МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Факультет компьютерных наук

Кафедра информационных систем

Блок управления вентилятором компьютера

Направление 09.03.02 Информационные системы и технологии

Информационные системы и сетевые технологии

Зав. кафедрой _____ *Д.Н. Борисов, к.т.н., доцент*

Студент _____ *А.А. Богачев-Воевудский, 3 курс, д/о*

Руководитель _____ *А.В. Максимов, ст. преподаватель*

Воронеж 2025

Содержание

Введение

В современных компьютерных системах эффективное управление температурным режимом играет ключевую роль в обеспечении стабильности и долговечности оборудования. Блок управления вентиляторами является важной частью этой системы, позволяя автоматически регулировать скорость вращения охлаждающих вентиляторов в зависимости от текущей нагрузки и температуры компонентов. В рамках данного проекта разрабатывается программно-аппаратное решение на основе микроконтроллера ESP32, обеспечивающее точное управление скоростью вентилятора с помощью ШИМ (широтно-импульсной модуляции), а также возможность удалённого контроля и мониторинга через веб-интерфейс. Это позволяет создать гибкую и удобную систему охлаждения, применимую как в персональных компьютерах, так и в промышленных устройствах.

1 Описание устройства

1.1 Определение

Разрабатываемое устройство является контроллером для управление вентилятором компьютера. Пользователь подключается с телефона через Home Assistant и управляет состоянием вентилятора.

1.2 Комплектующие

Устройство состоит из следующих комплектующих:

- Разъем питания DC 2.5 x 5.5
- ESP32 NODEMCU 38pin DEVKIT V1 WIFI + Bluetooth CP2102
- Беспаечная макетная плата
- DC-DC step-down понижающий преобразователь с поддержкой QC3.0
- Переходник USB OTG micro USB Defender, адаптер для передачи данных с телефона
- Блок питания (адаптер) T120100-2C1 12V 1A 5.5 x 2.1

2 Сборка, программирование и демонстрация работы

2.1 Сборка устройства

Для сборки устройства нужно подготовить рабочую область и выложить все комплектующие. Сборка будет происходить по схеме, которая находится в Приложении А. На Рисунке 1 все комплектующие:

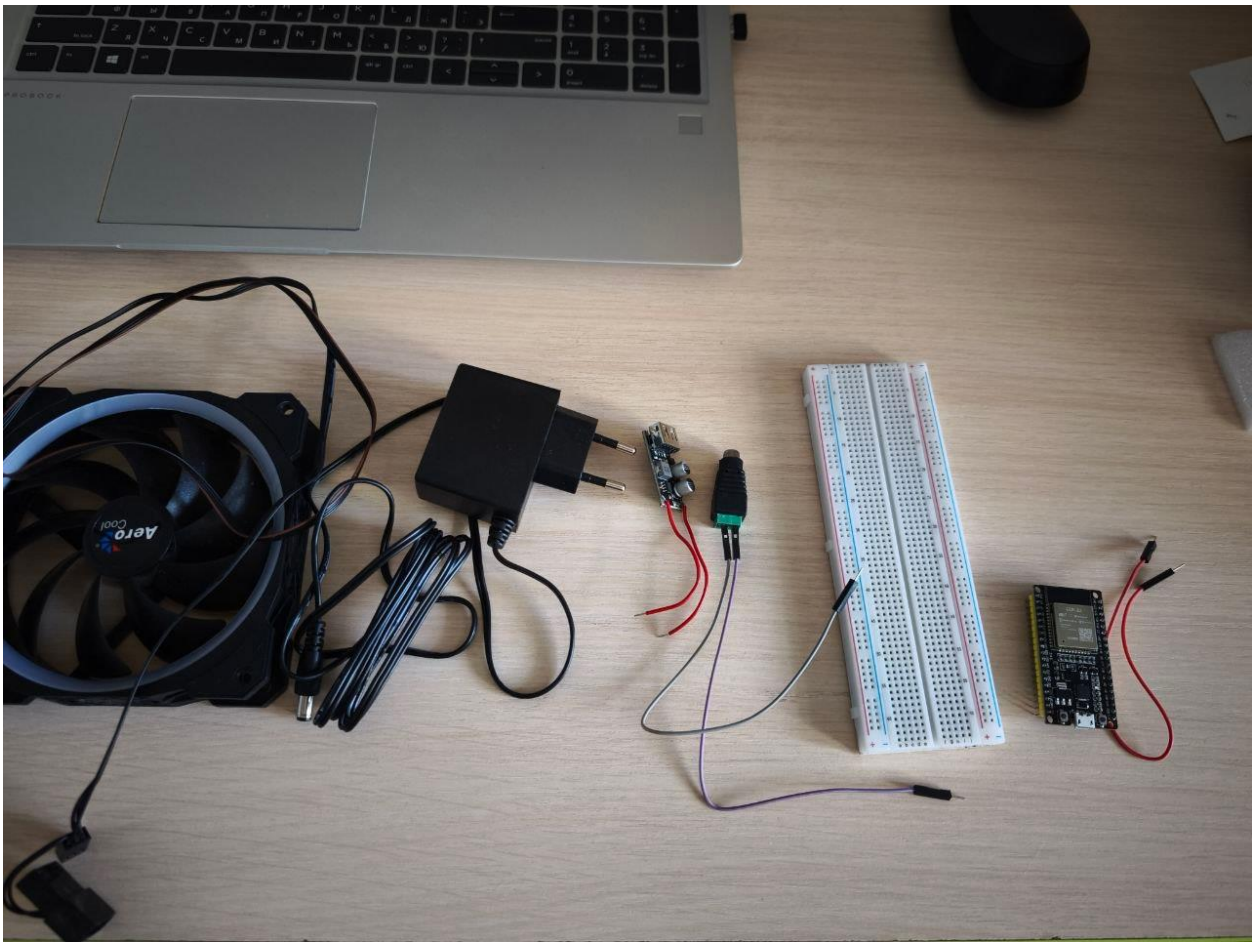


Рисунок 1 - Подготовка комплектующих

Теперь нужно всё правильно подключить. Сперва нужно вставить esp32 на макетную плату:

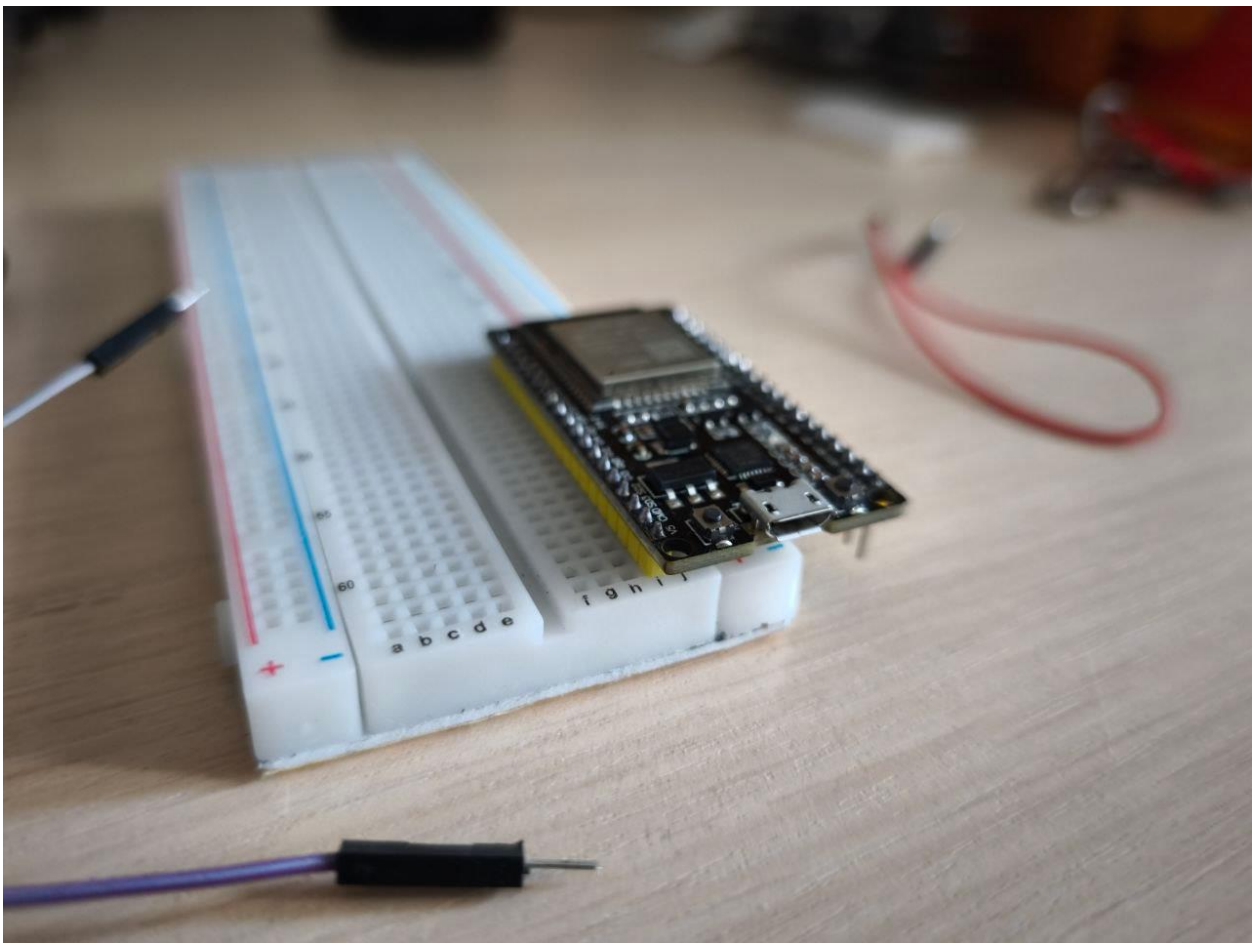


Рисунок 2 - Esp32 на макетной плате

Далее идет подключение вентилятора к питанию и проверка его вращения:

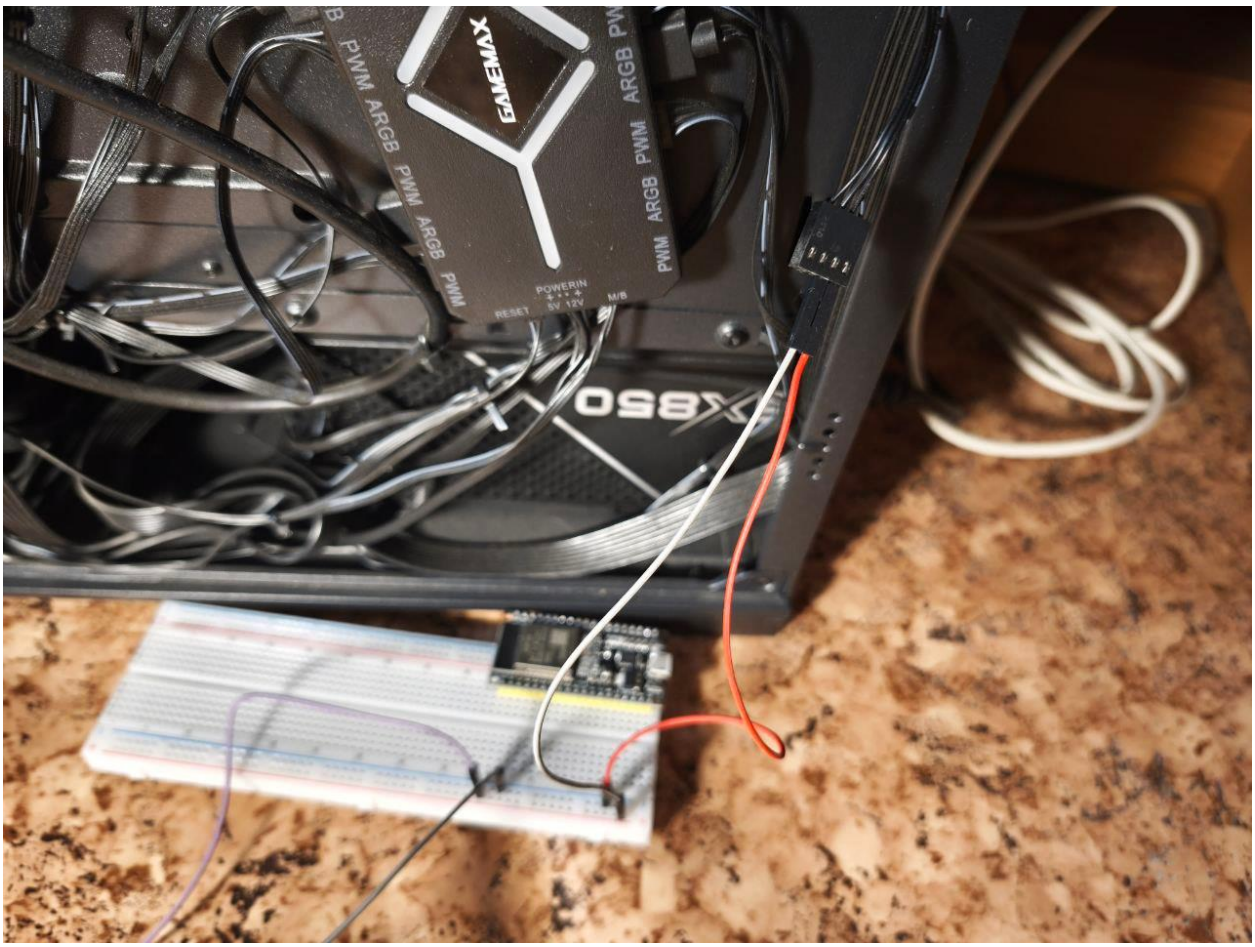


Рисунок 3 - Подключение питания к вентилятору

Вентилятор работает, питание есть:

Рисунок 4 - Вентилятор с питанием

Подключим остальные пины вентилятора к esp32 – RPM и PWM:

Рисунок 5 - Подключение вентилятора к esp32

Остается подключить только usb переходник для автономного питания:

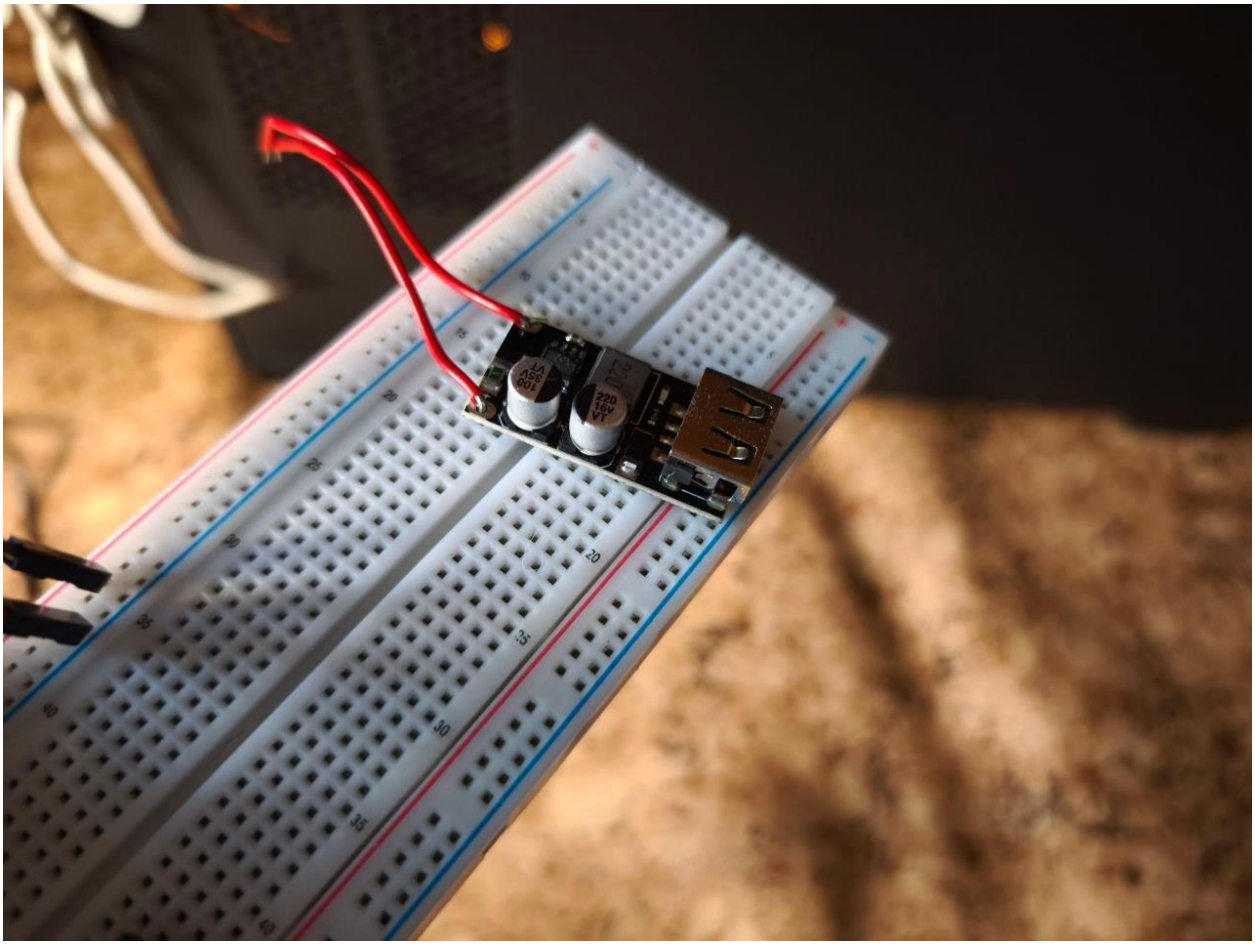


Рисунок 6 - Usb переходник

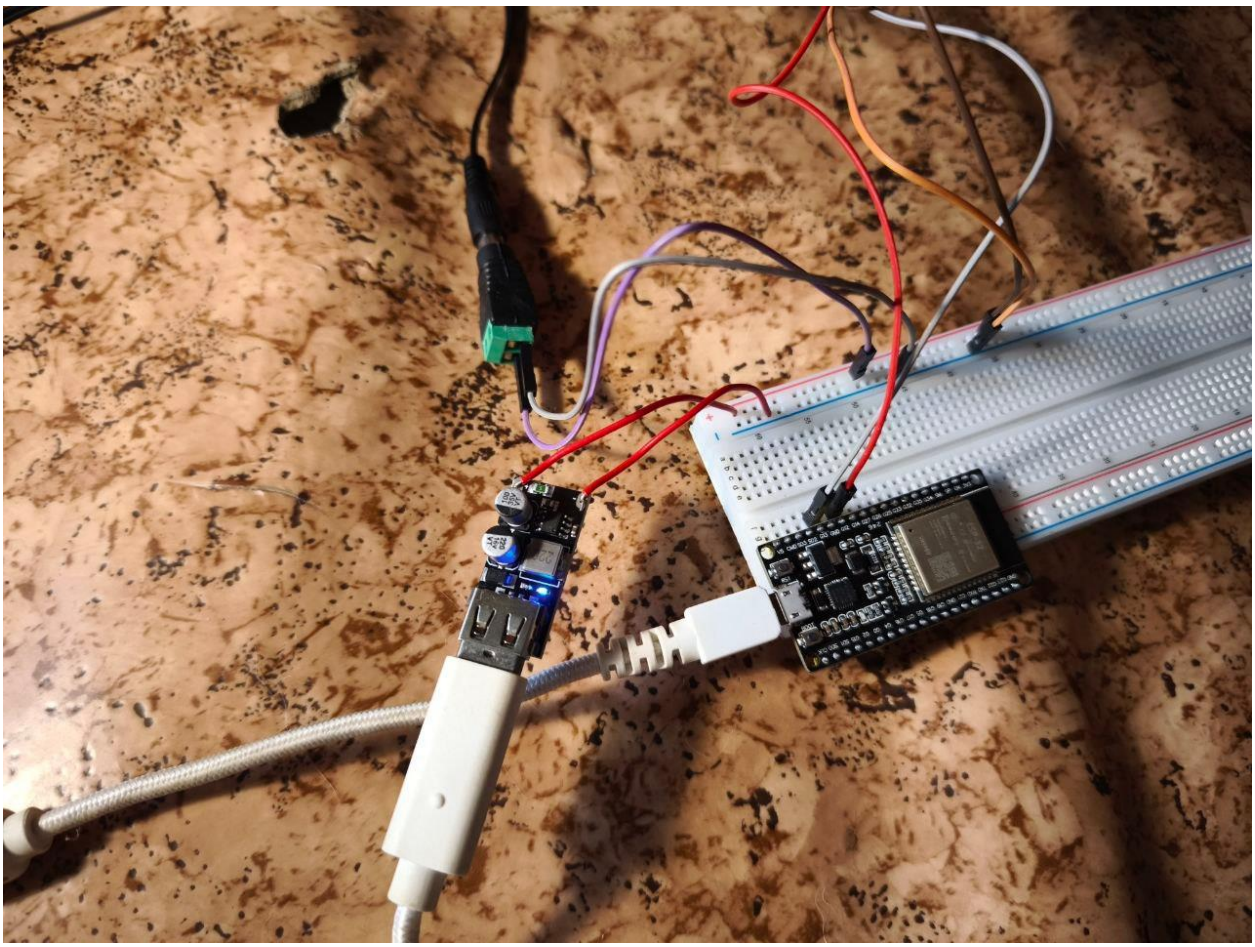


Рисунок 7 - Подключение с usb переходником

Устройство собрано, теперь осталось его запрограммировать.

2.2 Программирование устройства

Написание кода будет производится в Arduino IDE. Для начала подключим необходимые библиотеки:

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <HTTPClient.h>
```

- WiFi.h — для подключения к Wi-Fi.
- ESPAsyncWebServer.h — для запуска веб-сервера на ESP32.
- HTTPClient.h — для отправки HTTP-запросов (например, передача IP на сервер).

```
const char* ssid = "Bogachevich";
const char* password = "DNEAT78R";
#define FAN_PWM_PIN 12
#define FAN_RPM_PIN 13
```

Настройка Wifi подключения и подключений к esp32

```
volatile byte fanRPMCount = 0;
volatile unsigned long last_interrupt_time = 0;
unsigned long lastRPMSample = 0;
int fanSpeed = 128;
const int pulsesPerRevolution = 2;
```

Для подсчёта оборотов вентилятора через прерывания.

```
AsyncWebServer server(80);
```

Запуск веб-сервера на порту 80

```
void IRAM_ATTR rpmISR() {
  unsigned long interrupt_time = micros();
  if (interrupt_time - last_interrupt_time > 1000) {
    fanRPMCount++;
    last_interrupt_time = interrupt_time;
  }
}
```

Обработка прерывания для RPM

```
void setFanSpeed(int speed) {
  ledcWrite(0, speed);
```

```
}
```

Настройка скорости вентилятора.

```
void sendIPToBackend() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    String url =
"http://192.168.0.105:5000/receive_ip";  // ← Укажи IP
своего ПК
    http.begin(url);
    http.addHeader("Content-Type", "application/json");

    String jsonData = "{\"ip\":\"";
    jsonData += WiFi.localIP().toString();
    jsonData += "\"}";

    int httpResponseCode = http.POST(jsonData);

    if (httpResponseCode > 0) {
      Serial.printf("HTTP Response code: %d\n",
httpResponseCode);
      String response = http.getString();
      Serial.println("Response: " + response);
    } else {
      Serial.print("Error sending POST: ");
```

```
      Serial.println(http.errorToString(httpResponseCod
e).c_str());
    }

    http.end();
  } else {
    Serial.println("WiFi not connected");
  }
}
```

Отправка IP на Flask-бэкенд

```
void setup() {
  Serial.begin(115200);
  pinMode(FAN_PWM_PIN, OUTPUT);
  ledcSetup(0, 25000, 8);
  ledcAttachPin(FAN_PWM_PIN, 0);
  setFanSpeed(fanSpeed);
```

Настройка вентилятора

```
  pinMode(FAN_RPM_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(FAN_RPM_PIN),
rpmISR, FALLING);
```

Настройка RPM

```
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(1000);
    Serial.print(".");
  }
```

Подключение к Wi-Fi

```
  Serial.println("\nConnected to WiFi");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  sendIPToBackend();
  // ==== HTTP GET: /setSpeed?speed=... ====
  server.on("/setSpeed", HTTP_GET,
[](AsyncWebServerRequest *request){
    if (request->hasParam("speed")) {
      int newSpeed = request->getParam("speed")-
>value().toInt();
      if (newSpeed >= 0 && newSpeed <= 255) {
        fanSpeed = newSpeed;
        setFanSpeed(fanSpeed);
        Serial.printf("[INFO] Установлена скорость:
%d\n", newSpeed);
        request->send(200, "text/plain", "OK");
      } else {
        request->send(400, "text/plain", "Недопустимое
значение скорости (0–255)");
```

```
      }
    } else {
      request->send(400, "text/plain", "Не передан
параметр speed");
    }
  });
```

Отправка IP на Flask

```
  server.begin();
}
void loop() {
  if (millis() - lastRPMSample >= 1000) {
    int rpm = (fanRPMCount * 60) / pulsesPerRevolution;
    fanRPMCount = 0;
    lastRPMSample = millis();
    Serial.printf("RPM: %d\n", rpm);
  }
}
```

Вывод RPM каждую секунду

Теперь устройство настроено, вот вывод в serial monitor:

```
Connected to WiFi
IP address: 192.168.0.100
HTTP Response code: 200
Response: {
  "ip": "192.168.0.100",
  "status": "success"
}

RPM: 1800
RPM: 1230
RPM: 1350
RPM: 1650
RPM: 1020
RPM: 1290
RPM: 1590
RPM: 1650
RPM: 960
RPM: 2040
RPM: 990
RPM: 1800
RPM: 840
[INFO] Установлена скорость: 0
RPM: 690
RPM: 0
```

Рисунок 8 - Вывод в serial monitor

А так выглядит отправка запросов на веб-сервер:

```
[DEBUG] Запрос к ESP32: http://192.168.0.100/setSpeed?speed=0
127.0.0.1 - - [07/Jun/2025 10:37:12] "GET /update?speed=0 HTTP/1.1" 200 -
[DEBUG] Запрос к ESP32: http://192.168.0.100/setSpeed?speed=255
127.0.0.1 - - [07/Jun/2025 10:37:54] "GET /update?speed=255 HTTP/1.1" 200 -
[DEBUG] Запрос к ESP32: http://192.168.0.100/setSpeed?speed=0
127.0.0.1 - - [07/Jun/2025 10:38:01] "GET /update?speed=0 HTTP/1.1" 200 -
[DEBUG] Запрос к ESP32: http://192.168.0.100/setSpeed?speed=255
127.0.0.1 - - [07/Jun/2025 10:38:30] "GET /update?speed=255 HTTP/1.1" 200 -
[INFO] Получен IP от ESP32: http://192.168.0.100
192.168.0.100 - - [07/Jun/2025 10:39:02] "POST /receive_ip HTTP/1.1" 200 -
[DEBUG] Запрос к ESP32: http://192.168.0.100/setSpeed?speed=0
127.0.0.1 - - [07/Jun/2025 10:39:15] "GET /update?speed=0 HTTP/1.1" 200 -
```

Рисунок 9 - Отправка запросов от веб-сервера к esp32

Код для веб-сервера:

```
from flask import Flask, render_template_string,
request, jsonify
import requests

app = Flask(__name__)

# Глобально храним IP ESP32 (устанавливается через
/receive_ip)
esp32_ip = "http://192.168.0.100"  # для теста можно
оставить фиксированным

# HTML-шаблон страницы управления с тёмной темой
HTML_TEMPLATE = """
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Управление вентилятором ESP32</title>
    <style>
        :root {
            --bg-color: #121212;
            --text-color: #ffffff;
            --accent-color: #00d4ff;
            --slider-bg: #333;
            --slider-thumb: #00d4ff;
        }

        body {
            margin: 0;
            padding: 0;
```

```
            background-color: var(--bg-color);
            color: var(--text-color);
            font-family: 'Segoe UI', Tahoma, Geneva,
Verdana, sans-serif;
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: start;
            min-height: 100vh;
            padding-top: 50px;
        }

        h1 {
            color: var(--accent-color);
            margin-bottom: 10px;
        }

        .ip {
            font-size: 16px;
            color: #aaa;
            margin-bottom: 30px;
        }

        .slider-container {
            width: 80%;
            max-width: 500px;
            margin: 0 auto;
        }

        input[type=range] {
```

```
        -webkit-appearance: none;
        width: 100%;
        height: 10px;
        border-radius: 5px;
        background: var(--slider-bg);
        outline: none;
        margin: 20px 0;
    }

    input[type=range]::-webkit-slider-thumb {
        -webkit-appearance: none;
        appearance: none;
        width: 24px;
        height: 24px;
        border-radius: 50%;
        background: var(--slider-thumb);
        cursor: pointer;
        box-shadow: 0 0 8px rgba(0, 212, 255, 0.7);
    }

    #value {
        font-size: 28px;
        color: var(--accent-color);
        margin-top: 10px;
        transition: color 0.3s ease;
    }

    footer {
        margin-top: auto;
        padding: 20px;
```

```
            color: #666;
            font-size: 14px;
        }
    </style>
</head>
<body>
    <h1>🌀 Управление вентилятором ESP32</h1>
    <div class="ip">
        {% if esp32_ip %}
            🔑 IP ESP32: {{ esp32_ip }}
        {% else %}
            ❌ ESP32 не подключён
        {% endif %}
    </div>

    <div class="slider-container">
        <input type="range" min="0" max="255" value="0"
id="speedSlider">
        <div id="value">Скорость: 0</div>
    </div>

    <script>
        const slider =
document.getElementById("speedSlider");
        const valueDisplay =
document.getElementById("value");

        slider.oninput = function() {
            let speed = this.value;
            valueDisplay.textContent = "Скорость: " +
```

```
speed;

            fetch('/update?speed=' + speed)
                .catch(err => console.error('Ошибка
запроса:', err));
        };
    </script>
</body>
</html>
"""

@app.route("/")
def index():
    return render_template_string(HTML_TEMPLATE,
esp32_ip=esp32_ip)

@app.route("/update")
def update_speed():
    global esp32_ip
    speed = request.args.get("speed", default=None)

    if not esp32_ip:
        return "ESP32 не подключён", 400

    if speed is not None and speed.isdigit():
        speed_val = int(speed)
        if 0 <= speed_val <= 255:
            try:
                url =
f"{esp32_ip}/setSpeed?speed={speed_val}"
```

```
                print(f"[DEBUG] Запрос к ESP32: {url}")
                response = requests.get(url, timeout=2)
                return response.text,
response.status_code
            except Exception as e:
                print(f"[ERROR] Ошибка при подключении
к ESP32: {e}")
                return f"Ошибка подключения к ESP32:
{e}", 500
    return "Неверное значение скорости", 400

@app.route("/receive_ip", methods=["POST"])
def receive_ip():
    global esp32_ip
    data = request.get_json()
    ip = data.get("ip")
    if ip:
        esp32_ip = f"http://{ip}"
        print(f"[INFO] Получен IP от ESP32:
{esp32_ip}")
        return jsonify({"status": "success", "ip":
ip}), 200
    else:
        return jsonify({"status": "error", "message":
"IP не передан"}), 400

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

Внешний вид страницы веб-сервера:

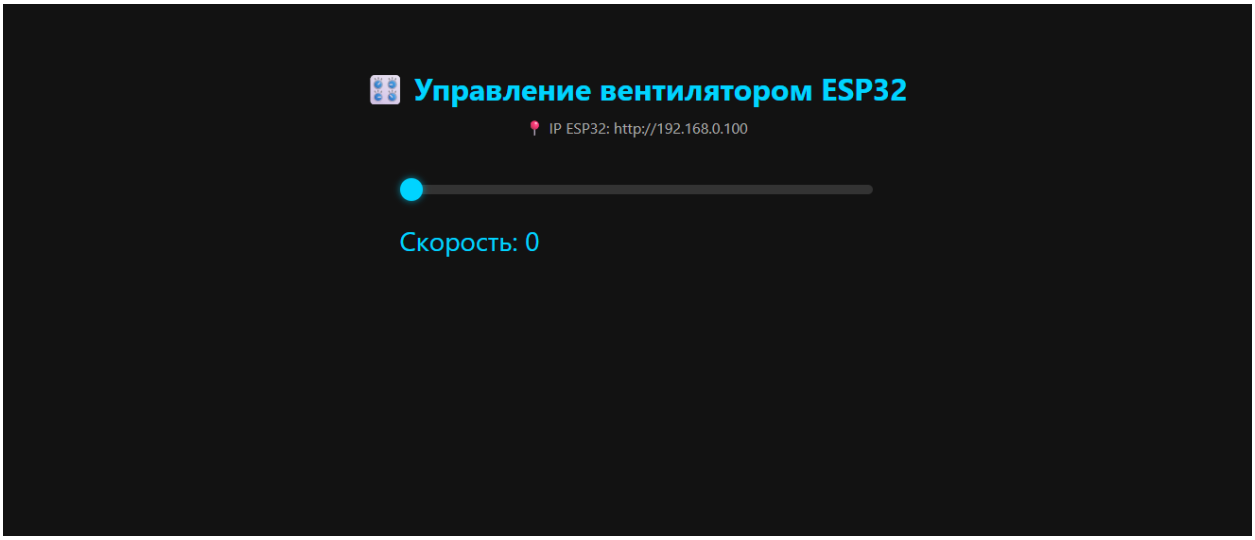


Рисунок 10 - Страница веб-сервера

Список литературы

1. YouTube. Secrets of Arduino PWM [Электронный ресурс] // Видео на YouTube. – Режим доступа: https://www.youtube.com/watch?v=9-AZF6udg-Q&t=307s (дата обращения: 26.05.2025).
2. Arduino Documentation. Secrets of Arduino PWM [Электронный ресурс] // Официальный сайт Arduino. – 2024. – Режим доступа: https://docs.arduino.cc/tutorials/generic/secrets-of-arduino-pwm/ (дата обращения: 26.05.2025).
3. Arduino Forum. Controlling PWM Fan w/PWM Pin from ESP32 [Электронный ресурс] // Форум Arduino. – 2023. – Режим доступа: https://forum.arduino.cc/t/controlling-pwm-fan-w-pwm-pin-from-esp32/1345009 (дата обращения: 26.05.2025).
4. ESP32IO.COM. ESP32 Controls Fan [Электронный ресурс] // ESP32 Tutorials. – 2021. – Режим доступа: https://esp32io.com/tutorials/esp32-controls-fan (дата обращения: 26.05.2025).

Приложение А

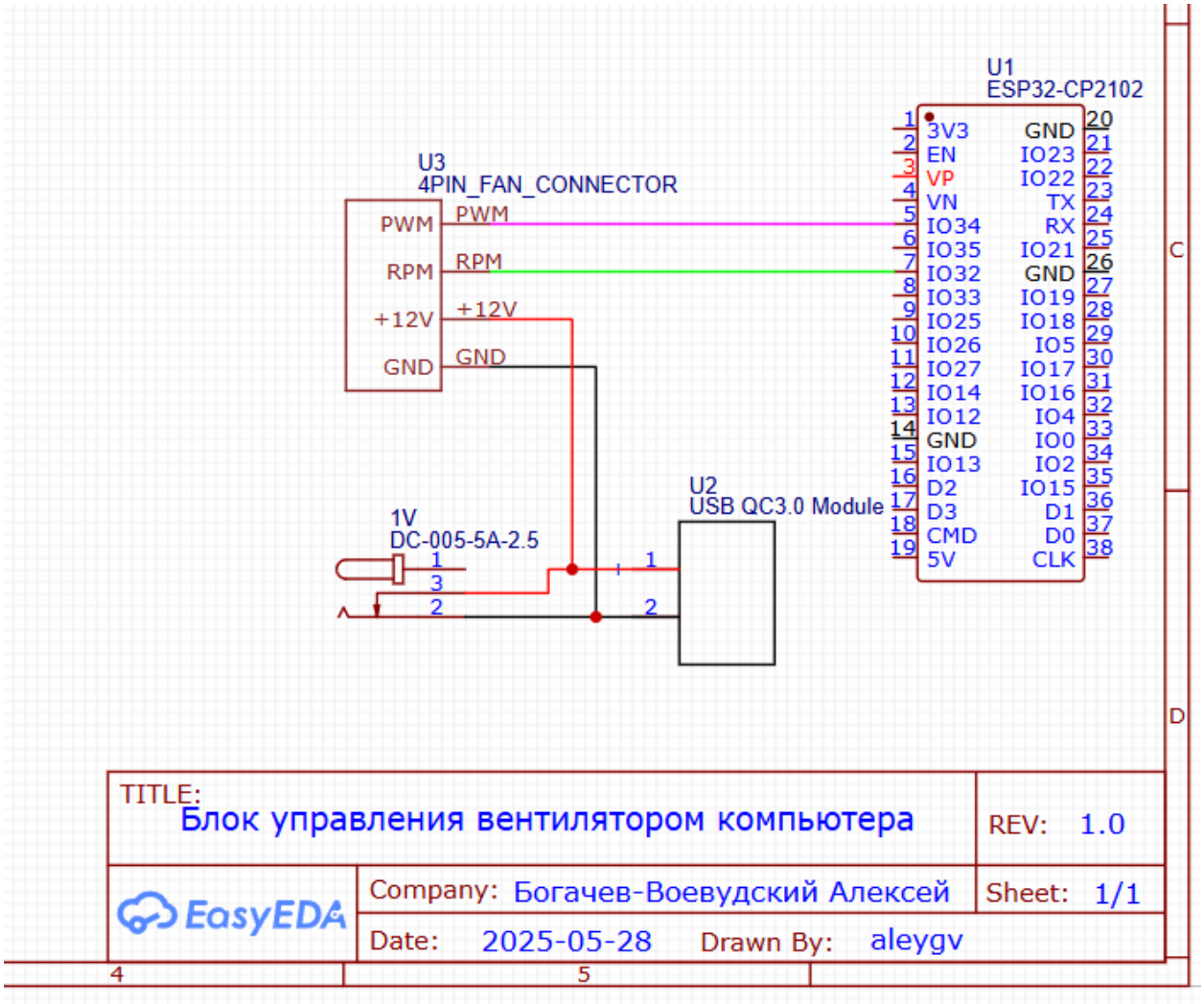
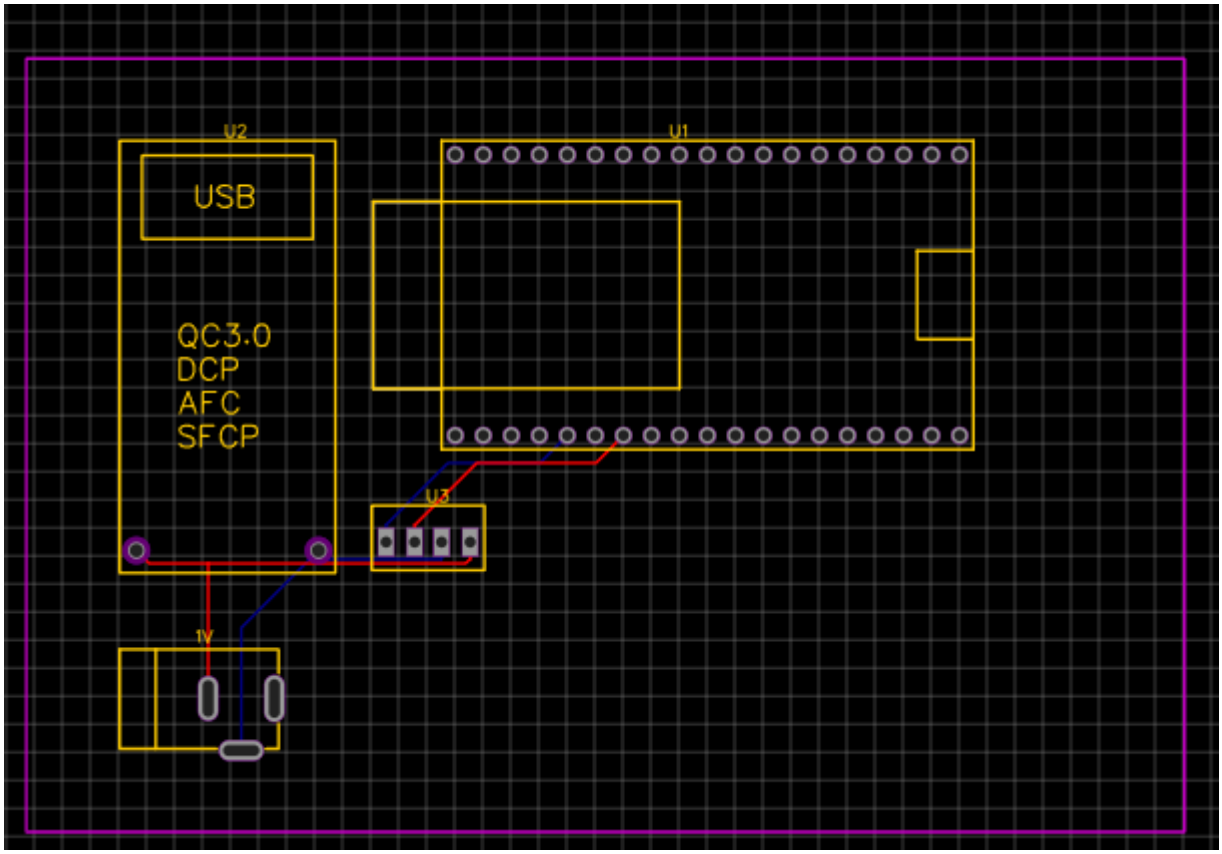


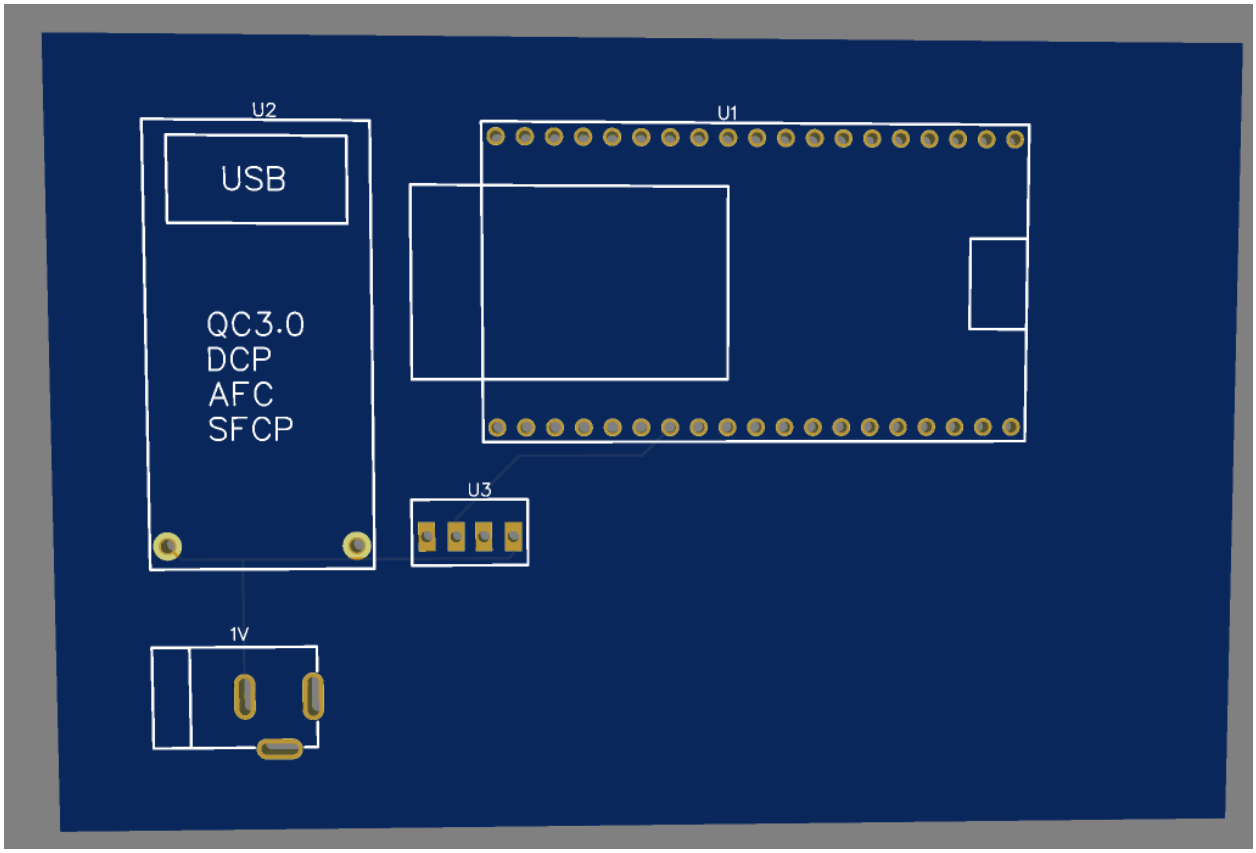
Схема устройства

Приложение В



Разводка на плате

Приложение С



3D модель устройства