МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Навчально-науковий інститут інженерії та інформаційних технологій
(повне найменування інституту, назва факультету)

Кафедра комп'ютерної інженерії та електромеханіки
(повна назва кафедри)

Дипломна магістерська робота

на тему **Система збору та аналізу даних для IoT платформ**

Виконав: студент групи МгЗКІ-21

спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

Стаценко В. В.
(прізвище та ініціали)

Керівник Осипенко В. В.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Київ 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Навчально-науковий інститут інженерії та інформаційних технологій
Кафедра комп'ютерної інженерії та електромеханіки
Спеціальність 123 «Комп'ютерна інженерія»
(шифр і назва)
Освітня програма Комп'ютерна інженерія
(назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Б.М. Злотенко
«___» _____2022 р.

З А В Д А Н Н Я

НА ДИПЛОМНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Стаценку Володимиру Володимировичу

1. Тема роботи **Система збору та аналізу даних для IoT платформ**

Науковий керівник роботи Осипенко Володимир Васильович, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "28" вересня 2022 року № 180-уч
2. Строк подання студентом роботи 14 листопада 2022 року
3. Вихідні дані до роботи: IoT платформи, бази даних, web сервіси, мікроконтролери, мережеве обладнання, датчики, наукові джерела, нормативна та довідкова література за темою магістерської роботи

4.Зміст дипломної роботи (перелік питань, які потрібно розробити)

1. Платформи і засоби для збору та аналізу даних.
2. Архітектура систем збору та аналізу даних.
3. Система зчитування та передачі сигналів від датчиків.
4. Централізована система збору та аналізу даних.

5. Консультанти розділів дипломної магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	Володимир ОСИПЕНКО, д.т.н., проф.		
Розділ 2	Володимир ОСИПЕНКО, д.т.н., проф.		
Розділ 3	Володимир ОСИПЕНКО, д.т.н., проф.		
Розділ 4	Володимир ОСИПЕНКО, д.т.н., проф.		

6. Дата видачі завдання 12.09.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	23.09.2022	
2	Розділ 1. Платформи і засоби для збору та аналізу даних	07.10.2022	
3	Розділ 2. Архітектура систем збору та аналізу даних	15.09.2022	
4	Розділ 3. Система зчитування та передачі сигналів від датчиків	19.10.2022	
5	Розділ 4. Централізована система збору та аналізу даних	01.11.2022	
6	Висновки	04.11.2022	
7	Оформлення дипломної магістерської роботи (чистовий варіант)	07.11.2022	
8	Здача дипломної магістерської роботи на кафедру для рецензування (за 14 днів до захисту)	07.11.2022	
9	Перевірка дипломної магістерської роботи на наявність ознак плагіату (за 10 днів до захисту)	11.11.2022	
10	Подання дипломної магістерської роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	14.11.2022	

Студент _____ Володимир СТАЦЕНКО
(підпис) (прізвище та ініціали)

Науковий керівник роботи _____ Володимир ОСИПЕНКО
(підпис) (прізвище та ініціали)

Директор НМЦУПФ _____ Олена ГРИГОРЕВСЬКА
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Стаценко В. В. Система збору та аналізу даних для IoT платформ. – Рукопис.

Дипломна магістерська робота за спеціальністю 123 «Комп'ютерна інженерія», освітньою програмою «Комп'ютерна інженерія». – Київський національний університет технологій та дизайну, Київ, 2022 рік.

Дипломну магістерську роботу присвячено розробці та дослідженню системи збору та аналізу даних для IoT платформ. Проведено аналіз процесу передачі даних в IoT системі, що побудована за трирівневою архітектурою. Досліджено процеси зчитування сигналів аналогових датчиків, перетворення їх в цифрову форму та визначено максимальну теоретично можливу швидкість передачі цих даних. Розроблено програмне забезпечення для системи збору даних від датчиків та передачі їх через Wi-Fi мережу. Розроблено схему підключення датчиків до аналого-цифрових перетворювачів та мікроконтролерів, що дозволяє передавати дані через Wi-Fi мережу. Розроблено серверне програмне забезпечення для підсистеми збору та аналізу даних.

Ключові слова: Інтернет речей, IoT, датчик, мікроконтролер, база даних, трирівнева архітектура, Wi-Fi мережа.

ABSTRACT

Statsenko V. V. Data collection and analysis system for IoT platforms. – Manuscript.

Master's degree work in specialty 123 «Computer Engineering», educational program «Computer Engineering». – Kyiv national university of technology and design, Kyiv, 2022.

The master's thesis is devoted to the development and research of a data collection and analysis system for IoT platforms. The data transfer process analysis in the IoT system, which is built according to three-level architecture, is carried out. The processes of analog sensors reading signals, converting them into digital form were studied, and the maximum theoretically possible speed of these data transmission was determined. Software has been developed for the data collecting system from sensors and transmitting them via a Wi-Fi network. A scheme for connecting sensors to analog-to-digital converters and microcontrollers has been developed, which allows data to be transmitted via a Wi-Fi network. Server software for the data collection and analysis subsystem has been developed.

Keywords: Internet of Things, IoT, sensor, microcontroller, database, three-level architecture, Wi-Fi network.

ЗМІСТ

ВСТУП

Сьогодні технології Інтернету речей (IoT) набули широкого поширення як у промисловому, так і у побутовому обладнанні. Їх основною перевагою є можливість централізованого збирання інформації про поточний стан підключеного обладнання, що дозволяє оптимізувати його використання та зробити використання цих пристроїв зручнішим для користувача. Створення сучасних IoT систем передбачає використання двох основних складових, що об’єднані через мережу Інтернет. Це фізичні пристрої з датчиками та мережевими адаптерами та централізовані або децентралізовані платформа, які здійснюють обробку даних та керування всім підключеним обладнанням.

Створенням пристроїв та програмного забезпечення для IoT сьогодні займаються сотні компаній. Випускається обладнання промислового рівня, що може працювати в складних зовнішніх умовах, та пристрої для проведення експериментів і швидкого прототипування, що дозволяють здійснювати підключення без використання паяних з’єднань. Також розроблено різноманітне програмне забезпечення як з комерційними, так і з відкритими ліцензіями, на базі якого будують хмарні платформи IoT.

Актуальність роботи. Не зважаючи на велику кількість готових розробок, при використанні універсальних рішень в конкретних умовах виникає ряд обмежень. Передусім проблема стосується хмарних IoT платформ. При їх використанні доступ до даних мають співробітники компаній, які є власниками платформ, що може бути неприйнятним з точки зору безпеки. Передача інформації здійснюється через Інтернет, що також знижує рівень захищеності порівняно з роботою всередині локальної мережі. Універсальні формати даних часто містять додаткові поля, які збільшують загальний обсяг даних, що негативно впливає на швидкість роботи системи. Також у більшості IoT платформ встановлені ліміти безкоштовного використання, що обмежують кількість даних, підключених пристроїв, можливості аналізу даних та інші параметри.

Розробка спеціалізованих систем збору та аналізу даних для IoT платформ дозволяє обійти зазначені вище обмеження та є актуальним завданням.

Метою роботи є створення системи збору та аналізу даних для IoT платформ, що надаватиме можливість збирати інформацію з датчиків, здійснювати її обробку та забезпечуватиме можливість роботи як з середини локальної мережі, так і через Інтернет.

Для досягнення поставленої мети у роботі було вирішено такі **задачі**:

- проведено огляд найбільш розповсюджених платформ Інтернету речей, проаналізовано їх основні можливості;
- запропоновано архітектуру системи збору та аналізу даних;
- розроблено схему підключення датчиків до АЦП та мікроконтролерів;
- створено програму зчитування сигналів від датчиків;
- розроблено алгоритм передачі даних через Wi-Fi мережу та створено його програмну реалізацію для модуля ESP8266 NodeMCU;
- створено підсистему зберігання даних на базі системи керування базами даних MySQL;
- розроблена підсистема обробки даних.

Об’єкт дослідження – процеси передачі та обробки даних в системах Інтернету речей.

Предмет дослідження – система збору та аналізу даних для IoT платформ.

Методи досліджень. Теоретичні дослідження базуються на основних положеннях теорії інформації, алгоритмів, обробки сигналів, принципів роботи комп’ютерних мереж та систем керування базами даних. Дослідження та розробка прикладного програмного забезпечення проводились з використанням інтегрованих середовищ розробки та інструментів аналізу роботи програм з відкритим програмним кодом.

Наукова новизна.

Проведено аналіз процесу передачі даних в IoT системі, що побудована за трирівневою архітектурою. Досліджено процеси зчитування сигналів аналогових датчиків, перетворення їх в цифрову форму та визначено максимальну теоретично можливу швидкість передачі цих даних.

Практичне значення отриманих результатів.

1. Розроблено програмне забезпечення для системи збору даних від датчиків та передачі їх через Wi-Fi мережу.

2. Розроблено схему підключення датчиків до аналого-цифрових перетворювачів та мікроконтролерів, що дозволяє передавати дані через Wi-Fi мережу.

3. Розроблено серверне програмне забезпечення для підсистеми збору та аналізу даних.

Структура та обсяг роботи. Магістерська робота складається зі вступу, 4 розділів, висновків, списку використаних джерел та додатків. Основний текст роботи викладений на 65 сторінках, містить 17 рисунків, 1 таблицю, список джерел з 25 найменувань. Загальний обсяг роботи, враховуючи 2 додатки, складає 79 аркушів.

РОЗДІЛ 1. ПЛАТФОРМИ І ЗАСОБИ ДЛЯ ЗБОРУ ТА АНАЛІЗУ ДАНИХ

1.1. Технологія Internet of Things (IoT)

Інтернет речей (Internet of Things, IoT) – це сукупність взаємозв'язаних фізичних пристроїв із вбудованими датчиками, виконавчими пристроями та програмним забезпеченням, що утворюють або входять до складу мережі [1, 2, 3]. Така система дозволяє здійснювати передачу даних між комп'ютерним обладнанням та фізичними об'єктами в автоматизованому режимі з використанням стандартних протоколів передачі даних через Інтернет.

В основу IoT покладено можливість підключення різноманітних речей або пристроїв, які використовують люди. IoT забезпечує зв'язок, передачу даних та керування цими пристроями. Для вирішення цих завдань, всі підключені об'єкти мають бути в обов'язковому порядку обладнані датчиками та пристроями передачі даних через Інтернет (WiFi, Ethernet адаптерами), що дозволяє забезпечити контроль їх роботи. Наявність керованих виконавчих пристроїв є необов'язковою, але суттєво розширює можливості системи, оскільки дозволяє змінювати режими роботи обладнання без участі людини.

Таким чином, до складу системи IoT входять два основні компоненти:

1) Фізичні об'єкти з датчиками та пристроями передачі даних через мережу Інтернет.

2) Централізовані або децентралізовані платформи, що забезпечують вирішення задач зберігання, обробки даних та керування підключеними пристроями.

Створення першого компонента передбачає встановлення датчиків та мережевих адаптеров на пристрої. Основними обмеженнями тут є можливість фізичної установки необхідного обладнання, наявність живлення з потрібними характеристиками для адаптерів, відстань до обладнання, необхідність його

переміщення в процесі роботи та ін. Сьогодні промисловість постійно збільшує номенклатуру виробів з вбудованими датчиками та мережевими адаптерами, що значно спрощує їх використання у IoT.

Другий компонент передбачає використання серверного обладнання, баз даних та програмного забезпечення, що забезпечує можливість передачі даних за протоколами HTTP, HTTPS, MQTT та іншими. Перелік функціональних можливостей, технічні характеристики обладнання та склад програмного забезпечення визначаються вимогами до конкретної платформи. Сьогодні ряд компаній пропонує хмарні IoT платформи, зокрема:

1) Arduino IoT Cloud [4] – дозволяє створювати проекти IoT із web інтерфейсом і комплексним рішенням для написання коду, налаштування, завантаження та візуалізації. До основних можливостей платформи відносяться:

– Моніторинг даних – можливість контролювати значення датчиків через інформаційну панель.

– Синхронізація змінних – дозволяє копіювати змінні між пристроями.

– Планувальник – забезпечує можливість виконання завдань у певний проміжок часу.

– Завантаження по повітрю (OTA) – оновлення програмного коду на пристрої без підключення до комп’ютера.

– Webhooks – інтеграція зі сторонніми сервісами.

– Керування правами доступу користувачів.

2) Google Cloud IoT [5] – це рішення IoT, створене на базі хмарної платформи Google. Вона включає повний набір інструментів для підключення, обробки, зберігання та аналізу даних. Рішення забезпечує інтегрований стек технологій Google Cloud для виконання обчислень. Google Cloud IoT є досить складним рішенням, що може бути проблемою для швидкого прототипування. Платформа орієнтована на використання у задачах прогнозного обслуговування, відстеження активів у реальному часі, логістики та управління ланцюгом поставок, створення розумних будинків та міст.

3) AWS IoT [6] – платформа від Amazon розрахована на підключення та керування мільярдами пристроїв. Дозволяє збирати, зберігати й аналізувати дані IoT для промислових, споживчих та комерційних задач. Платформа забезпечує можливості масштабування, шифрування, керування доступом та інтеграцію з іншими хмарними сервісами Amazon.

4) PTC ThingWorx [7] – платформа орієнтована на масштабне впровадження промислового Інтернету речей, включає деякі спеціально створені рішення, які доповнюють основні можливості Інтернету речей, наприклад платформи доповненої реальності та керування життєвим циклом продукту. Платформа часто використовується для вирішення наступних задач:

– Розповсюдження товару як послуги.

– Дистанційного моніторингу стану промислових об’єктів.

– Створення цифрових робочих інструкцій.

– Контроль параметрів у реальному часі.

5) Kaa IoT Platform [8] – орієнтована на зручність використання та швидкість впровадження IoT. Забезпечує повний набір функцій Інтернету речей, у тому числі вбудовану аналітику Інтернету речей, і не потребує інтеграції додаткових модулів або сервісів. Kaa добре підходить для широкого спектру випадків використання IoT. Платформа приділяє особливу увагу простоті самообслуговування та привабливості серед невеликих стартапів, а також великих компаній, які ще не готові до масштабного впровадження Інтернету речей. Kaa використовується у автомобільному транспорті, охороні здоров’я, промисловому IoT, логістиці, розумній роздрібній торгівлі.

6) Microsoft Azure IoT [9] – платформа від компанії Microsoft, яка є складовою частиною їх хмарного сервісу Azure. Позиціонується як рішення, що містить всі пристрої, інструменти, аналіз даних, а також засоби безпеки, необхідні для досягнення цілей впровадження IoT. Azure IoT складається з кількох компонентів і потребує певного досвіду для налаштувати його відповідно до потреб вашої компанії. Платформа використовується у

автомобільній, енергетичній галузях, охороні здоров'я, роздрібній торгівлі, транспорті та логістиці.

Практично всі зазначені платформи пропонують можливість безкоштовного використання з обмеженнями на кількість пристроїв та об’єм даних, що можуть бути збережені. Збільшення цих параметрів потребує переходу на платну версію платформи. Також в ряді випадків обробка даних у сторонніх компаніях може бути обмежена з точки зору безпеки.

1.2. Датчики фізичних величин

Датчики є невід’ємною складовою пристроїв, що входять до складу Інтернету речей. Оскільки більшість фізичних характеристик об’єктів (температура, світловий потік, тиск, електрична напруга, струм, тощо), які потрібно визначати, є за своєю природою аналоговими, для їх вимірювання використовуються аналогові датчики [10, 11].

Аналогові датчики відносяться до первинних перетворювачів, вихідний сигнал яких є пропорційним фізичній величині, що вимірюється. У системах IoT застосовуються датчики, що перетворюють фізичну величину у електричний сигнал (напругу), який зручно передавати на великі відстані та обробляти за допомогою обчислювальної техніки.

Сьогодні промисловістю випускається велика кількість датчиків, що можуть бути класифіковані за різними ознаками.

За видом вимірюваної величини: датчики механічних переміщень, пневматичні, швидкості, прискорення, електричні, витратоміри, зусилля, тиску, температури, тощо.

За видом вихідної величини розрізняють неелектричні та електричні датчики. До переваг електричних датчиків відносяться:

– простота передачі на великі відстані з високою швидкістю;

– можливість перетворення у цифровий код та забезпечення захисту від зовнішніх впливів.

За формою вихідної величини розрізняють наступні класи датчиків:

– аналогові датчики, тобто датчики, що формують аналоговий сигнал, пропорційно до зміни вхідної величини;

– цифрові датчики, що генерують послідовність імпульсів;

– двійкові датчики, які формують сигнал лише двох рівнів: 0 або 1.

Сьогодні широкого розповсюдження набули датчики, що сумісні з платформою Arduino. Вони не потребують використання додаткових перетворювачів для підключення до мікроконтролерів, що робить їх зручними для використаннях у задачах, що передбачають швидке створення прототипів. Існують десятки різновидів таких датчиків для визначення різних фізичних величин.

Зокрема до таких датчиків відносяться модулі визначення температури та вологості – DHT11 (рис. 1.1).

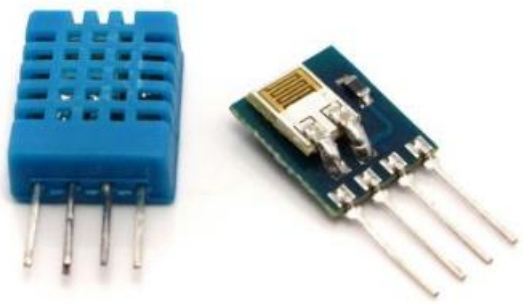


Рис. 1.1. Модуль датчика температури та вологості DHT11.

Датчик забезпечує:

– повний діапазон температурної компенсації;

– вимірювання відносної вологості та температури;

– відкалібрований цифровий сигнал;

– велику відстань передачі;

– низьке енергоспоживання.

Чутливий елемент DHT11 з'єднаний з 8-розрядним однокристальним комп'ютером. Кожен датчик цієї моделі має температурну компенсацію та відкалібрований у калібрувальній камері, а калібрувальний коефіцієнт

зберігається у вбудованій пам’яті. Невеликий розмір, низьке споживання та велика відстань передачі (20 м) дозволяють використовувати DHT11 в складних умовах. Технічні характеристики:

Напруга живлення – 3-5.5V DC.

Чутливий елемент – полімерний резистор.

Діапазон вимірювання: вологість – 20-90%RH; температури – 0-50 °C.

Похибка: вологості – ±4%RH (макс – ±5%RH); температури – ±2.0 °C.

Роздільна здатність або чутливість: вологість – 1%RH; температура – 0.1 °C.

Гістерезис вологості – ±1%RH.

Довгострокова стабільність: вологість – 0.5%RH/рік.

Середній період вимірювання – 2с.

Розміри – 12x15.5x5.5 мм

Також широко використовуються тензометричні датчики (рис. 1.2).

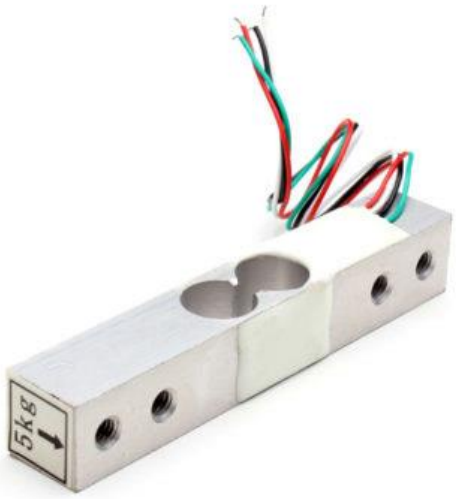


Рис. 1.2. Тензометричний датчик.

Тензометричний датчик виконаний з алюмінієвого сплаву у формі прямокутного бруску в центрі якого створено отвір. Тонкоплівкові резистори, що з'єднані за мостовою схемою, розташовані на його бічні поверхні. До

кожного з них під’єднані по 2 вихідні контакти. Алюмінієвий брусок має різьбові отвори для кріплення його до корпусу приладу. Пластина розташовується таким чином, щоб вантаж тиснув безпосередньо на неї. Кожен з датчиків такого типу має максимальне припустиме навантаження, що позначається на торцеву сторону датчика. Вихідний сигнал датчику є аналоговою величиною (напругою), яка є пропорційною зміні опору тонко плівкових резисторів. Комплексна похибка (повторюваність) датчиків такого типу знаходиться у межах від 0,03% до 0,05% вимірюваної величини.

Для подальшої обробки сигналу рекомендовано перетворити його в цифрову форму за допомогою аналого-цифрового перетворювача, наприклад, HX711.

1.3. Мікроконтролери та аналого-цифрові перетворювачі

При передачі аналогових сигналів виникають проблеми, пов’язані зі зниженням точності через вплив зовнішніх завад та втрати у лініях зв’язку. Визначення та компенсація цих аналогових завад є складним завданням, оскільки вони часто носять випадковий характер та залежать від зовнішніх умов. Поширеним рішення цієї проблеми є перетворення сигналу у цифрову форму та його подальша обробка та передача за допомогою комп’ютерного обладнання.

Перетворення аналогового сигналу у цифрову форму здійснюється за допомогою спеціального пристрою – аналого-цифрового перетворювача (АЦП). АЦП перетворює напругу у двійковий код [12].

Точність перетворення АЦП визначається його роздільною здатністю – мінімальною зміною величини аналогового сигналу, яка може бути перетворена даними АЦП. Цей параметр безпосередньо пов'язаний з розрядністю АЦП.

Розрядність АЦП визначає кількість дискретних значень, які АЦП може видати на виході. У двійкових АЦП, які сьогодні використовуються найчастіше, розрядність вимірюється у бітах. Наприклад, двійковий 10-розрядний АЦП здатний видати 1024 дискретних значень 2^{10}.

На практиці роздільна здатність АЦП обмежена співвідношенням сигнал/шум вхідного сигналу. Велика інтенсивність шумів робить неможливою розрізнення сусідніх рівнів вхідного сигналу, що погіршує роздільна здатність. При цьому реальна роздільна здатність описується ефективною розрядністю, яка менша, ніж паспортна розрядність АЦП. При перетворенні сигналу з високим рівнем шуму молодші розряди вихідного коду містять шум, що робить недоцільним їх використання. Для досягнення заявленої розрядності співвідношення сигнал/шум вхідного сигналу має бути приблизно 6 дБ на кожен біт розрядності.

Також усім АЦП мають похибки, пов’язані з нелінійностями, що є наслідком фізичної недосконалості АЦП. Це призводить до зниження точності вимірювань.

Іншим важливим параметром АЦП є частота дискретизації. Вона визначає час на протязі якого АЦП здійснює перетворення аналогового сигналу у цифровий код. Забезпечення високої точності перетворення вимагає фіксацію сигналу на вході АЦП на протязі всього часу вимірювання. Враховуючи, що реальні АЦП не можуть зробити аналого-цифровое перетворення миттєво, вхідний аналоговий сигнал має утримуватися постійним за допомогою спеціальних пристроїв. Такі пристрої називаються пристроями вибірки-зберігання (ПВЗ). ПВЗ, як правило, зберігає вхідну аналогову напругу на конденсаторі, який з'єднаний із входом через ключ: при замиканні ключа відбувається вибірка вхідного сигналу (конденсатор заряджається до вхідної напруги), при розмиканні – зберігання. Багато сучасних АЦП, що виконані у вигляді інтегральних мікросхем, містять вбудований ПВЗ.

Одним з широко розповсюджених АЦП є мікросхема HX711 [13] (рис. 1.3).

АЦП на базі інтегральної мікросхеми HX711 має частоту дискретизації 24 біт та вбудований малошумний операційний підсилювач. До складу мікросхеми входить мультиплексор, дозволяє вибирати один із двох наявних вхідних

каналів. Канал А дозволяє задати коефіцієнта підсилення – 64 або 128. Канал працює з фіксованим коефіцієнтом, що дорівнює 32.

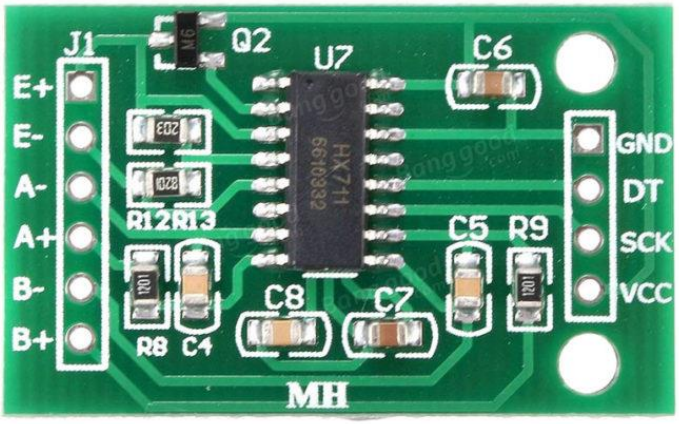


Рис. 1.3. АЦП на базі мікросхеми HX711.

До складу HX711 входить стабілізатор напруги. На вхід синхронізації може бути поданий будь-який імпульсний сигнал від зовнішнього джерела, також АЦП допускає роботу від вбудованого генератора.

Основні технічні характеристики HX711:

– Розрядність АЦП – 24 біт.

– Підсилення для входу А – 64 або 128.

– Підсилення для входу В – 32.

– Частота вимірювань – 10 або 80 разів на секунду.

– Напруга живлення – 2,6-5,5 В.

– Струм споживання – менше 10 мА.

– Вхідна напруга – ±40 мВ.

Однією з переваг HX711 є сумісність з платформою Arduino та іншими мікроконтролерами, що суттєво спрощує підключення та передачу даних від цього АЦП.

Обробку та передачу сигналу від АЦП може здійснювати будь-який обчислювальний пристрій, що має відповідні цифрові входи. Водночас, у випадку IoT такий пристрій має бути встановлений всередині або біля

обладнання до якого підключені датчики. Тому доцільно використовувати максимально дешеві та компактні пристрої з мінімальним споживанням електроенергії. Цим вимогам відповідають сучасні мікроконтролери.

Мікроконтролер – це мікросхема, призначена для управління електронними пристроями. Мікроконтролер поєднує функції процесора і периферійних пристроїв на одному кристалі, містить оперативну та (або) постійну пам’ять. Основною відмінністю від мікропроцесора є наявність інтегрованих в мікросхему переферійних пристроїв (вводу-виводу, таймерів, АЦП, компараторів та інших).

Виробники мікроконтролерів пропонують різноманітні лінійки пристроїв із різними технічними характеристиками та вартістю. До них відносяться: MCS 51 виробництва Intel; ESP8266 и ESP32 пропонує компанія Espressif; MSP430 від Texas Instruments; ARM розробником яких є ARM Limited; AVR виробництва Atmel; PIC від компанії Microchip та багато інших.

Безпосередньо використання мікросхеми потребує створення печатної плати та значної кількості роботи з монтажу всіх необхідних компонентів. Тому для задач прототипування широкого готові універсальні платформи, зокрема Arduino.

Arduino – це електронна платформа з відкритим вихідним кодом, яка включає просте у використанні апаратне та програмне забезпечення. Плати Arduino дозволяють зчитувати вхідні сигнали датчиків (світла, температури, маси, тощо) та отримувати інформацію з зовнішніх джерел. На основі цих даних мікроконтролер може формувати сигнали керування та надсилати повідомлення в інші системи. Для програмування використовується мову Arduino, що основана на мові Wiring, і програмне забезпечення Arduino (IDE), побудоване на базі Processing.

Класична версія цієї платформи побудована на базі мікроконтролерів Atmel, але сьогодні є моделі з мікроконтролерами Intel, AVR та іншими. На рис. 1.4 показана одна з найбільш розповсюджених моделей – Arduino Uno.

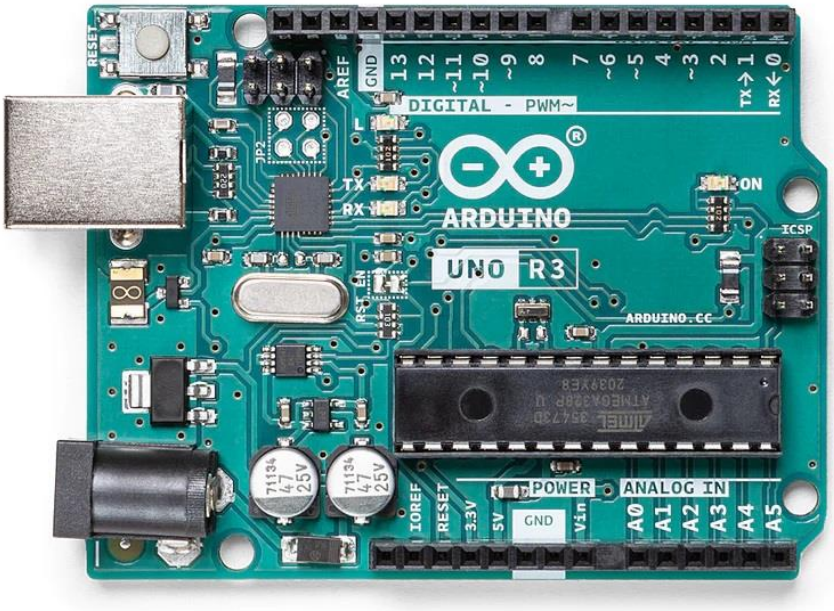


Рис. 1.4. Платформа Arduino Uno Rev 3.

Плата на основана на мікроконтролері ATmega328P, який має 14 цифрових входів/виходів. З них 6 можна використовувати як ШІМ-виходи, 6 аналогових входів, керамічний резонатор 16 МГц (CSTCE16M0V53-R0), USB-інтерфейс, роз'єм живлення, роз'єм ICSP і кнопку скидання. Плата містить програматор та схему живлення мікроконтролера. Тобто їх достатньо просто підключити до комп'ютера за допомогою USB-кабелю для того, щоб почати роботу з системою. Основні технічні характеристики плати Arduino Uno Rev 3:

– Мікроконтролер – ATmega328P.

– Робоча напруга – 5В.

– Вхідна рекомендована напруга – 7-12В.

– Максимальний діапазон вхідної напруги – 6-20В.

– Цифрові входи/виходи – 14 шт. (з них 6 підключені до ШІМ).

– Кількість виводі широтно-імпульсного модулятора (ШІМ) – 6.

– Кількість аналогових входів – 6.

– Постійний струм на портів вводу/виводу – 20 мА.

– Постійний струм виводу живлення 3.3В – 50 мА.

– Флеш пам’ять – 32 КБ (ATmega328P) з них 0.5 КБ використовує завантажувач програм.

– SRAM – 2 КБ (ATmega328P).

– EEPROM – 1 КБ (ATmega328P).

– Частота тактового генератора – 16 МГц.

– Габаритні розміри – 68,6 х 53,4 мм.

– Вага – 25 г.

1.4. Пристрої для передачі даних через мережу Інтернет

За принципом передачі сигналу всі мережеві пристрої поділяються на дві групи: дротові та бездротові [14, 15].

Найбільш поширеним стандартом для передачі даних через дротову мережу є технологія Ethernet. Для підключення пристрої за допомогою цієї технології використовується кабель типу «вита пара». До другої групи відносяться пристрої, що забезпечують передачу за стандартами IEEE 802.11 (Wi-Fi), Bluetooth, IEEE 802.16 (WiMAX) та іншими.

Для платформи Arduino випускаються модулі розширення, які дозволяють підключити платформу до мережі Ethernet. Прикладом такого модуля є плата розширення Arduino Ethernet R3 (рис. 1.5).

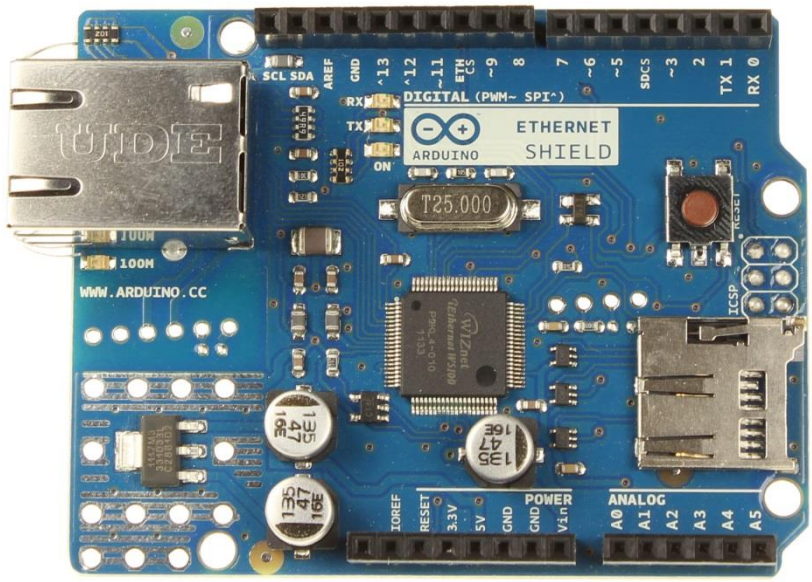


Рис. 1.5. Плата розширення Arduino Ethernet R3.

Плата Aduino Ethernet R3 побудована на базі Ethernet-контролера Wiznet W5100, що дозволяє підключатися до локальної мережі та Інтернет. Wiznet W5100 підтримує стек мережевих протоколів (IP), дозволяє працювати з протоколами TCP та UDP. Мікросхема може підтримувати чотири одночасно відкритих з’єднання через сокети. Для написання програм, що працюють з цією платою розширення, рекомендується використовувати стандартну бібліотеку Ethernet, що входить до пакету бібліотек які постачаються з платформою Arduino. Підключення плати розширення до платформи Arduino здійснюється через спеціальний роз’єм, що дозволяє підключити до Arduino декілька плат розширення, розміщуючи їх одна над іншою.

Основні характеристики Arduino Ethernet R3:

– Робоча напруга – 5В.

– Ethernet-контролер: Wiznet W5100 із вбудованим буфером об'ємом 16 КБ.

– Швидкість з'єднання: 10/100 Мбіт/с.

– Взаємодія з платою Arduino здійснюється через інтерфейс SPI.

З точки зору використання у складі Інтернету речей найбільшого розповсюдження набув метод передачі даних через Wi-Fi. Цьому сприяла поява мікросхем з підтримкою стандарту IEEE 802.11 та низькою вартістю. Зокрема, до них відноситься мікросхема ESP8266. На її базі створено ряд модулів, що дозволяють забезпечувати зв’язок IoT пристроїв через мережу Wi-Fi.

Одним з таких модулів є NodeMCU ESP8266 (рис. 1.6).

Цей модуль використовує чіп ESP8266 версії ESP12E. Мікросхема ESP8266 характеризується ультра низьким споживанням електроенергії та проектувалась спеціально для пристроїв Інтернету речей. Плата суттєво спрощує розробку, оскільки в ній реалізовано підключення по USB та встановлено блок живлення. Всі виводи чіпа розведені на контактни зі стандартним кроком 2.54 мм. За рахунок цього модуль можна вставити в макетну плату та використовувати для швидкого прототипування. Також плата

постачається виробником з прошивкою NodeMCU та можливістю програмування за допомогою мови Lua та через Arduino IDE.

Рис. 1.6. Wi-Fi модуль NodeMCU ESP8266.

Основні характеристики плати:

– Підтримка стандартів WiFi 802.11 b/g/n.

– Підтримка STA / AP / STA + AP режимів.

– Підтримка протоколів TCP/IP з підтримкою до 5 клієнтських підключень.

– Струм на виводах: 15 мА.

– Напруга живлення: 4,5 - 9В (10В максимум), живлення від USB.

– Споживання: у режимі обміну даними ~ 70 мА (200 мА максимум), очікування: <200 мкА.

– Швидкість передачі даних: 110-460800 б/сек.

– Підтримка інтерфейсів передачі даних UART / GPIO.

– Діапазон робочих температур: -40 ~ +125 °C.

– Вага: 18 г.

1.5. Висновки до розділу 1.

1. До складу систем Інтернету речей входять фізичні об’єкти з датчиками та пристроями передачі даних через мережу Інтернет та платформи, що забезпечують вирішення задач зберігання, обробки даних та керування підключеними пристроями.

2. Пристрої, що входять до складу IoT мають бути обладнані датчиками фізичних величин, що забезпечують перетворення цих величин у форму електричного сигналу. Для зниження впливу завад доцільно перетворити сигнал в цифрову форму.

3. Перетворення сигналів аналогових датчиків у цифрову форму здійснюється за допомогою АЦП. Подальшу обробку даних можна виконати за допомого мікроконтролера, що забезпечує мінімізацію витрат.

4. Передача даних через мережу Інтернет може здійснюватись з використанням технології Ethernet та Wi-Fi. Для обох технологій наявні спеціалізовані мікросхеми та модулі. Використання технології Wi-Fi забезпечує кращу мобільність пристроїв, за рахунок відсутності дротових з’єднань, а Ethernet – вищу швидкість передачі даних.

РОЗДІЛ 2. АРХІТЕКТУРА СИСТЕМИ ЗБОРУ ТА АНАЛІЗУ ДАНИХ

2.1. Трирівнева архітектура

Трирівнева архітектура – це модульна клієнт-серверна архітектурна модель програмного комплексу, що передбачає наявність в ньому трьох типів компонентів: клієнтських додатків (рівень клієнта), серверів додатків (рівень логіки) та серверів баз даних (рівень даних) [16].

З точки зору архітектури ці рівні є логічними. Фізичні рівні системи (сервери, комп'ютери, мобільні пристрої) не обов'язково мають відповідати цим трьом рівням. Наприклад, сервери додатків та бази даних можуть знаходитись на одному фізичному сервері. Взаємозв'язки між складовими компонентами цієї архітектури показані на рис. 2.1.

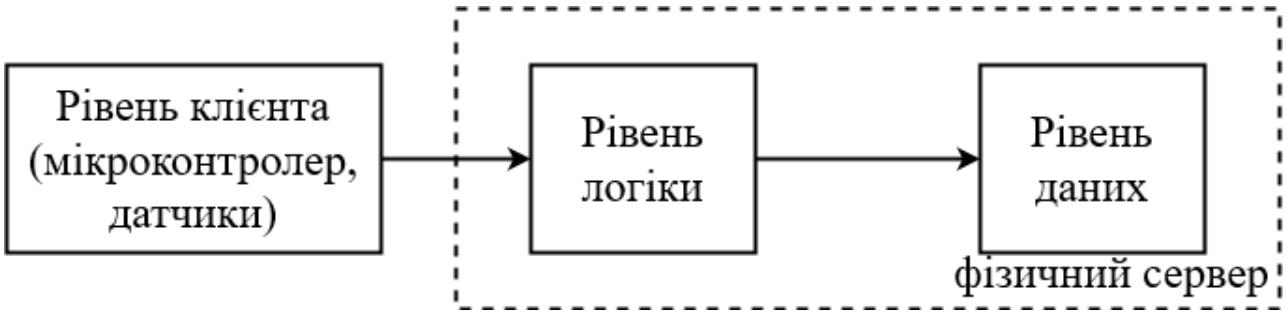


Рис. 2.1. Структурна схема трирівневої архітектури.

Рівень клієнта – це компонент комплексу з яким безпосередньо взаємодіє користувач. Цей рівень може взаємодіяти лише з рівнем логіки. В більшості випадків це серверний додаток, який обробляє запити та формує відповіді для клієнта. Зв'язків із рівнем даних рівень клієнта немає. Це дозволяє підвищити рівень безпеки даних та масштабованості системи. У системі, що проектується, до цього рівня відноситься фізичний об'єкт з розташованими на ньому датчиками та мікроконтролером. Мікроконтролер за заданою програмою зчитує сигнали датчиків та формує запити на зберігання цих даних до рівня логіки.

Рівень логіки (сервер додатків) реалізує основну частину бізнес-логіки. Він відповідає за обробку запитів від клієнта, перевірку прав доступу до системи, валідацію даних, збереження даних у базі даних та формування відповідей клієнту. Цей рівень має безпосередні зв’язки як з клієнтом, так і з базою даних. У більшості випадків сервер додатків являє собою серверне програмне забезпечення, що може бути написане мовами Java, C#, PHP, JavaScrypt, Python та іншими. У дипломній роботі для розробки цього компоненту використано мову JavaScript та платформу Node.js. Сервери додатків проектуються з урахуванням можливості горизонтального масштабування. Це дозволяє розміщати програмне забезпечення на декількох серверах, що забезпечує можливість обробки більшої кількості запитів від клієнтів.

Рівень даних забезпечує безпосередньо довгострокове зберігання та доступ до даних. В більшості сучасних систем цей рівень реалізований на основі спеціального програмного забезпечення – серверів баз даних або систем керування базами даних (СУБД). Водночас, в деяких випадках можуть використовуватись інші механізми збереження даних, наприклад, файлова система. Доступ до цього рівня має лише рівень логіки. У випадку використання СУБД на цей рівень може бути винесена частина бізнес-логіки системи. Найбільш розповсюдженими сьогодні є наступні СУБД: MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB та інші. У дипломній роботі для збереження даних обрано MySQL. Ця СУБД розповсюджується безкоштовно, має версії практично для всіх операційних систем та є однією з найбільш популярних СУБД у світі.

Як показано на рис. 2.1 програмне забезпечення для рівнів логіки та даних може бути встановлено на одному фізичному сервері. У розміщенні клієнтського рівня на сервері з точки зору систем IoT немає сенсу. При збільшенні навантаження (кількості підключених пристроїв та запитів від них) рівень логіки виносять на окремий сервер (або групу серверів).

До переваг трирівневої архітектури відносяться:

– можливість масштабування;

– можливість зміни конфігурації рівнів незалежно один від одного;

– забезпечення безпеки даних (за рахунок відсутності безпосереднього доступу у клієнтів до бази даних);

– відносно низькі вимоги до швидкості каналу передачі даних між клієнтами та сервером додатків;

– можливість зниження технічних вимог до клієнтської частини за рахунок розміщення бізнес-логіки на стороні серверу додатків.

Основними недоліками трирівневої архітектури є:

– необхідність створення програмного забезпечення для всіх рівнів;

– відносна складна процедура розгортання та адміністрування;

– у випадку високих навантажень (великої кількості клієнтів) реалізація серверів додатків та бази даних стає нетривіальною задачею і висуває високі вимоги як до програмного забезпечення, так і до апаратного обладнання.

2.2. Передача даних між компонентами системи

Основні компоненти системи та взаємозв’язки між ними показані на рис. 2.2.

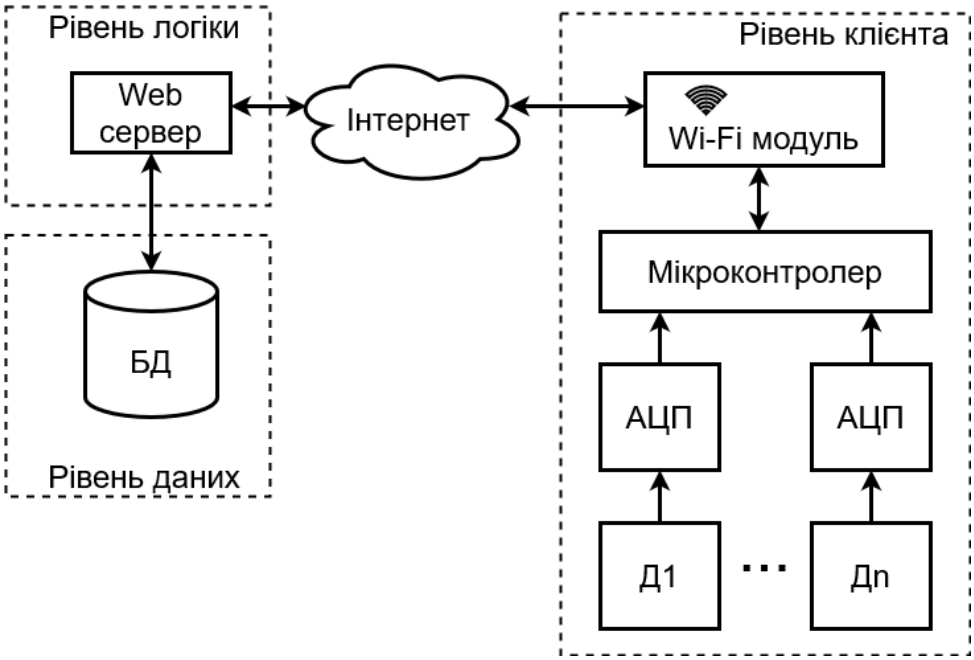


Рис. 2.2. Структурна схема системи збору та аналізу даних.

На схемі (рис. 2.1) літерами Д1…Дn позначені датчики, АЦП – аналого-цифровий перетворювач, БД – база даних.

Інформація в системі передається між компонентами системи в цифровій формі. На рівні клієнту відбувається зчитування сигналів від датчиків, їх обробка мікроконтролером та передача до рівня логіки. Інформація від датчиків надходить у двох формах:

– у вигляді аналогових сигналів;

– у цифровій формі.

Аналогові сигнали перетворюються за допомогою АЦП у цифрову форму, таким чином, всі сигнали надходять до мікроконтролера у формі цілих чисел, що відповідає типу даних integer.

Оскільки система, що проектується, дозволяє підключати декілька датчиків, необхідно сигнали від датчиків супроводжувати ідентифікаторами, які дозволяють визначати якому саме датчику належить сигнал. У найпростішому випадку таким ідентифікатором може бути порядковий номер датчика також представлений цілим числом (тип integer).

Іншою задачею є передача сигналів датчиків у хронологічній послідовності. Її вирішення потребує передачі інформації про час вимірювання. Основна складність полягає в тому, що підключення годинників, які синхронізуються між собою, до всіх мікроконтролерів збільшує вартість системи та складність її налаштування. Зокрема годинники точного часу потребують окремих елементів живлення, для того, щоб забезпечити їх роботу коли пристрій вимкнений. Частковим рішенням є використання інформації про час роботи мікроконтролерів. Робота мікроконтролера синхронізується тактовим генератором, що працює із відомою частотою. Тому кількість імпульсів, що пройшли після подання живлення, є пропорційною тривалості роботи мікроконтролера. Стандартна бібліотека для платформи Arduino містить функції, що дозволяють отримати цей час:

millis() – повертає кількість мілісекунд, тип unsigned long, значення може змінюватись від 1 до 4 294 967 295 мс (~ 50 діб), роздільна здатність дорівнює 1 мс. Після «переповнення» відлік починається з нуля.

micros() – мікросекунди, тип unsigned long, діапазон від 4 до 4294967295 мкс (~70 хвилин), роздільна здатність 4 мкс. Після «переповнення» відлік починається з нуля.

Таким чином, ці функції не дозволяють отримати астрономічний час, але дозволяють визначити різницю у часі, який пройшов між вимірюваннями.

В результаті для кожного вимірювання мікроконтролер може сформувати наступну структуру даних:

```
struct Measurement {
    unsigned int value;
    unsigned int sensor_id;
    unsigned long time;
}
```

де, `value` – результат вимірювання; `sensor_id` – ідентифікатор датчика; `time` – час.

У випадку передачі даних до рівня логіки по одному значенню можна використовувати час сервера та не передавати значення `time`.

Передача даних від клієнта до рівня логіки здійснюється по протоколу HTTP(S). За протоколом HTTP запит виглядає наступним чином:

```
POST /measurement HTTP/1.1
Host: {{host}}
Content-Type: application/json
Content-Length: 45

{
    "sensor_id": 1,
    "value": 56754
}
```

Рівень логіки здійснює перевірку та обробку цих запитів. Якщо вона проходить успішно, формуються запити на збереження інформації у базі даних. Структура даних показана на рис. 2.3. В базі даних створюються дві таблиці: sensor та measurement. Перша містить перелік датчиків та включає три поля:

id – ідентифікатор датчика (первинний ключ);

name – коротка назва датчика;

description – додаткова інформація про датчик (необов'язкове поле, текстовий опис).

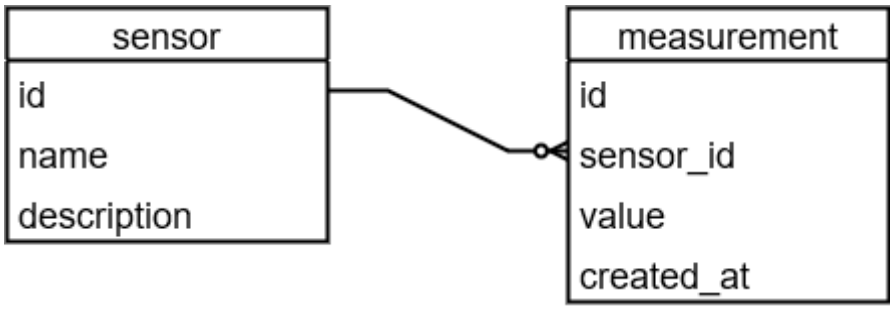


Рис. 2.3. Структура даних, що зберігаються у базі даних.

Друга таблиця містить інформацію про результати вимірювань:

id – ідентифікатор вимірювання (первинний ключ);

sensor_id – id датчика;

value – результат вимірювання;

created_at – дата та час створення запису.

Зв'язок між таблицями створено за полями sensor.id та measurement.id. Тип зв'язку «один-до-багатьох».

Таким чином, зміни у форматах та структурі даних, що передаються між складовими елементами системи, показані на рис. 2.4.

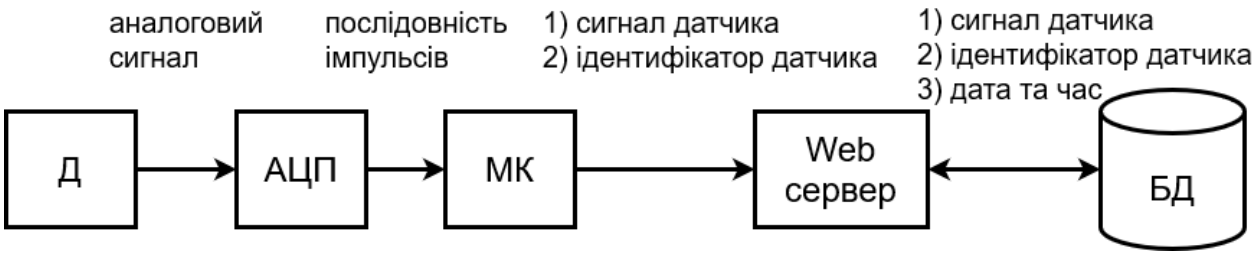


Рис. 2.4. Структура даних, що передається між компонентами системи.

Результати вимірювань (аналогові сигнали) передаються від датчиків (Д) до аналого-цифрового перетворювача (АЦП). Послідовність імпульсів (цифровий код) від АЦП надходить до мікроконтролера (МК), який формує структуру даних та передає її до web-сервера. Web-сервер обробляє отримані дані, доповнює їх інформацією про дату та час отримання та зберігає в базі даних (БД).

2.3. Висновки до розділу 2.

1. Централізована система збору інформації від пристроїв, що входять до складу мережі Інтернету речей, може бути побудована за трирівневою архітектурою. Це дозволить мінімізувати кількість обладнання та спростити обробку даних.

2. Розроблено формати передачі даних між складовими елементами системи, що дозволяють зберігати інформацію від практично необмеженої кількості датчиків.

РОЗДІЛ 3. СИСТЕМА ЗЧИТУВАННЯ ТА ПЕРЕДАЧІ СИГНАЛІВ ВІД ДАТЧИКІВ

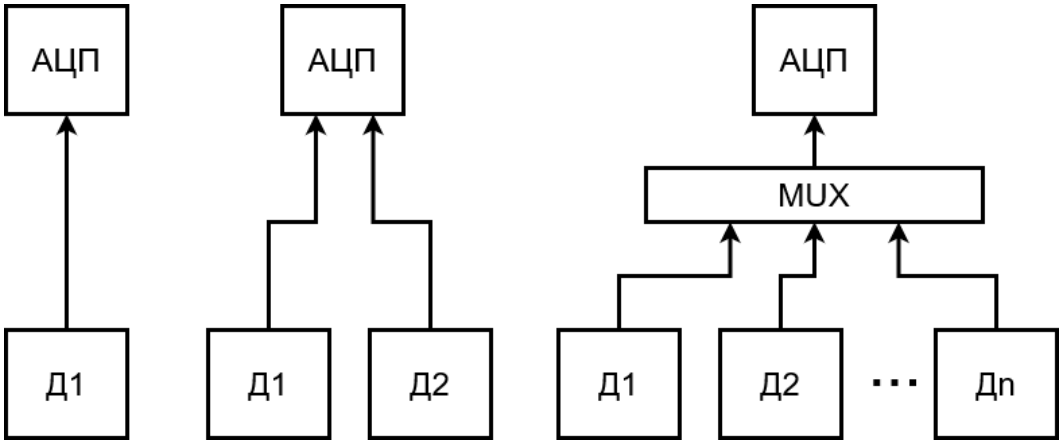
3.1. Підключення аналогових датчиків до мікроконтролерів

В залежності від кількості датчиків, їх типу та вимог до їх розташування, можуть бути використані три основні схеми підключення до мікроконтролерів (МК). На рис. 3.1 показані основні схеми підключення датчиків. На схемі датчики позначені літерою Д, аналогово-цифровий перетворювач – АЦП, мультиплексор – MUX. Також всі датчики на схемі вважаються аналоговими, тобто потребують АЦП для підключення до МК. Цифрові датчики можуть бути підключені напряму до МК.

Вибір цієї чи іншої схеми підключення значною мірою визначається параметрами лінії передачі даних. лінію передачі даних. Основна проблема полягає в тому, що сигнали, які надходять на вхід АЦП є аналоговими. Будь-яка лінія передачі даних має власний електричний опір та ємність, тому збільшення її довжини призводить до зміни сигналу датчика у процесі передачі (його затухання) [17]. Тобто для забезпечення високої точності вимірювань необхідно мінімізувати довжину ліній передачі за якими передається аналоговий сигнал. Це у свою чергу означає, що АЦП бажано розташувати безпосередньо біля датчика маси. Розглянемо детальніше схеми підключення, що представлені на рис. 3.1. Їх основні характерні особливості представлені в табл. 3.1.

Схема «один-до-одного» (рис. 3.1, а) передбачає підключення кожного датчика до власного АЦП. Таке рішення дозволяє забезпечити мінімально можливу відстань між АЦП та датчиком. Вплив параметрів лінії передачі на сигнал датчика при цьому зменшується. Але реалізація такої схеми означає, що ми використовуємо максимальну кількість АЦП, яка дорівнює кількості датчиків. До кожного АЦП необхідно підвести напругу живлення, забезпечити його захист від впливу зовнішніх факторів. Тобто підвищення точності вимірювань досягається за рахунок збільшення кількості обладнання та, відповідно, його вартості. Додатковою перевагою такої схеми є мінімізація часу опитування датчиків. Оскільки для кожного датчика використовується окремий

АЦЕ, цей час визначається часом перетворення АЦП та часом передачі результату до МК.



а) б) в)

Рис. 3.1. Схеми підключення аналогових датчиків до АЦП: а) – один-до-одного; б) – один АЦП – декілька датчиків; в) – схема з мультиплексором

Таблиця 3.1.

Характеристики схем підключення аналогових датчиків до АЦП

№ з/п	Схема підключення	Довжина ліній передачі	Час опитування датчиків	Наближена оцінка вартості
1	Один-до-одного	Мінімальна	Мінімальний	Максимальна
2	Один АЦП – декілька датчиків	Середня	Середній	Середня
3	Схема з мультиплексором	Максимальна	Максимальний	Мінімальна

Схема «один АЦП – декілька датчиків», що показана на рис. 3.1, б, передбачає наявність у мікросхеми АЦП декількох входів. Наприклад, мікросхема HX711 має два аналогових входи, що дозволяє підключити до неї напряму два датчики. Сьогодні такі мікросхеми є широко поширеними, випускаються серійно промисловістю та мають невелику вартість. Така схема є найбільш ефективною коли декілька датчиків мають бути розташовані поруч. Наприклад, одним із варіантів зменшення впливу випадкових похибок, є вимірювання однієї й тієї ж величини декількома датчиками. Оскільки датчики знаходяться поруч, довжина ліній передачі між датчиками та АЦП залишається низькою. Водночас, час опитування збільшується пропорційно кількості

датчиків, оскільки АЦП має виконати перетворення та передачу результатів вимірювань від кожного датчика до МК послідовно. Вартість системи буде меншою порівняно з першою схемою за рахунок меншої кількості АЦП.

Збільшення кількості датчиків, що підключені до одного АЦП, забезпечується у «схемі з мультиплексором» (рис. 3.1, в). Ця схема передбачає підключення датчиків до мультиплексора (MUX), який з'єднує їх з входом АЦП по черзі. Кількість датчиків в цьому випадку обмежена кількістю входів MUX. Така схема дозволяє зменшити кількість мікросхем та, відповідно, вартість системи в цілому. Але в цьому випадку складно забезпечити низьку довжину ліній передачі, особливо у випадках коли датчики мають бути встановлені на значній відстані один від одного. Час опитування датчиків збільшується пропорційно їх кількості. Використання такої схеми є доцільним у випадку компактного розташування груп датчиків.

Мікросхема АЦП HX711 передбачає передачу даних між до МК через послідовний інтерфейс. Сигнали мають форму прямокутних імпульсів. Це дозволяє зменшити вплив зовнішніх завад (порівняно з аналоговими сигналами). Оскільки при використанні прямокутних імпульсів достатньо визначити високий та низький рівень сигналу, а не точну величину сигналу. Тривалість і послідовність імпульсів наведена у документації до мікросхеми. АЦП HX711 здійснює передачу результатів вимірювань блоками по 24 біти та підтримує дві швидкості: 10 або 80 значень за секунду. Також змінити швидкість передачі можна за допомогою зовнішнього генератора тактових сигналів. Передача даних до МК здійснюється із використанням двох виводів мікросхеми: DOUT (цифровий вихід) та PD_SCK (цифровий вхід). Через ці виводи здійснюється передача даних та задається режим роботи мікросхеми. На рис. 3.2 показана послідовність імпульсів під час передачі даних.

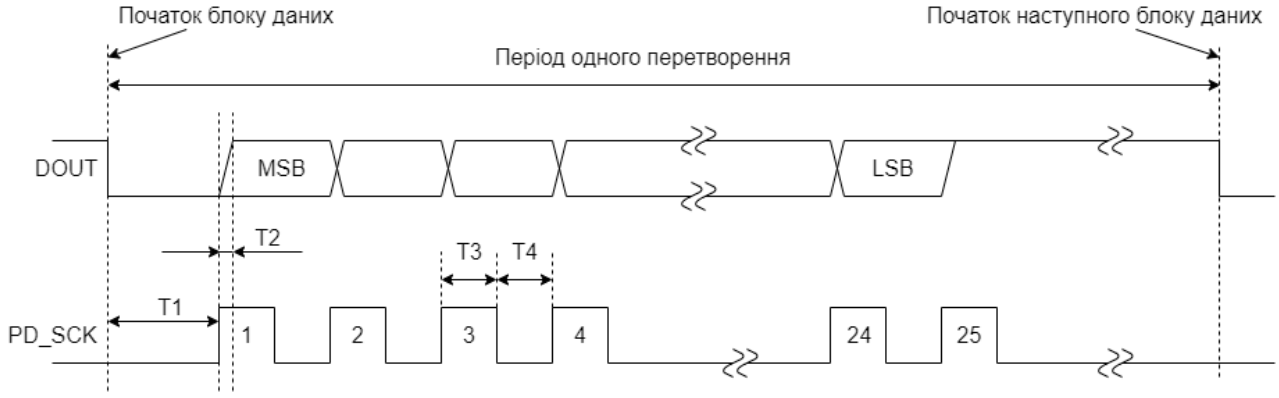


Рис. 3.2. Послідовність імпульсів на виходах МК при передачі даних

Перед початком перетворення даних АЦП формує на виході DOUT сигнал «логічна 1». Після завершення перетворення сигнал на цьому виводі змінюється на «0», що є сигналом готовності даних для зчитування. Коли МК отримує цей сигнал, він послідовно подає на вивід PD_SCK серію з 25-27 прямокутних імпульсів. Ці імпульси АЦП використовує для синхронізації виводу на DOUT бітів значення, що є результатом перетворення аналогового сигналу. Спочатку виводиться старший біт (MSB), 24-им – останній біт (LSB). 25-ий, 26-ий та 27-ий імпульси на виводі PD_SCK задають режим роботи мікросхеми HX711. Тривалості імпульсів та пауз повинні відповідати наступним вимогам: T1 > 0,1 мкс, T2 < 0,1 мкс, 0,2 мкс < T3 < 50 мкс, T4 > 0,2 мкс.

Вплив зовнішніх завад на імпульсний сигнал значно менший ніж на аналоговий, що дозволяє збільшити довжину ліній передачі. При використанні кабелю типу «вита пара» для з'єднання АЦП з МК, його довжина може сягати 100 м без використання додаткових підсилюючих (активних) пристроїв. Можливості під'єднання АЦП до МК, визначаються кількістю вільних виводів МК (два виводи на 1 АЦП) та заданою швидкістю опитування датчиків. Передача даних від АЦП здійснюється послідовно. Час, необхідний для виконання цієї операції визначається наступним чином. За даними виробника мікросхеми HX711, типовими значеннями T3 та T4 є 1 мс. При передачі 27 імпульсів час передачі результатів одного аналого-цифрового перетворення (Тп) становитиме щонайменше 27·2·1 = 54 мс. Таким чином, верхня межу швидкості передачі даних становитиме 1000/54 = 18,5 значень/с.

На рис. 3.3 показана схема підключення тензометричного датчика, АЦП та мікроконтролера.

Тензометричний датчик маси (рис. 3.3) ввімкнений за мостовою схемою. Опір двох з чотирьох його резисторів змінюється під дією зовнішнього навантаження. Відповідно змінюється опір у діагоналі мосту, що призводить до зміни напруги між виводами «INA-» та «INA+» мікросхеми АЦП. Замість тензометричного датчика у цю схему можна включити будь-який аналоговий датчик, головне – забезпечити його електричну сумісність з мікросхемою АЦП (U1, HX711-BF).

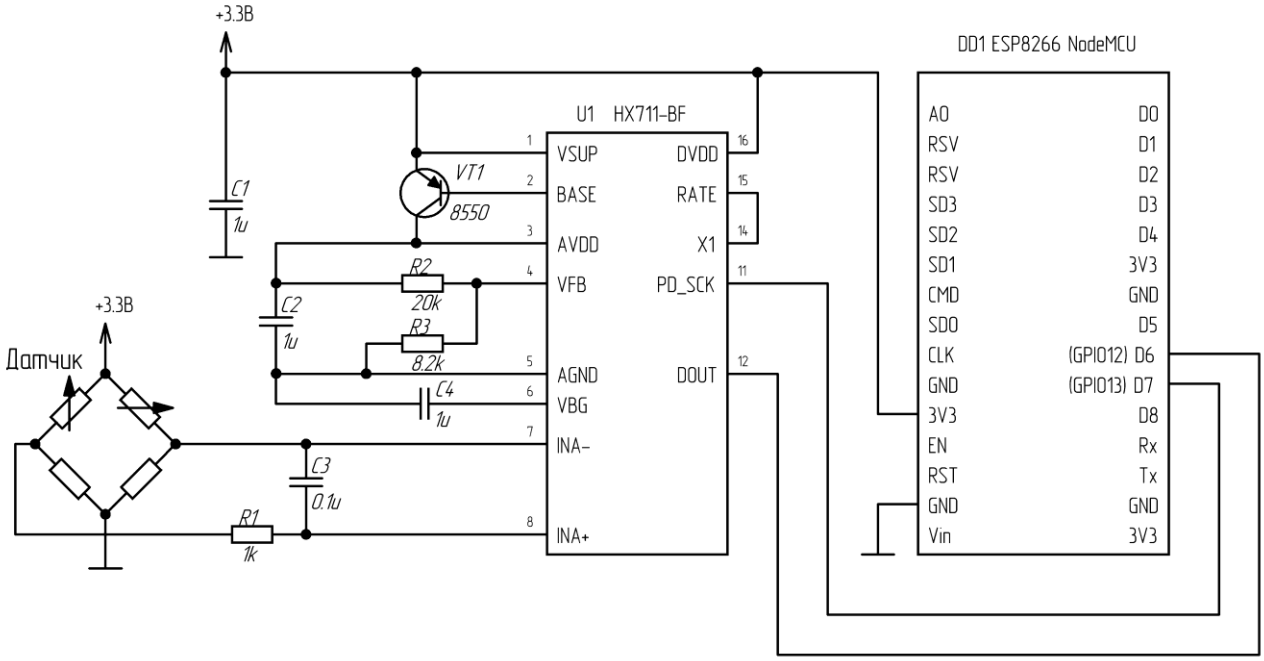


Рис. 3.3. Схема підключення датчика, АЦП та МК.

В якості МК в даному випадку використовується Wi-Fi модуль NodeMCU ESP8266 (DD1). До його складу входять мікроконтролер та Wi-Fi модуль, що значно спрощує підключення та налаштування всієї системи. Виводи PD_SCK та DOUT підключені до виводів D6 та D7 мікросхеми DD1, відповідно. D6 та D7 підключені до інтерфейсу введення-виведення загального призначення (GPIO) та використовуються для зчитування даних з АЦП.

Живлення всієї системи здійснюється від джерела постійної напруги 3,3В. Ця напруга обрана виходячи з технічних вимог мікросхеми DD1. АЦП може працювати у діапазоні напруг 2,6..5,5В, тому напруга 3,3В є достатньою для його нормальної роботи.

Номінали елементів C1, C2, C3, C4, R1, R2, R3 та VT1 обрані відповідно до документації на мікросхему HX711-BF [21].

3.2. Алгоритм зчитування сигналів датчиків

Аналогові датчики передають сигнали у вигляді напруги постійно, але їх перетворення у цифрову форму займає певний час. Передача цифрового значення здійснюється послідовно по одному білу, що у також займає деякий інтервал часу. Мікроконтролер зчитує сигнали з виходу АЦП. Безпосередньо процедура передачі даних визначається характерними особливостями мікросхеми АЦП. Порядок взаємодії з мікросхемою HX711 описаний у розділі 3.1. Водночас, для мікросхеми ESP8266 NodeMCU [19] можна використати бібліотеку HX711 Arduino Library [18], яка створена для платформи Arduino. Ця бібліотека дозволяє суттєво спростити роботу з АЦП. Бібліотека написана на мові програмування C++ та для роботи з АЦП надає об’єкт типу HX711, який містить методи для керування АЦП та зчитування даних.

Сигнали більшості датчиків можна зчитувати безпосередньо після підключення до АЦП. Але коректне визначення сигналів тензометричних датчиків маси потребує їх попереднього калібрування. В більшості випадків датчики маси розташовують під ємністю в якій розміщують об’єкт, масу якого потрібно виміряти. Таким чином, сигнал датчика збільшується на величину маси цієї ємності. Також у датчиків можуть відрізнятись коефіцієнти масштабування. Тому їх потрібно визначити перед використанням системи.

Схема алгоритму калібрування датчика маси показана на рис. 3.4, відповідна програма наведена у лістингу 3.1. На першому етапі підключаються бібліотеки Arduino.h та HX711.h. Після цього задаються номери виводів мікросхеми ESP8266 NodeMCU до яких підключені виводи АЦП DOUT_PIN та SCK_PIN.

На наступному кроці здійснюється скидання параметрів (функції set_scale та tare). Через послідовний порт (Serial) на персональний комп’ютер передаються повідомлення для користувача. Між повідомленнями встановлена затримка на 5000 мс за допомогою функції delay.

Рис. 3.4. Схема алгоритму калібрування датчика маси.

Лістинг 3.1. Калібрування датчика маси

```
#include <Arduino.h>
#include "HX711.h"

// HX711 номери виводів для підключення
const int DOUT_PIN = 12;
const int SCK_PIN = 13;

HX711 adc;
```

```
void setup() {
  Serial.begin(115200);
  adc.begin(DOUT_PIN, SCK_PIN);
}

void loop() {

  if (adc.is_ready()) {
    adc.set_scale();
    Serial.println("Визначення початкової маси... Приберіть навантаження з датчика.");
    delay(5000);
    adc.tare();
    Serial.println("Визначення початкової маси завершено...");
    Serial.print("Розмістіть об'єкт з відомою масою...");
    delay(5000);
    long res = adc.get_units(10);
    Serial.print("Результат: ");
    Serial.println(res);
  }
  else {
    Serial.println("HX711 не знайдено.");
  }
  delay(1000);
}
```

Після того, як користувач встановлює об’єкт з відомою масою m_0 на датчик, програма отримує його сигнал *res* за допомогою функції get_units та

виводить його через послідовний порт. Це значення дозволяє розрахувати коефіцієнт масштабування:

$$k_m = res / m_0.$$

Програма для зчитування сигналів датчика маси наведена у лістингу 2.

Лістинг 3.2. Програма для зчитування сигналів датчика маси.

```
#include <Arduino.h>
#include "HX711.h"

// виводи для підключення HX711
const int DOUT_PIN = 12;
const int SCK_PIN = 13;
const float km = 123; // коефіцієнт масштабування

HX711 adc;

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

  adc.begin(DOUT_PIN, SCK_PIN);

  adc.set_scale(km);
  adc.tare();

  Serial.println("Сигнали датчика:");
}

void loop() {
  Serial.print("маса:\t");
  Serial.print(adc.get_units(), 1);
```

```
    // переводимо АЦП в режим зниженого енергоспоживання
    adc.power_down();
    delay(10000);
    // переводимо АЦП в робочий режим
    adc.power_up();
}
```

Перед використанням програми (лістинг 2) необхідно у змінну `km` записати значення коефіцієнту масштабування, що був розрахований на попередньому кроці (k_m). Програма дозволяє перевірити роботу АЦП та мікроконтролера. У функції `setup` здійснюється налаштування бібліотеки `HX711` та послідовного порту (`Serial`), який дозволяє передавати інформацію на персональний комп'ютер. В основному циклі (функція `loop`) зчитується сигнал датчика (функція `get_units`) та передається через послідовний інтерфейс. Також важливою є можливість переведення АЦП у режим зниженого енергоспоживання. Здійснюється ця операція за допомогою функції `power_down`. Повернення у нормальний режим роботи – за допомогою `power_up`. Ці функції доцільно використовувати коли сигнали датчиків потрібно зчитувати через значний проміжок часу а вся система використовує автономне джерело живлення. Для забезпечення максимальної швидкості зчитування даних АЦП має постійно знаходитись у нормальному режимі роботи.

3.3. Алгоритм передачі даних до системи збору даних

Передача зчитаних сигналів датчиків здійснюється за протоколом HTTP через мережу Wi-Fi. Модуль ESP8266 NodeMCU містить всі необхідні вбудовані елементи (антену, адаптер) для підключення до Wi-Fi мережі. Налаштування та передача даних здійснюється за допомогою двох бібліотек: WiFi.h та HTTPClient.h.

Схема алгоритму підключення представлена на рис. 3.5, програма, яка реалізує алгоритм – у лістингу 3.3.

На етапі ініціалізації необхідно вказати параметри підключення до Wi-Fi мережі (змінні ssid та password) та параметри налаштування АЦП відповідно до прикладу у розділі 3.2.

Підключення до Wi-Fi мережі здійснюється за допомогою методу begin об’єкта WiFi (бібліотека WiFi.h). Встановлення підключення займає певний час, тому у програмі створено цикл, який з періодичністю 1 с перевіряє статус підключення (метод WiFi.status()). Як тільки цей метод повертає значення WL_CONNECTED цикл завершується і програма переходить до налаштування АЦП.

Всі повідомлення про статус підключення виводяться через послідовний інтерфейс (Serial). Це дозволяє підключити модуль ESP8266 NodeMCU до комп’ютера та контролювати його роботу.

У основному циклі програми (функція loop) перевіряється статус підключення до Wi-Fi, зчитується сигнал з АЦП, формується запит та здійснюється його відправка.

Запити відправляються за допомогою бібліотеки HTTPClient.h, яка забезпечує формування та відправку запитів по HTTP протоколу. Для відправки запиту необхідно вказати адресу сервера (у методі http.begin), встановити необхідні заголовки (http.addHeader) та відправити дані за допомогою методу http.POST.

Відправка результатів вимірювання на сервер здійснюється за допомогою запиту типу POST, тіло запиту повинно бути сформоване у JSON форматі, наприклад:

```
{
    "sensor_id": 1,
    "value": 56754
}
```

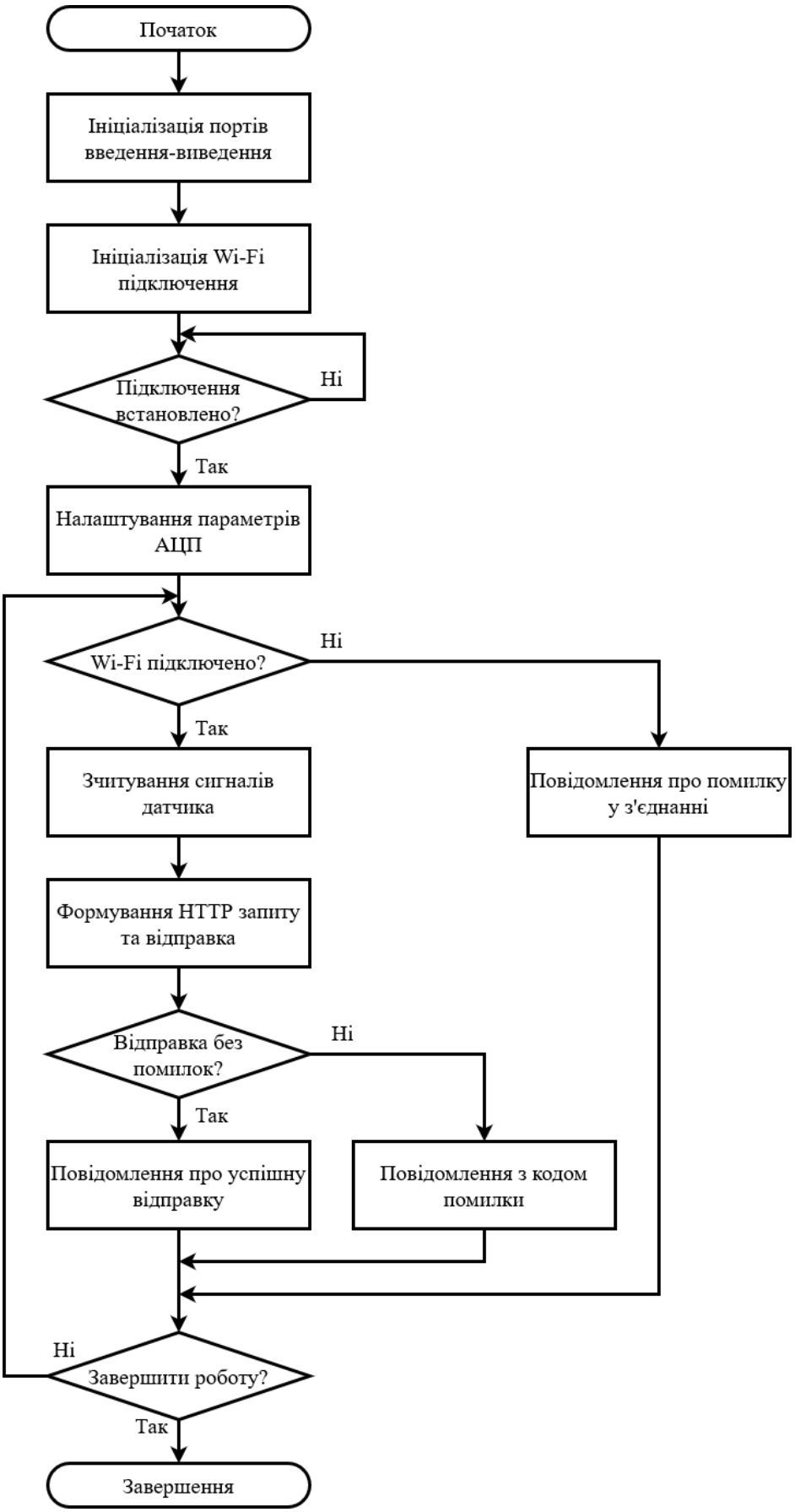


Рис. 3.5. Схема алгоритму відправки сигналів датчиків до системи збору даних.

Також для коректної передачі даних необхідно встановити заголовок:

```
Content-Type: application/json
```

Метод http.POST повертає HTTP код відповіді сервера. Отримати тіло відповіді можна за допомогою методу http.getString().

Результат успішної передачі даних та помилки виводяться через послідовний порт.

Лістинг 3.3. Програма відправки сигналів датчиків до системи збору даних.

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Arduino.h>
#include "HX711.h"
#include <ArduinoJson.h>

// налаштування Wi-Fi
const char* ssid = "network_name";
const char* password =  "password";

// виводи для підключення HX711
const int DOUT_PIN = 12;
const int SCK_PIN = 13;
const float KM = 123; // коефіцієнт масштабування
const int SENSOR_ID = 1; // ідентифікатор датчика

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

  // затримка на 4 с перед ініціалізацією Wi-Fi
```

```
  delay(4000);

  WiFi.begin(ssid, password);

  // чекаємо на підключення
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Підключення до Wi-Fi...");
  }

  Serial.println("Wi-Fi підключено");

  adc.begin(DOUT_PIN, SCK_PIN);

  adc.set_scale(KM);
  adc.tare();

  Serial.println("Сигнали датчика:");
}

void loop() {
  // Перевірка Wi-Fi підключення
  if(WiFi.status()== WL_CONNECTED){

    float sensorValue = adc.get_units();

    HTTPClient http;

    // відправка запиту
```

```
    http.begin("http://my.server.com/measurement");
    http.addHeader("Content-Type", "application/json");

    StaticJsonDocument<128> doc;
    doc["sensor_id"] = SENSOR_ID;
    doc["value"] = sensorValue;
    char jsonString[128];
    serializeJson(doc, jsonString);

    int httpResponseCode = http.POST(jsonString);

    if (httpResponseCode > 0) {
      String response = http.getString();

      Serial.println(httpResponseCode);
      Serial.println(response);
    } else {
      Serial.print("Помилка при відправці запиту: ");
      Serial.println(httpResponseCode);
    }

    http.end();

  } else {
    Serial.println("Помилка у Wi-Fi з'єднанні");
  }
}
```

3.4. Висновки до розділу 3.

1. Проаналізовано можливі схеми підключення аналогових датчиків до аналого-цифрових перетворювачів. Визначено схеми, що забезпечують мінімізацію вартості, впливу зовнішніх завад та максимізацію частоти опитування датчиків.

2. Розроблено алгоритм зчитування сигналів датчиків та перетворення їх у цифрову форму. Створено програму для платформи Arduino, яка дозволяє зчитувати сигнали тензометричних датчиків маси та здійснювати калібрування датчиків цього типу. Визначено, що верхня межу швидкості передачі даних при використанні мікросхеми HX711 становить 18,5 значень/с.

3. Розроблено алгоритм та написано програму передачі даних через мережу Wi-Fi для модуля ESP8266 NodeMCU.

РОЗДІЛ 4. ЦЕНТРАЛІЗОВАНА СИСТЕМА ЗБОРУ ТА АНАЛІЗУ ДАНИХ

4.1. Підсистема зберігання даних

Для зберігання результаті вимірювань у дипломній роботі обрана реляційна система керування базами даних MySQL [21]. Ця система має версії практично для всіх розповсюджених операційних систем, зокрема Windows, Linux, MacOS та відноситься до відкритого програмного забезпечення. Сьогодні підтримку та розробку MySQL здійснює корпорація Oracle. MySQL розповсюджується за подвійною ліцензією. Безкоштовна версія (ліцензія GPL) надає можливість розробникам вільно використовувати MySQL у власних проектах. Комерційна версія передбачає розширену технічну підтримку та можливість розповсюджувати бібліотеки MySQL у складі власних продуктів.

За даними компанії Statista у 2022 році MySQL займала друге місце за популярністю серед систем керування базами даних у світі [22].

Структура даних, які необхідно зберігати у базі даних, розглянута у розділі 2. В реляційних базах даних (БД) інформація зберігається у таблицях між якими можна встановлювати зв’язки. На рис. 2.3 наведено діаграму «сутність – зв’язок», яка відображає перелік необхідних полів з даними та зв’язки між ними. В даному випадку діаграма включає дві сутності sensor та measurement. Між ними встановлено один взаємозв’язок типу «один-до-багатьох». Для реалізації зазначеної діаграми у MySQL необхідно створити базу даних та дві таблиці.

SQL запит на створення бази даних:

```
CREATE DATABASE sensors
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;
```

Тут ми вказуємо назву бази даних (`sensors`) та кодування (`utf8mb4`), яке дозволяє використовувати символи кирилиці.

Перша таблиця sensor створюється за SQL допомогою запиту:

```
CREATE TABLE sensors.sensor (
  id int(11) NOT NULL AUTO_INCREMENT,
  name varchar(255) DEFAULT NULL,
  description text DEFAULT NULL,
  PRIMARY KEY (id)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_unicode_ci;
```

Вона містить три поля (`id`, `name` та `description`) в яких зберігається ідентифікатор, назва та опис датчика, відповідно. Поле `id` використовується в якості первинного ключа. Назва та опис є необов'язковими.

Для створення таблиці measurement можна використати наступний SQL запит:

```
CREATE TABLE sensors.measurement (
  id int(11) NOT NULL AUTO_INCREMENT,
  sensor_id int(11) NOT NULL,
  value int(11) DEFAULT NULL,
  created_at datetime DEFAULT NULL,
  PRIMARY KEY (id)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_unicode_ci;
```

Ця таблиця містить чотири поля (`id`, `sensor_id`, `value` та `created_at`), що містять ідентифікатор вимірювання, ідентифікатор датчика, результат вимірювання, дату та час вимірювання, відповідно. Поле є первинним `id` ключем. Поле `created_at` має тип `datetime` та використовується для зберігання дати та часу у форматі MySQL.

Створення взаємозв’язку між таблицями здійснюється за допомогою запиту:

```
ALTER TABLE sensors.measurement
ADD CONSTRAINT fk_sensor_id FOREIGN KEY (sensor_id)
REFERENCES sensors.sensor (id) ON DELETE CASCADE ON UPDATE CASCADE;
```

Після його виконання поле `measurement.sensor_id` стає зовнішнім ключем. При зміні ідентифікатора датчика (поле `sensor.id`) MySQL буде автоматично оновлювати значення у полі `measurement.sensor_id` і таким чином підтримувати цілісність інформації у базі даних.

Робота з даними у таблицях здійснюється за допомогою SQL запитів.

Основні операції для роботи з даними у таблиці `sensor`.

Читання повного списку всіх датчиків:

```
SELECT * FROM sensor
```

Пошук датчика за його ідентифікатором (замість символу «?» необхідно підставити ідентифікатор датчика):

```
SELECT * FROM sensor WHERE id = ?
```

Вставка запису про новий датчик (замість символів «?» необхідно підставити назву та опис датчика, ідентифікатор автоматично формує MySQL):

```
INSERT INTO sensor(name, description) VALUES (?, ?)
```

Зміна інформації про датчик (замість символів «?» необхідно підставити назву, опис та ідентифікатор):

```
UPDATE sensor SET name = ?, description = ? WHERE id = ?
```

Видалення інформації про датчик (замість символу «?» необхідно підставити ідентифікатор):

```
DELETE FROM sensor WHERE id = ?
```

SQL запити для роботи з даними у таблиці `measurement`.

Формування списку з усіх результатів вимірювань.

```
SELECT * FROM measurement
```

Формування списку з усіма результатами вимірювань для заданого датчика (замість символу «?» необхідно підставити ідентифікатор датчика):

```
SELECT * FROM measurement WHERE sensor_id = ?
```

Пошук вимірювання за його ідентифікатором (замість символу «?» необхідно підставити ідентифікатор вимірювання):

```
SELECT * FROM measurement WHERE id = ?
```

Створення нового запису про вимірювання (замість символі «?» необхідно підставити ідентифікатор датчика, результат вимірювання, дату):

```
INSERT INTO measurement(sensor_id, value, created_at) VALUES (?, ?, ?)
```

Видалення результату вимірювання (замість символу «?» необхідно підставити ідентифікатор вимірювання):

```
DELETE FROM measurement WHERE id = ?
```

Безумовною перевагою реляційних баз даних є можливість використання мови запитів SQL. Вона дозволяє будувати складні вибірки даних, сортувати та групувати їх. Також для бази MySQL існують інструменти, що суттєво спрощують роботу з нею, зокрема, забезпечують доступ до даних через графічний інтерфейс. Під час розробки системи для роботи з даними використовувалось середовище dbForge Studio 2020 for MySQL, що розповсюджується безкоштовно для некомерційного використання. На рис. 4.1 показано скріншот інтерфейсу dbForge Studio 2020 for MySQL.

Система керування базами даних MySQL дозволяє здійснювати резервне копіювання та відновлення даних, що є обов’язковим для забезпечення надійності системи. Експорт результатів вимірювання у різних форматах даних забезпечують додаткові інструменти, зокрема dbForge Studio 2020 for MySQL дозволяє експортувати дані у 14 форматах серед яких XML, CSV, MS Excel, JSON та інші. Результати експорту можуть використовуватись для аналізу даних у таких системах як Excel, MathCad, Matlab, тощо.

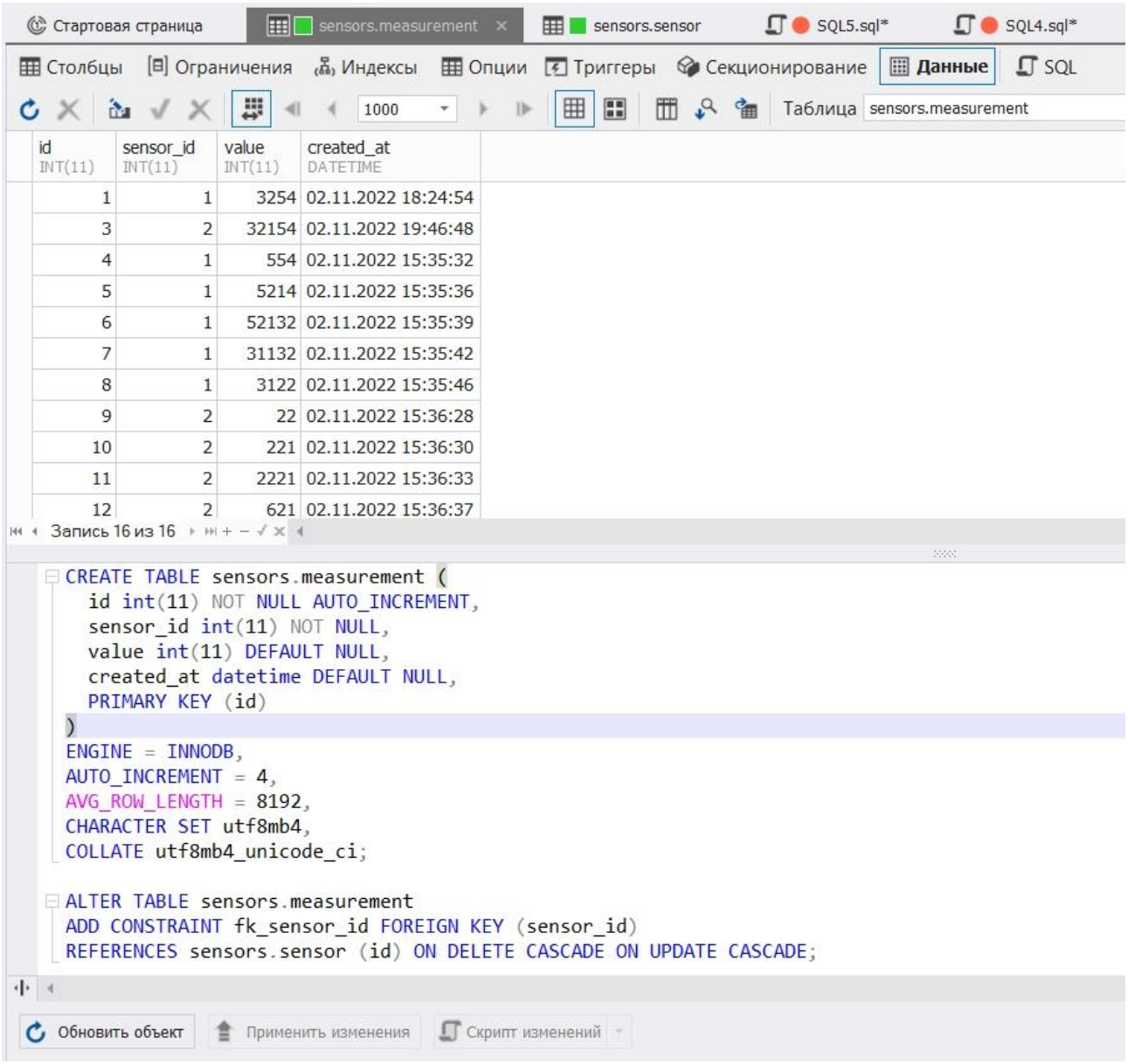


Рис. 4.1. Скріншот інтерфейсу dbForge Studio 2020 for MySQL.

4.2. Аналіз даних за допомогою SQL запитів

Мова SQL дозволяє здійснювати аналіз даних за допомогою вбудованих статистичних функцій. Ці функції використовуються у складі SQL запитів, що дозволяє поєднати можливості фільтрування, групування та розрахунку статистичних показників.

Визначення кількості вимірювань для кожного з датчиків здійснюється за допомогою функції `COUNT` та групування за ідентифікатором датчика:

```
SELECT sensor_id, COUNT(m.value) FROM measurement m
 GROUP BY sensor_id;
```

Визначення суми результатів вимірювань від всіх датчиків, що надійшли 1.11.2022 р.:

```
SELECT SUM(value) FROM measurement m
 WHERE m.created_at BETWEEN '2022-11-02T00:00:00' AND '2022-11-03T00:00:00';
```

Визначення максимального значення для кожного з датчиків за весь період вимірювань:

```
SELECT sensor_id, MAX(m.value) FROM measurement m
 GROUP BY sensor_id;
```

Визначення мінімального значення для кожного з датчиків за весь період вимірювань:

```
SELECT sensor_id, MIN (m.value) FROM measurement m
 GROUP BY sensor_id;
```

Визначення середнього значення для всіх сигналів датчиків, що надійшли 1.11.2022 р.:

```
SELECT AVG(value) FROM measurement m
 WHERE m.created_at BETWEEN '2022-11-02T00:00:00' AND '2022-11-03T00:00:00';
```

Визначення середніх значень окремо для кожного з сигналів датчиків, що надійшли 1.11.2022 р.:

```
SELECT sensor_id, AVG(value) FROM measurement m
 WHERE m.created_at BETWEEN '2022-11-02T00:00:00' AND '2022-11-03T00:00:00'
 GROUP BY sensor_id;
```

Розрахунок величини стандартних відхилень окремо для кожного з сигналів датчиків, що надійшли 1.11.2022 р.:

```
SELECT sensor_id, STDDEV(value) FROM measurement m
 WHERE m.created_at BETWEEN '2022-11-02T00:00:00' AND '2022-11-03T00:00:00'
```

```
 GROUP BY sensor_id;
```

Розрахунок величини стандартної дисперсії окремо для кожного з сигналів датчиків, що надійшли 1.11.2022 р.:

```
SELECT sensor_id, VAR_POP(m.value) FROM measurement m
 WHERE m.created_at BETWEEN '2022-11-02T00:00:00' AND
'2022-11-03T00:00:00'
 GROUP BY sensor_id;
```

4.3. Підсистема обробки даних

Підсистема обробки даних відноситься до рівня логіки у трирівневій архітектурі. Завданням цієї підсистеми є обробка запитів, що надходять за HTTP протоколом, перевірка (валідація) даних, виконання необхідних дій (відправка запитів до бази даних), формування та відправка відповіді клієнту.

Для створення цієї підсистеми у дипломній роботі використано мову JavaScript, платформу Node.js та фреймворк Express [23]. Цей фреймворк містить компоненти, що суттєво спрощують розробку web-додатків, зокрема, маршрутизатор (роутер), компоненти для роботи з запитами та відповідями, різноманітні допоміжні функції. Архітектурно підсистема включає наступні компоненти:

– `index.js` – точка входу, в цьому файлі здійснюється ініціалізація фреймворка, формування маршрутів та передається керування іншим компонентам системи;

– контролери (файли з назвою `…Controller.js`) – отримують об’єкт з даними запиту, передають ці дані компонентам, що виконують валідацію та працюють з базою даних, формують відповідь клієнту;

– валідатори (файли з назвою `validate….js`) – містять функції для валідації даних;

– репозіторії (файли з назвою `…Repository.js`) – містять методи для роботи з базою даних.

Окрім фреймворку Express підсистема використовує наступні бібліотеки:

– JOI [24] – надає можливість створювати правила перевірки даних та безпосередньо перевіряти дані, що надходять від клієнта;

– mysql2 [25] – забезпечує підключення та відправку запитів до MySQL.

В рамках дипломної роботи реалізовано методи для обробки HTTP запитів на читання, створення, оновлення та видалення інформації про датчики та результати вимірювань. Повний код підсистеми наведено у додатку Б.

Код ініціалізації підсистеми наведено у лістингу 4.1.

Лістинг 4.1.

```
const express = require('express');
const dotenv = require('dotenv');
const setupRoutes = require('./routes');

dotenv.config();

const app = express();
// парсинг body запроса
app.use(express.json());

const port = 3000;

setupRoutes(app, process.env);

app.listen(port, () => {
    console.log(`Started on port ${port}`);
});
```

Спочатку здійснюється підключення фреймворку `require('express')` та бібліотеки для роботи зі змінними оточення `require('dotenv')`. Змінні оточення використовуються для передачі параметрів під’єднання до бази даних:

```
MYSQL_PORT=3306
MYSQL_DATABASE=sensors
MYSQL_USER=sensors
MYSQL_PASSWORD=password
```

Після цього здійснюється підключення файлу, що містить налаштування маршрутів роутера `require('./routes')` лістинг 4.2.

Лістинг 4.2.

```
const SensorController =
require('./modules/sensors/controllers/SensorController');
const MeasurementController =
require('./modules/sensors/controllers/MeasurementController')
;
const getPool = require('./db');

module.exports = function setupRoutes(app, env) {
    const connectionPool = getPool();

    app.get('/', (req, res) => {
        res.send('Home');
    });

    let sensorController = new
SensorController(connectionPool);
    app.get('/sensor/index',
sensorController.getAll.bind(sensorController));
    app.get('/sensor/:id',
sensorController.getById.bind(sensorController));
    app.post('/sensor',
sensorController.create.bind(sensorController));
```

```
    app.put('/sensor/:id',
sensorController.update.bind(sensorController));
    app.delete('/sensor/:id',
sensorController.delete.bind(sensorController));

    let measurementController = new
MeasurementController(connectionPool);
    app.get('/measurement/index',
measurementController.getAll.bind(measurementController));
    app.get('/measurement/:id',
measurementController.getById.bind(measurementController));
    app.post('/measurement',
measurementController.create.bind(measurementController));
    app.delete('/measurement/:id',
measurementController.delete.bind(measurementController));
}
```

Всередині цієї функції створюється об’єкт, що містить пул з’єднань з базою даних, який передається в якості параметра в конструктори контролерів.

Далі здійснюється налаштування всіх маршрутів. Кожен з маршрутів налаштовується за допомогою одного з методів: `get`, `post`, `put`, `delete`. Ім’я методу відповідає назві HTTP методу. Перший параметр методу містить маршрут – частину HTTP запиту, яка вказується після імені домену. У другому параметрі передається назва методу контролера, який має бути викликаний якщо надійде запит, що відповідає першому параметру.

Для кожного методу контролера здійснюється виклик методу `bind`, якому передається створений контролер в якості аргументу. Це необхідно для того, щоб встановити контекст виконання, який відповідає створеному контролеру, та зробити доступним пул з’єднань з базою даних всередині метода контролера. Кожен з методів здійснює обробку параметрів запиту, відправляє необхідні запити до бази даних та повертає результат.

Розглянемо детальніше процес збереження інформації про результат вимірювання, діаграму послідовності операцій в якому показано на рис. 4.2. На цій діаграмі опущені службові компоненти фреймворка Express, що виконують обробку HTTP запиту оскільки їх робота детально розглядається в документації фреймоврка.

Порядок збереження результату вимірювання наступний. Клієнт надсилає HTTP запит типу POST, що містить результат вимірювання та ідентифікатор датчика. Структура цього запиту наведена у розділі 2.2.

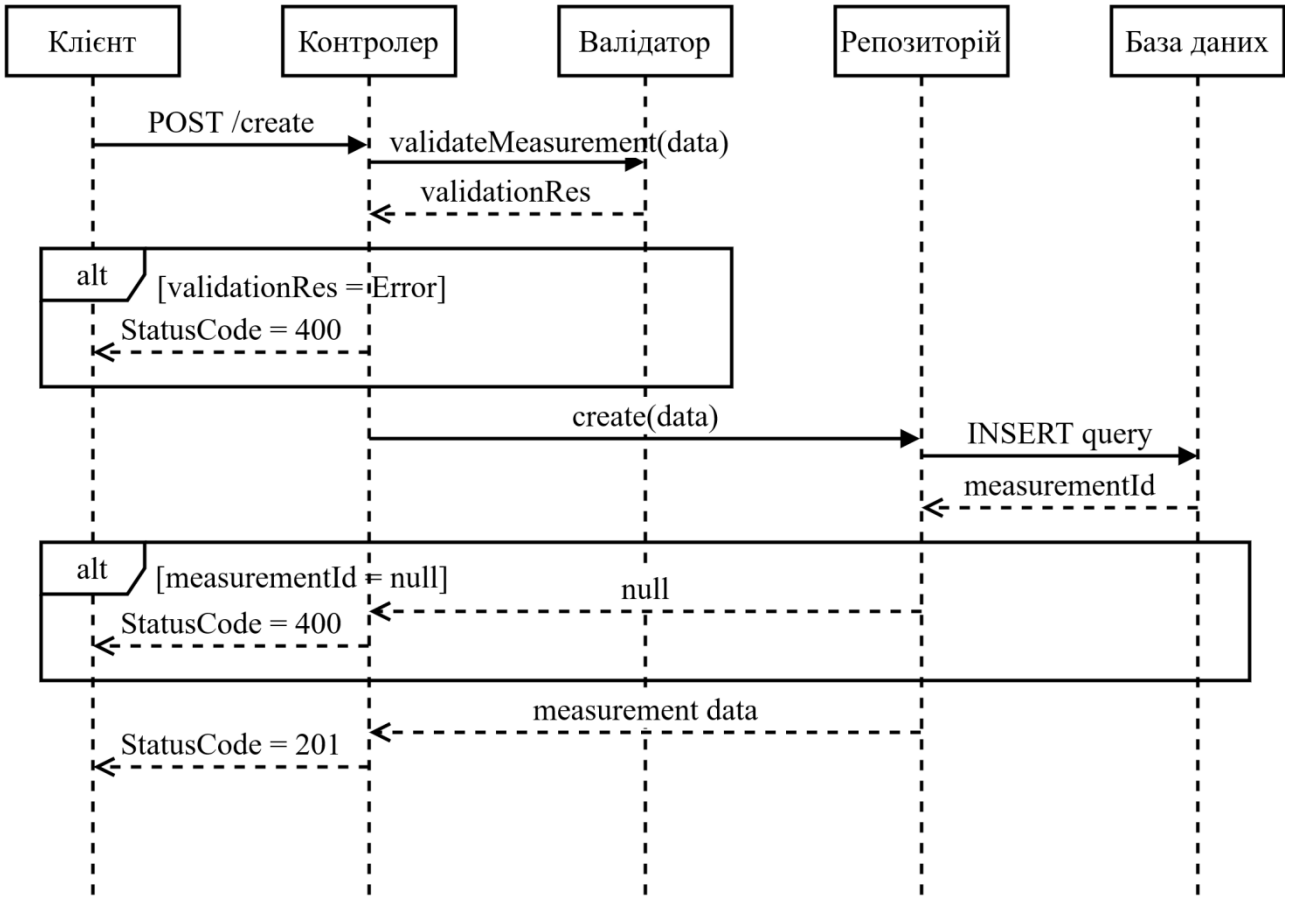


Рис. 4.2. Діаграма послідовності збереження результату вимірювання.

Роутер фреймворка Express розбирає параметри запиту та викликає відповідний метод контролера. В даному випадку буде викликаний метод create (лістинг 4.3).

Лістинг 4.3.

```
async create(req, res) {
    const body = req.body;
```

```
    const validationRes = validateMeasurement(body);
    if (validationRes.error) {
        return
res.status(400).json({"message":validationRes.error.message});
    }

    let measurementData = {...body};
    const now = new Date();
    measurementData.createdAt = now.getFullYear() + '-' +
(now.getMonth() + 1) + '-' + now.getDate() + ' ' +
now.getHours() + ':' + now.getMinutes() + ':' +
now.getSeconds();
    const measurementRepository = new
MeasurementRepository(this.pool);
    const measurement = await
measurementRepository.create(measurementData);
    if (null === measurement) {
        return res.status(400).json({"message":"measurement
creation error"});
    }

    res.status(201).json(measurement);
}
```

Цей метод отримує в якості параметрів два об’єкти: `req` (запит) та `res` (відповідь). Поле `body` об’єкту `req` містить параметри запиту. Вони передаються у функцію `validateMeasurement`, що здійснює їх перевірку. Код цієї функції наведено у лістингу 4.4.

Лістинг 4.4.

```
const Joi = require('joi');

module.exports = function validateMeasurement(data) {
    const schema = Joi.object({
        sensor_id: Joi.number().integer().min(1).required(),
        value: Joi.number().integer().min(0).required()
    });

    return schema.validate(data);
}
```

В цій функції використовується бібліотека JOI за допомогою якої спочатку створюється схема даних, що містить опис двох параметрів (`sensor_id` та `value`) та правила їх перевірки. Для параметру `sensor_id` вказано, що воно є обов'язковим та має бути цілим числом з мінімальним значенням 1. Для параметру `value` вказані аналогічні правила, але мінімальне значення дорівнює 0. Метод виконує `schema.validate` перевірку даних, що передаються функції на відповідність цим правилам. Функція повертає об'єкт з результатами перевірки. Метод контролера перевіряє значення поля `validationRes.error` та у випадку наявності помилок повертає клієнту відповідь з HTTP кодом 400 та описом помилки.

Якщо помилок не було, контролер формує рядок з поточною датою та часом, додає його до даних, що надійшли від клієнта. Далі він викликає метод `create` репозитарію та передає йому сформовані дані.

Репозитарій у свою чергу формує SQL запит на вставку даних у таблицю `measurement` та виконує його:

```
async create(body) {
    const promisePool = getPool().promise();
    const [res, _] = await promisePool.query(
        "INSERT INTO measurement(sensor_id, value,
created_at) VALUES (?, ?, ?)",
```

```
        [body.sensor_id, body.value, body.createdAt]
    );

    if (res.insertId) {
        let rowData = {...body};
        rowData.id = res.insertId;

        return rowData;
    }

    return null;
}
```

Слід зазначити, що Node.js дозволяє виконувати асинхронні запити до зовнішніх ресурсі, таких як бази даних, файлова система та інші. Це збільшує швидкість роботи програми, оскільки основний процес не чекає на результат запиту. Він відправляє запит та продовжує роботу, наприклад, починає обробляти запит від іншого клієнта. Як тільки операційна система чи зовнішні сервер нададуть відповідь на асинхронний запит, платформа Node.js передасть керування функції, яка визначена для обробки цього запиту. Недоліком такого підходу є те, що у програмах з'являється велика кількість функцій, які вкладені одна в одну. Сучасні версії мови JavaScript спрощують написання таких програм за рахунок спеціальних ключових слів: `async ... await`. Метод `create` оголошений як асинхронний (`async`). Він використовує пул з'єднань з базою даних, який створює бібліотека mysql2, та відправляє SQL запит за допомогою метода `query`. Перед викликом вказане ключове слово `await`, що «призупиняє» виконання подальшого коду до моменту повернення результату методом `query`. Якщо запит виконався без помилок у змінну `res` буде записано об'єкт, що містить інформацію про новий запис в базі даних і, в тому числі, ідентифікатор, який автоматично створює MySQL. Метод `create` додає цей

ідентифікатор до інших даних та повертає результат. У випадку помилки метод `create` повертає значення `null`.

Контролер перевіряє отримане значення і у випадку успішної вставки формує відповідь з HTTP кодом 201 та ідентифікатором нового запису, а у випадку помилки, повертає відповідь з кодом 400.

4.4. Висновки до розділу 4.

1. Підсистему зберігання даних створено на основі системи керування базами даних MySQL. Розроблено структуру таблиць, взаємозв’язки між ними та SQL запити, що дозволяють виконувати операції пошуку, створення, оновлення та видалення даних.

2. Розроблено SQL запити для статистичного аналізу сигналів датчиків, що входять до складу IoT системи.

3. Створена підсистема обробки даних на базі платформи Node.js та фреймворка Express, яка здійснює обробку HTTP-запитів, валідацію, збереження даних та формування відповідей.

ВИСНОВКИ

1. В дипломній роботі проаналізована структура та складові елементи IoT систем, визначені види сигналів, що передаються між компонентами системи, розглянуто перетворення сигналів з аналогової в цифрову форму та їх передача через мережу.

2. Розроблено трирівневу архітектуру системи збору та аналізу даних, визначено формати передачі даних між складовими елементами системи, що дозволяють зберігати інформацію від практично необмеженої кількості датчиків.

3. Проаналізовано можливі схеми підключення аналогових датчиків до систем на базі мікроконтролерів. Визначено, що верхня межу швидкості передачі даних при використанні мікросхеми АЦП HX711 становить 18,5 значень/с. Розроблено програми зчитування сигналів датчиків та передачі їх через мережу Wi-Fi для модуля ESP8266 NodeMCU.

4. Створено підсистему зберігання даних на основі системи керування базами даних MySQL. Розроблено SQL запити для статистичного аналізу сигналів датчиків, що входять до складу IoT системи.

5. Створена підсистема обробки даних на базі платформи Node.js та фреймворка Express, яка здійснює обробку HTTP-запитів, валідацію, збереження даних та формування відповідей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Klaus Schwab. The Fourth Industrial Revolution. Currency. – 2017. – 189 с.

2. Peter Waher. Learning Internet of Things. Packt Publishing. – 2015. – 242 с.

3. Cuno Pfister. Getting Started with the Internet of Things: Connecting Sensors and Microcontrollers to the Cloud (Make: Projects). Make Community, LLC; 1st edition. – 2011. – 255 с.

4. Arduino IoT Cloud [Електронний ресурс]. – Режим доступу: https://docs.arduino.cc/arduino-cloud/

5. Google Cloud IoT [Електронний ресурс]. – Режим доступу: https://cloud.google.com/iot-core

6. AWS IoT [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/ru/iot/

7. PTC ThingWorx [Електронний ресурс]. – Режим доступу: https://www.ptc.com/en/products/thingworx

8. Kaa IoT Platform [Електронний ресурс]. – Режим доступу: https://www.kaaiot.com/

9. Microsoft Azure IoT [Електронний ресурс]. – Режим доступу: https://azure.microsoft.com/ru-ru/products/iot-hub/

10. Інформаційні основи побудови телекомунікаційних мереж / В. В. Казимир, В.В. Литвинов, С.М. Шкарлет, С.В. Зайцев // Вісник Чернігівського ДТУ. – Чернігів : ЧД ТУ, 2013. –340 с.

11. Jacob Fraden. Handbook of Modern Sensors / Springer Cham. – 2016. – 758 p.

12. T. S. Rathore. Digital Measurement Techniques / CRC Press. – 2003. – 309 p.

13. AVIA Semiconductor. 24-Bit Analog-to-Digital Converter HX711. [Електронний ресурс]. – Режим доступу: http://image.dfrobot.com/image/data/SEN0160/hx711_english.pdf

14. Андрій Лунтовський, Ігор Мельник. Комп'ютерні мережі та телекомунікації. Університет «Україна». – 2007. – 274 с.

15. А.Г. Микитишин, М.М. Митник, П.Д. Стухляк, В.В. Пасічник. Комп’ютерні мережі [навчальний посібник] – Львів, «Магнолія 2006», 2013. – 256 с.

16. Роберт К. Мартін. Чиста архітектура. Мистецтво розробки програмного забезпечення. Фабула. – 2019. – 368 с.

17. Джулій В.М. Підвищення функціональності і стабільності завадостійких безпроводових інформаційно–комунікаційних систем / В.М. Джулій, Ю.П. Кльоц, В.С. Орленко, В.Ю. Тітова, Ю.В. Хмельницький // Вісник Хмельницького національного університету. Технічні науки. – 2021. – № 1 (293). – С. 12–16.

18. HX711 library [Електронний ресурс]. – Режим доступу: https://github.com/bogde/HX711#how-to-calibrate-your-load-cell

19. ESP8266 NodeMCU [Електронний ресурс]. – Режим доступу: https://randomnerdtutorials.com/esp8266-load-cell-hx711/

20. HX711-BF [Електронний ресурс]. – Режим доступу: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

21. Paul DuBois. MySQL Cookbook, 3rd Edition. O'Reilly Media, Inc. – 2014. – 864 с.

22. MySQL Statistics Ranking of the most popular database management systems worldwide, as of August 2022 [Електронний ресурс]. – Режим доступу: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/

23. Evan M. Hahn. Express in Action. Writing, building, and testing Node.js applications. – 2016. – 256 с.

24. JOI. The most powerful schema description language and data validator for JavaScript. [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/joi

25. Node MySQL 2 [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/mysql2

ДОДАТКИ

ДОДАТОК А

Technical sciences *ISSN 2307-5732*

DOI 10.31891/2307-5732-2021-297-3-46-50
УДК 004.77

В. В. СТАЦЕНКО, О. П. БУРМІСТЕНКОВ, Т. Я. БІЛА, Д. В. СТАЦЕНКО
Київський національний університет технологій та дизайну
О. І. ПАНАСЮК
Київський національний університет імені Тараса Шевченка

РОЗРОБЛЕННЯ КОМП'ЮТЕРНОЇ ЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ ЗБОРУ ДАНИХ ВІД АНАЛОГОВИХ ДАТЧИКІВ

У роботі представлено результати розроблення структури системи збору даних з аналогових тензометричних датчиків маси. Проаналізовано основні варіанти архітектури системи, визначено їх переваги та недоліки. Представлено алгоритм передачі даних від аналого-цифрового перетворювача до мікроконтролера. Розроблено рекомендації щодо проектування системи збору даних з урахуванням умов їх експлуатації.

Ключові слова: мікроконтролер, АЦП, система збору даних, передача даних, Wi-Fi, датчик маси.

V. V. STATSENKO, O. P. BURMISTENKOV, T. Y. BILA, D. V. STATSENKO
Kyiv National University of Technologies and Design
O. I. PANASIUK
Taras Shevchenko National University of Kyiv

DEVELOPMENT OF CENTRALIZED COMPUTER DATA COLLECTION SYSTEM FROM ANALOG SENSORS

The article presents the architecture and operation principles of the system for collecting and analyzing information from strain gauges. These systems are used to determine the equipment performance for transporting a variety of materials. In particular, they are used to control the movement of bulk materials mixtures components. For such technological processes, it is fundamentally important to ensure constant flows intensity.

The paper identifies three variants of sensor connection schemes, analyzes their advantages and disadvantages. It is established that the structure "one ADC – several sensors" allows to reduce equipment costs and at the same time to provide the minimum parameters influence of a transmission line on a useful sensor signal. The "one-to-one" scheme provides the connection of each sensor to its own ADC. "Circuit with multiplexer" allows to increase the number of sensors connected to one ADC. It is established that the best option in terms of reducing the interference effects on the analog signal and the cost of creating a system is the scheme "one ADC – several sensors".

The algorithm of information transfer from ADC to microcontroller (MC) is analysed. It is calculated that HX711 ADC chips provide the maximum data rate of 18.5 values/s.

It is proposed to transfer data between the MC and the server using the TCP protocol because it avoids data loss and provides the necessary data transfer speed. The structure and formats of data that are transmitted from the mass sensor to the ADC, microcontroller, web server and database are proposed.

The main speed, design parameters, advantages and disadvantages of wired and wireless data network between MK and the server are determined. Recommendations for the design of such a network depending on the characteristics of the premises in which the data collection system will be used have been developed.

Keywords: microcontroller, ADC, data acquisition system, data transmission, Wi-Fi, mass sensor.

Постановка проблеми

В умовах сучасного виробництва наявність актуальної та достовірної інформації про поточні параметри технологічних процесів розширює можливості використання обладнання, дозволяє зменшити кількість неякісної продукції та оперативно реагувати на аварійні ситуації. Сьогодні на ринку представлені промислові комплекси та технологічні лінії, які включають необхідні датчики, пристрої керування, мережеві інтерфейси та інше обладнання, що дозволяє максимально автоматизувати роботу, зменшити участь людини та надати оператору інформацію про поточні параметри систем.

Водночас, в ряді випадків на підприємствах використовується обладнання, при проектуванні якого не враховувалась можливість автоматизації його роботи. Також часто застосовується обладнання різних виробників, яке має власні інтерфейси для передачі даних та моніторингу технологічних параметрів. В цих випадках важливо забезпечити створення систем збору та аналізу даних з мінімальним втручанням у конструкцію обладнання та використанням спільних інтерфейсів передачі даних.

У роботі розглянуто архітектуру та принципи роботи системи збору та аналізу інформації з тензометричних датчиків маси. Такі системи застосовуються для визначення продуктивності різноманітних транспортерів. Зокрема, вони використовуються для контролю руху компонентів сумішей сипких матеріалів [1]. В цьому випадку принципово важливим є забезпечення постійної інтенсивності потоків. Датчики маси дозволяють отримати інформацію про кількість матеріалу, що знаходиться на поверхні транспортувальних пристроїв у кожен окремий момент часу. Ці дані можна використати для прогнозування якості суміші і корегування роботи технологічного обладнання. У сучасних виробництвах кількість зон, в яких необхідно здійснювати такий контроль, може вимірюватись десятками, що зумовлює доцільність централізованої обробки цих даних.

Система збору даних має відповідати наступним вимогам – забезпечувати:

– перетворення аналогових сигналів датчиків у цифровий формат.

– передавання даних через мережу Інтернет.

– можливість горизонтального масштабування (збільшення кількості пристроїв).

– можливість збереження даних для подальшого аналізу.

До складу системи мають входити прилади, які серійно випускаються промисловістю.

Кількість додаткових пристроїв має бути мінімальною.

Необхідно мінімізувати (в ідеальному випадку – уникнути) кількість монтажних робіт у промислових приміщеннях.

Аналіз останніх джерел

Сьогодні проблемам побудови комп'ютерних мереж присвячено велику кількість наукових робіт. Це обумовлено передусім широким використанням різноманітних каналів зв'язку та обмеженнями, що виникають при використанні того чи іншого методу передавання даних. Принципи побудови сучасних телекомунікаційних мереж представлені у роботі [2]. Авторами розглянуто види каналів передачі даних, методи кодування сигналів, захисту від перешкод, перетворення сигналів з аналогової у цифрову форму, принципи побудови комп'ютерних мереж. Також ряд робіт, зокрема [3], присвячено проблемам бездротової передачі даних. Створення таких мереж дозволяє уникнути додаткових витрат на проведення монтажних робіт. Водночас, такий спосіб передачі даних зумовлює появу додаткових проблем, пов'язаних з захистом від перешкод.

Таким чином, створення систем передачі даних в промислових умовах цілком можливе на базі обладнання, що серійно випускається. Водночас, побудова архітектури такої мережі має здійснюватись з урахуванням конкретних промислових умов, параметрів первинних інформаційних сигналів, вимог до швидкості передавання даних.

Метою роботи є розробка структури централізованої системи збору даних з аналогових тензометричних датчиків маси.

Виклад основного матеріалу

Структурна схема системи збору та аналізу даних представлена на рис. 1. Вимірювання маси матеріалу здійснюється за допомогою тензометричних датчиків маси (ТДМ1…ТДМn). Вибір датчиків саме цього типу зумовлений їх високою точністю та практично лінійною характеристикою (залежністю між масою та вихідним сигналом). Це дозволяє уникнути необхідності у додаткових перетвореннях під час аналізу сигналів датчиків. За своєю фізичною природою сигнали датчиків є аналоговими. Перетворення у цифрову форму здійснюється за допомогою аналогово-цифрових перетворювачів (АЦП). Інформація від АЦП зчитується мікроконтролером (МК). Отримані результати вимірювань надсилаються через мережу Інтернет до центрального сервера, який зберігає їх у базі даних (БД).

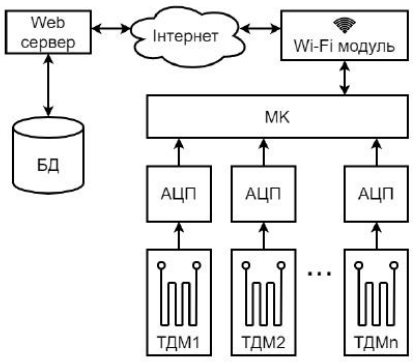


Рис. 1. Структурна схема системи збору даних

Розглянемо детальніше лінію передачі даних. Сигнали датчиків маси, які передаються на вхід АЦП, є аналоговими. Це автоматично накладає суттєві обмеження на довжину лінії передачі, оскільки її електричний опір та ємність призводитиме до появи додаткових похибок у результатах вимірювань. Мінімізація довжини цієї лінії дещо збільшує вартість системи, оскільки АЦП має знаходитись безпосередньо поряд з датчиком маси. Основні варіанти схем підключення датчиків до АЦП показані на рис. 2. Відповідні якісні характеристики запропонованих схем представлені у таблиці 1.

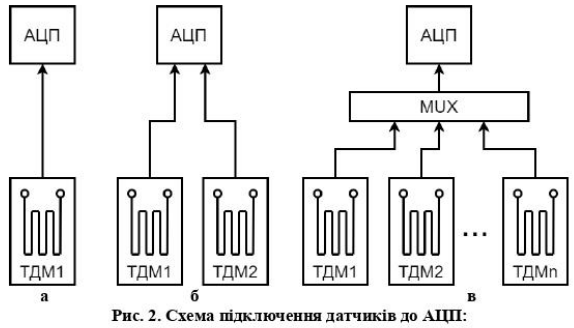


Рис. 2. Схема підключення датчиків до АЦП:
а) один-до-одного; б) один АЦП – декілька датчиків; в) схема з мультиплексором

Таблиця 1

Характеристики схем підключення датчиків до АЦП

№ з/п	Схема підключення	Довжина ліній передачі	Час опитування датчиків	Вартість
1	Один-до-одного	Мінімальна	Мінімальний	Максимальна
2	Один АЦП – декілька датчиків	Середня	Середній	Середня
3	Схема з мультиплексором	Максимальна	Максимальний	Мінімальна

У схемі «один-до-одного» (рис. 2, *а*) кожен датчик підключений до власного АЦП. В цьому випадку можна забезпечити мінімально можливу відстань між датчиком та АЦП, що зменшує вплив параметрів лінії

передачі на корисний сигнал. Водночас, при реалізації такої схеми кількість АЦП дорівнюватиме кількості датчиків, тобто буде максимальною. Окрім того, до кожного АЦП необхідно підвести постійну напругу живлення (в більшості випадків +5В), що також підвищує вартість системи в цілому. Безумовною перевагою такої схеми є мінімальна величина часу опитування датчиків, яка визначається часом перетворення АЦП та тривалістю передачі даних до МК.

Схему «один АЦП – декілька датчиків» (рис. 2, *б*) можливо реалізувати у випадку, якщо мікросхема АЦП має декілька аналогових входів. Сьогодні такі мікросхеми серійно випускаються промисловістю та мають невисоку вартість. Найбільш ефективно використовувати таку схему при вимірюванні маси декількома датчиками. В цьому випадку датчики розташовують поряд один з одним і порівнюють їх сигнали. Це дозволяє оцінити та знизити вплив випадкових похибок. Довжина ліній передачі між АЦП та датчиками залишається низькою, оскільки датчики розташовані поруч. Час опитування збільшується пропорційно кількості датчиків внаслідок того, що АЦП здійснює перетворення та передачу результатів до МК послідовно. Водночас, вартість системи зменшується за рахунок зменшення кількості АЦП.

Збільшити кількість датчиків, підключених до одного АЦП, можна за рахунок використання «схеми з мультиплексором» (рис. 2, *в*). В ній датчики підключаються до мультиплексора (MUX), який по черзі з'єднує їх з входом АЦП. В цьому випадку кількість датчиків обмежена кількістю входів MUX. Це дозволяє зменшити вартість системи за рахунок зменшення кількості мікросхем. Але при цьому суттєво збільшується довжина ліній передачі, що негативно впливає на точність вимірювань. Також час опитування збільшується пропорційно кількості датчиків, що підключені до одного АЦП. Рекомендувати використання такої схеми можна у випадку компактного розташування зон, в яких здійснюється вимірювання.

У реальних умовах місце встановлення датчиків визначається розмірами та розташуванням обладнання. Це означає, що в більшості випадків доцільним є використання схем «один-до-одного» та «один АЦП – декілька датчиків». У роботі запропоновано використання мікросхем АЦП HX711 [4], які мають низьку вартість та забезпечують можливість підключення до двох датчиків маси.

Передача даних між АЦП та МК здійснюється через послідовний інтерфейс. Сигнали передаються у вигляді прямокутних імпульсів, що значно зменшує вплив на них зовнішніх факторів (порівняно з аналоговими сигналами). Тривалість і послідовність імпульсів визначається розробником мікросхеми та наведена у документації. Мікросхема HX711 передає результати вимірювання блоками по 24 біти зі швидкістю 10 або 80 значень за секунду. Швидкість передачі може бути змінена у випадку використання зовнішнього генератора тактових сигналів. Для передачі даних до МК використовуються два виводи мікросхеми: PD_SCK (цифровий вхід) та DOUT (цифровий вихід). Вони дозволяють отримувати дані та задавати режим роботи АЦП. Послідовність передачі даних показаний на рис. 3.

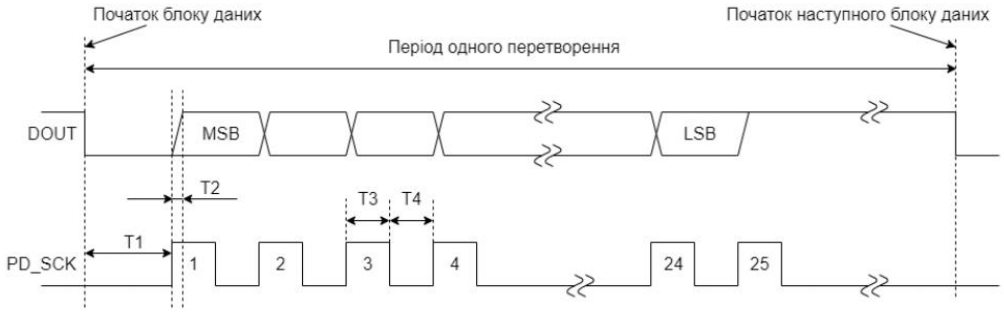


Рис. 3. Сигнали на виходах МК під час передачі даних

Під час перетворення даних АЦП подає на вихід DOUT логічну «1». Зміна сигналу на цьому виводі на «0» означає, що перетворення завершено і дані готові для зчитування. МК послідовно подає на PD_SCK серію з 25–27 прямокутних імпульсів. За появою кожного з цих імпульсів АЦП виводить на DOUT біти значення, що отримано в результаті перетворення аналогового сигналу. Першим виводиться старший біт (MSB), двадцять четвертим – останній біт (LSB). Наявність 25-, 26- та 27-го імпульсів на виводі PD_SCK задає режим роботи мікросхеми HX711. Тривалості імпульсів та пауз мають задовольняти наступним вимогам: $T1 \geq 0{,}1$ мкс, $T2 \leq 0{,}1$ мкс, $0{,}2$ мкс $\leq T3 \leq 50$ мкс, $T4 \geq 0{,}2$ мкс.

Вплив зовнішніх перешкод та електричних параметрів ліній передачі на імпульсний сигнал значно менший ніж на аналоговий, що дозволяє збільшити їх довжину. У випадку з'єднання АЦП з МК кабелем типу «вита пара» довжина лінії передачі може сягати 100 м без використання додаткових підсилюючих пристроїв. Кількість АЦП, які можна під'єднати до одного МК, визначається кількістю вільних виводів мікросхеми МК (два виводи на 1 АЦП) та необхідною періодичністю опитування датчиків. Оскільки зчитування та оброблення даних від АЦП здійснюється послідовно, необхідно оцінити час, що витрачається на ці операції. За даними виробника, типовими значеннями T3 та T4 є 1 мс. Тобто при передачі 27 імпульсів загальний час передачі результатів одного перетворення (Тп) становитиме не менше $27 \cdot 2 \cdot 1 = 54$ мс. Це значення дозволяє оцінити верхню межу швидкості передачі даних, яка становить $1000/54 = 18{,}5$ значень/с.

Передача даних від МК до сервера може здійснюватись через дротову або бездротову мережу за UDP або TCP/IP протоколом. Станом на сьогодні, всі зазначені варіанти передачі даних є стандартизованими, і відповідне апаратне та програмне забезпечення постачається виробниками з різних країн світу. Переваги та недоліки дротових та бездротових ліній передачі наведені у таблиці 2.

Таблиця 2

Основні характеристики ліній передачі даних

Найменування характеристики	Дротова лінія передачі	Бездротова лінія передачі
Максимальна швидкість передачі даних, Мбіт/с	До 1000 (вита пара, в більшості випадків – 100 Мбіт/с) До 10000 (оптоволокно)	До 320 (Wi-Fi, стандарт 802.11n)
Дальність передачі даних, м (без використання додаткового підсилювального обладнання)	100 (вита пара) 80–100 км (оптоволокно)	До 100 м (за умови прямої видимості)
Кількість монтажних робіт	Велика	Низька
Вплив перегородок у приміщеннях	Низький (необхідно створення технологічних отворів для прокладання кабелю)	Високий (необхідно збільшення кількості активного обладнання)

Слід зазначити, що значення швидкостей та дальності передачі, які наведені у таблиці 2, в багатьох випадках обмежені характеристиками комутуючого обладнання. Так при використанні витої пари застосовуються переважно комутатори з максимальною швидкістю передачі даних 100 Мбіт/с. Передача даних через оптоволокно потребує використання додаткових конверторів, що підвищує вартість системи.

Кожен блок даних, який надсилається від МК до сервера, має включати отримане значення від АЦП та порядковий номер датчика. В нашому випадку розмір значення від АЦП становить три байти, а для передачі порядкового номеру доцільно використати тип int (ціле число) розміром два байти (в цьому випадку до системи може бути підключено 2^{16} = 32768 датчиків). Таким чином, розмір 1 блоку даних становить 5 байт. Враховуючи швидкість передачі даних (18,5 блоків/с), ширина каналу передачі даних для одного датчика становитиме $18{,}5 \cdot 5 \cdot 8 = 740$ біт/с.

Передачу даних можна здійснювати за протоколами TCP або UDP. Перший (TCP) передбачає процедуру встановлення зв'язку та передачу підтверджуючих повідомлень після отримання кожного пакету даних, що збільшує кількість даних, які передаються. При використанні протоколу UDP перевірка отримання пакетів з даними приймаючою стороною не здійснюється, що збільшує швидкість передачі даних, але знижує надійність (частину даних сервер може не отримати). В реальних умовах кількість датчиків вимірюється десятками, в окремих випадках, сотнями одиниць. Відповідно, ширина каналу, що необхідна для передачі їх даних, буде на декілька порядків меншою за максимальну швидкість локальної мережі незалежно від типу та протоколу передачі даних.

Тому вибір типу мережі доцільно здійснювати виходячи з архітектурних особливостей приміщень, в яких розташовано промислове обладнання. При роботі у великих приміщеннях без залізобетонних або металевих перегородок доцільно використовувати бездротову мережу, оскільки вона дозволяє знизити кількість монтажних робіт. За наявності перешкод для радіосигналу – дротову мережу, що знизить витрати на активне обладнання (Wi-Fi-роутери). В якості протоколу передачі даних у роботі пропонується використовувати TCP, оскільки він дозволяє уникнути втрати даних, а швидкість передачі даних, як було зазначено вище, в даному випадку не є обмежувальним фактором.

Структура даних, що передаються між складовими елементами системи, показана на рис. 4.

Рис. 4. Структура даних, що передаються між елементами системи

Від тензометричного датчика до АЦП дані надходять в аналоговій формі, далі вони перетворюються у цифрову форму та передаються до МК. Мікроконтролер формує структуру даних, що складається з сигналу датчика та його номеру та надсилає ці дані на сервер. Сервер додає до отриманої структури дату та час отримання даних та зберігає цю інформацію в базі даних. Недоліком запропонованого алгоритму є системна похибка в інформації про дату та час вимірювання, яка визначається часом обробки та передачі даних до сервера. Усунути її можна за рахунок формування інформації про дату та час вимірювання у мікроконтролері. Але така реалізація передбачає підключення до МК мікросхеми точного часу, що збільшує вартість. Також при збиранні даних з декількох мікроконтролерів виникає проблема синхронізації мікросхем точного часу між собою, що у свою чергу призводить до появи додаткових похибок.

Висновки

У роботі розроблено структуру централізованої системи збору даних з аналогових тензометричних датчиків маси. Визначено основні варіанти схем підключення датчиків, проаналізовано їх переваги та недоліки. Встановлено, що структура «один АЦП – декілька датчиків» дозволяє знизити витрати на обладнання та одночасно забезпечити мінімальний вплив параметрів лінії передачі на корисний сигнал датчика.

Проаналізовано алгоритм передачі інформації від АЦП до МК. Встановлено, що при використанні мікросхем HX711 максимальна швидкість передачі даних становить 18,5 значень/с.

Визначено основні швидкісні та конструктивні параметри, переваги та недоліки дротової та бездротової мережі передачі даних від МК до сервера. Розроблено рекомендації щодо проектування такої мережі в залежності від особливостей приміщень, в яких використовуватиметься система збору даних.

Література

1. Автоматизовані комплекси безперервного приготування композицій сипких матеріалів : монографія / В. В. Стаценко, О. П. Бурмістенков, Т. Я. Біла. – Київ : КНУТД, 2017. – 220 с.

2. Інформаційні основи побудови телекомунікаційних мереж / В. В. Казимир, В.В. Литвинов, С.М. Шкарлет, С.В. Зайцев // Вісник Чернігівського ДТУ. – Чернігів : ЧД ТУ, 2013. – 340 с.

3. Джулій В.М. Підвищення функціональності і стабільності завадостійких безпроводових інформаційно-комунікаційних систем / В.М. Джулій, Ю.П. Кльоц, В.С. Орленко, В.Ю. Тітова, Ю.В. Хмельницький // Вісник Хмельницького національного університету. Технічні науки. – 2021. – № 1 (293). – С. 12–16.

4. AVIA Semiconductor. 24-Bit Analog-to-Digital Converter HX711. URL: http://image.dfrobot.com/image/data/SEN0160/hx711_english.pdf

References

1. Avtomatyzovani kompleksy bezperervnoho pryhotuvannia kompozytsii sypkykh materialiv : monohrafiia / V. V. Statsenko, O. P. Burmistenkov, T. Ya. Bila. – Kyiv : KNUTD, 2017. – 220 s.

2. Informatsiini osnovy pobudovy telekomunikatsiinykh merezh / V. V. Kazymyr, V.V. Lytvynov, S.M. Shkarlet, S.V. Zaitsev // Visnyk Chernihivskoho DTU. – Chernihiv : ChD TU, 2013. – 340 s.

3. Dzhulii V.M. Pidvyshchennia funktsionalnosti i stabilnosti zavadostiikykh bezprovodovykh informatsiino-komunikatsiinykh system / V.M. Dzhulii, Yu.P. Klots, V.S. Orlenko, V.Iu. Titova, Yu.V. Khmelnytskyi // Visnyk Khmelnytskoho natsionalnoho universytetu. Tekhnichni nauky. – 2021. – № 1 (293). – S. 12–16.

4. AVIA Semiconductor. 24-Bit Analog-to-Digital Converter HX711. URL: http://image.dfrobot.com/image/data/SEN0160/hx711_english.pdf

В. В. СТАЦЕНКО	ORCID ID: 0000-0002-3932-792X	statsenko.v@knutd.edu.ua
О. П. БУРМІСТЕНКОВ	ORCID ID: 0000-0003-0001-4229	bur42@ukr.net
Т. Я. БІЛА	ORCID ID: 0000-0002-5014-9052	bila.ty@knutd.edu.ua
Д. В. СТАЦЕНКО	ORCID ID: 0000-0002-3064-3109	statsenko.dv@knutd.edu.ua
О. І. ПАНАСЮК		vohigi@gmail.com

Рецензія/Peer review : 20.04.2021 р. Надрукована/Printed :30.06.2021 р.

ДОДАТОК Б

Код підсистеми обробки даних

Файл index.js

```
'use strict';

const express = require('express');
const dotenv = require('dotenv');
const setupRoutes = require('./routes');

dotenv.config();

const app = express();
// парсинг body запроса
app.use(express.json());

const port = 3000;

setupRoutes(app, process.env);

app.listen(port, () => {
    console.log(`Started on port ${port}`);
});
```

Файл routes.js

```
'use strict';

const SensorController = require('./modules/sensors/controllers/SensorController');
const MeasurementController =
require('./modules/sensors/controllers/MeasurementController');
const getPool = require('./db');

module.exports = function setupRoutes(app, env) {
    const connectionPool = getPool();

    app.get('/', (req, res) => {
        res.send('Home');
    });

    let sensorController = new SensorController(connectionPool);
    app.get('/sensor/index', sensorController.getAll.bind(sensorController));
    app.get('/sensor/:id', sensorController.getById.bind(sensorController));
    app.post('/sensor', sensorController.create.bind(sensorController));
    app.put('/sensor/:id', sensorController.update.bind(sensorController));
    app.delete('/sensor/:id', sensorController.delete.bind(sensorController));

    let measurementController = new MeasurementController(connectionPool);
    app.get('/measurement/index',
measurementController.getAll.bind(measurementController));
```

```
    app.get('/measurement/:id',
measurementController.getById.bind(measurementController));
    app.post('/measurement',
measurementController.create.bind(measurementController));
    app.delete('/measurement/:id',
measurementController.delete.bind(measurementController));
}
```

Файл .env

```
MYSQL_PORT=3306
MYSQL_DATABASE=sensors
MYSQL_USER=sensors
MYSQL_PASSWORD=password
```

Файл db.js

```
const mysql = require('mysql2');

let pool = null;

module.exports = function getPool() {
    if (null === pool) {
        pool = mysql.createPool({
            host:'mysql',
            user: process.env.MYSQL_USER,
            database: process.env.MYSQL_DATABASE,
            password: process.env.MYSQL_PASSWORD,
            port: process.env.MYSQL_PORT
        });
    }

    return pool;
}
```

Файл modules\sensors\controllers\MeasurementController.js

```
const MeasurementRepository = require('../repositories/MeasurementRepository');
const validateMeasurement = require('../validators/validateMeasurement');

module.exports = class MeasurementController
{
    constructor(pool) {
        this.pool = pool;
    }

    async getAll(req, res) {
        const sensorId = parseInt(req.query.sensor_id);

        const measurementRepository = new MeasurementRepository(this.pool);
        const measurements = await measurementRepository.getAll(sensorId);

        res.json(measurements);
```

```
    }

    async getById(req, res) {
        const id = parseInt(req.params.id);

        const measurementRepository = new MeasurementRepository(this.pool);
        const measurement = await measurementRepository.getById(id);

        if (null === measurement) {
            return res.status(404).json({'message': 'not found'});
        }

        res.json(measurement);
    }

    async create(req, res) {
        const body = req.body;

        const validationRes = validateMeasurement(body);
        if (validationRes.error) {
            return res.status(400).json({"message":validationRes.error.message});
        }

        let measurementData = {...body};
        const now = new Date();
        measurementData.createdAt = now.getFullYear() + '-' + (now.getMonth() + 1)
+ '-' + now.getDate() + ' '
                                        + now.getHours() + ':' + now.getMinutes() +
':' + now.getSeconds();
        const measurementRepository = new MeasurementRepository(this.pool);
        const measurement = await measurementRepository.create(measurementData);
        if (null === measurement) {
            return res.status(400).json({"message":"measurement creation error"});
        }

        res.status(201).json(measurement);
    }

    async delete(req, res) {
        const id = parseInt(req.params.id);

        const measurementRepository = new MeasurementRepository(this.pool);
        const measurement = await measurementRepository.getById(id);

        if (null === measurement) {
            return res.status(404).json({'message': 'not found'});
        }

        const delRes = await measurementRepository.delete(id);
        if (1 !== delRes.affectedRows) {
            return res.status(400).json({"message":"measurement delete error"});
```

```
        }

        res.json(measurement);
    }
}
```

Файл modules\sensors\controllers\SensorController.js

```
const SensorRepository = require('../repositories/SensorRepository');
const validateSensor = require('../validators/validateSensor');

module.exports = class SensorController
{
    constructor(pool) {
        this.pool = pool;
    }

    async getAll(req, res) {
        const sensorRepository = new SensorRepository(this.pool);
        const sensors = await sensorRepository.getAll();

        res.json(sensors);
    }

    async getById(req, res) {
        const id = parseInt(req.params.id);

        const sensorRepository = new SensorRepository(this.pool);
        const sensor = await sensorRepository.getById(id);

        if (null === sensor) {
            return res.status(404).json({'message': 'not found'});
        }

        res.json(sensor);
    }

    async create(req, res) {
        const body = req.body;

        const validationRes = validateSensor(body);
        if (validationRes.error) {
            return res.status(400).json({"message":validationRes.error.message});
        }

        const sensorRepository = new SensorRepository(this.pool);
        const sensor = await sensorRepository.create(body);
        if (null === sensor) {
            return res.status(400).json({"message":"sensor creation error"});
        }
```

```
        res.status(201).json(sensor);
    }

    async update(req, res) {
        const id = parseInt(req.params.id);
        const body = req.body;

        const validationRes = validateSensor(body);
        if (validationRes.error) {
            return res.status(400).json({"message":validationRes.error.message});
        }

        const sensorRepository = new SensorRepository(this.pool);
        const sensor = await sensorRepository.getById(id);

        if (null === sensor) {
            return res.status(404).json({'message': 'not found'});
        }

        const updatedSensor = await sensorRepository.update(id, body);
        if (null === updatedSensor) {
            return res.status(400).json({"message":"sensor update error"});
        }

        res.json(updatedSensor);
    }

    async delete(req, res) {
        const id = parseInt(req.params.id);

        const sensorRepository = new SensorRepository(this.pool);
        const sensor = await sensorRepository.getById(id);

        if (null === sensor) {
            return res.status(404).json({'message': 'not found'});
        }

        const delRes = await sensorRepository.delete(id);
        if (1 !== delRes.affectedRows) {
            return res.status(400).json({"message":"sensor delete error"});
        }

        res.json(sensor);
    }
}
```

Файл modules\sensors\validators\validateMeasurement.js

```
'use strict';

const Joi = require('joi');
```

```
module.exports = function validateMeasurement(data) {
    const schema = Joi.object({
        sensor_id: Joi.number().integer().min(1).required(),
        value: Joi.number().integer().min(0).required()
    });

    return schema.validate(data);
}
```

Файл modules\sensors\validators\validateSensor.js

```
'use strict';

const Joi = require('joi');

module.exports = function validateSensor(data) {
    const schema = Joi.object({
        name: Joi.string().min(1).max(255).required(),
        description: Joi.string().min(1).max(1000).required()
    });

    return schema.validate(data);
}
```

Файл modules\sensors\repositories\MeasurementRepository.js

```
const getPool = require('../../../db');

module.exports = class MeasurementRepository {
    async getAll(sensorId) {
        const promisePool = getPool().promise();
        let rows, _;
        if (sensorId) {
            [rows, _] = await promisePool.query("SELECT * FROM measurement WHERE
sensor_id = ?", [sensorId]);
        } else {
            [rows, _] = await promisePool.query("SELECT * FROM measurement");
        }

        return rows;
    }

    async getById(id) {
        const promisePool = getPool().promise();
        const [rows, _] = await promisePool.query("SELECT * FROM measurement WHERE
id = ?", [id]);
        if (0 === rows.length) {
            return null;
        }

        return rows[0];
```

```
    }

    async create(body) {
        const promisePool = getPool().promise();
        const [res, _] = await promisePool.query(
            "INSERT INTO measurement(sensor_id, value, created_at) VALUES (?, ?,
?)",
            [body.sensor_id, body.value, body.createdAt]
        );

        if (res.insertId) {
            let rowData = {...body};
            rowData.id = res.insertId;

            return rowData;
        }

        return null;
    }

    async delete(id) {
        const promisePool = getPool().promise();
        const [res, _] = await promisePool.query(
            "DELETE FROM measurement WHERE id = ?",
            [id]
        );

        return res;
    }
}
```

Файл modules\sensors\repositories\SensorRepository.js

```
const getPool = require('../../../db');

module.exports = class SensorRepository {
    async getAll() {
        const promisePool = getPool().promise();
        const [rows, _] = await promisePool.query("SELECT * FROM sensor");

        return rows;
    }

    async getById(id) {
        const promisePool = getPool().promise();
        const [rows, _] = await promisePool.query("SELECT * FROM sensor WHERE id =
?", [id]);
        if (0 === rows.length) {
            return null;
        }
```

```
        return rows[0];
    }

    async create(body) {
        const promisePool = getPool().promise();
        const [res, _] = await promisePool.query(
            "INSERT INTO sensor(name, description) VALUES (?, ?)",
            [body.name, body.description]
        );

        if (res.insertId) {
            let rowData = {...body};
            rowData.id = res.insertId;

            return rowData;
        }

        return null;
    }

    async update(id, body) {
        const promisePool = getPool().promise();
        const [res, _] = await promisePool.query(
            "UPDATE sensor SET name = ?, description = ? WHERE id = ?",
            [body.name, body.description, id]
        );

        if (1 === res.changedRows) {
            let rowData = {...body};
            rowData.id = id;

            return rowData;
        }

        return null;
    }

    async delete(id) {
        const promisePool = getPool().promise();
        const [res, _] = await promisePool.query(
            "DELETE FROM sensor WHERE id = ?",
            [id]
        );

        return res;
    }
}
```