Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Кіберебезпеки_____
(повна назва)

Кафедра _____Інформаційно-мережної інженерії_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____перший (бакалаврський)_____
___Система моніторингу експлуатаційних показників в IoT-середовищі_____
(тема)

Виконав:
здобувач ____4____ року навчання,
групи ____ТРІМІ-22-2_____
_____Микита ГРІНЧЕНКО_____
(власне ім'я, прізвище)

Спеціальність ___172 Телекомунікації та_____
радіотехніка_____
(код і повна назва спеціальності)
Тип програми ____освітньо-професійна_____
Освітня програма __Інформаційно-мережна_
інженерія_____
(повна назва освітньої програми)

Керівник _____доц. Галина ЛЯШЕНКО____
(посада, власне ім'я, прізвище)

Допускається до захисту

В.о. завідувача кафедри _ІМІ__ _____ __Дарія ЧЕБОТАРЬОВА__
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Кібербезпеки

Кафедра Інформаційно-мережної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 172 Телекомунікації та радіотехніка
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформаційно-мережна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:

В.о. зав. кафедри _____
(підпис)
«____»_____ 20 26 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувачеві Грінченку Микиті Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Система моніторингу експлуатаційних показників в IoT-середовищі*

затверджена наказом університету від 02 червня 2026 р. № 466 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 13 липня 2026р.

3. Вихідні дані до роботи *Проаналізувати комерційні та науково-технічні рішення IoT моніторингу об'єктів водного транспорту; обґрунтувати архітектуру IoT-системи на базі мікроконтролера ESP32, протоколу MQTT; провести моделювання IoT-системи моніторінгу експлуатаційних показників та створити вебдашборд для візуалізації телеметрії; виконати симуляційне тестування системи у середовищі Wokwi.*

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області та формування вимог
2. Обґрунтування вибору технологій для реалізації системи
3. Моделювання апаратної частини системи
4. Створення та тестування програмної частини системи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
Слайди презентації у Power Point _____
_____
_____
_____
_____
_____
_____
_____
_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Ознайомлення із завданням. Уточнення ТЗ.	02.06.26 – 03.06.26	Виконано
2	Підбір літератури за темою роботи.	04.06.26 – 06.06.26	Виконано
3	Виконання розділу 1	08.06.26 – 11.06.26	Виконано
4	Виконання розділу 2	11.06.26 – 19.06.26	Виконано
5	Виконання розділу 3	20.06.26 – 30.06.26	Виконано
6	Виконання розділу 4	01.07.26 – 09.07.26	Виконано
7	Оформлення пояснювальної записки	10.07.26 – 13.07.26	Виконано
8	Оформлення презентаційного	14.07.26 – 16.07.26	Виконано
	матеріалу. Підготовка до захисту у ЕК		

Дата видачі завдання 02 червня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Галина ЛЯШЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 77 с., 4 табл., 11 рис., 20 джерел.

ІНТЕРНЕТ РЕЧЕЙ, ESP32, MQTT, МОНІТОРИНГ, ВОДНИЙ ТРАНСПОРТ, ТЕЛЕМЕТРІЯ, WOKWI, СЕНСОР, МІКРОКОНТРОЛЕР, АВТОМАТИЗАЦІЯ.

Об’єкт дослідження – процес моніторингу експлуатаційних показників в IoT-середовищі

Мета роботи: розроблення IoT-системи моніторингу експлуатаційних показників об’єкта водного транспорту, здатної в автономному режимі контролювати ключові параметри судна та передавати зібрані дані власнику незалежно від його фізичної присутності на борту.

Для досягнення поставленої мети в роботі розв’язано такі завдання: проведено аналіз предметної області та огляд наявних комерційних і науково-технічних рішень; сформовано перелік вимог до системи; обґрунтовано вибір мікроконтролерної платформи ESP32, протоколу передавання даних MQTT та середовища розробки й симуляції VS Code, PlatformIO, Wokwi; обґрунтовано вибір сенсорів та розроблено розподіл виводів мікроконтролера з урахуванням апаратних обмежень блоків АЦП; розроблено принципову схему підключення периферії; реалізовано програмне забезпечення мікроконтролера з неблокуючою архітектурою головного циклу, модулем збору даних; реалізовано публікацію телеметрії за протоколом MQTT та вебдашборд для її візуалізації в реальному часі; проведено тестування розробленої системи в середовищі симуляції Wokwi.

Практичне значення роботи полягає в тому, що розроблена система є економічно доступною альтернативою комерційним рішенням моніторингу об’єктів водного транспорту з відкритою, легко масштабованою архітектурою.

ABSTRACT

Explanatory note to the qualification thesis: 77 p., 4 tables, 11 figures, 20 sources.

INTERNET OF THINGS, ESP32, MQTT, MONITORING, WATER TRANSPORT, TELEMETRY, WOKWI, SENSOR, MICROCONTROLLER, AUTOMATION.

Object of research – the process of monitoring operational parameters in an IoT environment.

Purpose of the work: development of an IoT monitoring system for the operational parameters of a water transport vessel, capable of autonomously controlling key vessel parameters and transmitting the collected data to the owner regardless of their physical presence on board.

To achieve this purpose, the following tasks were solved in the work: an analysis of the subject area and a review of existing commercial and scientific-technical solutions were conducted; a list of system requirements was formed; the choice of the ESP32 microcontroller platform, the MQTT data transmission protocol, and the VS Code, PlatformIO, and Wokwi development and simulation environment was substantiated; the choice of sensors was substantiated and the microcontroller pin allocation was developed taking into account the hardware limitations of the ADC blocks; a schematic diagram for connecting the peripherals was developed; the microcontroller software was implemented with a non-blocking main-loop architecture and a data acquisition module; publication of telemetry via the MQTT protocol and a web dashboard for its real-time visualization were implemented; testing of the developed system was carried out in the Wokwi simulation environment.

The practical significance of the work lies in the fact that the developed system is a cost-effective alternative to commercial water transport monitoring solutions, featuring an open, easily scalable architecture.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 7
ВСТУП 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ 10
1.1 Аналіз проблеми та огляд існуючих рішень 10
1.2 Огляд науково-технічних публікацій за темою дослідження 11
1.3 Вимоги до системи 13
2 ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ 15
2.1 Вибір мікроконтролерної платформи 15
2.2 Вибір протоколу передавання даних 18
2.3 Вибір середовища розробки та симуляції 21
3 АПАРАТНА ЧАСТИНА СИСТЕМИ 23
3.1 Вибір сенсорів для контрольованих параметрів 23
3.2 Схема підключення периферії до мікроконтролера 28
3.3 Принципова схема підключення 30
4 ПРОГРАМНА ЧАСТИНА СИСТЕМИ 33
4.1 Загальна архітектура програмного забезпечення 33
4.2 Модуль збору даних з сенсорів та обробка вимірювань 35
4.3 Логіка прийняття рішень та керування виконавчим пристроєм 37
4.4 Реалізація MQTT-комунікації 39
4.5 Вебдашборд для візуалізації телеметрії 43
4.6 Результати тестування системи 46
ВИСНОВОК 48
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50
ДОДАТОК А 53
ДОДАТОК Б 60
ДОДАТОК В 71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ADC (Analog-to-Digital Converter) – аналого-цифровий перетворювач;

ARM (Advanced RISC Machine) – архітектура процесорних ядер зі скороченим набором команд, на основі якої побудовано мікроконтролери STM32;

CoAP (Constrained Application Protocol) – протокол прикладного рівня для обмежених за ресурсами пристроїв, що працює поверх UDP;

GPIO (General Purpose Input/Output) – вивід мікроконтролера загального призначення;

HTTP (HyperText Transfer Protocol) – протокол передавання гіпертексту, побудований за моделлю «запит-відповідь»;

IoT (Internet of Things) – інтернет речей;

MQTT (Message Queuing Telemetry Transport) – легкий протокол обміну повідомленнями за моделлю публікація-підписка через центральний брокер;

QoS (Quality of Service) – рівень якості обслуговування, що визначає гарантію доставки повідомлення;

TCP (Transmission Control Protocol) – протокол керування передаванням, що забезпечує гарантовану доставку даних;

UART (Universal Asynchronous Receiver-Transmitter) – універсальний асинхронний приймач-передавач, апаратний вузол для послідовного обміну даними;

UDP (User Datagram Protocol) – протокол дейтаграм користувача, що не гарантує доставку повідомлень;

USB (Universal Serial Bus) – універсальна послідовна шина;

Wi-Fi (Wireless Fidelity) – технологія бездротового мережевого з’єднання.

WSN (Wireless Sensor Network) – безпровідна сенсорна мережа;

ВСТУП

Розвиток технологій Інтернету речей (IoT) за останнє десятиліття кардинально змінив підходи до моніторингу та керування технічними об’єктами в найрізноманітніших галузях. Компактні мікроконтролерні платформи з підключеними сенсорами та модулями бездротового зв’язку сьогодні застосовуються для автоматизації «розумного» житла, контролю параметрів у сільському господарстві, промислового моніторингу обладнання в межах концепції Індустрії 4.0, а також у логістиці та транспортній галузі, де дистанційний контроль стану рухомих об’єктів дозволяє власнику отримувати актуальну інформацію незалежно від фізичної присутності поруч з об’єктом.

Судноплавство залишається однією з галузей, де подібний підхід досі впроваджено обмежено, а прийняття рішень значною мірою залежить від особистої присутності людини поруч з контрольованим вузлом. Власник чи екіпаж перевіряють стан судна візуально або за приладами на борту, а щойно людина залишає об’єкт – інформація про його стан стає невідомою до наступної перевірки. Технології IoT здатні змінити цю модель і в цій сфері: невеликий мікроконтролер з підключеними сенсорами та бездротовим зв’язком спроможний цілодобово збирати дані про стан судна та передавати їх власнику незалежно від того, чи перебуває хтось на борту в даний момент.

Ключову роль тут відіграє поява доступних мікроконтролерних платформ, таких як ESP32 та Arduino. Вони можуть мати вбудований чи підключений Wi-Fi-модуль, здатні одночасно обробляти сигнали з кількох датчиків і головна їхня перевага в тому, що вони коштують у рази менше за промислові контролери, що зазвичай використовуються в суднобудуванні. У парі з легким та надійним протоколом MQTT, розробленим саме для пристроїв з обмеженим і нестабільним каналом зв’язку, такі мікроконтролери дозволяють передавати телеметрію навіть за умов слабкого сигналу в марині чи відкритому морі.

Практична цінність подібної системи не лише у зборі даних, а в тому, що їх можна легко обробляти. Порогові значення для рівня води, заряду акумулятора чи температури дозволяють системі не тільки фіксувати показники, а й діяти автономно: вмикати помпу, коли рівень води перевищує безпечну межу, або надсилати власнику сповіщення про критичне падіння напруги чи палива.

В кваліфікаційній роботі розглянуто розробку IoT-системи моніторингу експлуатаційних показників об’єкта водного транспорту: обґрунтування вибору апаратних компонентів, архітектуру збору, обробки й передавання даних, а також реалізацію логіки прийняття рішень за допомогою мікроконтролера.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ

1.1 Аналіз проблеми та огляд існуючих рішень

Безпечна експлуатація об'єктів водного транспорту спирається на періодичний ручний контроль та обслуговування з боку власника або екіпажу: перевірку рівня води у трюмі, стану акумуляторних батарей, рівня палива та показників мікроклімату машинного відділення. Такий підхід має суттєвий недолік – відсутність контролю в періоди, коли судно перебуває без нагляду (стоянка в марині, міжсезонне зберігання на суходолі, тривалі переходи без постійного знаходження екіпажу в машинному відділенні, або обслуговуючий персонал відсутній). Це суттєво підвищує ризики, зокрема:

- затоплення внаслідок помилки закриття судового клапана або пошкодження корпусу;
- глибокого розряду акумуляторної батареї, що унеможливлює запуск двигуна та живлення критичних систем судна;
- вичерпання палива в непередбачений момент, особливо критичне при плаванні у віддалених районах;
- перегріву або підвищеної вологості в машинному відділенні, що прискорює корозію та може призводити до пожежонебезпечних ситуацій.

Розвиток концепції Інтернету речей (IoT) та здешевлення мікроконтролерних платформ з вбудованими бездротовими інтерфейсами створили сприятливі умови для автоматизації подібного контролю. Це збігається з загальним напрямом розвитку так званого Maritime 4.0 – впровадження цифрових технологій, безпровідних сенсорних мереж (WSN) та хмарних сервісів у морську галузь [1-3].

На комерційному ринку вже представлені готові рішення подібного класу, зокрема Siren Marine та Garmin OnDeck, орієнтовані переважно на яхти та катери середнього й високого цінового сегмента. Аналіз таких систем виявляє такі спільні недоліки:

- висока вартість обладнання – базові комплекти орієнтовані на преміум-сегмент і суттєво перевищують бюджет типового власника невеликого катера;
- закритість екосистеми – неможливість самостійно розширити перелік датчиків, змінити логіку обробки даних або інтегрувати систему зі сторонніми сервісами;
- обов'язкова платна підписка на використання фірмових хмарних серверів для доступу до телеметрії, що робить систему постійним джерелом витрат навіть після придбання обладнання, встановлення та налаштування;
- відсутність можливості локального розгортання – усі дані проходять через хмарну інфраструктуру виробника, що створює залежність від стабільності стороннього сервісу.

Ці обмеження роблять такі системи економічно невигідними для власників невеликих суден і катерів, а також обмежують гнучкість реалізації можливих нетипових вимог конкретного кожного з них. Альтернативою є побудова системи моніторингу на базі відкритих апаратно-програмних платформ Arduino, ESP32 та із застосуванням відкритого протоколу обміну даними MQTT, що дозволяють реалізувати необхідний функціонал зі значно меншими фінансовими витратами, повним контролем над архітектурою та можливістю подальшого масштабування кількості контрольованих параметрів за допомогою різноманітних датчиків [4, 5].

1.2 Огляд науково-технічних публікацій за темою дослідження

Дослідження застосування IoT-технологій для моніторингу водного транспорту в науковій літературі представлені зазвичай роботами практичного та галузевого характеру, що розглядають моніторинг суднових систем у контексті цифровізації морської галузі та експлуатації суднових двигунів [1-3]. Водночас конкретні технічні рішення на базі мікроконтролерів для моніторингу

параметрів саме водного транспорту представлені обмежено, що підтверджує актуальність розробки подібної системи як предмета дипломного дослідження.

Значно ширше в літературі представлені дослідження IoT-моніторингу технічних об'єктів поза морською сферою, архітектурні рішення яких можуть бути адаптовані до задач аналізу телеметрії суден. Зокрема, в одній з робіт детально розглянуто процес проєктування та реалізації IoT-пристроїв на базі мікроконтролера ESP32, зокрема підключення аналогових і цифрових датчиків, таких як температури, вологості, та забезпечення збору й передавання даних на хмарну платформу. Схожа архітектура та алгоритми автоматизації зустрічаються й в інших галузевих дослідженнях, в яких описано систему моніторингу гідропонної установки на основі ESP32 з кількома аналоговими датчиками: рівень води, температура, показники хімічного розчину та автоматичним керуванням насосом [6, 7].

Окремий інтерес для дослідження становить застосування симуляційних середовищ на етапі проєктування та тестування прошивок для мікроконтролерів. Симулятор Wokwi забезпечує повноцінне віртуальне середовище для розробки, налагодження та перевірки коду мікроконтролерних систем без використання фізичного обладнання, він підтримує широкий спектр плат: ESP32, Arduino, STM32 та периферійних пристроїв, а також симулює Wi-Fi з’єднання в реальному часі. Дослідження застосування платформи Wokwi в інженерній розробці та освіті підтверджує, що використання симулятора дозволяє проводити безліч експериментів без апаратних обмежень та ризиків для справжнього обладнання, а віртуальне тестування дає змогу зосередитись на логіці програми та оптимізації – що обґрунтовує застосування Wokwi як інструмент для розробки системи в межах цього дослідження [8].

1.3 Вимоги до системи

На основі аналізу проблеми та огляду наявних науково-технічних дослідів та реалізацій сформовано перелік вимог, яким повинна відповідати розроблювана система моніторингу експлуатаційних показників об'єкта водного транспорту.

Функціональні вимоги:

- система повинна забезпечувати моніторинг температури та вологості повітря в машинному відділенні у режимі реального часу;
- система повинна забезпечувати контроль рівня води у трюмі та ініціювати сигнал тривоги при перевищенні допустимого значення;
- система повинна автоматично активувати відкачування води у разі перевищення критичного рівня, без необхідності втручання оператора;
- система повинна контролювати напругу бортової акумуляторної батареї та сигналізувати про її критичне зниження, щоб запобігти глибокому розряду;
- система повинна контролювати рівень палива в баку та сигналізувати про його критичне зниження;
- система повинна передавати зібрані дані телеметрії на віддалений сервер для забезпечення можливості дистанційного спостереження за станом об’єкта;
- система повинна формувати та надсилати сповіщення про позаштатну ситуацію окремо від звичайної телеметрії при виході контрольованих параметрів за встановлені порогові значення.

Нефункціональні вимоги:

- економічна доступність – вартість апаратної реалізації повинна бути суттєво нижчою за вартість комерційних аналогів;
- відкритість архітектури – система повинна допускати подальше масштабування без кардинальної переробки програмної та апаратної частини;

- автономність функціонування – система повинна збирати дані і приймати рішення без постійної присутності оператора;
- надійність передавання даних – система повинна коректно обробляти тимчасові розриви мережевого з’єднання та автоматично відновлювати підключення до сервера;
- періодичність оновлення даних – оновлення даних повинно відбуватися з встановленим фіксованим інтервалом, якого буде достатньо для своєчасного виявлення позаштатних ситуацій та їх контролю;
- можливість перевірки дієздатності системи на етапі проєктування без використання фізичного прототипу.

Наведений перелік вимог формує основу для подальшого проєктування. Функціональні вимоги визначають, які саме параметри судна підлягають контролю та яка логіка реагування системи на позаштатні ситуації, тоді як нефункціональні – задають критерії, за якими оцінюватиметься якість технічної реалізації.

Окремої уваги потребує вимога щодо автономності функціонування. Має бути реалізовано механізм який дає змогу прийняття рішень безпосередньо мікроконтролером зокрема, керування відкачуванням води при перевищенні порогового значення, тобто критичні для безпеки судна дії не мають залежати від наявності стабільного мережевого з’єднання чи доступності віддаленого сервера. Такий підхід підвищує відмовостійкість системи в цілому, оскільки функція миттєвого реагування зберігається навіть у разі тимчасової втрати зв’язку, а такі функції як передавання телеметрії та формування сповіщень є додатковою, але не критичною для безпеки.

2 ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Вибір мікроконтролерної платформи

Вибір мікроконтролерної платформи є важливим етапом підготовки, що впливає на всю подальшу розробку: перелік доступних периферійних інтерфейсів, спосіб реалізації бездротового зв’язку, вартість готового пристрою та складність програмної реалізації. Відповідно до сформульованих нефункціональних вимог, таких як економічна доступність, відкритість архітектури, можливість перевірки без фізичного прототипу, як основні кандидати для порівняння розглянуто три поширені платформи: Arduino, STM32 та ESP32.

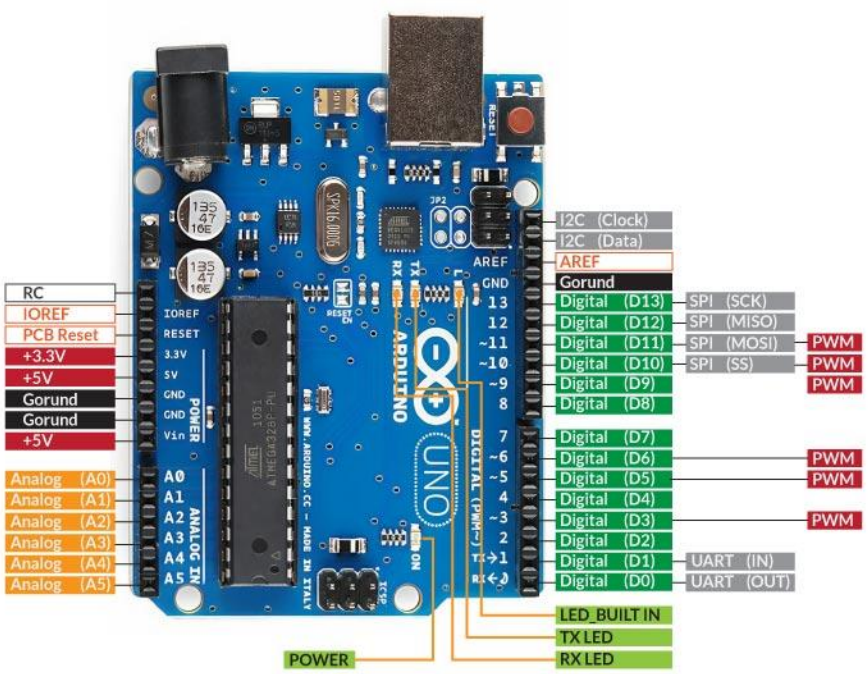


Рисунок 2.1 – Мікроконтролер Arduino Uno

Класичні плати Arduino (Uno, Nano, Mega) найлегші для освоєння та мають велику базу бібліотек, однак не мають вбудованого модуля бездротового зв’язку, а їх обчислювальна потужність суттєво нижча ніж у 32-бітних рішень. Для

системи, що має самостійно передавати телеметрію по Wi-Fi без додаткових модулів та кабелів, базові моделі Arduino вимагали б встановлення окремого радіомодуля, що ускладнює апаратну частину та підвищує вартість реалізації (рис.2.1) [9].

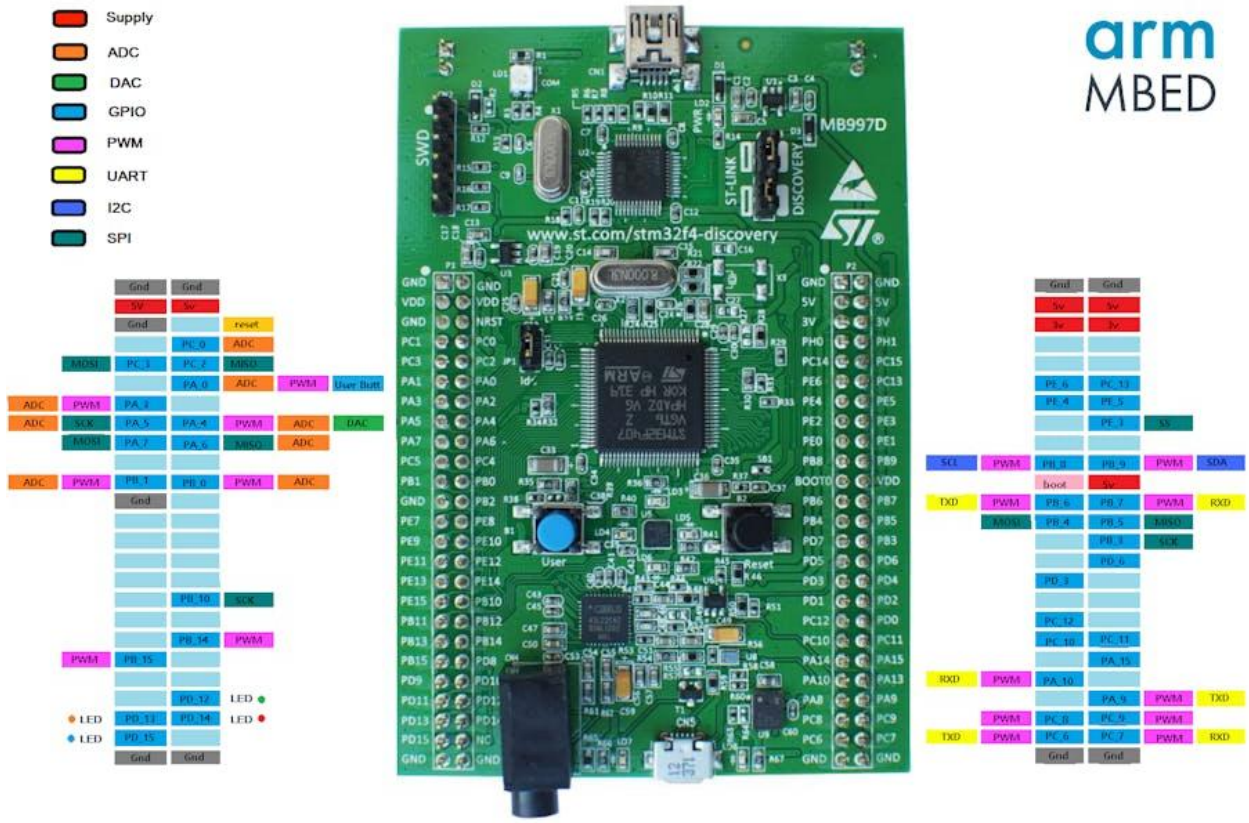


Рисунок 2.2 – Мікроконтролер STM32F4 Discovery

Мікроконтролери STM32 на ядрах ARM Cortex-M забезпечують високу точність таймерів, малу затримку переривань та якісні аналого-цифрові перетворювачі з розрядністю 12-16 біт, що робить їх оптимальним вибором для задач промислової автоматики та точного керування в реальному часі. Водночас STM32 не має вбудованого бездротового модуля, тому для підключення Wi-Fi чи Bluetooth потрібен окремий модуль з радіочіпем, що додає вартості, ускладнює друковану плату та вимагає додаткової інсталяції драйверів для правильного функціонування. Реалізована система першочергово орієнтована на бездротову передачу телеметрії та легкий монтаж, а не на високу точність

обчислень, ці можливості STM32 є некритичними відносно поставлених вимог (рис.2.2) [10, 11].

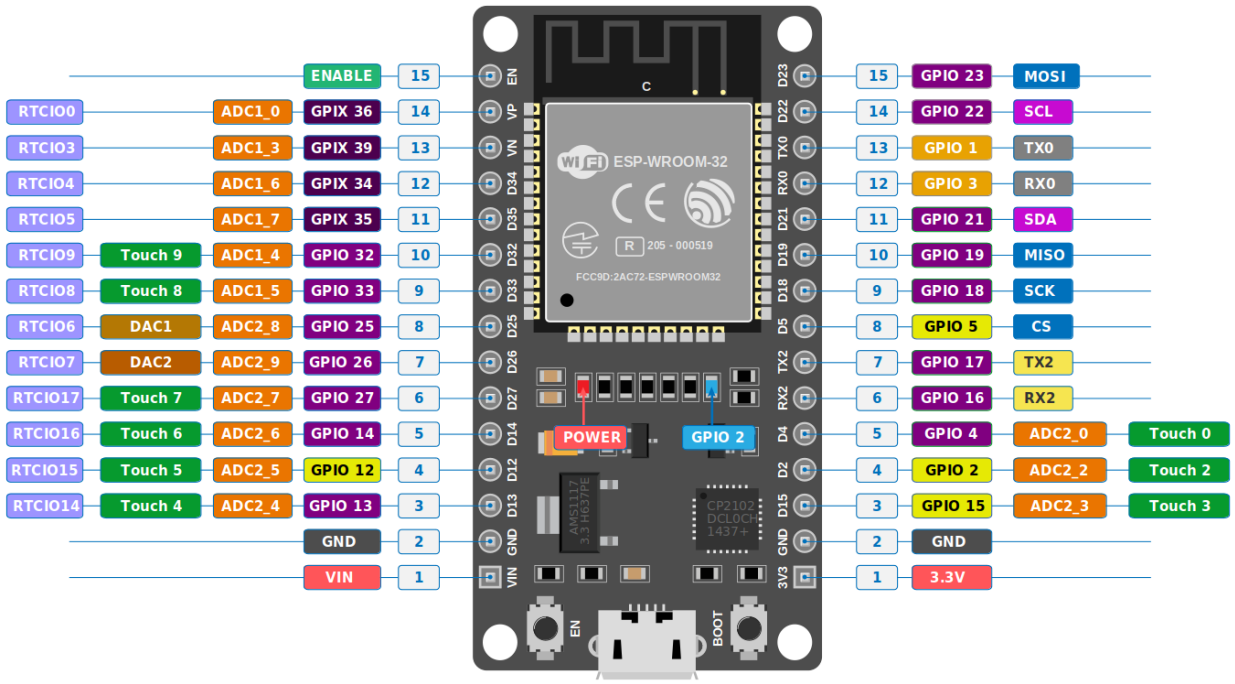


Рисунок 2.3 – Мікроконтролер ESP32 з підтримкою бездротових технологій Wi-Fi та Bluetooth

Платформа ESP32, на відміну від розглянутих альтернатив, поєднує двоядерний 32-бітний процесор із вбудованими модулями Wi-Fi та Bluetooth, що усуває необхідність в окремих радіомодулях і дозволяє напряму підключатись до локальної мережі чи віддаленого сервера. Одне з ядер процесора може обслуговувати функцію бездротового зв'язку, а інше в цей час виконуватиме прикладний код, що забезпечує стабільну одночасну роботу мережевої частини та логіки збору і обробки даних із датчиків. Крім цього, ESP32 підтримує режими глибокого сну, що економить енергію, а коштує вона, як правило, до 450 гривень за плату розробника, що напряму відповідає критерію економічної доступності. ESP32 має велику спільноту розробників, сумісність із Arduino IDE та фреймворком PlatformIO, ці переваги дають змогу знизити складність та часові витрати реалізації порівняно з складними промисловими середовищами розробки STM32 (рис.2.3) [10, 11, 12].

Фінальне порівняння розглянутих платформ за визначеними критеріями системи наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння мікроконтролерних платформ

Критерій	Arduino (Uno/Nano)	STM32	ESP32
Вбудований Wi-Fi	Ні	Ні	Так
Розрядність	8-біт	32-біт	32-біт
Точність АЦП і таймерів	Низька	Висока	Середня
Орієнтовна вартість плати	Низька	Середня-висока	Низька
Поріг входу	Низький	Високий	Низький–середній
Доцільність використання в готовому пристрої	Недостатні можливості та відсутність вбудованого радіомодуля	Надлишкова складність та відсутність вбудованого радіомодуля	Оптимальна – відповідає вимогам системи

Отже, з огляду на необхідність автономно та бездротово передавати дані, а також ціль максимально скоротити бюджет реалізації, платформою для реалізації системи обрано ESP32 DevKit V1.

2.2 Вибір протоколу передавання даних

Протокол передавання телеметрії напряму впливає на надійність зв’язку, обсяг споживання трафіку та енергоефективність пристрою. Ці параметри критичні для об’єкта водного транспорту, який може перебувати в зоні

нестабільного покриття мережі. Для порівняння розглянуто три поширені у сфері IoT протоколи прикладного рівня: HTTP, CoAP та MQTT.

HTTP побудований на так званій моделі «запит-відповідь» на базі протоколу TCP і є найбільш універсальним, використовуваним та підтримуваним протоколом, проте його текстові заголовки порівняно громіздкі: типовий HTTP-запит додає не менше кілька сотень байтів службової інформації, навіть для того, щоб передати одне невелике значення, проте за іншими протоколами, повідомлення може обходитись значно меншим обсягом даних. Крім того, HTTP не має вбудованого механізму оновлень, отже кожен клієнт, якому потрібні актуальні дані, повинен самостійно опитувати сервер, що є неефективним для періодичної телеметрії [13].

CoAP розроблявся спеціально для обмежених пристроїв і працює поверх UDP замість TCP, що суттєво знижує затримку встановлення з'єднання порівняно з HTTP, хоча зворотним боком є менша надійність доставки повідомлень, що є характерним для протоколів на основі UDP. У деяких дослідженнях CoAP демонструє нижче споживання трафіку порівнюючи з MQTT саме для невеликих корисних навантажень, проте ефективність CoAP залежить від конкретного сценарію й падає зі зростанням кількості переданих даних. Ще один суттєвий недолік CoAP, особливо для розробки та тестування в симуляторі – відсутність простих загальнодоступних публічних тестових серверів, на відміну від MQTT, а також менша кількість бібліотек в екосистемі Arduino/PlatformIO [13, 14].

MQTT, на відміну від двох попередніх варіантів, побудований на моделі публікація-підписка (publish-subscribe) через центральний брокер, що дозволяє одному пристрою публікувати дані одразу для довільної кількості підписників без необхідності встановлювати з'єднання з кожним із них окремо. Протокол спроєктований саме для середовищ з обмеженою пропускною здатністю, можливою високою затримкою та нестабільним підключенням. За даними знайдених досліджень, продуктивність обох протоколів залежить від конкретних умов мережі: при низькому рівні втрати пакетів MQTT демонструє нижчу

затримку доставки повідомлень порівняно з CoAP, тоді як за високого рівня втрат, приблизно від 15-20% ситуація змінюється на протилежну через більші накладні витрати TCP який використовується протоколом MQTT, при повторній передачі порівняно з UDP. Проте, при малих розмірах повідомлення, під час передачі телеметрії з датчиків, MQTT з увімкненим підтвердженням доставки QoS 1-2 генерує більший обсяг службового трафіку на одне повідомлення, ніж CoAP, оскільки в QoS 2 використовується чотирьохетапне підтвердження, яке додає службові дані незалежно від розміру корисних даних. Для системи що розробляється, де мережеве з'єднання Wi-Fi вважається відносно стабільним, а критичність доставки кожного окремого показника вища за економію трафіку, обрано рівень QoS 1, що гарантує доставку повідомлення щонайменше один раз [13, 15].

Узагальнене порівняння наведено в таблиці 2.2.

Таблиця 2.2 – Порівняння протоколів передавання даних

Критерій	HTTP	CoAP	MQTT
Транспортний рівень	TCP	UDP	TCP
Модель взаємодії	Запит-відповідь	Запит-відповідь	Публікація-підписка
Накладні витрати на повідомлення	Високі	Низькі	Низькі
Підтримка кількох підписників	Не передбачена	Обмежена	За замовчуванням
Гарантія доставки (QoS)	Немає	Обмежена	Так (QoS 1-2)
Доступність публічних брокерів та бібліотек для ESP32	Висока	Низька	Висока

Отже, з огляду на модель публікація-підписка, здатність MQTT забезпечувати нижчу затримку доставки за стабільного з'єднання з малою

кількістю втрачених пакетів, наявність гнучких рівнів QoS, а також нижче енергоспоживання порівняно з HTTP при тривалому моніторингу в реальному часі, для передавання телеметрії обрано протокол MQTT з використанням публічного брокера test.mosquitto.org [15, 16].

2.3 Вибір середовища розробки та симуляції

Розробка вбудованого програмного забезпечення передбачає цикл написання коду, прошивку мікроконтролера та перевірку на фізичному стенді, що для складної системи з кількома датчиками є трудомістким і уповільнює процес доопрацювання, а за відсутності фізичного макета взагалі унеможливлюється перевірка логіки програми. Тому розглядаються не лише мови програмування, а й повний набір засобів розробки: середовища написання коду, системи збірки прошивки та засоби попередньої перевірки без фізичного обладнання.

Середовище написання коду. Для розробки прошивки використано Visual Studio Code – розширюваний редактор коду з підтримкою інструментів для вбудованих систем. Порівняно із класичним Arduino IDE, який має обмежені можливості автодоповнення коду, налагодження та керування залежностями, VS Code у поєднанні з відповідними розширеннями забезпечує повноцінне середовище розробки: підсвічування синтаксису, інтеграцію з системою контролю версій Git та єдиний робочий простір для коду прошивки, файлів конфігурації симуляції та схеми підключення.

Система збірки прошивки. Безпосередньо збірку прошивки для ESP32 виконано за допомогою розширення PlatformIO для VS Code – крос-платформової системи керування залежностями та збірки для вбудованих проєктів. На відміну від Arduino IDE, PlatformIO автоматично завантажує потрібні версії фреймворку Arduino core та бібліотек, зафіксовані у файлі конфігурації platformio.ini, що гарантує відтворюваність збірки на будь-якому комп’ютері незалежно від того, які бібліотеки встановлені локально.

Саме це дозволяє точно зафіксувати версію платформи та бібліотек для проєкту [17].

Засіб симуляції. Для перевірки логіки роботи системи без фізичного стенду використано Wokwi – онлайн-симулятор електронних схем, що підтримує апаратну симуляцію мікроконтролера ESP32 разом з підключеною периферією, а також симуляцію Wi-Fi-з'єднання, яка дозволяє перевіряти реальну мережеву взаємодію прошивки, зокрема підключення до точки доступу та обмін даними за протоколом MQTT, без розгортання фізичної мережі. Ключовою перевагою Wokwi є розширення Wokwi для VS Code, яке інтегрує симулятор безпосередньо в те саме середовище розробки, де ведеться написання коду, це дає змогу запускати симуляцію без перемикання у браузер, використовуючи вже зібрану PlatformIO прошивку, шлях до якої вказано у файлі конфігурації wokwi.toml. Схема підключення периферії до мікроконтролера при цьому описується у файлі diagram.json, що дозволяє робити різні версії схем підключення разом із кодом у тому самому репозиторії Git [12, 18].

Порівняно з альтернативними симуляторами електронних схем, такими як Tinkercad Circuits чи Proteus, Wokwi має суттєву перевагу саме для цього проєкту: Tinkercad не підтримує симуляцію Wi-Fi та мережевого підключення ESP32, а Proteus, попри можливість точного моделювання аналогових схем, є комерційним продуктом і не має вбудованої інтеграції з PlatformIO та VS Code, що ускладнило процес розробки.

Таким чином, комбінація VS Code, PlatformIO та Wokwi для VS Code дозволяє побудувати єдиний робочий процес розробки в межах одного середовища, що охоплює етапи від написання коду до перевірки мережевої взаємодії за протоколом MQTT, без потреби у фізичному прототипі на етапі розробки та налагодження логіки системи.

3 АПАРАТНА ЧАСТИНА СИСТЕМИ

3.1 Вибір сенсорів для контрольованих параметрів

Відповідно до нефункціональних вимог, сформульованих у розділі 1, система моніторингу має відстежувати чотири групи параметрів об'єкта водного транспорту: мікрокліматичні умови машинного відділення (а саме температура та вологість), рівень води в трюмі, рівень заряду бортового акумулятора та рівень палива в баку, а також забезпечувати автоматичне керування помпою. Для кожної групи параметрів обґрунтовано вибір відповідного сенсора або способу вимірювання.

Сенсор температури та вологості. Для контролю мікрокліматичних умов машинного відділення розглянуто два поширені цифрові сенсори: DHT11 та DHT22. Обидва сенсора побудовані на основі ємнісного елемента вимірювання вологості та термістора для вимірювання температури, вони під’єднуються до мікроконтролера через єдину цифрову лінію даних та мають вбудований аналого-цифровий перетворювач, що не потребує додаткових зовнішніх компонентів для оцифрування сигналу. Однак характеристики цих сенсорів суттєво відрізняються: DHT11 забезпечує точність вимірювання температури ±2°C у діапазоні від 0 до 50°C та вологості ±5% у діапазоні від 20 до 80% відносної вологості, тоді як DHT22 забезпечує точність ±0,5 °C у суттєво ширшому діапазоні від -40 до 80°C та точність вимірювання вологості ±2% у повному діапазоні від 0 до 100%. Оскільки машинне відділення судна є більш вимогливим середовищем з можливими коливаннями температури, а точність вимірювання безпосередньо впливає на своєчасність реагування на позаштатні ситуації, для реалізації системи обрано сенсора DHT22, попри дещо вищу вартість порівняно з DHT11, він забезпечує суттєво кращі метрологічні характеристики (рис.3.1) [19].



Рисунок 3.1 – Сенсор DHT22

Сенсор рівня води в трюмі. У реальних умовах для вимірювання рівня води в трюмі поширеним рішенням є резистивний сенсор рівня води – плата з паралельними мідними доріжками, опір між якими змінюється залежно від глибини занурення у воду, що дозволяє отримати аналоговий сигнал, пропорційний рівню рідини (рис.3.2).

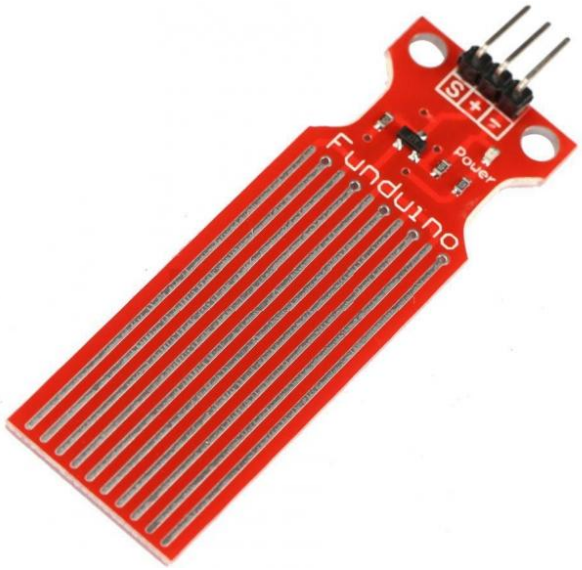


Рисунок 3.2 – Сенсор рівня води

Такий сенсор видає аналоговий сигнал у діапазоні від 0 до 3,3/5 В залежно від глибини занурення, який зчитується безпосередньо аналого-цифровим перетворювачем мікроконтролера. У межах дипломного проєкту, що передбачає розробку та перевірку системи в середовищі симуляції Wokwi (яке не підтримує моделювання занурення у рідину), як імітатор електричного інтерфейсу цього сенсора використано потенціометр, підключений до одного з ADC-входів мікроконтролера. Такий підхід дозволяє точно відтворити електричний інтерфейс реального сенсора та перевірити логіку спрацювання аварійного сповіщення й автоматичного керування помпою без потреби у фізичному макеті.

Сенсор рівня заряду акумулятора. Пряме підключення бортової мережі напругою 12 В або 24 В до аналогового входу мікроконтролера є неприпустимим, оскільки максимально допустима напруга на вході ADC ESP32 становить 3,3 В. Для безпечного вимірювання високої напруги стандартним рішенням є резистивний дільник напруги – два послідовно з’єднані резистори, що знижують вимірювану напругу до діапазону, придатного для ADC. Практичною реалізацією цього принципу є готовий модуль Voltage Sensor, що являє собою плату з дільником напруги з коефіцієнтом поділу 1:5, який дозволяє вимірювати вхідну напругу в діапазоні від 0 до 25 В, перетворюючи її на вихідний сигнал від 0 до 5 В, придатний для подачі на аналоговий вхід мікроконтролера. Модуль має клемники для підключення до вимірюваної ділянки кола та триконтактний вихід (живлення, земля та аналоговий сигнал), що спрощує інтеграцію з мікроконтролером без додаткового монтажу резисторів. У середовищі симуляції електричний інтерфейс такого дільника напруги відтворено за допомогою потенціометра, підключеного до окремого аналогового входу мікроконтролера (рис.3.3).



Рисунок 3.3 – Сенсор напруги

Сенсор рівня палива. Вимірювання рівня палива в баку в реальних умовах здійснюється сенсором поплавкового типу з резистивним аналоговим виходом. Конструктивно такий сенсор являє собою металевий стрижень, що монтується у горловину паливного бака через кільцевий фланець з ущільнювальною прокладкою, вздовж якого переміщується поплавок, з’єднаний з рухомим контактом реостата всередині корпусу сенсора. Під час зміни рівня палива поплавок піднімається або опускається разом зі стрижнем, змінюючи положення контакту реостата, а отже і вихідний опір та напругу сигналу пропорційно до рівня рідини в баку. Сенсор підключається двома провідними виводами до вимірювального кола, аналогічного дільнику напруги. У середовищі симуляції електричний інтерфейс такого сенсора так само відтворено за допомогою потенціометра, підключеного до окремого аналогового входу мікроконтролера (рис.3.4).

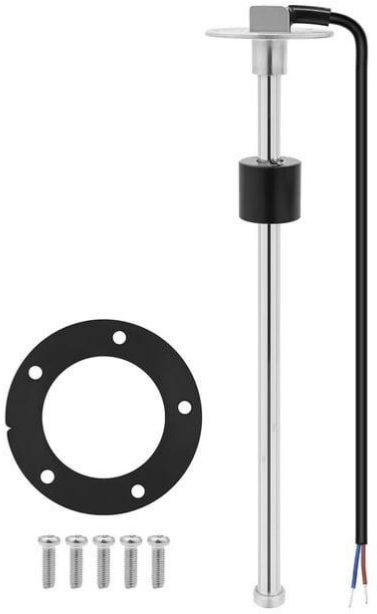


Рисунок 3.4 – Сенсор рівня палива

Виконавчий пристрій керування помпою. Для автоматичного вмикання трюмної помпи при перевищенні порогового рівня води використано релейний модуль, керований цифровим сигналом з мікроконтролера. Реле забезпечує гальванічну розв’язку між низьковольтною логікою мікроконтролера та силовим колом живлення помпи, що може працювати від бортової мережі 12 В або 24 В. У середовищі симуляції візуальну індикацію стану реле відтворено за допомогою світлодіода, підключеного через обмежувальний резистор до нормально розімкнутого контакту реле (рис.3.5).



Рисунок 3.5 – Релейний модуль

Узагальнені відомості про обрані компоненти та відповідні контрольовані параметри наведено в таблиці 3.1.

Таблиця 3.1 – Відповідність сенсорів контрольованим параметрам

Контрольований параметр	Реальний сенсор	Імітація у симуляції Wokwi	Тип сигналу
Температура, вологість машинного відділення	DHT22	DHT22	Цифровий
Рівень води в трюмі	Резистивний сенсор рівня води	Потенціометр	Аналоговий (ADC)
Напруга бортового акумулятора	Резистивний дільник напруги	Потенціометр	Аналоговий (ADC)
Рівень палива в баку	Поплавковий сенсор рівня палива	Потенціометр	Аналоговий (ADC)
Керування трюмною помпою	Релейний модуль + помпа	Релейний модуль + світлодіод	Цифровий вихід

3.2 Схема підключення периферії до мікроконтролера

Розподіл виводів (пінів) мікроконтролера ESP32 DevKit V1 між периферійними пристроями виконано з урахуванням трьох основних критеріїв: відповідності типу сигналу можливостям конкретного виводу (цифровий або аналоговий), уникнення виводів із спеціальним призначенням при старті мікроконтролера (strapping pins), а також зручності електричного та фізичного розведення провідників при подальшому монтажі на друковану чи макетну плату.

Аналогові входи (ADC). ESP32 має два незалежні блоки аналого-цифрового перетворювача – ADC1 (виводи GPIO32–GPIO39) та ADC2 (виводи GPIO0, GPIO2, GPIO4, GPIO12–GPIO15, GPIO25–GPIO27). Принципова

відмінність між ними полягає в тому, що блок ADC2 стає недоступним під час активного використання Wi-Fi, оскільки апаратно розділяє ресурси з радіомодулем. Оскільки розроблена система безперервно підтримує Wi-Fi-з'єднання для передавання телеметрії за протоколом MQTT, усі три аналогові сенсори (рівень води, напруга акумулятора, рівень палива) підключено виключно до виводів блоку ADC1: GPIO34, GPIO35 та GPIO32 відповідно. Виводи GPIO34 та GPIO35 додатково не працюють на вихід (input-only), але це не є обмеженням для даної задачі, оскільки всі три виводи використовуються виключно для зчитування аналогового сигналу [20].

Цифрові виводи. Сенсор температури й вологості DHT22 підключено до виводу GPIO15. Цей вивід за специфікацією мікроконтролера належить до каналу ADC2_3, проте в розробленій прошивці використовується виключно в цифровому режимі, без виклику функцій аналого-цифрового перетворення, тому обмеження блоку ADC2 щодо роботи з Wi-Fi на нього не поширюється. Керування релейним модулем трюмної помпи здійснено через вивід GPIO4 – цифровий вихід загального призначення, налаштований у прошивці в режим OUTPUT; так само як і GPIO15, цей вивід відноситься до ADC2, проте використовується виключно як цифровий, тому конфлікту з Wi-Fi не виникає.

Виводи живлення та землі. Живлення сенсорів (DHT22, потенціометри) здійснюється від виводу 3V3 мікроконтролера, що відповідає напрузі логічних рівнів ESP32 та номінальній напрузі живлення DHT22 (3–5,5 В). Реле за специфікацією потребує більший струм та стабільне джерело живлення, тому підключено до виводу VIN, який на платі ESP32 DevKit V1 з'єднаний безпосередньо з USB-живленням (5 В), минаючи внутрішній стабілізатор напруги 3,3 В. Оскільки плата ESP32 DevKit V1 має лише два фізичні виводи землі (GND.1, GND.2), а периферійних пристроїв, що потребують заземлення, більше двох, у розробленій схемі використовуються спільні вузли з'єднання, що об'єднують кілька виводів на один фізичний контакт мікроконтролера, що дозволяє під'єднати всі виводи GND периферії до обмеженої кількості фізичних контактів заземлення мікроконтролера.

Зведену відповідність виводів мікроконтролера підключеній периферії наведено в таблиці 3.2.

Таблиця 3.2 – Розподіл виводів мікроконтролера ESP32 DevKit V1

Периферійний пристрій	Вивід ESP32	Тип сигналу	Блок ADC
DHT22 (дані)	GPIO15	Цифровий	-
Сенсор рівня води (потенціометр)	GPIO34	Аналоговий	ADC1
Сенсор напруги акумулятора (потенціометр)	GPIO35	Аналоговий	ADC1
Сенсор рівня палива (потенціометр)	GPIO32	Аналоговий	ADC1
Реле керування помпою	GPIO4	Цифровий вихід	-

Такий розподіл виводів забезпечує стабільну одночасну роботу всіх аналогових сенсорів під час активного Wi-Fi-з’єднання, оскільки жоден з них не залежить від заблокованого під час роботи радіомодуля блоку ADC2.

3.3 Принципова схема підключення

На основі обґрунтованого в підрозділах 3.1–3.2 переліку компонентів та розподілу виводів мікроконтролера розроблено принципову схему підключення периферії до ESP32 DevKit V1. Схему реалізовано та перевірено в середовищі симуляції Wokwi, де конфігурація підключень описана у файлі diagram.json, з точним зазначенням типу кожного компонента, його розташування та переліку електричних з’єднань між виводами (рис.3.6).

Живлення периферії. Сенсор DHT22 та всі три потенціометри (імітатори сенсорів рівня води, напруги акумулятора та рівня палива) отримують живлення від виводу 3V3 мікроконтролера, що відповідає як напрузі логічних рівнів ESP32, так і номінальній напрузі живлення цих компонентів. Релейний модуль підключено до виводу VIN (5 В), оскільки котушка реле в більшості поширених

модулів розрахована на напругу живлення 5 В для надійного спрацювання. Усі периферійні пристрої з'єднано зі спільними шинами заземлення мікроконтролера.

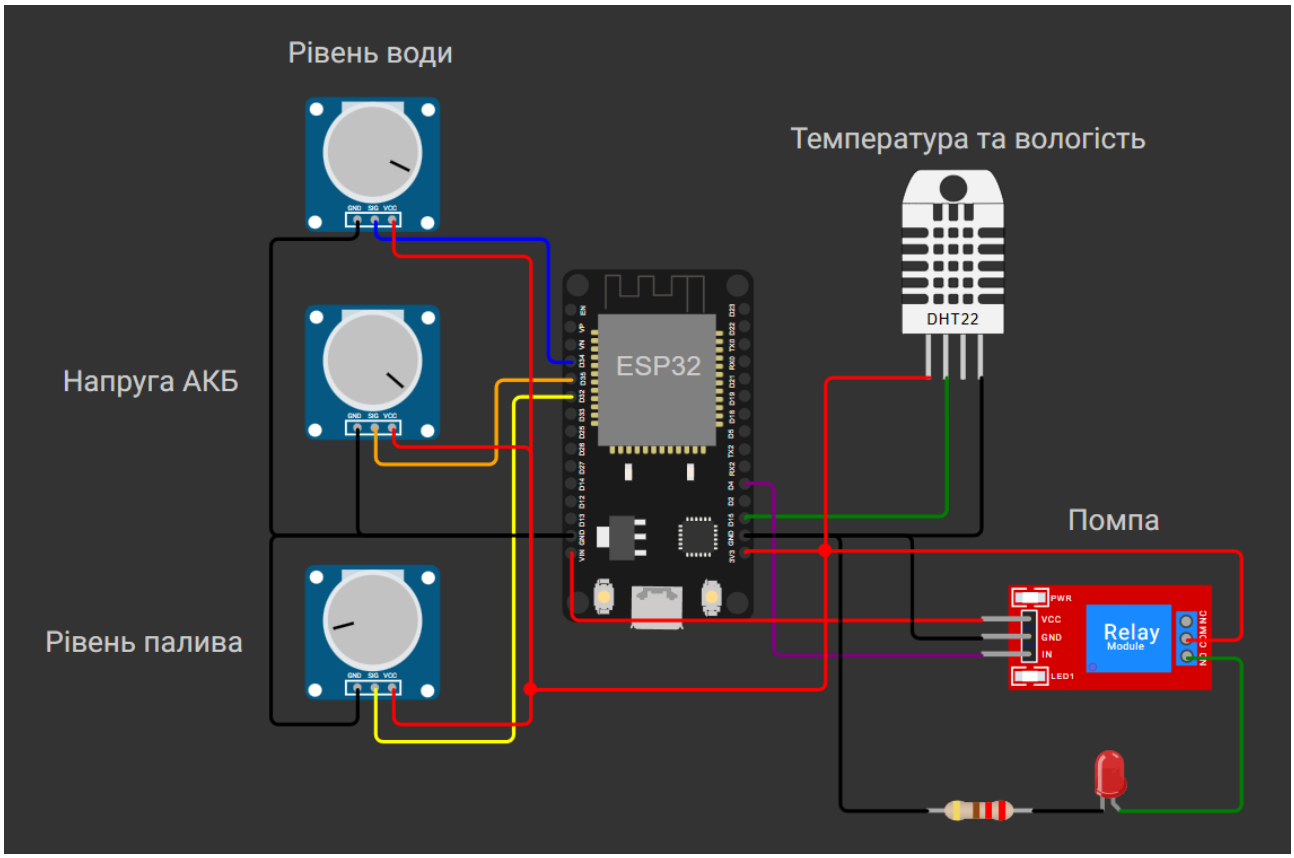


Рисунок 3.6 – Принципова схема підключення периферії до ESP32 DevKit V1 (середовище симуляції Wokwi)

Сигнальні з’єднання. Кожен сенсор з’єднано з відповідним виводом мікроконтролера одним сигнальним провідником:

- DHT22 (вивід SDA) до GPIO15;
- сенсор рівня води (потенціометр, вивід SIG) до GPIO34;
- сенсор напруги акумулятора (потенціометр, вивід SIG) до GPIO35;
- сенсор рівня палива (потенціометр, вивід SIG) до GPIO32;
- керуючий сигнал реле (вивід IN) до GPIO4.

Окремо від логічного кола керування реле, що працює на стороні мікроконтролера, у схемі передбачено силове коло індикації спрацювання помпи: нормально розімкнутий контакт реле (NO) з’єднано зі світлодіодом через обмежувальний резистор номіналом 220 Ом, який захищає світлодіод від надлишкового струму, а спільний контакт реле (COM) підключено до джерела живлення 3,3 В. У реальній реалізації системи це коло замінюється на силове коло живлення трюмної помпи від бортової мережі судна, а логіка керування реле з боку мікроконтролера залишається незмінною.

Для виведення діагностичних повідомлень та даних телеметрії в текстовому вигляді використано послідовний порт мікроконтролера (виводи TX0/RX0). У середовищі симуляції Wokwi цей порт підключено до вбудованого віртуального термінала ($serialMonitor), що дозволяє спостерігати за роботою системи в реальному часі без додаткового апаратного забезпечення. У реальних умовах експлуатації аналогічну діагностичну інформацію можна отримати, підключивши плату ESP32 до комп’ютера через USB-кабель: сигнали TX0/RX0 перетворюються вбудованою мікросхемою USB-UART в стандартний USB-інтерфейс, а самі повідомлення переглядаються за допомогою вбудованого в розширення PlatformIO для VS Code, або будь-якої іншої термінальної програми, налаштованої на швидкість обміну 115200 біт/с відповідно до значення, встановленого в прошивці (Serial.begin(115200)).

4 ПРОГРАМНА ЧАСТИНА СИСТЕМИ

4.1 Загальна архітектура програмного забезпечення

Програмне забезпечення мікроконтролера реалізовано на базі фреймворку Arduino для платформи ESP32 у середовищі PlatformIO. Прошивка побудована як єдиний модуль з чітким розподілом обов’язків між окремими функціями, що спрощує супровід коду в межах дипломного проєкту.

Структурно прошивку поділено на такі функціональні блоки: ініціалізація Wi-Fi-з’єднання (setupWiFi()), підтримка з’єднання з MQTT-брокером (reconnectMQTT()), збір даних та публікація телеметрії (collectAndPublishTelemetry()), а також стандартні для Arduino-фреймворку точки входу setup() та loop().

Ініціалізація системи. Перед входом у головний цикл функція setup() виконує послідовну підготовку всіх периферійних пристроїв та мережевих з’єднань (лістинг 4.1):

Лістинг 4.1 – Функція ініціалізації системи

```
void setup() {
  Serial.begin(115200);

  pinMode(PIN_RELAY_PUMP, OUTPUT);
  digitalWrite(PIN_RELAY_PUMP, LOW);

  dht.begin();

  setupWiFi();
  mqttClient.setServer(mqtt_server, mqtt_port);
}
```

Порядок дій: спочатку налаштовується вивід керування реле з явним встановленням початкового стану LOW (запобігає мимовільному вмиканню помпи одразу після подачі живлення, до отримання першого достовірного показника рівня води), потім ініціалізується бібліотека DHT22, і лише після цього виконується підключення до Wi-Fi та вказання адреси MQTT-брокера. Така послідовність гарантує, що на момент запуску головного циклу вся периферія та мережеві параметри вже перебувають у правильному стані.

Організація головного циклу. Головний цикл програми побудовано без використання функції затримки delay(), яка призупинила б виконання всієї програми на заданий час і унеможливила б своєчасну обробку MQTT-з’єднання. Замість цього застосовано порівняння поточного часу роботи мікроконтролера з часом останньої публікації (лістинг 4.2):

Лістинг 4.2 – Функція головного циклу програми

```
void loop() {
  if (!mqttClient.connected()) {
    reconnectMQTT();
  } else {
    mqttClient.loop();
  }

  unsigned long now = millis();
  if (now - lastPublishTime >= PUBLISH_INTERVAL_MS) {
    lastPublishTime = now;
    collectAndPublishTelemetry();
  }
}
```

Виклик mqttClient.loop() виконується на кожній ітерації циклу незалежно від того, настав час публікації чи ні. Саме це забезпечує безперервну обробку вхідних мережевих пакетів і підтримку з’єднання з брокером. Перевірка mqttClient.connected() на початку циклу з викликом reconnectMQTT() у разі

розриву з’єднання робить систему стійкою до тимчасових втрат зв'язку, характерних для роботи в марині чи на відкритій воді. При цьому сама функція reconnectMQTT() побудована за тим самим неблокуючим принципом, що й головний цикл: вона не зупиняє виконання loop() на час очікування з’єднання, тому опитування сенсорів та керування виконавчими пристроями продовжується з незмінним інтервалом незалежно від поточного стану MQTT-з’єднання.

4.2 Модуль збору даних з сенсорів та обробка вимірювань

Збір даних з усіх сенсорів виконується в межах функції collectAndPublishTelemetry(), що викликається з головного циклу програми з інтервалом 5 секунд.

Температура та вологість. Зчитування показників сенсора DHT22 виконується через виклики бібліотечних функцій readTemperature() та readHumidity() (лістинг 4.3):

Лістинг 4.3 – Функція зчитування показників сенсора DHT22

```
// Температура та вологість
float temperature = dht.readTemperature();
float humidity    = dht.readHumidity();

if (isnan(temperature) || isnan(humidity)) {
  Serial.println("Помилка зчитування DHT22, використано резервні значення.");
  temperature = 25.0;
  humidity    = 50.0;
}
```

Сенсори типу DHT22 можуть періодично повертають некоректний результат через особливості односигнального протоколу обміну даними, тому бібліотека сигналізує про це, повертаючи значення NaN (Not a Number).

Перевірка isnan() дозволяє виявити такий випадок і підставити безпечні резервні значення, запобігаючи публікації некоректних даних або аварійному завершенню роботи прошивки при подальших обчисленнях.

Рівень води в трюмі. Аналоговий сигнал з потенціометра, що імітує сенсор рівня води, зчитується функцією analogRead(), яка повертає 12-бітне значення в діапазоні від 0 до 4095 (лістинг 4.4):

Лістинг 4.4 – Функція зчитування та обробки показників сенсора рівня води

```
// Рівень води в трюмі
int waterRaw = analogRead(PIN_WATER_LEVEL);
bool waterAlert = waterRaw > WATER_LEVEL_THRESHOLD;
```

Порогове значення WATER_LEVEL_THRESHOLD (2500 з 4095) визначає межу, після перевищення якої рівень води вважається критичним. Результат порівняння одразу використовується логічним модулем прийняття рішень для керування помпою та подачі аварійного сповіщення.

Напруга бортового акумулятора. Оскільки сенсор напруги, обраний у підрозділі 3.1, апаратно має власний дільник напруги, який має номінальний коефіцієнт поділу 1:5, а вхід АЦП мікроконтролера розрахований на діапазон від 0 до 3,3 В, прошивка застосовує програмний коефіцієнт VOLTAGE_DIVIDER_RATIO для перетворення сирого значення АЦП у фізичну величину напруги бортової мережі (лістинг 4.5):

Лістинг 4.5 – Функція зчитування та обробки показників сенсора рівня води

```
// Напруга бортового акумулятора
int batteryRaw = analogRead(PIN_BATTERY_DIV);
float adcVoltage = (batteryRaw / 4095.0) * 3.3;
float batteryVoltage = adcVoltage * VOLTAGE_DIVIDER_RATIO;
bool batteryLow = batteryVoltage < BATTERY_LOW_THRESHOLD_V;
```

Спочатку необроблене 12-бітне значення нормалізується до напруги на вході АЦП (0–3,3 В), після чого множиться на коефіцієнт VOLTAGE_DIVIDER_RATIO. Значення цього коефіцієнта в прошивці підлягає калібруванню під конкретний екземпляр сенсора при реальному розгортанні системи та може відрізнятися від номінального заводського значення через неточні параметри компонентів і особливості конкретного вимірювального кола.

Рівень палива в баку. Аналогічно до сенсора рівня води, сигнал з потенціометра перетворюється у відсоткове значення заповнення бака (лістинг 4.6):

Лістинг 4.6 – Функція зчитування та обробки показників сенсора рівня палива

```
// Рівень палива в баку
int fuelRaw = analogRead(PIN_FUEL_LEVEL);
float fuelPercent = (fuelRaw / 4095.0) * 100.0;
bool fuelLow = fuelPercent < FUEL_LOW_THRESHOLD_PCT;
```

Такий підхід, який полягає в нормалізації сирого значення АЦП до фізично інтерпретованої величини (відсотка, напруги) безпосередньо перед прийняттям рішення та публікацією, послідовно застосовано для всіх трьох аналогових сенсорів системи, що забезпечує єдину логіку обробки незалежно від конкретного контрольованого параметра.

4.3 Логіка прийняття рішень та керування виконавчим пристроєм

Прийняття рішення про необхідність вмикання трюмної помпи реалізовано у вигляді простої порогової логіки, вбудованої прямо в функцію збору телеметрії. Такий підхід забезпечує автономну роботу системи навіть за умови тимчасової відсутності зв’язку з MQTT-брокером, оскільки керування виконавчим пристроєм не залежить від мережі – рішення приймається локально,

на стороні мікроконтролера, одразу після зчитування значення з сенсора (лістинг 4.7):

Лістинг 4.7 – Функція прийняття рішення про необхідність вмикання трюмної помпи

```
// Рівень води в трюмі
int waterRaw = analogRead(PIN_WATER_LEVEL);
bool waterAlert = waterRaw > WATER_LEVEL_THRESHOLD;
// Логіка керування трюмною помпою через реле
digitalWrite(PIN_RELAY_PUMP, waterAlert ? HIGH : LOW);
pumpActive = waterAlert;
```

Порівняння значення АЦП (0–4095) з порогом WATER_LEVEL_THRESHOLD (2500) відбувається до будь-якої публікації даних на брокер, а результат одразу подається на цифровий вивід керування реле. Це гарантує мінімальну затримку між моментом перевищення критичного рівня води та фактичним вмиканням помпи, інтервал реакції обмежується лише швидкістю виконання самого циклу loop(), а не циклом мережевого обміну.

Окремо в прошивці збережено булеву змінну стану pumpActive, яка дублює логічний стан реле. Це зроблено для того, щоб при публікації телеметрії не виконувати повторне зчитування даних виводу реле – стан помпи публікується як окреме значення на основі вже обчисленого результату порівняння, що спрощує код і виключає можливість розсинхронізації між фактичним станом реле та переданим значенням.

На відміну від рівня води, перетинання порогів BATTERY_LOW_THRESHOLD_V (11,5 В) та FUEL_LOW_THRESHOLD_PCT (15 %) не супроводжується керуванням жодним виконавчим пристроєм – це інформаційні пороги, призначені виключно для сповіщення власника чи екіпажу судна. У поточній реалізації прошивки логіка формування трьох булевих ознак (waterAlert, batteryLow, fuelLow) відокремлена від їх подальшого відображення: самі ознаки обчислюються в collectAndPublishTelemetry() одразу після

зчитування відповідного сенсора, а їх вивід у послідовний порт виконується окремою функцією printTelemetryReport(), що викликається наприкінці збору телеметрії (лістинг 4.8):

Лістинг 4.8 – Функція виводу сповіщень

```
if (batteryLow || fuelLow || waterAlert) {
  Serial.print(line);
  if (waterAlert)   Serial.println("  !! УВАГА: висока вода в трюмі, помпу увімкнено !!");
  if (batteryLow)   Serial.println("  !! УВАГА: критично низький заряд акумулятора !!");
  if (fuelLow)      Serial.println("  !! УВАГА: критично низький рівень палива !!");}
```

Умова if (batteryLow || fuelLow || waterAlert) друкує роздільник line лише за наявності хоча б одного критичного стану, а кожне з трьох попереджень виводиться незалежно одним рядком за наявності відповідної проблеми – це дозволяє одночасно відобразити кілька тривог у межах одного звіту без дублювання логіки прийняття рішень, яка вже виконана раніше при обчисленні самих булевих значень.

Таким чином, у розробленій системі реалізовано два принципово різні типи реакції на критичні показники: активне керування виконавчим пристроєм для параметра, що допускає автоматичне усунення небезпечної ситуації, та пасивне інформування для параметрів, усунення критичного стану яких потребує втручання людини і не може бути виконано автоматично засобами самої системи.

4.4 Реалізація MQTT-комунікації

Обмін даними між мікроконтролером та зовнішніми користувачами телеметрії реалізовано на базі протоколу MQTT з використанням бібліотеки

PubSubClient. Публічний тестовий брокер test.mosquitto.org обрано для цілей симуляції та відлагодження системи, оскільки він не потребує реєстрації облікового запису чи налаштування власної інфраструктури (лістинг 4.9):

Лістинг 4.9 – Налаштування параметрів MQTT-брокера

```
// Налаштування MQTT-брокера
const char* mqtt_server = "test.mosquitto.org";
const int   mqtt_port   = 1883;
const char* mqtt_client_id = "vessel-esp32-dyplom";
```

Підключення клієнта до брокера ініціалізується один раз під час запуску програми і виконання функції setup(), тоді як власне встановлення з’єднання та його підтримка винесені в окрему функцію reconnectMQTT(), що викликається з головного циклу за потреби (лістинг 4.10):

Лістинг 4.10 – Перепідключення до MQTT-брокера

```
// Перепідключення до MQTT-брокера
unsigned long lastReconnectAttempt = 0;
const unsigned long MQTT_RECONNECT_INTERVAL_MS = 2000;

void reconnectMQTT() {
  unsigned long now = millis();
  if (now - lastReconnectAttempt < MQTT_RECONNECT_INTERVAL_MS)
{
    return;
  }
  lastReconnectAttempt = now;

  Serial.print("Спроба підключення MQTT...");
  if (mqttClient.connect(mqtt_client_id)) {
    Serial.println("підключено до брокера MQTT!");
  } else {
    Serial.print("помилка підключення, rc=");
    Serial.print(mqttClient.state());
    Serial.println(" повторна спроба через 2с");
  }
}
```

Функція побудована без блокуючого очікування: замість циклу while із затримкою delay(2000), що призупинила б виконання всієї програми до моменту встановлення з'єднання, застосовано перевірку інтервалу через millis(), за тим самим принципом, що й у головному циклі. Якщо з моменту останньої спроби підключення минуло менше MQTT_RECONNECT_INTERVAL_MS (2 секунди), функція одразу повертає керування в loop(), не виконуючи повторної спроби. Це критично важливо для автономності системи: якщо reconnectMQTT() блокувала б виконання на час відсутності зв'язку, зчитування сенсорів і керування трюмною помпою також зупинялися б до відновлення з'єднання з брокером, що суперечило б вимозі локального, незалежного від мережі прийняття рішень. У поточній реалізації collectAndPublishTelemetry() викликається за розкладом millis() незалежно від стану MQTT-з'єднання. Виклики mqttClient.publish() під час розриву зв'язку просто не надсилають дані, не впливаючи на решту логіки функції.

Структура топіків. Кожен контрольований параметр публікується в окремий топік з ієрархічною назвою, що відображає його призначення та категорію (лістинг 4.11):

Лістинг 4.11 – Ініціалізація топіків телеметрії

```
// Топіки телеметрії
const char* TOPIC_TEMPERATURE =
"vessel/telemetry/temperature";
const char* TOPIC_HUMIDITY    = "vessel/telemetry/humidity";
const char* TOPIC_BATTERY     = "vessel/telemetry/battery";
const char* TOPIC_WATERLEVEL  = "vessel/alerts/water_level";
const char* TOPIC_FUEL        = "vessel/telemetry/fuel";
const char* TOPIC_PUMP_STATE  = "vessel/status/pump";
```

Назви топіків побудовано за єдиним принципом vessel/<категорія>/<параметр>, де категорія відображає характер даних: telemetry – для регулярних вимірюваних числових значень (температура, вологість, напруга акумулятора, рівень палива), alerts – для параметрів, що

безпосередньо сигналізують про потенційно небезпечну ситуацію (рівень води в трюмі), та status – для стану виконавчого пристрою (трюмна помпа). Такий поділ дозволяє клієнтам-споживачам підписуватися вибірково, наприклад, лише на топіки категорії alerts для реалізації окремого каналу сповіщень, не обробляючи при цьому весь потік телеметрії.

Формат повідомлень. Усі числові значення передаються у вигляді текстових рядків, а не бінарних даних, що спрощує читання повідомлень будь-яким MQTT-клієнтом без необхідності знати внутрішній формат кодування (лістинг 4.12):

Лістинг 4.12 – Публікація в MQTT

```
// Публікація в MQTT
char buf[16];

dtostrf(temperature, 4, 2, buf);
mqttClient.publish(TOPIC_TEMPERATURE, buf);

dtostrf(humidity, 4, 2, buf);
mqttClient.publish(TOPIC_HUMIDITY, buf);
```

Функція dtostrf() перетворює число з рухомою комою на рядок символів із фіксованою шириною поля (4 символи) та кількістю знаків після коми (2), що забезпечує однаковий, передбачуваний формат представлення для всіх вимірюваних величин. Для параметрів стану тривоги рівня води та стану помпи – замість числового представлення передається текстова мітка стану (лістинг 4.13):

Лістинг 4.13 – Публікація в MQTT (Передача текстової мітки стану)

```
const char* waterStatus = waterAlert ? "ALERT" : "NORMAL";
mqttClient.publish(TOPIC_WATERLEVEL, waterStatus);

mqttClient.publish(TOPIC_PUMP_STATE, pumpActive ? "ON" :
"OFF");
```

Публікація виконується з рівнем якості обслуговування за замовчуванням, що застосовується бібліотекою PubSubClient – QoS 0 («доставка щонайбільше один раз», без підтвердження отримання). Такий вибір є прийнятним для даної системи, оскільки кожне значення телеметрії публікується повторно з інтервалом 5 секунд: втрата одного повідомлення при нестабільному зв’язку не є критичною, адже актуальні дані буде надіслано вже під час наступного циклу публікації.

4.5 Вебдашборд для візуалізації телеметрії

Для наочного відображення телеметрії кінцевому користувачу, розроблено окремий вебдашборд – самостійну HTML-сторінку, що підключається до того самого MQTT-брокера, що й мікроконтролер, але через інший транспортний протокол.

Мікроконтролер обмінюється даними з брокером за протоколом MQTT поверх TCP-з’єднання, тоді як браузер із міркувань безпеки не може встановлювати низькорівневі TCP-з’єднання напряму. Тому дашборд використовує клієнтську бібліотеку MQTT.js та підключається до того самого брокера через MQTT через WebSocket (лістинг 4.14):

Лістинг 4.14 – Підключення до MQTT-брокера через WebSocket

```
    const client =
mqtt.connect('wss://test.mosquitto.org:8081/mqtt', {
      clientId: 'vessel-dashboard-' +
Math.random().toString(16).slice(2, 8),
      clean: true,
      reconnectPeriod: 3000
    });

    client.on('connect', () => {
      setConnectionState('waiting');
      TOPICS.forEach(t => client.subscribe(t));
    });
```

Ідентифікатор клієнта формується з випадкового суфікса, що дозволяє відкривати кілька екземплярів дашборду одночасно без конфлікту clientId на брокері. Після встановлення з'єднання дашборд підписується одразу на всі шість топіків, визначених у масиві TOPICS, що структурно повторює перелік топіків з підрозділу 4.4 (з прошивки мікроконтролера).

Оскільки дашборд і мікроконтролер – це два незалежні MQTT-клієнти, підключені до спільного брокера, працюючий стан «з'єднання встановлено» не гарантує, що дані від пристрою реально надходять. Для розрізнення цих ситуацій реалізовано індикатор із п'ятьма станами (лістинг 4.15):

Лістинг 4.15 – П'ятирівневий індикатор стану з'єднання

```
const labels = {
  connecting: "підключення до брокера...",
  waiting: "брокер онлайн — чекаємо дані з пристрою",
  flowing: "онлайн — дані надходять",
  stale: "брокер онлайн, але пристрій мовчить",
  error: "помилка з'єднання з брокером"
};
```

Перехід між станами waiting і flowing відбувається одразу при отриманні першого повідомлення від брокера незалежно від топіка (лістинг 4.16):

Лістинг 4.16 – Перехід між станами waiting і flowing

```
client.on('message', (topic, payloadBuf) => {
  lastMessageAt = Date.now();
  setConnectionState('flowing');
```

Стан stale (пристрій «мовчить») є окремим таймером, що періодично перевіряє час, який минув з моменту останнього отриманого повідомлення (лістинг 4.17):

Лістинг 4.17 – Таймер з моменту останнього отриманого повідомлення

```
let lastMessageAt = 0;
const STALE_TIMEOUT_MS = 12000; // трохи більше 2х циклів публікації ESP32 (5с кожен)

setInterval(() => {
  if (!client.connected) return;
  if (lastMessageAt && Date.now() - lastMessageAt > STALE_TIMEOUT_MS) {
    setConnectionState('stale');
  }
}, 2000);
```

Значення STALE_TIMEOUT_MS (12 секунд) свідомо обрано дещо більшим за подвоєний інтервал публікації мікроконтролера, який дорівнює 5 секундам, щоб короткочасна затримка одного циклу публікації через нестабільність мережі не сприймалася дашбордом як втрата зв’язку з пристроєм. Стан error встановлюється при розриві WebSocket-з’єднання з самим брокером, що дозволяє чітко розрізняти на інтерфейсі дві принципово різні несправності: недоступність брокера та мовчання пристрою при справному з’єднанні з брокером.

Обробка та відображення вхідних повідомлень. Кожне отримане повідомлення розбирається залежно від топіка, з якого воно надійшло, після чого оновлюється відповідний елемент інтерфейсу (лістинг 4.18):

Лістинг 18 – Обробка та відображення вхідних повідомлень

```
if (topic === TOPICS[2]) {
  document.getElementById('val-batt').innerHTML = lue.toFixed(2) + ' <span class="unit">V</span>';
  drawDial('dial-batt', value, 0, 15, value < 11.5);
}
if (topic === TOPICS[3]) {
  document.getElementById('val-fuel').innerHTML = lue.toFixed(1) + ' <span class="unit">%</span>';
  drawDial('dial-fuel', value, 0, 100, value < 15);
}
```

Числові параметри (температура, вологість, напруга акумулятора, рівень палива) відображаються одночасно у вигляді тексту та стрілочного індикатора, причому колір шкали змінюється на червоний, якщо значення виходить за межі порогів, ідентичних до порогів у прошивці мікроконтролера. Дискретні параметри (тривога рівня води, стан помпи) відображаються індикаторними лампами, що змінюють колір залежно від отриманого текстового значення.

Отримані повідомлення додатково дублюються в текстовий журнал на сторінці (функція logMessage()) із фіксацією часу отримання, назви топіка та значення – це дає можливість спостерігати за необробленим потоком даних безпосередньо в браузері, без потреби у сторонньому MQTT-клієнті чи послідовному моніторі, який при справжній реалізації доступний лише при фізичному підключенні до мікроконтролера.

4.6 Результати тестування системи

Працездатність розробленої системи перевірено в середовищі симуляції Wokwi шляхом запуску прошивки на віртуальній платі ESP32 DevKit V1 зі схемою підключення, описаною в підрозділі 3.3, та спостереження за результатами роботи як на стороні мікроконтролера (через послідовний порт), так і на стороні клієнта телеметрії (вебдашборд).

Діагностичний вивід мікроконтролера. У процесі роботи прошивка кожні 5 секунд формує в Serial Monitor форматований звіт телеметрії засобами функції printTelemetryReport(), що містить поточні показники всіх сенсорів, стан трюмної помпи та, за наявності, попередження про критичні значення. Приклад такого звіту наведено на рисунку 4.19.

```
----------------------------------------
  ЗВІТ ТЕЛЕМЕТРІЇ   [час з запуску:     5024 ms]
----------------------------------------
  Температура:   31.60 C
  Вологість:       55.00 %
  Напруга АКБ:    12.41 V
  Рівень палива:  11.14 %
  Рівень води (ADC):   2974
----------------------------------------
  Трюмна помпа ..... УВІМК (ON)
  Вода в трюмі ..... ТРИВОГА
  Заряд АКБ ..... норма
  Паливо ..... КРИТИЧНО НИЗЬКЕ
----------------------------------------
  !! УВАГА: висока вода в трюмі, помпу увімкнено !!
  !! УВАГА: критично низький рівень палива !!
----------------------------------------
```

Рисунок 4.19 – Приклад звіту телеметрії в Serial Monitor середовища Wokwi

Вебдашборд. Паралельно із роботою мікроконтролера в браузері запущено дашборд, підключений до того самого MQTT-брокера. Загальний вигляд інтерфейсу дашборда в робочому режимі наведено на рисунку 4.20.

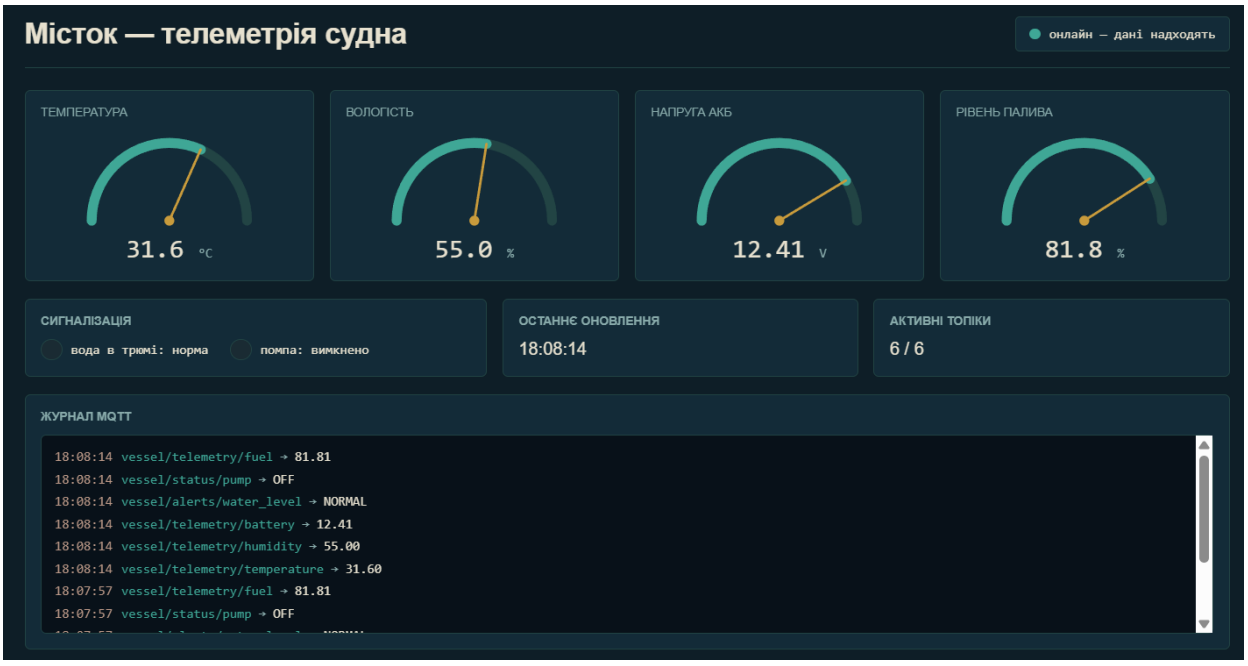


Рисунок 4.20 – Вебдашборд у стані отримання телеметрії від пристрою

Отримання першого повідомлення від пристрою переводить індикатор з’єднання в стан «flowing», після чого стрілочні індикатори, лампи сигналізації та журнал повідомлень оновлюються синхронно з циклом публікації телеметрії.

ВИСНОВОК

У межах кваліфікаційної роботи розроблено IoT-систему моніторингу експлуатаційних показників об’єкта водного транспорту на базі мікроконтролера ESP32, здатну в автономному режимі контролювати температуру та вологість машинного відділення, рівень води у трюмі, напругу бортового акумулятора та рівень палива, а також передавати зібрані дані власнику судна незалежно від його фізичної присутності на борту.

У першому розділі проаналізовано предметну область та виявлено характерний недолік наявних комерційних рішень – високу вартість, закритість екосистеми та обов’язкову платну підписку на хмарні сервіси, що робить їх економічно невигідними для власників невеликих суден. Огляд науково-технічних публікацій підтвердив, що архітектурні рішення IoT-моніторингу з інших галузей можуть бути успішно адаптовані до задач водного транспорту, а на основі проведеного аналізу сформовано перелік функціональних та нефункціональних вимог до системи, ключовою серед яких визначено вимогу автономності – здатність приймати критичні для безпеки рішення без залежності від мережевого з’єднання.

У другому розділі обґрунтовано вибір технологій реалізації: серед розглянутих мікроконтролерних платформ (Arduino, STM32, ESP32) перевагу віддано ESP32 DevKit V1 завдяки вбудованому Wi-Fi-модулю, двоядерній архітектурі та економічній доступності; серед протоколів передавання даних (HTTP, CoAP, MQTT) обрано MQTT як оптимальний для умов обмеженого й нестабільного мережевого покриття, характерного для експлуатації водного транспорту; для розробки та симуляції визначено середовище VS Code з розширеннями PlatformIO та Wokwi для VS Code, що дозволило проводити тестування без фізичного прототипу.

У третьому розділі обґрунтовано вибір конкретних сенсорів для кожного контрольованого параметра, розроблено розподіл виводів мікроконтролера з

урахуванням апаратних обмежень блоків АЦП ESP32 (конфлікт ADC2 з Wi-Fi-модулем) та розроблено й перевірено в середовищі Wokwi принципову схему підключення периферії.

У четвертому розділі реалізовано програмну частину системи: неблокуючу архітектуру головного циклу на основі millis(), модуль збору даних з обробкою даних отриманих в результаті вимірювань, порогову логіку прийняття рішень з автоматичним керуванням трюмною помпою через реле, а також публікацію телеметрії за протоколом MQTT у структуровані топіки. Окремо розроблено вебдашборд для візуалізації телеметрії в реальному часі через MQTT over WebSocket, з індикацією стану з'єднання, що розрізняє відсутність зв'язку з брокером та відсутність даних від самого пристрою. Проведене тестування в середовищі симуляції підтвердило коректність роботи всіх реалізованих функціональних блоків.

Ключовою інженерною особливістю розробленої системи є розмежування критичної та допоміжної функціональності: керування трюмною помпою виконується мікроконтролером локально й миттєво, одразу після зчитування показника з сенсора, без залежності від наявності мережевого з'єднання, тоді як передавання телеметрії та відновлення зв'язку з MQTT-брокером реалізовано неблокуючим чином і не впливає на швидкість реакції системи на позаштатну ситуацію. Такий підхід забезпечує відмовостійкість системи навіть за умов тимчасової втрати зв'язку, характерної для морського середовища.

Отримані результати підтверджують, що розроблена система відповідає сформульованим у першому розділі вимогам і може розглядатися як економічно доступна альтернатива комерційним рішенням моніторингу об'єктів водного транспорту з відкритою, легко масштабованою архітектурою. Подальший розвиток системи може передбачати додавання нових контрольованих параметрів, зокрема сенсорів положення чи навігаційних даних, реалізацію збереження історії телеметрії на стороні сервера для статистичного аналізу, а також перехід від симуляційного до фізичного прототипу з подальшим тестуванням в умовах реальної експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Maritime 4.0 – opportunities in digitalization and advanced manufacturing for vessel development / B. P. Sullivan et al. Procedia manufacturing. 2020. Vol. 42. P. 246–253. URL: https://doi.org/10.1016/j.promfg.2020.02.078.

2. Optimization of the operation of marine diesel engines using wireless monitoring systems / V. V. Nikolskyi et al. Scientific bulletin kherson state maritime academy. 2025. Vol. 1, no. 30. P. 80–91. URL: https://doi.org/10.33815/2313-4763.2025.1.30.080-091.

3. Zaborskyi L., Shaposhnikov D., Belakovskiy L. Specifics of forming the digital potential of maritime transport enterprises in the context of sustainable development. Development of management and entrepreneurship methods on transport (ONMU). 2024. Vol. 89, no. 4. P. 81–96. URL: https://doi.org/10.31375/2226-1915-2024-4-81-96.

4. Pravalika V., Rajendra Prasad C. Internet of things based home monitoring and device control using ESP32. International journal of recent technology and engineering. 2019. Vol. 8, no. 1S4. P. 58–62.

5. Kodali R. K., Soratkal S. R. MQTT based home automation system using ESP8266. *2016 IEEE region 10 humanitarian technology conference (R10-HTC)*. 2016. P. 1–5.

6. Design and implementation of esp32-based iot devices / D. Hercog et al. *Sensors*. 2023. Vol. 23, no. 15. P. 6739. URL: https://doi.org/10.3390/s23156739.

7. Abu Sneineh A., Shabaneh A. A. A. Design of a smart hydroponics monitoring system using an ESP32 microcontroller and the Internet of Things. *MethodsX*. 2023. Vol. 11. P. 102401. URL: https://doi.org/10.1016/j.mex.2023.102401.

8. Asparuhova K., Others. Using WOKWI simulator to support engineering student learning in microcontrollers and sensors. *2024 XXXIII international scientific conference electronics (ET)*. 2024. P. 1–4.

9. Kareem H., Dunaev D. The working principles of ESP32 and analytical comparison of using low-cost microcontroller modules in embedded systems design. *2021 4th international conference on circuits, systems and simulation (ICCSS)*, Kuala Lumpur, Malaysia, 26–28 May 2021. 2021. URL: https://doi.org/10.1109/iccss51193.2021.9464217.

10. Mathurkar A. Impact of microcontroller architecture on oscilloscope performance: A study of ESP32 vs. STM32. *International journal for research in applied science and engineering technology*. 2025. Vol. 13, no. 3. P. 2669–2674. URL: https://doi.org/10.22214/ijraset.2025.67896.

11. Ionescu V. M., Enescu F. M. Investigating the performance of MicroPython and C on ESP32 and STM32 microcontrollers. *2020 IEEE 26th international symposium for design and technology in electronic packaging (SIITME)*, Pitesti, Romania, 21–24 October 2020. 2020. URL: https://doi.org/10.1109/siitme50350.2020.9292199.

12. S. Sotnik, “Performance and simulation-based comparison of IoT access control systems using ESP32, Arduino Uno, and Raspberry Pi Pico W,” ACIG, vol. 5, no. 1, 2026, doi: 10.60097/ACIG/221089

13. Naik N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE international systems engineering symposium (ISSE)*, Vienna, Austria, 11–13 October 2017. 2017. URL: https://doi.org/10.1109/syseng.2017.8088251.

14. Ouakasse F., Rakrak S. Comparative study of MQTT and CoAP application layer protocols via performances evaluation. *Journal of Engineering and Applied Sciences*, vol. 13, no. 15, 2018, pp. 6053–6061, ISSN: 1816-949X. URL: https://makhillpublications.co/files/published-files/mak-jeas/2018/15-6053-6061.pdf.

15. Performance evaluation of MQTT and CoAP via a common middleware / D. Thangavel et al. *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*, Singapore, 21–24 April 2014. 2014. URL: https://doi.org/10.1109/issnip.2014.6827678.

16. Comparative analysis of power consumption between MQTT and HTTP protocols in an iot platform designed and implemented for remote real-time monitoring of long-term cold chain transport operations / H. J. Jara Ochoa et al. *Sensors*. 2023. Vol. 23, no. 10. P. 4896. URL: https://doi.org/10.3390/s23104896.

17. What is PlatformIO? – PlatformIO latest documentation. *PlatformIO*. URL: https://docs.platformio.org/en/latest/what-is-platformio.html.

18. Getting started with wokwi for VS code | wokwi docs. *Welcome to Wokwi! | Wokwi Docs*. URL: https://docs.wokwi.com/vscode/getting-started.

19. lady ada. DHT11, DHT22 and AM2302 Sensors. *Adafruit Learning System*. URL: https://learn.adafruit.com/dht/overview.

20. ESP32 Technical Reference Manual, Espressif Systems. URL: https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf .