



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Компьютерные системы и сети»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

K KУРСОВОЙ РАБОТЕ

на тему:

*«Система мониторинга состояния комнатного
растения»*

Студент ИУ6-74Б
(Группа)

(Подпись, дата)

Погиба И. О.
(И. О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

Хохлов С. А.
(И. О. Фамилия)

2024 г.

РЕФЕРАТ

МИКРОКОНТРОЛЛЕР, ESP-32, DHT-11, YL-69, HCSR-04, ТЕМТ6000, MQTT, PROMETHEUS, GRAFANA

Объектом разработки данной курсовой работы является система мониторинга состояния за комнатным растением.

Цель работы – закрепление знаний, полученных при изучении дисциплины «Микропроцессорные системы», в процессе самостоятельной работы при проектировании устройства для контроля работы электроприборов; развитие навыков и умений применять теоретические знания на практике при выполнении учебных проектов, а также по заказам промышленности и в порядке личной инициативы; освоение новых технологий проектирования при выполнении проектных работ.

В процессе выполнения курсового проекта были решены следующие задачи: анализ задания, выбор схемотехнического решения и элементов системы, анализ и выбор радиоэлементов схемы, расчет потребляемой мощности устройства, разработка алгоритмов управления и соответствующей программы микроконтроллера.

В результате было спроектировано требуемое устройства и получена сопутствующая документация, а именно: функциональная и принципиальная схемы, схемы алгоритмов управления и соответствующая программа микроконтроллера. Репозиторий со всеми исходными файлами находится по ссылке, доступ к метрикам растения по ссылке.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

МК — микроконтроллер

ESP32 — используемый микроконтроллер

DHT-11 — датчик температуры и влажности почвы

YL-69 — датчик влажности почвы

HCSR-04 — ультразвуковой датчик, вычисляющий расстояние до объекта

ТЕМТ6000 — датчик интенсивности света

MQTT — легковесный протокол обмена сообщениями для публикации/подписки сообщений

Алерт — механизм, который позволяет отслеживать состояние системы и уведомлять о проблемах

PROMETHEUS — база данных временных рядов, которая поддерживает сбор данных из различных источников посредством экспортёров

GRAFANA — система визуализации и анализа информации, которая позволяет работать с широким спектром источников данных

ПЭВМ — персональная электронно-вычислительная машина

СОДЕРЖАНИЕ

РЕФЕРАТ	2
ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	3
ВВЕДЕНИЕ	6
1 Конструкторская часть	8
1.1 Анализ требований	8
1.2 Архитектура проекта	9
1.3 Дифференцированная оценка состояния растения	9
1.4 Проектирование функциональной схемы	11
1.4.1 Организация блока Wi-Fi модуля в ESP32	11
1.4.2 Распределение портов	13
1.4.3 Описание датчика влажности и температуры воздуха DHT-11	15
1.4.4 Описание ультразвукового датчика HC-SR04	16
1.4.5 Описание датчика влажности почвы YL-69	17
1.4.6 Описание датчика TEMT6000	18
1.4.7 Функциональная схема	19
1.5 Проектирование принципиальной схемы	20
1.6 Описание алгоритмов основных программных процедур	24
1.6.1 Главная процедура программы	24
2 Технологическая часть	25
2.1 Моделирование и отладка системы на макетной плате	25
2.2 Разработка серверной части системы мониторинга	28
2.2.1 MQTT-брюкер	28
2.2.2 Деплой приложения	29
2.2.3 Prometheus	30
2.2.4 Grafana	30
2.2.5 AlertManager	32
ЗАКЛЮЧЕНИЕ	34

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	36
ПРИЛОЖЕНИЕ А	37
ПРИЛОЖЕНИЕ Б	66

ВВЕДЕНИЕ

Курсовой проект «Система мониторинга состояния комнатного растения» выполнялся на основании учебного плана кафедры ИУ6.

Цель данной курсовой работы – формировании навыков разработки и проектирования микропроцессорных систем путём освоения современных технологий проектирования систем на основе микроконтроллеров, а также программируемых систем на кристалле.

Проектирование контрольно-измерительного устройства состоит из двух основных частей: конструкторская часть и технологическая часть. Конструкторская часть включает в себя:

- описание архитектуры и технические характеристики использованного в проекте микроконтроллера;
- описание разработанной структурной и функциональной схемы микроконтроллерной системы;
- описание принципиальной электрической схемы МК-системы с обоснованием выбора используемых радиоэлементов;
- назначение функциональных элементов микроконтроллерной системы с описанием принципов работы используемых модулей микроконтроллера, а также других системных устройств;
- описание алгоритмов функционирования МК-системы;
- описание алгоритмов функционирования МК-системы;
- расчет потребляемой мощности устройства.

Технологическая часть включает в себя:

- характеристику использованных систем разработки и отладки программ;
- тестирование и отладку программы;
- описание и моделирование работы системы;
- описание способа программирования МК.

По завершении проектирования была выполнена проверка работоспособности схемы и программного обеспечения.

1 Конструкторская часть

1.1 Анализ требований

Согласно техническому заданию, необходимо разработать на основе микроконтроллера ESP32 систему мониторинга за комнатным растением, охватывающую показатели температуры и влажности воздуха, влажности почвы, интенсивность света, рост растения. Необходимо предусмотреть отправку данных о состоянии растения на сервер и отслеживание показателей с использованием временных рядов. Спроектировать и реализовать общую оценку состояния комнатного растения.

Основанием для выполнения данной работы являются:

- учебный план кафедры ИУ6;
- задание на курсовой проект;

В результате анализа требований к микроконтроллерной системе, можно сформулировать перечень блоков, которые необходимы для реализации устройства:

- микроконтроллер;
- программатор;
- датчик расстояния;
- датчик освещённости;
- датчик температуры и влажности;
- датчик влажности почвы;
- датчик освещённости;
- блок связи с ПЭВМ;

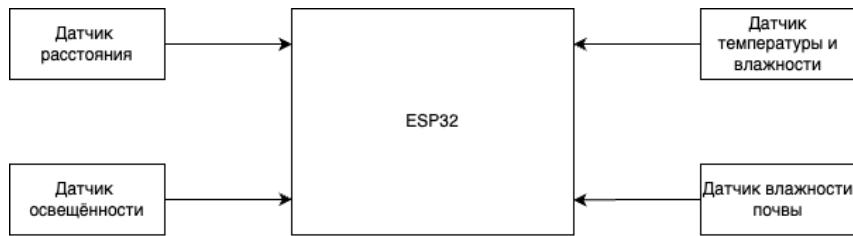


Рисунок 1.1 – Структурная диаграмма

1.2 Архитектура проекта

Алгоритм работы системы мониторинга за комнатным растение (см. рисунок 1.2):

1. микроконтроллер снимает показания с датчиков по соответствующим портам GPIO
2. микроконтроллер с помощью встроенного Wi-Fi модуля отправляет эти метрики в формате JSON по протоколу mqtt [1] на сервер
3. на сервере с помощью утилиты mqtt-exporter [2] метрики экспортируются в prometheus [3]
4. prometheus используется в качестве источника данных в grafana [4] и таким образом метрики появляются на борде

1.3 Дифференцированная оценка состояния растения

Для оценки состояния растения были выбраны следующие, получаемые метрики:

Определение общего состояния комнатного является дифференциированной оценкой, основанная на таких показателях, как:

- температура воздуха
- влажность воздуха
- влажность почвы
- освещённость

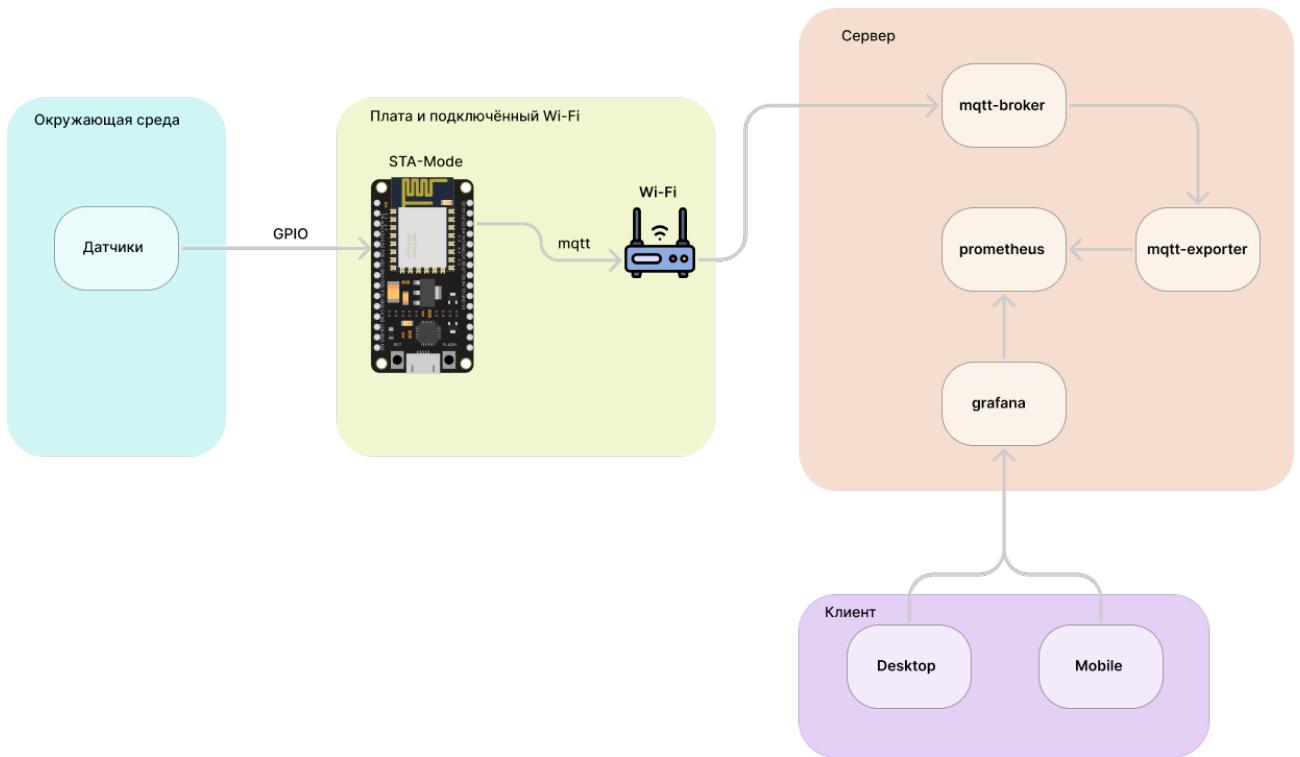


Рисунок 1.2 – Архитектура проекта "Система мониторинга за комнатным растением"

Для оценки состояния растения на основе указанных метрик можно использовать формулу, учитывающую оптимальные диапазоны для каждой метрики.

1. Определение оптимальных диапазонов (оптимум)

- температура воздуха: 20–25°C
- влажность воздуха: 40–60
- влажность почвы: 30–50
- освещённость: 200–800 люкс

2. Нормализация значений. Каждую метрику можно нормализовать в

диапазоне от 0 до 1:

$$\text{Температура} = \frac{\text{Текущая температура} - 20}{25 - 20}$$
$$\text{Влажность воздуха} = \frac{\text{Текущая влажность} - 40}{60 - 40}$$
$$\text{Влажность почвы} = \frac{\text{Текущая влажность почвы} - 30}{50 - 30}$$
$$\text{Освещённость} = \frac{\text{Текущая освещённость} - 200}{800 - 200}$$

3. Формула для оценки состояния. Зададим равные веса для каждой метрики, так как считаем, что метрики равнозначны.

$$\text{Состояние растения} = \frac{\text{Температура} + \text{Влажность воздуха} + \text{Влажность почвы} + \text{Освещённость}}{4} \times 100$$

1.4 Проектирование функциональной схемы

При разработке МК-системы был выбран микроконтроллер ESP32. Функциональная схема микроконтроллера ESP32 приведена на рисунке ??.

ESP32 основан на двухъядерном процессоре с архитектурой Xtensa, который имеет 32 рабочих регистра общего назначения. Каждый из регистров напрямую подключен к АЛУ, что позволяет выполнять операции с двумя регистрами за один такт.

В состав ESP32 входят: 4 МБ встроенной флэш-памяти, 520 кбайт статического ОЗУ(16 кбайт для кэша), 448 кбайт ПЗУ, поддержка внешней памяти через интерфейс SPI, до 34 линий ввода-вывода, два универсальных таймера, аппаратный модуль для работы с Wi-Fi и Bluetooth, несколько интерфейсов для связи (UART, I2C, SPI), а также встроенные функции для работы с прерываниями. Процессор поддерживает различные режимы энергосбережения, что позволяет оптимизировать потребление энергии в зависимости от задач. В программной памяти реализована конвейеризация, что обеспечивает высокую производительность за счет предварительной загрузки инструкций.

1.4.1 Организация блока Wi-Fi модуля в ESP32

Блок Wi-Fi модуля ESP32 обеспечивает беспроводную связь и поддерживает различные режимы работы, включая режим STA (Station) и AP(access

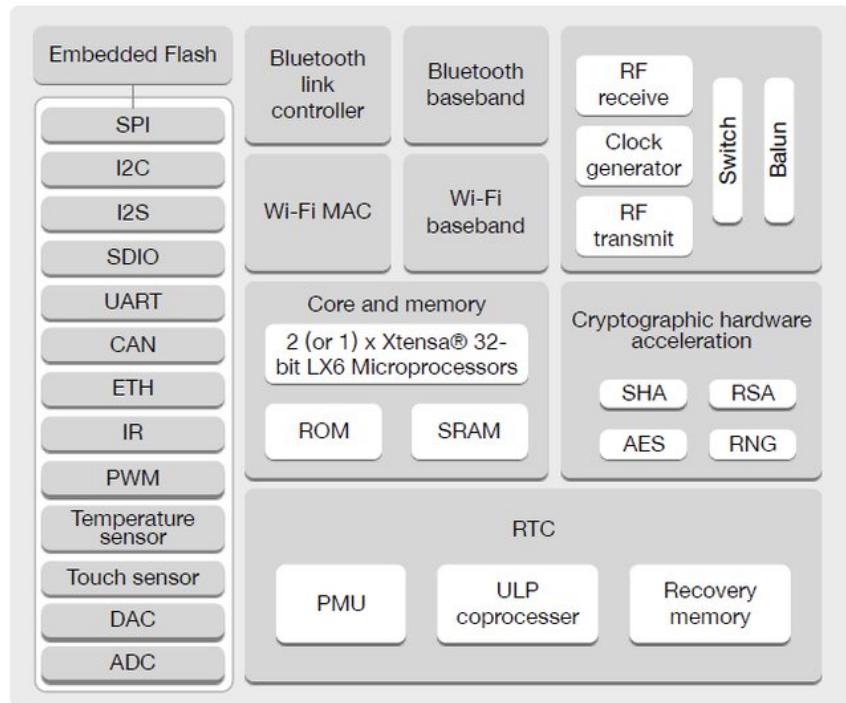


Рисунок 1.3 – Функциональная схема ESP32

point).

Wi-Fi модуль ESP32 включает в себя несколько ключевых компонентов:

1. RF-часть - отвечает за передачу и прием радиосигналов;
2. MAC-уровень - управляет доступом к среде передачи и обработкой данных;
3. Программное обеспечение - реализует стек протоколов, включая TCP/IP.

ESP32 использует Non-Volatile Memory (NVM) для хранения конфигурационных данных, что позволяет сохранять настройки даже при отключении питания. В NVM хранятся следующие данные:

- SSID и пароль для подключения к Wi-Fi сети;
- Настройки безопасности (например, тип шифрования);
- Параметры сети (IP-адрес, маска подсети, шлюз).

Режим работы STA

Режим STA (Station) позволяет ESP32 подключаться к существующей Wi-Fi сети. В этом режиме микроконтроллер функционирует как клиент,

который может обмениваться данными с сервером или другими устройствами в сети. Основные этапы работы в режиме STA:

1. Сканирование сетей - ESP32 ищет доступные Wi-Fi сети и выбирает нужную;
2. Подключение - после выбора сети происходит аутентификация с использованием SSID и пароля, хранящихся в NVM;
3. Получение IP-адреса - после успешного подключения ESP32 запрашивает IP-адрес через DHCP или использует статическую конфигурацию.

Режим работы AP

Режим Access Point (AP) позволяет ESP32 функционировать как точка доступа, к которой могут подключаться другие устройства. В этом режиме ESP32 создает собственную Wi-Fi сеть, предоставляя возможность подключения к ней.

Основные этапы работы в режиме AP:

1. Создание сети - ESP32 инициализирует Wi-Fi модуль и создает сеть с заданным SSID и паролем;
2. Настройка параметров - можно настроить параметры безопасности, такие как тип шифрования (WPA2, WPA, открытая сеть);
3. Подключение клиентов - другие устройства могут подключаться к созданной сети, используя указанные SSID и пароль.
4. Обмен данными - после подключения клиенты могут обмениваться данными с ESP32, который может выступать в роли сервера, обрабатывая запросы и отправляя ответы

1.4.2 Распределение портов

MK ESP32 имеет 34 вывода общего назначения (GPIO), которые могут использоваться для различных функций. Из них 15 выводов могут быть использованы для аналогового ввода (ADC), а также поддерживают функции PWM, I2C, SPI и UART. . Выводы микроконтроллера ESP32 показаны на рисунке 1.4.

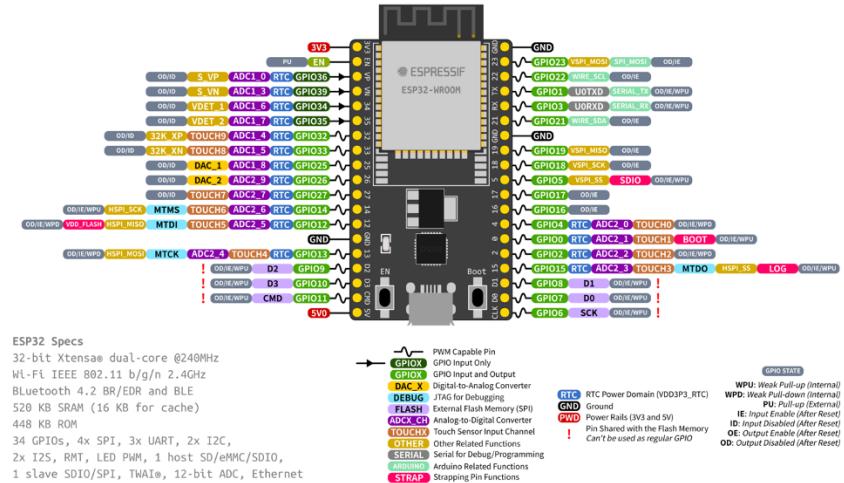


Рисунок 1.4 – Выводы MK ESP32

Классификация выводов

Выводы ESP32 можно классифицировать по некоторым критериям:

По назначению:

- **GPIO (General Purpose Input/Output):** Используются для общего назначения, могут быть настроены как входы или выходы;
- **ADC (Analog to Digital Converter):** Позволяют считывать аналоговые сигналы и преобразовывать их в цифровые значения;
- **PWM (Pulse Width Modulation):** Используются для управления мощностью, например, в светодиодах или моторах;
- **I2C, SPI, UART:** Интерфейсы для связи с другими устройствами;

По функциональности:

- цифровые выводы: Поддерживают только два состояния (высокий и низкий уровень);
- аналоговые выводы: Позволяют считывать диапазон значений, что делает их полезными для работы с датчиками.

По уровню напряжения:

- Логические уровни: Поддерживают уровни 0 и 1, что соответствует 0 В и 3.3 В соответственно.

Назначение выводов

Выводы микроконтроллера ESP32 предназначены для выполнения задач в рамках проектирования и разработки систем управления. Они позволяют:

- подключать и управлять внешними устройствами (датчиками, исполнительными механизмами).
- обеспечивать связь между микроконтроллером и другими компонентами системы.
- реализовывать функции управления и мониторинга в реальном времени.

1.4.3 Описание датчика влажности и температуры воздуха DHT-11

DHT-11[5] — это цифровой датчик, предназначенный для измерения температуры и влажности воздуха. Он используется в проектах, связанных с микропроцессорными системами, благодаря своей простоте в использовании и доступной цене.

Внутреннее устройство

Сенсорная часть: Внутри датчика находятся два основных сенсора — один для измерения температуры, другой для влажности. Сенсор влажности основан на изменении электрического сопротивления, которое зависит от уровня влажности. Сенсор температуры использует термистор, который изменяет свое сопротивление в зависимости от температуры.

Аналогово-цифровой преобразователь (АЦП): Этот компонент преобразует аналоговые сигналы, полученные от сенсоров, в цифровые данные, которые могут быть обработаны микроконтроллером.

Микроконтроллер: Внутренний микроконтроллер управляет работой датчика, обрабатывает данные и отправляет их через однопроводной интерфейс.

Корпус: Датчик защищен пластиковым корпусом, который обеспечивает защиту от внешних воздействий и позволяет воздуху свободно проходить к сенсорам.

Принцип работы

Измерение влажности: Сенсор влажности использует гигроскопический материал, который изменяет свое электрическое сопротивление в зависимости от уровня влажности. Когда влажность увеличивается, материал поглощает воду, что приводит к изменению его сопротивления. Это изменение фиксируется и преобразуется в цифровой сигнал.

Измерение температуры: Сенсор температуры использует термистор, который изменяет свое сопротивление в зависимости от температуры окружающей среды. Изменение температуры приводит к изменению сопротивления термистора, что также фиксируется и преобразуется в цифровой сигнал.

Передача данных: После обработки данных внутренним микроконтроллером, информация о температуре и влажности передается через однопроводной интерфейс к микроконтроллеру, который может использовать эти данные для дальнейшей обработки или отображения.

1.4.4 Описание ультразвукового датчика HC-SR04

HC-SR04[6] — это ультразвуковой датчик расстояния, который используется для измерения расстояния до объектов с помощью ультразвуковых волн. Он применяется в системах автоматизации и проектах, где требуется определение расстояния до препятствий.

Внутреннее устройство

- ультразвуковые излучатели: Датчик имеет два ультразвуковых излучателя — один для передачи ультразвуковых волн, другой для их приема. Излучатель генерирует звуковые волны с частотой около 40 кГц.
- микроконтроллер: внутренний микроконтроллер управляет работой датчика, генерирует сигналы для излучателя и обрабатывает возвращенные сигналы от приемника
- приемник: приемник фиксирует отраженные ультразвуковые волны, которые возвращаются от объектов, находящихся на определенном расстоянии

Принцип работы

Принцип работы основан на использовании ультразвуковых волн для измерения расстояния:

1. Генерация сигнала: При подаче сигнала на триггерный вход датчика, внутренний микроконтроллер активирует ультразвуковой излучатель, который генерирует короткий импульс ультразвуковых волн.
2. Отражение сигнала: Ультразвуковые волны распространяются в воздухе и отражаются от ближайшего объекта.
3. Прием сигнала: Отраженные волны возвращаются к датчику и фиксируются приемником. В этот момент микроконтроллер начинает отсчитывать время, прошедшее с момента отправки сигнала до его получения.
4. Расчет расстояния: После получения сигнала микроконтроллер останавливает отсчет времени и рассчитывает расстояние до объекта, используя формулу:

$$\text{Расстояние} = \frac{\text{Скорость звука} \times \text{Время}}{2}$$

. Скорость звука в воздухе составляет примерно 343 метра в секунду при температуре 20 °C. Деление на 2 необходимо, так как время учитывает путь туда и обратно.

1.4.5 Описание датчика влажности почвы YL-69

YL-69[7] — это датчик влажности почвы, который используется для измерения уровня влажности в почве. Он применяется в системах автоматического полива, агрономии и проектах, где необходимо контролировать состояние почвы.

Внутреннее устройство

- электроды: Датчик имеет два металлических электрода, которые погружаются в почву и измеряют её проводимость, что позволяет определить уровень влажности.
- аналоговый выход: YL-69 генерирует аналоговый сигнал, который пропорционален уровню влажности почвы. Этот сигнал можно подключить

к аналоговому входу микроконтроллера для дальнейшей обработки.

- защитный корпус: Датчик имеет защитный корпус, который предотвращает коррозию и повреждение электрических компонентов при контакте с почвой.

Принцип работы

Принцип работы основан на измерении проводимости почвы:

1. Измерение проводимости: Когда электроды датчика погружаются в почву, через них проходит небольшой электрический ток. Уровень проводимости зависит от содержания влаги в почве.
2. Генерация сигнала: На основе измеренной проводимости датчик генерирует аналоговый сигнал, который пропорционален уровню влажности. Чем выше влажность, тем ниже сопротивление почвы и выше проводимость.
3. Передача данных: Аналоговый сигнал передается на микроконтроллер, который может использовать его для анализа состояния почвы и принятия решений о необходимости полива.

Вот описание датчика ТЕМТ6000:

1.4.6 Описание датчика ТЕМТ6000

ТЕМТ6000[8] — это фотодатчик, который используется для измерения уровня освещенности в окружающей среде. Он применяется в системах автоматического управления освещением, метеорологии и проектах, где необходимо контролировать уровень света.

Внутреннее устройство

- фотосенсор: Датчик содержит фотосенсор, который реагирует на уровень освещения и преобразует световую энергию в электрический сигнал.
- аналоговый выход: ТЕМТ6000 генерирует аналоговый сигнал, пропорциональный уровню освещенности. Этот сигнал можно подключить к аналоговому входу микроконтроллера для дальнейшей обработки.

- защитный корпус: Датчик имеет защитный корпус, который обеспечивает защиту от внешних воздействий и позволяет устанавливать его в различных условиях.

Принцип работы

Принцип работы основан на фотопроводимости:

1. Поглощение света: Когда свет попадает на фотосенсор, он поглощает световую энергию, что приводит к изменению его проводимости.
2. Генерация сигнала: На основе уровня освещения фотосенсор генерирует аналоговый сигнал, который пропорционален интенсивности света. Чем больше света, тем ниже сопротивление и выше выходной сигнал.
3. Передача данных: Аналоговый сигнал передается на микроконтроллер, который может использовать его для анализа уровня освещенности и принятия решений о необходимости изменения условий освещения.

1.4.7 Функциональная схема

На основе вышеописанных сведений была спроектирована функциональная схема разрабатываемой системы, показанная на рисунке 1.5 в приложении Б.

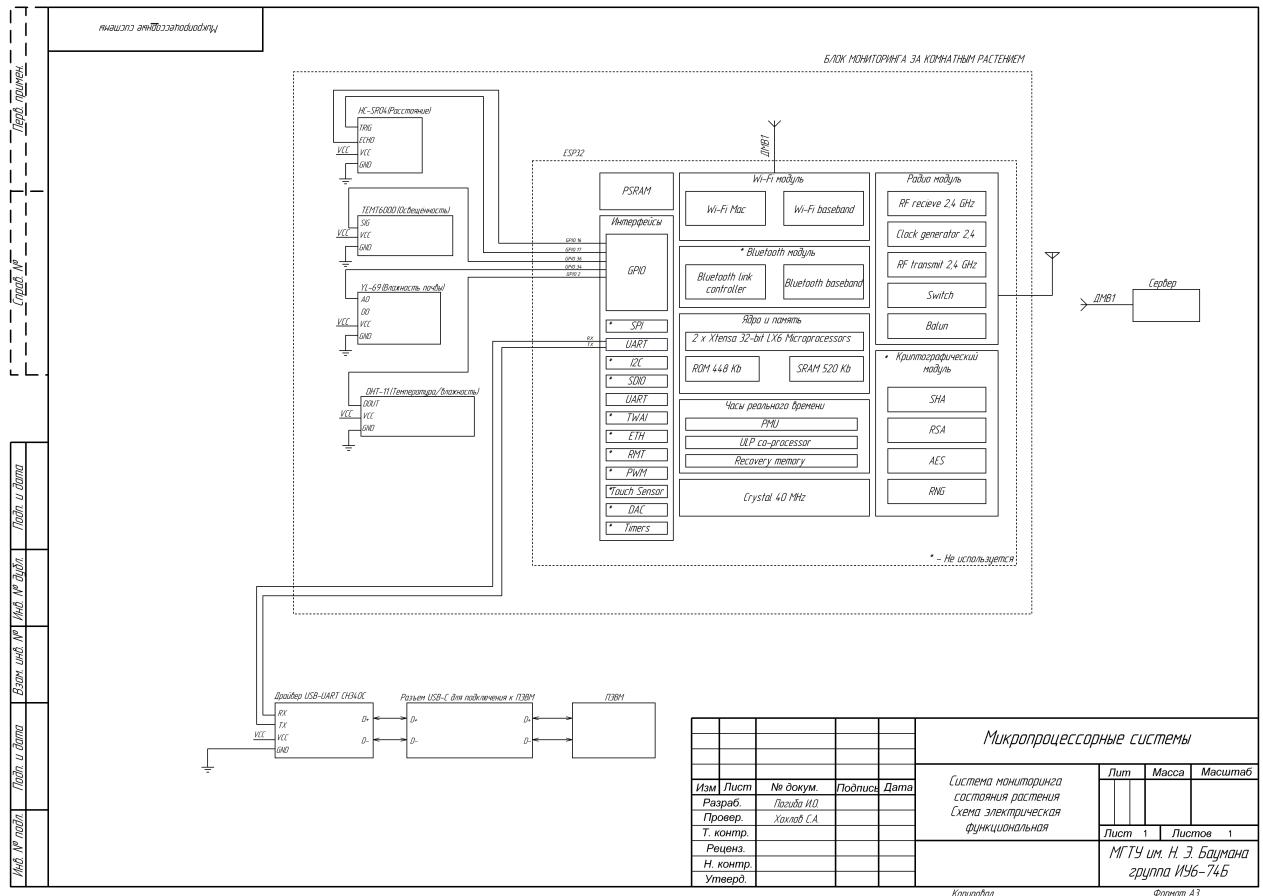


Рисунок 1.5 – Функциональная схема

1.5 Проектирование принципиальной схемы

Разъем программатора

Разъём программатора CH340 с USB Type-C используется для подключения к микроконтроллеру через интерфейс USB. CH340 – это USB-UART преобразователь, который позволяет компьютеру взаимодействовать с микроконтроллером через последовательный интерфейс.

Основные выводы и их функции:

1. VCC: Питание для микроконтроллера. Обычно 3.3V или 5V в зависимости от модели CH340
2. GND: Общий провод (земля)
3. TXD (Transmit Data): Передача данных от CH340 к микроконтроллеру
4. RXD (Receive Data): Приём данных от микроконтроллера к CH340

5. DTR (Data Terminal Ready): Используется для управления сигналом RESET микроконтроллера, чтобы ввести его в режим программирования
6. RTS (Request to Send): Может использоваться для дополнительных функций управления

Подключение через USB Type-C обеспечивает универсальный интерфейс, который поддерживает реверсивное подключение и высокую скорость передачи данных.

Подключение цепи питания

Для работы схемы с ESP32 необходимо подать напряжение, для этого будет использован блок питания с разъёмом USB, имеющий следующие технические характеристики:

- Выходной ток: до 3 А
- Выходное напряжение: 5 В
- Входное напряжение: 100 – 240 В (универсальный диапазон)

Для подключения питания к ESP32 будет использован стандартный USB кабель, который подключается к USB type-c разъёму на плате ESP32.

Основные элементы подключения:

1. Блок питания: обеспечивает стабильное напряжение 5 В и ток до 3 А, что достаточно для питания ESP32 и подключенных к нему периферийных устройств.
2. Кабель USB:
 - Один конец подключается к USB разъёму блока питания
 - Другой конец подключается к USB разъёму на плате ESP32
3. ESP32: Питание подаётся через USB разъём, который обеспечивает подключение к VCC (5V) и GND (земля) на плате ESP32.

Расчет потребляемой мощности

Потребляемая мощность – это мощность, потребляемая интегральной схемой, которая работает в заданном режиме соответствующего источника питания. Чтобы рассчитать суммарную мощность, рассчитаем мощность каждого элемента. На все микросхемы подается напряжение +3.3В.

Мощность, потребляемая одним устройством, в статическом режиме, рассчитывается формулой.

$$P = U * I$$

где U – напряжение питания (В); I – ток потребления микросхемы (мА).

Также в схеме присутствуют резисторы и транзисторы. Мощность для резисторов рассчитывается по формуле:

$$P = I^2 * R$$

где R – сопротивление резистора; I – ток, проходящий через резистор.

Для транзисторов по формуле:

$$P = U * I$$

где U – напряжение коллектор-эмиттер; I – ток коллектора.

Расчет потребляемого напряжения для каждой микросхемы показан в таблице 1.1.

Также в схеме используются 4 резистора с номиналом 10 кОм, 2 резистора с номиналом 1 кОм, и 2 транзистора J3Y.

$$\sum P = 660 + 33 + 33 + 0.99 + 6.6 + 115.5 + 66 + 3.3 + 4 * 10 * 3.3 + 2 * 1 = 986 \text{ mW}$$

Построение принципиальной схемы

На основе всех вышеописанных сведений была спроектирована принципиальная схема разрабатываемой системы, показанная на рисунке 1.6 и в приложении Б.

Таблица 1.1 – Потребляемая мощность

Микросхема	Ток потребления, мА	Потребляемая мощность, мВт	Количество устройств	Суммарная потребляемая мощность, мВт
ESP-32	200	660	1	660
CH340C	10	33	1	33
AMS117	10	33	1	33
DHT-11	0.3	0.99	1	0.99
HCSR-04	2	6.6	1	6.6
YL-69	35	115.5	1	115.5
TEMT6000	20	66	1	66
LED	1	3.3	1	3.3

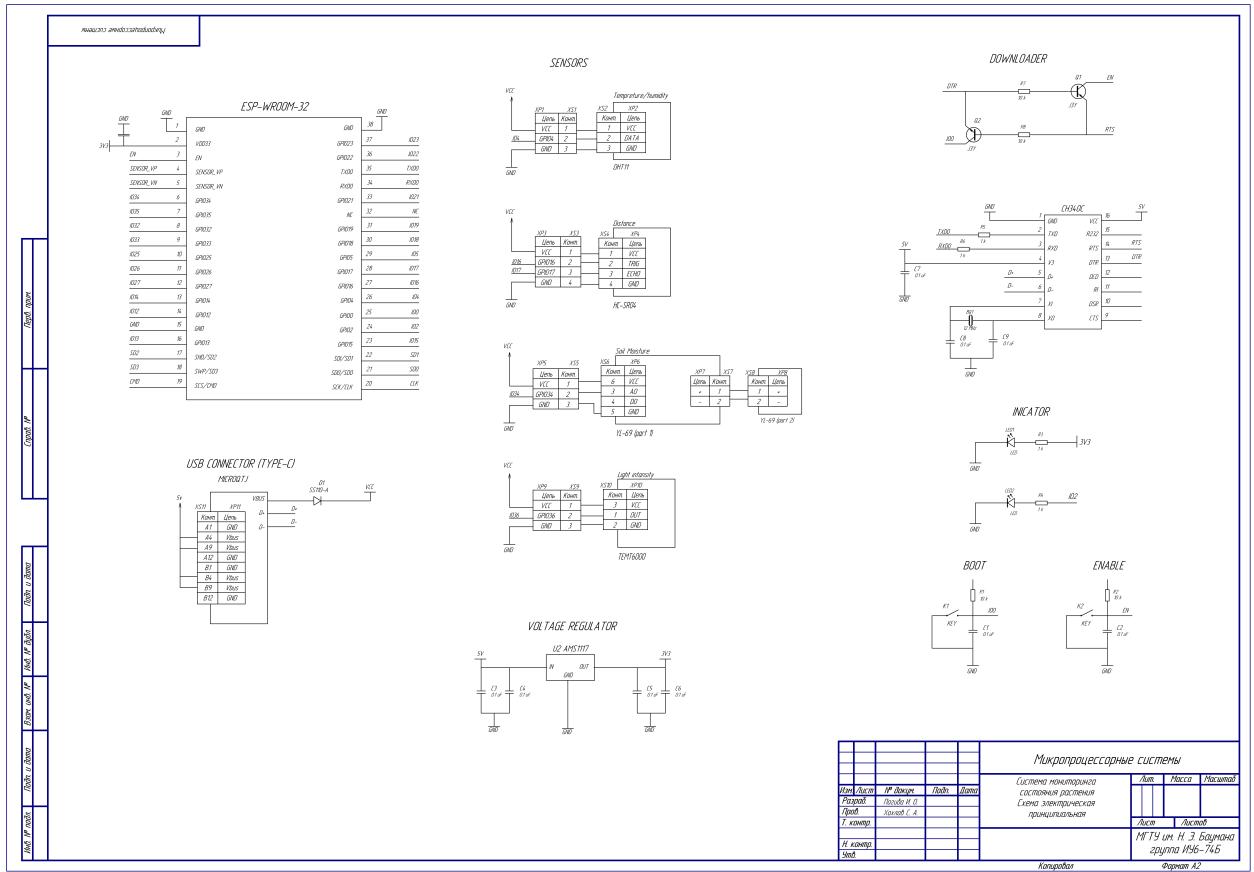


Рисунок 1.6 – Принципиальная схема

1.6 Описание алгоритмов основных программных процедур

1.6.1 Главная процедура программы

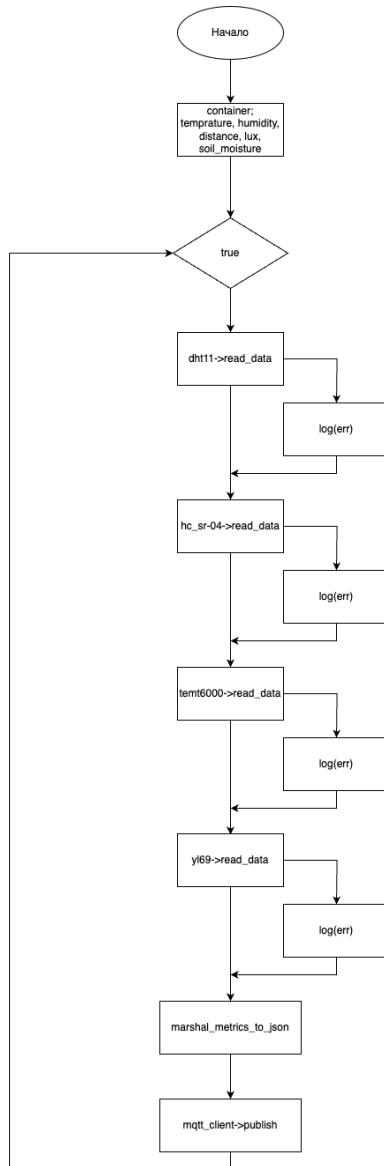


Рисунок 1.7 – Главная процедуря программы

2 Технологическая часть

2.1 Моделирование и отладка системы на макетной плате

Для разработки проекта использовалась среда CLion [9] от компании JetBrains, позволяющая разрабатывать программные продукты на языках C/C++. Для программирования и отладки на плате esp использовался фреймворк IDF-ESP [10].

При разработке в среде CLion с использованием плагина ESP-IDF для микроконтроллеров ESP32, процесс выглядит следующим образом:

- конфигурация проекта: основной конфигурационный файл проекта — sdkconfig. В него необходимо вручную добавлять переменные окружения, такие как настройки Wi-Fi, поскольку они не импортируются автоматически. Это позволяет управлять параметрами проекта.
- локальные конфигурации: файлы Kconfig.projbuild не импортируются автоматически в проект, поэтому их нужно обрабатывать вручную, если требуется специфическая конфигурация для отдельных компонентов.
- компиляция и прошивка: для прошивки платы используется команда flash, в которой необходимо указать ELF-файл, полученный после компиляции. Это позволяет загружать скомпилированную программу непосредственно на микроконтроллер (см. рисунок 2.1).
- отладка и тестирование: ESP-IDF предоставляет возможность отладки в виде запуска исполняемого файла monitor, сделать это можно с помощью скрипта. (см. листинг

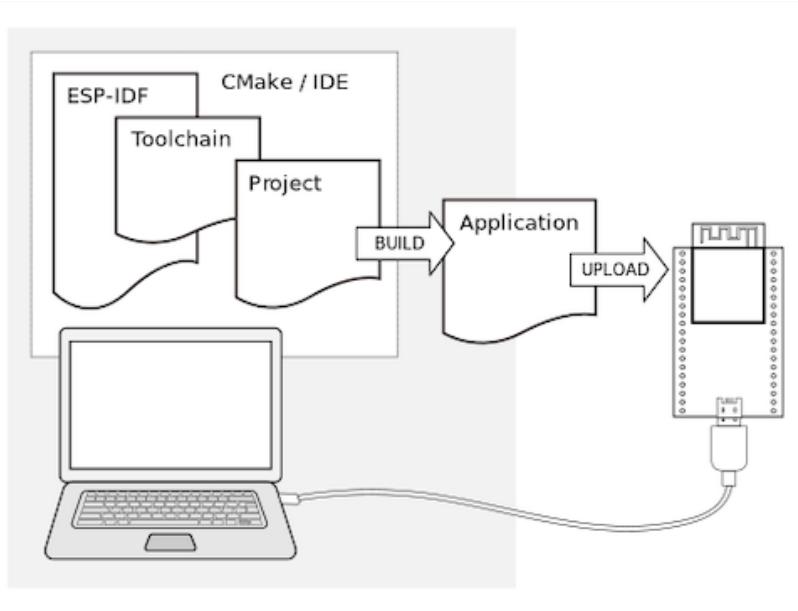


Рисунок 2.1 – Процесс разработки и прошивки платы ESP

Листинг 2.1 – shell

```

Команда для запуска отладки - мониторинга:
cd ~/esp/esp-idf && ./export.sh && cd
~/esp/plant_monitoring_project && idf.py monitor

##### Примечания

# Задаем путь проекта
PROJECT_PATH="~/esp/plant_monitoring_project"
FRAMEWORK_PATH="~/esp"

# Формируем команду mosquitto_pub
COMMAND="mosquitto_pub -L mqtt://$HOST/$TOPIC -m \"\$DATA\""

# Выполняем команду
eval "$COMMAND"

```

Скрипт для запуска отладки при разработке в фреймворке ESP-IDF

После сборки схемы, программа была прошита и протестирована. Итоговый результат можете видеть на рисунке 2.2 и 2.3. В итоговой конструкции макетная плата находится сверху над комнатным растением, там же расположены датчики освещённости, температуры и влажности воздуха. Ультразвуковой датчик прикреплён с обратной стороны конструкции, высота горшка и высота конструкции была заранее высчитана

и учитывается при подсчёте роста растения. Датчик влажности почвы расположен в почве горшка.

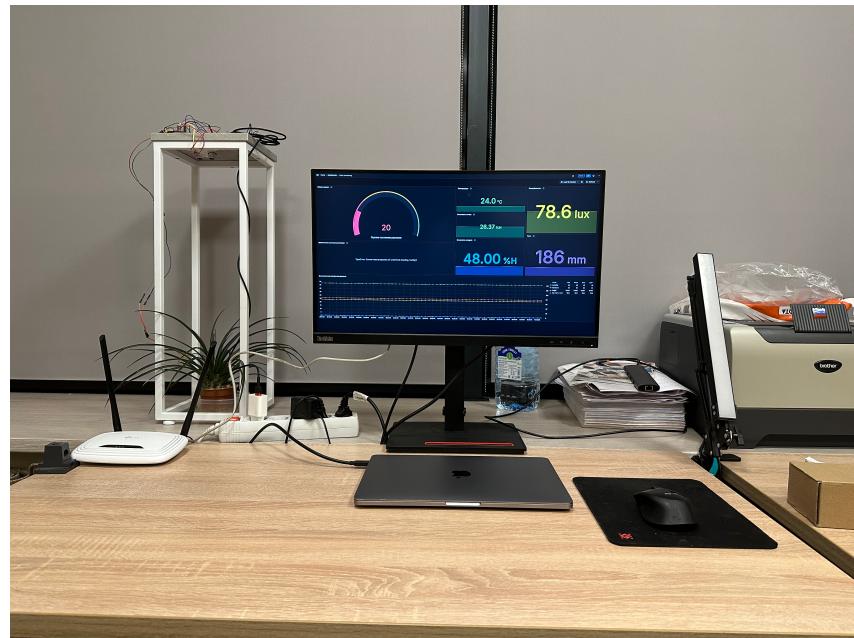


Рисунок 2.2 – Общая фотография комнатного растения, макетной платы и выводимых метрик

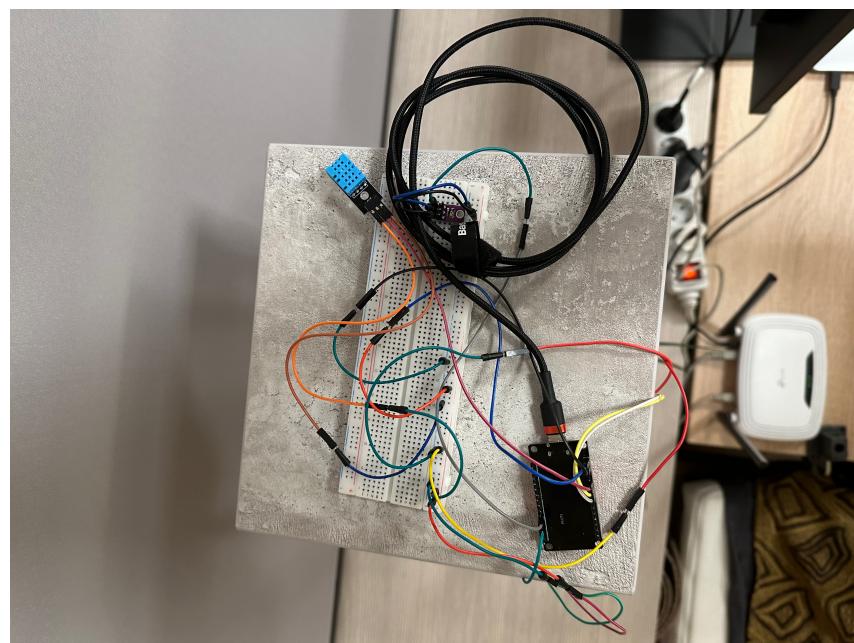


Рисунок 2.3 – Детализированная фотография макетной платы

2.2 Разработка серверной части системы мониторинга

Для разработки серверной части приложения был арендован VPS(Virtual private server) сервер по адресу.

2.2.1 MQTT-брокер

Брокер MQTT — главный узел (серверное ПО) для подключения IoT устройств и обмену сообщений между ними.

Топик - канал или адрес, по которому сообщения отправляются и принимаются.

На сервер был установлен MQTT-брокер mosquitto [11] с помощью следующих команд:

Установка

Листинг 2.2 – shell

```
sudo apt install -y mosquitto
sudo systemctl status mosquitto
```

Исходный код установки и запуска mosquitto

Отправка сообщений

На брокер не была добавлена авторизация, так как проект является академическим и технические требования при этом не нарушаются. Для отправки валидного сообщения нужно указать хост и порт, на который происходит отправка сообщений, сообщение в формате json и также указать топик, к которому идёт отправка.

Листинг 2.3 – shell

```
# Задаем переменную хоста
HOST="vm4481772.25ssd.had.wf:1883"

# Определяем JSON данные
DATA='{"temperature":90.25,"humidity":46.37,
       "linkquality":1.0}'
```

```

# Определяем топик
TOPIC="esp32/plant_monitoring/metrics"

# Формируем команду mosquitto_pub
COMMAND="mosquitto_pub -L mqtt://$HOST/$TOPIC -m \"\$DATA\""

# Выполняем команду
eval "$COMMAND"

```

Исходный код отправки сообщения на брокер mosquitto

Клиент MQTT

Для валидирования доставки отправки метрик можно воспользоваться MQTT-Explorer [12]. Настройки MQTT-explorer:

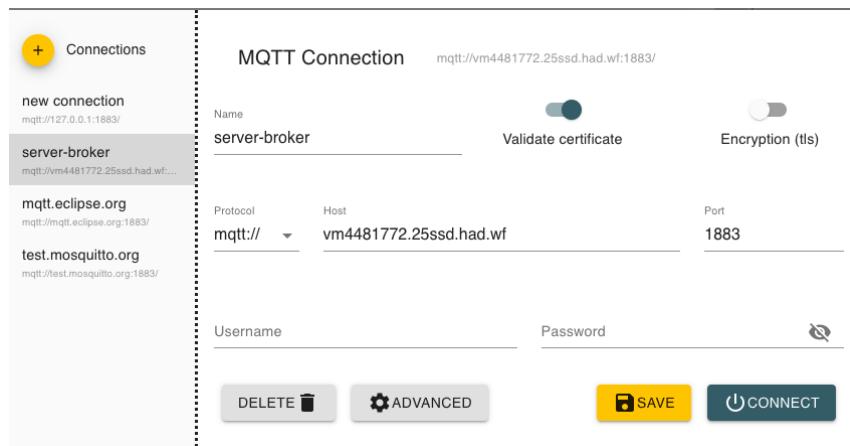


Рисунок 2.4 – Настройки MQTT-explorer

Во вкладке advanced необходимо подписаться на топик /esp32/plant_monitoring/metrics и удалите топик SYS.

2.2.2 Деплой приложения

Как было упомянуто в 1.2 на сервере с помощью cli утилиты docker-compose были запущены контейнеры следующих приложений:

- mqtt-exporter
- prometheus

- alertmanager
- grafana

Docker Compose — это инструмент для определения и запуска многоконтейнерных приложений с использованием Docker. Он позволяет описывать конфигурацию приложения в одном файле (обычно docker-compose.yml), где можно указать, какие контейнеры нужны, их настройки, зависимости и сети.

2.2.3 Prometheus

Из MQTT-брюкера с помощью утилиты mqtt-exporter метрики попадают в Prometheus. Prometheus обладает базовым функционалом визуализации метрик, поэтому для отображения было выбрано приложение Grafana.

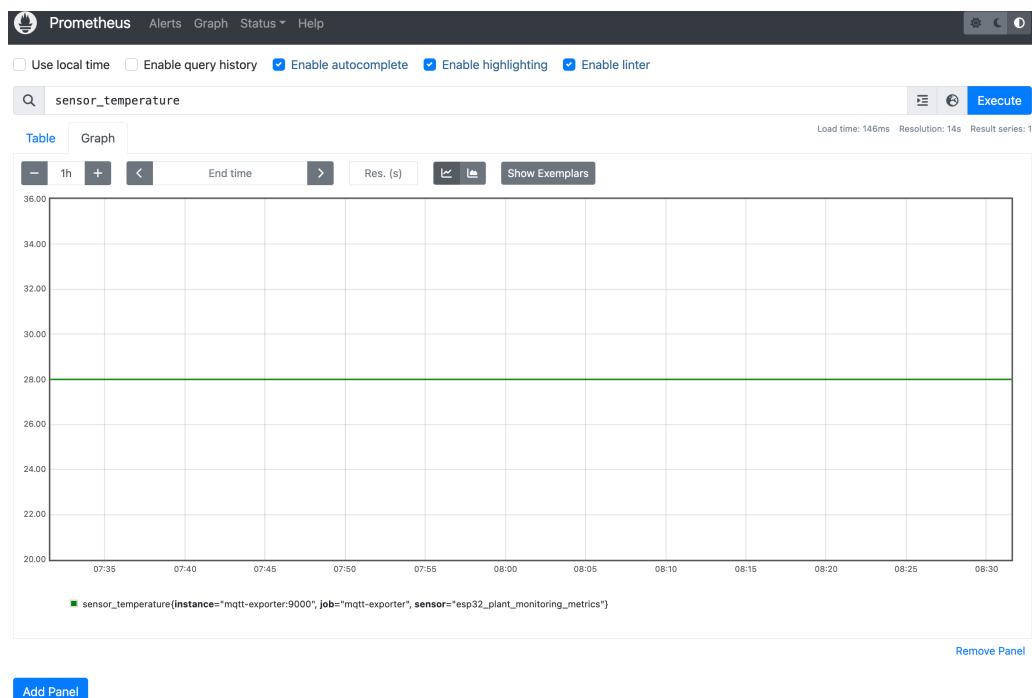


Рисунок 2.5 – Скриншот приложения Prometheus, запущенного на сервере

2.2.4 Grafana

Для визуализации метрик Grafana использовался язык PromQL [13].

PromQL (Prometheus Query Language) — это язык запросов, используемый в системе мониторинга Prometheus для извлечения и обработки

данных о метриках. Он позволяет пользователям формулировать запросы к временным рядам данных, собранным Prometheus, и выполнять операции, такие как агрегация, фильтрация и математические вычисления.

В 1.3 была выведена формула оценки общего состояния растения, для визуализации этой метрики в grafana было принято решение брать среднее взвешенное значение показателей датчиков за последние 5 минут.

Листинг 2.4 – promql

```
(  
    (avg_over_time(sensor_temperature[5m]) - 20) / (25 -  
        20) +  
    (avg_over_time(sensor_humidity[5m]) - 40) / (60 - 40) +  
    (avg_over_time(sensor_soil_moisture[5m]) - 30) / (50 -  
        30) +  
    (avg_over_time(sensor_lux[5m]) - 200) / (800 - 200)  
) / 4  
* 100
```

Запрос общей оценки состояния растения в языке PromQL

Для большей выразительности помимо текущего состояния метрик на борту была добавлена динамика метрик во времени и график алERTов.

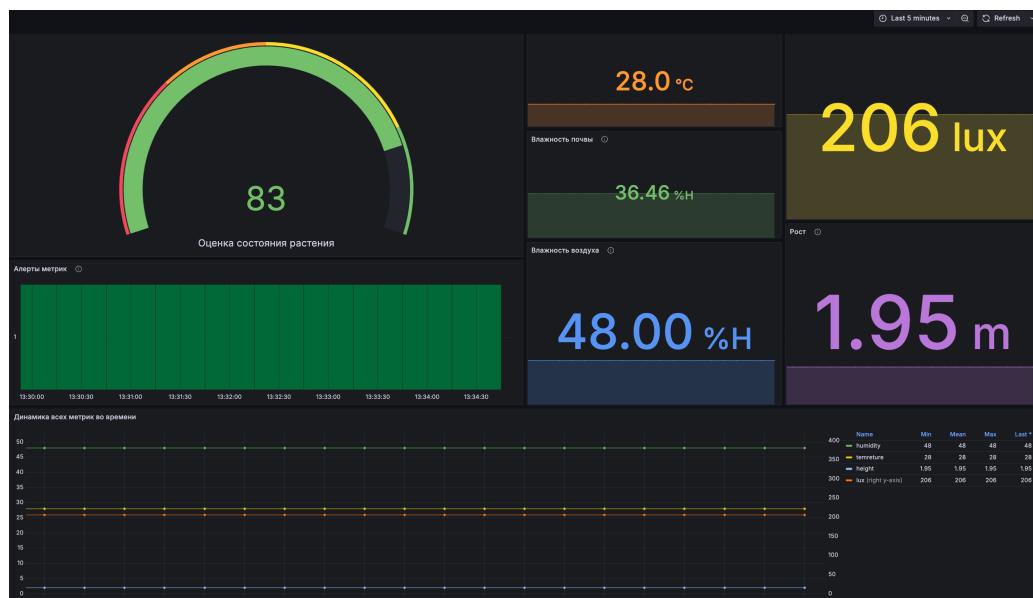


Рисунок 2.6 – Скриншот дэшборда Grafana, отображающий метрики растения

2.2.5 AlertManager

Alertmanager — это компонент системы мониторинга Prometheus, предназначенный для управления оповещениями. Он обрабатывает уведомления, которые генерируются Prometheus на основе заданных правил, и управляет их отправкой в различные каналы (например, электронная почта или мессенджер).

В качестве канала оповещения был выбран Telegram, так как он является наиболее популярным мессенджером в мире и имеет просто и понятное API.

Оповещения были настроены таким образом, чтобы алерт вызывался, если растение находится в неблагоприятных условиях в течение определенного времени. Рассмотрим на одном из правил алертинга, которое задавалось в файле alerts.rules. (см. листинг 2.5)

Листинг 2.5 – rules

```
- alert: critical_temperture
expr: '(sensor_temperature > 40 or sensor_temperature < 10)'
for: 20m
labels:
    severity: notification-critical
annotations:
    description: "Temperature out of range for 20 minutes:
        {{ $labels.sensor }}"
```

Правило определения экстремальной температуры в окружающей среде. Если датчик в течение 20 минут фиксирует температуру большую 40 или меньшую 10, то алерт срабатывает и ему присваивается критический уровень.

Настройка формата сообщения настраивалась с помощью шаблона – формат .tmpl. Шаблон был взят из открытых источников [14]. Тип получателя – канал, в который будут отправляться оповещения, настраивается с помощью .yaml файла. (см. листинг 2.6). Для использования бота, нужно определить chat_id и создать токен [15].

Листинг 2.6 – yml

```
route:
```

```

group_by: [ 'alertname' ]
group_wait: 10m
group_interval: 10m
repeat_interval: 20m
receiver: 'telegram'

receivers:
  - name: 'telegram'
    telegram_configs:
      - chat_id: "your_chat_id"
        bot_token: "your_bot_token"
        send_resolved: true
        parse_mode: 'HTML'
        message: '{{ template "telegram.message". }}'

templates:
  - './message_templates/telegram tmpl' # path to our
    telegram tmpl

```

Настройка типа получателя в виде бота в Telegram

Бот будет присыпать сообщение в чат после интервала времени group_interval. И далее слать их повторно, каждые repeat_interval, пока алерт не будет закрыт. После закрытия алерта также будет прислано сообщение. (см. рисунок 2.7)

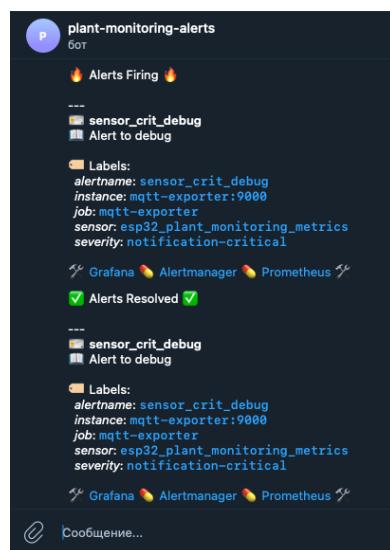


Рисунок 2.7 – Оповещений о критическом состоянии растения и уведомление о решении проблемы

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта был спроектирована система мониторинга состояния комнатного растения. Система работает на основе МК серии ESP – ESP32. Устройство разработано в соответствии с ТЗ.

В результате проектирования были разработаны принципиальная и функциональная электрические схемы для аппаратной части устройства. Код программы, написанный на языках C/C++, отлажен, протестирован, была собрана и прошита принципиальная схема, вывод метрик осуществляется на удалённый сервер.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. MQTT - The Standard for IoT Messaging. — URL: <https://mqtt.org/> (дата обращения: 23.11.2024).
2. Kpetremann/Mqtt-Exporter: Simple Generic MQTT Prometheus Exporter for IoT Working out of the Box. — URL: <https://github.com/kpetremann/mqtt-exporter> (дата обращения: 23.11.2024).
3. Overview | Prometheus. — URL: <https://prometheus.io/docs/introduction/overview/> (дата обращения: 23.11.2024).
4. Grafana OSS and Enterprise | Grafana Documentation / Grafana Labs. — URL: <https://grafana.com/docs/grafana/latest/> (: 23.11.2024).
5. DHT11 Temperature-Humidity Sensor, Датчик Температуры и Влажности На Базе DHT11, Цифровой Выход, Waveshare | Купить в Розницу и Оптом. — URL: <https://www.chipdip.ru/product/temperature-humidity-sensor> (дата обращения: 23.11.2024).
6. Ультразвуковой дальномер HC-SR04. — URL: <https://amperka.ru/product/hc-sr04-ultrasonic-sensor-distance-module> (дата обращения: 23.11.2024).
7. Guide for Soil Moisture Sensor YL-69 or HL-69 with the Arduino | Random Nerd Tutorials. — 07/14/2016. — URL: <https://randomnerdtutorials.com/guide-for-soil-moisture-sensor-yl-69-or-hl-69-with-the-arduino/> (: 23.11.2024).
8. *admin.* Датчик освещенности TEMT6000 | arduinoLab. — 19.08.2022. — URL: <https://arduinolab.pw/index.php/2022/08/19/datchik-osveshhennosti-temt6000/> (дата обращения: 23.11.2024).
9. ESP-IDF | CLion / CLion Help. — URL: <https://www.jetbrains.com/help/clion/esp-idf.html> (: 23.11.2024).

10. Get Started - ESP32 - — ESP-IDF Programming Guide v5.2.3 Documentation. — URL: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/index.html> (дата обращения: 22.11.2024).
11. Ndungu F. Install Mosquitto MQTT Broker On Ubuntu 20.04 Server | Vultr Docs. — URL: <https://docs.vultr.com/install-mosquitto-mqtt-broker-on-ubuntu-20-04-server> (: 25.11.2024).
12. Nordquist T. MQTT Explorer / MQTT Explorer. — URL: <http://mqtt-explorer.com/> (: 25.11.2024).
13. Cmёna M. Что такое PromQL и как с ним работать. — 07.12.2023. — URL: <https://timeweb.com/ru/community/articles/cto-takoe-promql-i-kak-s-nim-rabotat> (дата обращения: 25.11.2024).
14. Telegram Message Template for Alertmanager & How to Add It to Your Receiver in Alertmanager.Yml. — URL: <https://gist.github.com/sanchpet/7641275a42243d3667b3146c5402be40> (дата обращения: 25.11.2024).
15. 262588213843476. How to Get Telegram Bot Chat ID / Gist. — URL: <https://gist.github.com/nafiesl/4ad622f344cd1dc3bb1ecbe468ff9f8a> (: 25.11.2024).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Исходный код файла - main.cpp

```
1 //  
2 // Created by ipogiba on 06.11.2024.  
3 //  
4 #include "container.h"  
5  
6 extern "C" void app_main()  
7 {  
8     run();  
9 }
```

Листинг А.2 – Исходный код файла реализации класса контейнера - containter.cpp

```
1 #include "container.h"  
2  
3 #include <cJSON.h>  
4 #include <wifi.h>  
5  
6 #include <memory>  
7 #include <mqtt.hh>  
8  
9 #include "dht.h"  
10 #include "driver/gpio.h"  
11 #include "esp_log.h"  
12 #include "mqtt_client.h"  
13 #include "nvs_flash.h"  
14  
15 #define HC_SR04_ECHO GPIO_NUM_17  
16 #define HC_SR04_TRIGGER GPIO_NUM_16  
17 #define DHT_GPIO_DATA GPIO_NUM_4  
18 #define TEMT6000_PIN ADC1_CHANNEL_0  
19 #define YL_69_PIN ADC1_CHANNEL_6  
20 #define MAX_DISTANCE_CM 400  
21 #define COLLECT_METRICS_EVERY_N_SECONDS 10  
22  
23 SensorContainer::SensorContainer() {  
24     init_wifi_with_nvs();  
25 }
```

```

26     dht11 = std::make_unique<DHT11>(DHT_GPIO_DATA);
27     hc_sr04 = std::make_unique<Ultrasonic>(HC_SR04_ECHO,
28                                         HC_SR04_TRIGGER);
29     mqtt_client =
30         std::make_unique<MQTTClient>("esp32/plant_monitoring/");
31     temt6000 = std::make_unique<Tempt6000>(TEMT6000_PIN);
32     yl69 = std::make_unique<YL69>(YL_69_PIN);
33 }
34
35 _Noreturn void run()
36 {
37     const SensorContainer container;
38     float temperature, humidity, distance, lux,
39           soil_moisture;
40
41     while (true) {
42         // collect metrics from sensors - order is important
43         if (const auto dht_err =
44             container.dht11->read_data(temperature,
45             humidity); dht_err != ESP_OK) {
46             ESP_LOGE(DHT_TAG, "DHT11 read error: %s",
47                     esp_err_to_name(dht_err));
48         }
49
50         if (const auto ultrasonic_err =
51             container.hc_sr04->read_data(distance,
52             temperature); ultrasonic_err != ESP_OK) {
53             ESP_LOGE(DHT_TAG, "HC-SR-04 read error: %s",
54                     esp_err_to_name(ultrasonic_err));
55         }
56
57         if (const auto temt_err =
58             container.temt6000->read_data(lux); temt_err !=
59             ESP_OK) {
60             ESP_LOGE(DHT_TAG, "TEMT6000 read error: %s",
61                     esp_err_to_name(temt_err));
62         }
63
64         if (const auto soil_moisture_err =
65             container.yl69->read_data(soil_moisture);
66             soil_moisture_err != ESP_OK) {

```

```

53         ESP_LOGE(DHT_TAG, "YL-69 read error: %s",
54             esp_err_to_name(soil_moisture_err));
55
56     printf("Humidity: %.1f%% "
57         "Temp: %.1fC "
58         "Distance: %.4fm "
59         "Lux: %.1f "
60         "Soil Moisture: %.1f%% \n", humidity,
61             temperature, distance, lux,
62             soil_moisture);
63
64     // marshal metrics to json and send to the
65     // mqtt-broker
66     const auto metrics =
67         marshal_metrics_to_json(temperature, humidity,
68             lux, distance, soil_moisture);
69     container.mqtt_client->publish("metrics", metrics);
70
71     // If you read the sensor data too often, it will
72     // heat up
73     vTaskDelay(pdMS_TO_TICKS(2000));
74 }
75
76 std::string marshal_metrics_to_json(
77     const float temperature,
78     const float humidity,
79     const float lux,
80     const float distance,
81     const float soil_moisture
82 )
83 {
84     cJSON *root = cJSON_CreateObject();
85     cJSON_AddNumberToObject(root, "temperature",
86         temperature);
87     cJSON_AddNumberToObject(root, "humidity", humidity);
88     cJSON_AddNumberToObject(root, "lux", lux);
89     cJSON_AddNumberToObject(root, "distance", distance);
90     cJSON_AddNumberToObject(root, "soil_moisture",
91         soil_moisture);

```

```

85     return cJSON_Print(root);
86 }
87
88 // nvs - non voltage storage
89 void SensorContainer::init_wifi_with_nvs() {
90     esp_err_t ret = nvs_flash_init();
91     if (ret == ESP_ERR_NVS_NO_FREE_PAGES || 
92         ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
93         ESP_ERROR_CHECK(nvs_flash_erase());
94         ret = nvs_flash_init();
95     }
96     ESP_ERROR_CHECK(ret);
97
98     ESP_LOGI(WIFI_LOG_TAG, "ESP_WIFI_MODE_STA");
99     wifi_init_station_mode();
100}

```

Листинг А.3 – Исходный код заголовочного файла класса контейнера – containter.h

```

1 // 
2 // Created by ipogiba on 04.11.2024.
3 //
4
5 #ifndef SENSOR_CONTAINER_H
6 #define SENSOR_CONTAINER_H
7
8 #include <hc-sr04.h>
9 #include <temt6000.h>
10
11 #include <memory>
12 #include <mqtt.hh>
13 #include <string>
14 #include <y169.h>
15
16 #include "dht11.h"
17 #include "driver/adc.h"
18
19 class SensorContainer {
20 public:
21     SensorContainer(); // Constructor
22
23     std::unique_ptr<MQTTClient> mqtt_client;

```

```

24     std::unique_ptr<DHT11> dht11;
25     std::unique_ptr<Ultrasonic> hc_sr04;
26     std::unique_ptr<Temt6000> temt6000;
27     std::unique_ptr<YL69> yl69;
28 private:
29     static void init_wifi_with_nvs();
30 };
31
32 _Noreturn void run();
33 std::string marshal_metrics_to_json(float temperature,
34                                     float humidity, float lux, float distance, float
35                                     soil_moisture);
36 #endif // SENSOR_CONTAINER_H
```

Листинг А.4 – Исходный код заголовочного файла класса датчика dht11 - dht11.h

```

1 //
2 // Created by ipogiba on 10.11.2024.
3 //
4
5 #ifndef DHT11_H
6 #define DHT11_H
7 #include <esp_err.h>
8 #include <soc/gpio_num.h>
9
10 #define SENSOR_TYPE DHT_TYPE_DHT11
11 const auto DHT_TAG = "DHT sensor";
12
13 class DHT11 {
14 public:
15     explicit DHT11(gpio_num_t gpio_num);
16     esp_err_t read_data(float &temperature, float
17     &humidity) const;
18 private:
19     gpio_num_t data_dht_num;
20 };
21
22 #endif //DHT11_H
```

Листинг А.5 – Исходный код реализации класса датчика dht11 - dht11.cpp

```

1 //  

2 // Created by ipogiba on 10.11.2024.  

3 //  

4  

5 #include "dht11.h"  

6  

7 #include <hal/gpio_types.h>  

8 #include <soc/gpio_num.h>  

9 #include "dht.h"  

10 #include "driver/gpio.h"  

11  

12 DHT11::DHT11(const gpio_num_t gpio_num) {  

13     data_dht_num = gpio_num;  

14     gpio_set_pull_mode(data_dht_num, GPIO_PULLUP_ONLY);  

15 }  

16  

17 esp_err_t DHT11::read_data(float &temperature, float  

18     &humidity) const {  

19     return dht_read_float_data(SENSOR_TYPE, data_dht_num,  

        &humidity, &temperature);  

}

```

Листинг А.6 – Исходный код заголовочного файла класса датчика hc-sr04 - hc-sr04.h

```

1 //  

2 // Created by ipogiba on 10.11.2024.  

3 //  

4  

5 #ifndef ULTRASONIC_H  

6 #define ULTRASONIC_H  

7  

8 #include <esp_err.h>  

9 #include <soc/gpio_num.h>  

10 #include <ultrasonic.h>  

11  

12 const auto ULTRASONIC_TAG = "ULTRASONIC sensor";  

13  

14 class Ultrasonic {  

15 public:  

16     Ultrasonic(gpio_num_t hc_sr04_echo = GPIO_NUM_16,  

17                 gpio_num_t hc_sr04_trigger = GPIO_NUM_17);  

18     esp_err_t read_data(float &distance, const float

```

```

18     &tempreature) const;
19
20 private:
21     ultrasonic_sensor_t ultrasonic_sensor{};
22
23
24
25 #endif //ULTRASONIC_H

```

Листинг А.7 – Исходный код реализации класса датчика hc-sr04 - hc-sr04.cpp

```

1 // 
2 // Created by ipogiba on 10.11.2024.
3 //
4
5 #include "hc-sr04.h"
6
7 #include <esp_log.h>
8 #include <ultrasonic.h>
9 #define MAX_DISTANCE_CM 400
10
11 Ultrasonic::Ultrasonic(gpio_num_t hc_sr04_trigger,
12                         gpio_num_t hc_sr04_echo) {
13     ultrasonic_sensor = {
14         .trigger_pin = hc_sr04_trigger,
15         .echo_pin = hc_sr04_echo,
16     };
17
18     if (const auto ultrasonic_init_err =
19         ultrasonic_init(&ultrasonic_sensor));
20     ultrasonic_init_err != ESP_OK) {
21         ESP_LOGE(ULTRASONIC_TAG, "ultrasonic_init failed
22                 %s", esp_err_to_name(ultrasonic_init_err));
23     }
24
25     esp_err_t Ultrasonic::read_data(float &distance, const
26                                     float &tempreature) const {
27         if (const auto read_err =
28             ultrasonic_measure_temp_compensated(&ultrasonic_sensor,
29             MAX_DISTANCE_CM, &distance, tempreature); read_err
30             != ESP_OK) {

```

```

25         return read_err;
26     }
27     // так как это расстояние датчик -> объект -> датчик,
28     // то делим на 2
29     distance /= 2;
30     return ESP_OK;
}

```

Листинг А.8 – Исходный код заголовочного файла класса датчика hc-sr04 - hc-sr04.h

```

1 // 
2 // Created by ipogiba on 10.11.2024.
3 //
4
5 #ifndef ULTRASONIC_H
6 #define ULTRASONIC_H
7
8 #include <esp_err.h>
9 #include <soc/gpio_num.h>
10 #include <ultrasonic.h>
11
12 const auto ULTRASONIC_TAG = "ULTRASONIC sensor";
13
14 class Ultrasonic {
15 public:
16     Ultrasonic(gpio_num_t hc_sr04_echo = GPIO_NUM_16,
17                 gpio_num_t hc_sr04_trigger = GPIO_NUM_17);
18     esp_err_t read_data(float &distance, const float
19                         &temperature) const;
20
21 private:
22     ultrasonic_sensor_t ultrasonic_sensor{};
23
24
25 #endif //ULTRASONIC_H

```

Листинг А.9 – Исходный код реализации класса датчика hc-sr04 - hc-sr04.cpp

```

1 // 
2 // Created by ipogiba on 10.11.2024.
3 //

```

```

4
5 #include "hc-sr04.h"
6
7 #include <esp_log.h>
8 #include <ultrasonic.h>
9 #define MAX_DISTANCE_CM 400
10
11 Ultrasonic::Ultrasonic(gpio_num_t hc_sr04_trigger,
12     gpio_num_t hc_sr04_echo) {
13     ultrasonic_sensor = {
14         .trigger_pin = hc_sr04_trigger,
15         .echo_pin = hc_sr04_echo,
16     };
17
18     if (const auto ultrasonic_init_err =
19         ultrasonic_init(&ultrasonic_sensor));
20     ultrasonic_init_err != ESP_OK) {
21         ESP_LOGE(ULTRASONIC_TAG, "ultrasonic_init failed
22             %s", esp_err_to_name(ultrasonic_init_err));
23     }
24
25     esp_err_t Ultrasonic::read_data(float &distance, const
26         float &tempreature) const {
27         if (const auto read_err =
28             ultrasonic_measure_temp_compensated(&ultrasonic_sensor,
29             MAX_DISTANCE_CM, &distance, tempreature); read_err
30             != ESP_OK) {
31             return read_err;
32         }
33         // так как это расстояние датчик -> объект -> датчик,
34         // то делим на 2
35         distance /= 2;
36         return ESP_OK;
37     }

```

Листинг А.10 – Исходный код заголовочного файла класса датчика temt6000 - temt6000.h

```

1 //
2 // Created by ipogiba on 10.11.2024.
3 //
4

```

```

5 #ifndef TEMT6000_H
6 #define TEMT6000_H
7 #include <esp_adc_cal_types_legacy.h>
8 #include <esp_err.h>
9 #include <memory>
10
11
12 const auto LIGHT_TAG = "LIGHT sensor";
13
14 class Temt6000 {
15 public:
16     Temt6000(adc1_channel_t _channel);
17     esp_err_t read_data(float &lux) const;
18 private:
19     adc1_channel_t channel;
20     esp_adc_cal_characteristics_t adcCharacteristics; /**<
21         Characteristics of selected ADC unit */
22     uint32_t samplesNo = 10;
23 };
24 #endif //TEMT6000_H

```

Листинг А.11 – Исходный код реализации класса датчика temt6000 - temt6000.cpp

```

1 #include "temt6000.h"
2
3 #include <esp_adc_cal.h>
4 #include <driver/adc.h>
5 #include <esp_err.h>
6
7 #define TEMT6000_PIN ADC1_CHANNEL_0
8 #define TEMT6000_POWER_VOLTAGE 3.3
9 #define SAMPLES 10 // Количество сэмплов для усреднения
10
11
12 /** Output range of configured ADC unit */
13 #define TEMT6000_ADC_WIDTH
14     ADC_WIDTH_BIT_12
15 /** Default reference voltage of configured ADC unit */
16 #define TEMT6000_ADC_ATTEN_VREF 3900
17 /** Coefficient used for illuminance conversion from read

```

```

    output in voltage */
18 #define TEMT6000_ILLUMINANCE_CONVERTER_VALUE      0.2f
19 /** Specifies the value of lower margin reduction */
20 #define TEMT6000_ADC_MARGIN_LOW_REDUCED_VALUE     0
21
22
23 // Конструктор с инициализацией ADC
24 Temt6000::Tempt6000(adc1_channel_t _channel) {
25     channel = _channel;
26     esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_DB_0,
27                               ADC_WIDTH_BIT_10, TEMT6000_ADC_ATTEN_VREF,
28                               &adcCharacteristics);
29 }
30
31
32 // Метод для считывания данных
33 esp_err_t Temt6000::read_data(float &lux) const {
34     const int adc_value = adc1_get_raw(channel);
35     const float voltage = adc_value *
36         (TEMT6000_POWER_VOLTAGE / 1024); // Convert ADC
37         value to voltage
38     lux = 2640 - (voltage / 10000.0) * 2000000.0; // Convert voltage to lux
39     return ESP_OK;
40 }
```

Листинг А.12 – Исходный код заголовочного файла класса датчика yl-69 – yl69.h

```

1 //
2 // Created by ipogiba on 10.11.2024.
3 //
4
5 #ifndef YL69_H
6 #define YL69_H
7 #include <esp_adc_cal_types_legacy.h>
8 #include <esp_err.h>
9 #include <memory>
10
11 #include "driver/adc.h"
12
13 #define DEFAULT_VREF 1100 // Use adc2_vref_to_gpio() to
14     obtain a better estimate
```

```

15
16 class YL69 {
17 public:
18     YL69(adc1_channel_t _channel, uint32_t _num_of_samples
19           = 10, adc_atten_t _atten = ADC_ATTEN_DB_12,
20           adc_unit_t _unit = ADC_UNIT_1);
21     esp_err_t read_data(float &soil_moisture) const;
22 private:
23     adc1_channel_t channel;
24     uint32_t num_of_samples;
25     float normalize(uint32_t value_t) const;
26     uint32_t read_raw_data() const;
27     esp_adc_cal_characteristics_t    adcCharacteristics{};
28
29
30
31
32 #endif //YL69_H

```

Листинг A.13 – Исходный код реализации класса датчика yl-69 - yl69.cpp

```

1 // 
2 // Created by ipogiba on 10.11.2024.
3 //
4
5 #include "yl69.h"
6
7 #include <esp_adc_cal.h>
8
9 YL69::YL69(
10     const adc1_channel_t _channel,
11     const uint32_t _num_of_samples,
12     adc_atten_t _atten,
13     adc_unit_t _unit
14 ) {
15     channel = _channel;
16     num_of_samples = _num_of_samples;
17     esp_adc_cal_characterize(_unit, _atten,
18         ADC_WIDTH_BIT_12, DEFAULT_VREF, &adcCharacteristics);
19 }

```

```

20 uint32_t YL69::read_raw_data() const {
21     return adc1_get_raw(channel);
22 }
23
24 float YL69::normalize(uint32_t value_t) const {
25     return (value_t * 100) / value_max;
26 }
27
28
29 esp_err_t YL69::read_data(float& soil_moisture) const {
30     const uint32_t raw_value = read_raw_data();
31     soil_moisture = normalize(raw_value);
32
33     return ESP_OK; // Return success status
34 }
```

Листинг А.14 – Исходный код заголовочного файла инициализации Wi-Fi - wifi.h

```

1 #ifndef WIFI_H
2 #define WIFI_H
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 #include "esp_event.h"
9
10 const char* const WIFI_LOG_TAG = "wifi station";
11
12 void event_handler(void* arg, esp_event_base_t event_base,
13                     int32_t event_id, void* event_data);
14
15 void wifi_init_station_mode(void);
16
17 #ifdef __cplusplus
18 }
19 #endif
20
21 #endif /* ifndef WIFI_H */
```

Листинг А.15 – Исходный код реализации инициализации Wi-Fi - wi-fi.c

```

2 #include "wifi.h"
3
4 /* WiFi station Example
5    This example code is in the Public Domain (or CCO
       licensed, at your option.)
6    Unless required by applicable law or agreed to in
       writing, this
7    software is distributed on an "AS IS" BASIS, WITHOUT
       WARRANTIES OR
8    CONDITIONS OF ANY KIND, either express or implied.
9 */
10
11 #include <string.h>
12 #include "freertos/FreeRTOS.h"
13 #include "freertos/task.h"
14 #include "freertos/event_groups.h"
15 #include "esp_system.h"
16 #include "esp_wifi.h"
17 #include "esp_event.h"
18 #include "esp_log.h"
19 #include "nvs_flash.h"
20
21 #include "lwip/err.h"
22 #include "lwip/sys.h"
23
24
25 /* The examples use WiFi configuration that you can set via
       project configuration menu
26    If you'd rather not, just change the below entries to
       strings with
27    the config you want - ie #define EXAMPLE_WIFI_SSID
       "mywifissid"
28 */
29 #define EXAMPLE_ESP_WIFI_SSID      CONFIG_ESP_WIFI_SSID
30 #define EXAMPLE_ESP_WIFI_PASS     CONFIG_ESP_WIFI_PASSWORD
31 #define EXAMPLE_ESP_MAXIMUM_RETRY CONFIG_ESP_MAXIMUM_RETRY
32
33 #if CONFIG_ESP_WIFI_AUTH_OPEN
34 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_OPEN
35 #elif CONFIG_ESP_WIFI_AUTH_WEP
36 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WEP

```

```

37 #elif CONFIG_ESP_WIFI_AUTH_WPA_PSK
38 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_PSK
39 #elif CONFIG_ESP_WIFI_AUTH_WPA2_PSK
40 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_PSK
41 #elif CONFIG_ESP_WIFI_AUTH_WPA_WPA2_PSK
42 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD
43             WIFI_AUTH_WPA_WPA2_PSK
44 #elif CONFIG_ESP_WIFI_AUTH_WPA3_PSK
45 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA3_PSK
46 #elif CONFIG_ESP_WIFI_AUTH_WPA2_WPA3_PSK
47 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD
48             WIFI_AUTH_WPA2_WPA3_PSK
49 #endif
50
51 /* FreeRTOS event group to signal when we are connected*/
52 static EventGroupHandle_t s_wifi_event_group;
53
54 /* The event group allows multiple bits for each event , but
   we only care about two events:
   * - we are connected to the AP with an IP
   * - we failed to connect after the maximum amount of
     retries */
55 #define WIFI_CONNECTED_BIT BIT0
56 #define WIFI_FAIL_BIT      BIT1
57
58 static int s_retry_num = 0;
59
60 void event_handler(void* arg, esp_event_base_t event_base,
61                     int32_t event_id, void* event_data)
62 {
63     if (event_base == WIFI_EVENT && event_id ==
64         WIFI_EVENT_STA_START) {
65         esp_wifi_connect();
66     } else if (event_base == WIFI_EVENT && event_id ==
67         WIFI_EVENT_STA_DISCONNECTED) {
68         if (s_retry_num < EXAMPLE_ESP_MAXIMUM_RETRY) {
69             esp_wifi_connect();
70             s_retry_num++;
71             ESP_LOGI(WIFI_LOG_TAG, "retry to connect to the

```

```

                AP");
72     } else {
73         xEventGroupSetBits(s_wifi_event_group,
74                             WIFI_FAIL_BIT);
75     }
76     ESP_LOGI(WIFI_LOG_TAG, "connect to the AP fail");
77 } else if (event_base == IP_EVENT && event_id ==
78             IP_EVENT_STA_GOT_IP) {
79     ip_event_got_ip_t* event = (ip_event_got_ip_t*)
80             event_data;
81     ESP_LOGI(WIFI_LOG_TAG, "got ip:" IPSTR,
82             IP2STR(&event->ip_info.ip));
83     s_retry_num = 0;
84     xEventGroupSetBits(s_wifi_event_group,
85                         WIFI_CONNECTED_BIT);
86 }
87
88
89 void wifi_init_station_mode(void)
90 {
91     s_wifi_event_group = xEventGroupCreate();
92
93     ESP_ERROR_CHECK(esp_netif_init());
94
95     ESP_ERROR_CHECK(esp_event_loop_create_default());
96     esp_netif_create_default_wifi_sta();
97
98     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
99     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
100
101    esp_event_handler_instance_t instance_any_id;
102    esp_event_handler_instance_t instance_got_ip;
103    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
104                                                       ESP_EVENT_ANY_ID,
105                                                       &event_handler,
106                                                       NULL,
107                                                       &instance_any_id));
108
109    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
110                                                       ESP_EVENT_ANY_ID,
111                                                       &event_handler,
112                                                       NULL,
113                                                       &instance_got_ip));
114 }
```

```

107           NULL ,
108           &instance
109
110     wifi_config_t wifi_config = {
111         .sta = {
112             .ssid = EXAMPLE_ESP_WIFI_SSID,
113             .password = EXAMPLE_ESP_WIFI_PASS,
114             /* Authmode threshold resets to WPA2 as default
115                if password matches WPA2 standards (password
116                len => 8).
117                * If you want to connect the device to
118                  deprecated WEP/WPA networks, Please set the
119                  threshold value
120                * to WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK and set
121                  the password with length and format
122                  matching to
123                  * WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK standards.
124                  */
125             .threshold.authmode =
126                 ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
127             .sae_pwe_h2e = WPA3_SAE_PWE_BOTH,
128         },
129     };
130     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
131     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA,
132                                         &wifi_config));
133     ESP_ERROR_CHECK(esp_wifi_start());
134
135     ESP_LOGI(WIFI_LOG_TAG, "wifi_init_sta finished.");
136
137     /* Waiting until either the connection is established
138        (WIFI_CONNECTED_BIT) or connection failed for the
139        maximum
140        * number of re-tries (WIFI_FAIL_BIT). The bits are set
141        by event_handler() (see above) */
142     EventBits_t bits =
143         xEventGroupWaitBits(s_wifi_event_group,
144                             WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
145                             pdFALSE,
146                             pdFALSE,
147                             portMAX_DELAY);

```

```

136
137     /* xEventGroupWaitBits() returns the bits before the
138        call returned, hence we can test which event actually
139        happened. */
140     if (bits & WIFI_CONNECTED_BIT) {
141         ESP_LOGI(WIFI_LOG_TAG, "connected to ap SSID:%s
142                         password:%s",
143                         EXAMPLE_ESP_WIFI_SSID,
144                         EXAMPLE_ESP_WIFI_PASS);
145     } else if (bits & WIFI_FAIL_BIT) {
146         ESP_LOGI(WIFI_LOG_TAG, "Failed to connect to
147                         SSID:%s, password:%s",
148                         EXAMPLE_ESP_WIFI_SSID,
149                         EXAMPLE_ESP_WIFI_PASS);
150     } else {
151         ESP_LOGE(WIFI_LOG_TAG, "UNEXPECTED EVENT");
152     }
153 }
```

Листинг А.16 – Исходный код заголовочного файла инициализации Wi-Fi - wifi.h

```

1 #ifndef WIFI_H
2 #define WIFI_H
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 #include "esp_event.h"
9
10 const char* const WIFI_LOG_TAG = "wifi station";
11
12 void event_handler(void* arg, esp_event_base_t event_base,
13                     int32_t event_id, void* event_data);
14
15 void wifi_init_station_mode(void);
16
17 #ifdef __cplusplus
18 }
19 #endif
20
21#endif /* ifndef WIFI_H */
```

Листинг A.17 – Исходный код реализации инициализации Wi-Fi - wi-fi.c

```
1 #include "wifi.h"
2
3
4 /* WiFi station Example
5    This example code is in the Public Domain (or CCO
6       licensed, at your option.)
7    Unless required by applicable law or agreed to in
8       writing, this
9    software is distributed on an "AS IS" BASIS, WITHOUT
10      WARRANTIES OR
11      CONDITIONS OF ANY KIND, either express or implied.
12 */
13
14 #include <string.h>
15 #include "freertos/FreeRTOS.h"
16 #include "freertos/task.h"
17 #include "freertos/event_groups.h"
18 #include "esp_system.h"
19 #include "esp_wifi.h"
20 #include "esp_event.h"
21 #include "esp_log.h"
22 #include "nvs_flash.h"
23
24
25 /* The examples use WiFi configuration that you can set via
26    project configuration menu
27    If you'd rather not, just change the below entries to
28    strings with
29    the config you want - ie #define EXAMPLE_WIFI_SSID
30    "mywifissid"
31 */
32
33 #define EXAMPLE_ESP_WIFI_SSID           CONFIG_ESP_WIFI_SSID
34 #define EXAMPLE_ESP_WIFI_PASS          CONFIG_ESP_WIFI_PASSWORD
35 #define EXAMPLE_ESP_MAXIMUM_RETRY     CONFIG_ESP_MAXIMUM_RETRY
36
37 #if CONFIG_ESP_WIFI_AUTH_OPEN
38 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_OPEN
```

```

35 #elif CONFIG_ESP_WIFI_AUTH_WEP
36 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WEP
37 #elif CONFIG_ESP_WIFI_AUTH_WPA_PSK
38 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_PSK
39 #elif CONFIG_ESP_WIFI_AUTH_WPA2_PSK
40 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_PSK
41 #elif CONFIG_ESP_WIFI_AUTH_WPA_WPA2_PSK
42 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD
        WIFI_AUTH_WPA_WPA2_PSK
43 #elif CONFIG_ESP_WIFI_AUTH_WPA3_PSK
44 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA3_PSK
45 #elif CONFIG_ESP_WIFI_AUTH_WPA2_WPA3_PSK
46 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD
        WIFI_AUTH_WPA2_WPA3_PSK
47 #elif CONFIG_ESP_WIFI_AUTH_WAPI_PSK
48 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WAPI_PSK
49 #endif
50
51 /* FreeRTOS event group to signal when we are connected*/
52 static EventGroupHandle_t s_wifi_event_group;
53
54 /* The event group allows multiple bits for each event , but
   we only care about two events:
   * - we are connected to the AP with an IP
   * - we failed to connect after the maximum amount of
     retries */
55 #define WIFI_CONNECTED_BIT BIT0
56 #define WIFI_FAIL_BIT      BIT1
57
58 static int s_retry_num = 0;
59
60 void event_handler(void* arg, esp_event_base_t event_base,
61                   int32_t event_id, void* event_data)
62 {
63     if (event_base == WIFI_EVENT && event_id ==
64         WIFI_EVENT_STA_START) {
65         esp_wifi_connect();
66     } else if (event_base == WIFI_EVENT && event_id ==
67         WIFI_EVENT_STA_DISCONNECTED) {
68         if (s_retry_num < EXAMPLE_ESP_MAXIMUM_RETRY) {
69             esp_wifi_connect();

```

```

70         s_retry_num++;
71         ESP_LOGI(WIFI_LOG_TAG, "retry to connect to the
72             AP");
73     } else {
74         xEventGroupSetBits(s_wifi_event_group,
75             WIFI_FAIL_BIT);
76     }
77     ESP_LOGI(WIFI_LOG_TAG, "connect to the AP fail");
78 } else if (event_base == IP_EVENT && event_id ==
79     IP_EVENT_STA_GOT_IP) {
80     ip_event_got_ip_t* event = (ip_event_got_ip_t*)
81         event_data;
82     ESP_LOGI(WIFI_LOG_TAG, "got ip:" IPSTR,
83         IP2STR(&event->ip_info.ip));
84     s_retry_num = 0;
85     xEventGroupSetBits(s_wifi_event_group,
86         WIFI_CONNECTED_BIT);
87 }
88
89 void wifi_init_station_mode(void)
90 {
91     s_wifi_event_group = xEventGroupCreate();
92
93     ESP_ERROR_CHECK(esp_netif_init());
94
95     ESP_ERROR_CHECK(esp_event_loop_create_default());
96     esp_netif_create_default_wifi_sta();
97
98     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
99     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
100
101    esp_event_handler_instance_t instance_any_id;
102    esp_event_handler_instance_t instance_got_ip;
103    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT
104                                         ESP_EVENT_ANY_ID,
105                                         &event_handler,
106                                         NULL,
107                                         &instance_any_id));
108
109    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT
110                                         ESP_EVENT_ANY_ID,
111                                         &ip_event_handler,
112                                         NULL,
113                                         &instance_got_ip));
114 }
```

```

105          IP_EVENT_<IP>
106          &event_handler,
107          NULL,
108          &instance,
109
110      wifi_config_t wifi_config = {
111          .sta = {
112              .ssid = EXAMPLE_ESP_WIFI_SSID,
113              .password = EXAMPLE_ESP_WIFI_PASS,
114              /* Authmode threshold resets to WPA2 as default
115                 if password matches WPA2 standards (password
116                 len => 8).
117                 * If you want to connect the device to
118                 deprecated WEP/WPA networks, Please set the
119                 threshold value
120                 * to WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK and set
121                 the password with length and format
122                 matching to
123                 * WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK standards.
124
125                 */
126              .threshold.authmode =
127                  ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
128              .sae_pwe_h2e = WPA3_SAE_PWE_BOTH,
129          },
130      };
131      ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
132      ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA,
133          &wifi_config));
134      ESP_ERROR_CHECK(esp_wifi_start());
135
136      ESP_LOGI(WIFI_LOG_TAG, "wifi_init_sta finished.");
137
138      /* Waiting until either the connection is established
139         (WIFI_CONNECTED_BIT) or connection failed for the
140         maximum
141         * number of re-tries (WIFI_FAIL_BIT). The bits are set
142         by event_handler() (see above) */
143      EventBits_t bits =
144          xEventGroupWaitBits(s_wifi_event_group,
145              WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
146              pdFALSE,

```

```

134         pdFALSE ,
135         portMAX_DELAY) ;
136
137     /* xEventGroupWaitBits() returns the bits before the
138        call returned, hence we can test which event actually
139        happened. */
140     if (bits & WIFI_CONNECTED_BIT) {
141         ESP_LOGI(WIFI_LOG_TAG, "connected to ap SSID:%s
142                         password:%s",
143                         EXAMPLE_ESP_WIFI_SSID,
144                         EXAMPLE_ESP_WIFI_PASS);
145     } else if (bits & WIFI_FAIL_BIT) {
146         ESP_LOGI(WIFI_LOG_TAG, "Failed to connect to
147                         SSID:%s, password:%s",
148                         EXAMPLE_ESP_WIFI_SSID,
149                         EXAMPLE_ESP_WIFI_PASS);
150     } else {
151         ESP_LOGE(WIFI_LOG_TAG, "UNEXPECTED EVENT");
152     }
153 }
```

Листинг А.18 – Исходный код заголовочного файла клиента MQTT - mqtt.h

```

1 #ifndef MQTT_H
2 #define MQTT_H
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 #include "mqtt_client.h"
9
10 const char* const MQTT_LOG_TAG = "MQTT";
11
12 esp_mqtt_client_handle_t mqtt_app_start();
13
14 #ifdef __cplusplus
15 }
16 #endif
17
18 #endif /* ifndef MQTT_H */
```

Листинг А.19 – Исходный код заголовочного файла клиента MQTT - mqtt.h

```

1 #ifndef MQTT_HH
2 #define MQTT_HH
3
4 #include <string_view>
5 #include <string>
6
7 #include "mqtt_client.h"
8
9 class MQTTClient {
10 public:
11     MQTTClient(std::string_view base_topic);
12     ~MQTTClient();
13
14     // it currently makes no sense to copy or assign as the
15     // handle gets invalid on destruction
16     // for this to work the handle would have to be
17     // lifetime extended
18     MQTTClient(const MQTTClient& other) = delete;
19     MQTTClient& operator=(const MQTTClient& other) = delete;
20
21 private:
22     esp_mqtt_client_handle_t client_handle;
23     std::string base_topic;
24 };
25
26 #endif /* ifndef MQTT_HH */

```

Листинг А.20 – Исходный код заголовочного файла клиента MQTT - mqtt.c

```

1 #include "mqtt.h"
2
3 #include <stdio.h>
4 #include <stdint.h>
5 #include <stddef.h>
6 #include <string.h>
7 #include "esp_system.h"
8 #include "esp_event.h"
9
10 #include "freertos/FreeRTOS.h"
11 #include "freertos/task.h"

```

```

12 #include "freertos/sempwr.h"
13 #include "freertos/queue.h"
14
15 #include "lwip/sockets.h"
16 #include "lwip/dns.h"
17 #include "lwip/netdb.h"
18
19 #include "esp_log.h"
20 #include "mqtt_client.h"
21
22
23 void log_error_if_nonzero(const char *message, int
24 error_code)
25 {
26     if (error_code != 0) {
27         ESP_LOGE(MQTT_LOG_TAG, "Last error %s: 0x%x",
28                 message, error_code);
29 }
30
31 /**
32 * @brief Event handler registered to receive MQTT events
33 *
34 * This function is called by the MQTT client event loop.
35 *
36 * @param handler_args user data registered to the event.
37 * @param base Event base for the handler(always MQTT Base
38 * in this example).
39 * @param event_id The id for the received event.
40 * @param event_data The data for the event,
41 *                  esp_mqtt_event_handle_t.
42 */
43 void mqtt_event_handler(void *handler_args,
44 esp_event_base_t base, int32_t event_id, void
45 *event_data)
46 {
47     // Why did we have to modify the last format parameter
48     // from "%d" to "%ld"?
49     ESP_LOGD(MQTT_LOG_TAG, "Event dispatched from event
50             loop base=%s, event_id=%ld", base, event_id);
51     esp_mqtt_event_handle_t event = event_data;

```

```

45     esp_mqtt_client_handle_t client = event->client;
46     int msg_id;
47     switch ((esp_mqtt_event_id_t)event_id) {
48     case MQTT_EVENT_CONNECTED:
49         ESP_LOGI(MQTT_LOG_TAG, "MQTT_EVENT_CONNECTED");
50         break;
51     case MQTT_EVENT_DISCONNECTED:
52         ESP_LOGI(MQTT_LOG_TAG, "MQTT_EVENT_DISCONNECTED");
53         break;
54
55     case MQTT_EVENT_SUBSCRIBED:
56         ESP_LOGI(MQTT_LOG_TAG, "MQTT_EVENT_SUBSCRIBED",
57                  msg_id=%d", event->msg_id);
58         break;
59     case MQTT_EVENT_UNSUBSCRIBED:
60         ESP_LOGI(MQTT_LOG_TAG, "MQTT_EVENT_UNSUBSCRIBED",
61                  msg_id=%d", event->msg_id);
62         break;
63     case MQTT_EVENT_PUBLISHED:
64         /* ESP_LOGI(MQTT_LOG_TAG, "MQTT_EVENT_PUBLISHED,
65            msg_id=%d", event->msg_id); */
66         break;
67     case MQTT_EVENT_DATA:
68         ESP_LOGI(MQTT_LOG_TAG, "MQTT_EVENT_DATA");
69         printf("TOPIC=%.*s\r\n", event->topic_len,
70                event->topic);
71         printf("DATA=%.*s\r\n", event->data_len,
72                event->data);
73         break;
74     case MQTT_EVENT_ERROR:
75         ESP_LOGI(MQTT_LOG_TAG, "MQTT_EVENT_ERROR");
76         if (event->error_handle->error_type ==
77             MQTT_ERROR_TYPE_TCP_TRANSPORT) {
78             log_error_if_nonzero("reported from esp-tls",
79                                  event->error_handle->esp_tls_last_esp_err);
80             log_error_if_nonzero("reported from tls stack",
81                                  event->error_handle->esp_tls_stack_err);
82             log_error_if_nonzero("captured as transport's
83                                   socket errno",
84                                   event->error_handle->esp_transport_sock_errno);
85             ESP_LOGI(MQTT_LOG_TAG, "Last errno string

```

```

        ("%s" ,
strerror(event->error_handle->esp_transport_sock_e
76
77    }
78    break;
79 default:
80    ESP_LOGI(MQTT_LOG_TAG, "Other event id:%d",
81        event->event_id);
82    break;
83}
84
85 esp_mqtt_client_handle_t mqtt_app_start()
86 {
87     esp_mqtt_client_config_t mqtt_cfg = {
88         .broker.address.uri = CONFIG_BROKER_URL,
89     };
90
91     esp_log_level_set(MQTT_LOG_TAG, ESP_LOG_VERBOSE);
92
93 #if CONFIG_BROKER_URL_FROM_STDIN
94     char line[128];
95
96     if (strcmp(mqtt_cfg.broker.address.uri, "FROM_STDIN")
97         == 0) {
98         int count = 0;
99         printf("Please enter url of mqtt broker\n");
100        while (count < 128) {
101            int c = fgetc(stdin);
102            if (c == '\n') {
103                line[count] = '\0';
104                break;
105            } else if (c > 0 && c < 127) {
106                line[count] = c;
107                ++count;
108            }
109            vTaskDelay(10 / portTICK_PERIOD_MS);
110        }
111        mqtt_cfg.broker.address.uri = line;
112    } else {

```

```

113     ESP_LOGE(MQTT_LOG_TAG, "Configuration mismatch:
114         wrong broker url");
115     abort();
116 }
117
118 esp_mqtt_client_handle_t client =
119     esp_mqtt_client_init(&mqtt_cfg);
120 /* The last argument may be used to pass data to the
121    event handler, in this example mqtt_event_handler */
122 esp_mqtt_client_register_event(client,
123     ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
124 esp_mqtt_client_start(client);
125
126 return client;
127
128 }
```

Листинг А.21 – Исходный код файла реализации клиента MQTT - mqtt.cc

```

1
2 #include "esp_log.h"
3 #include "mqtt_client.h"
4
5 #include "mqtt.hh"
6 #include "mqtt.h"
7
8 MQTTCClient::MQTTCClient(std::string_view base_topic_input) :
9     client_handle(mqtt_app_start()),
10    base_topic(base_topic_input)
11 {};
12
13 MQTTCClient::~MQTTCClient() {
14     // Cleanup the mqtt client
15     ESP_ERROR_CHECK(esp_mqtt_client_destroy(client_handle));
16 }
17
18 int MQTTCClient::publish(std::string_view topic_suff,
19                         std::string_view message) {
20     std::string topic = std::string(base_topic) +
21         std::string(topic_suff);
22     int msg_id = esp_mqtt_client_publish(client_handle,
23                                         topic.data(), message.data(), 0, 1, 0);
24 }
```

```
21 |     ESP_LOGI(MQTT_LOG_TAG, "sent publish successful,  
22 |         msg_id=%d", msg_id);  
23 |     return msg_id;  
23 }
```

ПРИЛОЖЕНИЕ Б

Графическая часть

На 2 листах

Электрическая схема функциональная

Электрическая схема принципиальная