDE, которая позволила установить необходимы для работы библиотеки и настроить работу с ESP-32, хоть и данный микроконтроллер подразумевает работу с другой программой (PlatformIO).

5.2 Исходный код обработчика клавиатуры

Обработчик клавиатуры (которым выступает Arduino UNO) должен определять какая клавиша была нажата и отправить код клавиши на ESP-32. Исходный код устройства можно найти в приложении Д.

В глобальной области данных объявлены следующие переменные:

- `Wire.h` и `microLED.h`: Подключение необходимых библиотек для работы с I2C и светодиодной лентой.
- `STRIP_PIN`: Пин, к которому подключены светодиоды.
- `NUMLEDS`: Количество светодиодов в ленте.
- `COLOR_DEBTH`: Глубина цвета светодиодов.
- `BRIGHTNESS`: Яркость светодиодов.
- `strip`: Объект светодиодной ленты для управления светодиодами.
- `P[]` и `M[]`: Массивы для хранения пинов строк и столбцов клавиатуры.
- `k4x4[][]`: Массив символов клавиатуры.
- `matrix[][]`: Матрица для отслеживания нажатых кнопок.
- `startArduino`: Флаг для определения готовности устройства.
- `I2C_address`: Адрес устройства на шине I2C.

`receiveEvent(int howMany)`: Функция для приема данных от ESP32 (но в реальности получает только сигнал о завершении настройки).

`getPosDiode(char letter)`: Функция, которая определяет позицию светодиода в ленте на основе символа клавиши.

`randomColor()`: Функция, которая генерирует случайный цвет для светодиодов формата RGB.

`lightsBlink(mData color)`: Функция, которая мигает светодиодами три раза с заданным цветом.

`lightsWaitRunning()`: Функция, которая в режиме ожидания мигает светодиодами, перемещаясь по всей ленте.

`lightsWaitFilling()`: Функция, которая в режиме ожидания заполняет светодиоды по порядку и затем очищает их.

`setup()`: Функция для настройки микроконтроллера. Здесь устанавливаются выводы клавиатуры, инициализируется светодиодная лента, настраивается коммуникация по шине I2C. Также ожидается сигнал от ESP32 для начала работы устройства.

`loop()`: Основная функция, которая проверяет нажатие кнопок клавиатуры, отправляет данные на ESP32 и управляет светодиодной лентой в соответствии с нажатыми кнопками.

5.3 Исходный код ESP-32

ESP-32 в данной схеме выполнят небольшой ряд функций, рассмотрим их.

В глобальной области данных объявлены следующие переменные:

- `Wire.h`: для работы с интерфейсом I2C. `AudioGeneratorMP3.h`: для декодирования mp3 файлов. `AudioOutputI2S.h`: для работы с интерфейсом I2S. `AudioFileSourceSD.h`: для работы с файлами на считывателе карт памяти
- `I2S_WS, I2S_BCK, I2S_DO`: выводы для работы с интерфейсом I2S.
- `SD_CS`: вывод для общения со считывателем карты памяти
- `AmountOfFiles`: количество файлов загруженных на SD карту
- `files[]`: названия файлов (кодируются одним символом клавиатуры)
- `I2C_address`: адрес ArduinoUNO на интерфейсе I2C
- `needToPlay`: флаг обозначающий потребность в воспроизведении мелодии.

`testFileO(fs::FS &fs. const char *path)`: Функция которая проверяет наличие и доступность файла на чтение.

`recieveEvent(int howMany)`: Функция для приема данных от ArduinoUNO.

`setup()`: Функция для настройки микроконтроллера, Здесь проверяется наличие SD карты, настраиваются интерфейсы I2S, I2C.

`loop()`: Основный цикл микроконтроллера который осуществляет воспроизведение мелодии в случае необходимости.

5.4 Блок-схема алгоритма

Блок-схема – это схематичное представление процесса, системы или компьютерного алгоритма. Блок-схемы часто применяются в разных сферах

деятельности, чтобы документировать, изучать, планировать, совершенствовать и объяснять сложные процессы с помощью простых логичных диаграмм.

Рассмотрим блок-схемы алгоритмов программного обеспечения данного курсового проекта, представленные в приложении Е.

На первом листе приложения представлена блок схема алгоритма Arduino UNO. Блоки 2–10 представляют собой подготовку программы для дальнейшей работы (инициализация переменных и определение модулей, подключенных к микроконтроллеру). Ключевыми являются блоки 11 – 24, которые реализуют саму логику программы в бесконечном цикле. В первую очередь мы выставляем низкий сигнал на одном из четырёх выводов Arduino(Блок 13).В блоке 15 проверяется подается ли сигнал с кнопок на выводах микроконтроллера при, в случае если нет, происходит передачи данных по интерфейсу I2C и включение нужного светодиода, иначе цикл идёт дальше без передачи данных (блоки 14–17, отправка данных в блоке 16). При удачном определении по шине I2C передаётся символ нажатой кнопки (распределение символов можно увидеть в приложении Д. Затем алгоритм проходит по всем кнопкам и проверяет приходит ли сигнал на микроконтроллер. Это нужно для определения уже ранее нажатых кнопок и дальнейшего с ними взаимодействия.

В это время так же работает второй микроконтроллер ESP-32, блок-схему алгоритма этого микроконтроллера можно увидать так же в приложении Е. В начале своей работы микроконтроллер инициализирует библиотеки и константные значения, далее инициализирует считыватель карт памяти и проводит проверки на успешность операций (блоки 2-7). Далее идет основной, вечный цикл программы (блоки 8–24). Поскольку второй микроконтроллер полностью зависит от работы первого, то пока первый микроконтроллер не начнет присылать данные, второй микроконтроллер не начнет работу, это можно увидеть в блоках 19-24 где описывается цикл очистки шины I2C, если шина не хранила никакую информацию, то микроконтроллер переходил в режим ожидания. После того как микроконтроллер получил информацию об нажатой клавише, он воссоздает название файла. Формат названия файла имеет форму: «КодовыйСимволКнопки.mp3». Далее, если не удалось отрыть файл, то цикл начинается с начала, но как показывает практика, такое может произойти только если файлы на карте памяти были неправильно названы. Если прочитать файл всё же удалось, то ESP-32 по I2S интерфейсу пересылает данные на цифро-аналоговый преобразователь.

Заключение

В результат работы над данным курсовым проектом был разработан работоспособный музыкальный пад. Устройство способно проигрывать звуки как через динамики, так и через наушники. Несмотря на размеры устройства выходящий звук получается достаточно чистым (без всяких помех), а количество кнопок позволяет проигрывать полноценные мелодии любых современных жанров. Данный проект был спроектирован в соответствии с поставленными задачами, весь функционал был реализован в полной объеме.

Разработанное устройство обладает следующими достоинствами:

- Низкая стоимость по сравнению с существующими устройствами
- Чистый звук
- Простота использования
- Портативность

Данное устройство можно улучшить, реализуя следующие дополнительные функции:

1. Добавление поддержки беспроводной связи: Встроение модуля Bluetooth или Wi-Fi позволит устройству воспроизводить звуки с внешних источников, таких как смартфоны или компьютеры, без необходимости физического подключения.
2. Интеграция с музыкальными платформами: Подключение к онлайн-музыкальным сервисам, таким как Spotify или Apple Music, позволит пользователям транслировать музыку непосредственно с платформы и проигрывать ее на устройстве.
3. Добавление эффектов и настроек звука: Включение настройки эквалайзера, эффектов пространственного звучания или возможности регулировки громкости и тембра позволит пользователям настраивать звук в соответствии с их предпочтениями.
4. Улучшение дизайна и эргономики: Оптимизация размеров и формы устройства, добавление удобных кнопок управления, улучшение индикации статуса и возможность выбора различных цветов и схем освещения помогут создать более привлекательный и удобный пользовательский опыт.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1]. Документация Arduino [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.arduino.cc/ – дата доступа 10.03.2023

[2]. Документаиця Raspberry PI [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.raspberrypi.com/ – дата доступа 10.03.2023

[3]. Даташит ESP-32 [Электронный ресурс]. – Электронные данные. – Режим доступа https://www.espressif.com/site/default/files/documentation/esp32_datasheet_en.pdf – дата доступа 10.03.2023

[4]. Матричные клавиатуры и arduino. [Электронный ресурс]. – Электронные данные. – Режим доступа https://habr.com/ru/post/664892/ – дата доступа 18.03.2022

[5]. Интерфейс I2C [Электронные ресурс] – Электронные данные. – Режим доступа http://www.i2c-bus.org/i2c-Interface/ – дата доступа 19.03.2023

[6]. Интерфейс I2S [Электронный ресурс] – Электронные данные. – Режим доступа https://www.allaboutcircuits.com/technical-articles/introduction-to-the-i2s-interface/ – дата доступа 19.03.2023

[7]. Конфигурация считывателя карты памяти с ESP-32 [Электронный ресурс]. – Электронные данные. – Режим доступа https://www.mischianti.org/ – дата доступа 20.03.2023

[9]. Даташит цифроаналогово приобразователя PCM5102A [Электронный ресурс]. – Электронные данные. – Режим доступа https://static.chipdip.ru/lib/145/DOC004145096.pdf – дата доступа 02.04.2023

[10]. Вычислительные машины системы и сети: дипломное проектирование (методическое пособие) [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.bsuir.by/m/12_100229_1_136308.pdf – дата доступа 10.04.2023

[11]. ГОСТ 2.730-73 ЕСКД [Электронный ресурс]. – Электронные данные. – Режим доступа: https://znaytovar.ru/gost/2/GOST_273073_ESKD_Oboznacheniya.html – дата доступа 10.04.2023

[12]. Информация об цифроаналоговых преобразователях [Электронные ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Цифро-аналоговый_преобразователь – дата доступа 15.04.2023

[13]. Статья об диодах [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.ruselectronic.com/poluprovodnikovyj-diod-i-jego-vidy/ – дата доступа 16.04.2023

[14]. Статья об использовании Arduino IDE [Электронные ресурс]. – Электронные данные. – Режим доступа: https://alexgyver.ru/lessons/arduino-ide/ – дата доступа 20.04.2023

ПРИЛОЖЕНИЕ А

(обязательное)

Структурная схема устройства

ПРИЛОЖЕНИЕ Б

(обязательное)

Функциональная схема устройства

ПРИЛОЖЕНИЕ В

(обязательное)

Принципиальная схема устройства

ПРИЛОЖЕНИЕ Г
(обязательное)
Перечень элементов

ПРИЛОЖЕНИЕ Д
(обязательное)
Листинг кода

```
001. #include <Wire.h>       // I2C library
002. #include <microLED.h>   // LED library
003.
004. #define STRIP_PIN 11     // pin for the diodes
005. #define NUMLEDS 16      // number of diodes
006. #define COLOR_DEBTH 1  // color depth of the diodes
007. #define BRIGHTNESS 60 // brightness of the diodes
008. microLED<NUMLEDS, STRIP_PIN, MLED_NO_CLOCK, LED_WS2818, OR-
DER_GRB, CLI_AVER> strip; //Initializing LED
009.
010. const int P[] = {5, 4, 3, 2};     // rows
011. const int M[] = {9, 8, 7, 6};    // columns
012. const char k4x4 [4][4] = {      // keyboard symbols
013.   {'R', '6', 'I', 'A'},
014.   {'M', '1', '0', 'E'},
015.   {'U', '9', 'G', 'S'},
016.   {'D', 'F', 'N', 'B'}
017. };
018. int matrix[4][4];           // matrix for remembering
pressed buttons
019. bool startArduino = false;  // flag for starting arduino
020. const int I2C_address = 21; // ESP32 address
021.
022. // Function to receive data from ESP32 (basically receives
only success from esp32)
023. void receiveEvent(int howMany) {
024.   char letter = '\0';
025.   while (Wire.available())
026.     letter = Wire.read();
027.
028.   if(letter == 'S')
029.     startArduino = true;
030. }
031.
032. // Since diodes connect differently, we need to determine
which actual diode we need to light up
033. byte getPosDiode(char letter){
034.   char allLetters[] = {'B', 'A', 'S', 'E', '0', 'G', 'I',
'N', 'F', '6', '9', '1', 'M', 'U', 'R', 'D'};
035.   for(byte i = 0; i < 16; i++)
036.     if (allLetters[i] == letter)
037.       return i;
038.   Serial.println("WARNING: pressed letter was not found\
n");
039. }
040.
041. // Randoms color for diodes
042. mData randomColor(){
```

```
043.   // we end to 200 since all 255 is white color with is
very greedy
044.   // we start from 20 so diodes wound be blank
045.   return mRGB(random(20,200), random(20,200),
random(20,200));
046. }
047.
048. // Makes diodes blink 3 times with setted color
049. void lightsBlink(mData color){
050.   for(byte i = 0; i < 3; i++){
051.     strip.clear();
052.     strip.show();
053.     delay(500);
054.     for(byte j = 0; j < NUMLEDS; j++)
055.       strip.set(j, color);
056.     strip.show();
057.     delay(500);
058.   }
059. }
060.
061. // wait in running mode (yellow diode runs through all
diodes)
062. void lightsWaitRunning(){
063.   for(byte i = 0; i < NUMLEDS; i++){
064.     receiveEvent(1);
065.     if(startArduino == true)
066.       return;
067.     strip.clear();
068.     strip.set(i, mRGB(255, 255, 0));
069.     strip.show();
070.     delay(600);
071.   }
072.
073.   for(int i = NUMLEDS - 1; i >= 0; i--){
074.     receiveEvent(1);
075.     if(startArduino == true)
076.       return;
077.     strip.clear();
078.     strip.set(i, mRGB(255,255,0));
079.     strip.show();
080.     delay(600);
081.   }
082. }
083.
084. void lightsWaitFilling(){
085.   strip.clear();
086.   delay(100);
087.   for(byte i = 0; i < NUMLEDS; i++){
088.     receiveEvent(1);
089.     if(startArduino == true)
090.       return;
091.     strip.set(i, mRGB(255, 255, 0));
092.     strip.show();
093.     delay(600);
094.   }
```

```
095.   delay(100);
096.   for(int i = NUMLEDS - 1; i >= 0; i--){
097.     receiveEvent(1);
098.     if(startArduino == true)
099.       return;
100.     strip.set(i, mRGB(0,0,0));
101.     strip.show();
102.     delay(500);
103.   }
104. }
105.
106.
107. void setup() {
108.   Wire.begin(18);                 // First, we connect as
slave to i2c bus
109.   Wire.onReceive(receiveEvent);  // setting function for
receiving data
110.
111.   // keyboard set up
112.   for(byte i = 0; i < 4; i++)
113.     for(byte j = 0; j < 4; j++)
114.       matrix[i][j] = 0;
115.   for (byte i = 0; i < 4; i++) {  // making pin out rows -
exit, columns - enter
116.     pinMode(P[i], OUTPUT);
117.     pinMode(M[i], INPUT_PULLUP);
118.     digitalWrite(P[i], HIGH);
119.   }
120.
121.   Wire.setClock(400000);
122.   strip.setBrightness(BRIGHTNESS); // Setting up LED
123.
124.   Serial.begin(9600);
125.   Serial.println("begin");
126.
127.   // Waiting for ESP32 to start
128.   int waitingMode = 0;
129.   while(startArduino == false){
130.     if(waitingMode == 0){
131.       lightsWaitRunning();
132.       waitingMode = 1;
133.     }
134.     else if(waitingMode == 1){
135.       lightsWaitFilling();
136.       waitingMode = 0;
137.     }
138.   }
139.   Wire.end();                    // now we wont be slave
140.   Wire.begin(I2C_address);       // now we are master
141.   lightsBlink(mRGB(0, 255, 0)); // Showing user that set up
is complete
142.   delay(500);
143.   strip.clear();
144.   strip.show();
145. }
```

```
146.
147. void loop() {
148.   Wire.beginTransmission(I2C_address);
149.   for (byte p = 0; p < 4; p++) {    // placing LOW on rows
150.     digitalWrite(P[p], LOW);
151.     for (byte m = 0; m < 4; m++) {  // checking if row is
actually LOW
152.       if (!digitalRead(M[m]) && matrix[p][m] == 0) {
153.         Serial.print("Pressed Button: ");
154.         Serial.print(k4x4[p][m]);
155.         Serial.print("\n");
156.
157.         Wire.write(k4x4[p][m]);
158.         strip.set(getPosDiode(k4x4[p][m]), randomColor());
159.         strip.show();
160.         matrix[p][m] = 1;
161.       } else if (digitalRead(M[m]) && matrix[p][m] == 1){
162.         Serial.print("Unpressed Button: ");
163.         Serial.print(k4x4[p][m]);
164.         Serial.print("\n");
165.
166.         strip.set(getPosDiode(k4x4[p][m]), mRGB(0,0,0));
167.         strip.show();
168.         matrix[p][m] = 0;
169.       }
170.     }
171.     digitalWrite(P[p], HIGH);       // return pin to HIGH
172.   }
173.   Wire.endTransmission();
174. }
```

Код ESP-32

```
001. #include <Wire.h>                // I2C library
002. #include <AudioGeneratorMP3.h>  // library for mp3 decode
003. #include <AudioOutputI2S.h>     // I2S configuration li-
brary
004. #include <AudioFileSourceSD.h>  // library with ready-to-
use function for file reading and etc
005.
006. // DA converter pins
007. #define I2S_WS 26
008. #define I2S_BCK 27
009. #define I2S_DO 25
010. #define SD_CS 5 // pin for SD card
011. #define RATE 44100
012.
013. const byte AmountOfFiles        = 14;
014. const char files[AmountOfFiles] = {'D', 'R', 'U', 'M', 'F',
'6', '9', 'N', 'I', 'G', 'B', 'A', 'S', 'E'};
015. const int I2C_address           = 18; // Arduino address
016.
017. AudioGeneratorMP3 *mp3;
018. AudioFileSourceSD *file;
```

```
019. AudioOutputI2S    *out;
020. bool needToPlay = false;
021.
022. char fileName[] = "/Z.mp3\0";
023.
024. // test if FIle is accessible
025. int testFileO(fs::FS &fs, const char *path) {
026.   File file = fs.open(path);
027.   static uint8_t buf[101];
028.   size_t len = 0;
029.   if (file) {
030.     len         = 1;
031.     size_t flen = len;
032.     while (len != 0) {
033.       file.read(buf, len--);
034.     }
035.     file.close();
036.     return 0;
037.   }
038.   return 1;
039. }
040.
041. // function for handling I2C data transmitting
042. void receiveEvent(int howMany) {
043.   if(needToPlay)  // if some song still playing, we do not
take
044.     return;
045.   char songName = '\0'; // basically we get only one letter
from transmitting
046.   while (Wire.available()) { // in case if I2C wire is
filled up with some mess, we just skip all to the last character
047.     songName = Wire.read();
048.     Serial.print("Get from I2C: ");
049.     Serial.print(songName);
050.     Serial.print("\n");
051.   }
052.   fileName[1] = songName;
053.   // check if file exists
054.   if (testFileO(SD, fileName))
055.     return;
056.
057.   delete file;
058.   file       = new AudioFileSourceSD(fileName);
059.   needToPlay = true;
060. }
061.
062. void setup() {
063.   delay(2000); // this need to avoid situation when esp32
already send success code, but arduino UNO is not ready to get
it
064.   Serial.begin(115200);
065.   Serial.print("Begin\n");
066.
067.   pinMode(SD_CS, OUTPUT);
068.   digitalWrite(SD_CS, HIGH);
```

```
069.
070.   // setting up card reader
071.   while (!SD.begin(SD_CS)) {
072.     Serial.println("SD card is not mounted");
073.     delay(1500);
074.   }
075.   Serial.println("Successfully mounted SD card");
076.   uint8_t cardType = SD.cardType();
077.   while (cardType == CARD_NONE) {
078.     Serial.println("SD card is not attached");
079.     delay(1000);
080.   }
081.   Serial.println("SD card is attached");
082.
083.   // Initialize I2S
084.   out = new AudioOutputI2S();
085.   out->SetPinout(I2S_BCK, I2S_WS, I2S_DO);
086.   out->SetGain(0.5);
087.
088.   // Checking files (debug info)
089.   for (int i = 0; i < AmountOfFiles; i++) {
090.     fileName[1] = files[i];
091.     Serial.print(fileName);
092.     if (testFileO(SD, fileName)) {
093.       Serial.println(" -- not found");
094.       continue;
095.     } else
096.       Serial.println(" -- found");
097.   }
098.
099.   char temp = 'S'; // Success
100.   Wire.begin();   // joining I2C as master
101.   Wire.beginTransmission(18); // beginning transmitting
with arduino
102.   Wire.write(temp);
103.   Wire.endTransmission();
104.   Serial.print("Success sent\n");
105.   Wire.end();
106.   Wire.begin(21); // joining I2C as slave
107.   Wire.onReceive(receiveEvent); // setting function for
handling i2c transmitting
108.   file = new AudioFileSourceSD(fileName); // dummy file
109. }
110.
111. void loop() {
112.   delay(100);
113.   if (!needToPlay)
114.     return;
115.
116.   // Open MP3 file from SD card
117.   mp3 = new AudioGeneratorMP3();
118.   mp3->begin(file, out);
119.
120.   Serial.println("Playing music...");
121.
```

```
122.   // Play music until the end of the file
123.   while (mp3->isRunning()) {
124.     if (!mp3->loop()) {
125.       mp3->stop();
126.     }
127.   }
128.
129.   Serial.println("Music playback complete");
130.
131.   // Clean up
132.   delete mp3;
133.   needToPlay = false;
134. }
```

ПРИЛОЖЕНИЕ Е
(обязательное)
Блок-схемы программ