МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра информационных систем

Музыкальный звонок

Курсовая работа по дисциплине «Введение в интернет вещей»

09.03.02 Информационные системы и технологии

6 семестр 2024/2025 учебного года

Зав. кафедрой _____к. т. н., доцент Д.Н. Борисов

Обучающийся _____ст. 3 курса оч. отд. Д.В. Подповетный

Руководитель _____старший преподаватель, А.В. Максимов

Воронеж 2025

Содержание

Введение

В условиях стремительного развития цифровых технологий и увеличения числа подключенных устройств Интернет Вещей (IoT) становится неотъемлемой частью повседневной жизни. Концепция IoT предполагает интеграцию физических объектов в единую сеть, обеспечивая их взаимодействие между собой и удалённое управление через интернет. Это открывает широкие возможности для автоматизации процессов как на промышленных объектах, так и в быту. Одним из ключевых факторов популярности IoT стало развитие недорогих и функциональных аппаратных платформ, позволяющих разработчикам и энтузиастам создавать собственные устройства даже в домашних условиях. Особое место среди таких решений занимает микроконтроллер ESP32, выпускаемый компанией Espressif. Благодаря своей производительности, энергоэффективности и наличию встроенных модулей Wi-Fi и Bluetooth, ESP32 стал одним из самых востребованных чипов для реализации IoT-проектов. Его использование позволяет относительно просто организовать сбор данных с датчиков, обработку информации и передачу её по беспроводной сети. Это делает возможным создание автономных систем, управляемых через веб-интерфейс или мобильные приложения. В рамках данного подхода представляет интерес разработка музыкального звонка, подключённого к локальной Wi-Fi сети и доступного через веб-браузер по заданному IP-адресу. Такое устройство может стать примером реализации простого, но функционального IoT-приложения, демонстрирующим принципы построения сетевых интерфейсов, управления периферийными устройствами и организации удалённого доступа к ним.

1 Постановка задач

1.1 Цель и задачи

Целью данной курсовой работы является реализация музыкального звонка на esp32, который управляется по WIFI через веб-интерфейс. Стоит обозначить основные задачи, которые необходимо достичь, чтобы реализовать продукт данной работы:

- Выбрать необходимые комплектующие для сборки схемы устройства;
- Разработать схему в EasyEDA и сделать разводку печатной платы;
- Подсчитать себестоимость при итоговой сборке;
- Привести диаграмму вариантов использования данного устройства;
- Написать код для музыкального звонка для ESP32;
- Реализовать бэкенд и фронтенд и развернуть их через Docker;
- Протестировать систему и сделать выводы.

1.2 Актуальность и обзор аналогов

Актуальность данного устройства можно хорошо описать через реальные сценарии, в которых используются дверные звонки и к каким проблемам это приводит:

При использовании дверных звонков, вероятны поломки и ложные срабатывания из-за недобросовестных соседей. Мое устройство на ESP32, которое будет выполнять функцию звонка (с запуском музыкальных мелодий), не будет возможности физического доступа к нему, что повысит безопасность, и только доверенные лица будут иметь доступ к веб-интерфейсу, а именно те, у которых будет пройдена авторизация в вашей WiFi-сети.

Среди аналогов можно выделить слудющие:

1. TP-Link Kasa Smart Doorbell Button (KB100)

Это беспроводная кнопка звонка, подключается к Wi-Fi. При нажатии отправляет уведомление и активирует звуковой сигнал через интеграцию с другими устройствами.

2. Ring Video Doorbell

Подключается к WiFi. При нажатии на кнопку или обнаружении движения отправляет уведомления в приложение на смартфон.

3. Xiaomi Mi Doorbell Camera

Также работает через Wi-Fi. Отправляет уведомления и позволяет отвечать гостю через приложение.

Но у всех перечисленных аналогов есть общие проблемы:

- Физическое взаимодействие с устройствами (риск поломки);
- Требуется интеграция с другими устройствами (отдельно колонки);
- Завышенная цена, начиная с дешевого ценового сегмента.

.

2 Разработка устройства

2.1 Описание устройства

2.1.1 ESP-WROOM-32 Type C

В качестве центрального элемента системы выбран модуль ESP-WROOM-32 Type C, представляющий собой Wi-Fi и Bluetooth модуль на базе чипа ESP32. Он был выбран благодаря наличию встроенных беспроводных интерфейсов, что позволяет организовать подключение устройства к локальной Wi-Fi сети и обеспечить доступ к нему через веб-интерфейс по указанному IP-адресу. ESP32 имеет достаточное количество GPIO-пинов для управления внешними периферийными устройствами, а также поддерживает аналоговые и цифровые сигналы. Для корректной работы аудиосистемы используется встроенный ЦАП (DAC), который генерирует аналоговый сигнал для передачи на усилитель.

Подключение ESP-WROOM-32:

● DAC → IN усилителя TDA2030A

● GPIO, VCC, GND — для возможного расширения функционала в будущем

● Wi-Fi антенна — интегрирована в модуль

Основные характеристики ESP-WROOM-32:

● Процессор: двухъядерный Xtensa LX6 с тактовой частотой до 240 МГц.

● Интерфейсы: Wi-Fi 802.11 b/g/n, Bluetooth 4.2 BLE.

● Питание: 3,3 В.

● Диапазон рабочих температур: -40°C до +125°C. Встроенный ЦАП и ШИМ контроллер для управления аудио и другими периферийными устройствами.

Рисунок 1 - Модуль ESP-WROOM-32 (Type C)

2.1.2 Аудио-усилитель TDA2030A

В качестве усилителя звуковой частоты в проекте используется микросхема TDA2030A — это монофонический аудиоусилитель средней мощности, предназначенный для работы в составе различных звуковых устройств. Микросхема способна обеспечить выходную мощность до 18 Вт при питании от двуполярного источника, однако в данном проекте она работает в режиме с однополярным питанием, что позволяет обойтись более простой схемой питания и использовать стандартные источники напряжения. Усилитель обеспечивает высокое качество звучания, достаточное для воспроизведения музыкальных мелодий в бытовых условиях.

Подключение TDA2030A:

● IN → Выход ЦАП ESP32 (DAC)

● Vs+ → Питание +12 В

● Vs− → GND (при однополярном питании)

● OUT → Вход динамика YD103-08

● Дополнительно используются фильтрующие конденсаторы на входе и выходе усилителя для подавления шумов и стабилизации сигнала.

Основные характеристики TDA2030A:

● Выходная мощность: до 14 Вт на нагрузке 4 Ом при питании 12 В.

● Диапазон питающих напряжений: от 6 до 22 В.

● Низкие нелинейные искажения и высокая степень подавления пульсаций питания.

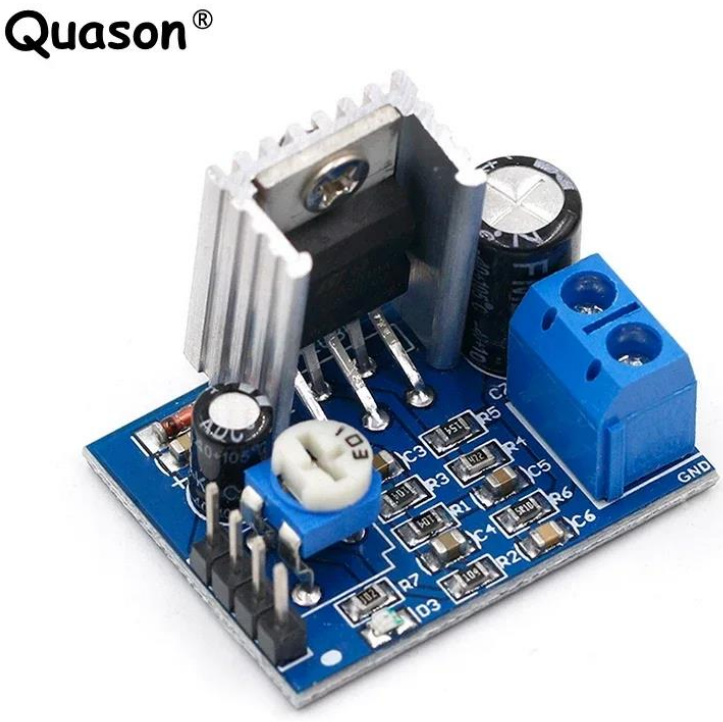


Рисунок 2 - Аудиоусилитель TDA2030A

2.1.3 Динамик YD103-08

Для вывода звукового сигнала используется широкополосный динамик YD103-08 с номинальной мощностью 8 Вт и сопротивлением катушки 4 Ом. Этот тип динамика обеспечивает хорошее качество звучания в диапазоне средних и высоких частот, что делает его подходящим для воспроизведения музыкальных мелодий. Благодаря компактным размерам и стандартному импедансу, динамик легко интегрируется в электронную схему и совместим с выходом усилителя TDA2030A.

Характеристики динамика YD103-08:

- Мощность: 8 Вт.
- Сопротивление: 4 Ом.
- Частотный диапазон: 200 Гц – 20 кГц.
- Тип: широкополосный динамик.

Рисунок 3 - Динамик YD103-08

2.1.4 Схема устройства

Схема устройства была разработана в программе EasyEDA и включает следующие основные компоненты:

● Микроконтроллер ESP-WROOM-32 (Type C).

● Аудиоусилитель TDA2030A.

● Динамик HY1201GP (в реальной сборке YD103-08).

● Подключение к источнику питания через USB Type-C с использованием стабилизированного напряжения 5 В.

Основные элементы схемы:

● ESP-WROOM-32: является центральным элементом системы. Он подключен к аудиоусилителю TDA2030A через ЦАП (DAC), который генерирует аналоговый сигнал для воспроизведения музыкальных мелодий. Для управления усилителем используется один из GPIO-пинов ESP32. Питание модуля осуществляется от стабилизированного напряжения 3,3 В, которое выделяется из общего блока питания 5 В через соответствующий стабилизатор напряжения.

● TDA2030A: Усилитель звуковой частоты, подключённый к выходу ЦАП ESP32. На вход усилителя поступает аналоговый сигнал, который усиливается до уровня, достаточного для воспроизведения через динамик. Выход усилителя подключен непосредственно к динамику YD103-08.

● Динамик HY1201GP (в реальной сборке YD103-08): Предназначен для воспроизведения звука. Его катушка подключена к выходу усилителя TDA2030A, а корпус заземлён для снижения помех.

На рисунке 4 представлена схема устройства в EasyEDA:

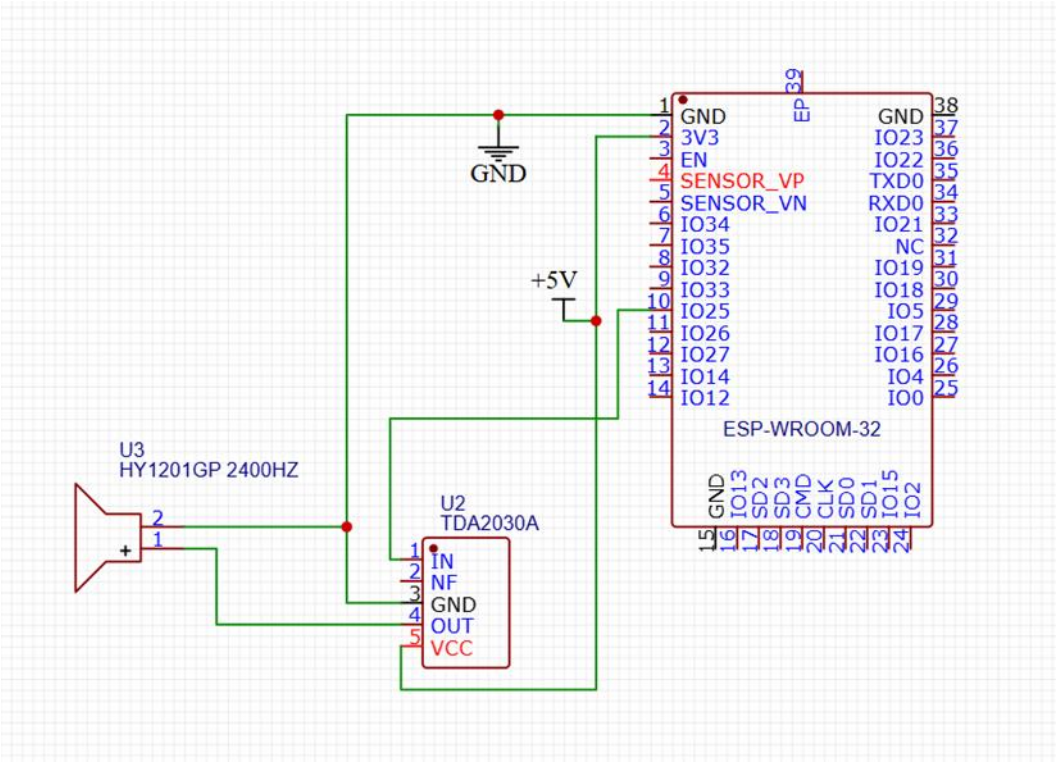


Рисунок 4 - Схема устройства

Описание подключения компонентов:

ESP-WROOM-32:

GPIO Pin (DAC) → Подключен к входу IN усилителя TDA2030A для передачи аналогового звукового сигнала.

VCC → Питание 3,3 В, получаемое через стабилизатор напряжения от общего блока питания 5 В.

GND → Общий земляной проводник.

TDA2030A:

IN → Подключен к выходу ЦАП ESP32 для получения аналогового звукового сигнала.

VCC → Питание +5 В, подключено непосредственно от источника питания.

GND → Заземление.

OUT → Подключен к динамику YD103-08 для воспроизведения звука.

Динамик YD103-08:

Вход → Подключен к выходу OUT усилителя TDA2030A.

Корпус → Заземлён для снижения помех.

Источник питания:

Используется разъем USB Type-C для подключения внешнего питания 5 В. Напряжение 5 В распределяется между всеми компонентами, а для питания ESP32 используется стабилизированный блок, преобразующий 5 В в 3,3 В.

2.1.5 Разводка платы

Печатная плата была разработана с учётом минимизации длины проводников, особенно для линий питания и сигналов. Особенности разводки:

- Линии питания (3,3 В, 5 В) имеют достаточно широкую проводку (не менее 0,8 мм), чтобы снизить падение напряжения.
- Печатная плата позволяет легко интегрировать устройство в корпус и подключить его к другим системам через USB или внешний источник питания.

На рисунке 5 и 6 изображены разводки платы в разных формах представления:

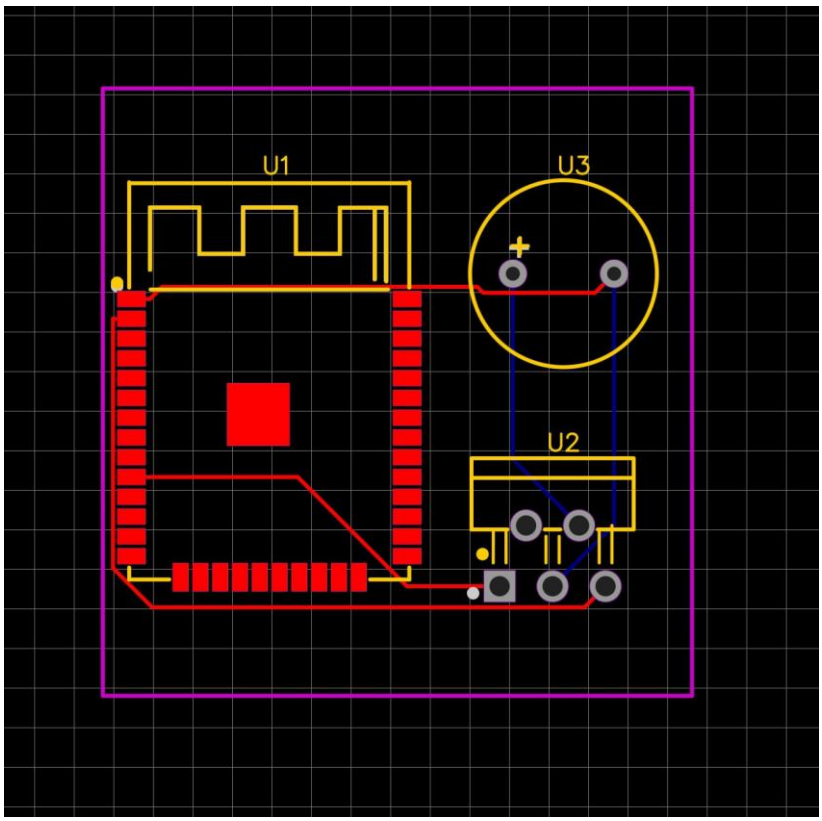


Рисунок 5 - Разводка платы

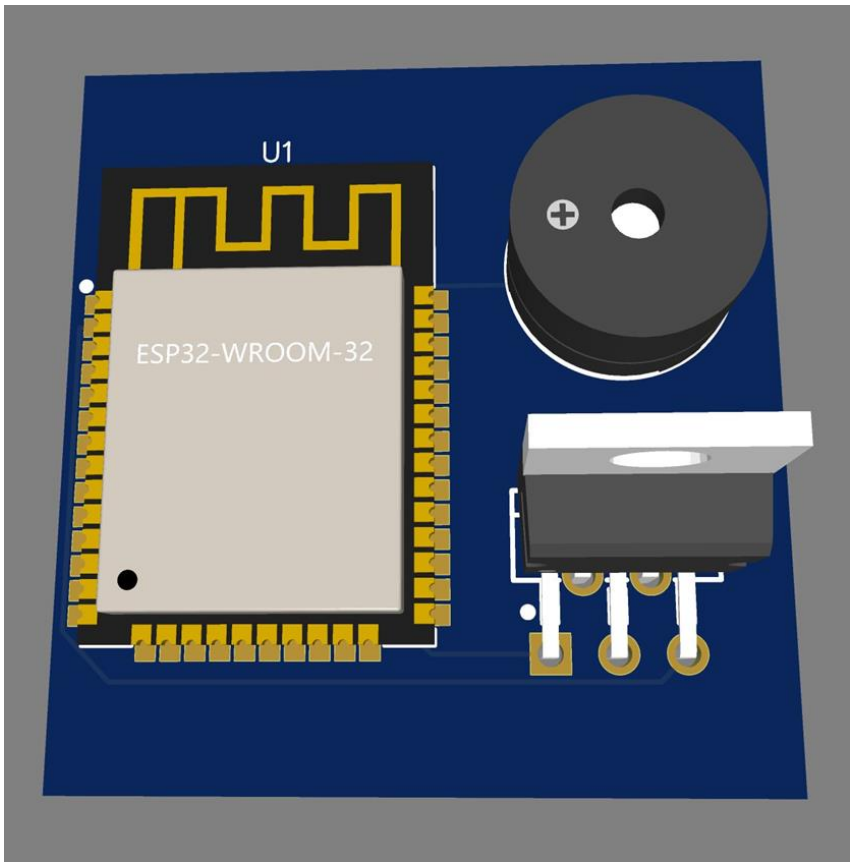


Рисунок 6 - 3D представление платы

2.1.6 Себестоимость

Итого при DIY-сборке схемы вышла следующая сумма (Рисунок 7).

Аудио-усилитель TDA2030A	56 руб.
Акустический динамик YD103-08, 8 Вт, 4 Ом	320 руб.
Соединительный провод	154 руб.
ESP-WROOM-32 CH340C	259 руб.
Итого (DIY сборка)	789 руб.

Рисунок 7 - Себестоимость DIY-сборки

Получившаяся сборка, комплектующие для которой были закуплены, чтобы максимально соответствовать тем, что были использованы в схеме в Easy EDA, представлена в Приложении В.

Если заказывать плату с завода, которая была создана и в Easy EDA, то это дополнительно обошлось примерно в 2000 рублей. Это только печать и монтаж платы.

2.2 Диаграммы

2.2.1 Use Case диаграмма

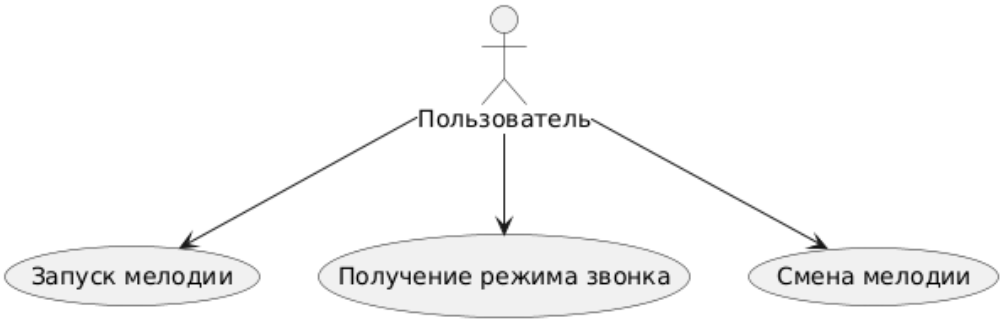


Рисунок 8 - Use Case диаграмма музыкального звонка

Разберем поподробнее основные три варианта использования, которые будут реализованы для музыкального звонка.

Запуск мелодии происходит при нажатии в веб-интерфейсе на кнопку «Позвонить» (Рисунок 9).

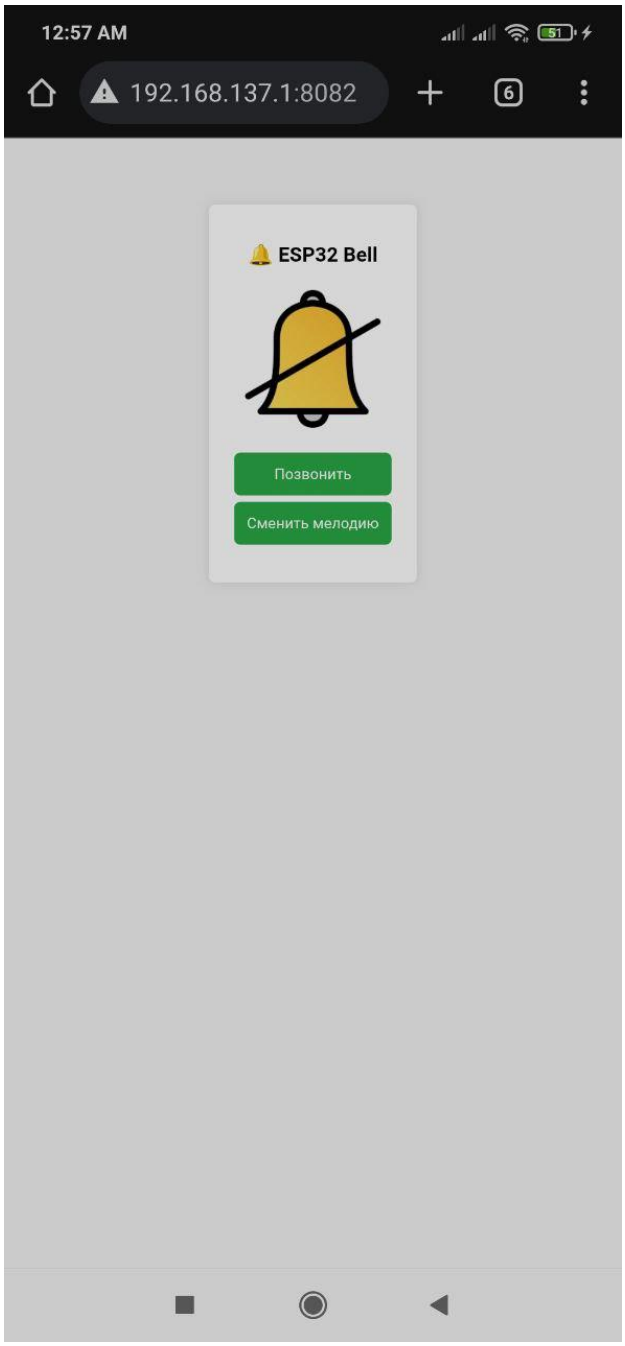


Рисунок 9 - Веб-интерфейс в стартовом режиме

Обновление статуса звонка (именно изображения в веб-интерфейсе), когда начинается/заканчивается проигрывание музыкальной мелодии (Рисунок 10).

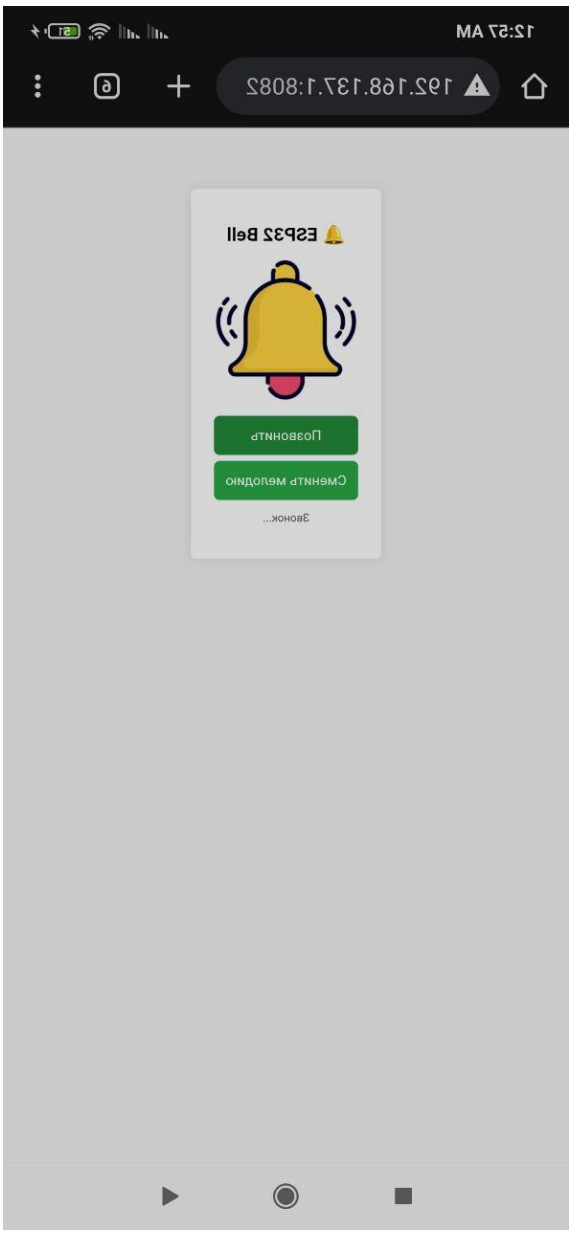


Рисунок 10 - Отображение статуса проигрывания мелодии

Смена мелодии в веб-интерфейсе на следующую мелодии (задана в коде для ESP32 по-умолчанию) (Рисунок 11).

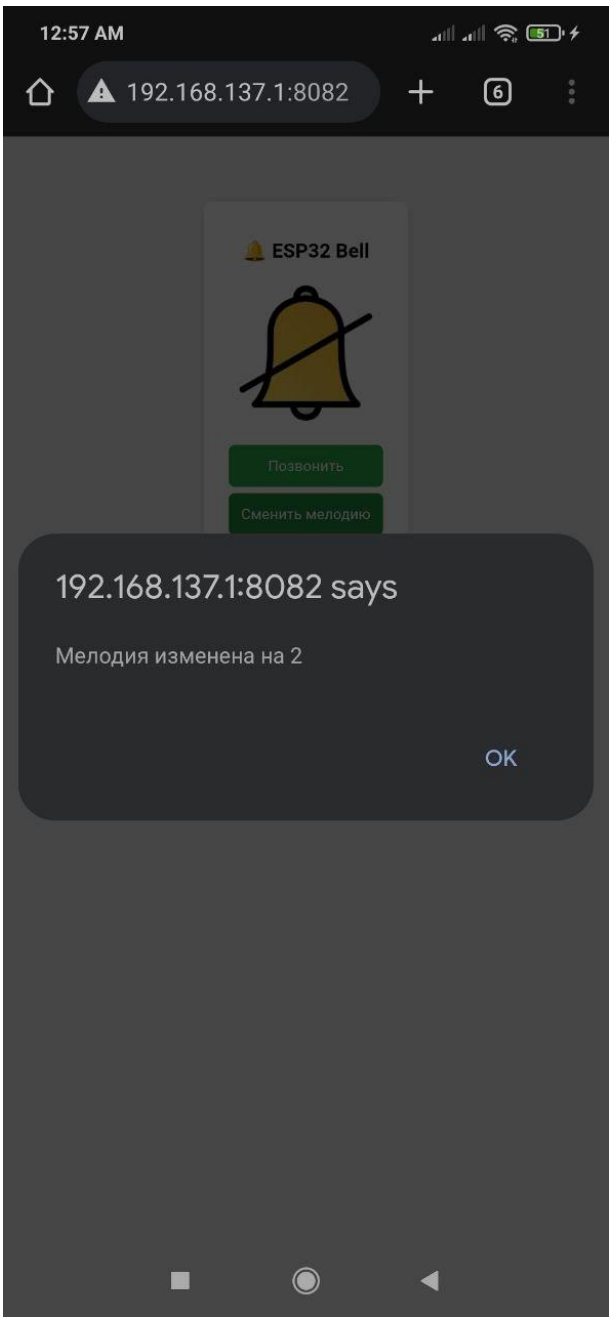


Рисунок 11 - Окно с информацией при смене мелодии

3 Разработка программного софта

Абсолютно весь код, который был реализован в проекте, находится в публичном доступе. Ссылка на публичный репозиторий GitHub приведена в Приложении А.

3.1 Код для ESP32

Данный программный код реализует устройство воспроизведения мелодий на базе микроконтроллера ESP32, которое позволяет управлять

воспроизведением с помощью веб-интерфейса и поддерживает переключение между несколькими предустановленными мелодиями.

1. Подключение необходимых библиотек

```
#include <WiFi.h>           // Для работы с Wi-Fi
#include <WebServer.h>      // Для создания веб-сервера
#include <ArduinoJson.h>    // Для формирования JSON-ответов API
```

- WiFi.h и WebServer.h обеспечивают подключение к беспроводной сети и создание локального веб-сервера для управления устройством.
- ArduinoJson.h используется для формирования структурированных JSON-ответов, возвращаемых по API.

2. Настройка вывода звука и хранения мелодий.

```
#define SPEAKER_PIN 25  // Пин подключения динамика
// Учетные данные для подключения к Wi-Fi
const char* ssid = "LAPTOP-5ORAQVBT";
const char* password = "w3R1644#";
// Три предустановленные мелодии
const int MELODY_1_SIZE = 35;
const int melody1[MELODY_1_SIZE] = { /* ... */ };
const int MELODY_2_SIZE = 8;
const int melody2[MELODY_2_SIZE] = { /* ... */ };
const int MELODY_3_SIZE = 9;
const int melody3[MELODY_3_SIZE] = { /* ... */ };
// Массивы указателей на мелодии и их длительности
const int* melodies[MELODIES_COUNT] = {melody1, melody2, melody3};
```

```
    const int melodySizes[MELODIES_COUNT] =
{MELODY_1_SIZE, MELODY_2_SIZE, MELODY_3_SIZE};
    const int noteDurations[MELODIES_COUNT] = {180,
200, 250};
    int currentMelodyIndex = 0;  // Индекс текущей
активной мелодии
```

- В коде определены три мелодии: оригинальная, фрагмент из «В лесу родилась ёлочка» и фрагмент из «Имперского марша».
- Каждая мелодия имеет свой размер (количество нот) и длительность ноты.
- Используются массивы указателей и длин для удобного переключения между мелодиями.

3. Реализация функций воспроизведения и управления через веб-интерфейс.

```
void playMelody()
```

Данная функция проигрывает текущую мелодию, используя функцию tone() для генерации звуковых частот на указанном пине. После каждой ноты добавляется пауза для разделения звуков.

Обработчики веб-запросов:

```
handleRingPost()
```

- Отправляет JSON-ответ с информацией о длительности мелодии и её индексе.
- Вызывает функцию воспроизведения после отправки ответа.

```
handleNextMelody()
```

- Переключает устройство на следующую мелодию.
- Отправляет JSON-ответ с информацией о новом номере мелодии.

4. Сканирование доступных сетей Wi-Fi.

```
void scanNetworks()
```

Функция сканирует доступные Wi-Fi сети и выводит информацию об SSID и уровне сигнала (RSSI) в последовательный порт для диагностики.

5. Инициализация устройства в функции setup()

- Инициализируется последовательный порт и вывод на динамик.
- Выполняется сканирование Wi-Fi сетей.
- Подключение к заданной точке доступа.
- Запуск веб-сервера и регистрация обработчиков эндпоинтов /ring и /nextMelody.

6. Основной цикл работы (loop)

```
void loop() {
  server.handleClient();  // Обработка входящих HTTP-запросов
}
```

Основной цикл программы отвечает за обслуживание клиентских HTTP-запросов через встроенный веб-сервер.

При запуске происходит сканирование всех ближайших WIFI сетей и попытка подключиться к указанной в ssid (Рисунок 12).

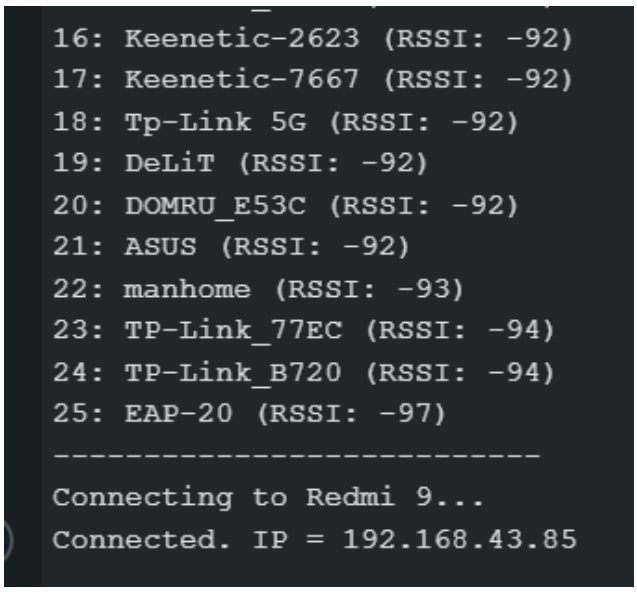


Рисунок 12 - Сканирование WIFI сетей на ESP32

3.2 Код бэкенда

Код реализует промежуточный сервер-адаптер на базе фреймворка Flask (Python), который служит посредником между внешними клиентами и устройством ESP32, управляя воспроизведением мелодий через HTTP-запросы. Сервер принимает команды от пользователей, перенаправляет их на ESP32 и возвращает ответ в формате JSON.

1. Подключение необходимых библиотек

```
import logging              # Для логирования событий
from flask import Flask, jsonify, request  # Для создания веб-сервера и обработки HTTP-запросов
import requests             # Для отправки HTTP-запросов ESP32
```

- logging используется для записи информации о состоянии и возможных ошибках в консоль.
- Flask предоставляет веб-фреймворк для обработки запросов и маршрутов.

– requests позволяет взаимодействовать с удалённым ESP32, отправляя ему POST-запросы.

– jsonify и request используются для работы с JSON-данными при обработке запросов.

2. Настройка логгирования и URL устройства ESP32

```
logging.basicConfig(level=logging.INFO,
format='%(asctime)s %(levelname)s %(message)s')
log = logging.getLogger()

ESP32_URL = "http://host.docker.internal:8089"
```

– Уровень логгирования установлен как INFO, выводится дата, уровень сообщения и текст.

– ESP32_URL — адрес, по которому доступно устройство ESP32 внутри Docker-сети (host.docker.internal).

3. Обработка эндпоинтов API

Эндпоинт /api/ring и /ring

```
@app.route("/api/ring", methods=["POST"])
@app.route("/ring",     methods=["POST"])
def trigger_ring()
```

Принимает POST-запросы и перенаправляет их на ESP32 по адресу /ring.

В случае успеха возвращает JSON:

```
{"status": "ok", "duration": <длительность>,
"melodyIndex": <номер мелодии>}
```

При ошибке возвращаются соответствующие ответы с кодами состояния 500 или 500+ и детализацией ошибки.

Эндпоинт /api/nextMelody и /nextMelody

```
@app.route("/api/nextMelody", methods=["POST"])
@app.route("/nextMelody",     methods=["POST"])
def next_melody()
```

Переключает текущую мелодию на ESP32, отправляя запрос на /nextMelody. При успешной смене возвращает:

```
{"status": "ok", "currentMelody": <номер новой мелодии>}
```

Логируется информация о переключении и возможных ошибках.

4. Запуск веб-сервера.

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8084, debug=False, use_reloader=False, threaded=True)
```

- Сервер запускается на порту 8084.
- host="0.0.0.0" делает его доступным извне контейнера.
- Отключен режим отладки (debug=False) и автоматический перезапуск (use_reloader=False).
- Используется многопоточный режим (threaded=True) для одновременной обработки нескольких запросов.

3.3 Код фронтенда

Данный программный код реализует веб-интерфейс пользователя для управления устройством ESP32, которое воспроизводит мелодии по команде. Интерфейс содержит две кнопки: для вызова звонка и смены мелодии. При нажатии отправляются HTTP-запросы на сервер (через API), а также отображается визуальная обратная связь.

1. Структура и подключение к веб-странице

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="utf-8">
    <title>ESP32 Bell</title>
```

- Определяется базовая структура HTML5-документа.
- Указана кодировка UTF-8 и заголовок страницы — "ESP32 Bell".

2. Стилизация интерфейса (<style>)

```
body {
    font-family: sans-serif;
    background: #f2f2f2;
    text-align: center;
    padding-top: 100px;
}
.box {
    background: white;
    padding: 40px;
    border-radius: 10px;
    box-shadow: 0 0 20px rgba(0,0,0,0.1);
    display: inline-block;
}
button {
    font-size: 1.5em;
    padding: 20px;
    background: #28a745;
    color: white;
    border: none;
    border-radius: 10px;
    cursor: pointer;
}
button:hover {
```

```
        background: #218838;
    }
    #status {
        margin-top: 20px;
        font-size: 1.2em;
        color: #555;
    }
```

- Внешний вид страницы оформлен с акцентом на простоту и удобство использования.
- Контент центрирован, окружён фоном и тенью.
- Кнопки имеют стиль Material-дизайна с эффектом при наведении.
- Блок #status отображает текущее состояние устройства (звонок, ошибка и т.д.).

3. Основной интерфейс (<body>)

```
    <div class="box">
        <h1>ESP32 Bell</h1>
        <img id="bellImage" src="/images/img-1.png"
alt="Bell">
        <br>
        <div class="buttons-container">
            <button onclick="ring()">Позвонить</button>
            <button onClick="changeMelody()">Сменить
мелодию</button>
        </div>
        <p id="status"> </p>
    </div>
```

- Основной контент расположен в центральной панели .box.
- Отображается изображение колокольчика (по умолчанию img-1.png).

- Две кнопки:
  - "Позвонить" — запускает функцию ring().
  - "Сменить мелодию" — запускает функцию changeMelody().

4. Логика работы (<script>)

Функция ring()

```
function ring()
```

- Отправляет POST-запрос на /api/ring.
- Меняет изображение на img-2.png во время звонка.
- Отображает статус выполнения:
  - После успешного ответа — ждет data.duration мс и возвращает исходное изображение и сообщение.
  - При ошибке выводится соответствующее уведомление, изображение возвращается.

Функция changeMelody()

```
function changeMelody()
```

- Отправляет POST-запрос на /api/nextMelody.
- При успехе показывает номер новой мелодии в виде alert.
- В случае ошибки записывает её в консоль.

5. Использование изображений

- Изображения хранятся в каталоге /images/:
  - img-1.png — основное изображение (состояние покоя).
  - img-2.png — изображение при активном звонке.

Также очень полезны команды для развертывания контейнеров на Docker и проброс портов (на Windows):

Команда для сборки:

```
docker-compose up --build --force-recreate --no-deps --remove-orphans -d
```

Команда для проброса порта:

```
netsh interface portproxy add v4tov4 listenport=8089 listenaddress=0.0.0.0 connectport=80 connectaddress=192.168.43.85
```

Заключение

В заключение, хочу отметить, что в ходе выполнения курсовой работы была разработана и реализована система музыкального звонка на базе микроконтроллера ESP32 с возможностью удалённого управления воспроизведением мелодий через веб-интерфейс. В процессе работы были изучены современные аппаратные и программные решения для построения подобных устройств, выбраны оптимальные компоненты — микроконтроллер ESP32 и пьезоизлучатель, а также реализованы все необходимые схемотехнические и программные решения.

В результате проектирования и сборки устройства удалось обеспечить стабильное воспроизведение нескольких предустановленных мелодий, переключение между ними и запуск звонка по команде от пользователя. На стороне программного обеспечения был реализован REST API для управления звонком и смены текущей мелодии. Бэкенд на Python позволил организовать взаимодействие между пользовательским интерфейсом и ESP32, обеспечив гибкость интеграции и обработку запросов в реальном времени. Фронтенд в виде простого веб-приложения предоставляет удобный графический интерфейс с визуальной обратной связью при каждом действии пользователя.

Таким образом, поставленные цели и задачи были успешно достигнуты. Разработанная система может быть использована как автономное устройство в системах умного дома, офиса или учебного проекта, а также служить основой для дальнейшего развития — например, добавления голосового управления, интеграции с мобильными приложениями или облачными IoT-платформами. Полученные в ходе работы знания и практические навыки могут быть применены при разработке других сетевых устройств и систем, ориентированных на взаимодействие с пользователем через веб-интерфейс.

Список использованных источников

1 Smart Musical Bell That Identifies The Person [Электронный ресурс]. – Режим доступа: https://www.electronicsforu.com/electronics-projects/smart-musical-bell-identifies-person. – (Дата обращения: 23.04.2025);

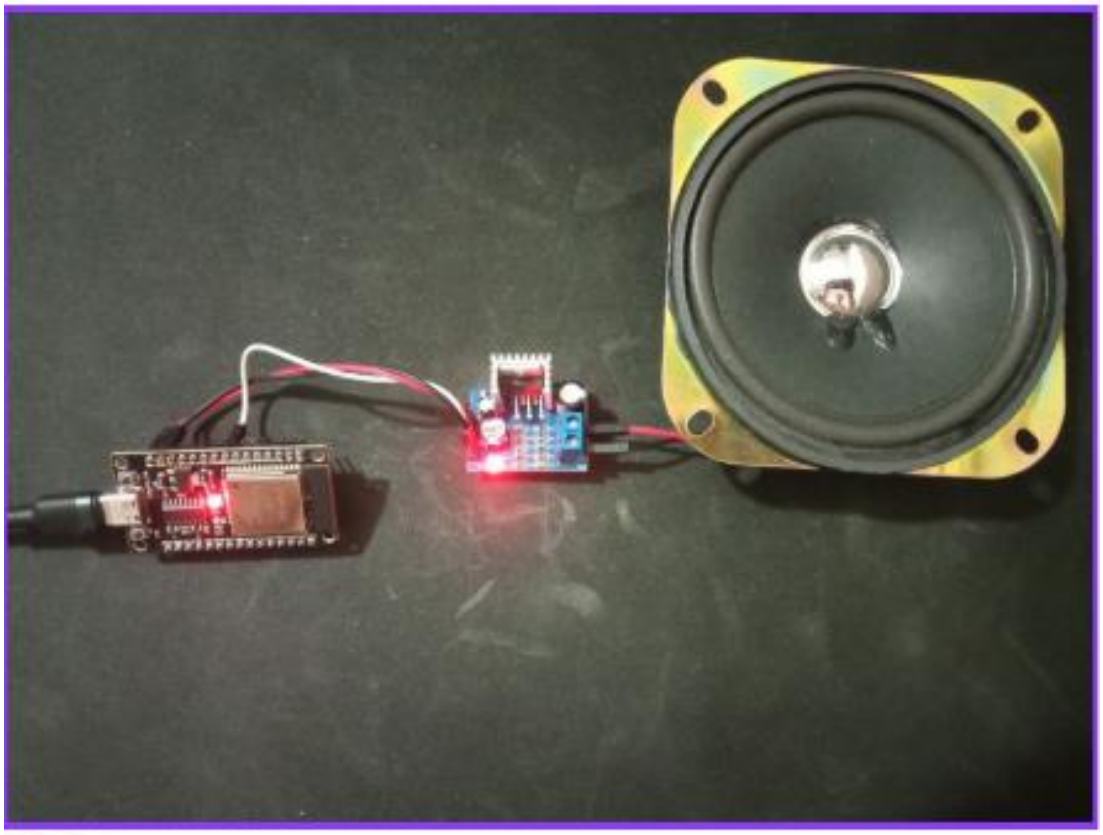
2 DIY ESP32 Based Audio Player [Электронный ресурс]. – Режим доступа: https://circuitdigest.com/microcontroller-projects/esp32-based-audio-player. – (Дата обращения: 10.05.2025);

Приложение А

Ссылка на публичный репозиторий

https://github.com/paincake00/music_bell

Приложение В



Итоговая DIY-сборка