МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

І. Ш. Невлюдов, С. П. Новоселов, О. В. Сичова

NODE-RED ТА ТЕХНОЛОГІЯ ПРОМИСЛОВОГО ІНТЕРНЕТУ РЕЧЕЙ

Навчальний посібник

ВИДАВНИЦТВО І. С. ІВАНЧЕНКА
ХАРКІВ 2024

УДК 681.51
Н40

Рекомендовано Вченою радою
Харківського національного університету радіоелектроніки
(протокол № 6/1 від 30.04.2024 р.)

Невлюдов І. Ш.

Н40 Node-RED та технологія промислового Інтернету речей : Навчальний посібник / І. Ш. Невлюдов, С. П. Новоселов, О. В. Сичова. – Харків: Видавництво Іванченка І. С., 2024. – 207 с.

ISBN 978-617-8332-58-7
DOI: 10.30837/978-617-8332-58-7

Навчальний посібник охоплює такі теми, як впровадження Інтернету речей і промислового Інтернету речей у виробництво, розглядається використання віртуальних приладів та цифрових двійників технологічного обладнання АСУТП в рамках концепції Індустрії 4.0. Подана інформація щодо використання мережі зв'язку 6G для поєднання датчиків та виконавчих пристроїв в інтелектуальній кібер-фізичній системі, а також використання специфічних протоколів комунікації, таких як Modbus та MQTT, для обміну даними та управління промисловим обладнанням. В навчальному посібнику надаються практичні приклади та інструкції щодо використання інструментів, таких як Node-RED та MQTT Mosquitto, для розробки та впровадження систем управління.

Навчальний посібник призначений для підготовки здобувачів освіти в галузі «Електроніка, автоматизація та електронні комунікації», інженерів та технічних спеціалістів у сфері автоматизації та управління виробничими процесами. Також буде корисний аспірантам та фахівцям промислової галузі, робота яких пов'язана з розробкою та організацією виробництв радіоелектронного приладобудування.

УДК 681.51

Рецензенти:

– **Чумаков В. І.**, д-р техн. наук, професор, професор кафедри проектування та експлуатації електронних апаратів, ХНУРЕ;

– **Євсєєв В.В.**, д-р техн. наук, професор, професор кафедри комп'ютерно-інтегрованих технологій, автоматизації та робототехніки, ХНУРЕ;

– **Мосьпан Д. В.**, канд. техн. наук, доцент, доцент кафедри комп'ютерної інженерії та електроніки Кременчугського національного університету ім. М. Остроградського.

ISBN 978-617-8332-58-7
DOI: 10.30837/ 978-617-8332-58-7

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ADU (Application Data Unit) – елемент даних додатка;
AI (Artificial Intelligence) – штучний інтелект;
AMQP (Advanced Message Queuing Protocol) – відкритий стандарт протоколу передачі повідомлень, який призначений для взаємодії між різними програмними системами;
API (Application Programming Interface) – прикладний програмний інтерфейс;
CIoT – стільниковий Інтернет речей;
CLI (CommandLine Interface) – інтерфейс командного рядка;
Cobots (Collaborative Robots) – колоборативні роботи;
CPS (CyberPhysical Systems) – кіберфізичні системи;
CPS PWG (CyberPhysical Systems Public Working Group Framework) – громадська робоча група CPS;
CSS (Cascading Style Sheets) – каскадні таблиці стилів;
DDS (Data Distribution Service) – служба поширення даних;
E2E (End-to-end) – процес перевірки функціональності та цілісності програмного продукту в реальних умовах використання;
HEX (Hexadecimal) – шістнадцяткова система числення;
HTML (Hypertext Markup Language) – мова розмітки гіпертексту;
ICPS (Industrial CyberPhysical Systems) – промислові кіберфізичні системи;
ICS (Industrial Control System) – промислова система управління;
ID (Identity Document) – ідентифікатор;
IIoT (Industrial Internet of Things) – промисловий інтернет речей;
IIRA (Industrial Internet Reference Architecture) – промислова еталонна архітектура Інтернету;
IoE (Internet of Everything) – інтернет всього;
IoT (Internet of Things) – інтернет речей;
IP (Internet Protocol) – протокол мережевого рівня для передавання датаграм між мережами;
KPI (Key Performance Indicators) – ключові показники ефективності;
MEC (Multiaccess Edge Computing;
ML (Machine Learning) – машинне навчання;

MQTT (Message Queuing Telemetry Transport) – протокол телеметрії для передачі повідомлень;
NBIoT (Narrow Band Internet of Things) – вузькосмуговий Інтернет речей;
NIST (The National Institute of Standards and Technology) – Національний інститут стандартів і технологій;
NFV (Network Function Virtualization) – віртуалізація функцій мережі;
OMG (Object Management Group) – некомерційна міжнародна організація у формі консорціуму, відповідає за розробку та затвердження незалежних IT стандартів об'єктно-орієнтованого програмування;
OPC UA (Open Platform Communications Unified Architecture) – уніфікована архітектура комунікацій відкритої платформи;
OWC (Optical Wireless Communications) – оптичний бездротовий зв’язок;
PDU (Protocol Data Unit) – елемент даних протоколу;
PLC (Programmable Logic Controller) – програмований логічний контролер;
RAMI 4.0 (Reference Architectural Model Industry) – еталонна архітектурна модель Індустрія 4.0;
RAN (Radio Access Network) – мережа радіодоступу;
RFID (Radio Frequency Identification) – радіочастотна ідентифікація;
TCP (Transmission Control Protocol) – протокол управління передачею;
TSN (TimeSensitive Networking) – чутлива до часу мережа;
UI (User Interface) – інтерфейс користувача;
WNOC (Wireless NetworkonChip) – бездротові мережі на кристалі;
АСУТП – автоматизована система управління технологічними процесами;
АЦП – аналого-цифровий перетворювач;
БПЛА – безпілотний літальний апарат;
ЄС – Європейський союз;
ІКТ – інформаційно-комунікаційні технології;
ІТ – інформаційні технології;
ОТ – операційні технології;
ПЛК – програмований логічний контролер;
ПК – персональний комп’ютер;
СУБД – система управління базами даних;
ТГц – терагерц;
ЦАП – цифро-аналоговий перетворювач.

ВСТУП

В сучасному світі промислової автоматизації та виробничого управління, технологія промислового інтернету речей (IIoT) є однією з найбільш важливих інновацій. Вона перетворює традиційні виробничі процеси, роблячи їх більш ефективними, надійними та зручними для управління. Одним з ключових інструментів у цьому цифровому перетворенні є Node-RED – потужний, гнучкий та інтуїтивно зрозумілий інструмент для обробки потоків даних.

Інформація, яка подана у навчальному посібнику, призначена допомогти читачам опанувати основи візуального програмування Node-RED, щоб мати можливість використовувати його для вирішення реальних завдань у сфері промислового виробництва. Авторами розглянуто базові концепції Node-RED, його можливості та інтеграцію з технологією IIoT.

Node-RED використовується як платформа для управління периферійними пристроями та роботи з великим обсягом даних від підключених до хмарного сховища пристроїв. Таке візуальне програмне середовище дає змогу створювати функції за допомогою графічного інтерфейсу, що значно спрощує розробку та інтеграцію різноманітних систем автоматизації.

У контексті Індустрії 4.0, де важливим є зв’язок інтернету речей (IoT), аналітики даних, автоматизації та керування, Node-RED виступає як інструмент для створення і керування великими потоками даних між пристроями, додатками та хмарними сервісами. Він може використовуватися для збору даних з сенсорів та пристроїв, обробки цих даних, запуску автоматизованих процесів та взаємодії з іншими системами, такими як бази даних або системи керування виробництвом.

Завдяки своїй гнучкості і простоті використання, Node-RED дозволяє швидко реалізовувати рішення для різних завдань у сфері Індустрії 4.0, таких як моніторинг та діагностика обладнання, оптимізація процесів виробництва, віддалене керування технологічними процесами та багато іншого.

Навчальний посібник містить чотири основних розділи. Перший розділ присвячено розгляду віртуальних приладів та цифрових двійників технологічного обладнання АСУТП. Подано основні характеристики Індустрії 4.0, розглянуто архітектуру кіберфізичних систем, а також технології зв’язку 6G для використання у кіберфізичних системах (CPS) та промисловому Інтернеті

речей (IIoT). Розкривається сутність CPS, як інтелектуальної системи, створеної шляхом поєднання обчислювальних, мережевих та інтерфейсних методів для взаємодії з кібернетичними та фізичними елементами за допомогою датчиків, приводів, вбудованих систем тощо.

В другому розділі розглядаються питання використання основних вузлів Node-RED, а також застосування механізму обміну даними між програмними потоками Node-RED.

В третьому розділі подана інформація щодо використання Node-RED для управління засобами автоматизації, подано приклади використання протоколу Modbus для управління промисловими об’єктами.

В четвертому розділі описані особливості використання протоколів промислового інтернету речей. Особлива увага приділяється протоколу MQTT.

Навчальний посібник призначено для підготовки здобувачів освіти в галузі електроніки, автоматизації та електронних комунікацій, інженерів та технічних спеціалістів у сфері автоматизації та управління виробничими процесами. Також буде корисний аспірантам та фахівцям промислової галузі, робота яких пов’язана з розробкою та організацією виробництв радіоелектронного приладобудування.

1 ВІРТУАЛЬНІ ПРИЛАДИ ТА ЦИФРОВІ ДВІЙНИКИ ТЕХНОЛОГІЧНОГО ОБЛАДНАННЯ АСУТП

1.1 Головні характеристики Індустрії 4.0

Четверта промислова революція або перехід до «Індустрії 4.0» передбачає максимально широке застосування інформаційних технологій на виробництві.

Зараз триває нова промислова революція, Індустрія 4.0, яка в основному пов'язана з сильними науковими досягненнями в галузі інформаційно-комунікаційних технологій (ІКТ). Головним завданням сьогодення є встановлення загального зв'язку між усіма суб'єктами одного або кількох середовищ (людей, речей, систем, пов'язаних даних). Ми можемо розділити цю проблему на два підходи, які потрібно вирішити, а саме: як розробити ефективні моделі для управління різнорідними й важливими обсягами даних, та як оптимально використовувати зібрану інформацію для виробництва знань і отримати послуги, адаптовані до обмежень епохи Індустрії 4.0.

Ця промислова революція, яку можна описати як цифрова, породила особливо складні проблеми, які необхідно вирішити, оскільки кілька останніх передових технологій мають працювати разом, наприклад Інтернет речей, великі дані, штучний інтелект і хмарні обчислення. Для цього необхідні методологія та еталонна архітектура, щоб забезпечити процес переходу від стандартної організації до підключеної або цифрової організації.

Індустрія 4.0 – це більше, ніж бачення майбутнього. Інтелектуальна мережа являє собою величезні можливості для промисловості. Гнучке виробництво може допомогти оптимально використовувати можливості промислового обладнання.

Сьогодні ми живемо цією новою промисловою революцією, яка безпосередньо пов'язана з прискореним прогресом, яким сприяють інформаційно-комунікаційні технології. Він покладається на передачу інформації в реальному часі для моніторингу та дії на фізичні системи, таким чином використовуючи нову парадигму – кіберфізичні системи. Різні системи спілкуються та співпрацюють одна з одною, але також і з людьми, щоб децентралізувати процес прийняття рішень. Індустрія 4.0 зосереджена на підключенні всіх речей до загального хмарного середовища, таким чином

сприяючи розробці нових процесів для комплексного управління виробництвом продукції та наданням послуг.

Сучасна цифрова революція в нашому суспільстві, представлена Інтернетом речей (IoT), соціальними мережами, хмарними обчисленнями, великими даними та штучним інтелектом (AI), пропонує необмежені можливості для розвитку компаній майбутнього. Сьогодні ми говоримо про цифрові організації, створені в результаті інтеграції людей, об'єктів, даних, процесів і послуг, тобто про Інтернет всього (IoE). Однак перехід від традиційної організації до «інтелектуальної» та «цифрової» організації є складним процесом.

Інтернет речей – це мережа фізичних «речей», інтегрованих із датчиками, програмним забезпеченням і відповідними технологіями для взаємодії та обміну інформацією між пристроями та системами через підключення до Інтернету. IoT складається з мільярдів розумних пристроїв, що використовуються в різних секторах, які вимагають вищої пропускної здатності, дуже низької затримки та масштабованості. Вони спілкуються автономно, агрегують і обмінюються повідомленнями через датчики та виконавчі механізми.

Ми використовуємо різні пристрої IoT у різних секторах відповідно до їхніх вимог і можливостей. IoT зробила революцію у світі, де більшість пристроїв можуть повсюдно підключатися. Промислові системи управління (ICS) переходять у промислові IoT із запровадженням технології IoT, яка охоплює всі програми, проблеми та реалізації для всіх галузей. Принцип поєднання пристроїв між собою, який використовується для з'єднання IoT, подано на рис. 1.1 [4].

IoT – це добре розроблена платформа зв'язку та передачі даних, яка постійно розвивається, добре регулюється простими у використанні апаратними та програмними рішеннями. IoT, що використовується в промисловості, називається Industrial IoT (IIoT). Для промислових об'єктів головною задачею є надійність та безпека, тому адаптація Інтернету речей стосується, в першу чергу, засобів забезпечення зв'язку. В першу чергу потрібно забезпечити постійне підключення засобів автоматизації до мережі, що досягається, наприклад, резервуванням каналів мережі зв'язку, використанням спеціального програмного забезпечення та інфраструктури.

Проблеми промислових комунікаційних мереж пов'язані з показниками продуктивності, такими як затримка, надійність, безпека та пропускна здатність. У цьому відношенні IIoT має базуватися на стільниковій мережі, що означає, що

кожен пристрій може підключатися до будь-якої доступної мережі з широким покриттям, пропонуючи надійне мережеве резервування.

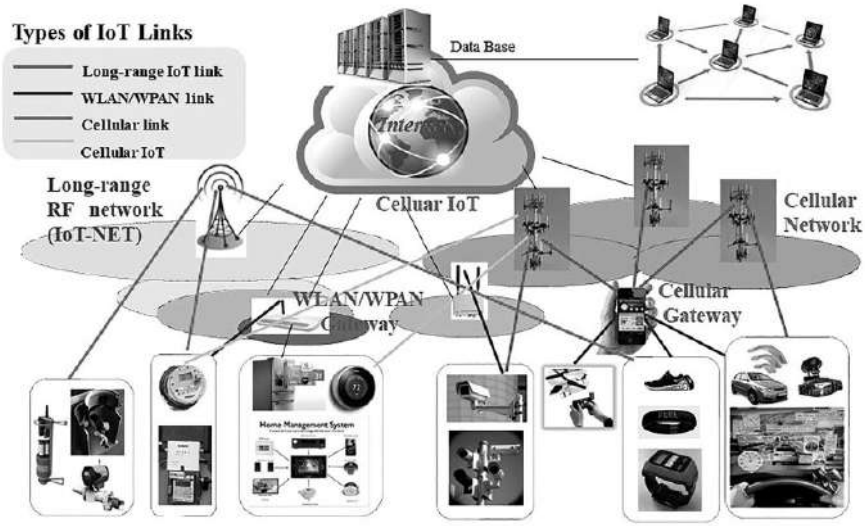


Рисунок 1.1 – Принцип поєднання пристроїв між собою в мережі IoT

Подібним чином резервування програмного забезпечення необхідне для розгортання багатьох мікросервісів для забезпечення основних функцій, щоб у разі збою одного інший міг взяти на себе керування та уникнути перерви в роботі служби.

З точки зору резервування інфраструктури, замість того, щоб покладатися на один центр обробки даних, центр обробки повинен бути доступним у кількох зонах, іншими словами, його слід децентралізувати. Цей розподілений центр обробки даних гарантує, що багато пристроїв продовжують отримувати безперебійні послуги, навіть якщо один центр обробки даних припиняє роботу.

Технології стільникового зв’язку четвертого та п’ятого поколінь уже досліджено та комерційно розгорнуто як засоби підтримки мереж і програм IoT. Однак через величезний попит на розумні або інтелектуальні пристрої зі швидким розширенням мереж IoT 5G не зможе задовольнити всі технологічні потреби, що зростають, такі як автономність, надвеликі, високодинамічні та повністю інтелектуальні послуги.

Прогнозується, що швидке розширення автоматизованих та інтелектуальних мереж IoT випередить можливості бездротових технологій 4G і 5G. Таким чином, дослідження 6G набули популярності в наукових колах і промисловості, що може прокласти шлях для передової розробки IoT і не

тільки. 6G має більшу пропускну здатність і покращені характеристики порівняно з попередніми стільниковими мережами для систем IoT.

Ці рівні потужності прискорять застосування та розгортання мереж IoT на основі 6G у таких сферах, як визначення даних, підключення обладнання, бездротовий зв’язок і керування мережею 6G.

Щоб досягти всіх вищезазначених бездротових з’єднань у загальній потужності, потрібно інтегрувати IoT і машинний інтелект із фізичними системами та процесами для створення кіберфізичних систем (CPS).

CPS – це інтелектуальні системи, створені шляхом поєднання обчислювальних, мережевих та інтерфейсних методів для взаємодії з кібернетичними та фізичними елементами за допомогою датчиків, приводів, вбудованих систем тощо.

Вони включають численні технології різних типів з головною метою керування фізичним процесом, де вони пристосовуються до мінливих ситуацій у режимі реального часу за допомогою систем зворотного зв’язку. CPS будуть присутні в більшості промислових секторів і відповідатимуть парадигмі Industry 5.0. На рис. 1.2 зображено промислову IoT, CPS із зв’язком 6G [4].

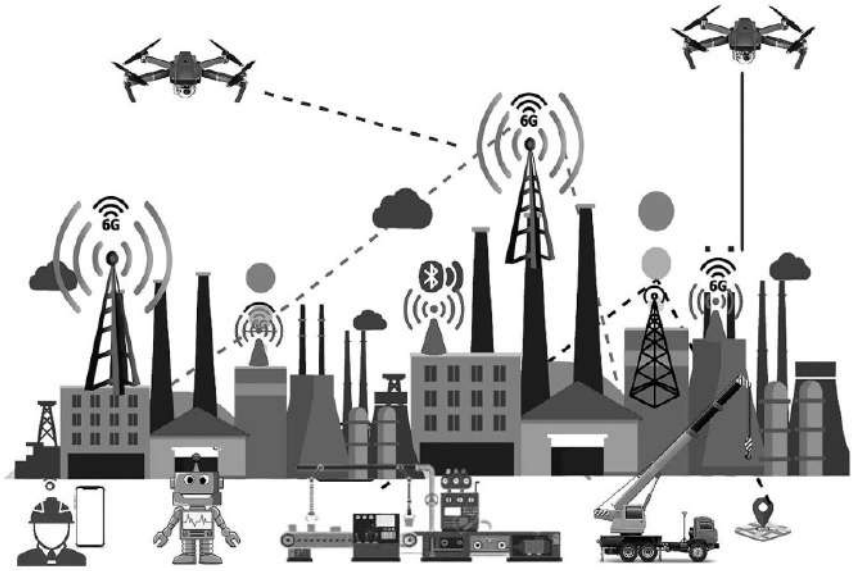


Рисунок 1.2 – Промисловий IoT та CPS із зв’язком 6G

Індустрія 5.0 – це концепція, яка випливає з суспільства 5.0, яка визначає галузь, єдиною метою якої є ефективність і продуктивність. Це підкреслює відповідальність галузі та внесок у суспільство. «Індустрія 5.0» зосереджена на людській присутності в галузях із співробітництвом «людина-машина», стійкій

економіці, яка супроводжує бізнес, і кібернетостійкості. Вона включає в себе інформаційні та промислові технології в тому, що називається CPS, щоб актуалізувати цифрову, інтелектуальну та стійку фабрику.

1.2 Архітектура кіберфізичної системи

Відповідно до Національного інституту стандартів і технологій (NIST), CPS зазвичай визначають як систему, що включає цифрові, аналогові, фізичні та людські компоненти, які взаємодіють, розроблені для функціонування за допомогою інтегрованої фізики та логіки.

Технології CPS включають IoT, промисловий Інтернет, розумні міста, розумну мережу та «розумне» все (наприклад, автомобілі, будівлі, будинки, виробництво, лікарні, побутову техніку), що робить їх придатними для IIoT. NIST заснував громадську робочу групу CPS (CPS PWG) у 2014 році, маючи намір розробити інтелектуальні системи, які включають спроєктовані взаємодіючі мережі фізичних і обчислювальних компонентів. Структура має п’ять підгруп (лексика та еталонна архітектура, варіанти використання, кібербезпека та конфіденційність, синхронізація та сумісність даних).

На рис. 1.3 подана CPS у термінах даної концептуальної моделі [4].

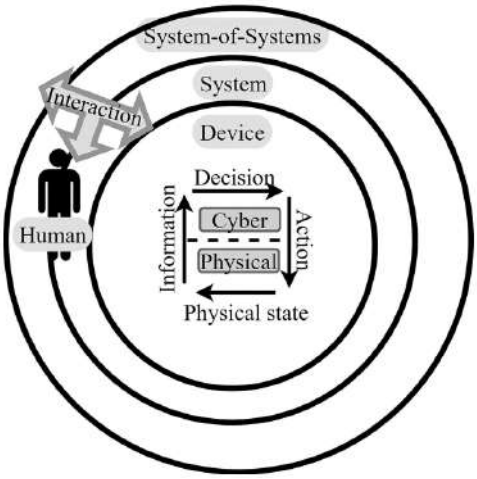


Рисунок 1.3 –Концептуальна модель кіберфізичної системи

На рис. 1.4 подана систематична модель CPS [4].

Четверта промислова революція або Індустрія 4.0 поєднала Інтернет з іншими машинами чи людьми. Таким чином, Індустрія 4.0 зосереджена на

розширенні ефекту цифрових технологій шляхом створення машин, які будуть більш самодостатніми, здатними спілкуватися одна з одною та оброблювати великі обсяги даних.

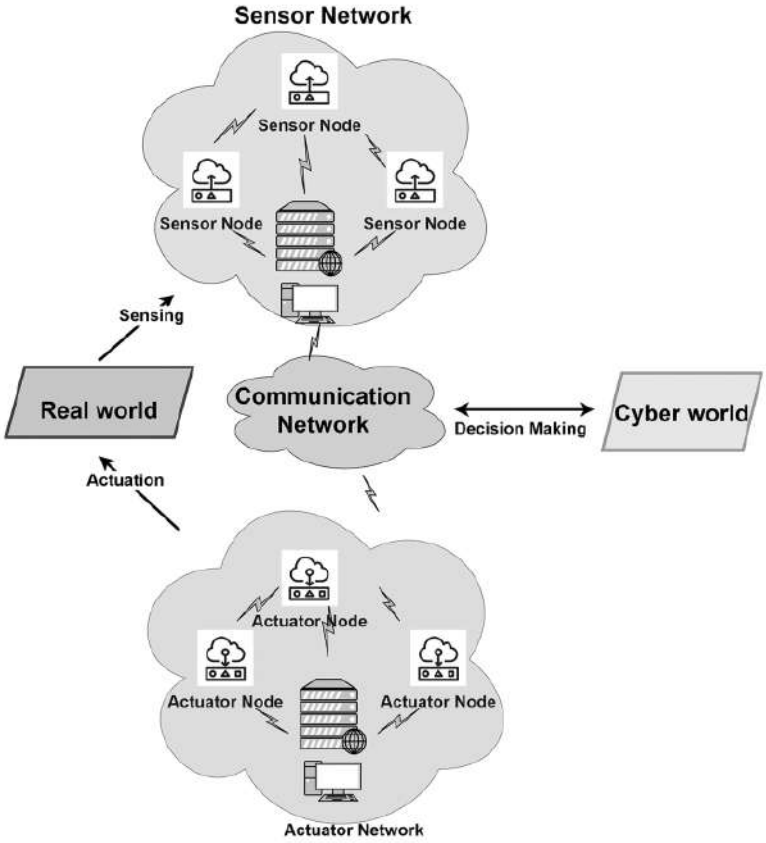


Рисунок 1.4 – Систематична модель CPS

IIoT стимулює Індустрію 4.0 і аналітику великих даних, промислові кіберфізичні системи (ICPS), штучний інтелект, периферійні хмарні обчислення, цифрові двійники та інші технології. IIoT є частиною Industrial 4.0, яка підкреслює поняття систематичної цифровізації та підключення всіх виробничих одиниць шляхом інтеграції звичайних галузевих можливостей з інтернет-технологіями.

Технології CPS інтегрують датчики, обчислення, керування та мережу у фізичні пристрої та інфраструктуру для підключення їх до Інтернету. Крім того, CPS та IoT включають логічні, фізичні та людські компоненти, що взаємодіють, шляхом інтеграції логіки та фізики. ICPS є об'єднанням IIoT і CPS.

Промислова еталонна архітектура Інтернету (IIRA, Industrial Internet Reference Architecture) і еталонна архітектурна модель Індустрія 4.0 (RAMI 4.0,

Reference Architectural Model Industry 4.0) є стандартизацією та еталонною архітектурою, задіяною в Індустрії 4.0 та Індустрії 5.0.

IIRA – це заснована на стандартах відкрита архітектура для систем IIoT. Вона забезпечує промислову застосовність, керуючи сумісністю, відображаючи застосовні технології, технологію маневрування та розробку стандартів для реального розгортання систем IIoT.

На рис. 1.5 подано функціональну точку зору архітектури IIRA, яка відображає функціональні компоненти та взаємозв'язок із взаємодією зовнішнього середовища [4].

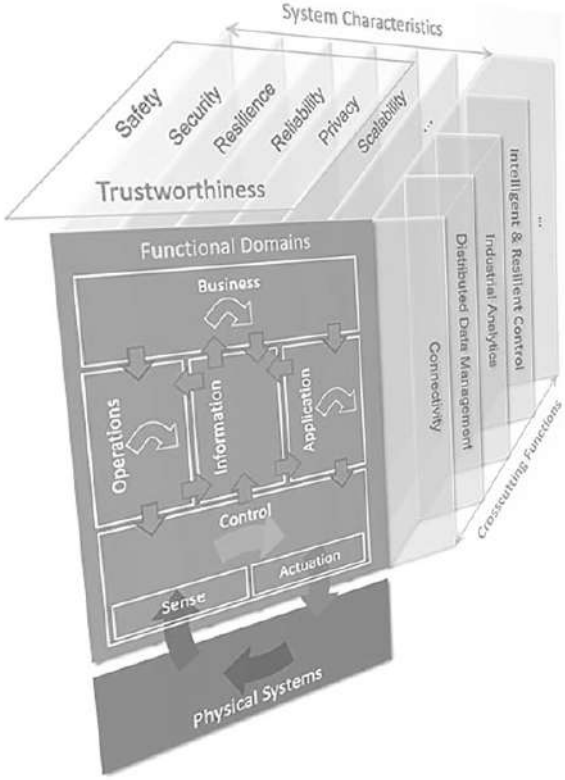


Рисунок 1.5 – Архітектури IIRA

Розглянемо класифікацію компонентів архітектури з точки зору IIRA.

Керування:

– промислові системи керування функціонують, як зчитування та запис даних датчиків (взаємодія датчиків, приводів, контролерів, шлюзів тощо);

– абстракція пристрою через представлення віртуальних об'єктів;

– інтерпретація даних, зібраних датчиками та іншим обладнанням;

– управління роботою систем керування, таких як конфігурація та оновлення мікропрограм, або програмного забезпечення;

– виконання логіки керування для розуміння станів, умов і поведінки системи.

Експлуатація:

– функції для прогнозування, управління, оптимізації та моніторингу систем у домені керування, наприклад, конфігурація, запис і відстеження ресурсів;

– передача команд керування;

– розпізнавання та виявлення проблем за допомогою моніторингу ресурсів у реальному часі;

– прогнозний аналіз системи IIoT на основі інформації про минулу роботу та продуктивність;

– зменшення споживання енергії для оптимізації системи.

Інформація:

– функції для збору даних домену з подальшою обробкою даних, моделюванням і аналізом для отримання високорівневої обізнаності про систему;

– складається з групи функцій, відповідальних за збір даних про роботу та статус датчиків у всіх доменах.

Програма:

– функції, які можуть виконувати логіку програми під час виконання спеціалізованих бізнес-функцій;

– цей домен застосовний для набору правил із необхідними певними функціями;

– набір функцій, що можуть надаватись іншим програмам для їх подальшого використання.

Бізнес:

– експлуатація систем IIoT від початку до кінця, поєднуючи її із конкретними бізнес-завданнями існуючих або нових типів систем.

На рис. 1.6 представлена модель еталонної архітектури, яка використовується для промислової цифровізації [4].

Ресурс відображає об’єкти у фізичному світі, як-то обладнання, документи, люди тощо.

Інтеграція – це перехід від фізичного світу до віртуального. Він відображає фізичні активи та їхні цифрові можливості, дозволяючи контролювати через комп’ютери та здатність генерувати ресурси для себе.

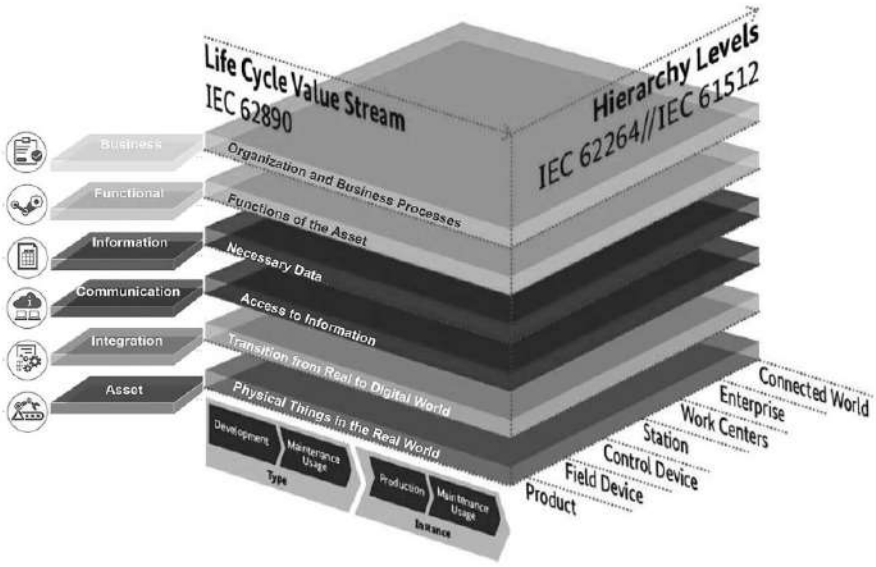


Рисунок 1.6 – Модель еталонної архітектури, яка використовується для промислової цифровізації

Зв'язок – стандартизований зв'язок між повідомленнями та подіями на інформаційному рівні або зв'язок між службами та інструкціями керування на інтеграційному рівні. Він зосереджений на методах передачі, виявлення мережі та підключення до мережі.

Інформація – технічні функції активу, що можуть пропонувати, використовувати, генерувати або змінювати послуги та дані, які можна надавати, використовувати, генерувати або змінювати.

Функціональність – в термінах Industry 4.0 та 5.0, опис логічних функцій активу, таких як його технологічна функціональність.

Бізнес – організація послуг для побудови бізнес-процесів і зв'язків між ними, а також для підтримки бізнес-моделей, дотримуючись правових і регуляторних обмежень.

Рис. 1.7 демонструє типову архітектуру 5C і відображення з IIRA та RAMI 4.0 [4]. Архітектура 5 C CPS складається з п'яти рівнів Connection, Conversion, Cyber, Cognition і Configuration. Він надає реальні вказівки для виробничих галузей щодо впровадження CPS, щоб покращити якість продукції та зробити свою систему надійною за рахунок інтелектуального та стійкого обладнання. Архітектура 5 C широко прийнята для розгортання CPS.

One Machine-to-Machine (oneM2M): глобальний стандарт, який використовується для опису вимог, архітектури, специфікацій, рішень безпеки та сумісності для машинно-машинних додатків і IoT-додатків. Він був

ініційований у 2012 році вісьмома відомими організаціями, які відповідають за розвиток мережі 6G для поєднання CPS та Industrial IoT.

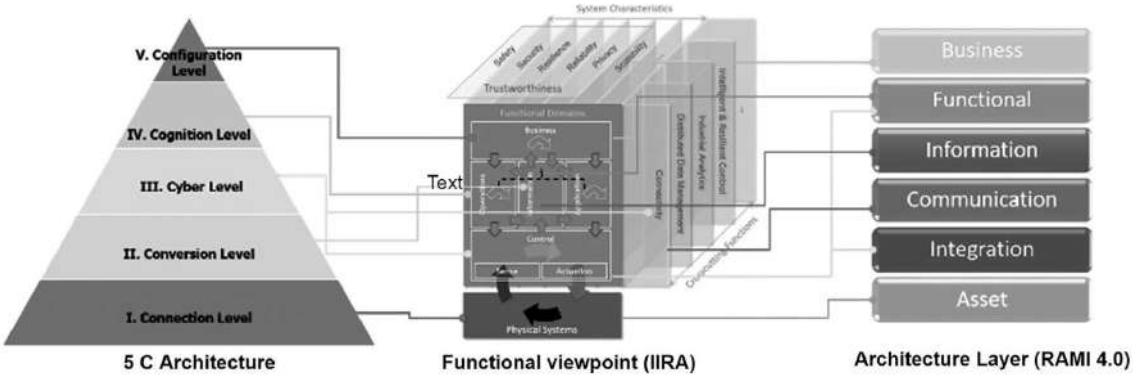


Рисунок 1.7 – Архітектура CPS і відображення архітектури 5C з різними стандартами

1.3 Індустрія та IoT

1.3.1 Індустрія

Промислова революція змінила світову економіку, змінивши примітивне сільське господарство на масове виробництво товарів. Промислова революція примусово замінила індивідуальну працю механізованими складальними лініями, що призвело до масового виробництва. Індустрія 3.0 пропонувала масове виробництво, а Індустрія 4.0 – масове налаштування.

Тим часом Industry 5.0 поверне робочу силу в промисловість за допомогою передових технологій для персоналізації та налаштування засобів виробництва, а також виробничих процесів. Виконаємо порівняння поточної Індустрії 4.0 і майбутньої Індустрії 5.0.

Індустрія 4.0: нещодавній прогрес у рамках четвертої промислової революції – це цифровізація промисловості, яка зосереджена на масовій кастомізації, що підтримується технологією IoT для створення цифрових двійників продуктів, що виробляються, для стимулювання промислового виробництва, логістики та багатьох інших галузей. Індустрія 4.0 вносить величезні досягнення в промисловий сектор, одним із найважливіших ефектів якого є конвергенція екосистем операційних технологій (OT) та інформаційних технологій (IT).

В Industry 4.0 IoT використовує тисячі датчиків і приводів для збору даних для контролю та моніторингу всіх виробничих інструментів, а інтелект машиного навчання (ML) використовується для виявлення та надання висновків про те, що відбувається на промисловому об’єкті, прогнозування результатів і, зрештою, реагування. щоб запобігти або пом’якшити проблеми. Іншими словами, створюється розумна фабрика, включаючи IoT і Cyber-Physical Systems, хмарні обчислення та когнітивні обчислення для забезпечення автоматизації та обміну цифровою інформацією.

Індустрія 5.0 ще не є революцією, але вона доповнює Індустрію 4.0. Індустрія 5.0 дає бачення виробництва поза межами ефективності та продуктивності, як ексклюзивних цілей. Це підкреслює відповідальність промисловості та її внесок у суспільство.

Концепція «Промисловість 5.0» походить від «Суспільства 5.0», де «Суспільство 5.0» намагається вирішити різноманітні проблеми, виходячи за межі оцифрування економіки та зосереджуючись на цифровізації на всіх рівнях суспільства, а також на цифровій трансформації самого суспільства.

У центрі уваги Індустрії 5.0 буде повторне впровадження людських рук і мозку в індустріалізацію. Іншими словами, Індустрія 5.0 знову об’єднує людей і машини для розробки нових технологій і співпраці для покращення виробничих ресурсів і ефективності, а також для забезпечення процвітання за межами зайнятості.

Індустрія 5.0 надає пріоритет добробуту людини в центрі виробничого процесу, поважаючи стійкість, суспільні зміни та екологічні обмеження. Дана концепція розглядає людське існування в індустрії з альянсом «людина-машина», рішення, орієнтовані на людину, різноманітність і стійку економіку, а також стійкість бізнесу та кібернетики.

Ідея Індустрії 5.0 базується на інтеграції інформаційно-комунікаційних технологій і складних промислових технологій у так звані кіберфізичні системи для створення цифрової, інтелектуальної та стійкої фабрики. Вона доповнює існуючу стратегію Індустрії 4.0, підкреслюючи роль досліджень та інновацій у переході до більш сталого, орієнтованого на людину та стійкого майбутнього.

Порівняння Індустрії 4.0 та Індустрії 5.0 подано на рис. 1.8 [4].

В Індустрії 5.0 розроблено новий тип спеціальних автономних роботів AI під назвою Collaborative Robots (Cobots, коботи), щоб прискорити виробничий процес і збільшити прибутковість. Люди та коботи працюють разом на

спільному робочому місці, де коботи відповідають за рутинну роботу, наприклад, видобуток даних. Навпаки, люди беруть на себе завдання високого рівня, які допоможуть вирішити кілька проблем. Роботи AI допомагають оптимізувати завдання та приймати рішення в режимі реального часу для покращення якості та промислового процесу.

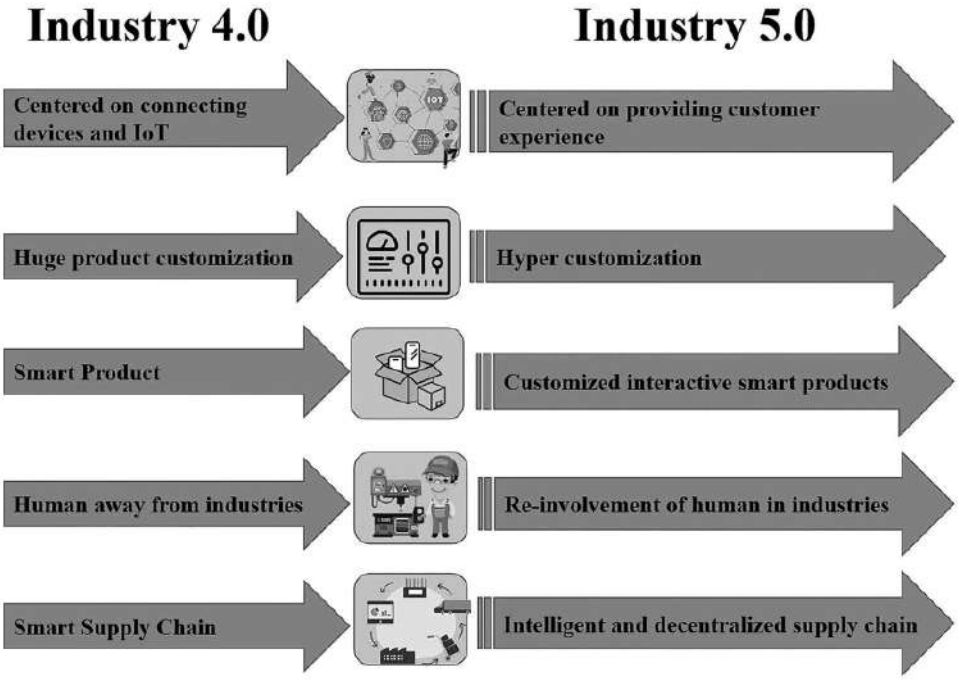


Рисунок 1.8 – Порівняння концепцій Індустрії 4.0 та Індустрії 5.0

Виходячи з рис. 1.8, можна виділити новітні функції, які є характерними для Індустрії 5.0:

– повторне введення людей у виробничий процес для персоналізації та творчого мислення разом із роботами AI;

– впровадження в промисловість колаборативних роботів;

– залучення працівників до автоматизації промисловості без їх заміни та незначної загрози безробіття;

– рівна можливість кар’єрного працевлаштування для інноваційних мислителів та спеціалістів з питань економіки;

– розвиток виробничої економіки на основі Індустрії 5.0 для задоволення потреб споживачів через гіперналаштування;

– коботів можна використовувати в небезпечних для життя ситуаціях, наприклад, отруйні гази, гострі інструменти, сценарії важкої промисловості тощо.

Компанії майбутнього повинні адаптуватися до Індустрії 5.0 на конкурентному ринку для сталого розвитку. Як наслідок, зміна діяльності кожної компанії та перетворення її на концепцію цифрової індустрії матиме вирішальне значення для підтримки конкурентоспроможності компаній.

1.3.2 Стільниковий IoT

Стільниковий Інтернет речей (C-IoT) і його застосування отримали широке визнання в усьому світі з розвитком стільникового зв'язку. Із запуском 5G і дослідженнями технології 6G прогнозується, що до 2030 року кількість стільникових з'єднань IoT досягне близько 3,5 мільярдів. C-IoT можна класифікувати як масовий IoT, широкосмуговий IoT, критичний IoT і промисловий IoT на основі їх функцій, характеристик, можливостей і варіантів використання. На рис. 1.9 зображено чотири групи IoT на основі різних атрибутів [4].

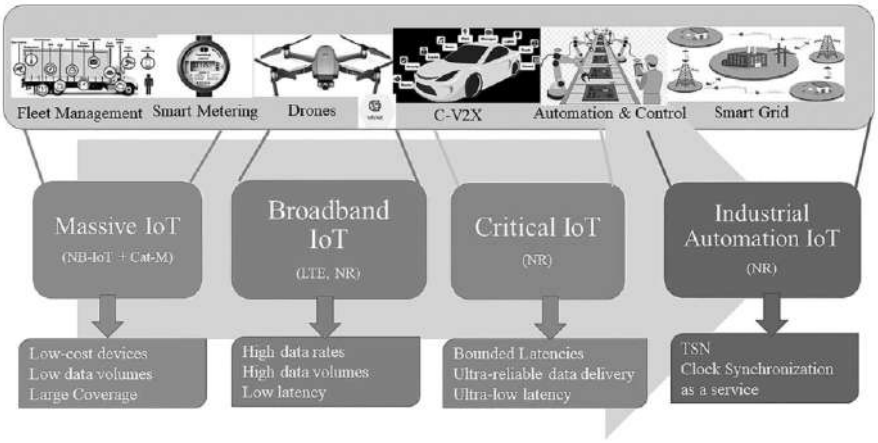


Рисунок 1.9 – Чотири категорії IoT на основі різних характеристик

Масовий IoT використовує NB-IoT для надання послуг стільникового зв'язку для пристроїв низької складності з великою кількістю об'ємів за меншу вартість пристрою. NB-IoT (Narrow Band Internet of Things) – стандарт стільникового зв'язку для пристроїв телеметрії з низькими обсягами обміну даними, який розроблено консорціумом 3GPP у рамках робіт над стандартами стільникових мереж нового покоління.

Такі пристрої IoT менш чутливі до затримок і вимагають менше енергії та непотрібних даних, однак вони вимагають широкого покриття, що потребує масштабного розгортання.

Широкосмуговий IoT використовується для різних випадків використання, які вимагають мобільного широкосмугового доступу, від базового до складного обсягу даних і швидкості передачі даних. Широкосмугові пристрої IoT вимагають величезних обсягів даних, швидкої пропускної здатності та низьких затримок, щоб подовжити термін служби батареї та покриття в порівнянні з Massive IoT. Крім того, ці критично важливі програми IoT вимагають наднизької затримки, високої доступності та наднадійної доставки даних із надзвичайно високою швидкістю передачі даних.

Категорії критичного Інтернету речей мають суворі вимоги до підключення для кількох передових глобальних і локальних додатків. Пристрої IoT, які використовуються в промисловій автоматизації, також відомі як Industrial IoT.

Промисловий IoT вимагає розширених функціональних можливостей стільникового IoT для чутливих до часу промислових випадків, таких як Industry 4.0 та Industry 5.0. Вони вимагають певного розташування в приміщенні, архітектурних деталей та елементів безпеки.

1.3.3 Промисловий IoT

IoT перетворив Індустрію 4.0 на розумну фабрику, зосереджену на масовій кастомізації та когнітивних обчисленнях. З іншого боку, IoT відіграє важливу роль у Індустрії 5.0 і повторному впровадженні людських рук і мозку в цикл для подальшого вдосконалення виробничих ресурсів. Індустрія 5.0 – це подвійна інтеграція людського та машинного інтелекту, яка перевіряє результати впровадження IoT в AI.

Індустрія 4.0 і Індустрія 5.0 в основному зосереджені на промислових секторах, тоді як IIoT охоплює все промислове та професійне обладнання, що використовується в галузі. У Індустрії 5.0 використовуються Cobots та AI, де взаємодія між людьми та машинами відбувається через мову, дотик або мозкові хвилі. Завдяки співпраці між людьми та інтелектуальними системами це створить синергію з об’єднанням людського когнітивного мислення та пристрою швидкої автоматизації. Таким чином, і роботи/машини, і люди співіснують у галузі, де роботи/машини зосереджені на повторюваних завданнях. Навички людей залучаються до творчого мислення для продуктів, налаштованих користувачами.

Для чутливих до часу промислових випадків, таких як Індустрія 4.0 і Індустрія 5.0, потрібні розширені функції стільникового Інтернету речей. Вони включають в себе характеристики мережі радіодоступу (RAN), які дозволяють створювати детерміновані мережі. У поєднанні з протоколами на основі Ethernet та іншими промисловими протоколами він забезпечить широкий спектр складних програм промислової автоматизації. Вони вимагають точного розміщення в приміщенні, спеціальної конструкції та заходів безпеки, наприклад, компонентів складальної лінії, розумної мережі, керування роботами та взаємозв'язків програмованого логічного контролера (PLC) як частину процесу автоматизації. На рис. 1.10 показано тенденції IoT та їх використання в Індустрії 5.0 [4].

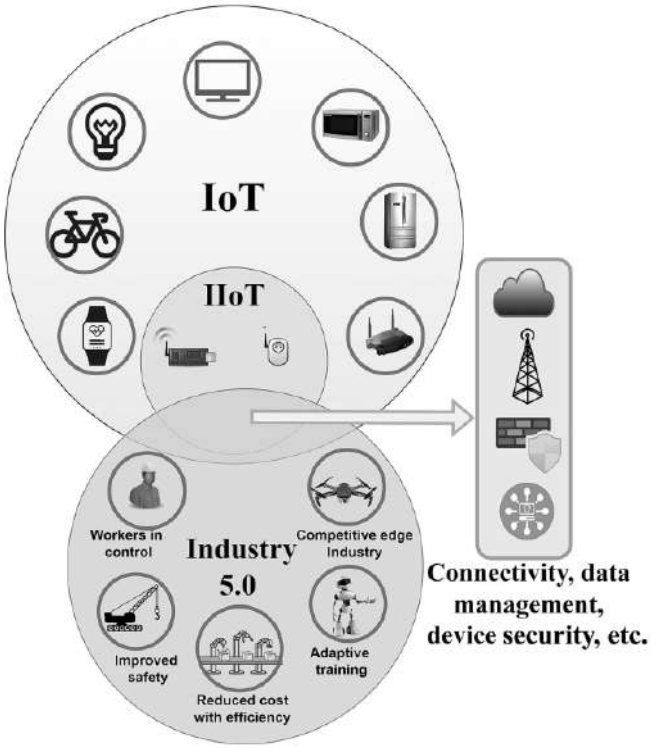


Рисунок 1.10 – Тенденції IoT та їх використання в Індустрії 5.0

IIoT, як підсистема IoT, відіграє життєво важливу роль в Індустрії 5.0. Різноманітні датчики та виконавчі механізми, які використовуються на розумній фабриці, наприклад IIoT, виявляють, прогнозують і виконують промислові операції на основі підходів машинного навчання. Цифровий обмін інформацією необхідний для команд, процесів і когнітивних обчислень між кінцевими механічними пристроями та системами керування. Типовий перетин між IIoT

та Індустрією 5.0 встановлено для підключення, керування даними, зв’язку, хмари та безпеки пристроїв, як показано на рис. 1.10.

Цей перетин діє як магістраль або нервова система для IIoT та Індустрії 5.0 у задачах керування та експлуатації. Він відіграє важливу роль у повсюдному підключенні та безпеці. Промислові комунікаційні мережі, які базуються на неліцензійних (наприклад, Wi-Fi) і ліцензійних (наприклад, стільникових) мережах, відповідають за такі показники продуктивності, як надійність, затримка, пропускна здатність і кібербезпека. Конфіденційність даних і кібербезпека є основними проблемами при впровадженні IIoT у галузі; таким чином, в архітектурі IIoT необхідна вбудована безпека E2E для потоку даних на основі і не на основі IP.

Крім того, такі параметри, як розташування обладнання, коефіцієнти розповсюдження та зміна навантаження трафіку з часом, мають значний вплив на відстеження та динамічну взаємодію між підсистемами CPS. Індустрія 5.0 базується на CPS і хмарних обчисленнях для покращення та автоматизації ланцюжків створення вартості в галузях. Хмарна робототехніка може забезпечити високу надійність і наднизьку затримку зв’язку в повсюдних програмах IoT.

Компоненти Індустрії 5.0 на рис. 1.10 забезпечують цифрову революцію в галузях завдяки технологіям IIoT, які відстежують і керують усім виробничим обладнанням, збираючи дані з тисяч датчиків для створення цифрового двійника для збільшення виробництва, логістики та багатьох інших промислових показників.

1.3.4 Впровадження Інтернету речей у виробництво за допомогою периферійних обчислень

На рис. 1.11 показано, як периферійні хмарні обчислення допомагають промисловості, завдяки IIoT, для високошвидкісного виробництва на основі зв’язку 6G [4]. IIoT дає змогу динамічно переналаштовувати виробництво і, отже, програмувати її для спільного використання, зниження витрат, прискорення виробництва та ранньої доступності на ринку. Останні розробки в інформаційних технологіях, хмарних обчисленнях, віртуалізації мережевих функцій (NFV) і 6G є незамінними для досягнення основних ключових показників ефективності (KPI) для швидкості, надійності, пропускної здатності та масштабованості. Хмарні обчислення забезпечують стійке до затримок

довгострокове зберігання додатків із високим обчислювальним ресурсом, необхідних для запобігання, обслуговування та підтримки бізнес-прийняття.

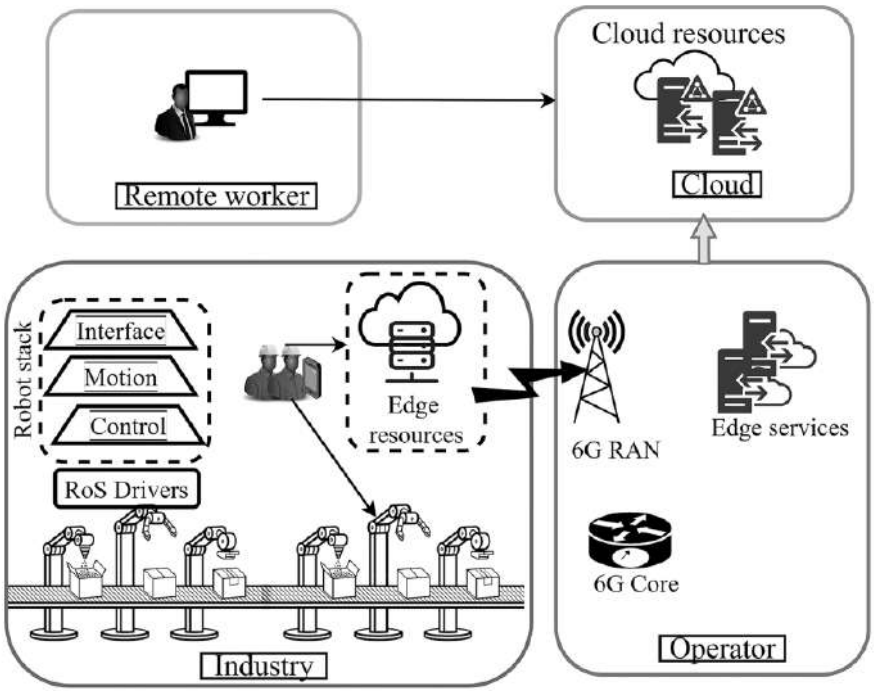


Рисунок 1.11 – Периферійні хмарні обчислення в промисловості завдяки IIoT і 6G

Основним використанням є зберігання даних і резервне копіювання системи, наприклад моніторинг, дані про продуктивність системи тощо; однак вони не підходять для мереж, стійких до затримок. Периферійні хмарні обчислення об’єднують периферійні пристрої IIoT і існуючі системи хмарних обчислень. Конфігурація edge-cloud забезпечує обчислювальні можливості та послуги поблизу заводських цехів (в межах одного переходу). Multi-Access Edge Computing (MEC) об’єднує стільниковий зв’язок, IT-хмару та виробничий сектор. Він може бути розгорнутий на околиці фабрики, забезпечуючи хмарні обчислення та можливості розвантаження даних у мережах радіодоступу, дозволяючи приймати локальні рішення та рішення в реальному часі.

Для впровадження IIoT на заводах потрібна чутлива до часу мережа (TSN), в той час, коли бездротова мережа з’єднує пристрої IIoT. Таким чином, забезпечується продуктивність і економічні переваги, розвантажуючи обчислювальні можливості та чутливі до часу програми, що становлять основу пристроїв IIoT завдяки концентрованій мережі. Це знижує вартість апаратних пристроїв, зменшує наскрізну затримку, збільшує економію пропускної

здатності в транспортній мережі та покращує загальну доступність послуг. Крім того, мікротранзакції між машинами можна зберігати в розподілених сховищах, таких як блокчейн, за допомогою смарт-контрактів для повної прозорості ланцюжка створення вартості між машинами та людьми, забезпечуючи питання безпеки IIoT.

Таким чином, впровадження Інтернету речей у промисловість за допомогою периферійних обчислень пропонує децентралізоване та регульоване середовище, яке може суттєво зменшити загальне споживання енергії та покращити зворотний зв'язок даних у реальному часі на виробництві.

1.4 Використання технології 6G для CPS та IIoT

1.4.1 Роль технології 6G в IIoT

Технологія 5G може підтримувати декілька послуг IoT, однак вона може не повністю підтримувати всі основні вимоги нових додатків IIoT у розвиненій інфраструктурі. Таким чином, технологія 6G розроблена для подолання обмежень технології 5G.

6G пропонує швидший і надійніший контроль і частоту зворотного зв'язку, ніж інші бездротові технології, такі як 5G і схильний до перешкод Wi-Fi. Деякі з особливостей 6G:

– надвисока швидкість з низькою затримкою зв'язку (HSLLC);

– повсюдний мобільний ультраширокосмуговий зв'язок (uMUB);

– надвисока щільність даних (uHDD);

– безперебійне підключення;

– надвисока швидкість передачі даних;

– можливість використання інтелектуальних датчиків;

– бездротова взаємодія мозку з комп'ютером (WBCI);

– точне позиціонування та розширене підключення до мережі.

Найважливішою інновацією, яка стане провідним фактором для 6G, є тривимірна мережа (тобто наземна, повітряна та космічна мережа), особливо включення неземного зв'язку в діапазоні терагерців (ТГц, THz), який підтримує безпілотний літальний апарат (БПЛА), супутникове з'єднання, інтелект, що пов'язаний із машинним навчанням і оптичний бездротовий зв'язок (OWC). Ці функції значно допоможуть у впровадженні промислового IoT.

6G діє як магістраль зв’язку та відіграє важливу роль в технологіях управління та експлуатації в галузі IIoT. На рис. 1.12 показані загальні тенденції 6G і сприятливі технології в індустрії [4].

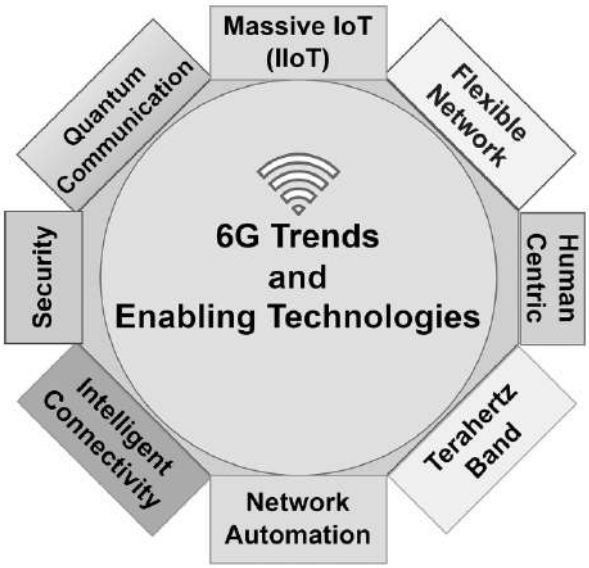


Рисунок 1.12 – Загальні тенденції 6G і сприятливі технології в індустрії

Серед поширених тенденцій і сприятливих технологій 6G – масовий IIoT, гнучкість мережі, орієнтованість на людину, квантовий зв’язок, кібербезпека, автоматизація мережі, терагерцевий зв’язок, інтелектуальне підключення тощо.

У промисловості 6G, як базова мережа для IIoT, забезпечує мобільний терадіапазон надійного зв’язку з низькою затримкою (MTRLLC) і велику пропускну здатність мережі. Це збільшує швидкість завантаження даних за допомогою периферійної хмарної обчислювальної платформи та дозволяє використовувати дані в реальному часі в інтелектуальних промислових процесах, щоб віддалений обмін даними був доступним.

Індустрія 5.0 інтегрує системи IIoT і CPS шляхом інтеграції інформаційних технологій, операційних технологій, людей і нових комунікацій 6G. Система IIoT із підтримкою 6G може збирати й аналізувати масивну інформацію в режимі реального часу з мільярдів підключених пристроїв, датчиків, БПЛА та машин, з’єднаних між собою через мережу 6G, а потім адаптувати її до всіх промислових застосувань.

IIoT з підтримкою 6G прискорить впровадження концепції Індустрії 5.0, яка забезпечує наступні переваги для створення істотної цінності:

– забезпечує в 100 разів вищий рівень бездротового з'єднання з підвищенням продуктивності в кілька разів, що сприяє підвищенню продуктивності з контролем якості;

– поєднує інтелект із процесом оптимізації та використання можливостей ML і THz для інтелектуальної автоматизації;

– покращує функції 6G для доступу до мережі та швидшого управління величезним обсягом трафіку, прискорення онлайн-покупок, розміщення замовлень, доставки клієнтам і віддалених робочих середовищ, що забезпечує стійкі галузі;

– надає галузям, які керуються AI, взаємопов'язані граничні хмарні обчислення 6G, які збирають величезні обсяги даних для виявлення інформації, яку інакше було б неможливо отримати;

– забезпечує кіберзахищеність мережі передачі даних за допомогою 6G, що надає переваги технологіям із підтримкою IIoT.

З'єднання 6G для CPS і IIoT забезпечує кілька переваг, таких як максимальне зростання доходу, зниження операційних витрат і підвищення ефективності виробництва в Індустрії 5.0.

1.4.2 Технологія 6G для CPS та IIoT

«Орієнтація на людину» та «підключення з неба» є основними характеристиками зв'язку 6G. 6G долає цифровий розрив і звужує відмінності між фізичним, кіберсвітом і світом людей. 6G стосується взаємодії між трьома світами:

– людський світ, який включає людські органи чуття, тіла, інтелект і цінності;

– цифровий світ, який включає інформацію, зв'язок і обчислення;

– фізичний світ, який включає речі та процеси.

Взаємодія між цими світами стає в центрі уваги інновацій:

– через підключений інтелект, який дозволяє пристроям підключатися без обмежень;

– імерсивний зв'язок, який усуває відстань як бар'єр для людського досвіду;

– поєднання фізичних датчиків/приводів;

– програмовані цифрові представлення.

Щоб поєднати три світи, CPS складається з чотирьох стовпів: люди, процеси, дані та речі розумно пов’язані один з одним у світі CPS, щоб розширити можливості користувача, як показано на рис. 1.13 [4].

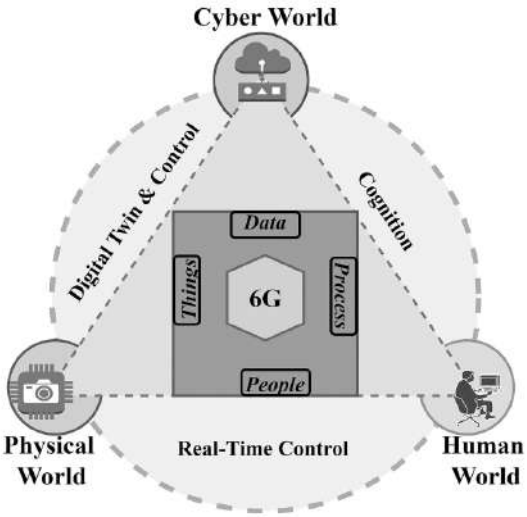


Рисунок 1.13 – Інтеграція кібернетичного, фізичного та людського світу

Технологія 6G безпосередньо сприяє впровадженню CPS. CPS у виробництві має інтегрувати віртуалізацію, обчислення та бездротові технології, щоб відповідати найсуворішим критеріям додатків реального часу.

В разі поєднання IIoT та CPS на основі 6G, ця система має вирішити деякі з наведених нижче проблеми.

Цифрова інтеграція: оскільки світ стає цифровим і CPS інтегрується з IIoT, технологія 6G повинна принести величезні цифрові переваги, виступаючи в якості магістралі та інтегруючи всі три світи.

Надійність: кожен пристрій буде підключено до кожного через 6G і вбудовану технологію навчання AI, де машини приймають більшість рішень. Для запобігання серйозних впливів на людей і промисловість, мережі 6G повинні підтримувати конфіденційність і цілісність комунікацій E2E, також повинні мати здатність до виявлення вторгнень, захисту даних, надійності і кібербезпеці.

Інтелектуальне підключення: усі IIoT, інтелектуальні пристрої та обладнання, що використовуються в галузі, можуть використовувати технологію AI і працювати автономно. Система повинна відігравати важливу роль і нести певну відповідальність за масштабне розгортання розвідувальних даних у суспільстві.

Екологічний розвиток: вкрай важливо зменшити цифрові сліди, створені завдяки технологіям і промисловості. Технологія 6G повинна розвивати енергоефективну цифрову інфраструктуру, сприяти реалізації Цілей сталого розвитку ООН і допомагати впровадженні та виконанні Зеленої угоди ЄС.

Таким чином, Інтернет речей відіграє одну з важливих ролей у промисловості, яка стимулює масове виробництво. Промисловий IoT допомагає в автоматизації та швидкому виробництві товарів на основі методів ML.

Перша промислова революція почалася з винаходом парового двигуна в кінці XVIII століття і переходу від ручної праці до механічного виробництва (рис. 1.14) [16].

Рисунок 1.14 – Стадії розвитку промисловості

Друга промислова революція настала приблизно через 100 років із застосуванням конвеєрного виробництва з електроприводом. З першої третини XX століття воно зробило можливим економічно ефективне серійне виробництво.

Електронні системи управління, інформаційні технології, електроніка, роботи і розширення використання датчиків роблять можливим подальшу автоматизацію виробничих, складальних і логістичних процесів та перехід до третьої промислової революції.

Важливо розрізняти терміни «Четверта промислова революція» і «Індустрія 4.0». Перший визначає впровадження нових технологій 4.0 і їх вплив на всю економіку і соціальну сферу – розумні міста, будинки, сільське

господарство, енергетика, інфраструктурні об'єкти, фінанси, державне управління, охорона здоров'я, освіта тощо. Індустрія 4.0 належить, насамперед, до сфери виробництва матеріальних продуктів (рис. 1.15).

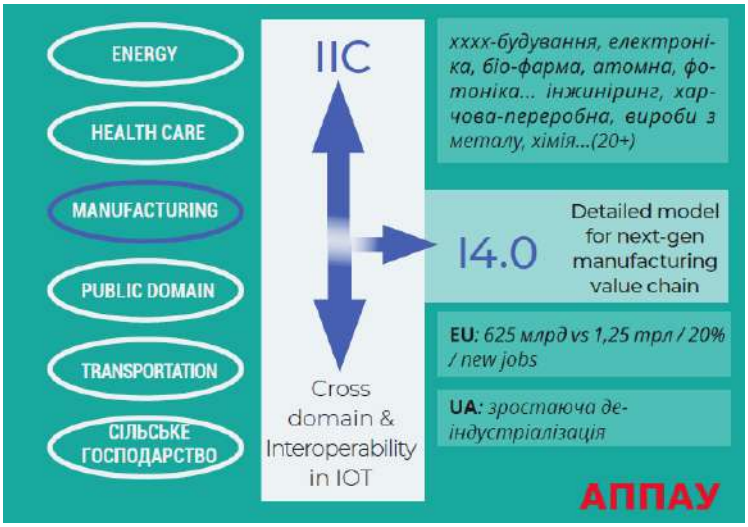


Рисунок 1.15 – Різниця між термінами «Четверта промислова революція» і «Індустрія 4.0»

Водночас, невірно ізолювати Індустрію 4.0 від інших сфер економіки. Delloite вказує, що Розумні Фабрики є дотичними до багатьох сфер, пов'язаних з промисловими виробництвами, і утворюють цілісну технологічну екосистему (рис. 1.16).

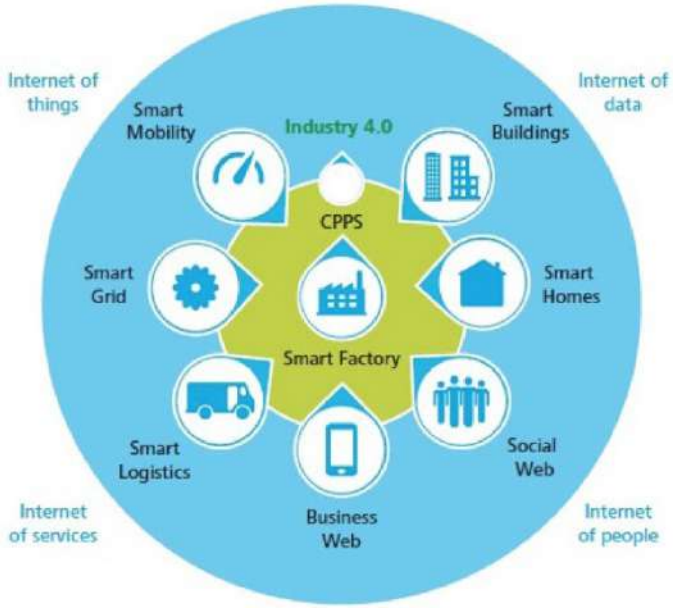


Рисунок 1.16 – Цілісна технологічна екосистема

Більшість експертів в області світової Індустрії 4.0 єдині в розумінні трьох загальних характеристик (рис. 1.17). Цей фреймворк також показує, що 4.0 певною мірою є еволюцією (продовженням 3.0).

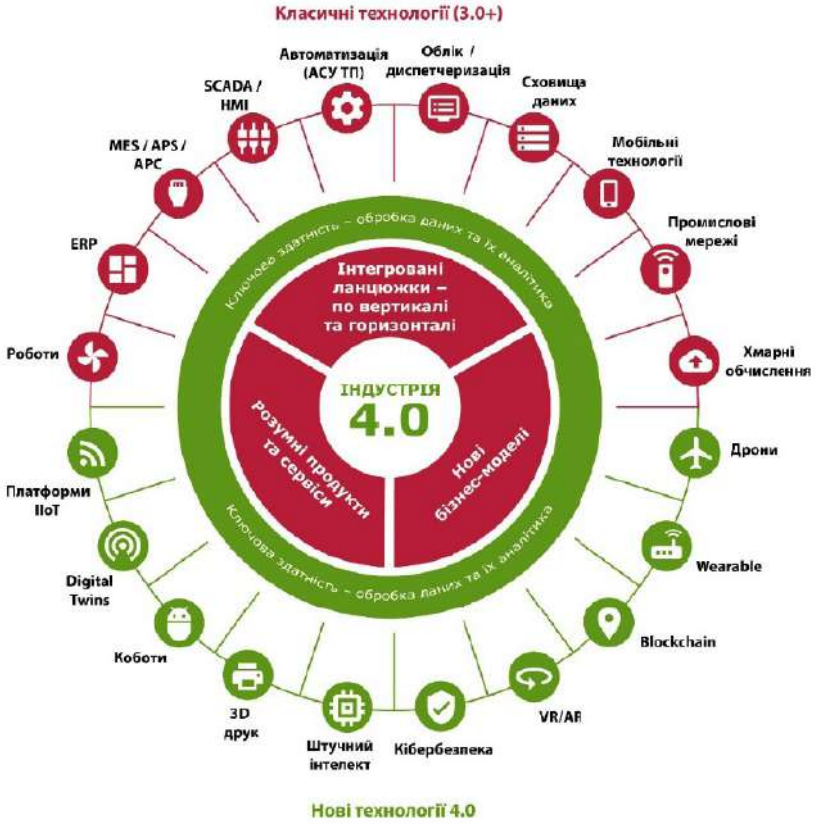


Рисунок 1.17 – Головні характеристики і технології 4.0

Інші важливі характеристики, які не наведені на рис. 1.17 і які в результаті роблять фабрики і заводи «розумними», це:

– інтероперабельність: кіберфізичні системи дозволяють людям і розумному обладнанню (фабрикам) ефективніше поєднуватися один з одним;

– віртуалізація: в 4.0 можна створювати віртуальні копії розумних фізичних об’єктів (масштабованих від окремих пристроїв або машин до цілих заводів) і, відповідно, запускати різні механізми симуляцій, моделювання, а також оцінки реального стану;

– децентралізація: на відміну від високоцентралізованих підходів, у 3.0 і 4.0 кожна кіберфізична підсистема може робити власні рішення і взаємодіяти з іншими найбільш оптимальним способом;

– реальний час: всі дані та їх аналітику можливо отримувати в реальному часі;

– орієнтація на сервіси: кількість різних сервісів, як із взаємодії пристроїв і систем між собою, так і з взаємодії з людьми та учасниками екосистеми, зростає в рази;

– модульність: гнучка адаптація розумних фабрик до зовнішніх змін так само зростає, оскільки можна легко змінювати або розширювати окремі модулі систем керування.

У той час, коли зазначені характеристики є абсолютно новими, зв'язок 3.0 з 4.0, поданий на рис. 1.17, є вкрай важливим для цілого ряду індустрій. Він стверджує, що не можна перестрибувати через рівень 3.0, якого до цих пір на 100% немає в більшості промислових галузей України.

Велика частина впровадження технології 4.0 – особливо це стосується великих даних і штучного інтелекту – базується на тому, що ці дані вже оцифровані на польовому рівні. Тобто на підприємствах вже налагоджений облік і встановлені датчики.

1.5 Комунікаційні технології та цифровізація на інтелектуальному заводі

Комунікація на виробництві в рамках технології 4.0 займає передове місце. В інтелектуальному виробництві обладнання, контролери і датчики взаємодіють як один з одним, так і безпосередньо з Ethernet або у хмарі. Закрита система стає відкритою. Але змінюється не тільки кількість інформації, що обробляється безпосередньо на місці. Підвищується і якість до абсолютно нового рівня.

Інформація про стан виробничого обладнання та пов'язане з ним прогнозування про можливі простої виробництва за допомогою інноваційних систем зворотного зв'язку подає тільки один із прикладів. Це стало можливим завдяки швидкому збільшенню обчислювальної потужності, яку також можна вже використовувати децентралізовано в периферії, на краю мережі або в основах виробництва. Результатом цього стає більш гнучке й динамічне виробництво, яке в будь-який час може індивідуально і швидко реагувати на вимоги клієнтів.

Поширена досі форма комунікації між датчиками і блоками управління, і розташованими вище рівнями управління технологічними процесами, виробництвом і підприємством, є закритою системою (рис. 1.18).

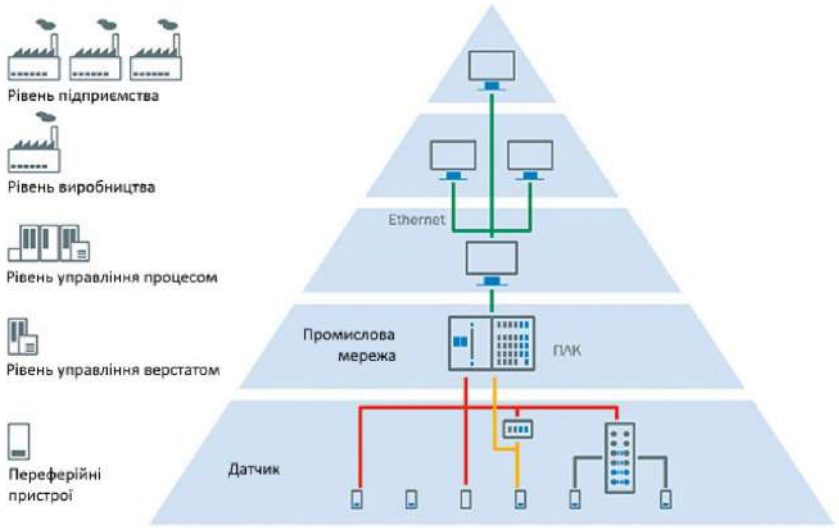


Рисунок 1.18 – Рівні комунікації на інтелектуальному заводі сьогодні

При цьому дані передаються від периферійних пристроїв, тобто датчиків і виконавчих механізмів, у програмований логічний контролер (ПЛК).

Децентралізована обчислювальна потужність в ідеології Індустрії 4.0 переробляє дані в інформацію безпосередньо в датчику. Рішення приймаються децентралізовано. Релевантна для технологічного процесу, виробництва і підприємства інформація направляється безпосередньо в Ethernet і хмарний сервіс. На рис. 1.19 показана схема децентралізованого управління виробництвом.

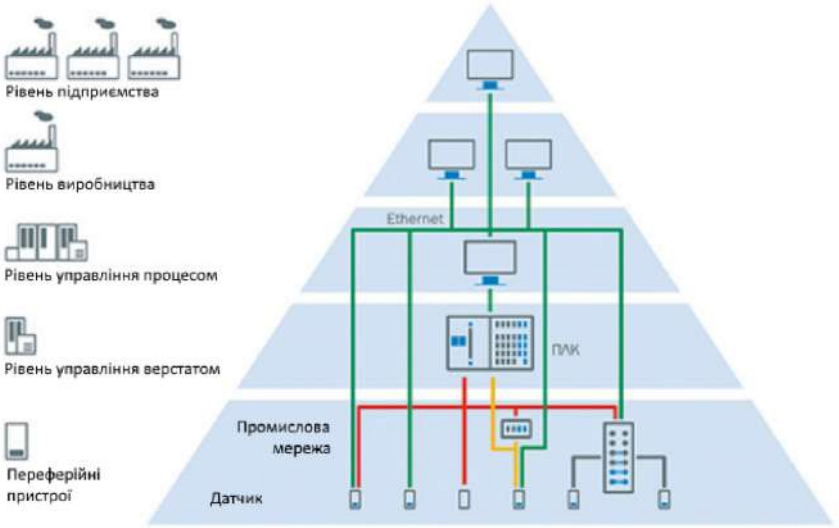


Рисунок 1.19 – Схема децентралізованого управління виробництвом

Подальший розвиток цифрового виробництва передбачає перенесення обчислень у хмарний сервіс, як показано на рис. 1.20.

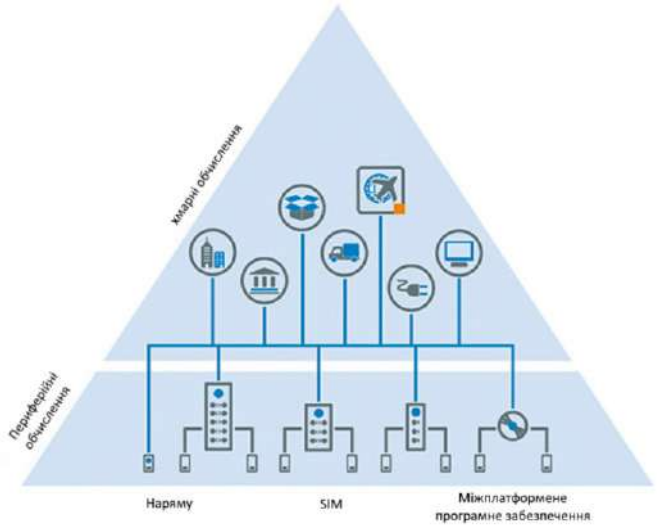


Рисунок 1.20 – Мережна інформація на цифровому виробництві

Таким чином, хмарний сервіс стає все більш важливим для управління загальними процесами. Але основна обчислювальна потужність все більше переміщується на периферію. Датчики готують зібрані дані для передачі інформації, яка потім обробляється в Ethernet або у хмарному сервісі для подальшого процесу.

1.6 Виробничі процеси та контроль стану технологічного процесу

1.6.1 Прозорі виробничі та логістичні процеси

Позитивні ефекти об’єднання в мережу в рамках Індустрії 4.0 описуються як послідовне прозоре виробництво на всьому виробничому процесі. При успішному об’єднанні в мережу всіх компонентів такий вид прозорого виробництва забезпечує огляд усіх виробничих і логістичних процесів за усім ланцюжком поставок, аж до оформлення замовлень і доставки продукції замовникам (рис. 1.21). Це зменшує споживання матеріалів і ресурсів. Додатково цілісно оптимізуються виробничі і постачальні мережі.

Інтелектуальні рішення з відстеження та контролю за проходженням генерують дані та інформацію, які в мережному технологічному ланцюжку забезпечують повне виявлення, ідентифікацію та відстеження продукту і матеріалу.

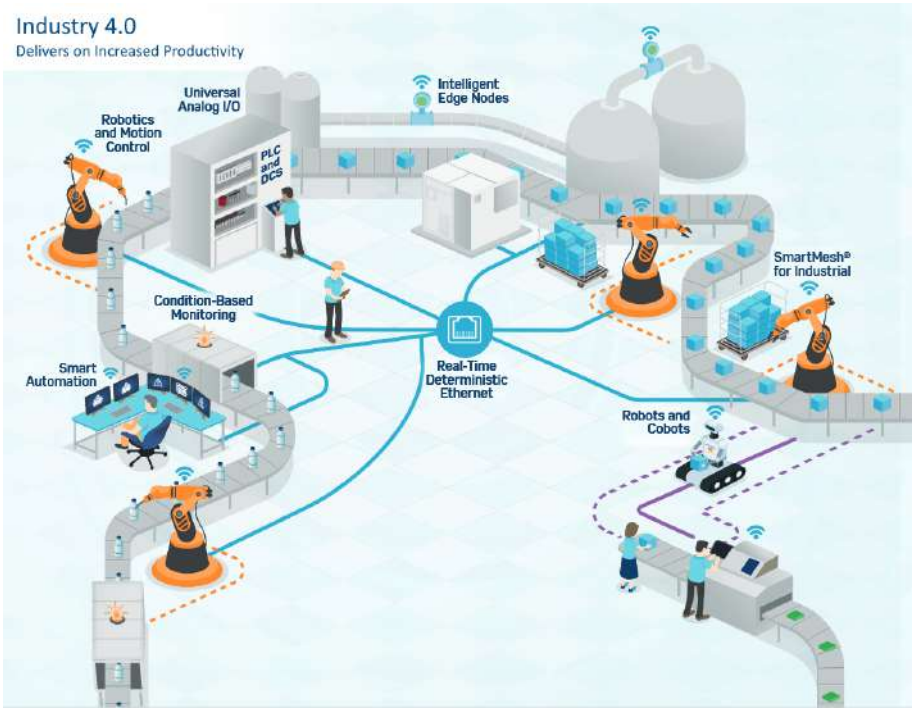


Рисунок 1.21 – Прозорі виробничі та логістичні процеси

Технічні можливості щодо реалізації рішень з відстеження та контролю за проходженням різноманітні. Вибір відповідної технології ідентифікації для найкращої ефективності зчитування і системної інтеграції варіюється залежно від вимог.

Для використання в Індустрії 4.0 рішень на розумному заводі використовуються, перш за все, RFID і програмовані камери.

Датчики по всьому виробничому ланцюжку на основі носіїв даних безпосередньо розпізнають, які кроки зі збірки необхідно ініціювати, і забезпечують постійну прозорість аж до відвантаження. Сьогодні інтелектуальні датчики означають не тільки збір даних про реальність, а й відповідну підготовку інформації вже в датчиках.

На цьому фоні кожна технологія матиме своє повноваження в майбутньому: RFID, наприклад, дозволяє зчитування і запис а, отже, багаторазове використання носіїв даних, крім того, немає необхідності в прямому «візуальному контакті».

З іншого боку, зчитувачі коду на основі камери також зчитують 2D-коди і звичайний текст. Збережені зображення можуть бути заархівовані та проаналізовані.

1.6.2 Динамічне і гнучке виробництво

Прогресивна автоматизація сприяє гнучкому виробництву і дуже невеликим обсягам партій. Побажання клієнтів стають головними. Розмір партії в одну штуку більше не є дорогою складністю, а повсякденним успішним бізнесом.

Невеликі обсяги партій та індивідуалізовані продукти масового виробництва є девізами Індустрії 4.0. Для її виправдання машина або устаткування повинні вміти обходитися зі змінною подачею продукту і бути адаптованими до різних форматів. Тільки тоді можна гнучко й ефективно виробляти товари відповідно до побажань замовника, навіть розміром партії в одну штуку, і в адаптації до коливань попиту.

Інтелектуальні датчики забезпечують нову якість гнучкості. Вони створюють дані з виробництва в режимі реального часу. Датчики підтримують і спрощують обробку даних за рахунок того, що результати вимірювань обробляються за допомогою інтелектуальних функцій відразу на місці і передаються у вигляді підготовленої інформації, що містить тільки корисні дані.

Зі збільшенням міри автоматизації устаткування також ростуть і завдання окремих компонентів: у різних галузях вже використовуються, наприклад, фотоелектричні датчики з гнучкими настройками і діагностичними функціями.

Наприклад, індуктивні датчики наближення з IO-Link вирішують складні завдання безпосередньо в самому датчику. Датчики контрасту, датчики рівня та електронні реле тиску обмінюються даними налаштувань параметрів за допомогою інтегрованих інтерфейсів IO-Link.

Вимірювальні високоавтоматизовані світлові завіси знижують витрати на проводку у виробничих умовах і забезпечують доступ до діагностики та переналаштування формату. Енкодери з EtherNet/IP™ мають як активний веб-сервер, так і функціональні блоки для інтеграції в польову шину. Компактні датчики 2D-LiDAR (також лазерні 2D-сканери) надійно виявляють об’єкти під час контролю території.

Стандарт зв’язку з датчиками IO-link дає можливість легкої заміни кінцевих пристроїв. Ведучій пристрій завантажує параметри у встановлений як заміна пристрій, що значно спрощує технічне обслуговування – не потрібне додаткове налаштування. Серед доступних різновидів датчиків IO-Link: виконавчі датчики, вимірювальні датчики, фотоелектричні датчики,

RFID-датчики, лазерні датчики відстані, датчики кольору. Стандарт IO-Link регулюється відкритим консорціумом виробників.

Комунікації IO-Link на фізичному рівні реалізуються за допомогою звичайного провідникового кабелю і стандартних конекторів, таких як M12, M8 і M5. Система IO-Link складається з пристроїв IO-Link, включаючи датчики і приводи, а також провідного пристрою. Оскільки IO-Link має архітектуру типу «точка-точка», тільки один пристрій може бути під'єднаний до кожного порту ведучого пристрою.

Ведучі пристрої IO-Link можуть бути різними, включаючи карту в стійці ПЛК, або окремий пристрій з інтерфейсами до стандартних польових шин, таких, як PROFIBUS або PROFINET.

Протокол IO-Link підтримує різні типи даних – циклічні (процесні), ациклічні, сервісні, події. Пристрій IO-Link посилає дані тільки після запита від ведучого пристрою IO-Link. Ациклічні дані і події безпосередньо запитуються ведучим пристроєм, а циклічні дані відправляються після телеграми IDLE від ведучого пристрою.

Об'єднання в мережу всіх задіяних пристроїв і безпечний, децентралізований обмін даними призводить до широкого спектру варіантів застосування. Їх можна запропонувати як через хмару, так і через програмовані логічні контролери на рівні машин і систем.

Покращені обчислювальні можливості також змінюють візуальні здібності рішень на основі камери для забезпечення якості та управління виробництвом на основі датчиків.

Забезпечення якості є необхідною умовою для стійкого бізнесу і стабільного доходу. Воно включає в себе як управління матеріалами, так і функціональний контроль, контроль за машиною і виробництвом. Це дозволяє знизити рівень складських запасів і скоротити час обробки.

Сенсорні рішення для контролю за технологічним процесом і забезпечення якості підвищують гнучкість завдяки автономному коригуванню зі зміною якості і зміною продукту. Вони забезпечують ресурсоефективність, скорочення виробничого браку і високу пропускну здатність.

1.6.3 Взаємодія людини і робота на мережевому заводі

У розумному підприємстві (Smart Factory) люди і роботи зближуються ще дужче. У розумінні сучасного розподілу праці датчики підтримують роботів у їх

роботі – даючи їм «очі» для виконання завдань у промислових умовах. Більш сильна взаємодія між людиною і машиною вимагає абсолютно надійних і гнучких рішень для забезпечення безпеки. Кооперація і співіснування мають стати справжньою співпрацею.

Розумне підприємство робить ставку на тісну взаємодію роботів з людьми. У цих взаємодіючих сценаріях сила, швидкість, траєкторії руху робота і заготовки становлять небезпеку для робітника. Ці небезпеки мають бути обмежені застосуванням спеціальних запобіжних заходів. Уже сьогодні датчики безпеки дозволяють реалізувати тонку адаптацію під поточний автоматизований процес. Інтелектуальні алгоритми дозволяють, наприклад, перехід від техніки безпеки з цифровою технологією перемикання до безперервної машинної реакції залежно від руху людини.

Отже, наближення робочого більше не призводить до загального відключення машини, а скоріше до розумного зниження робочої швидкості або коригування напрямків руху.

«Safe Motion» нині є найкращою технологією. Вона в будь-який час забезпечує безпеку людей і, незважаючи на це, немає необхідності переривати виробництво. Внаслідок цього значно скорочується час простою і кількість помилкових відключень, часи циклів стають коротшими, а ефективність та експлуатаційна готовність машин і систем збільшуються.

Якщо людина і машина співпрацюють щільно й надійно, функціональна безпека в сучасних виробничих системах є важливим кроком до більшої гнучкості.

На шляху до повноцінного співробітництва – люди і роботи ділять між собою один і той самий робочий простір і працюють у ньому водночас – існують також рішення для співіснування або кооперації.

За одночасний захист великої кількості небезпечних об’єктів відповідає, наприклад, програмований контролер безпеки з супровідним програмним забезпеченням, також у поєднанні з безпечним каскадом датчиків.

Небезпечні зони, входи і небезпечні об’єкти абсолютно надійно захищає нове покоління лазерних сканерів безпеки. Високопродуктивні світлові завіси безпеки підходять як компактні альтернативи вибіркового відключення без додаткових датчиків, а також для забезпечення високодоступного захисту небезпечних об’єктів і зон.

1.6.4 Інтелектуальні датчики як невід'ємна складова частина Індустрії 4.0

Цифровізація та об'єднання обладнання в мережу – головна риса четвертої промислової революції. Нові технології роблять можливим злиття фізичного і віртуального світу в області виробництва і логістики в кіберфізичні системи. З 2011 року ця розробка об'єднується під терміном «Індустрія 4.0». Машини можуть автономно обмінюватися одна з одною даними і, таким чином, оптимізувати перебіг процесів. При цьому Індустрія 4.0 чітко посилається на створення мереж у промисловому секторі.

Сенсорна техніка створює умови для прозорих процесів в Індустрії 4.0. При цьому датчик утворює основу всіх наступних застосувань.

На відміну від класичних, немережевих датчиків, датчики Індустрії 4.0 надають більше, ніж просто дані вимірювань. Інтегрована, децентралізована обчислювальна потужність і гнучка програмованість є ключовими характеристиками, які роблять виробництво більш гнучким, динамічним і ефективним.

Технологічний прогрес завжди є передумовою для змін у промисловості. Основою для динамічних, оптимізованих у реальному часі і самоорганізованих промислових процесів є отримання інформації та її подальша обробка. У них датчики виконують роль постачальників даних для інтелектуального виробництва. Сенсорна техніка є обов'язковою умовою для успішної реалізації Індустрії 4.0 (рис. 1.22).

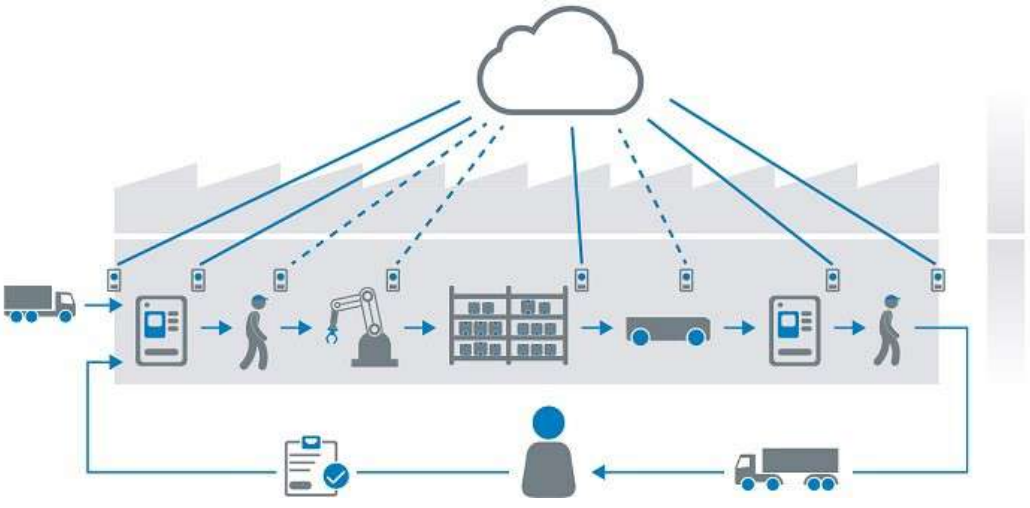


Рисунок 1.22 – Інтелектуальні датчики на мережевому заводі

Мережевий завод є обов'язковою умовою для Індустрії 4.0. Кожен сенсор, кожна машина і всі залучені люди в будь-який час можуть спілкуватися між собою. Але цей обмін інформацією не закінчується на самому заводі. Взаємодія між периферією і хмарним сервісом дозволяє управляти виробництвом і даними навіть ззовні.

Таким чином, це інтенсивне співробітництво між технологією і людиною робить процес більш прозорим, продуктивним і прибутковим.

1.7 Компоненти архітектури автоматизованої навчальної системи

1.7.1 Загальна архітектура системи

Розглянемо архітектуру автоматизованої системи, що використовується для навчання основам технології промислового інтернету речей (рис. 1.23).

Автоматизована навчальна система складається з чотирьох компонентів:

– хмарний сервер зі встановленим брокером Mosquito для організації взаємодії між компонентами за протоколом MQTT;

– засоби автоматизації дослідної лабораторії для безпосереднього доступу до обладнання: датчиків, виконавчих пристроїв, засобів робототехніки;

– віртуальна лабораторія з набором цифрових двійників реальних пристроїв та навчальних макетів;

– засоби диспетчеризації та віддаленого управління технічними засобами автоматизації на основі стаціонарних та мобільних технологій.

Хмарний сервер може бути розгорнуто на одному з доступних ресурсів, наприклад AWS, Hostico, Digital Ocean та іншому. Основна задача хмарного сервера – реалізація віддаленого доступу до ресурсів лабораторій в реальному часі для різних користувачів. В якості операційної системи на віддаленому сервері встановлюється ОС Linux.

Основу хмарного сервера становить брокер MQTT Mosquitto.

Mosquitto – це відкритий, безкоштовний і легкий брокер MQTT (Message Queuing Telemetry Transport), який дозволяє пристроям і програмам обмінюватися даними в режимі реального часу. MQTT – це протокол мережевого рівня, який дозволяє підключати пристрої до мережі Інтернет речей та обмінюватися повідомленнями між ними.

Даний сервіс розробляється за допомогою мови програмування C і підтримується на різних платформах, таких як Windows, Linux та macOS.

Mosquitto є легким та швидким брокером MQTT, що дозволяє обробляти великий потік повідомлень з мінімальними затримками та ресурсами.

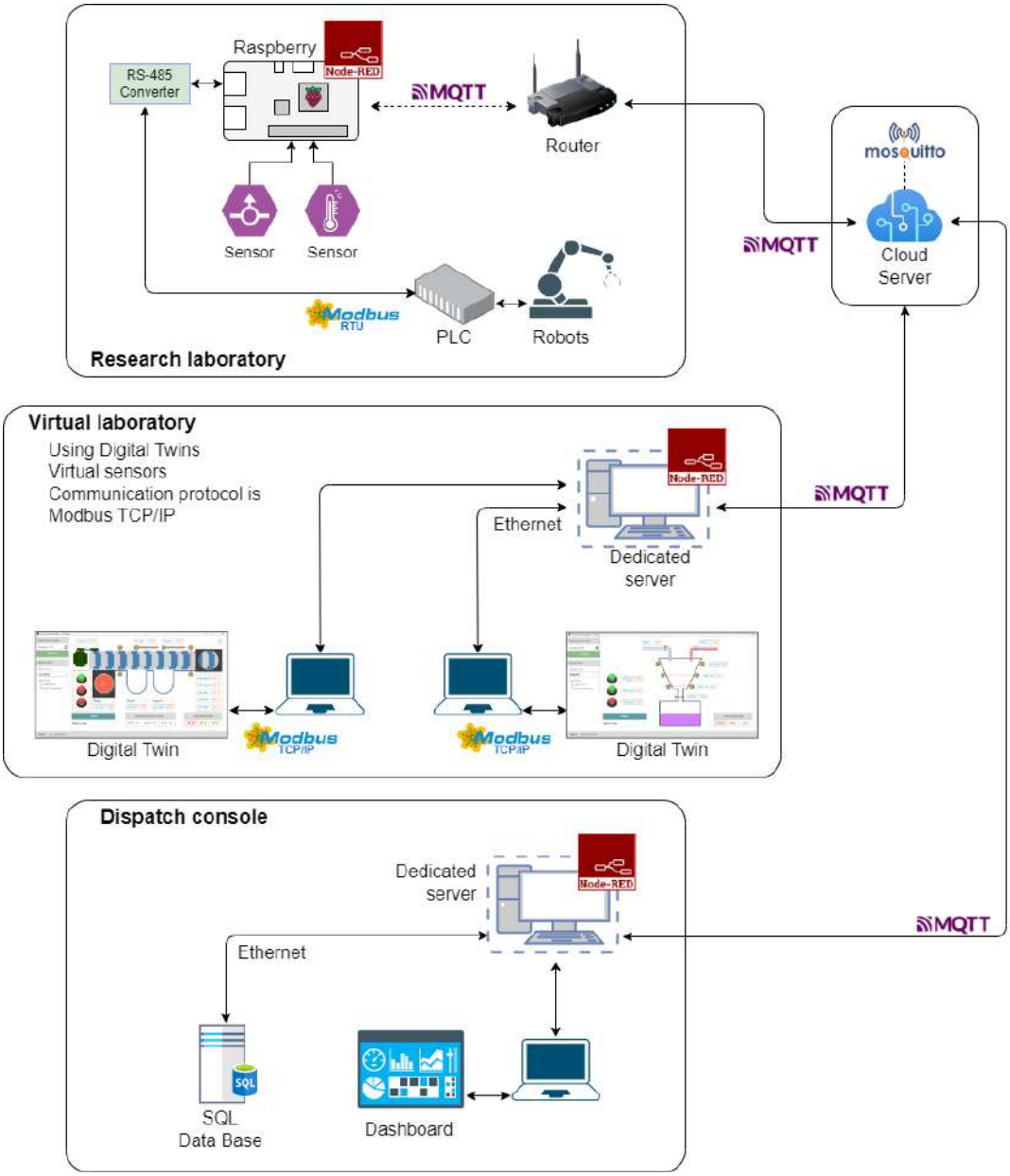


Рисунок 1.23 – Архітектура автоматизованої системи, що використовується для навчання основам технології промислового інтернету речей

Брокер MQTT забезпечує зв’язок між пристроями, що підключені до мережі, та дозволяє виконувати різні операції з повідомленнями, такі як підписка, публікація, збереження повідомлень тощо. Mosquitto також дозволяє стежити за статусом підключених пристроїв та обробляти помилки підключення.

В системі, що описується, Mosquitto використовується для реалізації задач IoT-проектів, таких як моніторинг даних, збір даних, управління промисловими процесами та інші.

1.7.2 Взаємодія між обладнанням дослідної лабораторії

На рис. 1.24 подана структурна схема автоматизації дослідної лабораторії.

Обладнання дослідної лабораторії підключається до міні-ПК Raspberry PI зі встановленою операційною системою Linux та програмою Node-RED.

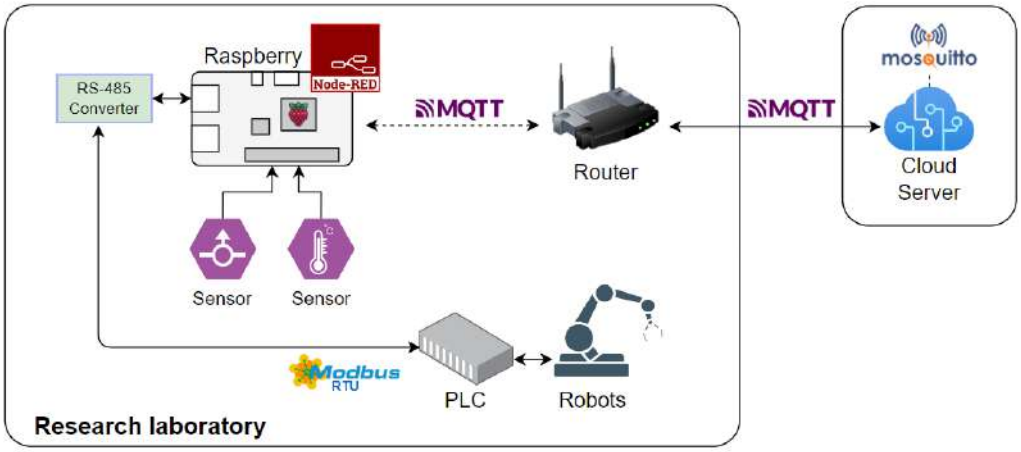


Рисунок 1.24 – Приклад автоматизації дослідної лабораторії

Node-RED – це візуальна програмна оболонка, яка дозволяє розробникам створювати взаємодію між різними системами та пристроями шляхом використання потокової парадигми програмування. Він може бути встановлений на локальному комп'ютері або використовуватися в хмарному середовищі.

Node-RED підтримується на різних платформах, включаючи Windows, Linux та macOS, і може бути встановлений на різні пристрої, включаючи Raspberry Pi та інші одноплатні комп'ютери.

За допомогою Node-RED можна створювати потоки даних, які складаються з вузлів, що виконують різні операції. Вузли можуть бути підключені один до одного, щоб перетворювати, аналізувати та передавати дані. Node-RED має вбудовані вузли для підключення до різних пристроїв та сервісів, таких як сенсори, бази даних, мережеві служби та інші. Node-RED також дозволяє створювати власні вузли та функції.

Node-RED має інтуїтивний інтерфейс, що дозволяє візуально зображувати потоки даних та редагувати їх за допомогою миші та клавіатури. Завдяки

вбудованим бібліотекам, Node-RED пропонує широкі можливості для створення потоків даних та інтеграції різних систем.

Датчики вимірювання температури, датчики положення, стану контактів та інші підключаються до міні-ПК Raspberry як безпосередньо до його портів вводу-виводу, так і через пристрої розширення, або перетворювачі інтерфейсів.

Управління периферійними пристроями, наприклад, такими, як роботизовані маніпулятори, відбувається за допомогою промислових логічних контролерів, що підключаються до мережі RS-485 за допомогою відповідних перетворювачів.

Завдяки використанню Node-RED стає можливим досить просто оброблювати всі вище перелічені сигнали в єдиній програмній системі та надсилати отриману інформацію на вимогу віддалених пристроїв за допомогою протоколу MQTT.

Обмін даними між Raspberry та ПЛК відбувається через протокол Modbus RTU, що також підтримується Node-RED.

Взаємодія з хмарним середовищем відбувається за допомогою бездротового інтерфейсу через Wi-Fi роутер.

1.7.3 Віртуальна лабораторія та цифрові двійники

Відпрацювання методів керування технологічними процесами з використанням сучасних засобів автоматизації відбувається із застосуванням віртуальних макетів та цифрових двійників складного промислового обладнання, що поєднані у віртуальну лабораторію (рис. 1.25).

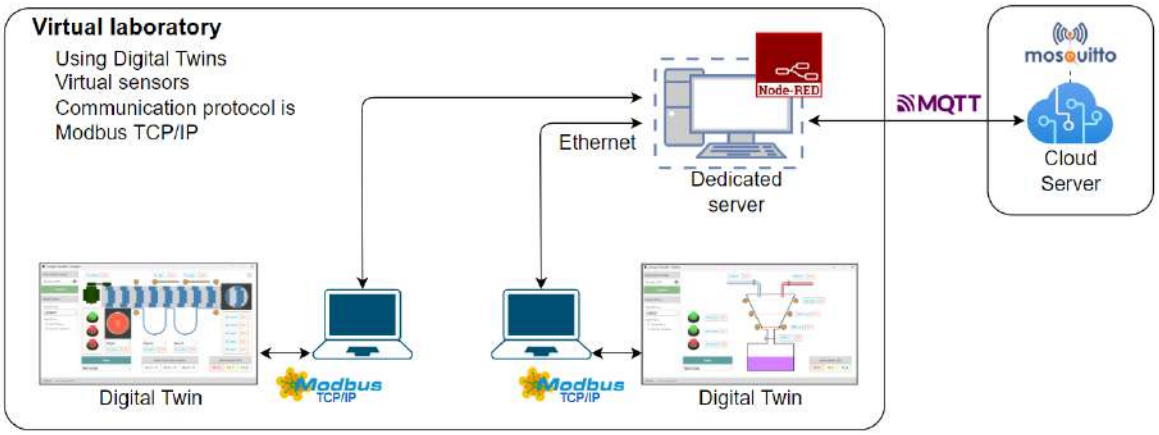


Рисунок 1.25 – Мережа віртуальних засобів автоматизації

Віртуальна лабораторія має власний виділений сервер, що працює під управлінням операційної системи Linux на віртуальному комп'ютері в середовищі VMware. Віртуальні прилади представлені наступним набором цифрових копій лабораторних макетів:

– світлова колона;

– штампувальний автомат;

– конвеєр та технологічна лінія;

– модуль формування та відображення стану дискретних сигналів;

– програмований логічний контролер Модулі вводу-виводу дискретних сигналів;

– дозатор речовин;

– аналого-цифровий перетворювач;

– семісегментний чотирьох розрядний цифровий індикатор.

Взаємодія віртуального персонального комп'ютера (ПК) з макетами відбувається завдяки промисловому протоколу Modbus TCP/IP. Керує процесом отримання інформації від віртуальних датчиків, в також управляє виконавчими пристроями програма Node-RED.

Через розроблені потоки в Node-RED відбувається передача повідомлень між хмарним сервером та віртуальними макетами з використанням протоколу MQTT. В будь-який момент часу віддалені користувачі (дослідники) можуть впливати на роботу цифрових двійників засобів автоматизації, керувати виконавчими механізмами та отримувати інформацію від віртуальних датчиків макетів.

1.7.4 Підсистема моніторингу та диспетчеризації

Зберігання отриманих даних в процесі обміну інформацією з макетами дослідної та віртуальної лабораторій відбувається в підсистемі моніторингу та диспетчеризації (рис. 1.26).

Підсистема диспетчеризації (dispatching) в Industrial Internet of Things – це система, яка керує передачею даних та задач між різними пристроями та компонентами в мережі IIoT. Функції цієї підсистеми можуть включати:

– розподілення завдань, та прийняття рішень, який пристрій або компонент IIoT отримає нове завдання та які дії потрібно виконати, щоб завдання було успішно завершено;

– керування маршрутизацією даних та задач між різними пристроями та компонентами IIoT;

– контроль потоку даних та задач між різними пристроями та компонентами мережі для забезпечення рівноцінного навантаження на пристрої;

– керування залежностями між різними задачами та компонентами мережі IIoT, та відстежування залежностей між задачами для забезпечення їх виконання в правильному порядку;

– забезпечення моніторингу різних параметрів та метрик мережі пристроїв для виявлення можливих проблем та відслідковування показників ефективності;

– забезпечення збереження отриманих даних та організації доступ до них в будь-який час за запитом.

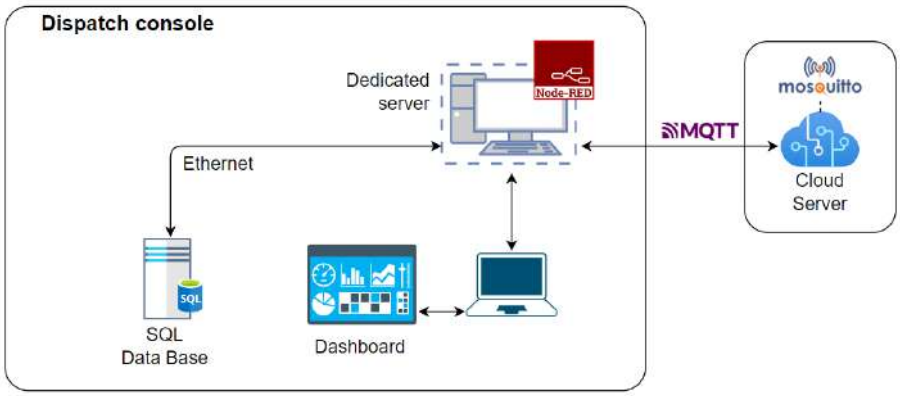


Рисунок 1.26 – Підсистема моніторингу та диспетчеризації

В якості сервера бази даних виступає СУБД Postgresql. Взаємодія з хмарним сервером відбувається через виділений сервер під управлінням ОС Linux. Завдяки встановленій на ньому програми Node-RED виконується обмін повідомленнями в форматі протоколу MQTT.

Набір інструментів Dashboard від Node-RED дозволяє швидко створювати та візуалізувати графіки, таблиці, кнопки та інші елементи інтерфейсу користувача (UI) для керування та моніторингу пристроїв та даних в мережі IoT.

1.8 Контрольні запитання та завдання

1. Які основні характеристики Індустрії 4.0 і як вони впливають на технологічне обладнання АСУТП?

2. Опишіть архітектуру кіберфізичної системи та її складові частини.

3. Яка роль Інтернету речей (IoT) в Індустрії 4.0?

4. Як периферійні обчислення допомагають впроваджувати Інтернет речей у виробництво?

5. Яку роль відіграє технологія 6G в промисловому Інтернеті речей (IIoT)?

6. Як технологія 6G може бути використана для кіберфізичних систем (CPS) та IIoT?

7. Які комунікаційні технології використовуються для цифровізації на інтелектуальному заводі?

8. Як прозорість виробничих та логістичних процесів сприяє контролю стану технологічного процесу?

9. Що таке віртуальна лабораторія та цифрові двійники і як вони використовуються у дослідній лабораторії?

2 ОСНОВИ NODE-RED

2.1 Встановлення Node-RED

Node-RED – це інструмент програмування для об’єднання апаратних пристроїв, API та онлайн-сервісів новими та цікавими способами. Він надає редактор на основі браузера, який спрощує об’єднання потоків за допомогою широкого діапазону вузлів у палітрі, які можна розгорнути у середовищі виконання одним клацанням миші.

Node-RED може бути використаний в IIoT для розробки та побудови різноманітних застосунків, які об’єднують промислові пристрої та датчики з системами збору, обробки та аналізу даних.

Даний програмний засіб може використовуватись для збору даних з різних промислових пристроїв та датчиків, а також для обробки і аналізу цих даних. Для цього в Node-RED існує багато різноманітних вузлів, які дозволяють підключатись до різних пристроїв та протоколів, таких як Modbus, OPC UA, MQTT тощо.

Node-RED містить набір візуальних вузлів, які дозволяють створювати графіки, таблиці та інші візуальні елементи для відображення даних, що може допомогти операторам та інженерам в реальному часі відстежувати роботу промислових систем та виявляти можливі проблеми.

Програма дозволяє створювати складні процеси керування та автоматизації промислових систем з використанням логічних вузлів та вузлів затримки, а також може бути підключений до різноманітних систем зберігання та обробки даних, таких як бази даних та хмарні сервіси, для збереження та аналізу даних.

Node-RED надає редактор потоків на основі браузера, який дозволяє легко об’єднувати потоки за допомогою широкого діапазону вузлів у палітрі. Потім потоки можна розгорнути у середовищі виконання в один клік.

Функції JavaScript можна створювати в редакторі за допомогою форматованого текстового редактора. Вбудована бібліотека дозволяє зберігати корисні функції, шаблони або потоки для повторного використання.

Полегшене середовище виконання побудовано на основі Node.js, повністю використовуючи переваги керованої подіями моделі без блокування. Це робить

його ідеальним для роботи на межі мережі на недорогому обладнанні, такому як Raspberry Pi, а також у хмарі. Маючи понад 225000 модулів у сховищі пакетів Node, можна легко розширити діапазон вузлів палітри, щоб додати нові можливості.

Розглянемо порядок дій для встановлення Node-RED на комп'ютері з операційною системою Linux.

Для локального застосування Node-RED на комп'ютері, перш за все, необхідно встановити Node.js. Порядок дій по встановленню залежить від типу дистрибутиву операційної системи Linux. Для прикладу, розглянемо основні кроки по налаштуванню Node.js для дистрибутивів Linux, що базуються на основі Debian і Ubuntu. Опис дій надано на сторінці https://nodejs.org/en/download/package-manager/ та https://github.com/nodesource/distributions/blob/master/README.md:

```
curl -fsSL https://deb.nodesource.com/setup_19.x | sudo -E bash - &&\
sudo apt-get install -y nodejs
```

В результаті виконання команди виконається встановлення середовища Node.js (рис. 2.1).

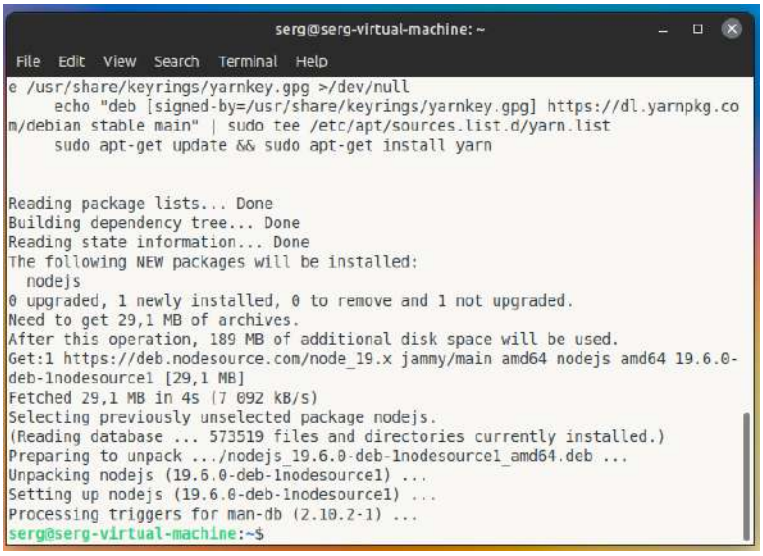


Рисунок 2.1 – Встановлення середовища Node.js

Щоб перевірити, чи працює інсталяція (і чи працюють сценарії інсталяції), використаємо наступну команду:

```
curl -fsSL https://deb.nodesource.com/test | bash -
```

Результат перевірки подано на рис. 2.2.

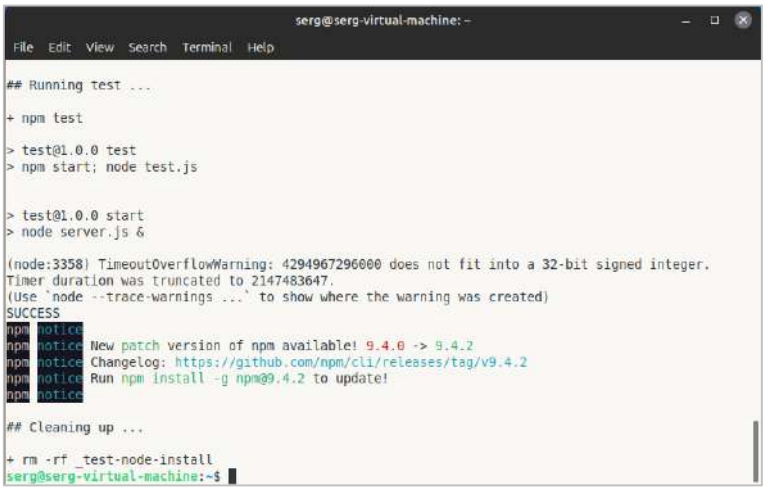


Рисунок 2.2 – Результат перевірки правильності встановлення Node.js

Можна бачити, що в даному випадку в результаті тестування пропонується оновити поточну версію npm до версії 9.4.2. Виконаємо це командою:

```
sudo npm install -g npm@9.4.2
```

Виконавши повторне тестування (рис. 2.3), можна бачити, що проблем не виявлено.

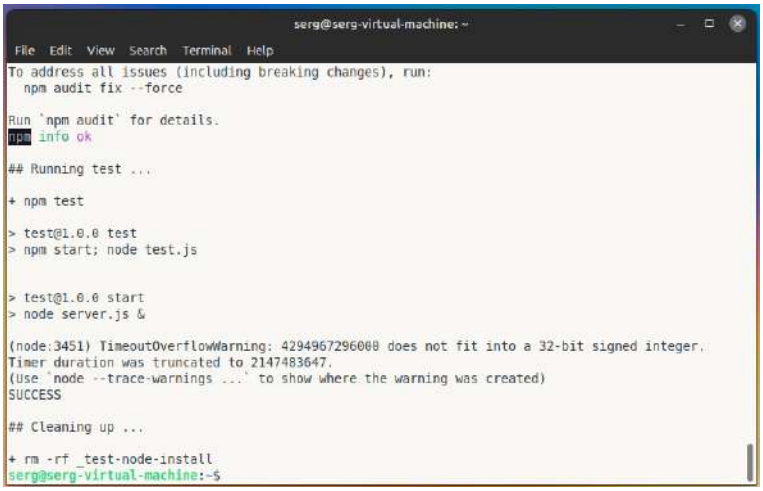


Рисунок 2.3 – Повторне тестування після встановлення оновлення для npm

Npm – це менеджер пакетів JavaScript (JavaScript package manager). Npm є найбільшим у світі реєстром програмного забезпечення. Розробники програм з відкритим кодом використовують npm для спільного використання та запозичення пакетів, а багато організацій також використовують npm для керування приватною розробкою.

Npm складається з трьох окремих компонентів:

– веб-сайт;

– інтерфейс командного рядка (CLI);

– реєстр.

Можна використовувати веб-сайт, щоб знаходити пакети, налаштовувати профілі та керувати іншими аспектами роботи з npm. Наприклад, можна виконати налаштування для керування доступом до публічних або приватних пакетів.

В свою чергу, CLI запускається з терміналу і більшість розробників саме таким чином взаємодіють із npm.

Реєстр – це велика загальнодоступна база даних програмного забезпечення JavaScript і метаінформації, що його оточує.

Npm використовується для:

– адаптації пакетів коду для своїх програм або включення пакетів в тому вигляді, в якому вони є;

– завантаження окремих інструментів, якими можна скористатися відразу;

– запуску пакетів без завантаження за допомогою npx;

– обміну кодом з будь-яким користувачем npm;

– обмеження коду для певних розробників;

– створення організації для координації обслуговування пакетів, кодування та розробників;

– створення віртуальної команди за допомогою організацій;

– керування кількома версіями коду та залежностями коду;

– оновлення програми після оновлення основного коду;

– знаходження інших розробників, які працюють над подібними проблемами та проєктами.

Перевірка версії Node.js:

```
node -v
```

Перевірка версії npm:

```
npm -v
```

Результати перевірки показані на рис. 2.4.

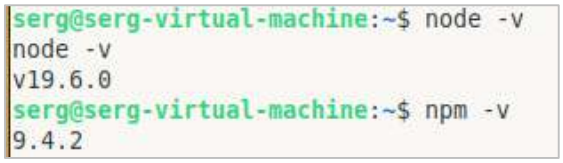


Рисунок 2.4 – Результати перевірки версій встановлених програм

Наступним кроком встановлюємо Node-RED:

```
sudo npm install -g --unsafe-perm node-red
```

Дана команда встановить Node-RED як глобальний модуль разом із його залежностями. Результат роботи команди nolano на рис. 2.5.

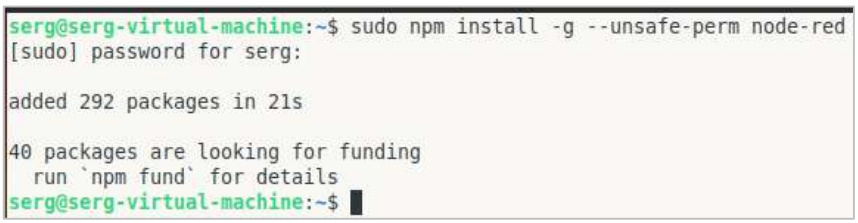


Рисунок 2.5 – Результат роботи команди встановлення Node-RED

Після встановлення використаємо команду

```
node-red
```

для запуску Node-RED у локальному терміналі (рис. 2.6). Для завершення роботи програми можна скористатися Ctrl+C або закрити вікно терміналу, щоб зупинити Node-RED.

Після вдалого запуску можна отримати доступ до редактора Node-RED, вказавши у браузері http://localhost:1880.

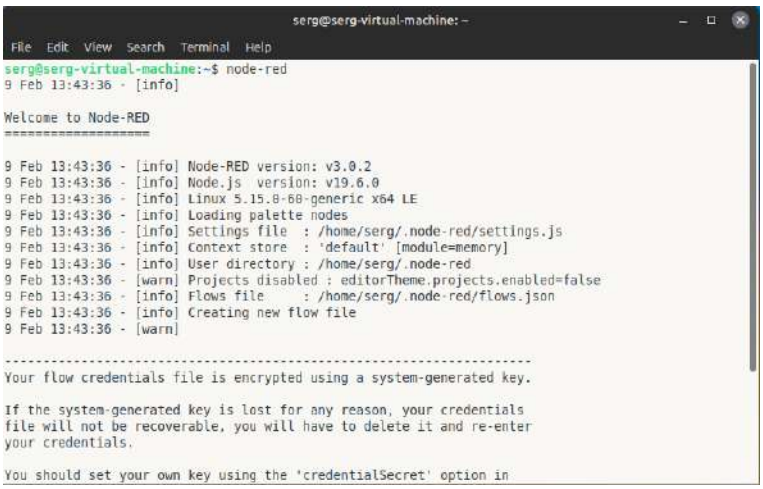


Рисунок 2.6 – Запуск Node-RED

В звіті журналу роботи програми можна дізнатися різноманітну інформацію:

– версії Node-RED і Node.js;

– будь-які помилки, коли він намагався завантажити вузли палітри;

– розташування файлу налаштувань і каталогу користувача;

– ім’я файлу потоків, який він використовує.

Node-RED використовує flows_<hostname>.json як файл потоків за замовчуванням. Це можна змінити, надавши назву файлу потоку як аргумент для команди node-red.

Також можна використовувати завантажений код Node-RED для доступу до нього з будь-якого комп’ютеру локальної мережі. Для цього потрібно дізнатися IP-адресу комп’ютера, де завантажено ядро Node-RED та звернутися до нього в такому форматі:

```
http://<IP-address>:1880.
```

На рис. 2.7 подано приклад віддаленого доступу до віртуального ПК з IP-адресою 192.168.130.128.

2.2 Основні вузли Node-RED

2.2.1 Класифікація вузлів

Класифікація вузлів за функціональністю:

– ввідні вузли (Input Nodes) – ці вузли отримують дані з різних джерел, таких як датчики, бази даних, соціальні мережі, веб-сайти та інше;

– вузли обробки даних (Processing Nodes) – ці вузли здійснюють обробку даних, яка може включати у себе фільтрацію, маніпулювання, обчислення, форматування, зберігання та інше;

– вихідні вузли (Output Nodes) – ці вузли відправляють оброблені дані в різні кінцеві точки, такі як бази даних, веб-сервіси, інші системи, електронна пошта та інше;

– утиліти (Utility Nodes) – ці вузли забезпечують додаткові функції, такі як таймери, логіку, обробку помилок, управління потоками даних та інше.

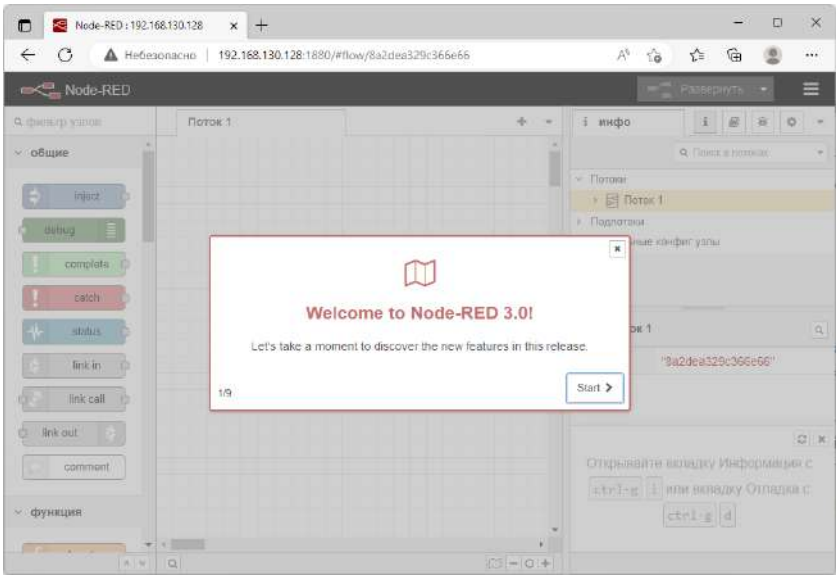


Рисунок 2.7 – Приклад віддаленого доступу до віртуального ПК з IP-адресою 192.168.130.128

Класифікація вузлів за джерелом походження:

– вбудовані вузли (Built-in Nodes) – це стандартні вузли, які надаються Node-RED для різних функцій та задач. До вбудованих вузлів загального призначення можна віднести: inject, debug, function, change, switch, range;

– сторонні вузли (Third-party Nodes) – ці вузли розробляються сторонніми розробниками, які додають до Node-RED нові функції та можливості;

– користувацькі вузли (Custom Nodes) – ці вузли розробляються користувачами для виконання спеціальних функцій та задач, які не входять до стандартного набору вузлів Node-RED.

Для класифікації вузлів за джерелом походження, можна також виділити наступні категорії:

– інтеграційні вузли (Integration Nodes) – ці вузли дозволяють інтегрувати Node-RED з іншими системами та сервісами, такими як MQTT, OPC UA, REST API, SOAP тощо;

– інструментальні вузли (Tool Nodes) – ці вузли надають інструменти для розробки та тестування Node-RED, такі як вузли для налагодження, запису та відтворення даних, генерації тестових даних та інші;

– компоненти вузлів (Node Components) – це вузли, які складаються з кількох компонентів, таких як HTML, CSS та JavaScript. Ці вузли дозволяють розробникам створювати власні кастомізовані вузли з унікальними функціями та зовнішнім виглядом.

2.2.2 Структура повідомлення

Потік Node-RED складається з серії взаємопов'язаних вузлів (провідних вузлів). Усі вузли мають вхід і можуть мати нуль або декілька виходів. Вузли обмінюються даними між собою за допомогою об'єкта msg.

Кожен вузол отримує об'єкт повідомлення від попереднього вузла, а потім може передати цей об'єкт повідомлення наступному вузлу в потоці.

Об'єкт msg є стандартним об'єктом JavaScript і має кілька властивостей або частин залежно від того, звідки він походить. В процесі роботи можна побачити властивості повідомлення, надіславши його до вузла «debug» (рис. 2.8).

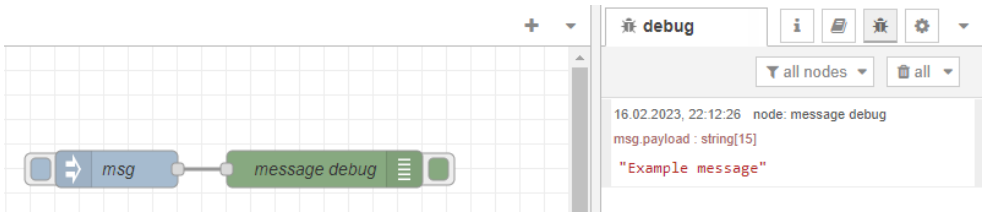


Рисунок 2.8 – Приклад відображення вмісту payload повідомлення

Зазвичай вузли мають властивість корисного навантаження (payload) – це властивість за замовчуванням, з якою працюватиме більшість вузлів.

Під час створення потоків вибір властивостей, які використовуються для повідомлення, значною мірою визначатиметься потребами вузлів у потоці.

Більшість вузлів очікують на вході повідомлення з типом msg.payload, і це характерно для більшості вузлів з якими стикається розробник при виконанні повсякденних задач.

Наприклад, розглянемо потік, який отримує ідентифікатор у корисному навантаженні (payload) повідомлення MQTT. Потім він використовує цей ідентифікатор для запита до бази даних, щоб знайти відповідний запис (рис. 2.9).

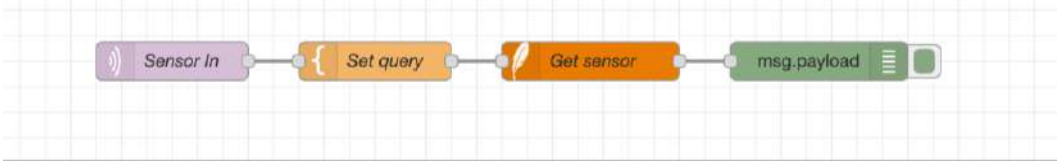


Рисунок 2.9 – Приклад обробки повідомлення MQTT

Вузол бази даних помістить свій результат у корисне навантаження повідомлення, яке він надсилає, перезаписуючи вихідне значення ідентифікатора. Якщо потік потребує можливості посилатися на це значення ідентифікатора пізніше, він може використати вузол зміни, щоб скопіювати значення в іншу властивість, яка не буде перезаписана (рис. 2.10).

Рисунок 2.10 – Використання вузла Change для
копіювання msg.id до msg.payload

Це підкреслює важливий принцип: вузли не повинні змінювати або видаляти властивості повідомлень, які не пов’язані з їх функціональністю. Наприклад, у більшості випадків вузол Function повинен надсилати той самий об’єкт повідомлення, який він отримав, а не створювати новий об’єкт повідомлення.

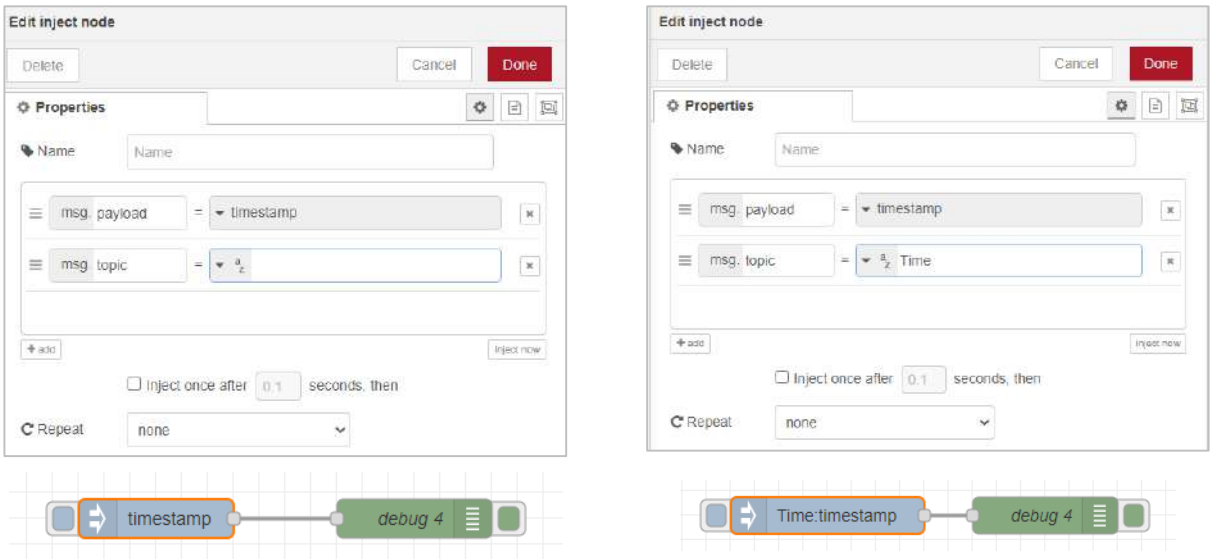
В деяких випадках використовується msg.topic. Властивість msg.topic відображається в бічній панелі виводу повідомлень в процесі налагодження (рис. 2.11).

Рисунок 2.11 – Приклад відображення msg.topic

Деякі вузли також розглядають msg.topic як такий, що має особливе значення. Його можна використовувати для ідентифікації джерела повідомлення або для ідентифікації різних «потоків» повідомлень в одному потоці. Він також відображається на бічній панелі налагодження з кожним повідомленням.

Наприклад, вузол «MQTT In» встановлює msg.topic в якості теми повідомлення, що було отримано. Потім вузол затримки можна налаштувати на обмеження частоти повідомлень відповідно до їх теми.

Хоча потік може не використовувати вузли, які безпосередньо залежать від msg.topic, його можна використовувати для надання додаткової контекстної інформації про повідомлення. На рис. 2.12 можна побачити два варіанти налаштування повідомлення. На рис. 2.12, а – повідомлення без msg.topic, тоді як на рис. 2.12, б – повідомлення з використанням властивості msg.topic.



а) б)

Рисунок 2.12 – Різні варіанти відображення повідомлення:
а) повідомлення без msg.topic; б) повідомлення з використанням msg.topic

На рис. 2.13 подано результат застосування властивості msg.topic. Верхнє повідомлення демонструє використання потоку без встановлення msg.topic, а нижнє повідомлення відображає варіант використання значення «Time» в якості властивості msg.topic.

Рисунок 2.13 – Приклад застосування властивості msg.topic

За замовчуванням вузол «debug» відображатиме властивість msg.payload, але можна змінити налаштування, щоб відобразити всі властивості повідомлення (повний об’єкт повідомлення) (рис. 2.14).

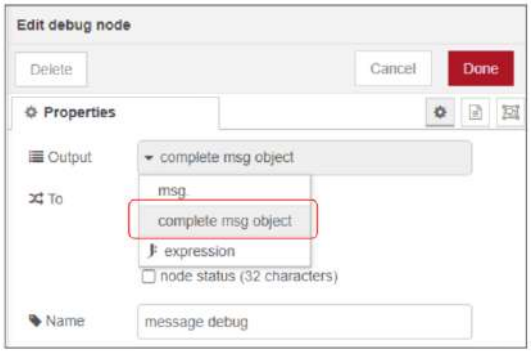


Рисунок 2.14 – Зміна властивості вузла «debug»

В результаті відправлене повідомлення буде відображено в форматі, що подано на рис. 2.15. Тут можна бачити, що крім власне повідомлення, виводиться ще службова інформація, наприклад параметр «_msgid». В залежності від джерела надходження інформації, зміст повідомлення у вікні debug буде відрізнятися.

Властивість _msgid – це ідентифікатор повідомлення, доданий Node-RED і може використовуватися для відстеження об’єктів повідомлення.

Властивості об’єкта повідомлення можуть мати будь-який дійсний тип JavaScript, наприклад: String, Integer, Object, або Boolean.

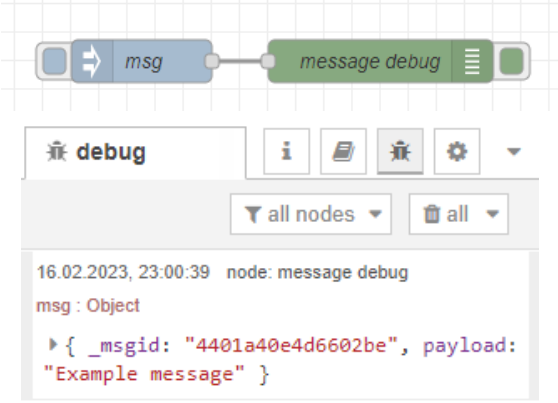


Рисунок 2.15 – Повна інформація у вікні debug

Node-RED надає ряд вузлів, що можуть змінювати об’єкт повідомлення без необхідності писати код JavaScript. Основними вузлами є «change» (заміна), «split» розділення, «join» приєднання, «switch» перемикання.

Доступ до властивостей повідомлення msg можна отримати так само, як і до будь-якого об’єкта JavaScript. Для цього використовується вузол «function».

Усередині функції також можна створити новий об’єкт повідомлення за допомогою наступних рядків:

```
var newMsg = { payload: msg.payload,topic:msg.topic };
return newMsg;
```

Доступ до payload повідомлення можна отримати за допомогою запису:

```
var payload = msg.payload;
```

Для зміни вмісту повідомлення можна використати наступний запис:

```
msg.payload = payload;
```

Так само, доступ до теми повідомлення можна отримати за допомогою рядка коду:

```
var topic = msg.topic;
```

Змінити тему повідомлення можна за допомогою наступного запису:

```
msg.topic = topic;
```

Node-RED надає можливість додавати нові властивості до об’єкта повідомлення. Наприклад, якщо є об’єкт, що містить значення датчиків, які виглядають так:

```
sensors = {sensor1:20,sensor2:21}
```

то можна додати його до об’єкта msg за допомогою команди:

```
msg.sensors = sensors
```

Нове повідомлення буде передано до наступного вузла разом із наявними властивостями існуючого повідомлення (рис. 2.16).

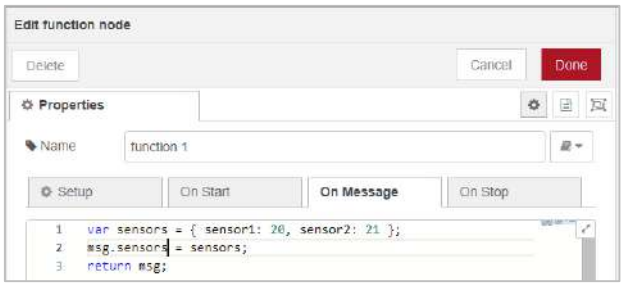


Рисунок 2.16 – Додавання нового поля до повідомлення

В результаті роботи потоку буде виведено наступне повідомлення у вікно debug (рис. 2.17).

Рисунок 2.17 – Виведення доданого поля у вікні debug

Вигляд тестового потоку з використанням функції подано на рис. 2.18.

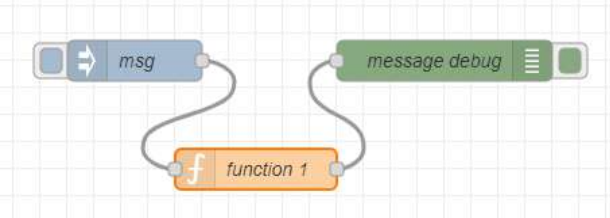


Рисунок 2.18 – Вигляд тестового потоку з використанням функції

2.2.3 Команда «Inject»

Команда «Inject» найчастіше використовується в режимі налагодження програми. Вона вкидає задане повідомлення до потоку вручну або через рівні проміжки часу (рис. 2.19).

Рисунок 2.19 – Команда Inject

Дані в повідомленні можуть мати різні типи, включаючи рядок, об’єкт JavaScript або мітку поточного часу (рис. 2.20).

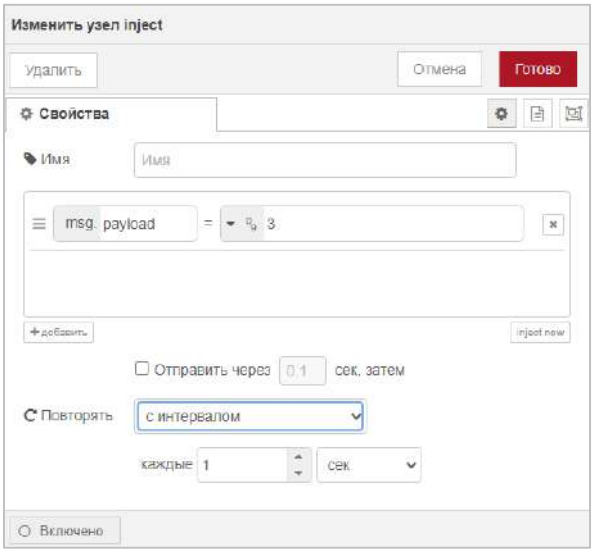


Рисунок 2.20 – Вікно налаштування властивостей команди Inject

Вузол Inject може ініціювати виконання потоку з певними даними (значення payload). Дані за замовчуванням – це позначка поточного часу

в мілісекундах, що пройшли з 1 січня 1970 року. Команда також підтримує виведення рядків, чисел, логічних значень, об’єктів JavaScript або потокових/глобальних контекстів.

За замовчуванням вузол запускається вручну при натисканні кнопки в редакторі. Його також можна налаштувати на автоматичний запуск через рівні проміжки часу або за розкладом. Також він також може бути налаштований на одноразове вкидання повідомлення при кожному перезапуску потоків.

Максимальний інтервал, який можна вказати, становить близько 596 год./24 дні. Однак, якщо потрібні інтервали, що перевищують один день, слід розглянути можливість використання функцій планувальника у вузлі, які зможуть коректно працювати з перебоями електроенергії та перезапуском.

У параметрах «з інтервалом у проміжку» та «у певний час» використовується стандартна система «cron». Це означає, що «20 хвилин» будуть в наступній годині, через 20 хвилин і через 40 хвилин – а не через 20 хвилин. Якщо потрібно вкинути повідомлення кожні 20 хвилин, необхідно використовувати параметр «з інтервалом».

Щоб включити багаторядковий текст до рядкового значення, необхідно використовувати вузол Function для формування даних.

2.2.4 Вузол «function»

Функціональний вузол використовується для запуску коду JavaScript з метою обробки об’єкта msg. Функціональний вузол приймає об’єкт msg як вхідні дані та може повертати 0 або більше об’єктів повідомлення як вихідні дані.

Цей об’єкт повідомлення повинен мати властивість корисного навантаження (msg.payload) і, як правило, має інші властивості залежно від вузлів обробки. Зовнішній вигляд вузла function подано на рис.2.21.

Рисунок 2.21 – Зовнішній вигляд вузла function

Розглянемо основні принципи використання вузла «function».

Коли у потік додається новий функціональний вузол, зазвичай відразу переходять до його редагування. В даному випадку за замовчуванням можна

побачити один рядок коду, який повертає об’єкт msg, і порожній рядок вище, де можна почати вводити свій власний код (рис. 2.22).

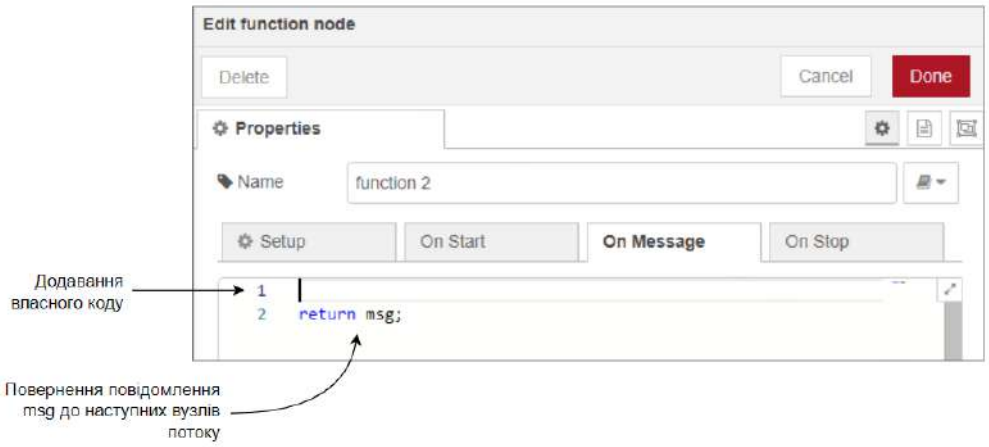


Рисунок 2.22 – Принцип редагування вмісту функції

Якщо не відбудеться повернення об’єкту msg, потік припиняється.

Потік нижче використовує вузол функції з кодом за замовчуванням, який просто повертає об’єкт msg. Ефект полягає в тому, щоб просто передати об’єкт msg і всі його властивості від вхідного вузла до вихідного вузла та наступного вузла в потоці (debug) (рис. 2.23).

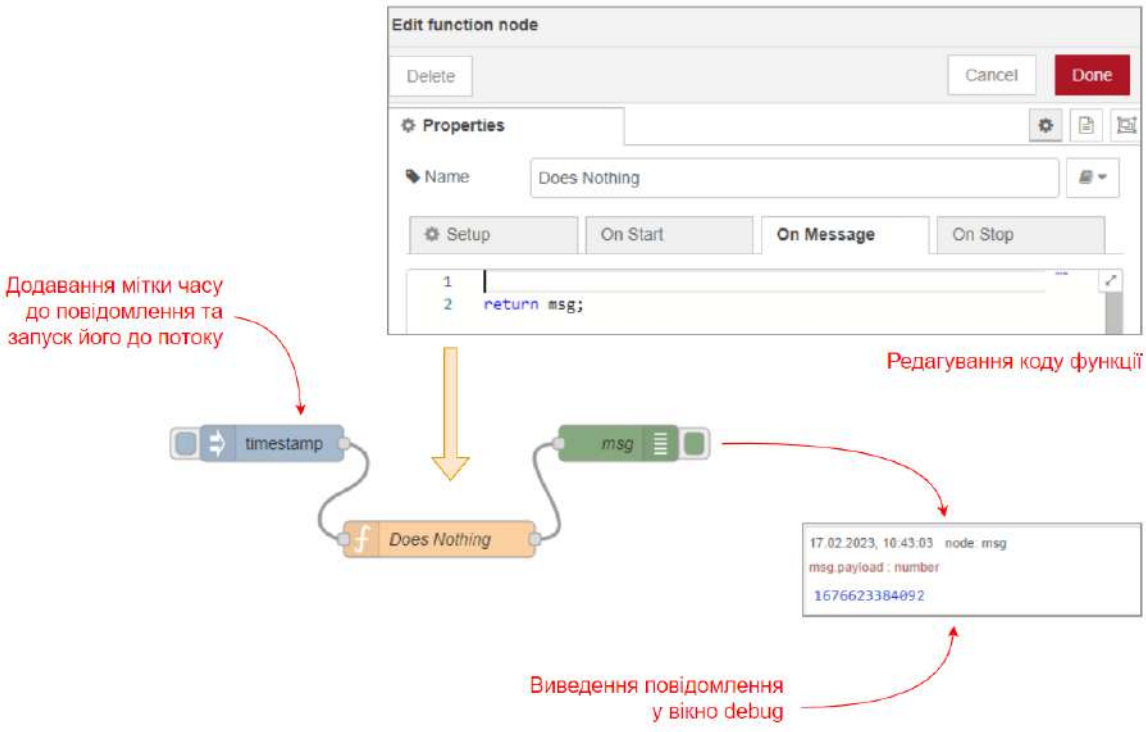


Рисунок 2.23 – Простий приклад застосування функції в потоці

Вузол inject вставляє об’єкт msg у потік із міткою часу Unix як корисним навантаженням і порожньою темою. Він передається через вузол функції, який нічого не робить, і можна бачити, що це відображається у вікні налагодження за допомогою вузла debug.

Розглянемо приклад модифікації повідомлення за допомогою функції. Для цього відредагуємо вузол inject для введення в потік корисного навантаження з рядком «test string» і темою «Test» (рис. 2.24).

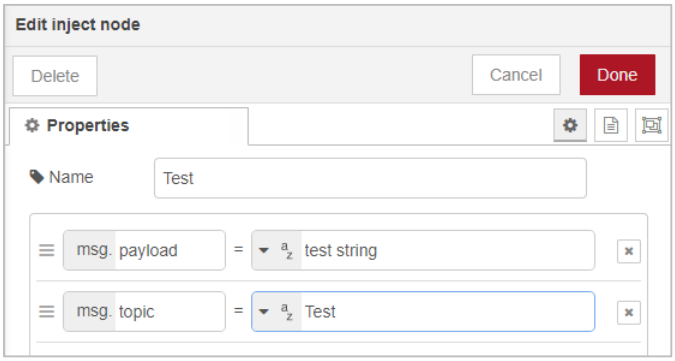


Рисунок 2.24 – Редагування вузла inject

Якщо ми передамо це в нашу функцію, яка ще нічого не робить, як і раніше, то в результаті ми отримаємо просте виведення інформації у вікно налагодження (рис. 2.25).

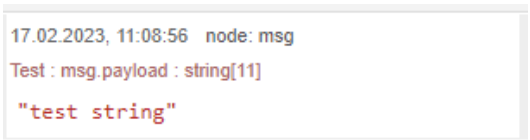


Рисунок 2.25 – Виведення тестової інформації

Наразі ми бачимо тему повідомлення, як тест, а також корисне навантаження, що теж відображається, як тестовий рядок.

Створимо функцію, що буде змінювати корисне навантаження та тему повідомлення на верхній регістр. Для цього додамо до функції наступний код:

```
var payload = msg.payload; //get payload
msg.payload = payload.toUpperCase(); //convert to uppercase
var topic = msg.topic; //get topic
msg.topic = topic.toUpperCase();//convert to uppercase
return msg;
```

Після завантаження модифікація програми та натискання на кнопку керування вузлом inject у вікні налагодження можна побачити, що вміст об’єкту msg змінився (рис.2.26).

Рисунок 2.26 – Застосування функції UpperCase

Функціональний вузол може мати кілька входів. Зазвичай потрібно відрізняти один вхід від іншого, і це робиться за допомогою використання тем (topic). Побудуємо тестовий потік з двома входами, що поєднуються з функцією (рис. 2.27).

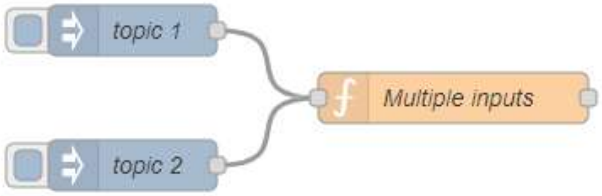


Рисунок 2.27 – Два входи поєднані з однією функцією

Для розпізнавання вхідних потоків функціональний вузол може використовувати оператор if і властивість topic, як показано у фрагменті коду нижче:

```
if (msg.topic == "topic1")
    node.log("message from topic1");
if (msg.topic == "topic2")
    node.log("message from topic2");
return msg;
```

В даному прикладі використовується команда node.log(…) для виведення повідомлення в терміналі, з якого виконувався запуск Node-RED. На рис. 2.28 подано приклад виводу даної інформації.

```
17 Feb 16:11:38 - [info] Updated flows
17 Feb 16:11:38 - [info] Starting flows
17 Feb 16:11:38 - [info] Started flows
17 Feb 16:11:41 - [info] [function:Multiple inputs] message from topic2
17 Feb 16:12:22 - [info] [function:Multiple inputs] message from topic1
```

Рисунок 2.28 – Приклад виводу інформації в терміналі

Зустрічаються завдання, в яких потрібно, щоб функція мала більш одного виходу. Це корисно, коли потік розбивається на окремі шляхи залежно від властивості повідомлення.

Щоб налаштувати кілька виходів, потрібно відкрити функціональний вузол та перейти у вкладку Setup. В даній вкладці можна обрати необхідну кількість виходів для функціонального блоку (рис. 2.29).

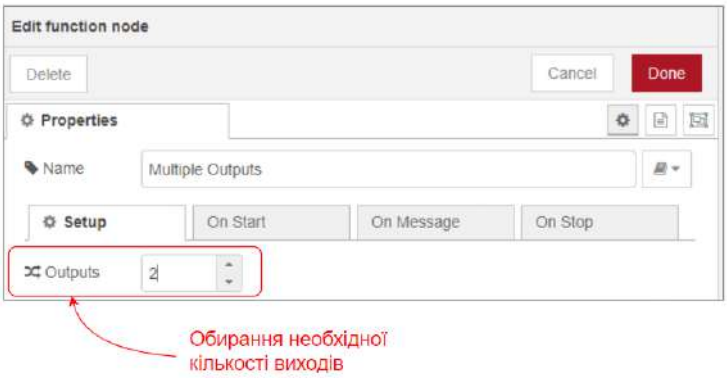


Рисунок 2.29 – Обрання необхідної кількості виходів функціонального блоку

Щоб повернути повідомлення на кілька виходів, наприкінці виконання функції потрібно повернути масив. Повернення виглядає так:

```
return [msg1, msg2];
```

В результаті, об’єкт msg1 з’явиться на виході output1 і msg2 на іншому виході (output2). Щоб зупинити один з потоків, необхідно повернути порожній об’єкт (null) на цьому виході. Отже, щоб повернути об’єкт msg на output1 і нічого не повертати на output2, використовують наступний запис:

```
return [msg1,null];          або          return [msg1];
```

В якості прикладу розглянемо задачу, коли необхідно отримати два повідомлення на вхід функції, обробити ці повідомлення та сформувати два вихідних потоки.

Для рішення даної задачі додамо на діаграму два вузла inject, щоб створити повідомлення з двома різними темами (topic_A та topic_B). Також створимо функцію з двома виходами. Функція буде надсилати повідомлення на вихід на основі назви отриманої теми (рис. 2.30).

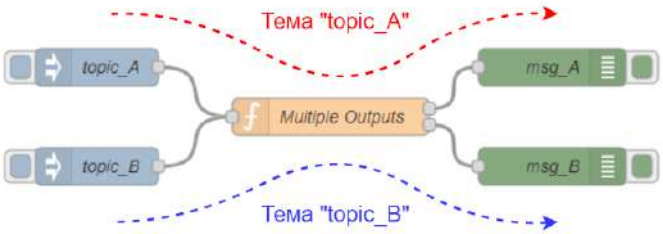


Рисунок 2.30 – Функція для розподілення потоків в залежності від теми повідомлення

Наступний код використовується у функціональному вузлі, щоб визначити шлях повідомлення на основі назви теми:

```
var topic = msg.topic;
if (topic == "topic_A") {
    return [msg, null];
}
if (topic == "topic_B") {
    return [null, msg];
}
```

Результат роботи функції подано на рис. 2.31.

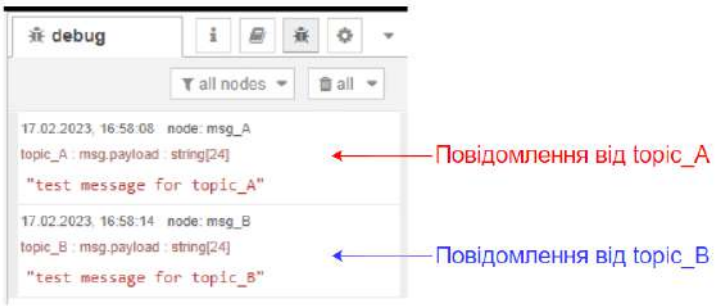


Рисунок 2.31 – Результат роботи функції розподілення потоків

Входи та виходи можуть мати назву для пришвидшення їх ідентифікації в процесі розробки. Для цього використовується вкладка налаштування (рис. 2.32).

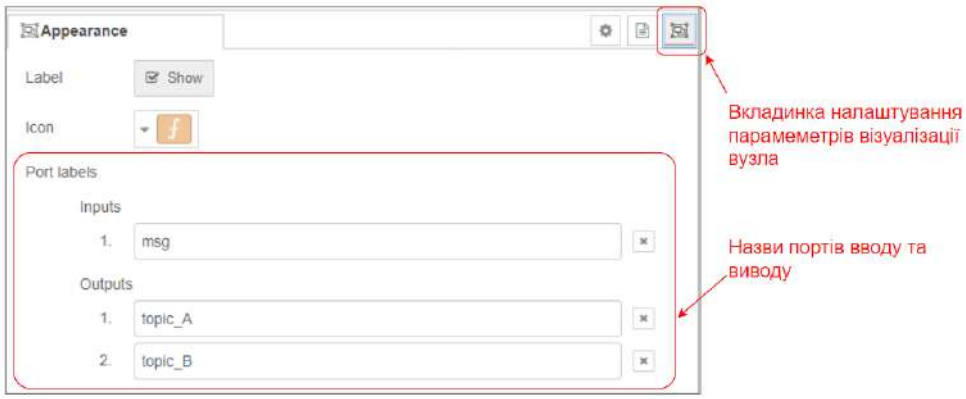


Рисунок 2.32 – Вкладка налаштування параметрів візуалізації

В результаті налаштування отримаємо підказку, що з'являється при наведені курсора миші на певний вхід або вихід (рис. 2.33).

Рисунок 2.33 – Ідентифікація входів та виходів при наведенні курсору миші

Розглянемо ще один приклад, в якому ми використовуємо вузол введення, щоб вставити тестовий рядок в повідомлення за допомогою вузла функції.

Функціональний вузол бере новий рядок тексту, що надходить і використовує його для корисного навантаження повідомлення, але замість звичайного передавання результату далі у потік, він використовує цикл for, який створює три нових повідомлення та поміщає їх у масив. Функція повертає масив рядків. Тестовий потік має вигляд, як подано на рис. 2.34.

Код вузла функції наступний:

```
var m_out = []; //array for message objects
var message = "";
```

```
for (let i = 0; i < 3; i++) {
    message = msg.payload + " " + i; //add count to message
    var newmsg = { payload: message, topic: msg.topic }
    m_out.push(newmsg);
}
return [m_out];
```

Рисунок 2.34 – Тестовий потік для створення масиву текстових рядків

Результат роботи функції подано на рис. 2.35.

Рисунок 2.35 – Результат роботи функції створення масиву текстових рядків

2.2.5 Вузол «change»

Даний вузол встановлює, змінює, видаляє чи переміщує властивості повідомлення, контексту потоку чи глобального контексту. У вузлі можна вказати кілька операцій, які будуть застосовуватися в порядку, в якому вони задані.

Доступні операції:

– встановити властивість: значення може бути різних типів або може бути взято з існуючих властивостей повідомлення або контексту;

– змінити: шукає та замінює частини тексту у властивості повідомлення. Якщо використовується регулярний вираз, налаштування «замінити на» може містити групи захоплення, наприклад $1. При повному збігу замінює лише тип;

– видалити: видаляє властивість;

– перемістити: переміщує або перейменовує властивість.

Приклад застосування подано на рис. 2.36.

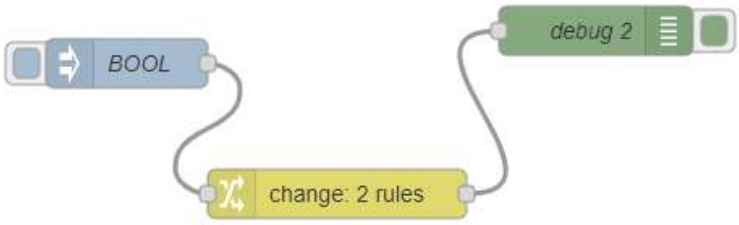


Рисунок 2.36 – Використання вузлу «change»

В даному прикладі використовується два правила заміни вмісту властивості payload:

– якщо вміст payload дорівнює «false» то в потік передається нове значення повідомлення з числом «0»;

– якщо вміст payload дорівнює «true» то в потік передається нове значення повідомлення з числом «1».

На рис. 2.37 подано приклад налаштування властивостей вузла «change».

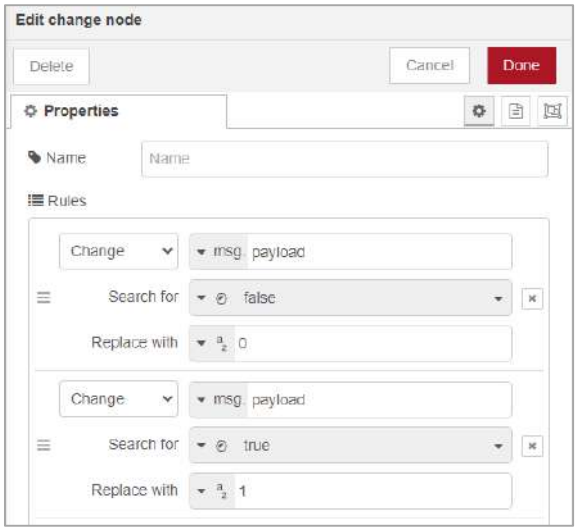


Рисунок 2.37 – Приклад налаштування властивостей вузла «change»

2.2.6 Вузол «switch»

Вузол Switch дозволяє передавати повідомлення до різних гілок потоку, оцінюючи набір правил для кожного повідомлення. Цей вузол приймає для перевірки один з наступних можливих варіантів повідомлень. Це може бути або

властивість повідомлення (message property), або властивість контексту (context property). Також в якості першого аргументу для порівняння може використовуватися вираз JSONata або змінна середовища (рис. 2.38).

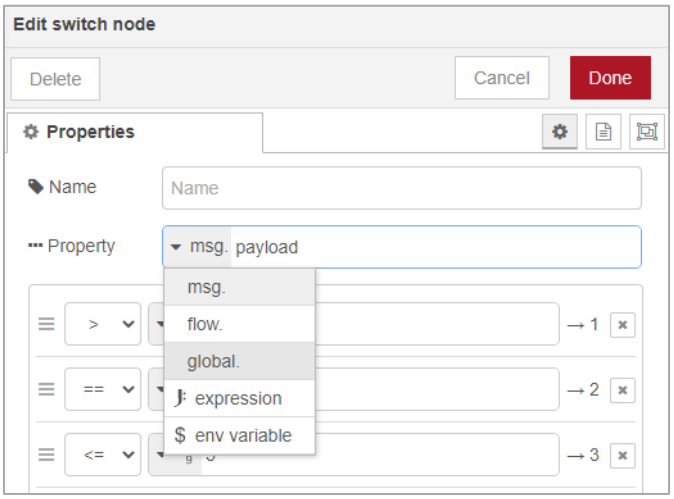


Рисунок 2.38 – Вибір варіанта вхідного повідомлення

Існує чотири типи правил за якими відбувається перевірка (рис. 2.39):

– Value rules – ці правила порівнюють значення обраної властивості повідомлення з введеним користувачем значенням;

– Sequence rules – правила можуть застосовуватись для послідовностей повідомлень, таких як згенеровані вузлом Split;

– перевірка на основі виразу JSONata – результат буде істинним, якщо отримані дані будуть відповідати вказаному виразу.

– Otherwise rules – це правило може бути використане в тому разі, якщо жодне з попередніх правил не співпало.

У вузлі «switch» можна використовувати мову запитів JSONata для перевірки вхідних даних та вибору відповідного шляху виконання дій. Наприклад, можна виконати такий запит:

```
payload.temperature > 30 ? "hot" : "cool"
```

що перевіряє значення «temperature» у вхідних даних. Якщо воно більше 30, то вибирається шлях «hot», в іншому випадку – «cool».

Щоб використовувати мову запитів JSONata у вузлі «switch», необхідно вибрати режим «JSONata expression» у вікні налаштувань вузла та ввести

відповідний запит. При цьому можна використовувати функції та операції мови JSONata для обробки даних.

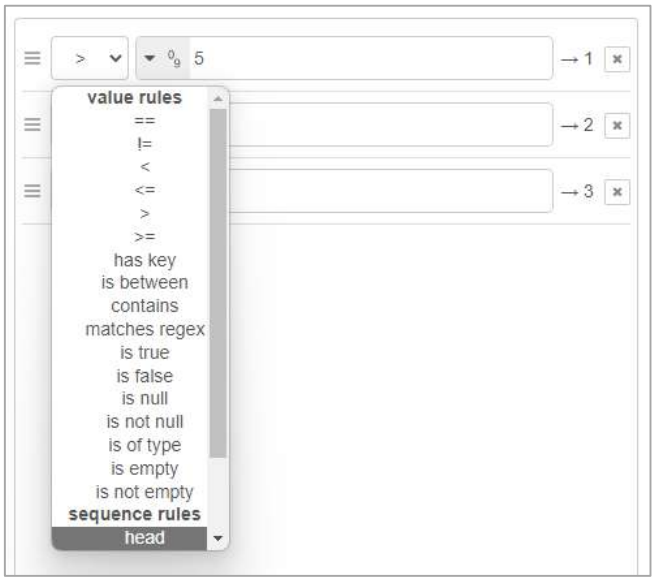


Рисунок 2.39 – Вибір правила перевірки

Вузол «switch» переводить повідомлення на всі виходи, що відповідають правилам відповідності (checking all rules), але він також може бути налаштований таким чином, щоб зупинити перевірку правил одразу, як знайдеться правило, що виконується (stopping after first match).

Правило «is empty» використовується для рядків, масивів та буферів, що мають довжину 0, або об’єктів, які не мають властивостей. Він не передає значення null або undefined.

За замовчуванням вузол не змінює властивість msg.parts повідомлень, які є частиною послідовності.

Параметр «Recreate message sequences» можна включити для створення нових послідовностей повідомлень для кожного з правил, якому відповідає вхідне значення. У цьому режимі вузол буде заносити в буфер всю вхідну послідовність, перш ніж надсилати нові послідовності. Налаштування параметру «nodeMessageBufferMaxLength» можна використовувати для обмеження кількості буферів вузлів повідомлень.

Розглянемо приклад застосування вузла «switch». Нехай, необхідно розділити потік на три напрямки в залежності від значення вхідного параметра. Порівняння відбувається за трьома умовами: msg.payload > 5, msg.payload = 5 та msg.payload < 5.

Для рішення даної задачі необхідно створити наступну послідовність нодів (рис. 2.40).

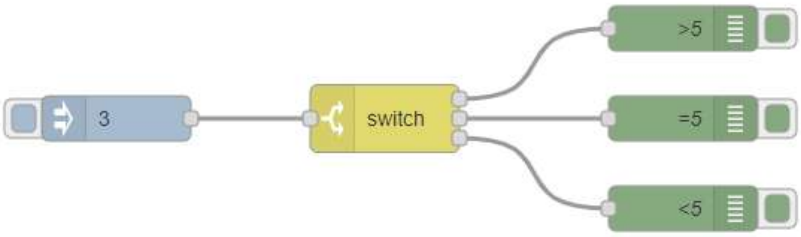


Рисунок 2.40 – Послідовність вузлів для розділення потоку

Вузол «switch» повинен мати наступні налаштування (рис. 2.41).

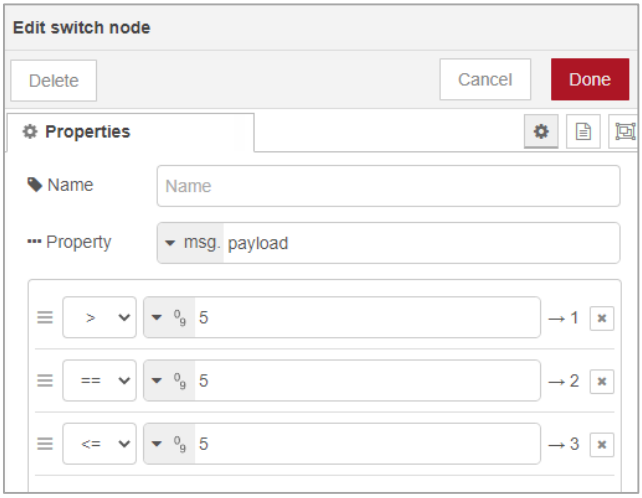


Рисунок 2.41 – Налаштування вузла «switch» для розподілення потоку

При такому налаштуванні повідомлення з корисним навантаженням менше п’яти будуть перенаправлятись на перший вихід, повідомлення з корисним навантаженням, що дорівнює п’яти будуть перенаправлятись на другий, а всі інші – на третій вихід.

2.2.7 Вузол «range»

Даний вузол співвідносить значення одного числового діапазону з іншим числовим діапазоном. Цей вузол лінійно масштабуватиме отримане значення у вихідне payload.

Існує три варіанти співвідношення:

– Scale the message property (масштабувати властивість повідомлення);

– Scale and limit to target range (масштабувати та обмежити до цільового діапазону);

– Scale and wrap within the target range (масштабувати та обгортати в межах цільового діапазону).

Опція «Масштабувати властивість повідомлення» (Scale the message property) дозволяє масштабувати вхідні дані відповідно до заданого діапазону. Ця опція змінює значення властивості повідомлення payload, що проходить через вузол, так щоб вихідне значення відповідало би заданому діапазону.

Для налаштування роботи необхідно вказати вхідний та вихідний діапазони (рис. 2.42).

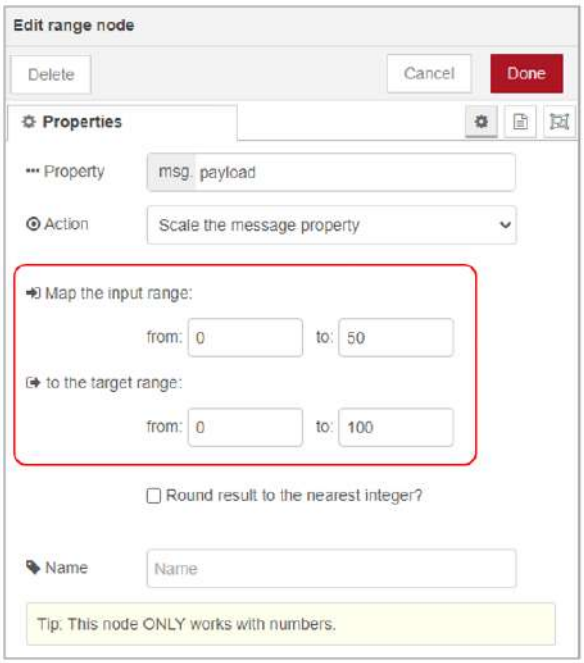


Рисунок 2.42 – Налаштування вузла Range

Наприклад, якщо діапазон встановлено від 0 до 100, а вхідні дані мають значення від 0 до 50, то опція «Scale the message property» вказує змінити значення payload так, щоб вони відповідали діапазону від 0 до 100.

У цьому випадку, якщо вхідне значення дорівнює 25, то воно буде масштабоване до 50 (рис. 2.43). Якщо вхідні значення будуть виходити за межі діапазону, то і вихідні значення також будуть пропорційно масштабуватись.

Режим «Scale the message property» може бути корисним для конвертації вхідних даних у формат, що вимагається іншими частинами системи або для зменшення розміру вхідних даних.

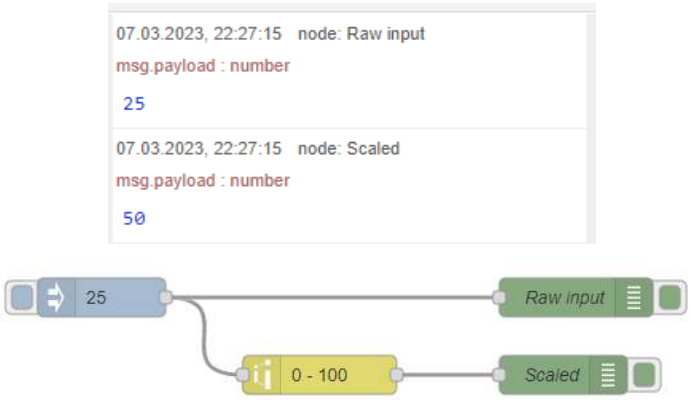


Рисунок 2.43 – Застосування режиму роботи «Scale the message property»

Опція «Масштабувати та обмежити до цільового діапазону» (Scale and limit to target range) дозволяє масштабувати вхідні дані відповідно до нового діапазону і обмежити їх значення, якщо вони перевищують максимальне або мінімальне значення цільового діапазону.

Наприклад, якщо максимальне значення вхідних даних перевищує максимальне значення цільового діапазону, або мінімальне значення вхідних даних менше мінімального значення цільового діапазону, то в цьому режимі роботи значення властивості повідомлення будуть обмежені до максимального або мінімального значення цільового діапазону.

На рис. 2.44 подано відмінність між режимами роботи Scale the message property та Scale and limit to target range. В першому випадку на виході ми отримали пропорційно збільшене значення вхідного числа, а в другому випадку – обмеження вихідного значення на рівні максимального значення вказаного вихідного діапазону (максимальне значення 100).

Опція «Масштабувати та обгортати в межах цільового діапазону» (Scale and wrap within the target range) дозволяє масштабувати вхідні дані відповідно до нового діапазону і «обгортати» їх значення, якщо вони перевищують максимальне або мінімальне значення цільового діапазону.

Наприклад, якщо вхідні дані мають значення від 0 до 50, а новий цільовий діапазон встановлено від 0 до 100, опція «Масштабувати та обгортати в межах цільового діапазону» може змінити значення властивості повідомлення так, щоб вони відповідали новому діапазону і «обгортались» в межах цього діапазону, якщо вони виходять за його межі.

Рисунок 2.44 – Відмінність між двома режимами роботи

У цьому випадку, якщо вхідне значення дорівнює 75, то воно буде масштабоване до 50, оскільки це є максимальним значенням у вхідному діапазоні (рис. 2.45).

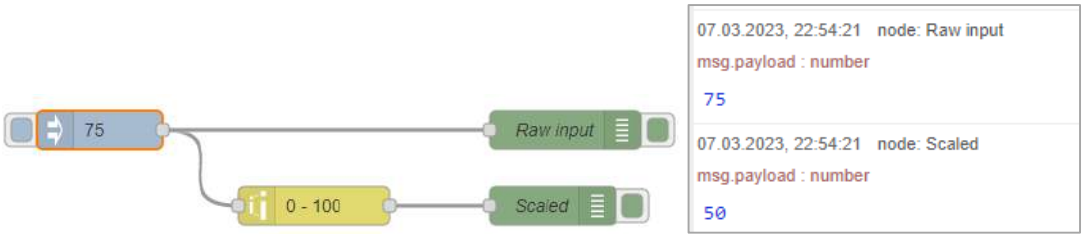


Рисунок 2.45 – Застосування режиму роботи «Scale and wrap within the target range»

Опція «Масштабувати та обгортати в межах цільового діапазону» може бути корисна для використання в системах, де значення властивостей повідомлень можуть виходити за межі певного діапазону, але при цьому вони повинні залишатися в межах цього діапазону.

2.2.8 Вузол «trigger»

Вузол «trigger» у Node-RED призначений для створення тригерів (спрацьовувачів), що дозволяє відстежувати зміни значень вхідних даних та автоматично генерувати вихідні дані.

Цей вузол можна використовувати для створення тайм-ауту в потоці. За замовчуванням, коли він отримує повідомлення, він надсилає повідомлення із payload рівним «1». Потім він чекає 250 мс, перш ніж надіслати друге

повідомлення з payload «0». Це може бути використано, наприклад, для миготіння світлодіода, прикріпленого до одного з вихідних контактів GPIO Raspberry Pi.

Дані кожного надісланого повідомлення можуть бути налаштовані на різні значення, включаючи можливість нічого не надсилати. Наприклад, якщо встановити початкове повідомлення на «nothing» та вибрати опцію продовження таймера з кожним новим повідомленням, вузол діятиме як сторожовий таймер «watchdog», відправляючи повідомлення лише в тому випадку, якщо нічого не отримано протягом встановленого інтервалу.

Є кілька режимів роботи тригеру:

– wait for – відправка двох повідомлень з вказаною затримкою;

– wait to be reset – відправка одного повідомлення і блокування пересилання повідомлення, аж до надходження повідомлення з властивістю скидання (reset);

– resend it every – відправка повідомлень з вказаною періодичністю.

Якщо вузол отримує повідомлення з властивістю reset або корисне навантаження, яке відповідає варіанту, що налаштовано у вузлі, то будь-який тайм-аут або повторення, що зараз виконується, буде очищено, і жодне повідомлення не запускатиметься.

Вузол можна налаштувати для повторного надсилання повідомлення через регулярний інтервал, доки його не буде скинуто отриманим повідомленням.

За бажанням вузол можна налаштувати для обробки повідомлень як окремих потоків, використовуючи властивість msg для ідентифікації кожного потоку. За замовчуванням – це msg.topic.

Статус вказує, що вузол активний. Якщо використовується декілька потоків, статус вказує кількість потоків, що утримуються.

Наступний приклад показує, як можна з двох тригерів зробити мультивібратор, який видає на виході з періодичністю 1 секунду «0» або «1». Скидання мультивібратора відбувається вузлом з назвою «Stop», який виставляє повідомлення з полем «reset».

Порядок поєднання вузлів мультивібратора подано на рис. 2.46. Перший вузол «inject» із назвою «Start» не містить жодної властивості. Він необхідний лише для запуску першого тригера. Другий вузол «inject» із назвою «Stop» необхідний для завершення роботи мультивібратора. Він створює повідомлення з властивістю «reset» типу Boolean, що містить значення «true».

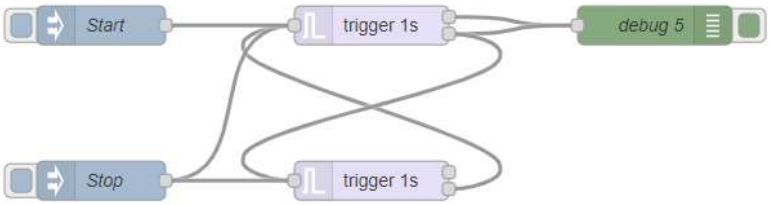


Рисунок 2.46 – Реалізація мультивібратора з вузлів «trigger»

На рис. 2.47 подано приклад налаштування другого вузла «inject».

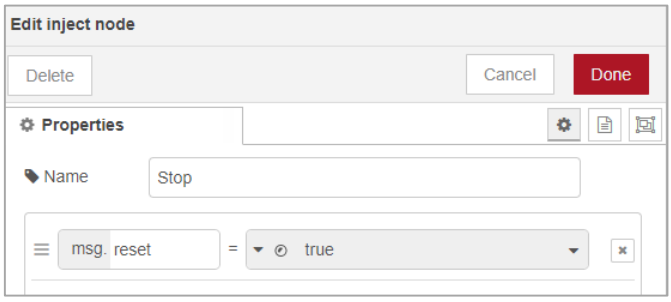


Рисунок 2.47 – Налаштування вузла «inject» із назвою «Stop»

Даний вузол підключається до входів обох тригерів для правильного завершення роботи мультивібратора.

Кожен вузол «trigger» має однакові налаштування, як подано на рис. 2.48.

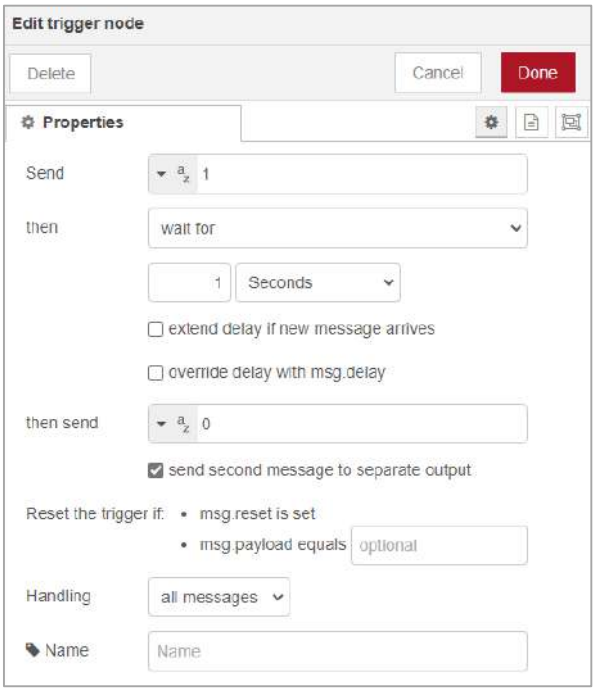


Рисунок 2.48 – Налаштування тригерів

В якості інтервалу часу обрана одна секунда та встановлена відмітка напроти властивості «Надсилати повідомлення на другий вихід вузла». Саме за допомогою другого виходу пов’язані між собою тригери. Другий вихід першого тригера підключається до входу другого тригера, а другий вихід другого тригера – до входу першого.

Для контроля за роботою мультивібратора на обидва виходи першого тригера підключено вузол «debug». При включеному мультивібраторі на виході послідовно отримуються повідомлення з вмістом «1» або «0» із затримкою в одну секунду (рис. 2.49).

Рисунок 2.49 – Результат роботи мультивібратора

2.3 Використання Context в Node-RED

2.3.1 Загальні відомості про контекст

Node-RED забезпечує спосіб зберігання інформації, якою можна ділитися між різними вузлами без використання повідомлень, які проходять через потік. Ця технологія називається «контекст».

«Область видимості» певного значення контексту визначає, з ким він взаємодіє. Існує три рівні контексту:

– context (вузол) – видимий лише для вузла, який встановив значення;

– flow (потік) – видимий для всіх вузлів одного потоку (або вкладки в редакторі);

– global (глобальний) – видимий для всіх вузлів.

Вибір області для будь-якого конкретного значення залежатиме від того, як воно використовується в потоці. Якщо до значення потрібно отримати доступ лише з одного вузла, наприклад вузла Function, тоді достатньо контексту Node. Частіше контекст дозволяє ділитися певним станом між кількома вузлами. Наприклад, датчик може регулярно публікувати нові значення в одному потоці, а ви хочете створити окремий потік, ініційований HTTP, щоб повертати останнє значення. Зберігаючи показання датчика в контексті, він стає доступним для повернення потоку HTTP.

Глобальний контекст можна попередньо налаштувати зі значеннями за допомогою властивості functionGlobalContext у файлі налаштувань.

Для вузлів у підпотоці контекст потоку спільно використовується цими вузлами, а не потоком, у якому знаходиться підпотік. Починаючи з Node-RED 0.20, вузли всередині підпотоку можуть отримати доступ до контексту батьківського потоку, додавши перед $parent. до контекстного ключа. Наприклад:

```
var color = flow.get('$parent.colour');
```

За замовчуванням контекст зберігається лише в пам'яті. Це означає, що його вміст очищається щоразу, коли Node-RED перезапускається. У випуску 0.19 можна налаштувати Node-RED для збереження даних контексту, щоб вони були доступні під час перезапуску.

Для налаштування способу зберігання даних контексту можна використовувати властивість contextStorage у settings.js. Node-RED надає для цього два вбудованих модуля: пам'ять і локальну файлову систему. Також можна створити власні плагіни сховища, щоб зберегти дані в іншому місці.

Щоб увімкнути зберігання на основі файлів, можна використати такий параметр:

```
contextStorage: {
   default: {
       module: "localfilesystem"
   }
}
```

Це встановлює за замовчуванням сховищем контексту екземпляр плагіна локальної файлової системи. Це означає, що:

– він зберігатиме контекстні дані у файлах в папці «/.node-red/context/»;

– він кешує значення в пам’яті та записує їх у файлову систему лише кожні 30 секунд.

Також можна налаштувати більше одного сховища, щоб деякі значення зберігалися в локальній файловій системі, а деякі зберігалися лише в пам’яті. Наприклад, щоб налаштувати сховище за замовчуванням лише в пам’яті, а друге сховище – для файлової системи, можна використати такі параметри:

```
contextStorage: {
   default: "memoryOnly",
   memoryOnly: { module: ‘memory’ },
   file: { module: ‘localfilesystem’ }
}
```

У цьому прикладі властивість за замовчуванням повідомляє Node-RED, яке сховище використовувати, якщо запит на доступ до контексту явно не вказує тип цього сховища.

Найпростіший спосіб записати значення в контекст – скористатися вузлом «Change». Наприклад, наступне налаштування вузла «Change» зберігатиме значення msg.payload у контексті потоку під ключем myData (рис. 2.50).

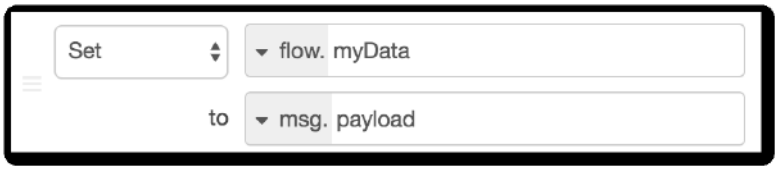


Рисунок 2.50 – Налаштування способу зберігання значення в контекст myData

Різні вузли можуть отримувати прямий доступ до контексту. Наприклад, вузол Inject можна налаштувати для введення значення з контексту, а вузол Switch може маршрутизувати повідомлення на основі значення, що зберігається в контексті.

Якщо в програмі налаштовано кілька сховищ контексту, інтерфейс користувача дозволить вам вибрати, у якому сховищі потрібно зберігати значення (рис. 2.51).

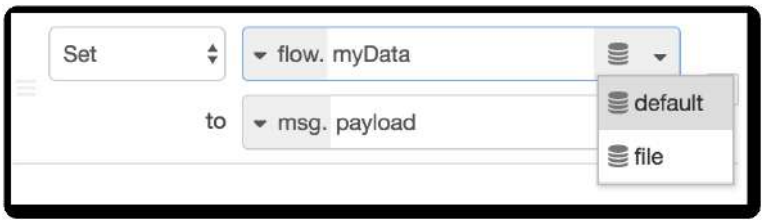


Рисунок 2.51 – Вибір варіанту зберігання контексту

Контекст можна остаточно видалити за допомогою вузла Change, налаштованого на видалення (рис. 2.52).

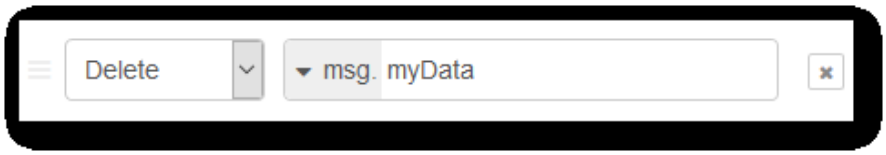


Рисунок 2.52 – Видалення контексту

2.3.2 Доступ до збережених даних із функції у сховище

Для доступу до збережених даних із функції можна скористатись наступним методом. У наступних прикладах використовується контекст потоку, але вони однаково добре застосовуються до контексту ноди та глобального.

Існує два режими доступу до контексту – синхронний або асинхронний. Вбудовані сховища контексту забезпечують обидва режими. Деякі сховища можуть надавати лише асинхронний доступ і викидають помилку, якщо доступ до них здійснюється синхронно.

Щоб отримати значення з контексту у тілі функції необхідно записати наступний код:

```
var myCount = flow.get("count");
```

Для запису даних в контекст необхідно використовувати наступний код:

```
flow.set("count", 123);
```

В лапках вказується назва контексту, а після коми – значення, що записуються в сховище.

У наступному прикладі підраховується кількість запущених функцій:

```
// initialize the counter to 0 if it doesn’t exist already
var count = context.get("count") || 0;
count += 1;
// store the value back
context.set("count", count);
// make it part of the outgoing msg object
msg.count = count;
return msg;
```

В даному прикладі використовується тип контексту «context», який означає, що дані зберігаються та будуть доступні лише даному вузлу, в якому вони ініційовані.

Починаючи з Node-RED 0.19, також можна отримати або встановити кілька значень за один раз:

```
flow.set(["count", "colour", "temperature"], [123, "red", "12.5"]);
…
var values = flow.get(["count", "colour", "temperature"]);
// values[0] is the ‘count’ value
// values[1] is the ‘colour’ value
// values[2] is the ‘temperature’ value
```

У цьому випадку будь-які відсутні значення встановлюються як нуль.

Якщо сховище контексту потребує асинхронного доступу, функції get і set потребують додаткового параметра зворотного виклику (рис. 2.53).

```
// Get single value
flow.get("count", function(err, myCount) { ... });

// Get multiple values
flow.get(["count", "colour"], function(err, count, colour) { ... })

// Set single value
flow.set("count", 123, function(err) { ... })

// Set multiple values
flow.set(["count", "colour", [123, "red"], function(err) { ... })
```

Рисунок 2.53 – Асинхронний доступ до контексту

В даному прикладі, перший аргумент «err», який передається зворотному виклику, встановлюється лише, якщо сталася помилка під час доступу до контексту.

Асинхронна версія прикладу підрахунку кількості запущених функцій подана на рис. 2.54.

```
context.get('count', function(err, count) {
    if (err) {
        node.error(err, msg);
    } else {
        // initialise the counter to 0 if it doesn't exist already
        count = count || 0;
        count += 1;
        // store the value back
        context.set('count',count, function(err) {
            if (err) {
                node.error(err, msg);
            } else {
                // make it part of the outgoing msg object
                msg.count = count;
                // send the message
                node.send(msg);
            }
        });
    }
});
```

Рисунок 2.54 – Приклад коду для підрахунку кількості запущених функцій з використанням асинхронного доступу

2.3.3 Приклад застосовування контексту для побудови циклічного автогенератора

Розглянемо приклад створення автогенератора з можливістю налаштування діапазону вихідних значень. Генератор має виконувати наступні функції:

– скидання в нуль, або на початок діапазону циклу;

– послідовна генерація чисел в межах вказаного діапазону;

– автоматичне скидання на мінімальне значення діапазону при досягненні максимального значення.

Реалізація даного генератора буде здійснювались за допомогою контексту.

Функціональна схема автогенератора подана на рис. 2.55. Робота автогенератора побудована з використанням контексту. Контекст виконує роль міжвузлового сховища, що забезпечує доступ до даних із різних потоків.

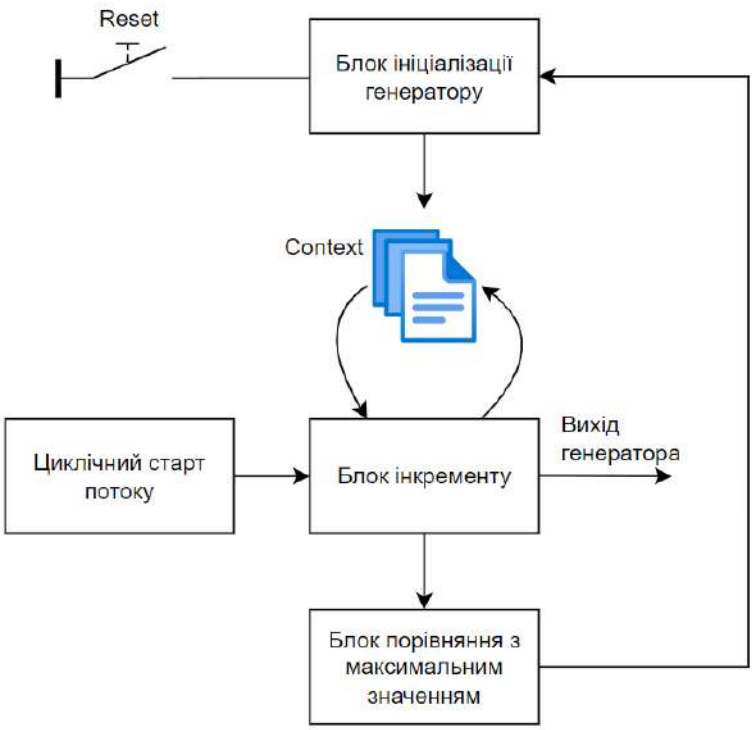


Рисунок 2.55 – Функціональна схема автогенератора

Блок ініціалізації генератора створює змінну Count та розміщує її в контексті для сумісного доступу до неї з інших потоків. В цьому блоці також виконується попередній запис в змінну Count нульового значення, що вважається скиданням на початок припустимого діапазону циклу.

Основний потік починається з циклічного генератору, який запускає блок інкременту змінної Count. Сама змінна зберігається в контексті, що дає можливість запам’ятовувати її останнє значення і збільшувати його в кожному новому циклі.

Блок порівняння з максимальним значенням реалізує функцію автоматичного скидання лічильника на мінімальне значення діапазону циклу при досягненні максимального значення. З цього блоку викликається блок початкової ініціалізації при кожному досягненні максимально припустимого значення.

Для створення блоку ініціалізації в Node-RED використовуватиме вузли «inject» та «change» (рис. 2.56).

Рисунок 2.56 – Блок ініціалізації автогенератора

В налаштуваннях вузла «inject» потрібно прибрати всі властивості та встановити checkbox «Inject once after» і вказати час 1 секунда (рис. 2.57).

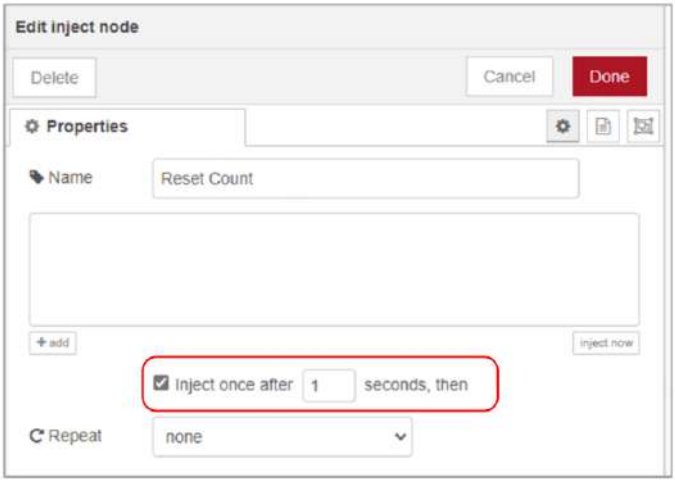


Рисунок 2.57 – Налаштування властивостей вузла «inject»

Включення функції «Inject once after» надає можливість ініціювати новий потік кожен раз, коли оновлюється сторінка, або зініціюється глобальний потік повідомлень. Для нашої задачі це означає, що на старті програми через одну секунду буде гарантовано викликано функцію скидання автогенератора в нульове значення.

Сама функція скидання автогенератора реалізована на елементі «change». Приклад налаштування вузла подано на рис. 2.58.

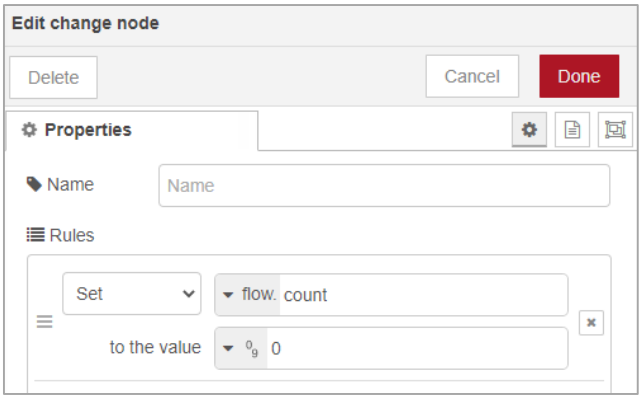


Рисунок 2.58 – Приклад налаштування вузла«change» функції скидання автогенератора

В даному вузлі обирається функція «Set» та в якості цілі встановлюється контекст потоку, якому призначається зміна «count». Разом з тим записується

початкове значення в цю змінну, що дорівнює «0». Таким чином, якщо перейти в цей вузол, створюється нова змінна «count», якій присвоюється значення 0.

Основний потік генератора подано на рис. 2.59.

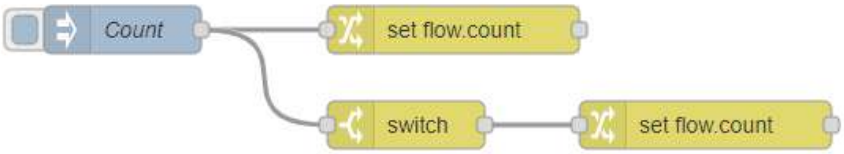


Рисунок 2.59 – Основний потік генератора

Даний потік складається з вузлів «inject», «change» та «switch». Перший вузол «inject» необхідно налаштувати в режим автогенерації повідомлень з інтервалом в 1 секунду. Приклад налаштування подано на рис. 2.60.

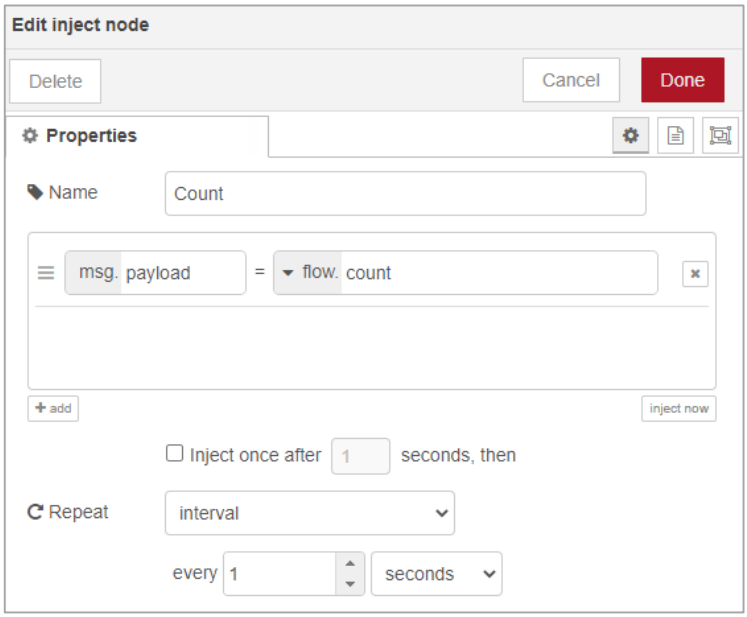


Рисунок 2.60 – Приклад налаштування вузла «inject» основного потоку

В якості корисного навантаження повідомлення передається вміст змінної контексту «count».

Функція автоінкременту реалізована у вузлі «change». На рис. 2.61 подано приклад застосування виразу в якості аргументу команди «Set».

В якості виразу вказана наступна формула:

```
$flowContext("count") + 1.
```

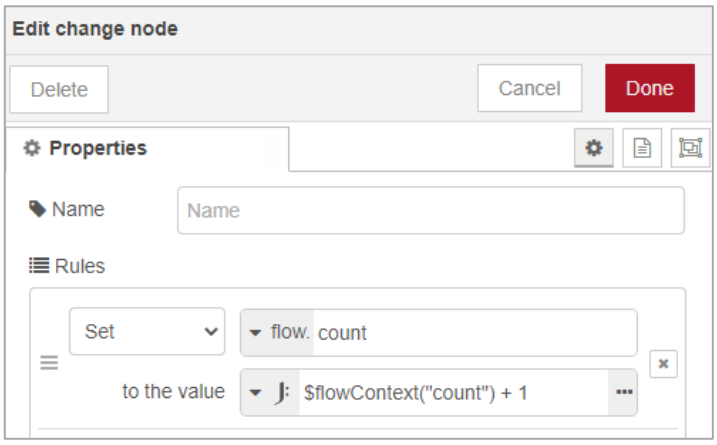


Рисунок 2.61 – Налаштування вузла «change» основного потоку

В цій формулі за допомогою вбудованій функції `$flowContext` з контексту потоку зчитуються значення змінної «count» та потім збільшується на одиницю. Отримане нове значення знову заноситься в зміну «count» та зберігається в контексті.

Додаткова функція порівняння змінної «count» з максимальним значенням реалізує функцію автоматичного скидання лічильника на мінімальне значення діапазону циклу при досягненні максимального значення реалізована на елементах «switch» та «change».

Вузол «switch» перевіряє вміст змінної «count» із заданим максимальним значенням, наприклад 10, та переводить потік до наступного вузла (рис. 2.62).

Наступний вузол реалізує функцію скидання за принципом, що був описаний вище. В змінну «count» записується значення «0» та зберігається в контекст потоку.

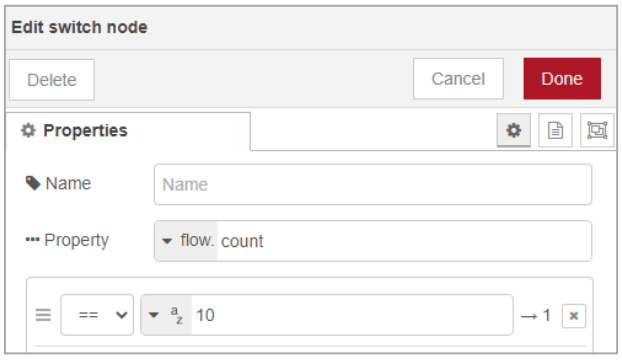


Рисунок 2.62 – Налаштування вузла «switch»

Додатковий потік реалізує функцію моніторингу стану автогенератора. До його складу входять два вузли: «inject» та «debug» (рис. 2.63).

Рисунок 2.63 – Потік моніторингу стану автогенератора

Вузол «inject» налаштований в режимі автоматичної генерації повідомлень з інтервалом 1 секунда (рис. 2.64).

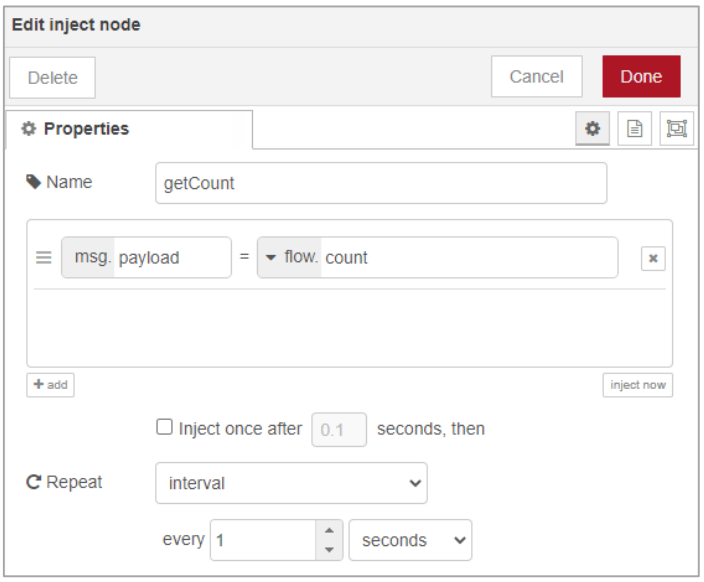


Рисунок 2.64 – Налаштування вузлу «inject» інформаційного потоку

В якості корисного навантаження використовується значення змінної «count», що зчитується з контексту потоку. Вузол «debug» виводить вміст змінної «count» у вікно налагодження (рис. 2.65).

Рисунок 2.65 – Виведення вмісту змінної «count» у вікно налагодження

2.3.4 Створення toggle-trigger

Перекидні тригери (toggle-trigger) є основними компонентами цифрових лічильників, і всі пристрої типу D адаптовані для такого використання.

Коли електронний лічильник використовується для підрахунку, фактично підраховуються імпульси, що з'являються на вході CK, які можуть бути або регулярними імпульсами, отриманими від внутрішнього годинника, або вони можуть бути нерегулярними імпульсами, створеними якоюсь зовнішньою подією.

Коли перемикаючий тригер використовується як один з етапів лічильника, його вихід Q змінюється на протилежний стан (перемикається на високий або низький рівень) під час кожного тактового імпульсу.

Більшість тригерів із перемиканням можна використовувати як тригери типу D, які можна перетворити на тригери з перемиканням простою модифікацією. Теоретично все, що необхідно для перетворення типу D, що запускається фронтом, у тип T, це підключити вихід Q безпосередньо до входу D, як подано на рис. 2.66.

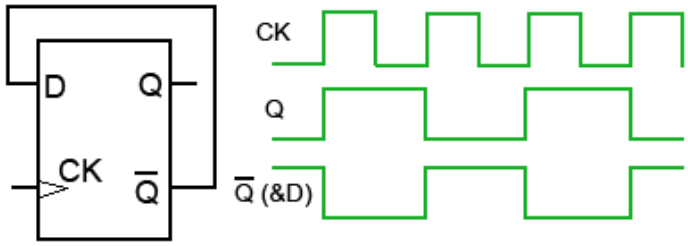


Рисунок 2.66 – Умовне позначення та часова діаграма роботи перекидного тригеру

Фактично, входом такого тригеру є його імпульсний вхід CK. Вплив цього режиму роботи тригеру також показано на часовій діаграмі на рис. 2.66.

Для реалізації такого тригеру в NodeRED можна застосувати вузол «function» та технологію контексту.

Фрагмент коду вузла «function» показано нижче:

```
var currentState = context.get('state') || "on";
if (currentState == "on") {
    context.set("state", "off");
    msg.payload = "off";
```

```
} else if (currentState == "off") {
    context.set("state", "on");
    msg.payload = "on";
}
else {
    return;
}
return msg;
```

В даному фрагменті в першому рядку з контексту в змінну `currentState` записується інформація про минулий стан тригеру. Якщо ми запустили функцію перший раз і даної змінної ще не існує, то вона створюється та в неї записується початкове значення «on».

Стан тригеру визначається двома станами «on» та «off». Основна задача даної функції – перевірка поточного значення в змінній `currentState` та заміна його на протилежній.

Окрім збереження поточного значення в контекст воно також записується в корисне навантаження повідомлення, наприклад: `msg.payload = "off"`.

Вміст функціонального блока подано на рис. 2.67.

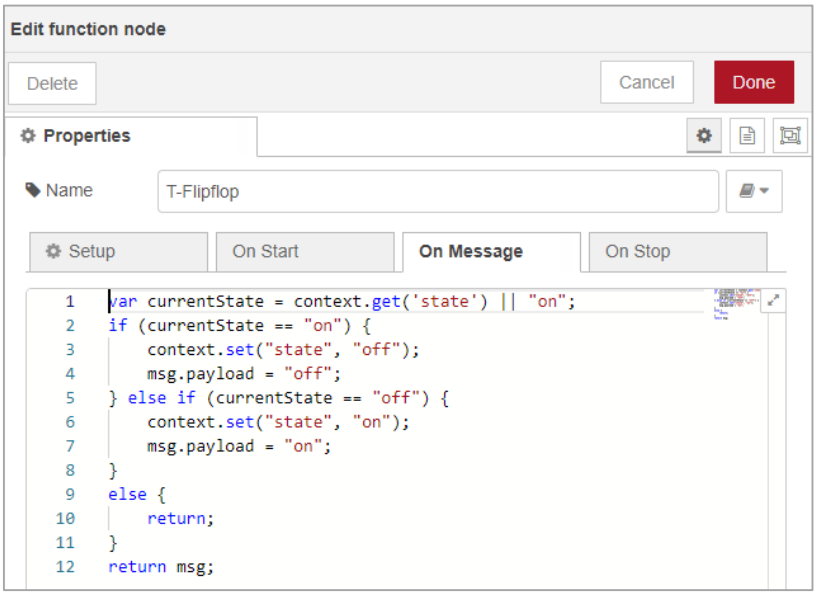


Рисунок 2.67 – Вміст функціонального блока

Для демонстрації роботи перекидного тригеру створимо простий потік з трьома вузлами: inject, function, debug (рис. 2.68).

Рисунок 2.68 – Приклад використання перекидного тригеру

Вміст вузла «inject» подано на рис. 2.69.

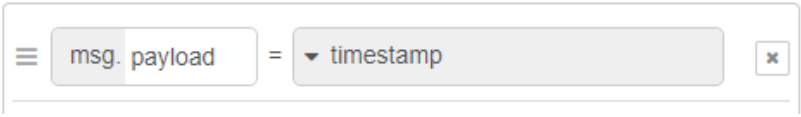


Рисунок 2.69 – Вміст вузла «inject»

Призначення цього вузла – ініціалізація потоку. В якості повідомлення може виступати будь-яка послідовність, наприклад, timestamp. Отримання цього повідомлення функціональним вузлом запускає вбудовану функцію зміни поточного стану змінної «state» в контексті.

За допомогою вузла «debug» можна дослідити стан змінної «state» з кожним натисканням на вузол «inject» (рис. 2.70).

Рисунок 2.70 – Дослідження стану змінної «state»

З наведеного рисунку можна бачити реакцію потоку на кожну ініціалізацію потоку. Кожен раз ми бачимо зміну поточного значення з «on» на «off».

2.4 Контрольні запитання та завдання

1. Які основні вузли використовуються в Node-RED і яка їх класифікація?
2. Опишіть структуру повідомлення у Node-RED.
3. Як використовується команда «Inject» у Node-RED?
4. Яке призначення вузла «function» у Node-RED та як його налаштувати?
5. Як використовувати вузол «change» для зміни властивостей повідомлення?
6. Як працює вузол «switch» і для чого він використовується?
7. Що робить вузол «range» і як його налаштувати?
8. Яка функція вузла «trigger» і як його використовувати?
9. Що таке контекст у Node-RED і як отримати доступ до збережених даних у сховищі з функції?

3 ВИКОРИСТАННЯ NODE-RED ДЛЯ УПРАВЛІННЯ ЗАСОБАМИ АВТОМАТИЗАЦІЇ

3.1 Протокол Modbus. Базові поняття

Протокол Modbus та мережа Modbus [Modbus – Modicon] є найпоширенішими у світі. Незважаючи на свій вік (стандартом Modbus став ще у 1979 році), Modbus не лише не застарів, але, навпаки, суттєво зросла кількість нових розробок та обсяг організаційної підтримки цього протоколу. Мільйони Modbus-пристроїв у всьому світі продовжують успішно працювати.

Однією з переваг Modbus є відсутність потреби у спеціальних інтерфейсних контролерах (Profibus та CAN вимагають для своєї реалізації замовні мікросхеми), простота програмної реалізації та елегантність принципів функціонування. Все це знижує витрати на опанування стандарту системними інтеграторами та розробниками контролерного обладнання.

Основним недоліком Modbus є мережевий обмін за типом «провідний/підлеглий», що не дозволяє підлеглим пристроям передавати дані в міру їх появи і тому потребує інтенсивного опитування провідними пристроями.

Різновидами Modbus є протоколи Modbus Plus – багатомайстерний протокол із кільцевою передачею маркера та Modbus TCP, розрахований на використання в мережах Ethernet та інтернет.

Протокол Modbus має два режими передачі: RTU (Remote Terminal Unit – «віддалений термінальний пристрій») та ASCII. Стандарт передбачає, що режим RTU в протоколі Modbus повинен бути обов’язково, а режим ASCII є опціонним. Користувач може вибирати будь-який з них, але всі модулі, включені в мережу Modbus, повинні мати той самий режим передачі.

Modbus RTU допускає один активний (ініціатор) пристрій в лінії (master), який може передавати команди одному або декільком пасивним пристроям (slave), звертаючись до них за унікальною в лінії адресою. Синтаксис команд протоколу дозволяє адресувати до 255 пристроїв на одній лінії зв’язку стандарту RS-232 або RS-485.

Ініціатива проведення обміну завжди виходить від активного (master) пристрою, наприклад ПК. Пасивні пристрої прослуховують лінію зв’язку.

Активний пристрій надсилає запит (посилка, послідовність байт) в лінію і переходить в стан прослуховування лінії зв'язку. Пасивний пристрій відповідає на запит, який прийшов на його адресу. Активний пристрій має встановлювати інтервал часу на приймання посилки даних. Якщо цей інтервал перевищив заданий час то генерується таймаут приймання даних.

Протокол забезпечує взаємодію у режимі Request/Response. Майстер ініціює запит до підлеглого пристрою, передаючи в PDU код функції та дані. Залежно від фізичного рівня мережі в пакеті можуть бути додаткові поля.

На рис. 3.1 показано принцип роботи протоколу Modbus у випадку відсутності помилок на підлеглому пристрої. Якщо обробка запита проходить без помилок, підлеглий пристрій повертає пакет, що містить вихідний код функції та запитані дані.

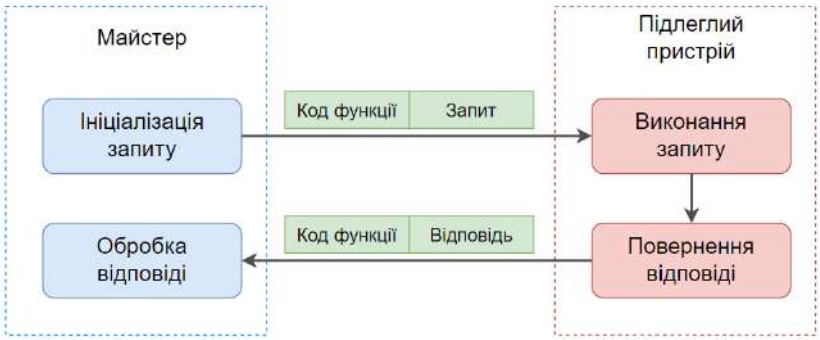


Рисунок 3.1 – Принцип роботи протоколу Modbus у випадку відсутності помилок на підлеглому пристрої

При виникненні помилки підлеглий пристрій повертає в якості даних код помилки, а замість вихідного коду функції – значення цієї помилки, збільшене на 128 (0x80 в шістнадцятковій системі HEX) (рис. 3.2).

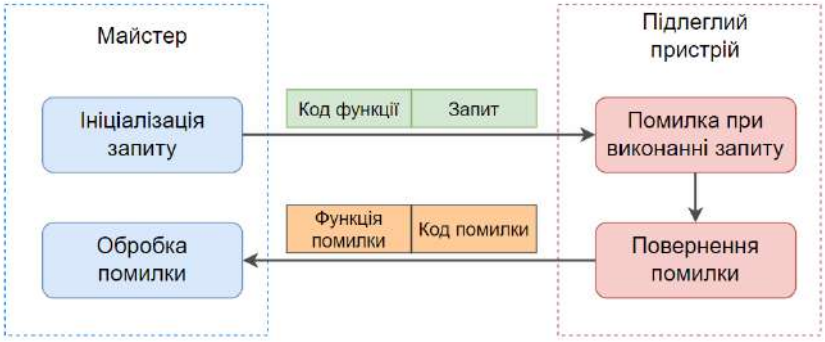


Рисунок 3.2 – Схема роботи Modbus у випадку помилок на підлеглому пристрої

Також передбачені тайм-аути на стороні майстра, щоб уникнути тривалого очікування відповіді від пристроїв, що вийшли з ладу.

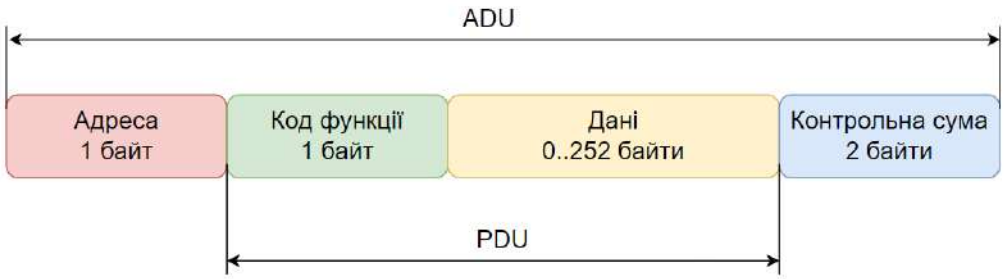
3.2 Організація обміну даними за протоколом Modbus

Протокол Modbus передбачає, що тільки один провідний пристрій (контролер) і до 247 підлеглих (модулів вводу-виводу) можуть бути об'єднані в промислову мережу. Обмін даними завжди ініціюється провідним. Підлеглі пристрої ніколи не починають передачу даних, доки не отримають запит від провідного. Підлеглі пристрої не можуть обмінюватися даними один з одним. Тому будь-якої миті часу в мережі Modbus може відбуватися лише один акт обміну [15].

Адреси з 1 до 247 є адресами Modbus пристроїв у мережі, а з 248 до 255 зарезервовані. Ведучий пристрій не повинен мати адреси та в мережі не повинно бути двох пристроїв з однаковими адресами.

Провідний пристрій може надсилати запити всім пристроям одночасно (широкомовний режим) або тільки одному. Для широкомовного режиму зарезервовано адресу «0» (при використанні команди цієї адреси вона приймається всіма пристроями мережі).

У протоколі Modbus RTU повідомлення починає сприйматися як нове після паузи (тиші) на шині тривалістю щонайменше 3,5 символів (14 біт), тобто величина паузи за секунди залежить від швидкості передачі. На рис. 3.3 показано формат кадру протоколу Modbus.



PDU – Protocol Data Unit (елемент даних протоколу);
ADU – Application Data Unit (елемент даних додатка)

Рисунок 3.3 – Формат кадру Modbus

Поле адреси завжди містить тільки адресу пристрою, навіть у відповідях на команду, надіслану майстром. Завдяки цьому провідний пристрій знає, від якого модуля надійшла відповідь.

Поле «Адреса серверного пристрою» (Additional address) визначає, за якою адресою слід надіслати запит клієнта. Може приймати значення від 1 до 247. Адреса 0 використовується для широкомовної передачі даних від клієнта всім серверним пристроям (відповідь сервера при цьому не передбачена), а адреси 248…255 вважаються зарезервованими. У деяких реалізаціях протоколу поле ігнорується – наприклад, Modbus TCP, де найчастіше застосовується стандартна IP-адресація.

Поле «Код функції» говорить модулю про те, яку дію потрібно виконати. Це поле визначає, яку дію необхідно виконати на серверному пристрої. Значення кодів функцій лежать від 1 до 255, причому коди від 128 до 255 зарезервовані для повідомлень про помилки. Код 0 не використовується.

Для кодів з діапазонів 65-72 та 100-110 користувачі можуть реалізувати власні функції (User-Defined Function Codes). Деякі коди, наприклад 9, 10, 13 та інші, зарезервовані певними постачальниками для обладнання та закриті для загального використання (Reserved Function Codes). Коди, що не входять до цих двох підмножин, відносяться до публічних (Public Function Codes) – це задокументовані функції, що знаходяться у відкритому доступі.

Поле «Дані» може містити довільну кількість байт. У ньому може бути інформація про параметри, що використовуються в запитах контролера або відповіді модуля. Дані, необхідні для виконання вибраної функції серверного пристрою. Найчастіше це адреси регістрів для читання чи запису, їх кількість тощо. Довжина та формат поля залежать від коду функції. Деякі функції не потребують передачі даних.

Поле "Контрольна сума" містить контрольну суму CRC довжиною 2 байти. Дане поле містить розраховане з допомогою спеціального алгоритму число перевірки цілісності пакета. Як алгоритм для розрахунків використовується CRC-16 або LRC-8. У деяких реалізаціях протоколу поле відсутнє – наприклад, Modbus TCP, де контроль цілісності пакета забезпечується засобами протоколу TCP/IP.

3.3 Різновиди протоколу Modbus

Modbus – це протокол прикладного (сьомого) рівня моделі OSI. Він не залежить від нижчих рівнів і може використовуватися спільно з іншими протоколами, наприклад Ethernet TCP/IP або UDP/IP, а як фізичне середовище для передачі сигналів застосовувати послідовні інтерфейси RS-232, RS-422, RS-485, оптоволокно, радіоканали та інше (рис. 3.4).

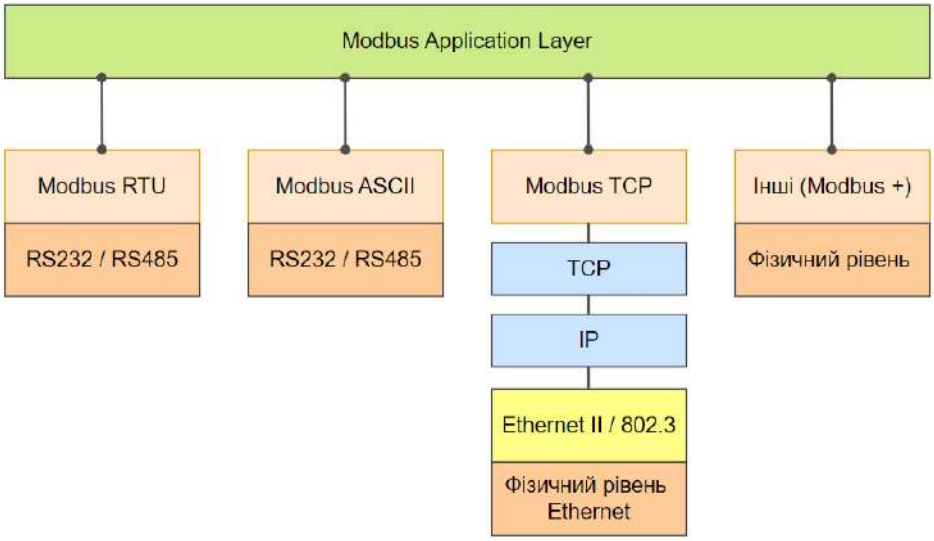


Рисунок 3.4 – Різновиди протоколу Modbus

3.3.1 Modbus RTU (Remote Terminal Unit)

Modbus RTU (Remote Terminal Unit) – це різновид протоколу, який як фізичний рівень мережі найчастіше використовує послідовний інтерфейс RS-485, рідше RS-232 і RS-422. По суті, всі ці інтерфейси визначають зв'язок за допомогою кручених пар, але відрізняються характеристиками виду максимальної довжини кабелю, кількості вузлів і таке інше.

Формат пакета Modbus RTU загалом збігається з узагальненою формою, описаною раніше: додаткові поля не використовуються. Контроль цілісності пакетів здійснюється за допомогою алгоритму CRC-16.

За замовчуванням в RTU режимі біт паритету встановлюють рівним «1», якщо кількість двійкових одиниць у байті непарне, і рівним «0», якщо воно парне. Такий паритет називають парним (even parity) та метод контролю називають контролем парності (рис. 3.5).

Рисунок 3.5 – Послідовність бітів у режимі RTU

За відсутності біта паритету на його місце записується другий стоп-біт. При парній кількості двійкових одиниць у байті, біт паритету може дорівнювати «1». У цьому випадку кажуть, що паритет є непарним (odd parity).

Контроль парності може бути відсутнім взагалі. У цьому випадку замість біту паритету має використовуватися другий стоповий біт. Для забезпечення максимальної сумісності з іншими продуктами рекомендується використовувати заміну біта паритету на другий стоповий біт.

Підлеглі пристрої можуть сприймати будь-який з варіантів: парний, непарний паритет або його відсутність.

Повідомлення Modbus RTU передаються у вигляді кадрів, кожному з яких відомо початок і кінець. Ознакою початку кадру є пауза (тиша) тривалістю щонайменше 3,5 шістнадцяткових символів (14 біт). Кадр повинен передаватися безперервно. Якщо в процесі передачі кадру виявляється пауза тривалістю більше 1,5 шістнадцяткових символів (6 біт), то вважається, що кадр містить помилку і повинен бути відхилений приймаючим модулем. Ці величини пауз повинні суворо дотримуватися при швидкостях нижче 19200 біт/с, проте для більш високих швидкостей рекомендується використовувати фіксовані паузи, 1,75 мс і 750 мкс відповідно. Приклад пакету даних в форматі Modbus RTU подано на рис. 3.6.

Рисунок 3.6 – Приклад пакету даних в форматі Modbus RTU

У режимі RTU є два рівні контролю помилок у повідомленні:

– контроль паритету для кожного байту (опційно);

– контроль кадру загалом з допомогою CRC методу.

Метод CRC використовується незалежно від перевірки паритету. Значення CRC встановлюється у провідному пристрої перед передачею. Коли повідомлення приймається, обчислюється CRC всього повідомлення і порівнюється з його значенням, зазначеним у полі CRC кадру. Якщо обидва значення збігаються, вважається, що повідомлення не містить помилки.

Стартові, стопові біти та біт паритету у обчисленні CRC не беруть участі.

3.3.2 Modbus ASCII

Modbus ASCII – це різновид протоколу Modbus, що також працює поверх інтерфейсів RS-232/RS-485, але для кодування повідомлень використовує ASCII-символи.

У порівнянні з Modbus RTU у форматі пакета додаються ще два поля – спеціальні символи для позначки початку та кінця повідомлення: двокрапка та символи повернення каретки / переведення рядка (рис. 3.7). Часові паузи між пакетами не потрібні. Для перевірки цілісності використовується алгоритм LRC-8.

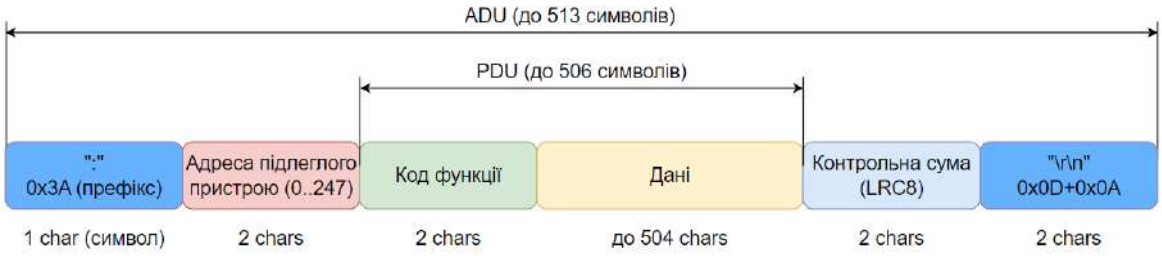


Рисунок 3.7 – Структура пакета даних в Modbus ASCII

Загалом цей варіант протоколу зараз використовується вкрай рідко – через складності кодування та великий розмір повідомлень. Однак він може стати гарною альтернативою Modbus RTU на лініях з мережними затримками та устаткуванні з менш точними таймерами.

3.3.3 Modbus TCP

Modbus TCP – це реалізація ModBus у мережах Ethernet, що працює поверх TCP/IP стека. На відміну від Modbus RTU і ASCII, Modbus TCP з’єднання

встановлюється з конкретним пристроєм засобами TCP/IP. Тому адреса в пакеті Modbus найчастіше ігнорується, а широкомовне розсилання повідомлень не використовується. Однак адреса може бути потрібна, якщо з'єднання встановлюється зі шлюзом, який, у свою чергу, виводить на мережу RS-485, щоб далі спілкуватися з пристроями вже мовою Modbus.

Контроль цілісності пакетів також забезпечується засобами протоколу TCP/IP, тому немає необхідності його Modbus-реалізації.

Поряд з адресою в заголовку пакета Modbus TCP є ряд додаткових полів (рис. 3.8):

– ID транзакції (або ID обміну). Це поле найчастіше заповнюється нулями. Поле необхідно для випадків, коли клієнтський пристрій надсилає кілька повідомлень, не чекаючи відповіді на попередні, щоб потім зв'язати відповіді із запитами;

– ID протоколу. Поле завжди заповнюється нулями, зарезервовано для майбутнього використання;

– довжина залишку пакета. Довжина частини пакета, що залишилася: адреси і PDU (коду функції і даних).

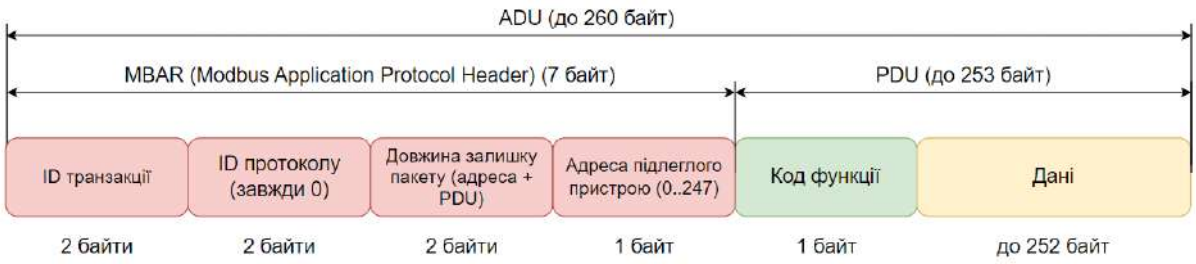


Рисунок 3.8 – Структура пакету протоколу Modbus TCP

3.4 Регістри та функції протоколу Modbus

3.4.1 Стандартна адресація регістрів Modbus

Оскільки Modbus призначений для роботи з промисловою автоматикою, обмін даними з Modbus-пристроями відбувається через регістри. Вони діляться на входи та виходи. Входи можна лише читати, а виходи – читати та писати. Бувають 1-бітні регістри Modbus для опису дискретних входів/виходів (Discrete Inputs та Coils) та 16-бітові регістри для аналогових входів/виходів (Input Registers та Holding Registers).

Доступ до регістрів здійснюється за допомогою 16-бітної адреси. Першому елементу в кожній групі регістрів відповідає адреса 0. Тобто адреса будь-якого регістру може приймати значення діапазону 0-65535 (0x0000-0xFFFF в HEX-форматі). При цьому специфікація протоколу не визначає, що фізично представляють собою адресні простори і за якими внутрішніми адресами пристрою повинні бути доступні регістри. У випадку значення регістрів з однаковою адресою, але різними типами відрізняються один від одного.

В таблиці 3.1 наведено стандартну адресацію регістрів Modbus.

Зазвичай, дискретні виходи починаються з адреси (осередка) 00001, а дискретні входи – з адреси 10001. Кожному з них потрібно один біт пам'яті. Вміст вхідних регістрів (в термінології контролерів вони іменувалися contacts) можна тільки читати, в той час як вміст вихідних регістрів (в термінології ПЛК вони іменувалися coils) можна і читати, і записувати.

Таблиця 3.1 – Таблиця розподілення регістрів Modbus

Адреса регістрів	Опис
00001 - 10000	1-бітні дискретні виходи (читання / запис) «Coils»
10001 - 20000	1-бітні дискретні входи (читання) «Discrete Inputs»
30001 - 40000	16-бітові аналогові входи (читання) «Input Registers»
40001 - 50000	16-бітові регістри зберігання (читання / запис) «Holding Registers»

Регістри аналогових входів і входів є 16-розрядними. Їх адреси починаються з 30001 – це адреса першого аналогового входу (тільки читання, наприклад, для введення сигналів від датчика температури).

З адреси 40001 починається діапазон універсальних регістрів (читання і запис), які можуть служити також і аналоговими виходами.

Залежно від фірми-виробника ПЛК ці регістри можуть бути внутрішніми регістрами, аналоговими входами, аналоговими виходами і навіть дискретними входами і виходами. Однак не всі функції працюють з адресами цих регістрів.

Перелік основних функцій протоколу ModBus наведено у таблиці 3.2.

Незважаючи на те що кодам функцій відведений діапазон від 1 до 127, в якості призначених для загального користування кодів визначені приблизно 20 кодів. В цей же діапазон входять коди, призначення яких визначається

кожним окремим користувачем. Слід мати на увазі, що багато Modbus-пристроїв підтримують тільки невеликі підмножини наявних кодів.

Таблиця 3.2 – Стандартні функції протоколу ModBus

Код	Розрядність	Опис	Діапазон адрес входів-виходів
1	1	Read coils Читання поточного стану (ON / OFF) дискретних виходів	00001 - 10000
2	1	Read contacts Читання поточного стану (ON / OFF) дискретних входів	10001 - 20000
5	1	Write a single coil Зміна стану дискретного виходу в ON або OFF	00001 - 10000
15	1	Write multiple coils Зміна стану (ON / OFF) декількох дискретних виходів	00001 - 10000
3	16	Read holding registers Читання регістрів зберігання	40001 - 50000
4	16	Read input registers Читання вхідних регістрів	30001 - 40000
6	16	Write single register Запис одного регістра	40001 - 50000
16	16	Write multiple registers Запис декількох регістрів	40001 - 50000
22	16	Mask write register Маскований запис регістра	40001 - 50000
23	16	Read/write multiple registers Читання / запис декількох регістрів	40001 - 50000
24	16	Read FIFO queue Читання вмісту черги FIFO	40001 - 50000

3.4.2 Опис функції читання вихідних контактів (дискретних виходів) (01)

Функція дозволяє користувачеві отримати статус вихідних контактів. Широкомовний режим не підтримується.

Крім полів адреси та функції, повідомлення вимагає, щоб інформаційне поле містило логічну адресу початкового біту і кількість бітів, статус яких необхідно отримати.

Адресація дозволяє отримати за один запит до 2000 логічних осередків. Однак, деякі прилади мають обмеження на максимальне число бітів, статус яких можна отримати за один запит. Біти нумеруються з нуля (біт 1 = 0, біт 2 = 1 і т.д.).

Припустимо, нам потрібно звернутися до підлеглого пристрою з адресою 11 та прочитати 19 його Coil-регістрів з номерами 20-38. Адресація дискретних виходів починається з 0, тому адреса першого потрібного нам виходу буде 0x13 (це 19 у HEX-системі). Необхідна для читання кількість виходів також дорівнюватиме 0x13 (для читання запитано 19 виходів). Адреса пристрою 0x11 (число 17 в HEX-форматі) та код функції 01. Контрольна сума формується за алгоритмом CRC-16 на основі інших полів пакета.

У таблиці 3.3 представлений запит на читання дискретних виходів 0020-0038 з приладу з адресою 17 (0x11). У таблиці 3.4 представлений приклад відповіді.

Таблиця 3.3 – Приклад запита для читання дискретних виходів 0020-0038

Байт	Опис полів запита
11	Адреса підлеглого пристрою
01	Код функції
00	Адреса початкового біта, Hi байт
13	Адреса початкового біта, Lo байт
00	Кількість дискретних виходів, Hi байт
13	Кількість дискретних виходів, Lo байт
8E	Контрольна сума CRC
92	Контрольна сума CRC

Таблиця 3.4 – Приклад повідомлення у відповідь

Байт	Опис полів відповіді
11	Адреса підлеглого пристрою
01	Код функції
03	Кількість байт в полі даних
A3	Статус вихідних контактів, перший байт
24	Статус вихідних контактів, другий байт
07	Статус вихідних контактів, третій байт
94	Контрольна сума CRC
3E	Контрольна сума CRC

На рис. 3.9 подано приклад обміну повідомленнями при використанні функції 01 (читання дискретних виходів).

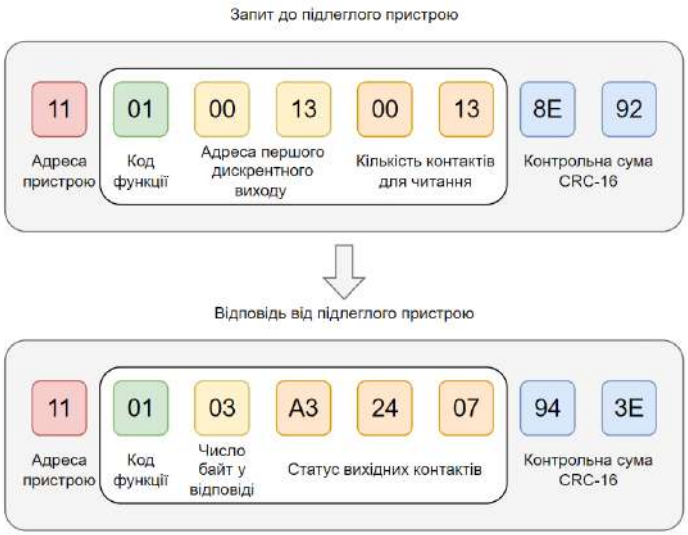


Рисунок 3.9 – Приклад обміну повідомленнями при використанні функції 01 (читання дискретних виходів)

Відповідь містить адресу, код функції, число байт в полі даних, дані і контрольну суму. Дані в полі даних у відповіді представлені у вигляді один біт на кожен осередок.

Молодший значущий біт першого байту поля даних містить перший осередок, за яким слідують інші. Якщо число осередків не ділиться на 8, то інші біти заповнюються нулями в порядку від старших бітів до молодших.

Статус осередків 20-27 дорівнює 0xA3 = 1010 0011. Читаючи зліва направо, бачимо, що осередки 27, 25, 21 і 20 встановлені. Інші дані розбираються так само. Так як було запитано число регістрів, що не ділиться на 8, старші п’ять бітів в останньому байті даних (0x07) заповнені нулями.

Принцип побудови послідовності відповідних байтів у зворотному повідомленні подано на рис. 3.10.

На даному прикладі показано випадок, коли робиться запит на читання десяти вхідних контактів, починаючи з третього входу (адреса входу DI2). Область, що підлягає зчитуванню виділена синім в модулі введення дискретних сигналів.

Нижче на рис. 3.10 подано принцип збирання бітів у байти відповіді. Молодші біти переходять до правої частини результуючого байту, а старші – до

лівої. Якщо запит прийшов на кількість бітів, що менша за кількість бітів в байті, біти, що залишаються, заповнюються нулями (другий байт відповіді 0x02).

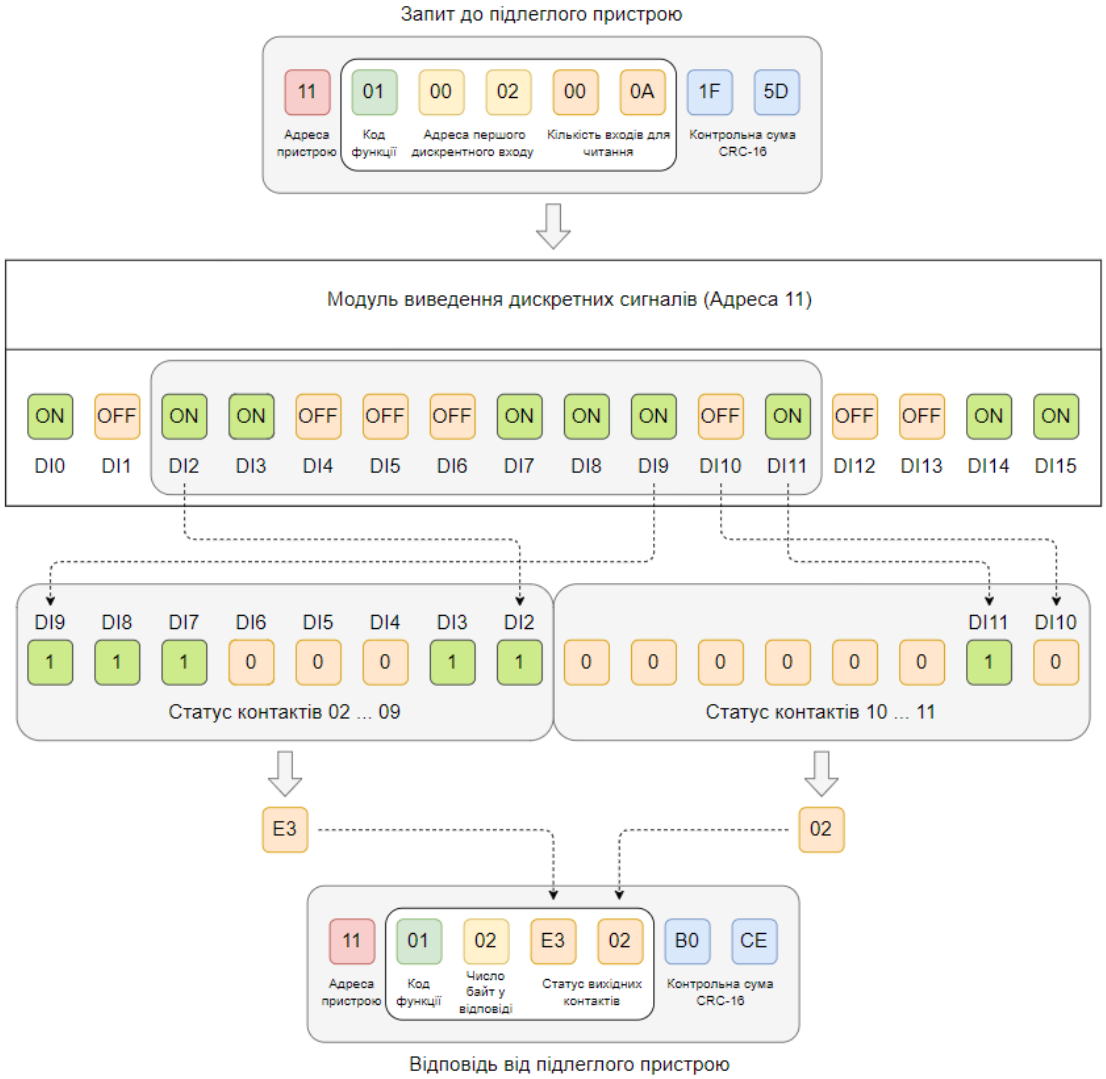


Рисунок 3.10 – Принцип побудови послідовності відповідних байтів у зворотному повідомленні

Запит обслуговується в кінці робочого циклу приладу, тому потрібно мати на увазі, що дані у відповідному повідомленні відображають стан регістрів на той момент.

3.4.3 Функція читання дискретних входів (02)

Ця функція дозволяє користувачеві отримати стан (ВМК / ВИМК) вхідних дискретних ліній пристрою, що адресується. Циклічний запит не підтримується. На додаток до адреси пристрою і номеру функції, запит вимагає, щоб інформаційне поле містило початкову адресу і кількість необхідних ліній.

Адресація дозволяє отримати за один запит до 2000 ліній. Однак, деякі пристрої мають обмеження на максимальну кількість ліній, одержуваних за один запит. Вхідні лінії нумеруються з нуля (10001 = 0, 10002 = 1 і т.д.).

У таблиці 3.5 представлений приклад запита на читання дискретних входів 10197-10218 з пристрою з адресою 17 (0x11).

Таблиця 3.5 – Приклад запита на читання дискретних входів 10197-10218

Байт	Опис полів запита
11	Адреса підлеглого пристрою
02	Код функції
00	Адреса початкового біта, Hi байт
C4	Адреса початкового біта, Lo байт
00	Кількість дискретних входів, Hi байт
16	Кількість дискретних входів, Lo байт
BA	Контрольна сума CRC
A9	Контрольна сума CRC

Приклад відповіді на цей запит представлений в таблиці 3.6. Відповідь включає адресу пристрою, код функції, кількість байт даних, дані і поле контрольної суми.

Таблиця 3.6 – Приклад повідомлення у відповідь на функцію 02

Байт	Опис полів відповіді
11	Адреса підлеглого пристрою
02	Код функції
03	Кількість байт в полі даних
AC	Статус вхідних контактів, перший байт
DB	Статус вхідних контактів, другий байт
35	Статус вхідних контактів, третій байт
20	Контрольна сума CRC
18	Контрольна сума CRC

Дані упаковані по біту на кожен вхід (1 = ON, 0 = OFF). Молодший біт першого байту містить значення першого входу, що адресується, за яким

слідують інші. Якщо кількість запитаних входів не кратне 8, то інші біти заповнюються нулями. Кількість байт даних завжди визначається як кількість RTU даних.

Так як пристрій обслуговує запит в кінці робочого циклу, дані у відповіді відображають стан входів на даний момент. Деякі пристрої мають обмеження на максимальну кількість входів, запитуваних за один запит.

Статус входів 10197-10204 = 0xAC = 1010 1100. Читаючи зліва направо, бачимо, що входи 10204, 10202, 10200 і 10199 в стані ON. Всі інші байти даних розпаковуються аналогічно.

Так як була запрошена інформація про стан 22 вхідних контактів, останній байт даних (35h = 0011 0101) містить інформацію тільки про 6 входів (10213-10218) замість 8-ми. Два останніх біта заповнюються нулями.

Приклад обміну повідомленнями при використанні функції 02 (читання дискретних входів) подано на рис. 3.11 та 3.12.

Рисунок 3.11 – Приклад запита для читання дискретних входів 10197-10204

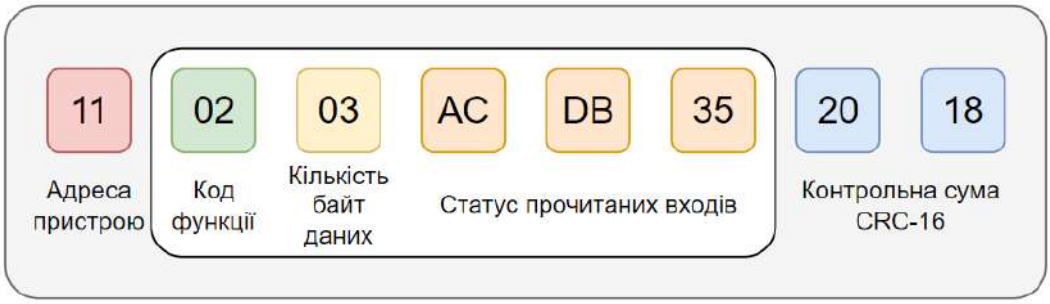


Рисунок 3.12 – Приклад відповіді на запит читання дискретних входів

3.4.4 Функція читання регістрів зберігання (03)

Ця функція дозволяє отримати бінарний вміст 16-ти розрядних регістрів підлеглого пристрою. Адресація дозволяє отримати за кожен запит до 125 регістрів. Однак, деякі пристрої мають обмеження на максимальну кількість

регістрів, які отримуються за один запит. Регістри нумеруються з нуля (40001 = 0, 40002 = 1 і т.д.). Широкомовний режим не допускається.

У таблиці 3.7 представлений приклад запита на читання регістрів 40108-40110 з пристрою з адресою 17 (0x11).

Таблиця 3.7 – Приклад повідомлення у відповідь на функцію 03

Байт	Опис полів запита
11	Адреса підлеглого пристрою
03	Код функції
00	Адреса першого байта, Hi байт
6B	Адреса першого байта, Lo байт
00	Число регістрів для читання, Hi байт
03	Число регістрів для читання, Lo байт
76	Контрольна сума CRC
87	Контрольна сума CRC

Пристрій, що адресується посилає у відповіді свою адресу, код виконаної функції і інформаційне поле.

Інформаційне поле містить 2 байти, які описують кількість байт даних, що повертаються. Довжина кожного регістру даних – 2 байти. Перший байт даних в посилці є старшим байтом регістру, другий – молодшим.

Так як пристрій зазвичай обслуговує запит в кінці свого робочого циклу, дані у відповіді відображають вміст регістрів в даний момент.

Деякі пристрої обмежують кількість регістрів, переданих за один запит. У цьому випадку для отримання, більшого числа регістрів, необхідно виконати кілька послідовних запитів.

У таблиці 3.8 представлений приклад відповіді на запит читання регістрів 40108-40110, які мають вміст, відповідно, 555, 0, 100, з пристрою з адресою 17 (0x11).

На рис. 3.13 подано приклад контролера «МС-1», що управляє частотою обертів вентилятора. На даному прикладі розглянемо принцип моніторингу стану системи управління вентиляцією за допомогою протоколу Modbus. В пристрої, що зображений на рис. 3.13 присутні п’ять внутрішніх регістрів для зберігання поточних даних роботи пристрою управляння вентилятором.

Таблиця 3.8 – Приклад відповіді на функцію 03

Байт	Опис полів відповіді
11	Адреса підлеглого пристрою
03	Код функції
06	Кількість байт в полі даних
02	Значення першого байта даних, Hi байт
2B	Значення першого байта даних, Lo байт
00	Значення другого байта даних, Hi байт
00	Значення другого байта даних, Lo байт
00	Значення третього байта даних, Hi байт
64	Значення третього байта даних, Lo байт
C8	Контрольна сума CRC
BA	Контрольна сума CRC

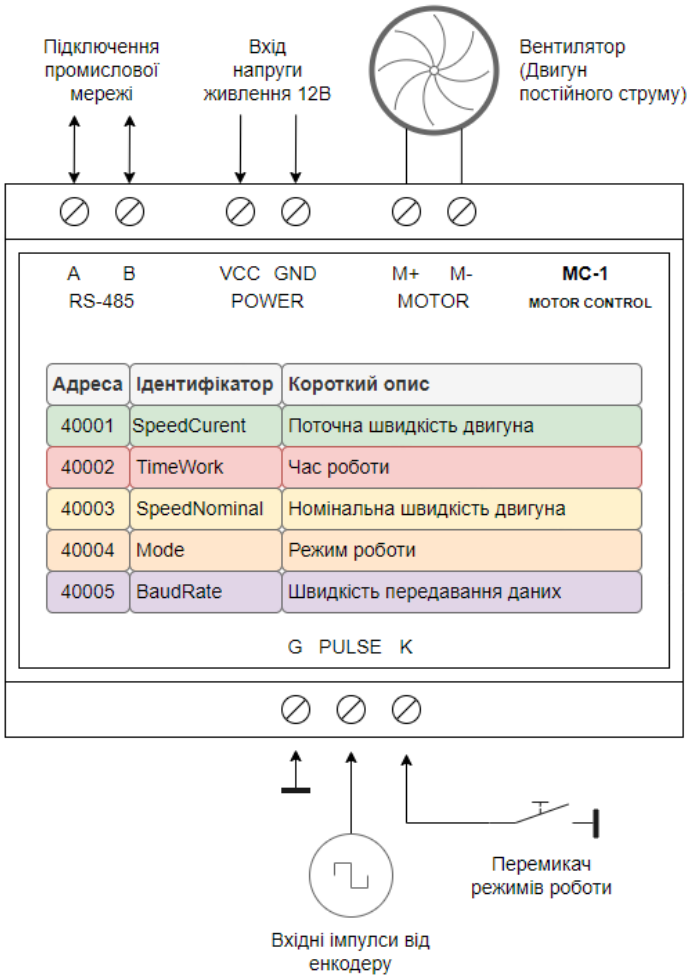


Рисунок 3.13 – Приклад контролеру «МС-1», який управляє частотою обертів вентилятора

Регістр 40001 зберігає поточну швидкість двигуна вентилятора. Отримавши значення з цього регістру можна дізнатися, в якому стані зараз знаходиться двигун та як швидко він обертається. Інформація в даному регістрі оновлюється на основі даних від датчика обертів (енкодеру).

Регістр 40002 зберігає поточний час роботи двигуна вентилятора з моменту останнього включення. Після зупинки, та нового старту, це значення скидається та оновлюється. Значення даного регістру збільшується на одиницю кожну секунду. Максимальне значення, що може зберігатися в регістрі – 65535 с.

Регістр 40003 зберігає номінальну, або задану швидкість роботи двигуна вентилятора. Це значення автоматично змінюється при зміні режиму роботи вентилятора за допомогою кнопки, або заноситься в даний регістр за допомогою протоколу Modbus через промислову мережу. Значення в регістрі задається в форматі «число обертів за хвилину», наприклад, 500 об/хв.

Регістр 40004 зберігає поточний режим роботи вентилятора (число в діапазоні від 0 до 4). Всього передбачено 5 режимів, або 5 швидкостей:

– 0: двигун вимкнено, швидкість 0 об/хв.;

– 1: двигун обертається з мінімальною швидкістю 150 об/хв.;

– 2: двигун обертається зі швидкістю 500 об/хв.;

– 3: двигун обертається зі швидкістю 1000 об/хв.;

– 4: двигун обертається з максимальною швидкістю 1500 об/хв.

Регістр 40005 зберігає налаштування швидкості передавання даних в мережі Modbus. Можливі значення: 1200, 2400, 4800, 9600, 19200 та 38400 біт/с.

Розглянемо приклад перевірки значення поточної швидкості роботи двигуна. Виходячи з рис. 3.13, поточна швидкість зберігається в регістрі 40001. Для читання цього значення формуємо запит, та отримуємо відповідь (рис. 3.14).

В запиті в якості початкової адреси вказуємо число «00 00» тому, що 40001 – це перший, тобто нульовий регістр (відлік починається з нуля). Після початкової адреси вказуємо кількість регістрів, інформацію з яких ми хочемо отримати. В даному прикладі – це один регістр. Таким чином, наступні два байти будуть мати значення «00 01» (відповідно до таблиці 3.7).

В результаті вдалого обміну даними, головний пристрій отримає наступне повідомлення:

```
11 03 02 05 E1 BA 9F
```

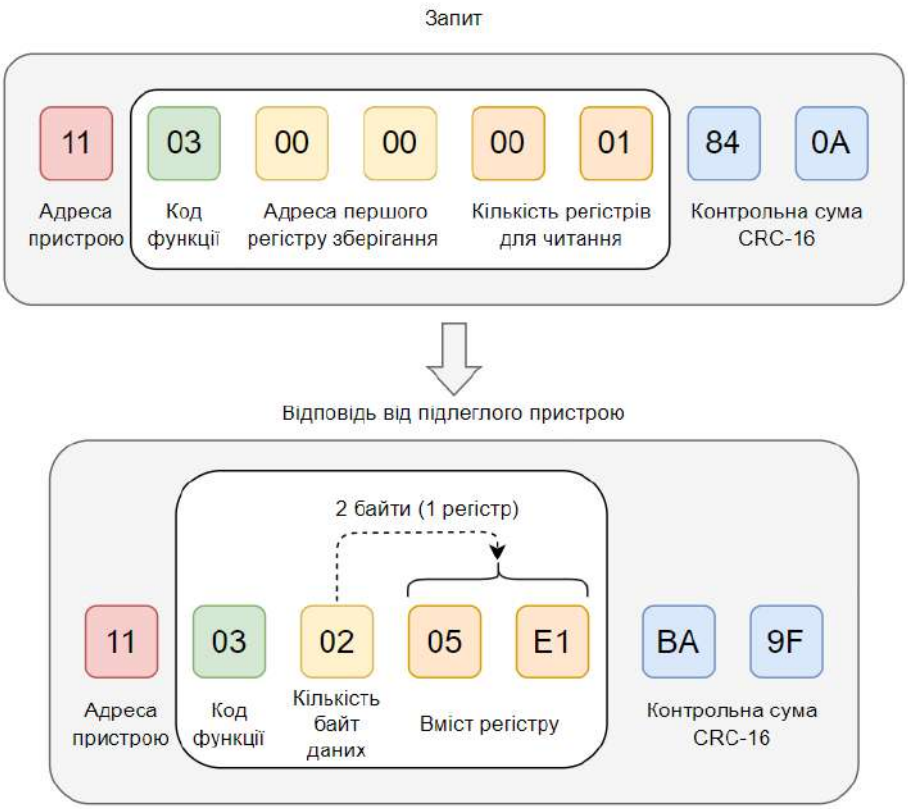


Рисунок 3.14 – Приклад обміну даними між пристроями при читанні значення регістру 40001

В цьому повідомленні, після номеру функції «03» йде кількість переданих байт («02») та самі байти даних («05 E1»).

Для того , щоб дізнатися реальну швидкість обертів двигуна вентилятора необхідно перевести число 05E1 із шістнадцятирічного формату запису в десятковий (рис. 3.15).

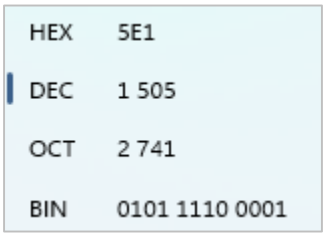


Рисунок 3.15 – Представлення числа 05E1 в десятковому вигляді

Таким чином, ми бачимо, що поточне значення швидкості оберту становить 1505 об/хв. Для того, щоб дізнатись, чи відповідає така швидкість поточному режиму вентилятора, та визначити задану номінальну швидкість, необхідно прочитати вміст регістрів зберігання за адресами 40003 та 40004.

На рис. 3.16 подано сформований запит до пристрою МС-1.

Рисунок 3.16 – Запит до контролера «МС-1»

В запиті вказується адреса першого регістра для читання – 0002, що відповідає адресі 40003. Кількість регістрів, що необхідно прочитати – 2, тому наступні два байти в запиті: 0002. В результаті читання вказаних регістрів отримуємо таку відповідь (рис. 3.17).

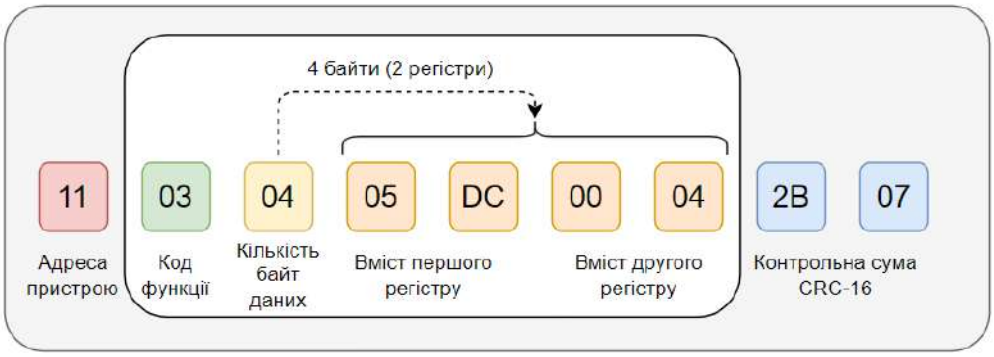


Рисунок 3.17 – Відповідь від контролера МС-1

Проведемо аналіз отриманої відповіді у відповідності до таблиці 3.8. Відповідь містить номер функції (03) та інформацію про кількість байтів даних, що містять інформацію з регістрів зберігання. Таких байтів – чотири (по два байти на один 16-ти розрядний регістр).

Виходячи з отриманої інформації, перший з прочитаних регістрів (адреса в пристрої МС-1 40003) містить число 05DC. Відповідно таблиці призначення регістрів, ми отримали номінальну швидкість двигуна вентилятора. Якщо перевести це число в десятковий формат, отримаємо число 1500 (рис. 3.18).

Таким чином, ми бачимо, що номінальне значення швидкості оберту двигуна вентилятора становить 1500 об/хв. Порівнюючи з поточною швидкістю двигуна, що була отримана в попередньому сеансі обміну даними з пристроєм

керування (1505 об/хв), можна бачити, що різниця невелика, та становить 0,3 %. Таке відхилення в швидкості роботи двигуна є припустимим та лежить в межах норми.

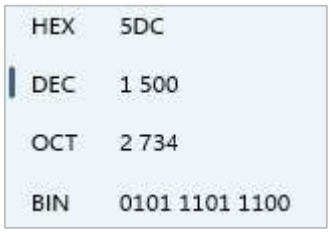


Рисунок 3.18 – Представлення числа 05DC в десятковому вигляді

Другий байт містить число, що відповідає режиму роботи вентилятора. Виходячи з опису роботи контролеру МС-1, швидкості 1500 об/хв. повинен відповідати 4 режим роботи. Якщо подивитись на рис. 3.17, то саме це число й міститься у відповіді (0004). Таким чином, сеанс обміну інформацією з контролером управляння вентилятором МС-1 показав, що прилад працює в штатному режимі, відхилень в роботі не виявлено.

3.4.5 Запис одного регістру зберігання (06)

Ця функція використовується для запису одного регістра зберігання у віддаленому пристрої. PDU запита вказує адресу регістра, який потрібно записати. Адреси регістрів починаються з нуля, тому регістр під номером 1 адресується як 0. В таблиці 3.9 подано приклад запита для запису в регістр 40136 числа 926 (0x039E). Адреса пристрою 17 (0x11).

Таблиця 3.9 – Приклад запису в регістр 40136 числа 926

Байт	Опис полів запита
11	Адреса підлеглого пристрою
06	Код функції
00	Адреса першого байта, Hi байт
87	Адреса другого байта, Lo байт
03	Значення байта даних, Hi байт
9E	Значення байта даних, Lo байт
5A	Контрольна сума CRC
D3	Контрольна сума CRC

У випадку вдалого виконання запита у відповідь буде надіслане повідомлення, що ідентичне запита. Приклад відповіді подано в таблиці 3.10.

Таблиця 3.10 - Приклад відповіді на запит запису в регістр 40136 числа 926

Байт	Опис полів відповіді
11	Адреса підлеглого пристрою
06	Код функції
00	Адреса першого байта, Hi байт
87	Адреса другого байта, Lo байт
03	Значення байта даних, Hi байт
9E	Значення байта даних, Lo байт
5A	Контрольна сума CRC
D3	Контрольна сума CRC

Розглянемо приклад застосування даної функції протоколу Modbus на практиці. Наприклад, необхідно змінити режим роботи контролера МС-1, що керує роботою вентилятора.

Для зміни режиму роботи необхідно змінити зміст регістра 40004 (рис. 3.13). Якщо ми хочемо перевести вентилятор в економний режим, нам потрібно змінити режим роботи на «1». В даному режимі двигун буде обертатися з мінімальною швидкістю 150 об/хв.

На рис. 3.18 подано приклад запита для переведення вентилятора в режим роботи №1.

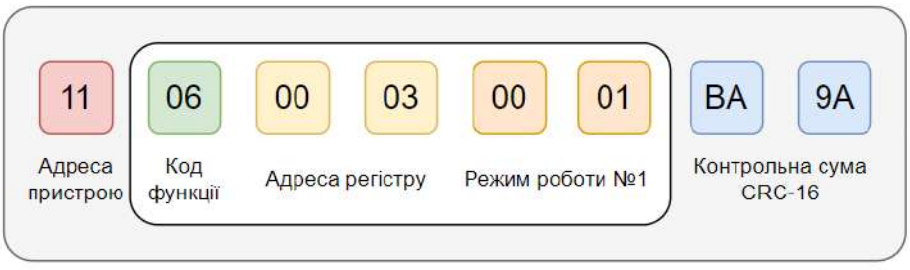


Рисунок 3.18 – Приклад запита для зміни режиму роботи двигуна вентилятора

Очікувана відповідь повторює запит (рис. 3.19).

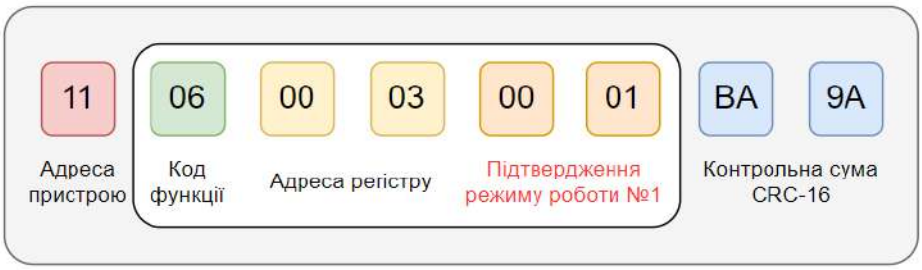


Рисунок 3.19 – Підтвердження запису в регістр 40003 режиму роботи двигуна вентилятора №1

3.4.6 Запис декількох регістрів зберігання (0x10)

Часто виникають задачі модифікації відразу декількох регістрів зберігання. Для цього в протоколі Modbus передбачена функція з номером 16 (0x10). В таблиці 3.11 подано приклад запита для запису в регістри 40136 та 40137 значень 0x00a0 та 0x0102. Адреса пристрою 17 (0x11).

Таблиця 3.11 – Приклад запита для запису значень в регістри 40136 та 40137

Байт	Опис полів запита
11	Адреса підлеглого пристрою
10	Код функції
00	Адреса першого байта, Hi байт
87	Адреса першого байта, Lo байт
00	Кількість регістрів для запису, Hi байт
02	Кількість регістрів для запису, Lo байт
04	Кількість байт в полі даних
00	Значення першого байта даних, Lo байт
0A	Значення першого байта даних, Lo байт
01	Значення другого байта даних, Hi байт
02	Значення другого байта даних, Lo байт
4E	Контрольна сума CRC
BA	Контрольна сума CRC

Повідомлення дозволяє записувати регістри з максимальною логічною адресою до 0xFFFF. Не використані старші біти адреси регістрів повинні заповнюватися нулями. Якщо використовується широкомовний запит, то вміст поля даних записується в усі пристрої, підключені до шини. У випадку вдалого

виконання запита у відповідь буде надіслане повідомлення, що містить адресу першого байту, та кількість байт даних, що було записано. Приклад відповіді подано в таблиці 3.12.

Таблиця 3.12 – Відповідь на запит щодо запису даних в два регістри зберігання

Байт	Опис полів відповіді
11	Адреса підлеглого пристрою
10	Код функції
00	Адреса першого байта, Hi байт
87	Адреса другого байта, Lo байт
00	Кількість записаних регістрів, Hi байт
02	Кількість записаних регістрів, Lo байт
F3	Контрольна сума CRC
71	Контрольна сума CRC

Розглянемо приклад застосування даної функції протоколу Modbus на практиці. Наприклад, необхідно за один сеанс обміну інформацією змінити режим роботи контролеру МС-1, що керує роботою вентилятора, та швидкість передавання даних в мережі.

Виходячи з рис. 3.13, режим роботи пристрою змінюється за допомогою регістра 40004, до якого записується число в діапазоні від 0 до 4. Швидкість передавання даних задається в залежності від значення, що міститься в регістрі 40005. На рис. 3.20 подано приклад запита, в якому міститься інформація про новий режим роботи (режим №2), та про нову швидкість передавання даних (19200 біт/с).

Рисунок 3.20 – Приклад запита для зміни режиму роботи та швидкості передавання даних

На рис. 3.21 подано отриману відповідь від контролера. Тут можна бачити, що в контролері МС-1 було змінено два регістри, починаючи з адреси 40005.

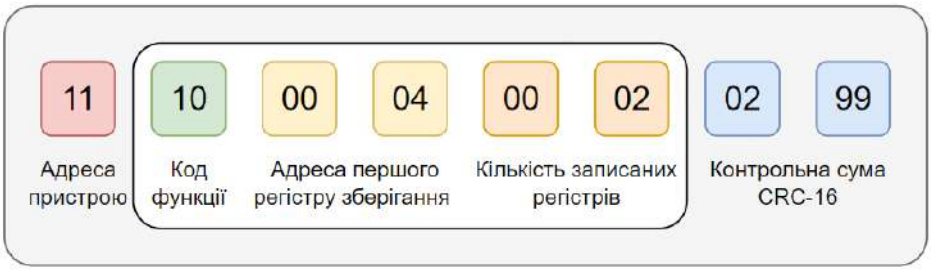


Рисунок 3.21 – Отримана відповідь від контролера

3.4.7 Зміна стану одного вихідного контакту (05)

Це повідомлення модифікує один логічний осередок, що являє собою один дискретний вихід контролеру. Осередки нумеруються з нуля (осередок 1 = 0, осередок 2 = 1 і т.д.).

Для увімкнення дискретного виходу необхідно передати число 0xFF00. Ця команда встановлює певний осередок в «1». Якщо необхідно вимкнути дискретний вихід, необхідно передати число 0x0000. Ця команда переведе певний осередок в стан логічного «0». Інші передані числа не впливають на вміст осередка, що адресується. Ця функція може використовуватися в широкомовному режимі. У таблиці 3.13 подано приклад установки в «1» дискретного виходу з адресою 0173 (0x00AD) в пристрій з адресою 17 (0x11). У відповідь повинне надійти повідомлення, яке повністю співпадає з запитом.

Таблиця 3.13 – Приклад встановлення в стан логічної «1» осередка 0173

Байт	Опис полів запиту
11	Адреса підлеглого пристрою
05	Код функції
00	Адреса дискретного виходу, Hi байт
AD	Адреса дискретного виходу, Lo байт
FF	Ознака встановлення (0xFF), або скидання (0x00)
00	Завжди містить значення 0x00
1F	Контрольна сума CRC
4B	Контрольна сума CRC

3.4.8 Зміна стану декількох вихідних контактів (0F)

Цей код функції використовується для примусового ввімкнення або вимкнення кожного вихідного контакту (котушки) в межах адресного простору у віддаленому пристрої. PDU запита вказує посилання на певні контакти, які потрібно примусово встановити, або вимкнути. Контакти адресуються починаючи з нуля. Тому котушка під номером 1 адресується як 0.

Необхідні стани On/Off визначаються вмістом поля даних запита. Логічна «1» у бітовій позиції поля вимагає, щоб відповідний вихід був увімкнений. Логічний «0» вимагає, щоб його було вимкнено.

У таблиці 3.14 наведено приклад зміни стану 9 дискретних виходів з адресами починаючи з 00028 до 00036 для віддаленого пристрою з адресою 11 (0x0B).

Таблиця 3.14 – Приклад зміни стану 9 дискретних виходів

Байт	Опис полів запиту
0B	Адреса підлеглого пристрою
0F	Код функції
00	Початкова адреса дискретного виходу, Hi байт
1B	Початкова адреса дискретного виходу, Lo байт
00	Кількість виходів, стани яких потрібно змінити, Hi байт
09	Кількість виходів, стани яких потрібно змінити, Lo байт
02	Кількість байт даних
4D	Перший байт даних
01	Другий байт даних
6C	Контрольна сума CRC
A7	Контрольна сума CRC

Принцип побудови байт даних подано на рис. 3.22. Кількість байт даних визначається таким чином:

9 дискретних виходів = 1 байт + 1 біт + 7 порожніх біт = 2 байти.

Далі збираються байти даних. Дискретні виходи 35-28 мають наступну комбінацію: 0100 1101. Дискретний вихід 36 повинен бути увімкнений

(логічна «1»). Після додавання семи порожніх біт, отримаємо комбінацію: 0000 0001. Таким чином, два байти даних будуть мати наступне значення: 4D 01.

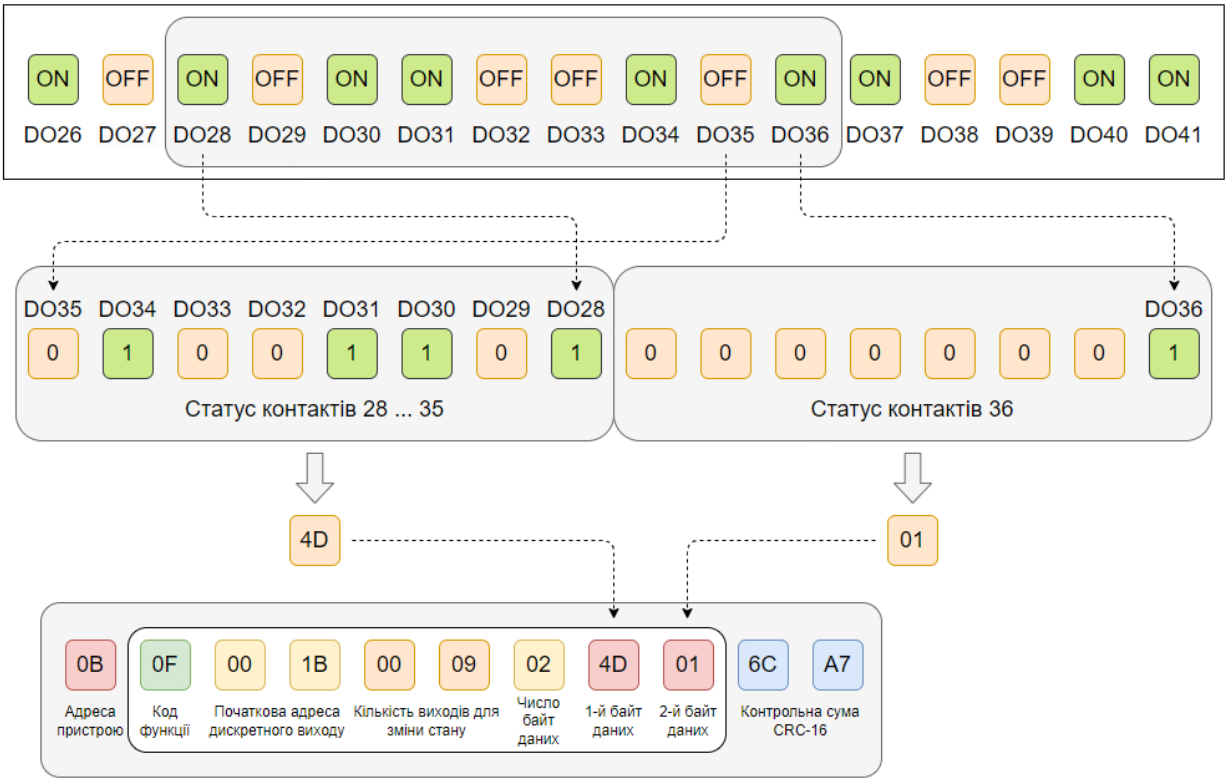


Рисунок 3.22 – Принцип побудови байт даних для формування повідомлення зміни декількох дискретних виходів

Як можна бачити з рис. 3.22, старші біти містять старші змінні вихідних контактів. З наведеного прикладу видно, що котушка 28 увімкнена (логічна «1»), а котушка 35 вимкнена (логічний «0»). Останнє поле даних містить статус лише одного дискретного виходу (логічна «1»). Невикористані біти в останньому байті даних заповнюються нулями – це тримачі простору.

Відповідь від підлеглого пристрою містить адресу розміщення першого дискретного виходу та підтвердження кількості записаних дискретних виходів (табл. 3.15).

3.4.9 Читання стану вхідних регістрів (04)

Вхідні регістри в промислових контролерах зазвичай використовуються для запису даних, що отримуються від аналого-цифрових перетворювачів. Це можуть біти значення виміряного струму, напруги або, наприклад, температури. Вхідні регістри 16-розрядні. Якщо значення параметру, що

вимірюється, перевищують 16 біт (наприклад, АЦП розрядністю 24 біти), то в пам’яті ПЛК вони займатимуть два регістри.

Таблиця 3.15 – Відповідь на запит щодо зміни стану декількох вихідних контактів

Байт	Опис полів відповіді
0B	Адреса підлеглого пристрою
0F	Код функції
00	Адреса першого байта, Hi байт
1B	Адреса другого байта, Lo байт
00	Кількість записаних дискретних виходів, Hi байт
1B	Кількість записаних дискретних виходів, Lo байт
E5	Контрольна сума CRC
60	Контрольна сума CRC

В протоколі Modbus для читання вхідних регістрів використовується функція 0x04. Вона використовується для безперервного читання від 1 до 125 вхідних регістрів з віддаленого пристрою. PDU запита вказує початкову адресу регістра та кількість регістрів. В PDU адресація регістрів починається з нуля. Тому вхідний регістр під номером «1» адресується як «0».

У таблиці 3.16 поданий приклад запита на читання значення АЦП з вхідного регістра 30011 з пристрою з адресою 17 (0x11).

Таблиця 3.16 – Приклад запиту на читання значення АЦП

Байт	Опис полів запиту
11	Адреса підлеглого пристрою
04	Код функції
00	Адреса першого байта вхідного регістра, Hi байт
0A	Адреса першого байта вхідного регістра, Lo байт
00	Число регістрів для читання, Hi байт
01	Число регістрів для читання, Lo байт
13	Контрольна сума CRC
58	Контрольна сума CRC

Пристрій, що адресується посилає у відповіді свою адресу, код виконаної функції і інформаційне поле.

Для даного прикладу інформаційне поле містить 2 байти, які описують кількість байт даних, що повертаються. Довжина кожного регістра даних – 2 байти. Перший байт даних в посилці є старшим байтом регістру, другий – молодшим.

У таблиці 3.17 поданий приклад відповіді на запит читання вхідного регістру 30011, в якому міститься число 0x102F (4143), що у двійковому форматі має наступний вигляд – 0001000000101111.

Таблиця 3.17 – Приклад відповіді на функцію 04

Байт	Опис полів відповіді
11	Адреса підлеглого пристрою
04	Код функції
02	Кількість байт в полі даних
10	Значення першого байта даних, Hi байт
2F	Значення першого байта даних, Lo байт
34	Контрольна сума CRC
EF	Контрольна сума CRC

3.5 Застосування протоколу Modbus для керування віртуальними приладами

3.5.1 Стислий опис макета світлової колони

Даний віртуальний прилад є цифровим двійником світлосигнальної колони для візуалізації стану промислового обладнання.

На рис. 3.23 подано зовнішній вигляд інтерфейсу віртуального приладу.

Віртуальний прилад повністю реалізує поведінку реального приладу та може працювати із зовнішною програмою або іншим пристроєм за одним із можливих протоколів:

– Named Pipes для Inter-Process Communication;

– Modbus TCP/IP;

– Serial Protocol.

Для управління роботою даного приладу будемо використовувати протокол Modbus TCP/IP. Перед натисканням на кнопку «Connect» потрібно обрати саме цей тип протоколу у вікні налаштування параметрів підключення.

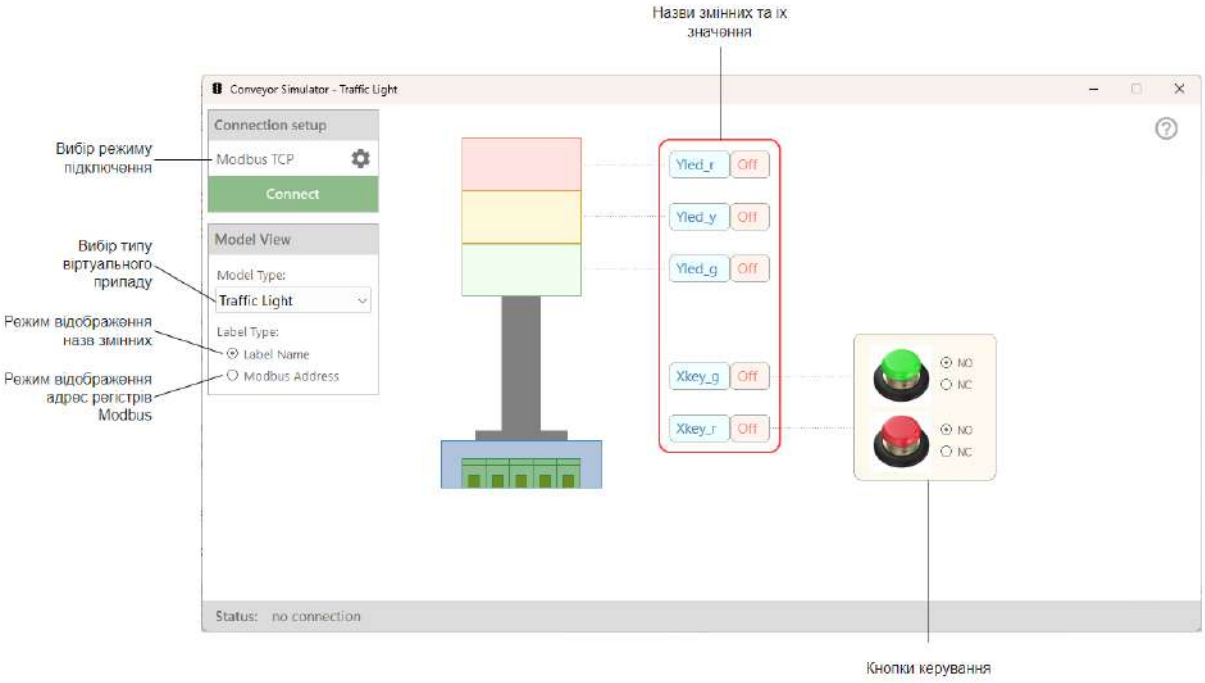


Рисунок 3.23 – Зовнішній вигляд інтерфейсу віртуального приладу «Світлова колона»

В інтерфейсі програми передбачено наявність візуалізації стану внутрішніх змінних, що відображають стан елементів віртуального макета. Внутрішні змінні мають назву для доступу до них з інших програм (наприклад, LDmicro), або з віртуального ПЛК.

До змінних можна звернутись за їх адресою в пам’яті програми за допомогою інтерфейсу Modbus. На рис. 3.24 подано варіант інтерфейсу віртуального приладу при обиранні режиму відображення адрес регістрів Modbus.

Також, в даному режимі на екран виводиться довідник адрес регістрів Modbus та їх класифікація в залежності від типу.

Віртуальний прилад «Світлова колона» реалізує два типи регістрів:

– 1-бітні дискретні виходи (читання / запис) «Coils» (адресний простір 00001-10000);

– 1-бітні дискретні входи (читання) «Discrete Inputs» (адресний простір 10001-20000).

Кнопки керування можуть бути налаштовані, як пристрій з нормально відкритими контактами (Normal Open, або NO) та з нормально закритими контактами (Normal Close, або NC).

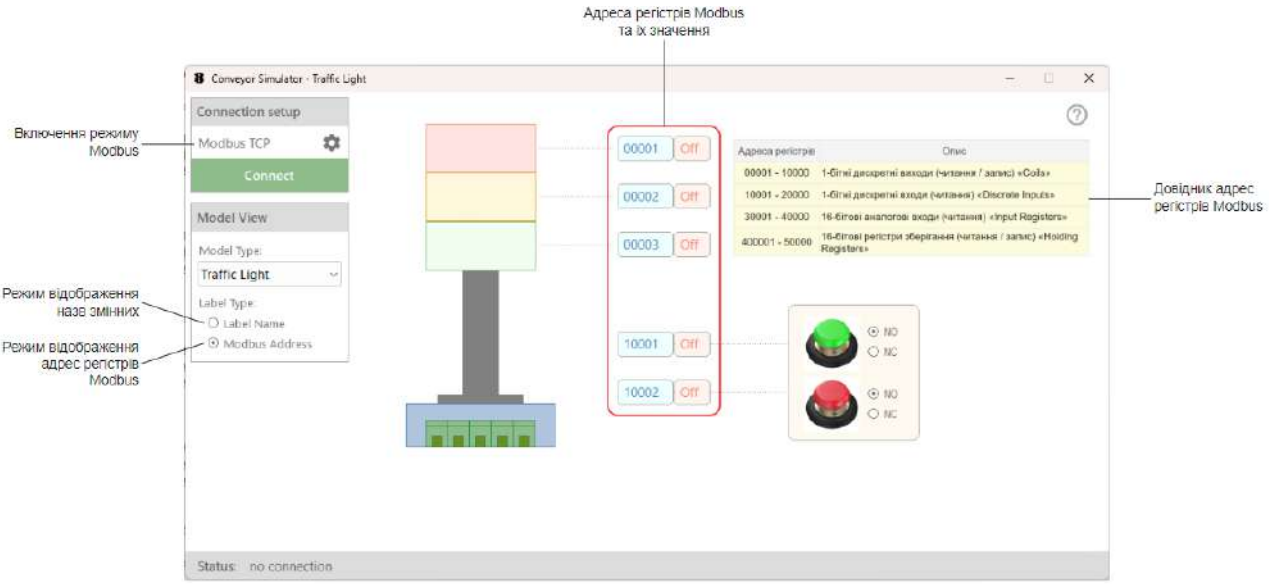


Рисунок 3.24 – Інтерфейс віртуального приладу обрання режиму відображення адрес регістрів Modbus

3.5.2 Приклад керування макетом світлової колони

Розглянемо приклад керування світловою колоною за допомогою програмного симулятора «Майстер мережі Modbus TCP/IP».

Наприклад, необхідно включити жовтий сегмент колони. По-перше, необхідно дізнатися адресу вихідного контакту, що відповідає за цей сегмент у віртуальному приладі. Для цього перемикаємо інтерфейс у режим «Modbus Address» в секції «Model View». Навпроти жовтого сегменту знаходиться адреса «00002» та його поточне значення «Off» (вимкнено) (рис. 3.24).

Для того, щоб сегмент увімкнути, необхідно записати в регістр «00002» значення логічної «1». Сформуємо відповідний Modbus-запит, обравши функцію 0x05. Виходячи з опису протоколу Modbus та функції 0x05, до підлеглого пристрою потрібно передати:

– адресу підлеглого пристрою (01);

– код функції (05);

– адресу дискретного виходу, Hi байт (00);

– адресу дискретного виходу, Lo байт (01);

– ознаку встановлення (0xFF 00), або скидання (0x00 00);

– контрольну суму CRC.

Таким чином, запит до віртуального макета матиме наступний вигляд:

```
01 05 00 01 FF 00 CRC
```

Необхідно звернути увагу, що адреса в запиті вказується на одиницю менша, тому що за правилами протоколу Modbus рахунок починається з «0» (00001 = 0x0000, 00002 = 0x0001, 00003 = 0x0002)

Перед перевіркою правильності побудови запита виконуємо налаштування віртуального макета. Для цього викликаємо вікно конфігурації, натискаючи на кнопку з шестернею. На екрані ПК повинно з’явитись вікно налаштування параметрів підключення до мережі Modbus (рис. 3.25).

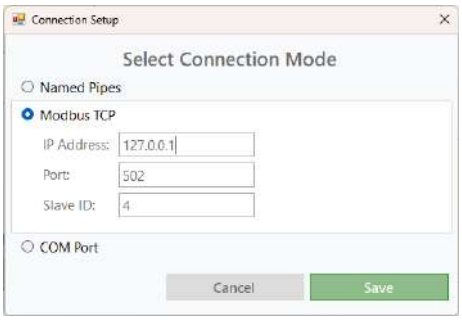


Рисунок 3.25 – Вікно налаштування параметрів підключення до мережі Modbus

В даному вікні обов’язково потрібно перевірити IP-адресу, за якою буде звертатися майстер мережі до віртуального приладу. Зазвичай ця адреса відповідає IP-адресі того ПК, на якому завантажена ця програма, або це може бути стандартна адреса localhost (127.0.0.1), якщо і віртуальній прилад і майстер мережі завантажені на одному ПК.

Після перевірки IP-адреси необхідно натиснути на кнопку «Connect» для підключення до мережі Modbus TCP/IP та переходу в режим очікування команд від майстра мережі. Результат виконання запиту подано на рис. 3.26.

Відповідь, що отримана від віртуального пристрою:

```
01 05 00 01 FF 00 DD FA
```

показує, що запит оброблено правильно і відповідний контакт увімкнуто.

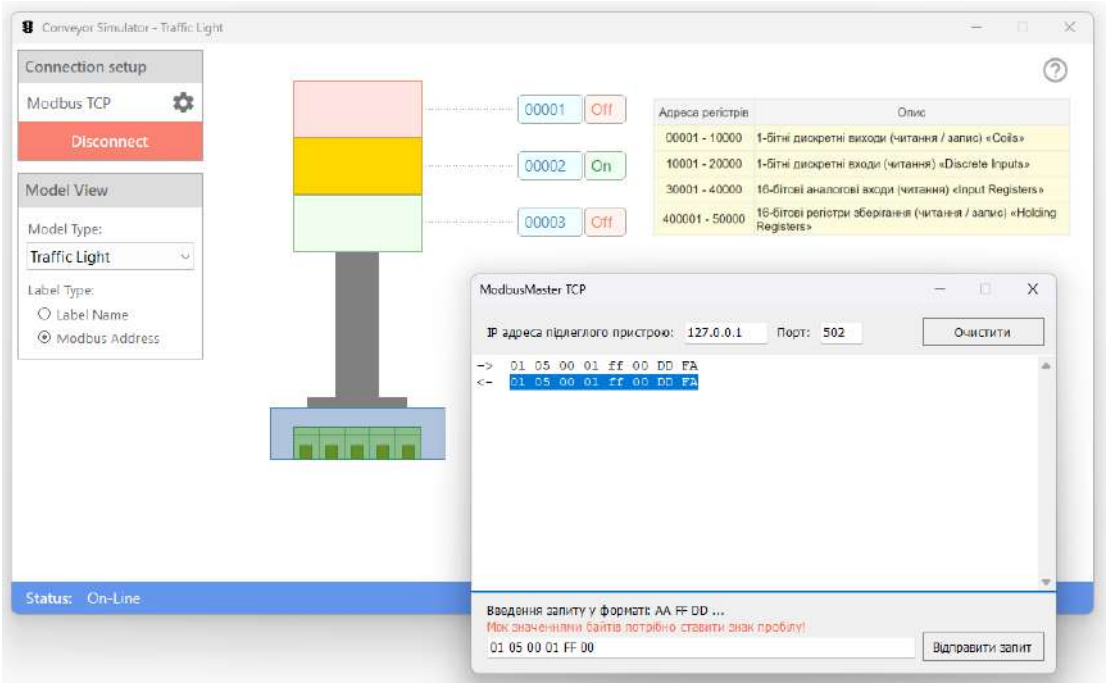


Рисунок 3.26 – Результат виконання запиту для увімкнення жовтого сегменту світлової колони

В наступному прикладі увімкнемо червоний та зелений сегменти колони, а жовтий сегмент вимкнемо. Виконуючи аналіз задачі, бачимо, що нам потрібно змінити стан одночасно трьох різних регістрів: «00001» (червоний), «00002» (жовтий) та «00001» (зелений).

Для рішення даної задачі необхідно застосувати іншу функцію протоколу Modbus – 0x0F. Відповідно до стандарту протоколу необхідно сформувати запит у вигляді:

– адреса підлеглого пристрою (01);

– код функції (0F);

– початкова адреса дискретного виходу, Hi байт (00);

– початкова адреса дискретного виходу, Lo байт (00);

– кількість виходів, стани яких потрібно змінити, Hi байт (00);

– кількість виходів, стани яких потрібно змінити, Lo байт (03);

– кількість байт даних (01);

– перший байт даних (00000101 = 05);

– контрольна сума CRC.

Формування байту даних відбувається відповідно до рис. 3.27. Кожному сегменту світлової колони відповідає один біт. Розташування бітів відбувається з права на ліво.

Рисунок 3.27 – Формування байта даних

Таким чином, запит до віртуального макету буде мати наступний вигляд:

```
01 0F 00 00 00 03 01 05 CRC
```

Результат виконання запиту подано на рис. 3.28.

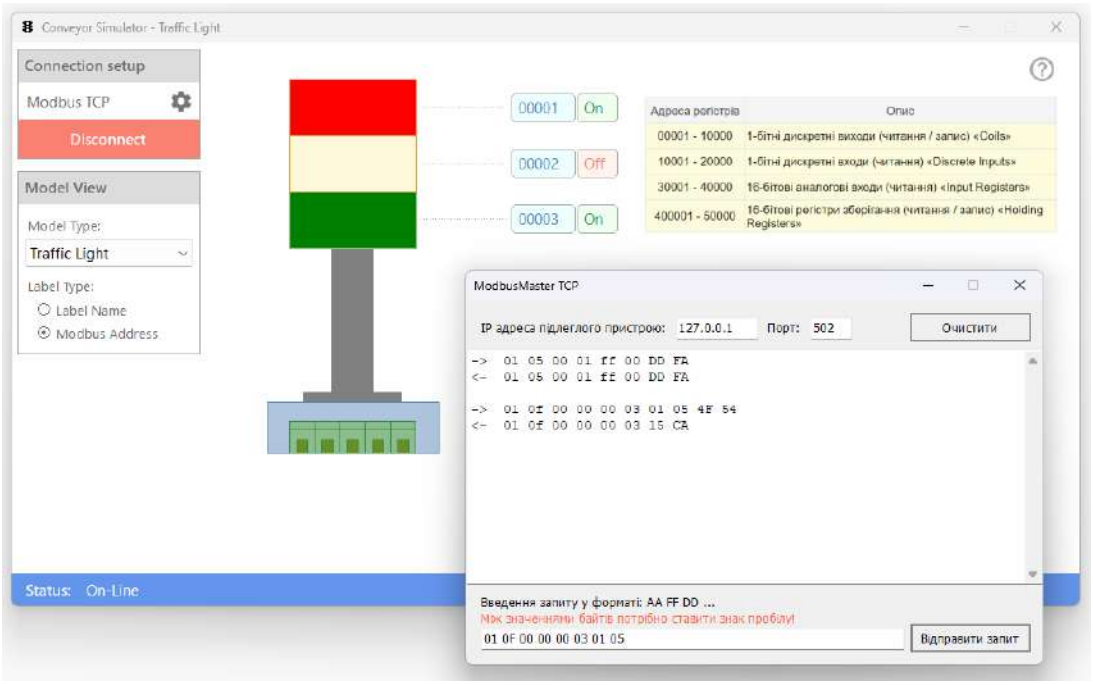


Рисунок 3.28 – Увімкнення сегментів 1 та 3 світлової колони

Як можна бачити з рис. 3.28, умови завдання виконані. Відповідні сегменти світлової колони змінили свій стан. Отримана відповідь, також підтверджує правильність виконання запиту.

3.5.3 Опис віртуального макета промислового конвеєра

Докладний опис віртуального макета промислового конвеєра подано в навчальному посібнику «Застосування цифрових двійників технічних засобів автоматизації для розроблення програмно-технічних комплексів АСУ ТП» [15]. На рис. 3.29 подано зовнішній вигляд інтерфейсу програми.

Рисунок 3.29 – Зовнішній вигляд інтерфейсу віртуального макета промислового конвеєра

Конвеєр запускається записом логічної одиниці в змінну «YC_Moto». В цьому випадку починає рухатись полотно конвеєрної лінії, що візуально відображається на екрані ПК.

Видача деталей відбувається короткочасним записом логічної одиниці в змінну «YC_Stor1». Час утримання сигналу високого рівня має бути не менше 0,5 с, та не більше 2,5 с.

Видача деталей може здійснюватися автоматично. Для цього потрібно обрати режим «Auto» в списку, що випадає, який розташовано над змінною «YC_Stor1». В даному випадку зовнішні сигнали керування ігноруються.

Наприкінці конвеєрної лінії розташовано приймач деталей. Деталі, що не були переспрямованні в процесі руху лінією, потрапляють в цей приймач та зберігаються там до моменту натискання на кнопку «Reset».

Кількість деталей, що міститься в магазині, та в кінцевому приймачі відображається у вигляді цифри на фоні деталі в відповідному пристрої.

Наприклад, цифра «5», що розташована на червоній деталі, відповідає їх початковій кількості в магазині.

В процесі переміщення деталей конвеєрною лінією, їх напрям руху можна змінити за допомогою двох важелів «YC_Sp1» та «YC_Sp2». Управління важелями відбувається записом логічної одиниці у відповідну змінну. Увімкнений важіль візуально перекриває напрямок руху деталей та змушує її рухатись в один з двох накопичувачів «YC_MSt1» та «YC_ MSt2».

Кількість деталей в накопичувачах відображається цифрою на фоні останньої отриманої виробничої одиниці.

За допомогою запитів в форматі протоколу Modbus можна дізнатися про кількість деталей в магазині та у всіх накопичувачах. Відповідне значення записується в регістри зберігання. Розподіл адрес між регістрами зберігання та їх призначення подано в таблиці 3.18.

Таблиця 3.18 – Розподіл адрес між регістрами зберігання та їх призначення

Адреса регістра зберігання	Призначення регістра зберігання
40101	Кількість деталей в магазині
40102	Кількість деталей в накопичувачі №1
40103	Кількість деталей в накопичувачі №2
40104	Кількість деталей в кінцевому накопичувачі

Регістри зберігання не мають відповідних змінних тому доступ до них можна здійснити тільки за допомогою протоколу Modbus. Вказані регістри призначенні тільки для читання. Запис в них значень не впливає на роботу конвеєра тому що їх вміст оновлюється автоматично в процесі роботи програми керування віртуальним макетом.

В віртуальному макеті передбачено наявність трьох кнопок керування, функції яких визначає програміст під час створення технологічної програми.

Кожна кнопка пов’язана зі своєю змінною, тому змінна «XC_key1» відповідає верхній зеленій кнопці, змінна «XC_key2» відповідає середній червоній кнопці, а змінна «XC_key3» відповідає нижній червоній кнопці.

Також в макеті передбачені два сенсора. Перший сенсор «XC_sens1» встановлено на вході в конвеєр а другий «XC_sens2» – на виході. Перший сенсор крім дискретного виходу має ще три аналогових: «AC_R», «AC_G», «AC_B». Кожен з вказаних виходів пов’язано з внутрішнім 8-бітним АЦП. Значення цих

трьох каналів дають змогу визначити колір деталі, що проходить повз перший сенсор.

Доступ до всіх змінних можна виконати за допомогою запитів за протоколом Modbus. Кнопки і сенсори організовані в блок вхідних контактів, виконавчі пристрої – блок вихідних контактів, АЦП – вхідні регістри.

Для зручності користування віртуальним макетом в програмі передбачено режим відображення Modbus-адрес замість реальних назв змінних (рис. 3.30).

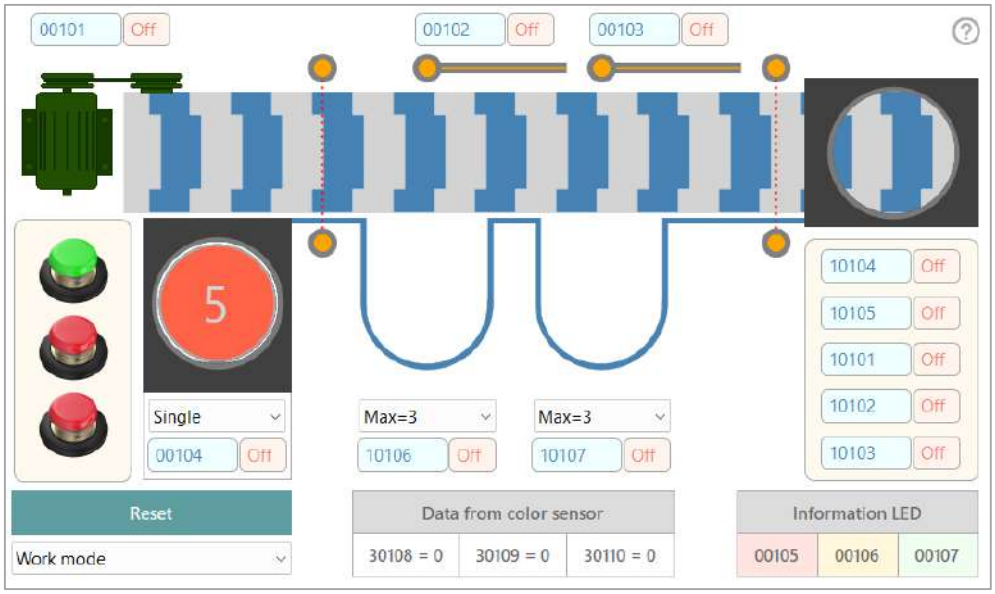


Рисунок 3.30 – Інтерфейс програми в режимі відображення Modbus-адрес

В таблиці 3.19 подано відомості про відповідність адрес вихідних контактів в форматі Modbus до програмних змінних.

В усіх вихідних контактів активний рівень – логічна одиниця. Таким чином, поява логічної одиниці призводить до увімкнення відповідного виконавчого пристрою.

Таблиця 3.19 – Відповідність адрес вихідних контактів в форматі Modbus до програмних змінних

Адреса контакту	Назва змінної	Призначення контакту
00101	YC_Moto	Управління двигуном, що рухає конвеєрну стрічку
00102	YC_Sp1	Управління першим важелем розподілювача поту деталей
00103	YC_Sp2	Управління другим важелем розподілювача поту деталей

Продовження таблиці 3.19

Адреса контакту	Назва змінної	Призначення контакту
00104	YC_Stor1	Увімкнення або вимкнення приводу, що видає деталі з магазину
00105	YC_R	Контакт управління червоним сегментом інформаційного дисплею
00106	YC_Y	Контакт управління жовтим сегментом інформаційного дисплею
00107	YC_G	Контакт управління зеленим сегментом інформаційного дисплею

В таблиці 3.20 подано відомості про відповідність адрес вхідних контактів в форматі Modbus до програмних змінних.

Таблиця 3.20 – Відповідність адрес вхідних контактів в форматі Modbus до програмних змінних

Адреса контакту	Назва змінної	Призначення контакту
10101	XC_key1	Верхня зелена кнопка керування макетом
10102	XC_key2	Середня червона кнопка керування макетом
10103	XC_key3	Нижня червона кнопка керування макетом
10104	XC_Sens1	Дискретний вихід сенсору наявності деталі, що розташована на вході конвеєрної лінії
10105	XC_Sens2	Дискретний вихід сенсору наявності деталі, що розташована на виході з конвеєрної лінії
10106	XC_MSt1	Дискретний вихід сенсору що сигналізує про досягнення максимальної кількості деталей в першому накопичувачі
10107	XC_MSt2	Дискретний вихід сенсору що сигналізує про досягнення максимальної кількості деталей в другому накопичувачі

Спрацювання будь-якого датчика призводить до формування сигналу логічної одиниці на виході відповідного контакту.

В таблиці 3.21 подано відомості про відповідність Modbus-адрес каналам АЦП.

В макеті використовується 8-бітний АЦП, враховуючи, що вхідні регістри 16-бітні, старший байт у відповіді завжди дорівнюватиме нулю.

Таблиця 3.21 – Відповідність Modbus-адрес каналам АЦП

Адреса контакту	Назва змінної	Призначення контакту
30108	AC_R	Дані про колір деталі, R-канал
30109	AC_G	Дані про колір деталі, G-канал
30110	AC_B	Дані про колір деталі, B-канал

3.5.4 Приклад керування макетом промислового конвеєра

Керування макетом буде відбуватись за наступним сценарієм:

– крок №1: включення двигуна;

– крок №2: переведення першого важеля в положення «зачинено»;

– крок №3: видача деталі;

– крок №4: переведення першого важеля в положення «відчинено»;

– крок №5: видача деталі;

– крок №6: вимкнення двигуна конвеєра.

Виконаємо перший крок і увімкнемо двигун, що рухає конвеєрну стрічку.

Відповідно до таблиці 3.19 двигун вмикається, якщо на вихідний контакт з адресою 00101 надходить логічна одиниця. Для управління єдиним вихідним контактом використовується функція 0x05.

Сформуємо команду протоколу Modbus з використанням функції 0x05 для передачі логічної одиниці за адресою 00101. Якщо перевести 00101 в шістнадцятирічний код отримаємо 0x65 (рис. 3.31).

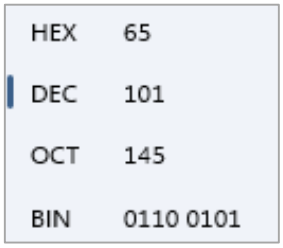


Рисунок 3.31 – Перетворення 00101 в шістнадцятирічний код

При використанні протоколу Modbus всі адреси зменшуються на одиницю, тому при формуванні запиту будемо використовувати адресу вихідного контакту 0x64. Таким чином, остаточний запит має наступний вигляд:

```
01 05 00 64 FF 00.
```

Після виконання даної команди конвеєрна стрічка повинна почати рухатися, а зображення двигуна змінитися та стане помітнішим на фоні програми. Також, навпроти мітки з адресою 00101 повинна з’явитися ознака «On» (рис. 3.32).

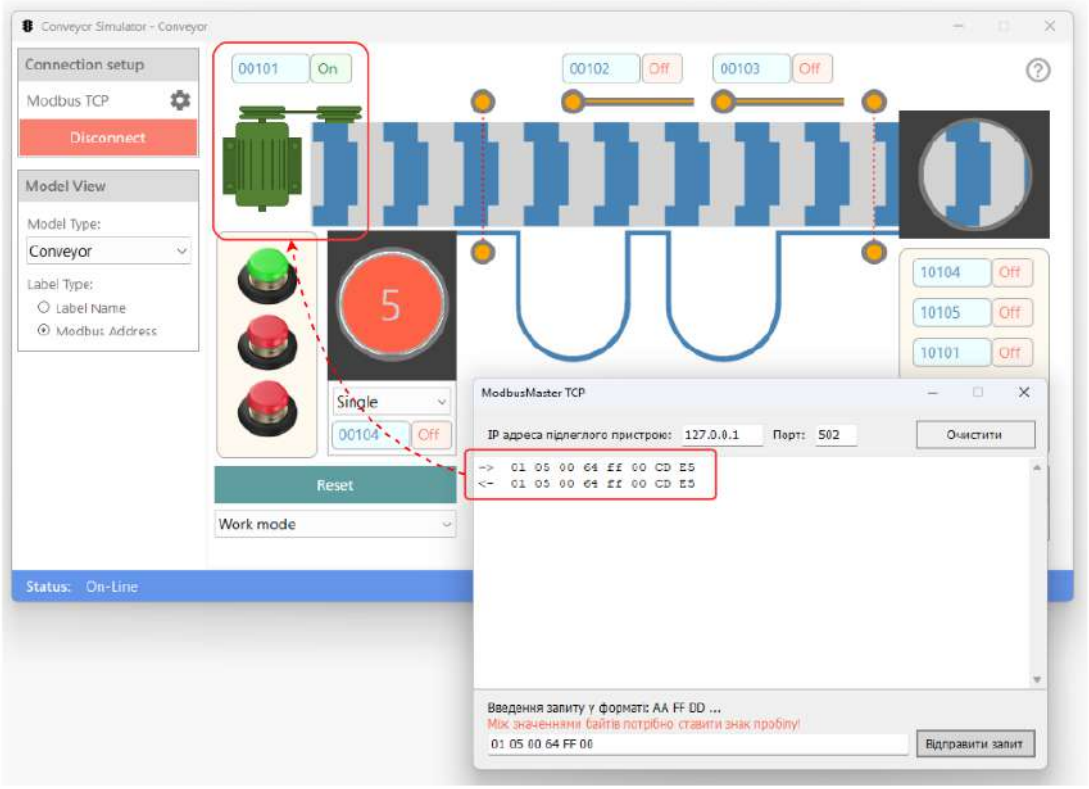


Рисунок 3.32 – Виконання кроку №1, увімкнення двигуна

Наступним кроком виконаємо переведення першого важеля в положення «зачинено». Для цього використовуємо також функцію 0x05 та записуємо до вихідного контакту з адресою 00102 логічну одиницю:

```
01 05 00 65 FF 00.
```

Результат виконання команди подано на рис. 5.34. Як можна бачити з даного рисунку, після виконання даної команди перший важіль перекриває рух конвеєрної стрічки під кутом 45 градусів, а навпроти мітки з адресою 00102 з’явилася ознака «On» (рис. 3.33).

Відповідь від підлеглого пристрою повністю ідентична запиту. Це свідчить про правильність виконання запиту.

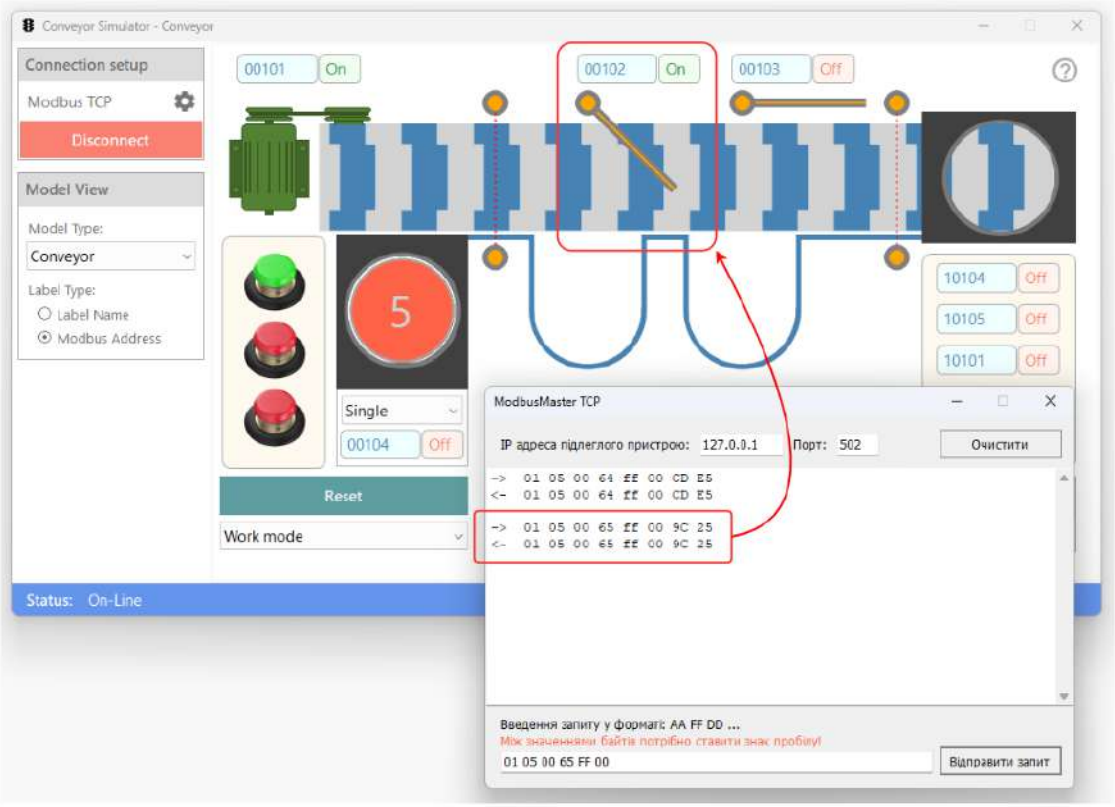


Рисунок 3.33 – Увімкнення першого важеля

На третьому кроці необхідно видати одну деталь з магазину. Магазин деталей має адресу 00104. Для видачі тільки однієї деталі за вказаною адресою необхідно спочатку записати логічну одиницю та не пізніше 2,5 секунди записати логічний нуль.

Таким чином, спершу до віртуального макету буде передано команду:

```
01 05 00 67 FF 00
```

а далі, не пізніше, чим за 2,5 секунди, передається команда вимкнення штоку магазину:

```
01 05 00 67 00 00
```

Результат виконання команди подано на рис. 3.34.

З даного рисунку видно, що деталь, рухаючись конвеєрною стрічкою, змінила свій напрям та потрапила до першого накопичувача. Кількість деталей в магазині зменшилася до 4, а в першому накопичувачі зросла до 1.

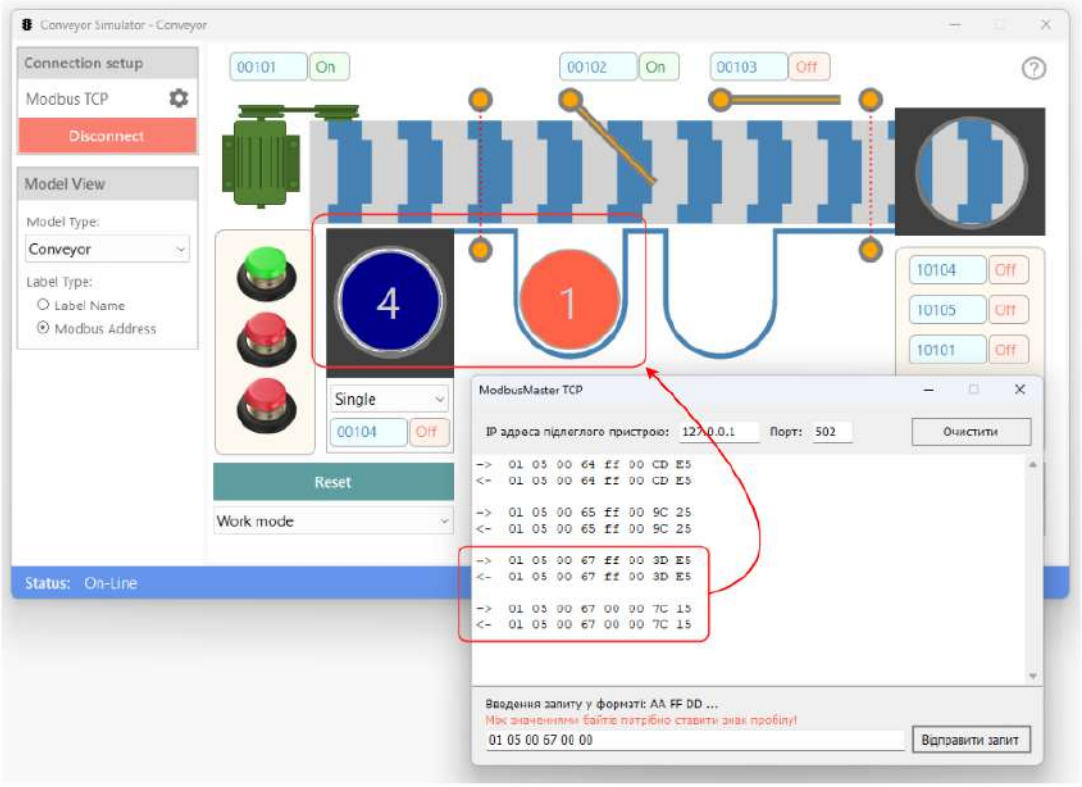


Рисунок 3.34 – Видача першої деталі

Четвертим кроком виконаємо повернення першого важеля в початкове положення «відчинено»:

```
01 05 00 65 00 00
```

На п’ятому кроці виконаємо видачу ще однієї деталі на лінію за допомогою послідовності команд:

```
01 05 00 67 FF 00
01 05 00 67 00 00
```

На останньому кроці виконаємо зупинку конвеєра, передавши команду:

```
01 05 00 64 00 00
```

Результат виконання послідовності кроків 4-6 поданий на рис. 3.35.

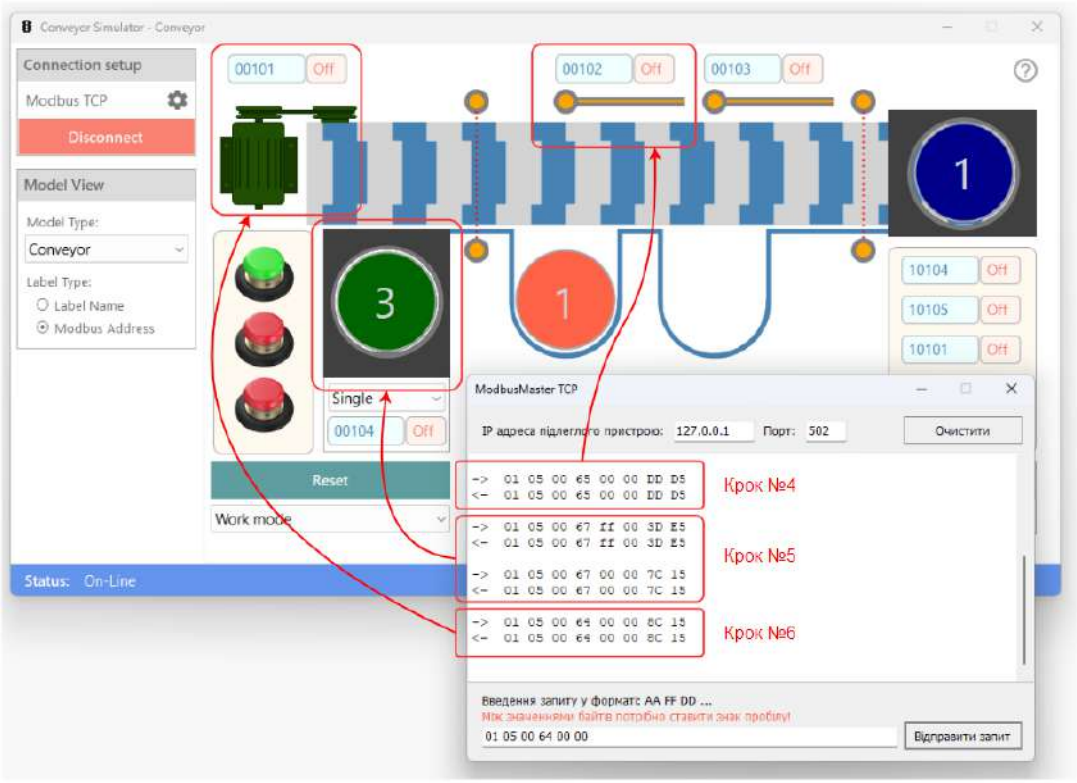


Рисунок 3.35 – Результат виконання послідовності кроків 4-6

На даному етапі можна бачити, що в магазині та в накопичувачах знаходиться певна кількість деталей. Для того, щоб на віддаленому пристрої дізнатися про поточне розподілення деталей на виробничій лінії потрібно виконати читання регістрів зберігання, що пов’язані з кожним із зазначених накопичувачів.

Сформуємо команду для читання даних з чотирьох регістрів зберігання з адресами 40101-40104. Для цього використовуємо функцію Modbus 0x03. В запиті потрібно вказати:

– адреса підлеглого пристрою (01);

– код функції (03);

– адреса першого байта, Hi байт (00);

– адреса першого байта, Lo байт (64);

– число регістрів для читання, Hi байт (00);

– число регістрів для читання, Lo байт (04);

– контрольна сума CRC.

Таким чином, запит матиме наступний вигляд:

```
01 03 00 64 00 04
```

Результат виконання запиту подано на рис. 3.36.

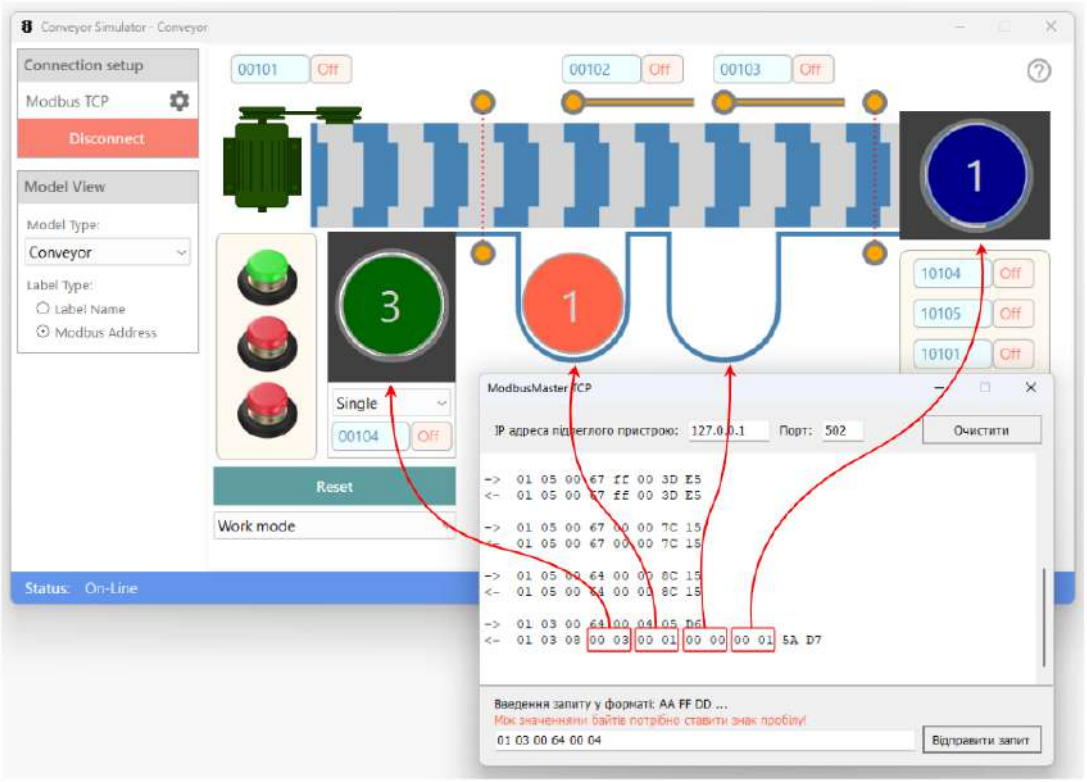


Рисунок 3.36 – Результат виконання запиту читання вмісту регістрів зберігання віртуального макета

Якщо розібрати отриману відповідь:

```
01 03 08 00 03 00 01 00 00 00 01 5A D7
```

то у відповідності до таблиці розподілення адресів регістрів, можна побачити наступну інформацію:

– регістр 40101, що відповідає магазину деталей, містить значення 0x0003;

– регістр 40102, що відповідає першому накопичувачу, містить значення 0x0001;

– регістр 40103, що відповідає другому накопичувачу, містить значення 0x0000;

– регістр 40104, що відповідає кінцевому накопичувачу, містить значення 0x0001.

Таким чином, отримані дані повністю відповідають реальному розподіленню деталей на виробничій лінії.

3.6 Доступ до промислового обладнання за допомогою протоколу Modbus

3.6.1 Основні вузли для роботи з протоколом Modbus

Node-RED надає можливість працювати з протоколом Modbus. Для цього використовується додатковий пакет «node-red-contrib-modbus». Пакет «node-red-contrib-modbus» містить ряд вузлів для роботи з протоколом Modbus:

– Modbus-Server, що реалізує серверну частину протоколу Modbus і дозволяє зчитувати та записувати дані на пристрої Modbus;

– Modbus-Client, що реалізує клієнтську частину протоколу Modbus і дозволяє зчитувати та записувати дані з пристроїв Modbus;

– Modbus-Read дозволяє зчитувати дані з пристроїв Modbus за допомогою функції читання (read function) протоколу Modbus;

– Modbus-Write дозволяє записувати дані на пристрої Modbus за допомогою функції запису (write function) протоколу Modbus;

– Modbus-Flex-Getter дозволяє зчитувати дані з пристроїв Modbus, з підтримкою різних форматів даних та додаткових параметрів;

– Modbus-Flex-Writer дозволяє записувати дані на пристрої Modbus, з підтримкою різних форматів даних та додаткових параметрів;

– Modbus-Response: дозволяє розбирати та створювати відповіді на запити до пристроїв Modbus.

Ці вузли забезпечують основні функції роботи з протоколом Modbus в середовищі Node-RED. Також пакет «node-red-contrib-modbus» містить додаткові вузли для більш специфічних задач, наприклад, Modbus-RTU та Modbus-TCP.

Реалізація можливості роботи з протоколом Modbus здійснюється шляхом додавання нового компонента до палітри інструментів:

```
npm install node-red-contrib-modbus,
```

або:

```
sudo npm install -g node-red-contrib-modbus
```

для глобального встановлення.

В результаті будуть додані наступні компоненти Modbus (рис. 3.37).

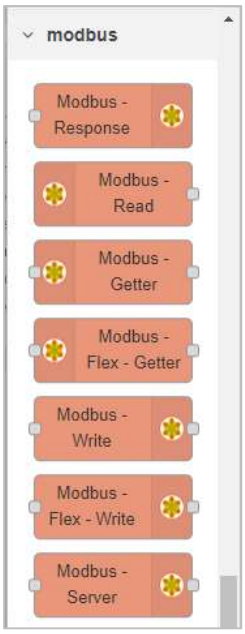


Рисунок 3.37 – Компоненти для роботи з Modbus в Node-RED

3.6.2 Робота Node-RED в режимі тестового сервера Modbus

Node-RED може працювати, як в режимі сервера, так і в режимі клієнта.

В режимі сервера Node-RED виконує функцію підлеглого (веденого) пристрою. Режим клієнта означає, що даний пристрій буде виконувати функції майстра мережі та саме він зможе керувати роботою підлеглих.

Node-RED може бути налаштований для роботи в режимі сервера Modbus за допомогою вузла Modbus-Server з пакету node-red-contrib-modbus. Цей вузол дозволяє створити сервер Modbus, який може отримувати запити на зчитування та запис даних від клієнтів, які працюють з пристроями Modbus. Вигляд вузла Modbus-Server подано на рис. 3.38.

Рисунок 3.38 – Вигляд вузла Modbus-Server

Даний вузол може використовуватись для тестування сервера Modbus TCP на основі node-modbus (jsmodbus), а також, як самостійний пристрій, що виконує команди майстра мережі.

Після активації за допомогою вузла «injection» сервер надсилає вміст буфера на окремі виходи:

– вихід 1: буфер регістрів зберігання (holding register);

– вихід 2: буфер вихідних дискретних контактів (coils);

– вихід 3: буфер вхідних регістрів (input register);

– вихід 4: буфер вхідних дискретних контактів (discrete input).

Змінювати вміст регістрів можна за допомогою функції запису Modbus-Write. Зчитувати поточне значення з регістрів можна за допомогою вузла Modbus-Read. Також можна задавати попереднє значення в регістрах безпосередньо з вхідного повідомлення, що передається на вхід. Для цього потрібно використовувати функціональний вузол між вузлами injection та Modbus-Server (рис. 3.39).

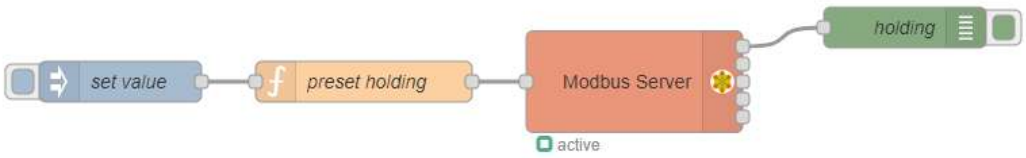


Рисунок 3.39 – Запис початкового значення в регістр зберігання

Параметри налаштування вузлів подано на рис. 3.40. Для налаштування вузла Modbus-Server необхідно вказати наступні параметри:

– «Hostname»: IP-адреса пристрою, на якому запущено сервер;

– «Port»: номер порту, на якому буде запущений сервер Modbus (за замовчуванням для даного вузла – це 10502);

– «Coils»: список дискретних виходів (coil), які можна зчитувати та записувати;

– «Input Registers»: список вхідних регістрів, які можна зчитувати;

– «Holding Registers»: список регістрів утримування, які можна зчитувати та записувати;

– «Discretes Input»: список дискретних входів, стан яких можна зчитувати.

Після налаштування вузла Modbus-Server необхідно зберегти та запустити потік. Після запуску сервера Node-RED буде очікувати запити від клієнтів Modbus та відповідати на них, відповідно до налаштувань вузла Modbus-Server.

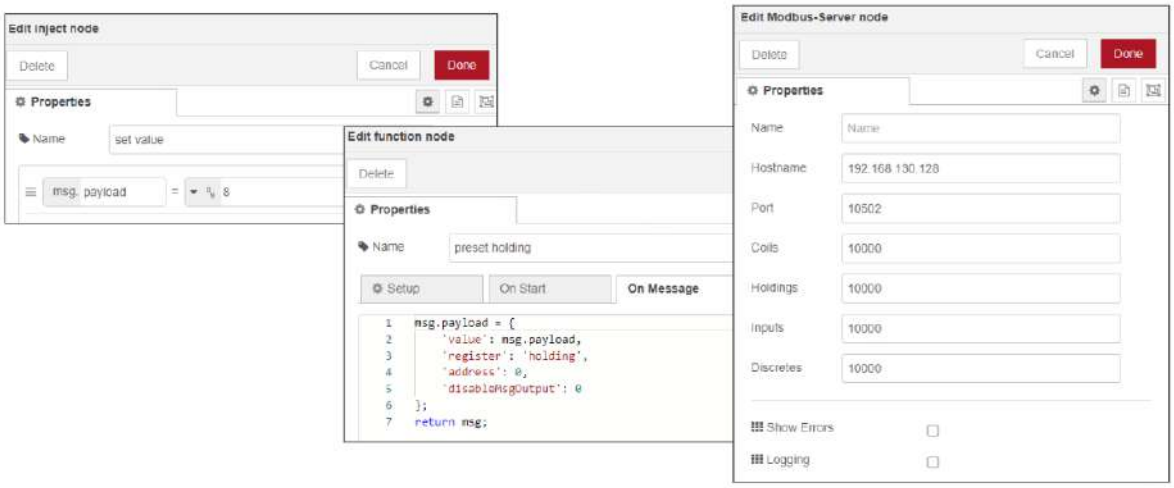


Рисунок 3.40 – Параметри налаштування вузлів для запису початкового значення в регістр зберігання

Запис значення в регістр виконується за допомогою наступного коду:

```
msg.payload = {
    ‘value’: msg.payload,
    ‘register’: ‘holding’,
    ‘address’: 0,
    ‘disableMsgOutput’: 0
};
return msg;
```

У цьому коді властивість payload змінюється на об’єкт, який містить наступні ключі:

– ‘value’: встановлює значення для відправки з повідомлення, що надійшло від попереднього вузлу, наприклад inject;

– ‘register’: вказує тип регістра Modbus, який буде використовуватись для збереження даних. У цьому випадку використовується регістр holding, але можна використовувати такі типи: coils, input та discrete;

– ‘address’: встановлює адресу регістру, до якої записуватимуться дані;

– ‘disableMsgOutput’: вказує, чи повинен вузол Modbus відправляти повідомлення з результатом операції. У цьому випадку вказано значення 0, що означає, що повідомлення відправлятиметься.

Для дослідження роботи вузла Modbus-Server створимо наступну структуру вузлів (рис. 3.41).

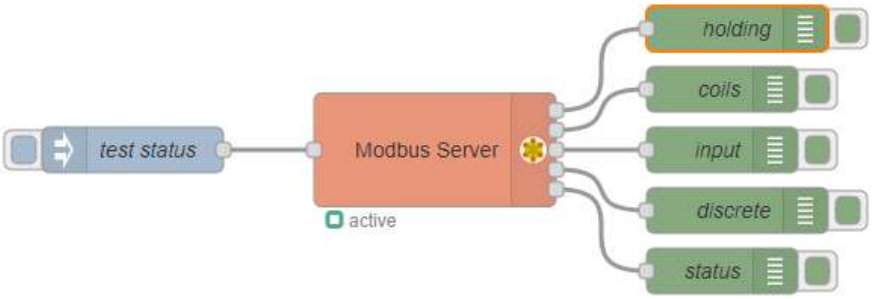


Рисунок 3.41 – Структура потоку для дослідження роботи вузла Modbus-Server

В даному потоці, вузол inject передає будь-яке повідомлення для ініціалізації роботи вузла Modbus-Server. Наприклад, може передаватися «timestamp» (рис. 3.42).

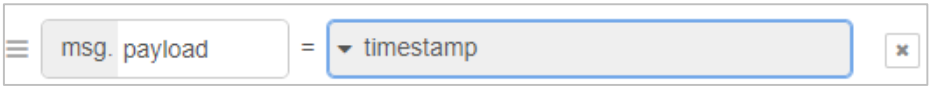


Рисунок 3.42 – Налаштування вузла inject

До виходів сервера підключимо п’ять вузлів debug для контролю за станом усіх внутрішніх буферів. Щоб записати або прочитати значення з будь-якого регістру скористаємось програмою «Modbus_Master_TCP». В налаштуваннях потрібно вказати IP-адресу того комп’ютера, на якому завантажено Node-RED. Наприклад, виконаємо підключення до ПК з адресою 192.168.130.128 (рис. 3.43).

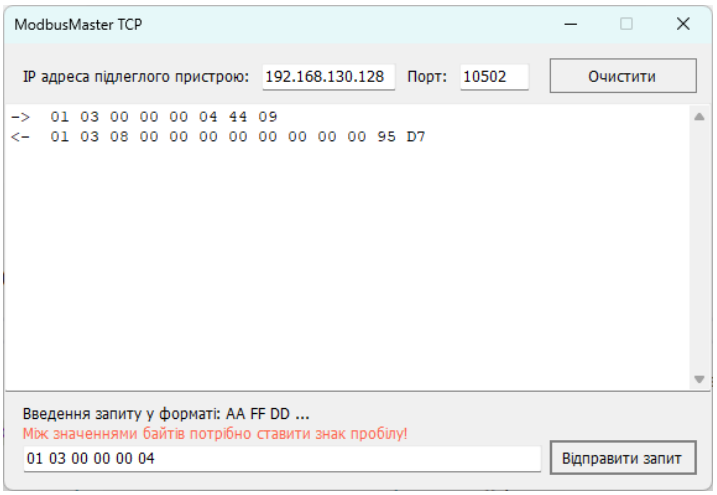


Рисунок 3.43 – Підключення до сервера Node-RED

З даного рисунку також можна бачити, що номер порту сервера дорівнює 10502, що відповідає налаштуванням вузла Modbus-Server.

На сервер було передано запит на читання стану чотирьох регістрів зберігання, починаючи з адреси «00 00», та у відповідь отримано порожнє значення, тому що при старті сервера всі комірки буферу не заповнені та мають нульове значення (рис. 3.44).

Рисунок 3.44 – Стан регістрів сервера Modbus

Виконаємо запис до чотирьох регістрів зберігання такі значення: 43, 123, 255, 85, починаючи з першого регістру. Для цього відправимо на сервер команду:

```
01 10 00 00 00 04 08 00 2b 00 7b 00 ff 00 55
```

Також, для прикладу, увімкнемо першій та третій вихідні контакти за допомогою команди:

```
01 0f 00 00 00 03 01 05
```

На рис. 3.45 подано послідовність виконання команд та відповіді від сервера Node-RED. В результаті виконання запитів на сервері було змінено вміст відповідних буферів (рис. 3.46).

З поданого рисунку можна бачити, що в залежності від відправленої команди сервер виконав зміну регістрів, що зберігаються за різними адресами.

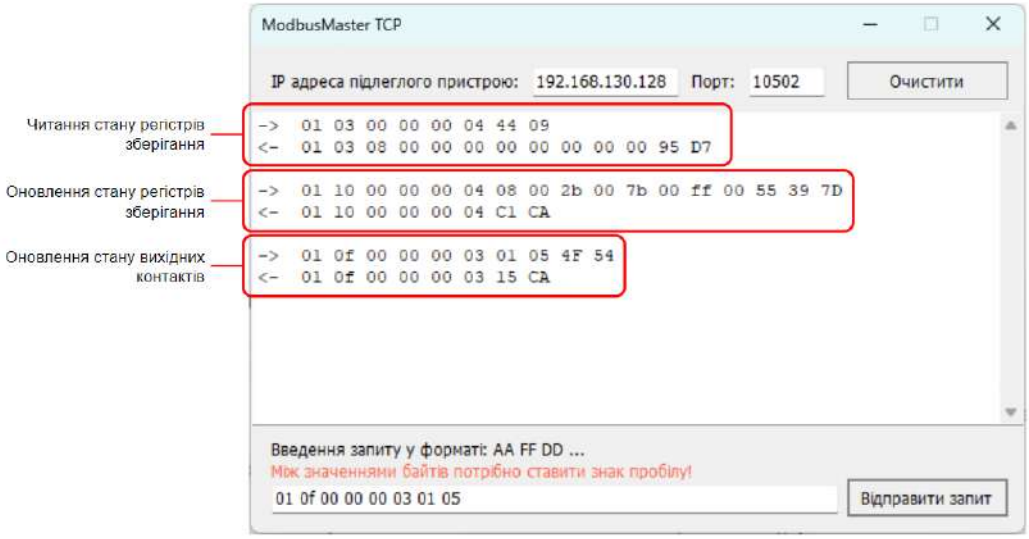


Рисунок 3.45 – послідовність виконання команд та відповіді від сервера Node-RED

Також, необхідно відмітити, що регістри зберігання 16-розрядні і на екрані представлені двома байтами в масиві значень відповідного буферу.

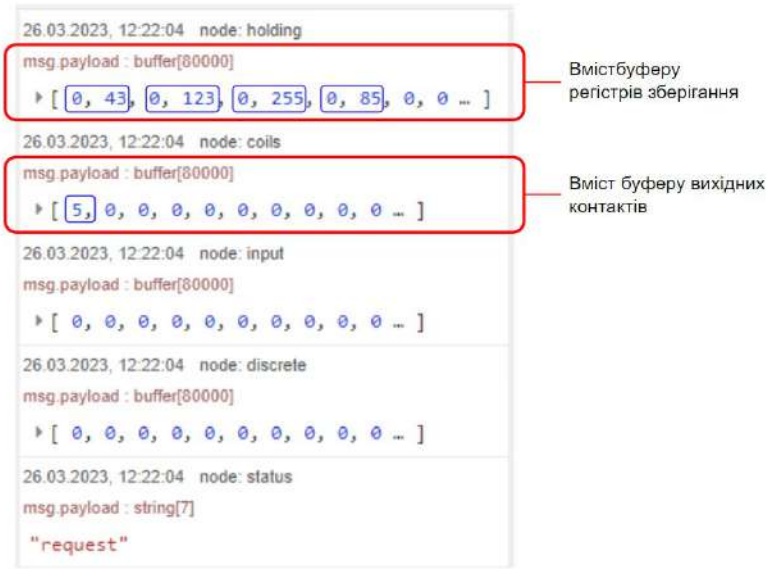


Рисунок 3.46 – Вміст відповідних буферів регістрів сервера Modbus

Вихідні контакти однобітні, тому на екрані вони представленні у вигляді масиву байтів, кожен з яких вміщує інформацію про вісім окремих вихідних контактів. Наприклад, в нашому випадку увімкнення першого і третього контактів записано у вигляді числа «05», що є поданням послідовності 00000101 у HEX-форматі.

3.6.3 Конфігурація Node-RED для роботи з Modbus

Важливо зазначити, що для роботи в режимі сервера Modbus необхідно мати досвід роботи з протоколом та розуміти основні поняття, такі як адреса вузла, функції зчитування та запису даних, формати даних та інші. Також необхідно враховувати технічні характеристики пристроїв Modbus, з якими працює сервер, та дотримуватись рекомендацій виробників щодо налаштування цих пристроїв.

Вузли отримання даних за допомогою протоколу Modbus та вузли запису вимагають, щоб спершу було налаштовано сервер для підключення до мережі. Параметри для цього сервера зберігаються в конфігурації та доступні для всіх вузлів у робочому просторі. Налаштувати сервер можна безпосередньо з вікна зміни властивостей будь-якого вузла Modbus (рис. 3.47).

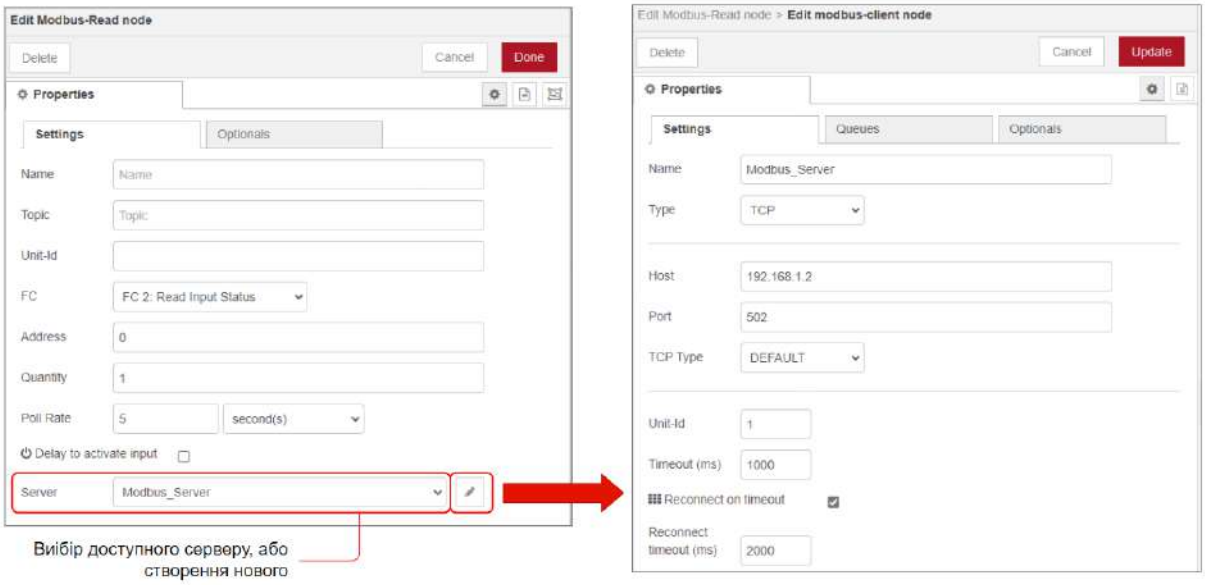


Рисунок 3.47 – Налаштування сервера у вікні зміни властивостей вузла читання даних

Для налаштування сервера необхідно вказати наступні параметри:

– «Port»: номер порта, на якому буде запущений сервер Modbus;

– «Hostname»: IP-адреса пристрою, на якому запущено сервер;

– «Type»: тип протокола (Modbus TCP або Modbus RTU);

– «Unit ID»: ідентифікатор пристрою.

Якщо в якості сервера Modbus використовується віддалений пристрій або віртуальний прилад, номер порту буде становити 502.

3.6.4 Вузли для реалізації функції запису в регістри Modbus

Функції запису в регістри Modbus в Node-RED представлені двома вузлами Modbus-Write та Modbus-Write-Flex (рис. 3.48).

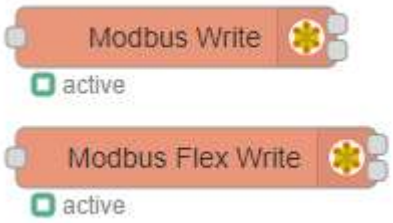


Рисунок 3.48 – Вузли запису в регістри Modbus-Write
та Modbus-Write-Flex

При використанні будь-якого вузла, що працює із пристроями за протоколом Modbus, необхідно спочатку налаштувати підключення до віддаленого або віртуального пристрою.

Основна відмінність між вузлами Modbus-Write та Modbus-Write-Flex полягає в тому, що Modbus-Write дозволяє записувати лише одне значення за раз, тоді як Modbus-Write-Flex може записувати кілька значень одночасно завдяки використанню гнучкого підходу до налаштування параметрів передавання. Це вимагає застосування додаткового функціонального вузлу (function) перед Modbus-Write-Flex.

Для Modbus-Write значення для запису передається через властивість payload, а регістр та адреса запису вказуються в налаштуваннях вузла. У свою чергу, Modbus-Write-Flex дає можливість вибрати декілька адрес для запису та відповідні значення через об’єкт JSON, що передається через властивість payload.

Крім того, в Modbus-Write-Flex можна працювати з додатковими параметрами, які дозволяють встановити тип і порядок байтів для передачі даних на пристрій Modbus. Це важливо, оскільки у різних пристроях Modbus можуть використовуватись різні типи та порядок байтів. Це також робиться в коді функції.

Приклад вікна налаштування вузла Modbus-Write подано на рис. 3.49.

В даному вікні необхідно вказати:

– назву вузла (Name);

– ідентифікатор (адресу) підлеглого пристрою (Unit-id);

– тип функції, за допомогою якої здійснюється запис значень в регістри Modbus (FC);

– адресу регістра, з якого починається відлік;

– сервер для поєднання з віддаленим пристроєм.

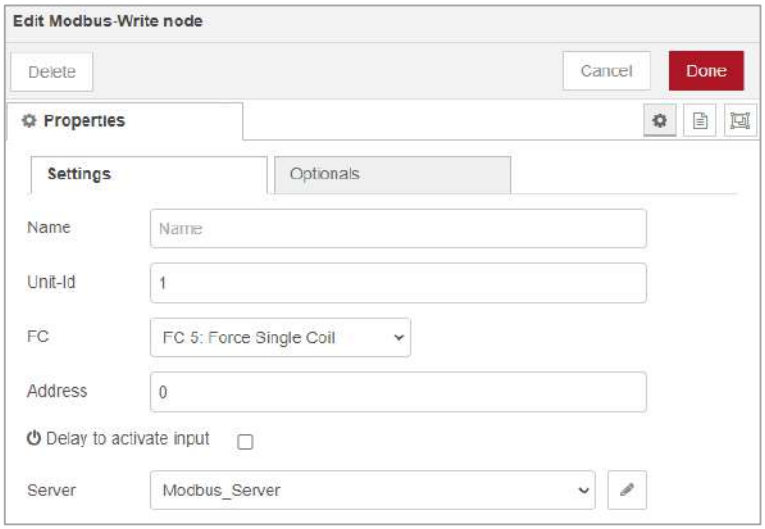


Рисунок 3.49 – Приклад вікна налаштування вузла Modbus-Write

Можливі варіанти обрання типів функцій (рис. 3.50):

– Force Single Coil (FC5): використовується для запису біта в регістр Coil на пристрої Modbus;

– Preset Single Register (FC6): використовується для запису одного значення в один регістр Modbus;

– Force Multiple Coils (FC15): використовується для запису бітів у вихідні дискретні регістри Coil на пристрої Modbus;

– Preset Multiple Registers (FC16): використовується для запису декількох значень в декілька регістрів Modbus.

Рисунок 3.50 – Можливі варіанти вибору типів функцій

При виборі функції для вузла Modbus-Write, важливо враховувати тип даних, який потрібно записати, та тип регістрів Modbus, які підтримуються пристроєм.

Дані, які треба записати в регістр Modbus передаються на вхід від додаткового вузла, наприклад, Inject.

В якості прикладу розглянемо потік вузлів, що дозволяє керувати жовтим сегментом світлофора віртуального макета в ручному режимі. Нехай, за умовами задачі, необхідно реалізувати такі функції:

– натискання першої кнопки призводить до запалювання жовтого сегмента;

– натискання другої кнопки призводить до згасання жовтого сегмента світлофора.

Приклад поєднання вузлів подано на рис. 3.51. В якості кнопок в даному прикладі будемо використовувати вузли Inject.

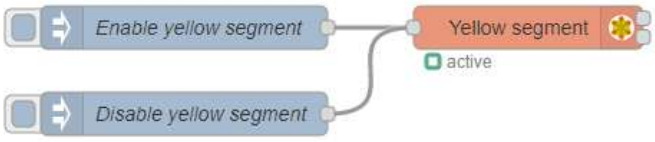


Рисунок 3.51 – Приклад поєднання вузлів для управління жовтим сегментом світлофора в ручному режимі

Налаштування вузлів Inject подано на рис. 3.52.

Рисунок 3.52 – Налаштування вузлів Inject

В першому вузлі з назвою «Enable yellow segment» в корисному навантаженні передається число «1». В другому вузлі з назвою «Disable yellow

segment» в корисному навантаженні передається число «0». В обох вузлах тип корисного навантаження обрано – «число».

Налаштування вузла Modbus-Write подано на рис. 3.53.

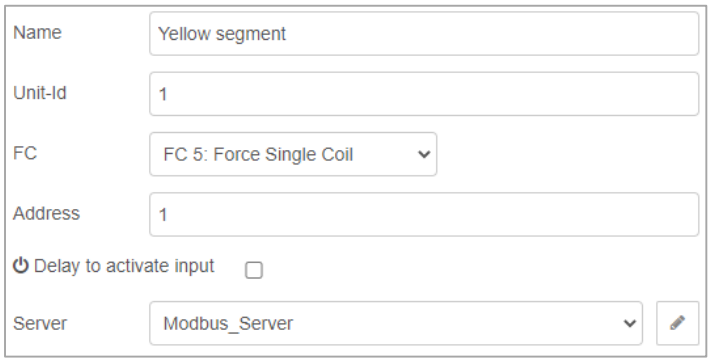


Рисунок 3.53– Налаштування вузла Modbus-Write

В якості функції керування обрано FC5 – зміна стану одного вихідного контакту.

На рис. 3.54 подано стадії надсилання запитів на віддалений пристрій. Для наочності в потік додано вузол Midbus-Responce для відображення відповіді від підлеглого пристрою на відправлений запит. Даний вузол підключений до другого виходу вузла Modbus-Write. Цей вихід призначений саме для виведення статусу відпрацювання запиту.

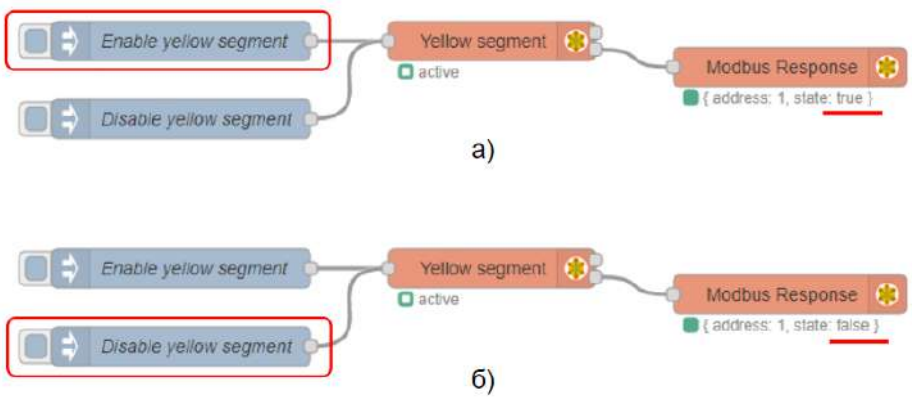


Рисунок 3.54 – Стадії надсилання запитів на віддалений пристрій

На рис. 3.54, а можна бачити, що користувач натиснув на вузол Inject «Enable yellow segment», що призвело до передавання запиту на встановлення логічної одиниці на відповідному дискретному виході. В даному прикладі таким виходом є регістр з адресою «00002».

У відповідь прийнято підтвердження виконання операції зміни стану вихідного контакту – «Address: 1, State: true» (рис. 3.54, а).

При натисканні на вузол Inject «Disable yellow segment» передається запит на встановлення логічного нуля на дискретному виході, що керується. У відповідь прийнято підтвердження виконання операції зміни стану вихідного контакту – «Address: 1, State: false» (рис. 3.54, б).

Приклад роботи віртуального макету світлової колони подано на рис. 3.55.

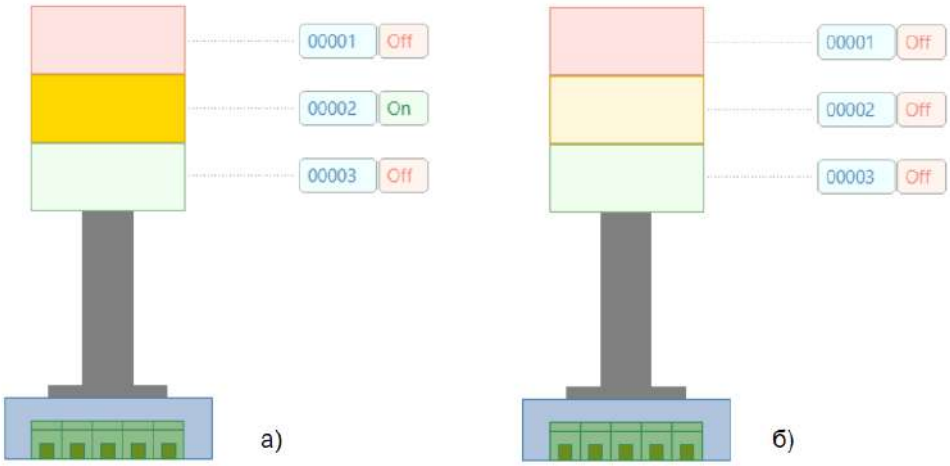


Рисунок 3.55 – Приклад роботи світлофора

Відповідні стадії увімкнення та вимкнення подані на рис. 3.55, а та 3.55, б.

Розглянемо ще один приклад управління світловою колоною з використанням вузла Modbus-Flex-Write. Нехай, необхідно одночасно увімкнути два сегменти – червоний та зелений. Приклад поєднання вузлів подано на рис. 3.56.

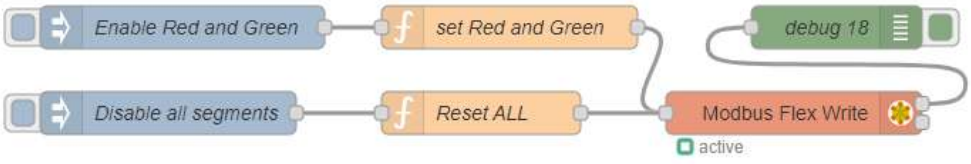


Рисунок 3.56 – Приклад використання вузла Modbus-Flex-Write

В даному прикладі вузлові Modbus-Flex-Write передує два функціональних блоки. Основна логіка роботи програми сконцентрована саме в цих вузлах. Вузол Modbus-Flex-Write має мінімум налаштувань (рис. 3.57).

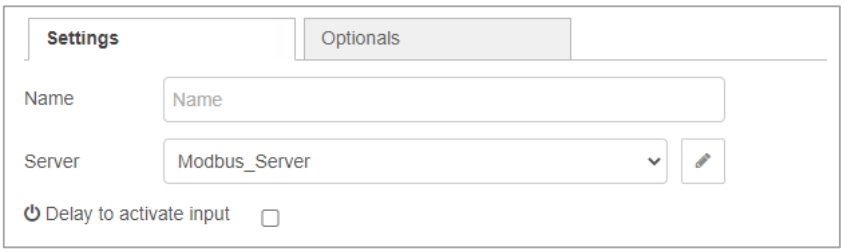


Рисунок 3.57 – Налаштування вузла Modbus-Flex-Write

Єдиним необхідним налаштуванням є вибір сервера Modbus з наявного списку або створення нового.

Щоб включити червоний та зелений сегменти необхідно передати на підлеглий пристрій комбінацію «101» починаючи з адреси «00». Щоб вимкнути всі сегменти необхідно передати «000».

Для функції «set Red and Green» напишемо такий код:

```
var dataArray = [1, 0, 1]; // дані, які потрібно записати
msg.payload = {
    ‘value’: dataArray,
    ‘fc’:15,
    ‘unitid’:4,
    ‘address’:0,
    ‘quantity’:3
}
return msg;
```

Ця функція готує об’єкт повідомлення (msg), який містить інформацію для виконання запису даних (FC15) до вихідних регістрів на пристрої Modbus з ідентифікатором 4.

Запис починається з адреси 00, а кількість виходів, що змінюють свій стан дорівнює 3. Дані, які потрібно записати, зберігаються в масиві dataArray:

```
var dataArray = [1, 0, 1];
```

Результат виконання даної функції при натисканні на вузол «Inject» з назвою «Enable Red and Green» подано на рис. 3.58. Також на даному рисунку подано протокол роботи вузла Modbus-Flex-Write, що виведено у вікно налагодження за допомогою вузла «Debug».

Рисунок 3.58 – Результат виконання функції увімкнення червоного та зеленого сегментів світлофора і протокол роботи вузла Modbus-Flex-Write

Вимкнення всіх сегментів виконується натисканням на вузол «Inject» з назвою «Disable all segments». Вміст функції «Reset ALL» таий:

```
var dataArray = [0, 0, 0]; // дані, які потрібно записати
msg.payload = {
    ‘value’: dataArray,
    ‘fc’: 15,
    ‘unitid’: 4,
    ‘address’: 0,
    ‘quantity’: 3
}
return msg;
```

Відмінність від попередньої функції полягає в масиві dataArray:

```
var dataArray = [0, 0, 0].
```

В масив через кому записуються нульові значення.

3.6.5 Вузли для реалізації функції зчитування з регістрів Modbus

Вузли «Modbus-Read», «Modbus-Getter» та «Modbus-Flex-Getter» в Node-RED призначені для зчитування даних з пристроїв за протоколом Modbus. Однак, є деякі відмінності між цими вузлами.

«Modbus-Read» – це стандартний вузол для зчитування даних з пристроїв Modbus. Він має вбудовану підтримку популярних форматів даних, таких як 16-бітні цілі числа та 32-бітні числа з плаваючою крапкою. Зазвичай цей вузол використовують для зчитування невеликої кількості значень з пристроїв Modbus.

«Modbus-Getter» – це вузол для зчитування даних з пристроїв Modbus, який має додаткові можливості налаштування, що дозволяє зчитувати різні типи даних з пристроїв Modbus. Наприклад, ви можете налаштувати цей вузол для зчитування рядків або бітових значень.

«Modbus-Flex-Getter» – це вузол, який надає ще більше можливостей налаштування для зчитування даних з пристроїв Modbus. Він дозволяє зчитувати дані з декількох різних адрес, вказувати розмір даних, які потрібно зчитати, та налаштовувати порядок байтів для зчитування даних з різними форматами. Також він має вбудовану підтримку кешування даних, що дозволяє зменшити кількість запитів до пристроїв Modbus.

3.7 Контрольні запитання та завдання

1. Що таке протокол Modbus і які його базові поняття?
2. Як організовується обмін даними за протоколом Modbus?
3. Які різновиди протоколу Modbus існують і які їх основні характеристики?
4. У чому полягають основні відмінності між Modbus RTU, Modbus ASCII та Modbus TCP?
5. Яка функція виконує читання вихідних контактів (дискретних виходів) у Modbus і як вона працює?
6. Як реалізується функція читання дискретних входів у Modbus?
7. Як працює функція читання регістрів зберігання у Modbus і які її особливості?
8. Опишіть процес запису одного та декількох регістрів зберігання в Modbus.
9. Як застосовується протокол Modbus для керування віртуальними приладами, зокрема макетом світлової колони та промислового конвеєра?

4 ПРОТОКОЛИ ПРОМИСЛОВОГО ІНТЕРНЕТУ РЕЧЕЙ

4.1 Рівні архітектури IIOT

В процесі розробки засобів автоматизації, що використовуються в мережі IIOT пристроїв, необхідно враховувати особливості побудови даної мережі в концепції Інтернету речей. На рис. 4.1 подані відмінності між традиційними інтернет протоколами та тими, що застосовуються в IIOT.

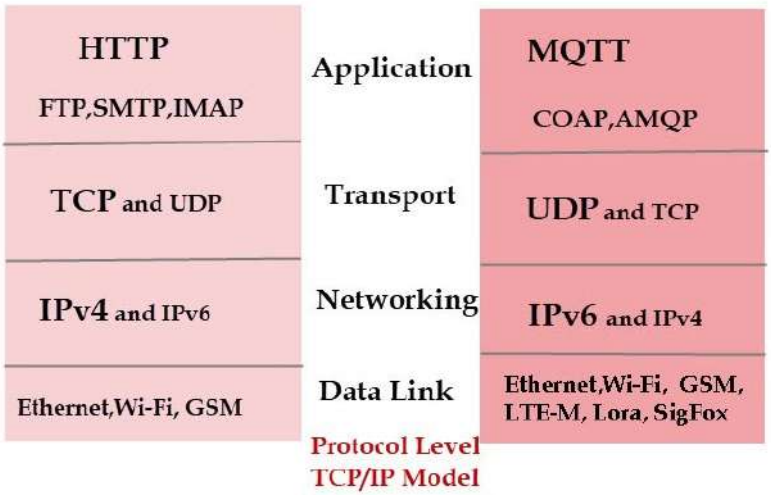


Рисунок 4.1 – Відмінності між традиційними інтернет протоколами та тими, що застосовуються в IIOT

На відміну від типової WEB-системи, що працює за принципом клієнт-серверної програми, модель IIoT-рішень набагато складніша. Аналогічно клієнт-серверному принципу в IIoT-архітектурі можна виділити дві різні за фізичним розташуванням групи обов'язкових компонентів:

– периферія (Edge) – це кінцеві smart-пристрої, розташовані на технологічному обладнанні, за яким здійснюється віддалений моніторинг та керування;

– потужні Big Data інструменти, що розгорнуті у центрі обробки даних на серверах чи в хмарі (Backend).

Тим не менш, через особливості використання IIoT-рішень, пов'язаних з умовами експлуатації та прикладної специфіки, наприклад, екстремальні температури, вібрації, опади, велика довжина каналів передачі даних тощо,

в архітектурі Industrial Internet of Things можна виділити цілих 12 рівнів (шарів) (рис. 4.2).

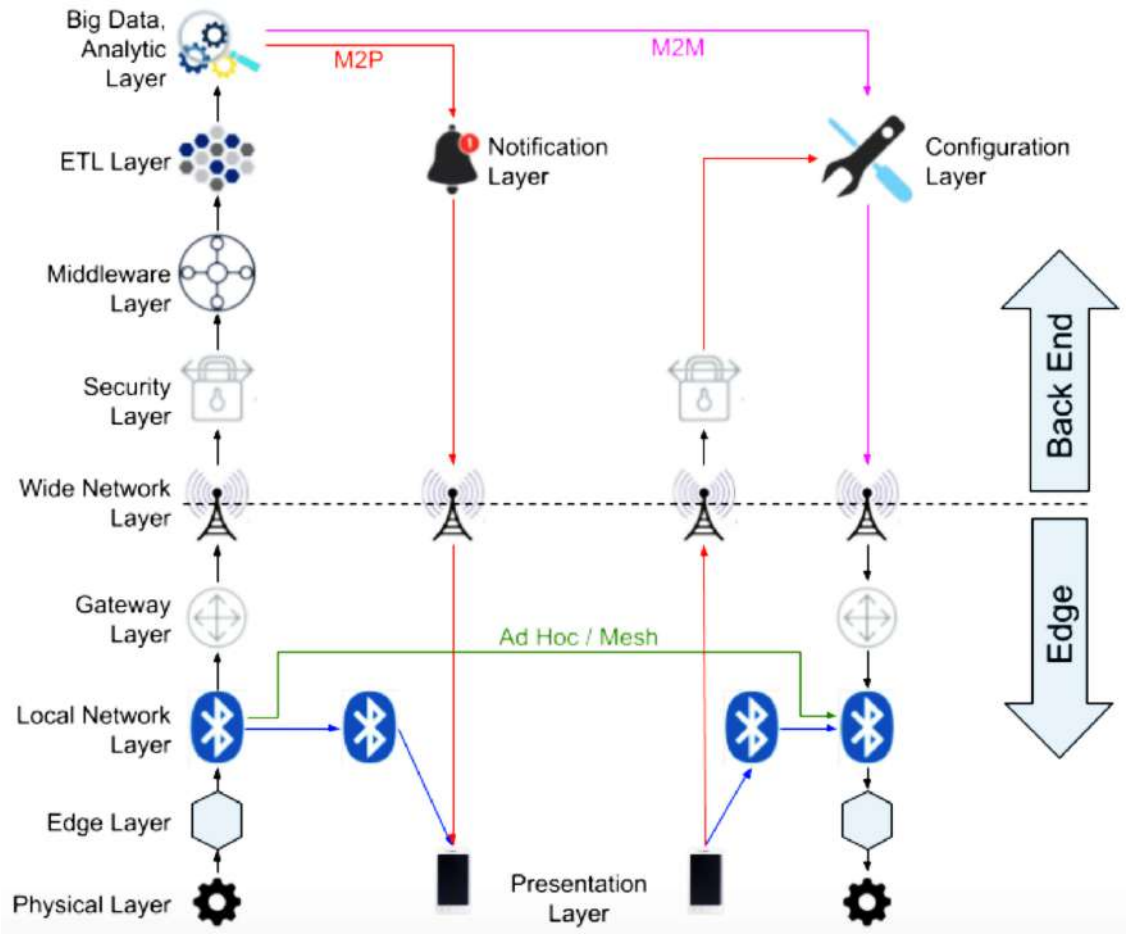


Рисунок 4.2 – Рівні архітектури IIOT

На фізичному рівні (Physical Layer) розташовані кінцеві пристрої – датчики та сенсори, камери та інше обладнання (таймери, п'єзоелементи, мікрофони, фотодіоди/транзистори/резистори, вимикачі, акселерометри, GPS-трекери, баркод-рідери тощо), що збирає інформацію та надсилає її далі, на рівень периферійної обробки даних (Edge Layer).

Периферійні обчислення (Edge Layer) забезпечують мінімальну обробку даних, зокрема перетворення між аналоговим та цифровим поданням інформації (АЦП/ЦАП). Як правило, ці функції реалізуються за допомогою мікроконтролерів Raspberry Pi, Arduino та інші однокристальні системи SoC-модулі (System-on-a-Chip) – електронні схеми, що виконує функції цілого пристрою, в т. ч. бездротові мережі на кристалі (Wireless network-on-chip, WNOC).

Основною вимогою до пристроїв фізичного та периферійного рівнів є низьке енергоспоживання (живлення від батарейки), низька вартість купівлі та експлуатації (безперебійна робота без обслуговування від 1 до 10 років, мінімальні витрати на придбання, встановлення та обслуговування). Для зниження енергоспоживання периферійні пристрої зазвичай мають чотири режими роботи: сон, режим вимірювання та збору інформації з датчиків, режим зв'язку, передачі та отримання інформації, а також режим встановлення та підключення.

Периферійна комунікація (Local Network Layer) застосовується коли дані після первинної обробки вирушають на шлюз (Gateway), який може бути дуже далеко (але не більше кількох кілометрів). Для надійної передачі даних на далекі відстані застосовуються бездротові динамічні мережі, що самоорганізуються: однорангова Ad Hoc і ієрархічна Mesh. Зазвичай при цьому використовуються протоколи ZigBee/Zwave, BLE, LoRa, Proprietary low band, WiFi, Ethernet, CAN, LTE, PLC (рис. 4.3).

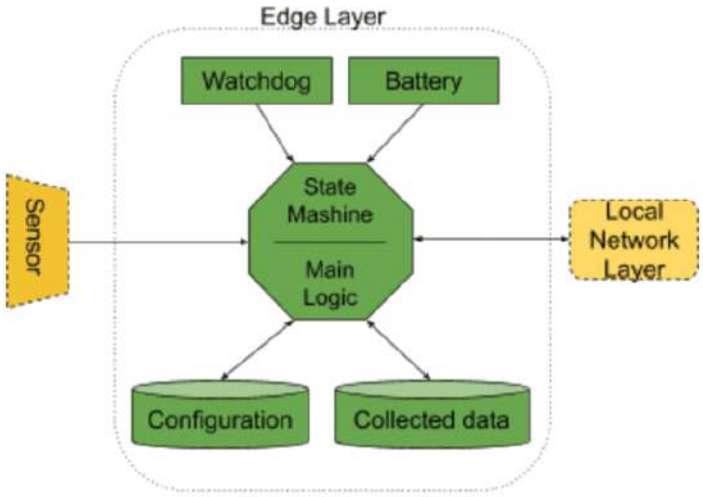


Рисунок 4.3 – Схема периферійного IIoT-пристрою

Шлюз (Gateway Layer), який виконує ETL-операції (Extract, Transform, Load) – отримання, перетворення та збереження інформації з периферійних пристроїв, забезпечує важливі дії у разі критичної ситуації навіть без зв'язку з Backend, а також спілкується із сервером з використанням мобільного бездротового зв'язку (4G/5G) або дротового доступу до Інтернету, надсилаючи туди оброблену інформацію та одержуючи дані конфігурації для кінцевих пристроїв.

Шлюз необхідний IIoT-рішенням, тому що якщо Backend отримуватиме необроблену інформацію, це збільшить його потужність та комунікаційні витрати. Крім того, віддалений Backend-сервер не може гарантувати реакцію в реальному часі для великої кількості периферійних пристроїв, наприклад, камери вуличного спостереження тощо.

Рівень зовнішнього зв'язку (Wide Network Layer) розділяє периферію та BackEnd. Тут використовується стандартизований протокол для IoT-рішень LwM2M (Lightweight M2M, легковажний міжмашинний протокол), розроблений для доступу до кожного периферійного пристрою. Якщо периферійний пристрій не підтримує інтерфейс LwM2M, шлюз вирішить цю проблему, забезпечивши зв'язок з периферією. Рівень зовнішнього зв'язку містить також комунікаційні послуги та моделі ISO, в т. ч. служби балансування та визначення розташування, засновані на DNS-сервісі, транспортний протокол COAP, протокол шифрування з ключами безпеки та сесією встановлення з'єднання DTLS та багато інших компонентів. При цьому DNS використовується як балансувальник навантаження, періодично генеруючи DNS-запит для отримання нової IP-адреси екземпляра рівня безпеки. Крім LwM2M лише на рівні зовнішнього зв'язку також використовуються протоколи PLC, Ethernet, LTE, FOTA/SOTA, Radio.

Рівень безпеки (Security Layer) забезпечує автентифікацію, авторизацію, облік та шифрування/дешифрування на основі ролей та дозволів. Як правило, тут використовуються засоби інформаційної безпеки від хмарних провайдерів, у яких розгорнуті компоненти IIoT-рішень, що агрегують і аналізують, наприклад, Azure Cloud, AWS R53/IAM/EC2 тощо.

Рівень внутрішньосерверного зв'язку (Middleware Layer), що забезпечує внутрішню функціональність балансування навантаження у хмарі, черги повідомлень та передачі потокової інформації на основі мікросервісів або PaaS-рішень від хмарних провайдерів. Компоненти цього шару повинні бути дубльовані та автоматично масштабуватися у зв'язку з нестаціонарним характером передачі даних, щоб забезпечити асинхронну передачу повідомлень з буферизацією та перерозподілом навантаження. Саме на цьому рівні працюють Big Data брокери повідомлень та управління чергами, такі як Apache Kafka або RabbitMQ. Для швидкого завантаження даних із периферії часто використовується платформа обробки подій (повідомлень) Apache NiFi або її спрощена модифікація Apache MiNiFi.

Рівень збору, обробки та зберігання даних (ETL), що реалізується не тільки в периферійних пристроях та шлюзах, а й у Backend, за допомогою збору даних, їх агрегації, уніфікації подання, збереження для подальшого використання, повідомлення інших сервісів про надходження нових даних та загального управління життєвим циклом інформації, у тому числі архівування та знищення. Тут також працюють Big Data-засоби управління чергами повідомлень (Apache Kafka або RabbitMQ) та потокове завантаження даних (Apache NiFi та MiNiFi).

Big Data аналітика, машинне навчання та інші методи штучного інтелекту (Artificial Intelligence, AI). Інструментальні засоби цього рівня найменш стандартизовані та залежать від конкретного випадку. Наприклад, для прогнозування можливих відмов обладнання може використовуватися бібліотека MLLib фреймворків Spark або Flink, для генерації OLAP-кубів та складних SQL-запитів – Spark SQL та FlinkQL відповідно.

Рівень повідомлень (Notification layer), що повідомляє про настання якоїсь події, наприклад, у формі листа електронною поштою, SMS-повідомлення або телефонного дзвінка (у крайньому, екстреному випадку). Для зниження енергоспоживання мобільні програми переходять у сплячий режим, але iOS та Android мають механізм повідомлень, що вказує на прибуття нових даних. Алгоритмічно таке M2P-спілкування (machine to person) реалізується у вигляді моделі «видавець-передплатник», коли клієнтська програма підписується на необхідні події і, у разі їх настання, отримує інформаційний сигнал – повідомлення.

Рівень представлення (Presentation Layer) застосовується коли оброблена інформація від периферійних пристроїв презентується кінцевому користувачеві за допомогою UI/UX-інтерфейсів на екрані комп’ютерного монітора, планшета або мобільного телефону. Рівень подання також відповідає за обслуговування, конфігурацію та зміни стану кожного компонента IIoT-системи, дозволяючи віддалено керувати навіть периферійними пристроями. Поки не існує стандартизованих UI/UX-засобів для рівня представлення IIoT-системи, незважаючи на наявність міжнародних та національних стандартів, які регламентують поняття людино-машинного інтерфейсу, що широко використовуються в SCADA-системах.

Рівень конфігурацій (Configuration Layer) забезпечує сховище статусів периферійних пристроїв (актуальний стан, новий стан, який буде завантажено та проміжний статус, що вказує на процес оновлення стану). Це необхідно через

особливості зв’язку периферії та шлюзу з Backend-сервером, тому що комунікація кінцевих пристроїв з хмарою тримається не завжди, а періодичними сеансами, під час яких відбувається відправлення та отримання даних.

4.2 Основні протоколи організації зв’язку в IIOT

Серед багатьох протоколів, що застосовуються для організації обміну повідомленнями в IIOT можна виділити декілька тих, що застосовуються найчастіше. До них відносяться:

– DDS OMG;
– AMQP;
– MQTT;
– JMS;
– REST;
– CoAP;
– XMPP.

Data Distribution Service (DDS) – це протокол проміжного програмного забезпечення та стандарт API для з’єднання, орієнтованого на дані, від Object Management Group (OMG). Він об’єднує компоненти системи разом, забезпечуючи підключення до даних з низькою затримкою, надзвичайну надійність і масштабовану архітектуру, яка потрібна для бізнесу та критично важливих додатків Інтернету речей (IoT). Це протокол мережевої комунікації, розроблений для використання в розподілених системах реального часу (Real-Time Distributed Systems). DDS забезпечує механізми розподілу даних та спільного використання даних між різними системами та процесами в реальному часі.

Даний протокол був розроблений з урахуванням вимог, що ставляться до систем реального часу, де час від реакції на події може залежати від мікросекунд до мілісекунд. DDS надає механізми забезпечення надійності та якості обслуговування (Quality of Service), таких як маршрутизація повідомлень, контроль доступу та механізми синхронізації.

DDS використовує поняття тем, що дозволяє підписникам підписуватися на теми, які вони хочуть отримувати, і видавцям використовувати ці теми для відправки повідомлень (рис. 4.4). Протокол також дозволяє підписникам створювати власні теми та публікувати на них повідомлення.

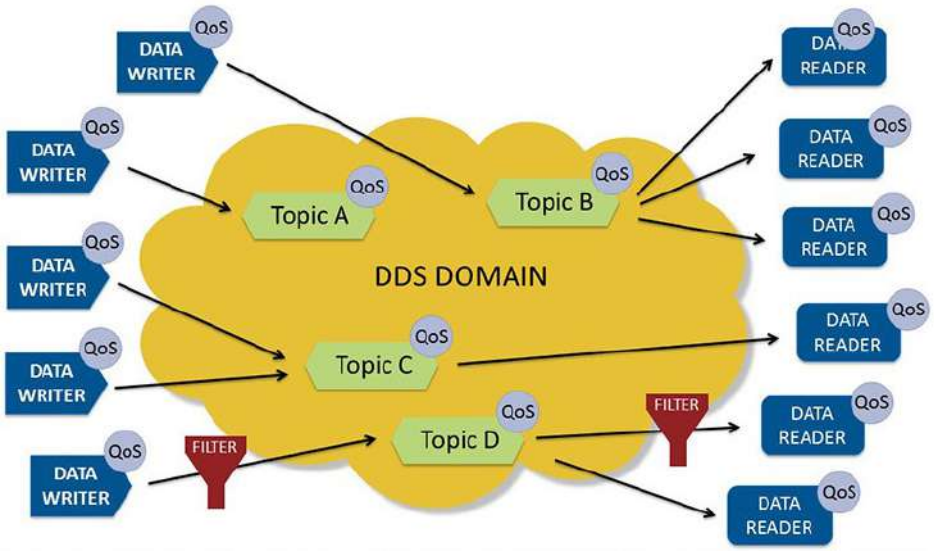


Рисунок 4.4 – Контрольований обмін даними за допомогою протоколу DDS

DDS забезпечує контрольований Qos обмін даними. Програми спілкуються, публікуючи теми та підписуючись на них. У підписках можна вказати фільтри часу та вмісту та отримати лише частину даних, які публікуються в темі. Різні домени DDS повністю незалежні один від одного. Немає спільного використання даних між доменами DDS.

DDS широко використовується в різних галузях, включаючи авіацію, оборонну та космічну промисловість, медицину, телекомунікації та виробничі системи.

AMQP (Advanced Message Queuing Protocol) – це відкритий стандарт протоколу передачі повідомлень, який призначений для взаємодії між різними програмними системами. Протокол забезпечує безпеку, надійність та масштабованість передачі повідомлень.

Цей протокол був створений для вирішення проблеми інтеграції між різними протоколами, платформами та мовами програмування. Він дозволяє відправляти та отримувати повідомлення з одного додатка в інший, що дозволяє програмістам легко створювати розподілені системи.

AMQP пропонує поведінку провайдера повідомлень і клієнта в тій мірі, що рішення від різних постачальників справді сумісні. Він є двійковим протоколом рівня програми, призначеним для ефективної підтримки широкого спектру програм для обміну повідомленнями та патернів (шаблонів). Він забезпечує потік контрольованих повідомлень з гарантіями доставки

повідомлень, такі як AT-MOST-ONCE (де кожне повідомлення доставляється один раз або ніколи), AT-LIST-ONCE (де кожне повідомлення буде доставлено, але може виявитися так, що це відбудеться декілька разів) і EXACTLY-ONCE (коли повідомлення буде завжди доставлено тільки один раз) та автентифікацію або шифрування на основі SASL та TLS. Це передбачає використання надійного протоколу транспортного рівня, як-от протокол управління передачею (TCP).

Брокер (або сервер) відіграє вирішальну роль у активації протоколу AMQP. Він відповідає за створення з'єднань, які забезпечують кращу маршрутизацію даних і чергування на стороні клієнта. Робота з формування черг і підтвердження повідомлень виконується споживачем.

Спрямування даних, отриманих з бірж, і розміщення їх у чергах здійснюється виробником (рис. 4.5).

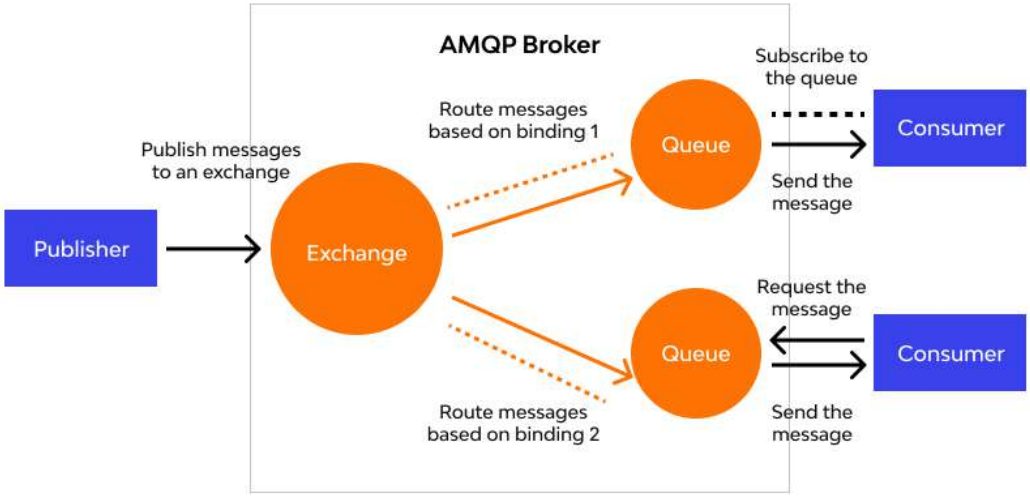


Рисунок 4.5 – Принцип взаємодії компонентів за протоколом AMQP

Протокол AMQP складається з наступних компонентів:

– клієнт – програма, що відправляє та отримує повідомлення;

– брокер повідомлень – посередник, який приймає повідомлення від клієнтів та передає їх до інших клієнтів. Брокер може зберігати повідомлення у черзі до того часу, поки вони не будуть оброблені;

– повідомлення – носій інформації, який передається між клієнтами;

– канали – пов'язують клієнтів з брокером та дозволяють розділити потік повідомлень на різні теми;

– правила маршрутизації – правила, що вказують як повідомлення має бути доставлено до клієнта. Можна налаштувати правила маршрутизації, щоб направляти повідомлення на певний канал або до певного клієнта;

– моделі повідомлень – моделі, що вказують як повідомлення має бути оброблено. Наприклад, можна використовувати модель «request-response», щоб відправити запит та отримати відповідь від іншого клієнта.

Ці компоненти дозволяють програмістам легко створювати розподілені системи, які можуть передавати повідомлення між різними компонентами. Крім того, AMQP є відкритим стандартом, що дозволяє різним системам взаємодіяти між собою без обмежень з боку вендора.

Протокол MQTT (Message queuing telemetry support) визначається як міжмашинний протокол з низьким споживанням пропускної здатності, який допомагає пристроям IoT спілкуватися один з одним, з мінімальними вимогами до коду та площею мережі (рис. 4.6).

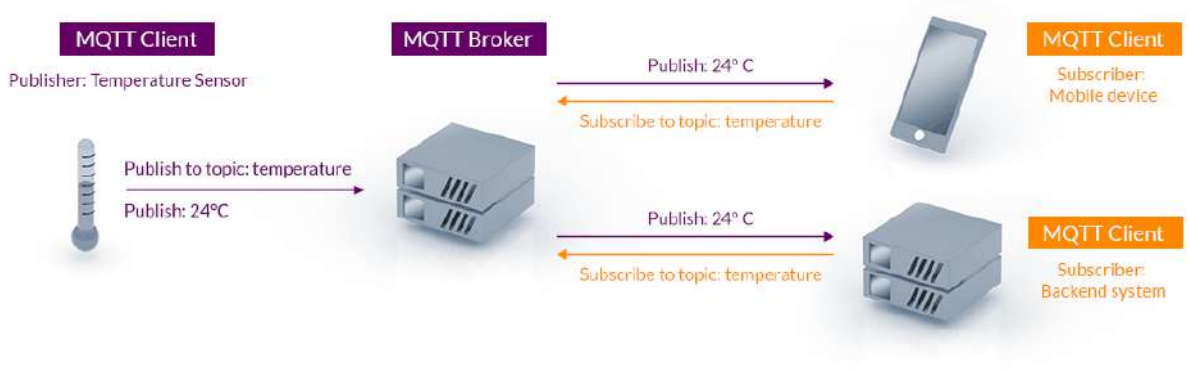


Рисунок 4.6 – Принцип взаємодії між пристроями за протоколом MQTT

MQTT – це розумне рішення для бездротових з’єднань із різними рівнями затримки через періодичні обмеження широкосмугового зв’язку або ненадійні з’єднання. Він підходить для підключення пристроїв з невеликим кодом. Стандарт використовується в багатьох галузях промисловості, включаючи автомобілі, енергетику та зв’язок.

Message Queuing Telemetry Transport підходить для зв’язку між машинами (M2M) через його оптимізацію для середовищ із низькою пропускною здатністю та високою затримкою.

MQTT – це протокол видавця/передплатника, яким керує основний брокер. Це означає, що між передавачем і отримувачем немає прямого зв’язку.

Незважаючи на те, що MQTT часто згадують як телеметричний транспорт із чергою повідомлень, спілкування MQTT не використовує черги повідомлень. Оскільки всі отримувачі, зацікавлені в конкретних повідомленнях («позначених темою»), зараховуються як користувачі, джерела інформації публікують її, і всі отримувачі, які бажають одержати конкретні повідомлення («позначені темою»), отримують доступ до цієї інформації. MQTT використовується для всього, від IoT та IIoT до зв'язування хмарних середовищ в IoT та IIoT.

Java Message Service (JMS) – це стандарт проміжного програмного забезпечення для розсилки повідомлень, що дозволяє standalone або веб-додаткам на платформі Java створювати, надсилати, отримувати та читати повідомлення в рамках інтеграції інформаційних систем. При цьому відправлення повідомлень виконується в асинхронному режимі, тобто, відправник не чекає на відповідь від отримувача (4.7).

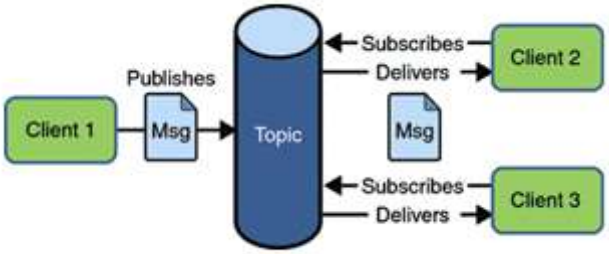


Рисунок 4.7 – Принцип взаємодії між пристроями за протоколом JMS

Отримувачі можуть бути таких типів:

– MQ (Message Queue) – проміжне програмне забезпечення для повідомлення (Message Oriented Middleware), яке дозволяє незалежним і не обов'язково синхронним додаткам у розподіленій системі обмінюватися даними один з одним через канал обміну повідомленнями;

– WEB-контейнер типу JBoss, GlassFish або іншого J2EE-сервера програм з відкритим вихідним кодом, який забезпечує обмін повідомленнями між програмами контейнера або мережею за допомогою JNDI (Java Naming and Directory Interface), API для доступу до служб імен та каталогів.

Крім Rabbit MQ, з яким найчастіше порівнюють Apache Kafka, до JMS-брокерів відносяться Open MQ, Apache ActiveMQ, OpenJMS, JBoss Messaging, Glassfish, TIBCO EMS, Sonic MQ, IBM MQ та інші відкриті та пропрієтарні рішення. JMS підтримує дві моделі обміну повідомленнями:

– із черги або точка-точка, коли кожне повідомлення має лише одного адресата. Відправлене повідомлення потрапляє в чергу, яка відіграє роль поштової скриньки, і може бути прочитано звідти будь-коли. Якщо адресат не працював у момент надсилання повідомлення, повідомлення не пропаде. А після отримання повідомлення повідомлення-отримувач надсилає повідомлення відправнику. Тому для отримання повідомлень отримувачу слід періодично підключатися до потрібної черги та читати у ній повідомлення;

– за передплатою або видавець-підписувач, коли передплатник підписується на певний топік, куди видавець публікує повідомлення, яке отримують усі передплатники цього топіка. У цьому випадку програма-отримувач завжди повинна працювати і бути підписана на топік у момент відправлення повідомлення.

REST – мігрував в IOT як модель проєктування Web API. Архітектура REST умовно складається з клієнтів і серверів. Клієнти ініціюють запити до серверів, сервери обробляють запити та повертають відповідні відповіді. Запити та відповіді будуються навколо передачі представлених ресурсів (рис. 4.8).

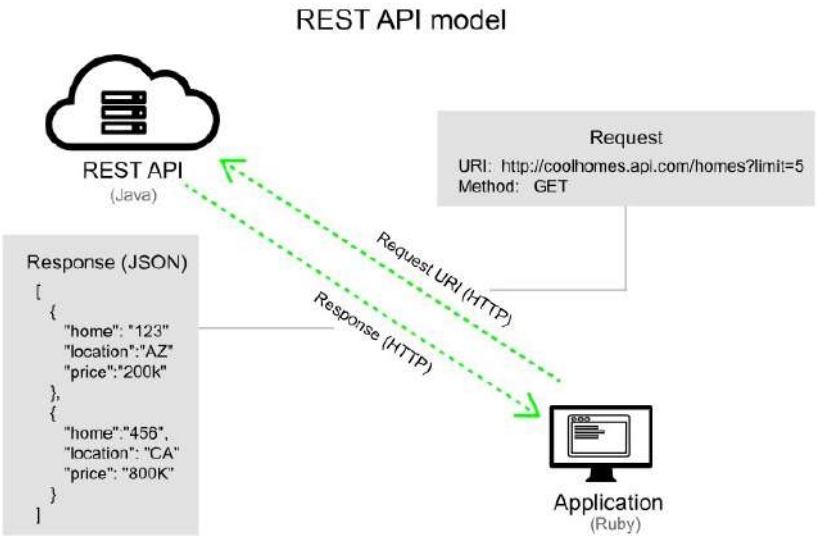


Рисунок 4.8 – Принцип обміну повідомлення за протоколом REST

API, що використовують протокол HTTP для передачі запитів та відповідей, розглядаються як веб-сервіси. У разі веб-сервісів клієнт, який запитує на ресурс, і сервер API, що надає відповідь, можуть використовувати будь-яку мову програмування або платформу. Не має значення, яка мова

програмування, або платформа будуть використовуватися, тому що запит повідомлення та відповідь зроблено через загальний веб-протокол HTTP.

Веб-протокол є частиною веб-сервісів: вони незалежні від мови і тому сумісні між різними платформами та системами. При документуванні REST API не має значення, чи будують інженери API за допомогою Java, Ruby, Python або іншої мови. Запити виконуються через HTTP і відповіді повертаються через HTTP.

Constrained Application Protocol (CoAP) - це протокол, який було розроблено для передачі даних у мережах з обмеженими ресурсами, таких як IoT. CoAP базується на протоколі UDP і забезпечує передачу даних між клієнтом та сервером.

CoAP – звичайний клієнт-серверверний протокол IoT. Він дозволяє клієнтам робити запити на веб-перекази відповідно до потреби. З іншого боку, це також дозволяє допоміжним серверам відповідати на запити, що надходять. Таким чином, вузли пристроїв в екосистемі IoT можуть взаємодіяти лише через CoAP.

CoAP і HTTP дотримуються однакової робочої процедури. Однак CoAP досягає своєї функціональності через асинхронні транзакції (за допомогою UDP). Він використовує виклики POST, GET, PUT і DELETE. Ось чому безпека API має вищий рівень, коли CoAP активний, оскільки це сертифікований протокол RPK і PSK.

CoAP має чотири типи обміну інформацією:

– запит (Request) – це повідомлення від клієнта до сервера, яке містить запит на певну інформацію;

– відповідь (Response) – це повідомлення від сервера до клієнта, яке містить відповідь на запит;

– сповіщення (Notification) – це повідомлення від сервера до клієнта, яке містить оновлення стану або даних, що стосуються клієнта;

– підписка (Subscription) – це повідомлення від клієнта до сервера, що запускає механізм підписки на сповіщення від сервера.

За допомогою цих типів обміну інформацією, CoAP забезпечує зв’язок між обмеженими пристроями та хмарами та дозволяє передавати дані в обмеженому мережевому середовищі з обмеженими ресурсами.

Архітектура CoAP складається з декількох компонентів (рис. 4.9):

– клієнт (Client) – пристрій, який ініціює запити до сервера;

– сервер (Server) – пристрій, який надає відповіді на запити від клієнта;

– Proxy – пристрій, який здійснює пересилання запитів від клієнта до сервера або від сервера до клієнта;

– Resource Directory – директорія ресурсів, яка зберігає інформацію про доступні ресурси на серверах.

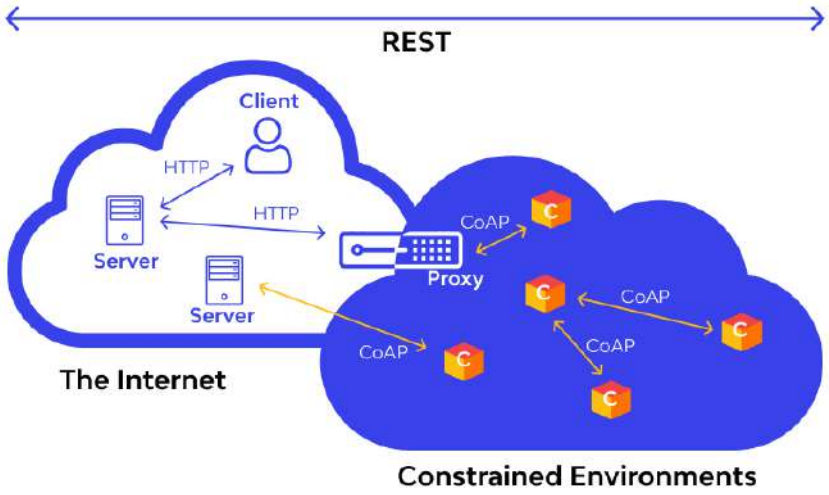


Рисунок 4.9 – Архітектура CoAP

Основним елементом архітектури CoAP є ресурс (Resource). Ресурс може бути створений на сервері і мати унікальну адресу (URI). Клієнт може звертатися до цього ресурсу за допомогою запиту GET або POST і отримувати відповідь від сервера. Ресурс може мати додаткові атрибути, такі як назва, тип та стан, які дозволяють клієнту дізнатися більше про ресурс і його поточний стан.

Оскільки CoAP є протоколом з обмеженими ресурсами, він має декілька обмежень. Наприклад, він не підтримує безпеку на рівні протоколу, але це можна вирішити за допомогою протоколів, таких як DTLS або IPSec. Крім того, він не підтримує повідомлення більшого розміру, ніж MTU мережі.

Протокол XMPP (Extensible Messaging and Presence Protocol) – це відкритий стандарт комунікації в режимі реального часу для обміну повідомленнями, створення мережевих служб та роботи з мережевими пристроями. Він базується на протоколі XML і зазвичай використовується для створення мережевих додатків для обміну повідомленнями в режимі реального часу.

Основні характеристики протоколу XMPP:

– відкритий стандарт, що дає змогу кожному створювати власні мережеві додатки та розширювати функціонал протоколу;

– підтримує різні методи автентифікації, такі як ім’я користувача та пароль, SSL-сертифікати, SASL (Simple Authentication and Security Layer) та інші;

– дає змогу користувачам керувати своєю присутністю в мережі та обмінюватися цією інформацією з іншими користувачами;

– дозволяє користувачам обмінюватися повідомленнями в режимі реального часу;

– дозволяє розширювати функціонал протоколу за допомогою стандартів, які приймаються спільнотою розробників;

– може бути використаний в мережах будь-якого розміру, від невеликих локальних мереж до глобальних мереж з мільйонами користувачів;

– підтримує шифрування за допомогою TLS (Transport Layer Security) та SSL (Secure Sockets Layer), що забезпечує безпеку обміну повідомленнями;

– підтримує різні методи автентифікації, такі як ім’я користувача та пароль, механізм SASL (Simple Authentication and Security Layer), та інші;

– підтримує маршрутизацію повідомлень, що дозволяє користувачам встановлювати зв’язок з іншими користувачами через різні мережі та протоколи;

– дозволяє користувачам об’єднуватися в групові чати з можливістю передачі повідомлень в реальному часі.

Архітектура XMPP ґрунтується на розподіленій архітектурі клієнт-сервер (Client-Server Architecture), що дозволяє розробляти масштабовані системи з безліччю користувачів та серверів (рис. 4.10).

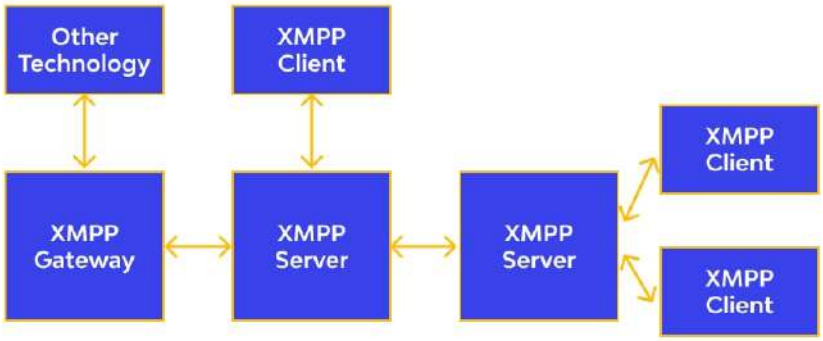


Рисунок 4.10 – Архітектура XMPP

XMPP складається з двох основних компонентів: клієнти та сервери. Клієнти – це програми для обміну повідомленнями у реальному часі, а сервери –

це програми, що забезпечують мережеву інфраструктуру для обробки повідомлень та керування присутністю.

XMPP використовує модель взаємодії «запит-відповідь» (Request-Response Interaction Model) для обміну повідомленнями та станом присутності. Клієнти та сервери взаємодіють за допомогою XML-повідомлень, що передаються через мережу. XMPP також містить протоколи для обміну файлами, автентифікації, шифрування та інших операцій.

Однією з особливостей XMPP є можливість використання різних серверів у мережі, що дозволяє користувачам взаємодіяти зі своїми контактами з різних доменів та мереж. XMPP також підтримує групові розмови та відеозв'язок, а також можливість використання різних клієнтів на різних пристроях (наприклад, комп'ютерах, смартфонах тощо).

4.3 Особливості та принцип використання протоколу MQTT

4.3.1 Основні характеристики протоколу MQTT

Комунікаційна стратегія публікації/передплати (pub/sub) MQTT, спрямована на максимізацію використання пропускної здатності, є заміною традиційної споживчої архітектури, яка безпосередньо взаємодіє з кінцевою точкою. Однак у парадигмі pub/sub клієнт, який передає новини (видавець), відокремлений від клієнтів, які отримують інформацію (або передплатників). Оскільки ані автори, ані клієнти не спілкуються один з одним одразу, їхньою взаємодією в них керують треті сторони, які називаються брокерами (рис. 4.11).

MQTT надає спосіб створення ієрархії каналів зв'язку – свого роду гілка з листям. Щоразу, коли видавець має нові дані для поширення серед клієнтів, повідомлення супроводжується приміткою контролю доставки. Клієнти вищого рівня можуть отримувати кожне повідомлення, тоді як клієнти нижчого рівня можуть отримувати повідомлення, які стосуються лише одному чи двом базовим каналам, «відгалуженим» у нижній частині ієрархії. Це полегшує обмін інформацією розміром від двох байт до 256 Мб.

Основні риси протоколу MQTT:

– обмін повідомленнями відбувається за принципом видавець-передплатник (Pub-Sub);

– розмір заголовка повідомлення становить 2 байти, а корисне навантаження може змінюватись від 1 байта до 256 Мб;

– у протоколі закладено можливість вибору одного із трьох рівнів обслуговування.

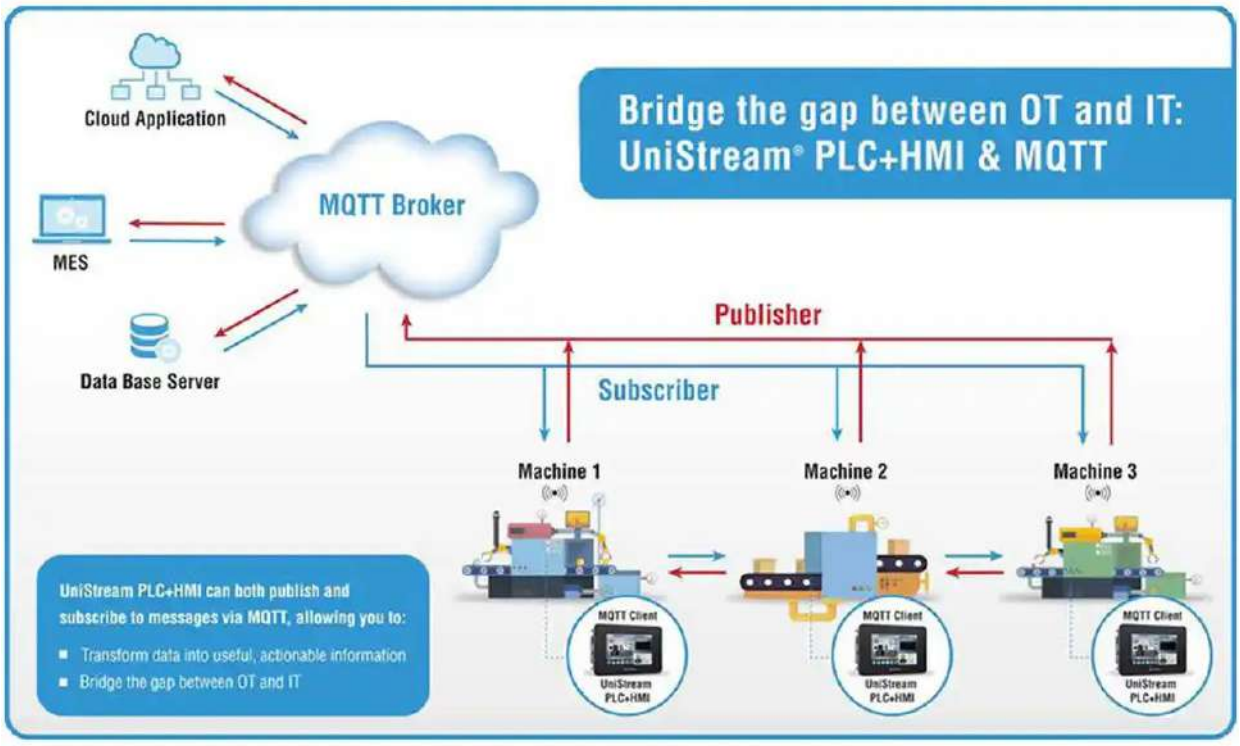


Рисунок 4.11 – MQTT, архітектура «видавець-передплатник»

Відмінною особливістю принципу «видавець-передплатник» від клієнт-серверного підходу є те, що клієнти, які надсилають повідомлення (видавці, Publisher), та клієнти, які приймають повідомлення (підписники, Subscriber), як правило, розділені. Розподіл може бути організований у трьох площинах:

– простір – видавець і передплатник нічого не повинні знати один про одного;

– час – видавець і передплатник не повинні бути увімкнені в один і той самий час;

– синхронізація – операції на обох сторонах не повинні припинятися протягом публікації чи отримання інформації.

Видавець та передплатник не передають одне одному повідомлення безпосередньо, не встановлюють прямий контакт, можуть не знати про існування одне одного. Координує та керує передачею повідомлень від видавця до передплатника та від передплатника до видавця брокер (Broker).

Паралельне виконання операцій на брокері є другою важливою особливістю принципу взаємодії «видавець-передплатник».

MQTT-клієнт – це, зазвичай, пристрій, оснащений мікроконтролером, який підтримує стек TCP/IP. Клієнтські бібліотеки MQTT доступні для багатьох мов програмування, наприклад Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, NET.

Брокер є основним елементом системи «видавець-передплатник». Він відповідає за прийом усіх повідомлень, їх фільтрацію, ухвалення рішення про те, кому цікаві ці повідомлення, і, зрештою, за пересилання повідомлень усім клієнтам-передплатникам.

Серед серверних реалізацій брокера можна назвати:

– IBM WebSphere MQ;

– відкрите програмне забезпечення Mosquitto;

– рішення, що базується на хмарному сервісі Eurotech Everywhere Device Cloud;

– легко масштабований та високопродуктивний відкритий сервер emqttd, остання версія (0,17) дозволяє обслуговувати 1,3 мільйони з'єднань;

– брокер HiveMQ, що забезпечує корпоративну безпеку та максимальну масштабованість.

Спрощений процес обміну інформацією можна описати так (рис. 4.12):

– видавець передає повідомлення з даними (наприклад, інформація з датчиків температури) на брокер, вказуючи при цьому тему (Topic), до якої ці дані належать (наприклад, «Temp»);

– брокер аналізує, які з передплатників мають передплату на певні теми, в даному випадку – на тему «Temp»;

– передплатникам, які підписані на тему «Temp», брокером буде надіслано повідомлення з інформацією від датчиків температури.

Таким чином, безліч передплатників може бути підписано на різноманітні теми та залежно від цих передплат отримувати необхідну їм інформацію, не спілкуючись безпосередньо з видавцем.

4.3.2 Формат повідомлень за протоколом MQTT

Центральним елементом комунікації є брокер (Broker), що виконує роль сервера, який відповідає за обробку та надсилання всіх повідомлень.

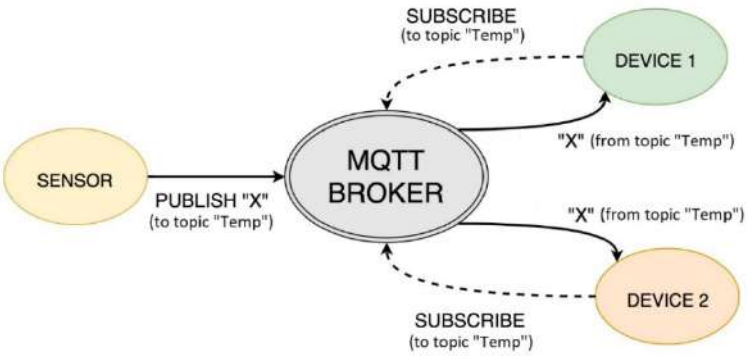


Рисунок 4.12 – MQTT, архітектура «видавець-передплатник»

Кожен клієнт, який хоче надіслати повідомлення через сервер, в термінології протоколу MQTT, є видавцем (Publisher).

Брокер фільтрує вхідні повідомлення та перенаправляє клієнтам, які приймають повідомлення. Клієнти, зареєстровані в брокері та підписані на відповідні топіки, називаються передплатники (Subscriber).

Безпосередньо видавець та передплатник повідомлення не передають і між собою ніяк не пов’язані. Для взаємодії з брокером пристрою використовують стандартизований набір повідомлень:

– Connect – встановити з’єднання з брокером;

– Disconnect – розірвати з’єднання з брокером;

– Publish – опублікувати дані в топік на брокері;

– Subscribe – підписатися на топік на брокері;

– Unsubscribe – відписатися від топіка.

Сервер MQTT відповідає за автентифікацію та авторизацію клієнтів. Після успішної авторизації та автентифікації клієнти зможуть стати видавцями та передплатниками.

Спрощено зв’язок між усіма учасниками можна представити так. Видавець відправляє дані MQTT брокеру, вказуючи при цьому в повідомленні певну тему – топік, до якої ці дані відносяться. Далі брокер аналізує отримане повідомлення та пересилає його тим клієнтам, які підписані на цю тему. Таким чином, на стороні брокера відбувається фільтрація, що дозволяє передплатникам отримувати лише інформацію, у якій вони зацікавлені. При цьому клієнт може бути підписаний одразу на кілька топіків.

Будь-який IoT-пристрій може виконувати функції як відправника, так і одержувача, тобто кожен передплатник може також виступати у ролі видавця.

Протокол MQTT дозволяє пристроям спілкуватися та взаємодіяти з різними типами серверів. При плануванні та розробці системи необхідно заздалегідь уточнити, які можливості пропонують доступні на ринку сервери та вибрати той, який найкраще задовольняє вашим очікуванням.

У протоколі MQTT існує 15 типів повідомлень, які можуть складатися з кількох частин (рис. 4.13):

– фіксований заголовок;

– змінний заголовок;

– корисне навантаження (дані, що передаються).

Рисунок 4.13 – Загальна структура повідомлення MQTT

Фіксований заголовок є у всіх типах повідомлень. В цьому заголовку містяться такі поля (рис 4.14):

– Message Type (тип повідомлення) – наприклад: CONNECT, SUBSCRIBE, PUBLISH тощо;

– прапори, специфічні для кожного пакета MQTT – ці 4 біти використовуються для допоміжних прапорів, наявність і статус яких залежить від типу повідомлення;

– Remaining Length (довжина, що залишається) – поточна довжина повідомлення (заголовок змінної + дані), розмір від 1 до 4 байт.

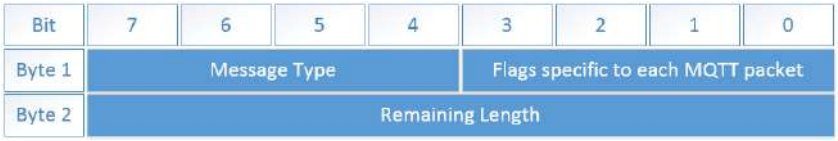


Рисунок 4.14 – Поля фіксованого заголовка

Всього в протоколі MQTT існує 15 типів повідомлень (табл. 4.1).

Таблиця 4.1 – Типи повідомлень протоколу MQTT

Тип повідомлення	Значення	Напрям передачі	Опис
Зарезервовано	0000 (0)	Заборонено	Зарезервовано
CONNECT	0001 (1)	Клієнт → Сервер	Запит клієнта на підключення до сервера
CONNACK	0010 (2)	Клієнт ← Сервер	Підтвердження підключення
PUBLISH	0011 (3)	Клієнт ← Сервер Клієнт → Сервер	Опублікувати повідомлення
PUBACK	0100 (4)	Клієнт ← Сервер Клієнт → Сервер	Опублікувати підтвердження
PUBREC	0101 (5)	Клієнт ← Сервер Клієнт → Сервер	Публікацію отримано
PUBREL	0110 (6)	Клієнт ← Сервер Клієнт → Сервер	Публікацію випущено
PUBCOMP	0111 (7)	Клієнт ← Сервер Клієнт → Сервер	Публікацію завершено
SUBSCRIBE	1000 (8)	Клієнт → Сервер	Запит клієнта на підписку
SUBACK	1001 (9)	Клієнт ← Сервер	Підписка підтверджена
UNSUBSCRIBE	1010 (10)	Клієнт → Сервер	Запит на скасування підписки
UNSUBACK	1011 (11)	Клієнт ← Сервер	Підтвердження скасування підписки
PINGREQ	1100 (12)	Клієнт → Сервер	Запит PING
PINGRESP	1101 (13)	Клієнт ← Сервер	Відповідь PING
DISCONNECT	1110 (14)	Клієнт → Сервер	Відключення клієнту
Зарезервовано	1111 (15)	Заборонено	Зарезервовано

Чотири старші біти першого байту використовуються як специфічні прапори (рис. 4.15):

– DUP – встановлюється, коли клієнт або брокер MQTT затвердить повторне відправлення пакета (використовується в PUBLISH, SUBSCRIBE, UNSUBSCRIBE, PUBREL). Якщо прапорець встановлено, заголовок змінної має містити ідентифікатор повідомлення;

– QoS – якість обслуговування (0, 1, 2);

– RETAIN – в процесі публікації даних із встановленим прапором retain, брокер збереже його. З наступною передплатою на цей топік брокер негайно

відправить повідомлення з цим прапором. Використовується лише у повідомленнях із типом PUBLISH.

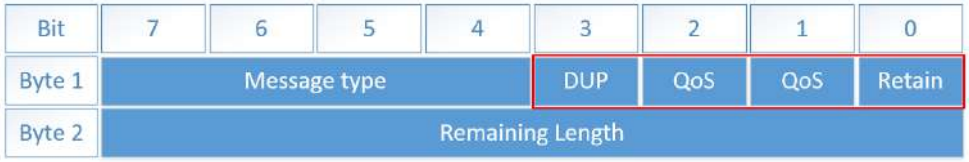


Рисунок 4.15 – Специфічні прапори повідомлення

Змінний заголовок є у деяких типах заголовків і містить такі дані:

– Packet identifier (ідентифікатор пакета) – є у більшості типах повідомлень;

– Protocol name (назва протоколу) – відбувається лише у типі повідомлення CONNECT;

– Protocol version (версія протоколу) – відбувається лише у типі повідомлення CONNECT;

– Connect flags (прапори підключення) – прапори, які визначають поведінку клієнта під час підключення.

На рис. 4.16 показана структура змінного заголовка.

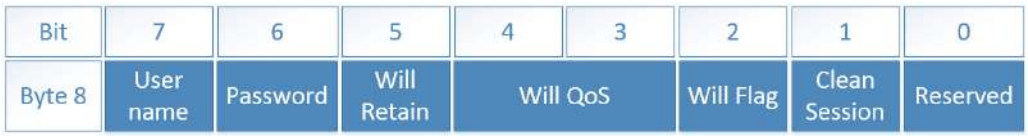


Рисунок 4.16 – Структура змінного заголовка

Байт, що становить заголовок має такі поля:

– User name – якщо прапор встановлений, тоді ім’я користувача має бути в «корисному навантаженні»;

– Password – якщо прапор встановлений, то пароль має бути вказаний в «корисному навантаженні»;

– Will Retain – якщо прапор встановлений, брокер зберігає повідомлення;

– Will QoS – визначає якість послуги повідомлення;

– Will Flag – якщо встановлений прапор, клієнт відключиться від сервера без надсилання команди DISCONNECT; брокер сповіщає всіх підключених клієнтів про цю подію за допомогою Will Message;

– Clean Session – якщо прапор не встановлений, брокер зберігає сеанс, а також усі підписки клієнта, а при наступному підключенні із замовником відправляє всі повідомлення з QoS1 та QoS2, отримані через брокера, коли клієнт був вимкнений. Якщо прапорець встановлений, під час спроби встановлення наступного з'єднання клієнт повинен підписатися на всі теми.

Розмір даних або «корисного навантаження», можна обчислити, віднявши довжину змінного заголовка від решти довжини. Зміст і формат даних, які надсилаються за допомогою повідомлень MQTT, генеруються у програмі, що використовує цей протокол.

QoS є технологією надання різним класам трафіку різних пріоритетів в обслуговуванні. Специфікація протоколу MQTT визначає три рівні якості обслуговування, що визначається спеціальним прапорцем QoS. Як зазначалося вище, основною відмінністю протоколу MQTT є можливість використання різних рівнів обслуговування, які задаються значенням даного прапора. Це робить MQTT більш гнучким, на відміну від протоколу CoAP (Constrained Application Protocol), повідомлення якого можуть підтверджуватись або оброблятися без підтвердження.

QoS 0: (At most once) – не більше одного. Видавець публікує повідомлення (PUBLISH) на брокері, а брокер публікує його для передплатника. Проте видавець не вимагає, щоб це повідомлення було гарантовано передано передплатнику. Іншими словами, передплатник може і не одержати це повідомлення, але це не відслідковується видавцем (рис. 4.17).

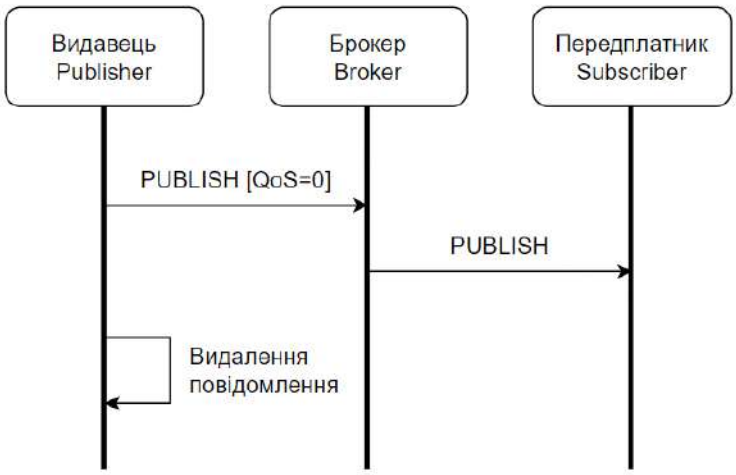


Рисунок 4.17 – Сценарій QoS 0: (At most once) – не більше одного

Описаний сценарій використовується для випадків, коли втрати даних не критичні. Наприклад, під час постійного моніторингу температури, коли втрата одиничного вимірювання не відіграє істотну роль загальної картини.

QoS 1: At least once – хоча б один. Видавець публікує повідомлення на брокері (PUBLISH). Брокер зберігає це повідомлення та публікує його для передплатника. Тільки після того, як повідомлення буде опубліковано для передплатника, брокер надсилає підтвердження публікації видавцю (PUBACK). Сценарій такої взаємодії подано на рис. 4.18.

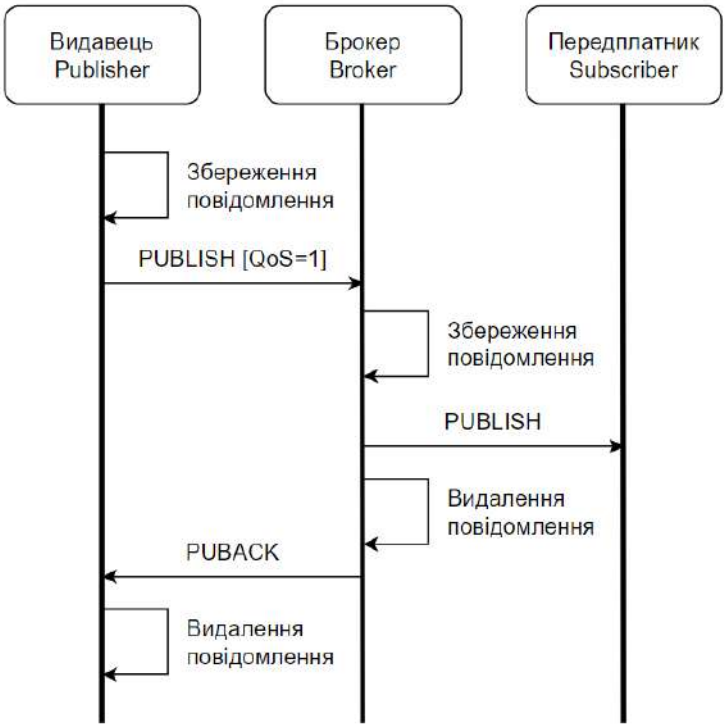


Рисунок 4.18 – Сценарій QoS 1: At least once – хоча б один

Доки видавець не отримає підтвердження публікації передплатнику, дана публікація надсилатиметься брокеру і далі передплатнику. Таким чином, передплатник повинен отримати це повідомлення як мінімум один раз.

QoS 2: Exactly one – гарантовано один. Даний рівень QoS забезпечує найвищу гарантію доставки повідомлень за рахунок використання додаткових процедур підтвердження та завершення публікації (PUBREC, PUBREL, PUBCOMP). Сценарій представлено на рис. 4.19 і застосовується для ситуацій, коли потрібно виключити будь-які втрати та дублювання даних від датчиків. Наприклад, коли від отриманого повідомлення спрацьовує сигналізація, виклик екстрених служб.

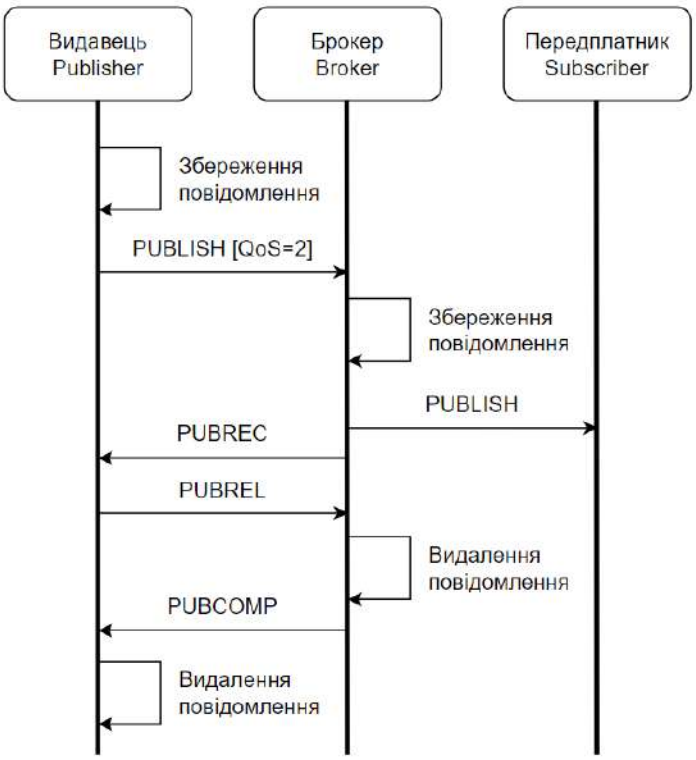


Рисунок 4.19 – Сценарій QoS 2: Exactly one – гарантовано один

Спеціальний прапор RETAIN служить для індикації збереження останнього прийнятого брокером повідомлення (рис. 4.20). Тобто прапор RETAIN = 1 у повідомленні PUBLISH від видавця повідомляє брокеру про те, що повідомлення на цю тему потрібно зберегти і, коли новий передплатник приєднається до теми, надіслати йому це повідомлення.

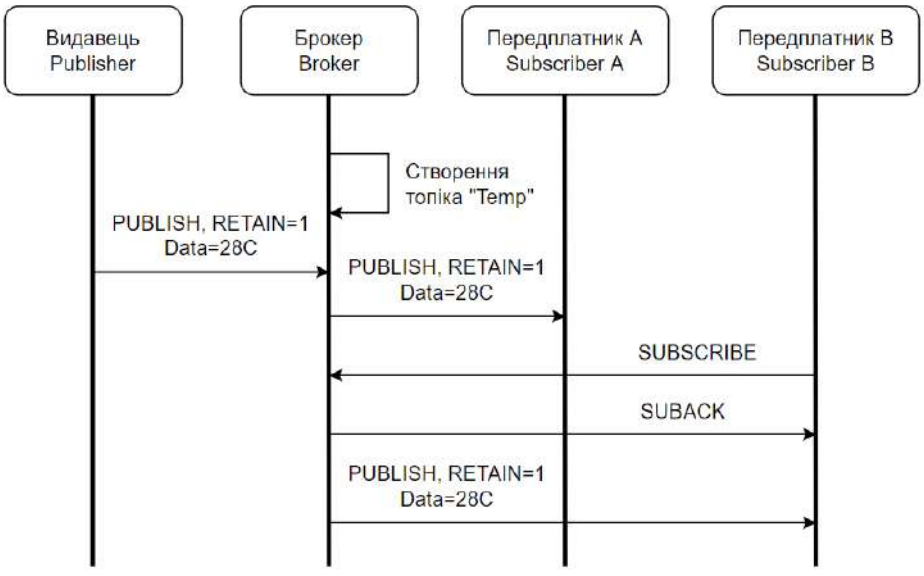


Рисунок 4.20 – Сценарій для прапорця RETAIN

Процес встановлення з’єднання показано на рис. 4.21.

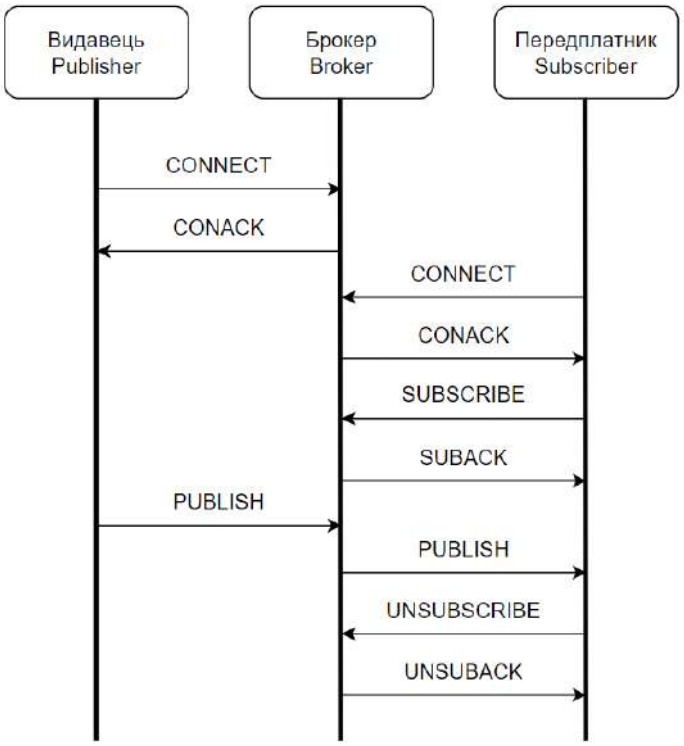


Рисунок 4.21 – Процес встановлення з’єднання

Встановлення з’єднання починається з передачі від клієнта брокеру повідомлення CONNECT, в якому зазначаються:

– ClientId – унікальний ідентифікатор кожного клієнта, що підключається до брокера;

– CleanSession – прапор видалення збережених повідомлень із попередніх сесій для даного клієнта;

– Username/Password – ім’я користувача та пароль для ідентифікації та авторизації клієнта;

– KeepAlive – тимчасовий інтервал, що регулює передачу ping-запитів та ping-відповідей для контролю відключення однієї зі сторін.

Брокер у відповідь надсилає клієнту повідомлення CONACK, що складається з:

– Session Present Flag – вказує чи існують для даного клієнта сесії, що діють, від попередніх підключень;

– Connect Acknowledge Flag – повідомляє клієнта про успішне підключення або про будь-які помилки.

Після того, як клієнта MQTT було підключено до брокера, він може публікувати повідомлення. Публікація відбувається шляхом відправки брокеру від клієнта повідомлення PUBLISH, де зазначаються:

– Topic Name – назва теми, до якої належить це повідомлення. Це поле є обов'язковим, тому що MQTT-брокер приймає рішення про пересилання того чи іншого повідомлення клієнту, виходячи з тем, на які клієнт підписаний;

– спеціальні прапори – QoS, DUP та RETAIN;

– корисне навантаження, де передаються самі дані.

Таким чином, після отримання повідомлення PUBLISH брокер відправляє підтвердження прийому публікації (якщо це задано QoS) і надсилає отримане повідомлення всім клієнтам, які підписані на цю тему.

Щоб отримувати повідомлення з необхідними даними, клієнт MQTT повинен спочатку підписатися на їх отримання за допомогою повідомлення SUBSCRIBE. Це повідомлення складається з двох частин:

– Packet Identifier – необхідно для QoS 1 та QoS 2;

– List of Subscriptions – назви тем, які клієнт хоче передплатити, та необхідне значення QoS.

Слід зазначити, що у протоколі MQTT прийнята ієрархічна структура побудови тем, тому для зручності застосовуються wildcard-символи, завдяки яким передплатник може передплатити всі підтеми цієї теми (символ #) чи теми певного рівня (символ +).

У відповідь на повідомлення SUBSCRIBE брокер відправляє клієнту підтвердження SUBACK, в якому повідомляє результат підписки (успішна чи ні). Також клієнт може відписатися від теми, яка більше не представляє для нього інтересу, відправивши брокеру повідомлення UNSUBSCRIBE, у якому буде зазначена дана тема. Брокер підтверджує відмову від інформації на цю тему повідомленням UNSUBACK.

4.4 Розгортання брокера MQTT Mosquitto

4.4.1 Основні відомості про Mosquitto

Mosquitto є популярним брокером повідомлень MQTT, який забезпечує зв'язок між сенсорами та іншими IoT-пристроями. Він був розроблений за допомогою мови програмування C і може працювати як на Windows, так і на Linux.

Mosquitto простий у налаштуванні та використанні. Він має низькі вимоги до ресурсів, що дозволяє йому працювати на вбудованих пристроях та малих комп'ютерах. Mosquitto є повноцінним брокером MQTT. Він підтримує версії 3.1 та 3.1.1 протоколу MQTT.

За технологією pub/sub Mosquitto підтримує можливість підписки, які дозволяють отримувати повідомлення тільки на певні теми.

Mosquitto підтримує три рівні якості обслуговування (QoS):

– QoS 0 (кожен раз надсилається тільки один раз);

– QoS 1 (надсилається щонайменше один раз);

– QoS 2 (надсилається рівно два рази).

Даний брокер підтримує автентифікацію та шифрування. Він може використовувати базу даних автентифікації для перевірки користувачів та паролів. Крім того, він підтримує SSL / TLS для шифрування з'єднань. Також, він підтримує мости між різними брокерами MQTT. Це дозволяє повідомленням з одного брокера MQTT бути переданими на інший брокер MQTT.

4.4.2 Порядок встановлення брокера Mosquitto на ОС Linux

Розглянемо порядок дій для встановлення брокера Mosquitto на ОС Linux. Першим кроком, необхідно виконати команду:

```
sudo apt update
```

Дана команда виконує оновлення списку доступних пакетів для встановлення з віддалених репозиторіїв. Це дозволяє системі знайти та завантажити оновлення для встановлених пакетів та інші корисні програми. Оновлення списку доступних пакетів дозволяє зберегти систему оновленою та забезпечити стабільну роботу пакетного менеджера при встановленні нових програм. Перед встановленням будь-якої програми на ОС Linux, зазвичай рекомендується виконати команду sudo apt update для забезпечення актуальності списку пакетів.

Наступним кроком, встановлюємо Mosquitto за допомогою команди (рис. 4.22):

```
sudo apt install mosquitto
```

Рисунок 4.22 – Встановлення брокеру Mosquitto

В результаті виконання команди на ОС буде встановлено необхідне програмне забезпечення для роботи з брокером Mosquitto.

В версії ОС, наприклад Ubuntu 22.04 LTS крім базової команди встановлення брокера, необхідно додатково виконати перевірку та встановлення додаткових бібліотек. Таким чином, якщо попередньо не було встановлено менеджер пакетів Python pip, його необхідно додати до системи за допомогою команди:

```
sudo apt install python3-pip
```

Далі встановлюємо бібліотеку Paho MQTT для Python за допомогою pip (рис. 4.23):

```
pip3 install paho-mqtt
```

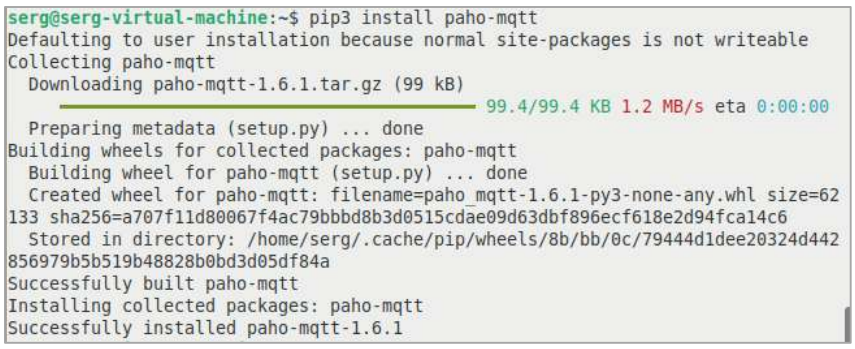


Рисунок 4.23 – Встановлення бібліотеки Paho MQTT для Python

Бібліотека Paho MQTT для Python необхідна для підтримки MQTT-протоколу в програмах на мові програмування Python. Вона містить функції для підключення до брокера MQTT, публікації повідомлень та підписки

на топіки. Встановлення цієї бібліотеки дозволить створювати програми, які взаємодіють з іншими пристроями за допомогою MQTT-протоколу.

Щоб перевірити, чи встановлено Mosquitto правильно, необхідно запустити службу Mosquitto та перевірити її стан за допомогою таких команд:

```
sudo systemctl start mosquitto
sudo systemctl status mosquitto
```

На рис. 4.24 подано результат виконання цих команд.

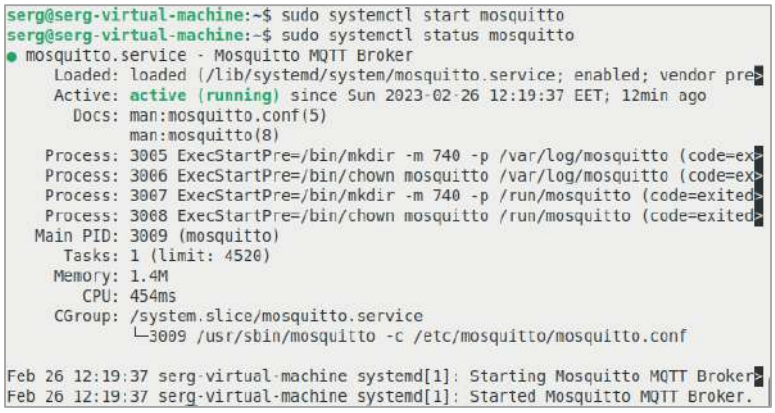


Рисунок 4.24 – Результат запуску Mosquitto та перевірки його стану

Якщо служба Mosquitto запускається успішно, перевіримо, чи можна підключитись до неї з використанням інструментів, таких як mosquitto_sub та mosquitto_pub. Якщо ці команди планується запускати на тій самій операційній системі, що й сам сервер, то необхідно додати до неї програму-клієнт. Також це необхідно зробити й на іншому ПК, де планується виконувати задачі підключення до брокера за допомогою вищевказаних команд.

Mosquitto-clients – це пакет програм для командного рядка, які дозволяють взаємодіяти з брокером Mosquitto через протокол MQTT. Встановлення mosquitto-clients дозволяє виконувати наступні операції:

– перевірка підключення до брокера Mosquitto;

– підписка на теми для отримання повідомлень;

– публікація повідомлень на теми;

– тестування і налагодження роботи з брокером Mosquitto.

Встановлення mosquitto-clients є корисним для тих, хто працює з MQTT-брокером Mosquitto та бажає взаємодіяти з ним через командний рядок.

Виконаємо встановлення клієнта за допомогою наступної команди:

```
sudo apt install mosquitto-clients
```

Перевірка роботи брокера виконується в декілька кроків (рис. 4.25):

– запускаємо два окремих термінали;

– в першому терміналі вводимо та виконуємо наступну команду (рис. 4.25, 1)

```
mosquitto_sub -h localhost -t test
```

для передплати на тему «test», після чого термінал переходить в режим очікування;

– в другому терміналі виконаємо публікацію даних в темі «test» (рис. 4.25, 2):

```
mosquitto_pub -h localhost -t test -m "hello world"
```

– після публікації даних в першому терміналі повинні з’явитись такі самі дані, які були відправлені (рис. 4.25, 3).

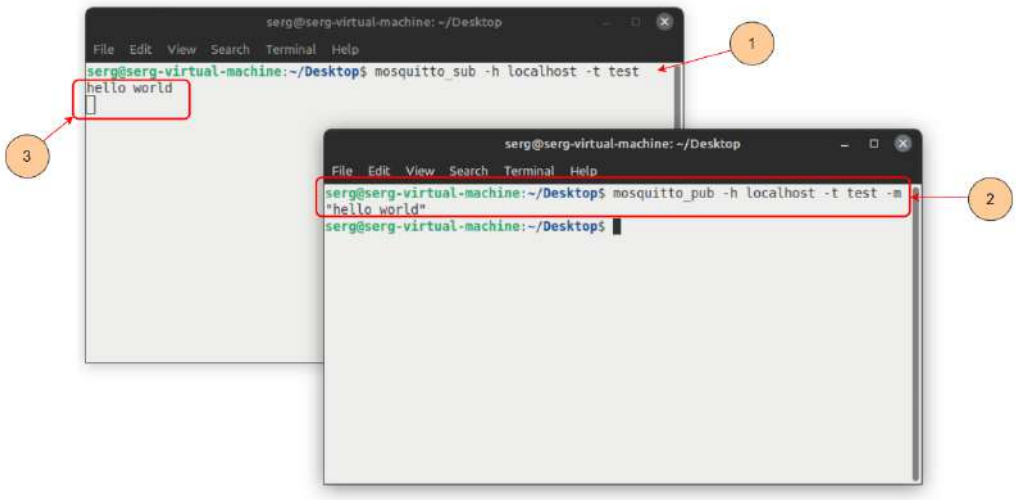


Рисунок 4.25 – Перевірка роботи брокера Mosquitto

В більшості випадках практичного застосування брокера Mosquitto використовується автентифікація піл час виконання передплати на теми та публікації самих тем.

Для увімкнення підтримки автентифікації в Mosquitto потрібно налаштувати файл конфігурації mosquitto.conf. Відкриємо цей файл командою:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

Виконаємо додавання наступних рядків в кінець файлу:

```
# Enable authentication
allow_anonymous false
password_file /etc/mosquitto/passwd
```

На рис. 4.26 подано вміст файлу конфігурації після додавання необхідних налаштувань. Перший рядок «allow_anonymous false» вимикає можливість анонімного підключення до брокера. Другий рядок «password_file …» вказує на шлях до файлу з паролями користувачів. Після додавання нових рядків необхідно зберегти зміни та закрити файл конфігурації.

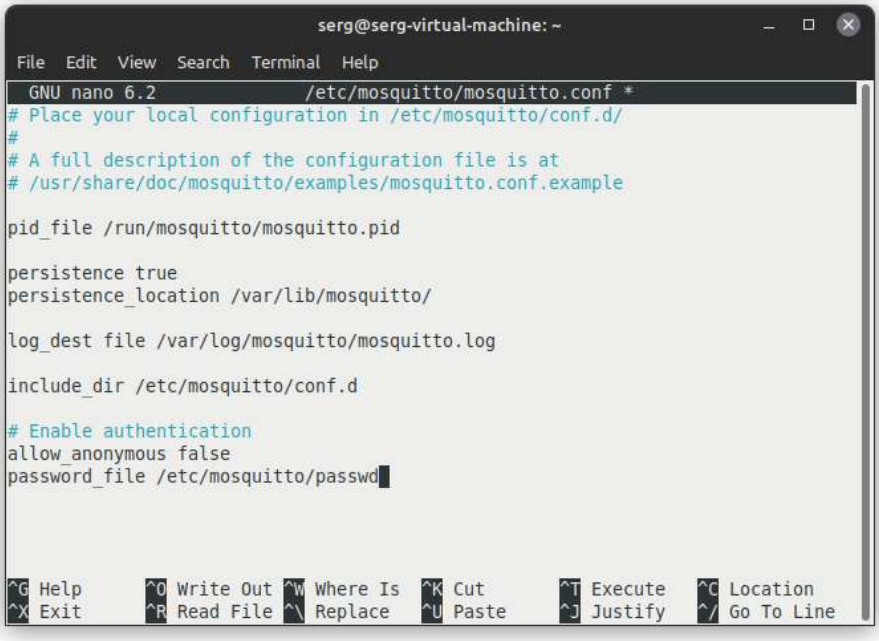


Рисунок 4.26 – Вміст файлу конфігурації

Далі необхідно створити файл з паролями користувачів командою:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd <username>
```

Замість <username> необхідно вказати ім’я користувача, якому потрібно створити пароль. Після цього система попросить ввести пароль для користувача (рис. 4.27). Цей крок необхідно повторити для кожного користувача, якому потрібно створити пароль.

Після створення паролів необхідно перезавантажити службу Mosquitto, щоб зміни набрали чинності:

```
sudo systemctl restart mosquitto
```

```
serg@serg-virtual-machine:~$ sudo mosquitto_passwd -c /etc/mosquitto/passwd testMQTT
Password:
Reenter password:
```

Рисунок 4.27 – Створення пароля для користувача testMQTT

Тепер при спробі підключитись до Mosquitto буде необхідно ввести ім’я користувача та пароль. Для підключення до Mosquitto з ім’ям користувача та паролем, необхідно використовувати параметри -u та -P під час використання командного рядка mosquitto_sub або mosquitto_pub.

Наприклад, якщо ми хочемо підписатись на повідомлення топіка «example» з ім’ям користувача «testMQTT» та відповідним паролем «testMQTT», ми можемо виконати наступну команду:

```
mosquitto_sub -h <hostname> -t example -u testMQTT -P testMQTT
```

Аналогічно, якщо ми хочемо надіслати повідомлення до топіка «example» з ім’ям користувача «testMQTT» та паролем «testMQTT», ми можемо виконати наступну команду:

```
mosquitto_pub -h <hostname> -t example -m "Hello, World!" -u testMQTT -P testMQTT
```

Необхідно звернути увагу, що необхідно замінити <hostname> на ім’я або IP-адресу хоста, на якому працює Mosquitto. Приклад використання автентифікації подано на рис. 4.28.

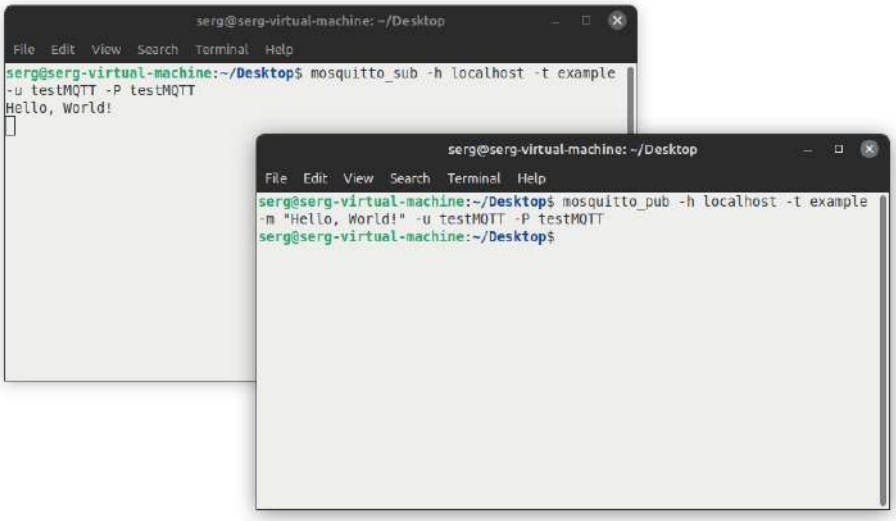


Рисунок 4.28 – Приклад використання автентифікації під час роботи з брокером Mosquitto

4.5 Використання графічної платформи Cedalo для управління брокером Mosquitto

Cedalo є платформою для розробки та розгортання рішень Інтернету речей, яка надає простий та інтуїтивно зрозумілий інтерфейс, що дозволяє швидко створювати та налаштовувати IIoT-проєкти без спеціалізованих знань.

Cedalo надає широкий спектр інструментів для розробки та інтеграції рішень Інтернету речей, таких як Node-RED, Apache Kafka та MQTT. Ця платформ має вбудовану підтримку захисту від кібератак, а також забезпечує захист персональних даних користувачів.

Cedalo може бути використана для створення як малих, так і великих IIoT-проєктів, які можуть масштабуватися залежно від потреб користувачів. Даний інструмент може бути використаний на різних платформах, включаючи Linux, Windows та MacOS.

Також, Cedalo має активну спільноту користувачів та розробників, які надають допомогу та розробляють нові функції та плагіни для платформи.

Наряду з платною, існує безкоштовна версія платформи управління з відкритим кодом. Її можна використовувати для керування клієнтами, групами та ролями, що доступно у новому плагіні Dynamic Security. Щоб встановити та налаштувати цю версію платформи, потрібна платформа Docker на робочій версії Linux для доступу до графічного інтерфейсу керування.

Встановлення Docker розпочинається з оновлення списку доступних пакетів для встановлення з віддалених репозиторіїв:

```
sudo apt update.
```

Наступною командою є:

```
sudo apt install ca-certificates curl apt-transport-https
```

Ця команда встановлює необхідні залежності для того, щоб здійснити успішний доступ до репозиторію Cedalo, зокрема:

– ca-certificates – пакет з сертифікатами довірених організацій, що використовуються для перевірки підписів цифрових сертифікатів під час з'єднання з захищеними ресурсами через HTTPS;

– curl – утиліта для отримання вмісту з інтернету за допомогою протоколу HTTP або HTTPS;

– apt-transport-https – модуль для підтримки передачі даних через HTTPS у менеджері пакетів apt.

Щоб завантажити пакети Docker на Ubuntu, нам потрібно додати ключ GPG, який використовується розробником для підпису пакетів Docker, інакше система поверне помилку та не зможе використовувати репозиторій:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Далі, ми можемо встановити докер за допомогою стандартного системного репозиторію Ubuntu Jammy, однак доступна версія не буде останньою. Отже, необхідно додати офіційне сховище Docker вручну за допомогою поданого нижче блоку команд:

```
echo "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable" \
| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Нарешті ми налаштували всі необхідні речі. Тепер необхідно запустити команду системного оновлення, щоб оновити кеш сховища та оновити вже встановлені пакети (рис. 4.29):

```
sudo apt update
```

Після цього скористаємось пакетами APT, щоб отримати всі інструменти Docker, необхідні для створення контейнерів:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose
```

```
serg@serg-virtual-machine:~/Cedalo$ sudo apt update
Hit:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:3 https://deb.nodesource.com/node_19.x jammy InRelease
Get:4 https://download.docker.com/linux/ubuntu focal InRelease [57,7 kB]
Hit:5 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Ign:6 http://packages.linuxmint.com vera InRelease
Hit:7 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:8 http://packages.linuxmint.com vera Release
Get:10 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [24,7 kB]
Fetched 82,4 kB in 1s (57,4 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
21 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: https://download.docker.com/linux/ubuntu/dists/focal/InRelease: Key is stored in lega
cy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8)
 for details.
```

Рисунок 4.29 – Оновлення кешу репозіторію

На рис. 4.30 подано результат виконання даної команди.

```
serg@serg-virtual-machine:~/Cedalo$ sudo apt-get install docker-ce docker-ce-cli contain
erd.io docker-compose
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-buildx-plugin docker-ce-rootless-extras docker-compose-plugin
  docker-scan-plugin git git-man liberror-perl libslirp0 pigz python3-attr
  python3-docker python3-dockerpty python3-docopt python3-dotenv python3-jsonschema
  python3-pyrsistent python3-texttable python3-websocket slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite git-daemon-run | git-daemon-sysvinit git-doc
  git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn python-attr-doc
  python-jsonschema-doc
Recommended packages:
  docker.io
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras
  docker-compose docker-compose-plugin docker-scan-plugin git git-man liberror-perl
  libslirp0 pigz python3-attr python3-docker python3-dockerpty python3-docopt
  python3-dotenv python3-jsonschema python3-pyrsistent python3-texttable
  python3-websocket slirp4netns
```

Рисунок 4.30 – Встановлення пакетів, що необхідні для роботи Docker на Ubuntu

Ця команда встановлює пакети, необхідні для роботи Docker на Ubuntu:

– docker-ce: це пакет Docker Community Edition, який містить основні компоненти Docker Engine, необхідні для запуску контейнерів;

– docker-ce-cli: це пакет Docker CLI (Command Line Interface), який містить команди для управління Docker з командного рядка;

– containerd.io: це пакет Containerd, який забезпечує рівень абстракції над Docker Engine і дозволяє управляти процесами контейнерів;

– docker-compose: це пакет Docker Compose, який дозволяє визначати та запускати багатоконтейнерні застосунки з конфігурацією в YAML-форматі.

Після встановлення перевіряємо роботу служби Docker:

```
systemctl status docker
```

Результат перевірки подано на рис. 4.31.

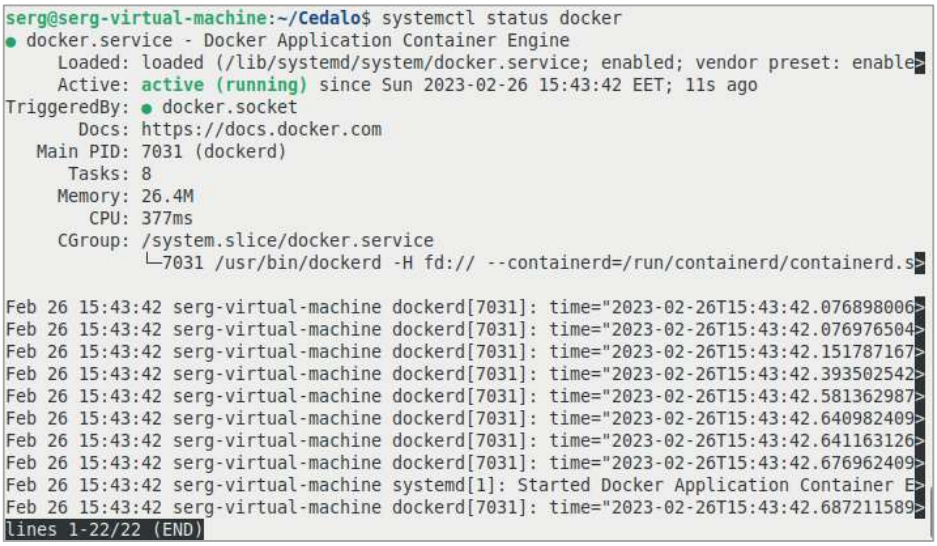


Рисунок 4.31 – Перевірка роботи служби Docker

Якщо служба не запущена, запустити її можна за допомогою команди:

```
sudo systemctl start docker
```

Після встановлення та запуску Docker, можна виконати команду:

```
docker run -it -v ~/cedalo_platform:/cedalo cedalo/installer:2-linux
```

Ця команда запускає Docker-контейнер з ім’ям «cedalo/installer:2-linux» і виконується в інтерактивному режимі з підключенням терміналу (-it). Також вказується опція -v, яка монтує локальну директорію ~/cedalo_platform до директорії /cedalo в контейнері.

Команда використовується для запуску інсталятора Cedalo Platform в Docker-контейнері і монтування директорії, яка містить налаштування та дані для платформи. Команда дозволяє інсталювати та налаштовувати платформу в середовищі, що ізольоване від решти системи, що дозволяє зберегти стабільність роботи та безпеку. Результат роботи команди подано на рис. 4.32.

```
serg@serg-virtual-machine:~/Cedalo$ sudo docker run -it -v ~/cedalo_platform:/cedalo ced
alo/installer:2-linux
Unable to find image 'cedalo/installer:2-linux' locally
2-linux: Pulling from cedalo/installer
a0d0a0d46f8b: Pull complete
4684278ccdc1: Downloading  3.757MB/36.51MB
cb39e3b315fc: Download complete
90bb485869f4: Download complete
a9b6e68f09a6: Download complete
da18d27ebad3: Download complete
9f4029d62cf1: Download complete
c01a6f0e2030: Waiting
92f5561ced99: Waiting
b2a26f7dd1b0: Waiting
2a40628157d4: Waiting
b09ba9722997: Waiting
cc699223b320: Waiting
56b108e4e071: Waiting
```

Рисунок 4.32 – Запуск інсталятора Cedalo Platform в Docker-контейнері

Коли докер завантажить платформу Cedalo, він запитає, які компоненти необхідно встановити. Можна погодитись із попереднім вибором та просто натиснути клавішу Enter (рис. 4.33).

```
alo/installer:2-linux
Unable to find image 'cedalo/installer:2-linux' locally
2-linux: Pulling from cedalo/installer
a0d0a0d46f8b: Pull complete
4684278ccdc1: Pull complete
cb39e3b315fc: Pull complete
90bb485869f4: Pull complete
a9b6e68f09a6: Pull complete
da18d27ebad3: Pull complete
9f4029d62cf1: Pull complete
c01a6f0e2030: Pull complete
92f5561ced99: Pull complete
b2a26f7dd1b0: Pull complete
2a40628157d4: Pull complete
b09ba9722997: Pull complete
cc699223b320: Pull complete
56b108e4e071: Pull complete
Digest: sha256:d37638ce9aec2f3cd2925f864929b5b656cc9ada51282467ededd5f2ab8968ff
Status: Downloaded newer image for cedalo/installer:2-linux
? Select what to install › - Space to select. Return to submit
◉   Management Center for Eclipse Mosquitto
◉   Eclipse Streamsheets
◉   Eclipse Mosquitto 2.0
○   Eclipse Mosquitto 1.6
```

Рисунок 4.33 – Підтвердження вибору компонентів

Якщо виконується тільки встановлення графічного інтерфейсу керування, то не слід вибирати встановлення Mosquitto, оскільки його вже встановлено.

В іншому випадку можна залишити налаштування за замовчуванням без змін (рис. 4.34).

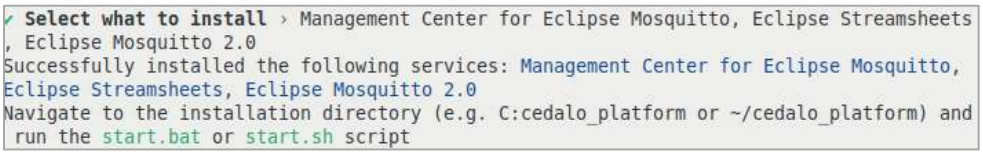


Рисунок 4.34 – Результат вибору та інсталяції пакетів

Після завершення процесу встановлення ми маємо каталог cedalo_platform у нашому домашньому кталозі /home з усіма необхідними файлами, які потрібні для запуску цього інтерфейсу керування GUI Mosquitto.

Переходимо в цей каталог:

```
cd ~/cedalo_platform
```

та запускаємо платформу Cedalo командою:

```
sudo ./start.sh
```

На рис. 4.35 подано приклад запуску платформи Cedalo.

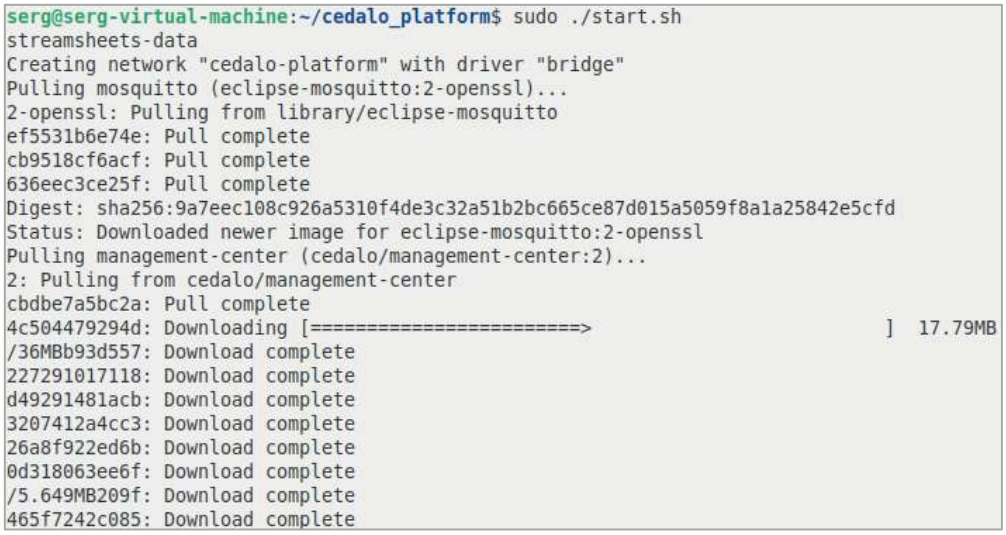


Рисунок 3.36 – Приклад запуску платформи Cedalo

Докер виконає деякі завантаження, якщо сценарій запускається вперше. Після завершення завантаження та запуску служби не треба закривати термінал, оскільки це також закриє й веб-інтерфейс керування.

За замовчуванням веб-інтерфейс центру керування працює на порту 8088, таким чином, щоб отримати доступ до нього, необхідно відкрити браузер і ввести:

```
http://localhost:8088
```

Приклад першого запуску подано на рис. 4.37.

Коли система запитує ім’я користувача та пароль, вводяться облікові дані за замовчуванням:

– користувач: cedalo;

– пароль: mmcisawesome.

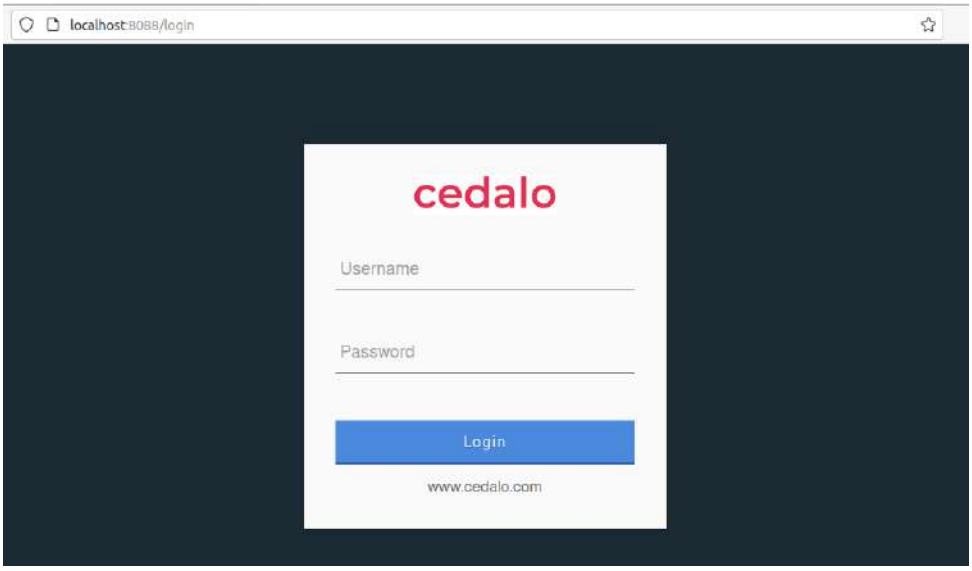


Рисунок 4.37 – Вхід в систему Cedalo

Щоб змінити дані користувача за замовчуванням, можна відредагувати файл docker-compose.yml. Крім того, URL-адреса підключення Eclipse Mosquitto:

```
mqtt://localhost:1883
```

В наступний раз вхід в систему буде виглядати таким чином (рис. 4.38).

Рисунок 4.38 – Вхід в систему за допомогою логіну та пароля

На рис. 4.39 подано приклад графічного інтерфейсу програми Cedalo.

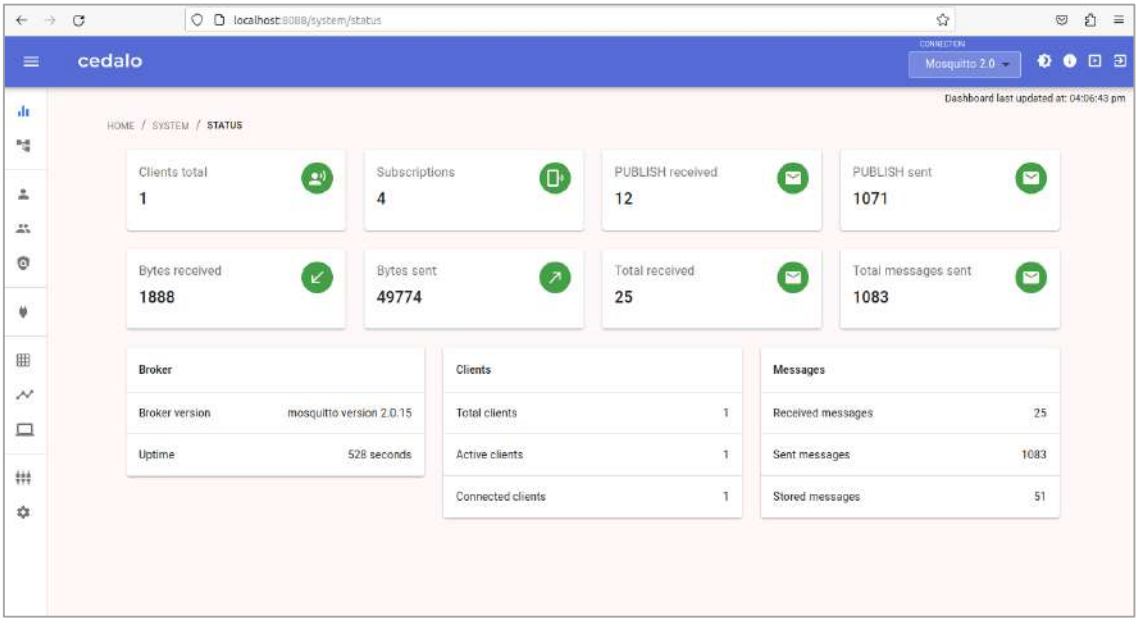


Рисунок 4.39 – Приклад графічного інтерфейсу програми Cedalo

Графічний інтерфейс надає можливість створювати нового користувача системи. Для цього потрібно в головному меню обрати пункт «Client» (1) та заповнити потрібні поля (2 та 3) (рис. 4.40).

В якості прикладу створимо користувача з ім’ям «testCedalo» та паролем «pas_testCedalo». Після заповнення полів потрібно зберегти інформацію, натиснувши кнопку «Save» (рис. 4.40, 4).

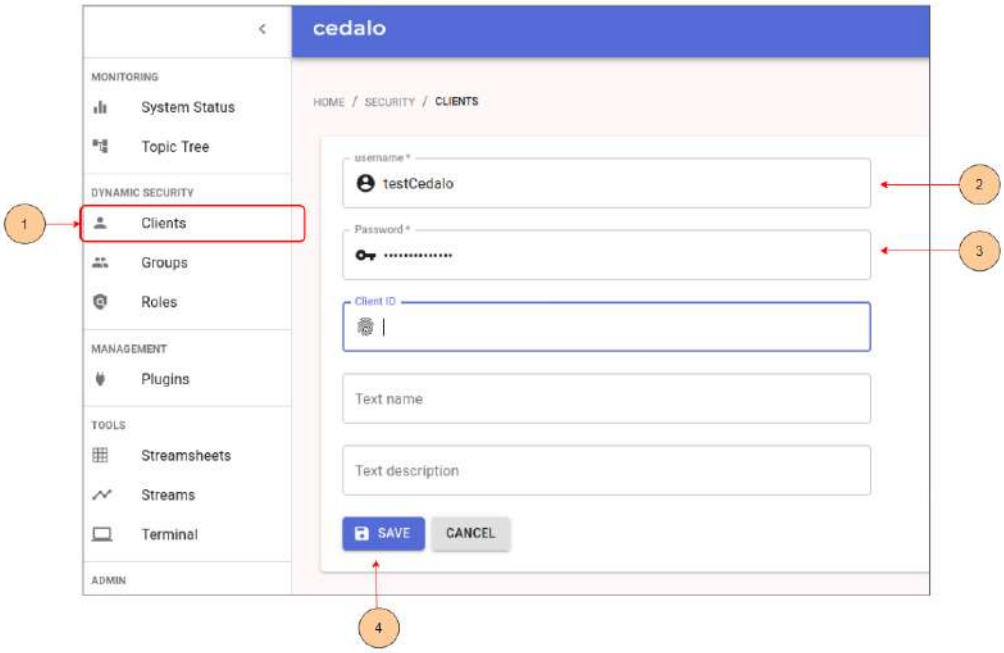


Рисунок 4.40 – Створення нового користувача

Після створення нового клієнта необхідно обрати для нього одну або декілька ролей (рис. 4.41).

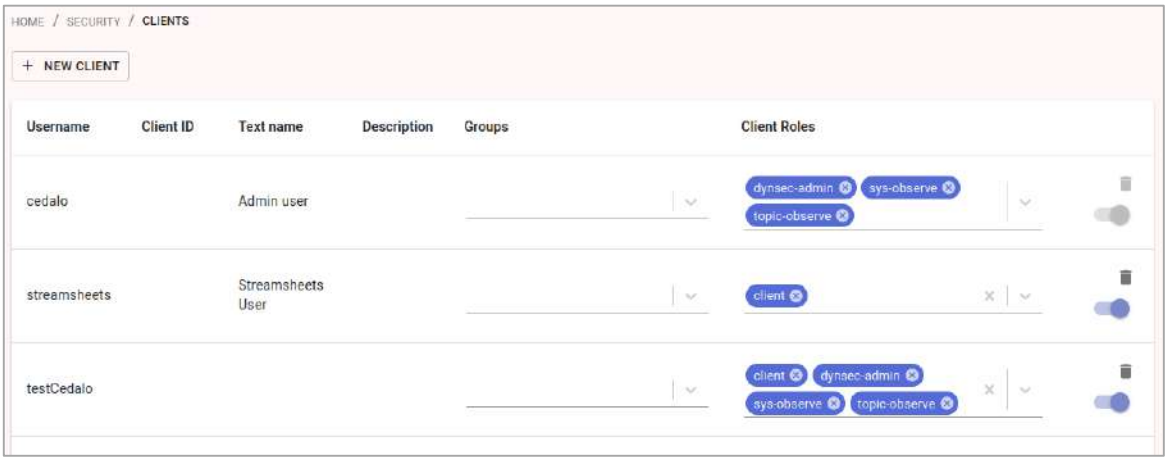


Рисунок 4.41 – Вибір необхідної ролі для користувача

Роль «Client» є обов’язковою для реалізації можливості створення та передплати на теми. Виконаємо перевірку правильності створення користувача, створивши нову тему та підписавшись на неї (рис. 4.42).

Підписка на тему відбувається за допомогою команди:

```
mosquitto_sub -h localhost -t example -u testCedalo -P pas_testCedalo
```

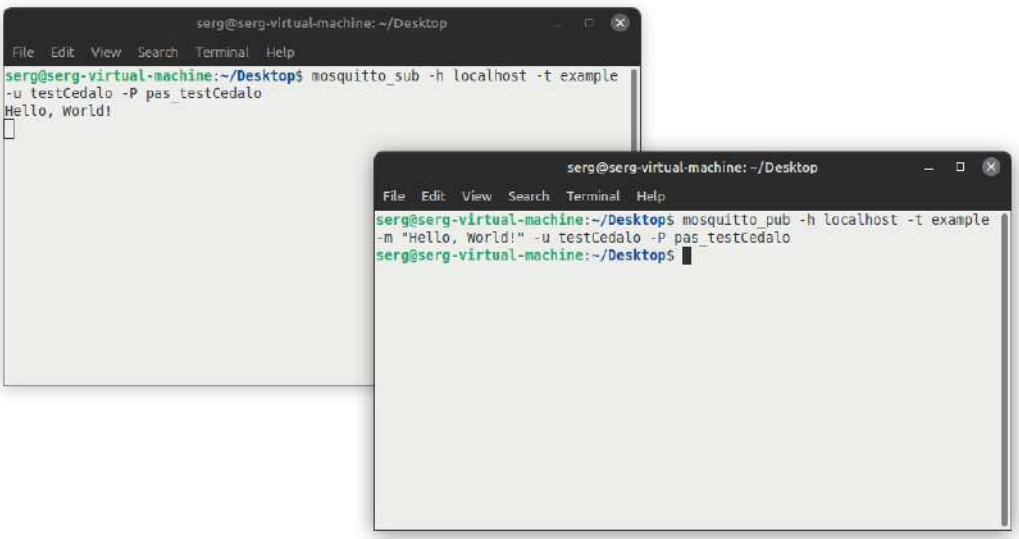


Рисунок 4.42 – Створення нової теми та передплата на неї

Публікація даних виконується наступним чином:

```
mosquitto_pub -h localhost -t example -m "Hello, World!" -u testCedalo -P pas_testCedalo
```

Поданий на рис. 4.42 приклад показав, що повідомлення доходять до підписника після публікації.

За допомогою графічного інтерфейсу Cedalo можна дізнатися інформацію про поточні теми та їх наповнення (рис. 4.43).

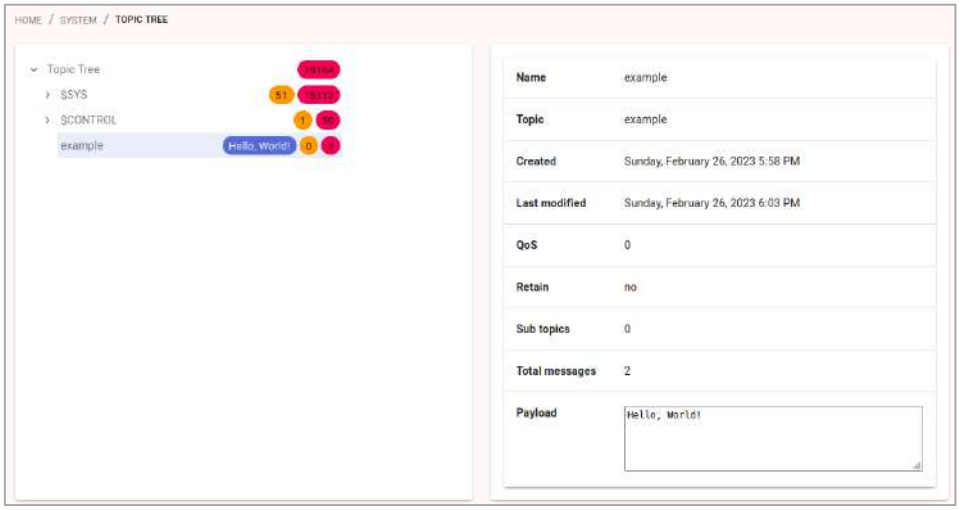


Рисунок 4.43 – Детальна інформація про поточні теми в брокері Mosquitto

Кожна тема має бути досліджена за допомогою графічного інструменту. Можна дізнатись інформацію про назву теми, вміст останнього повідомлення,

кількість підтем в темі та кількість повідомлень, що були надіслані в межах конкретної теми. На рис. 4.44 показана інформація про структуру теми.

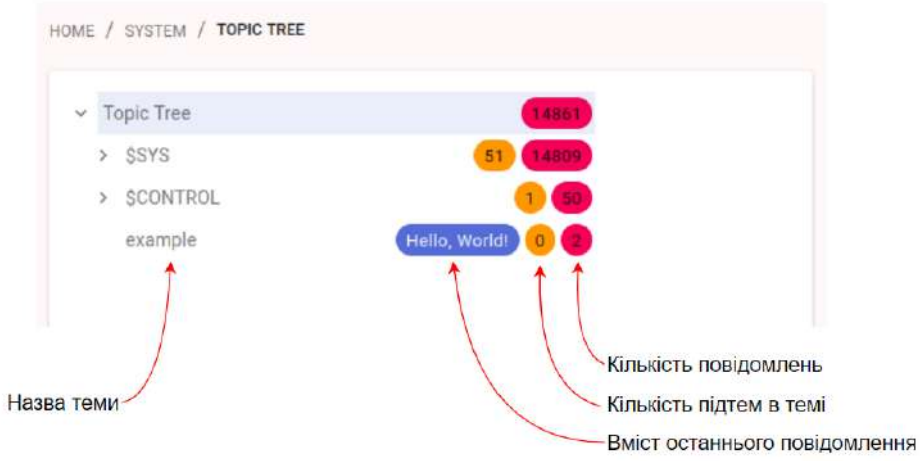


Рисунок 4.44 – Інформація про структуру теми

Для створення підтеми (subtopic) в Mosquitto необхідно вказати її ім’я у підписці (subscribe) на відповідну тему (topic) з використанням символу «+» або «#» для вказання більш загальних або конкретних підтем.

Наприклад, якщо необхідно створити підтему «sensor1» в темі «example», то можна підписатися на тему «example/sensor1» з використанням символу «/» для розділення назви теми та підтеми.

Таким чином, всі повідомлення, які будуть публікуватися в темі «example/sensor1» або будь-якій її підтемі, будуть отримуватися підписаним на цю тему.

Наприклад, щоб підписатися на всі підтеми «sensors», можна використовувати підписку на тему «sensors/#». Або для підписки на всі підтеми для певного датчика, наприклад «sensor1», можна використовувати підписку на тему «example/sensor1/#».

Щоб створити публікацію повідомлення в певну підтему, необхідно вказати її ім’я у темі (topic) при публікації повідомлення. Наприклад:

```
mosquitto_pub -t " example/sensor1/temperature" -m "25"
```

Ця команда публікує повідомлення з темою «example/sensor1/temperature» і текстом «25». Всі підписані на цю підтему клієнти отримають це повідомлення (рис. 4.45).

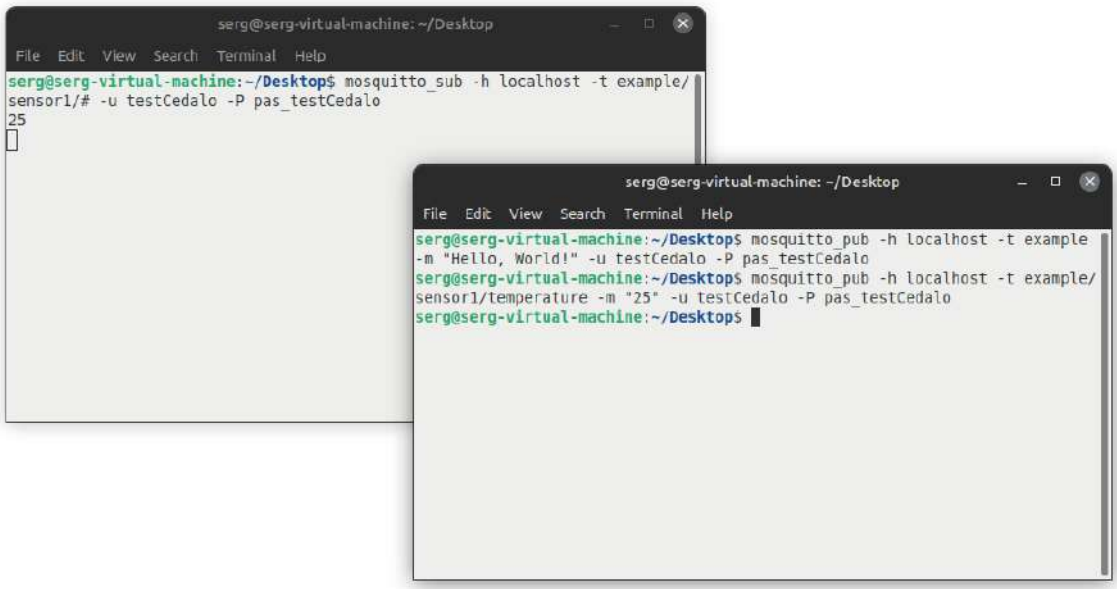


Рисунок 4.45 – Підписка та публікація в підтемі sensor1 теми example

Відповідно до доданої інформації зміниться вікно графічного інтерфейсу в Cedalo (рис. 4.46).

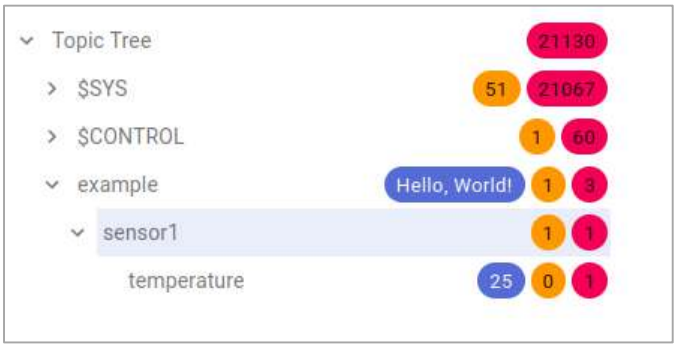


Рисунок 4.46 – Представлення інформації про підтеми та отримані повідомлення за допомогою Cedalo

Інформація в панелі керування оновлюється автоматично в реальному часі без необхідності перезавантаження інтерфейсу.

4.6 Контрольні запитання та завдання

1. Які рівні архітектури IIOT існують і які їх основні функції?
2. Які основні протоколи організації зв’язку використовуються в IIOT?
3. Які особливості протоколу MQTT і чому він популярний в IIOT?
4. Які основні характеристики протоколу MQTT?

5. Опишіть формат повідомлень за протоколом MQTT.

6. Що таке брокер MQTT Mosquitto і для чого він використовується?

7. Який порядок встановлення брокера Mosquitto на ОС Linux?

8. Як розгорнути брокер MQTT Mosquitto і які основні кроки для цього необхідні?

9. Що таке графічна платформа Cedalo і як вона використовується для управління брокером Mosquitto?

10. Які основні переваги використання графічної платформи Cedalo для управління брокером MQTT Mosquitto?

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISO/IEC Organization. 2019. ISO/IEC 21823-1 Internet of things (IoT) – Interoperability for iot systems – Part 1: Framework.

2. Standardization for emerging technologies and innovations [Електронний ресурс] : JETI. – Режим доступу: https://jtc1info.org/technology/advisory-groups/jeti/. – Станом на 01.05.2023. – Назва з екрану.

3. ISO/TC 184. Ad Hoc Group: Data Architecture of the Digital Twin. [Електронний ресурс] : International Organization for Standardization . – Режим доступу: https://www.ththry.org/activities/2020/AdHocGroup_DigitalTwin_V1R8.pdf. – Станом на 01.05.2023. – Назва з екрану.

4. Martínez-García E. A. Cyber-Physical Robotics. Cyber-Physical Systems for Industrial Transformation. Boca Raton, 2023. P. 57–74. URL: https://doi.org/10.1201/9781003262527-4 (date of access: 02.03.2024).

5. Details of the Administration Shell. Federal Ministry for Economic Affairs and Energy (BMWi) [Електронний ресурс] : ZVEI & Plattform Indutrie 4.0. – Режим доступу: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/ Publikation/ Details-of-theAsset-Administration-Shell-Part1.html. – Станом на 01.05.2023. – Назва з екрану.

6. Structure of the Administration Shell. Trilateral Perspectives from France, Italy and Germany. Ministry of Economy and Finances & Federal Ministry for Economic Affairs and Energy (BMWi) [Електронний ресурс] : Alliance Industrie du Futur, Piano Industria 4.0 & Plattform Industrie 4.0. – Режим доступу : https://www.plattformi40.de/I40/Redaktion/EN/Downloads/ Publikation/hm-2018-trilaterale-coop.html. – Станом на 01.05.2023. – Назва з екрану.

7. Details of the Administration Shell. Federal Ministry for Economic Affairs and Energy (BMWi) [Електронний ресурс] : ZVEI & Plattform Indutrie 4.0. – Режим доступу : https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/ Publikation/2018 -details-ofthe-asset-administration-shell.html. – Станом на 01.05.2023. – Назва з екрану.

8. Eclipse IoT [Електронний ресурс] : Eclipse Foundation. – Режим доступу : https://projects.eclipse.org/projects/iot. – Станом на 01.05.2023. – Назва з екрану.

9. Невлюдов І.Ш. Пневматичні пристрої та засоби автоматизації мехатронних систем: Навчальний посібник / І.Ш. Невлюдов, Л.О. Кривопляс-Володіна, С.П. Новоселов, О.В. Сичова. – Харків: ФОП Панов А.М., 2020 . – 256 с. DOI: 10.30837/978-617-7859-58-0. ISBN 978-617-7859-58-0.

10. Novoselov S., Sychova O. Automated system of technological preparation of production. Intelligent computer-integrated information technology in project and program management : Collective monograph edited by I. Linde, I. Chumachenko. Riga : ISMA. pp.207-224, 2020. DOI: https://doi.org/10.30837/ MMP.2020.207. ISBN 978-9984-891-15-6.

11. Невлюдов І.Ш. Електропневмоавтоматичні приводи в автоматизованих системах керування: Навчальний посібник / І.Ш. Невлюдов, Л.О. Кривопляс-Володіна, С.П. Новоселов, О.В. Сичова. – Харків: ХНУРЕ, 2021 . – 292 с. DOI: 10.30837/978-966-659-332-3. ISBN 978-966-659-332-3.

12. Download Node.js [Електронний ресурс] // NODE. – Режим доступу: https://nodejs.org/en/download/package-manager/. – Станом на 17.04.2024. – Назва з екрану.

13. NodeSource Node.js Binary Distributions [Електронний ресурс] // Let’s build from here. The world’s leading AI-powered developer platform.. – Режим доступу: https://github.com/nodesource/distributions/blob/master/README.md. – Станом на 17.04.2024. – Назва з екрану.

14. About npm [Електронний ресурс] // npm Docs Documentation for the npm registry, website, and command-line interface. – Режим доступу: https://docs.npmjs.com/about-npm. – Станом на 17.04.2024. – Назва з екрану.

15. Невлюдов І. Ш. Застосування цифрових двійників технічних засобів автоматизації для розроблення програмно-технічних комплексів АСУ ТП : Навчальний посібник / І. Ш. Невлюдов, С. П. Новоселов, О. В. Сичова. – Харків: Видавництво Іванченка І. С., 2023. – 267 с. ISBN 978-617-8059-95-8, DOI: 10.30837/978-617-8059-95-8/

16. Невлюдов І. Ш. Технологія програмування промислових контролерів в інтегрованому середовищі CODESYS : навч. посіб. / І. Ш. Невлюдов, С. П. Новоселов, О. В. Сичова ; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2019. – 264 с. : іл. – ISBN 978-966-659-265-4.

17. Košťál, Peter & Holubek, Radovan. (2012). The Intelligent Manufacturing Systems. Advanced science letter. 19. 10.1166/asl.2013.4816.

18. Day, Chia-Peng. (2018). Robotics in Industry – Their Role in Intelligent Manufacturing. Engineering. 4. 10.1016/j.eng.2018.07.012.

19. Griffor, E. , Greer, C. , Wollman, D. and Burns, M. (2017), Framework for Cyber-Physical Systems: Volume 1, Overview, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.SP.1500-201 (Accessed February 27, 2024).

20. Z. Ding-yi, W. Peng, Q. Yan-li and F. Lin-shen, "Research on Intelligent Manufacturing System of Sustainable Development," 2019 2nd World Conference on Mechanical Engineering and Intelligent Manufacturing (WCMEIM), Shanghai, China, 2019, pp. 657-660, doi: 10.1109/WCMEIM48965.2019.00139.

21. Z. Shen, X. Dong, Q. Fang, G. Xiong, C. -C. Ge and F. -Y. Wang, "Parallel Additive Manufacturing Systems," in IEEE Journal of Radio Frequency Identification, vol. 6, pp. 758-763, 2022, doi: 10.1109/JRFID.2022.3215600.

22. Y. Liu, Y. Zhao, L. Tao, K. Zhao and K. Li, "The Application of Digital Flexible Intelligent Manufacturing System in Machine Manufacturing Industry," 2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER), Tianjin, China, 2018, pp. 664-668, doi: 10.1109/CYBER.2018.8688303.

23. M. Segovia, J. Rubio-Hernan, A. R. Cavalli and J. Garcia-Alfaro, "Cyber-Resilience Evaluation of Cyber-Physical Systems," 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 2020, pp. 1-8, doi: 10.1109/NCA51143.2020.9306741.

24. F. Meziane, S. Vadera. K. Kobbacy. a N. Proudlove, “Intelligent systems in manufacturing: cunent developments and future prospects”, Integrated Manufacturing Systems, roc. 11, c. 4, s. 218-238, 2000.

25. Nevliudov, S. Novoselov, O. Sychova, D. Mospan and O. Klymenko, "Modeling of a Decentralized System for Maintenance of Production Equipment Based on Transport Robots," 2023 IEEE 5th International Conference on Modern Electrical and Energy System (MEES), Kremenchuk, Ukraine, 2023, pp. 1-6, doi: 10.1109/MEES61502.2023.10402485.

26. J. Zhang, B. Luo, H. Meng and Q. Li, "Research on Intelligent Warehousing Based on AGV Handling Robot," 2023 5th International Conference on Frontiers Technology of Information and Computer (ICFTIC), Qiangdao, China, 2023, pp. 1224-1227, doi: 10.1109/ICFTIC59930.2023.10455823.

27. S. A. Soundattikar, V. R. Naik and C. V. Adake, "Component Handing Automated Guided Vehicle – A Cyber Physical System Case Study," 2022 Second International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), Bhilai, India, 2022, pp. 1-6, doi: 10.1109/ICAECT54875.2022.9808022.

28. P. Tangtrustham, M. Okabe and Y. Takayanagi, "Industrial Control Product(s) with Virtualizing Technology Appropriate for CPS Platform," 2020 59th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), Chiang Mai, Thailand, 2020, pp. 13-18, doi: 10.23919/SICE48898.2020.9240412.

A. Wadsworth, M. I. Thanoon, C. McCurry and S. Z. Sabatto, "Development of IIoT Monitoring and Control Security Scheme for Cyber Physical Systems," 2019 SoutheastCon, Huntsville, AL, USA, 2019, pp. 1-5, doi: 10.1109/SoutheastCon42311.2019.9020516.

29. K. Kerliu et al., "Secure Over-The-Air Firmware Updates for Sensor Networks," 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW), Monterey, CA, USA, 2019, pp. 97-100, doi: 10.1109/MASSW.2019.00026.

30. Nevliudov I., Novoselov S., Sychova O. Modeling and practical implementation of the optimal wireless security gateway for the industrial automation network. Serbian Journal of Electrical Engineering. 2022. Vol. 19, no. 3. P. 303-327. URL: https://doi.org/10.2298/sjee2203303n (date of access: 24.03.2024).

31. S. Novoselov, "Wireless Sensor Network for Communication Between Base Stations in the Local Positioning System," 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine, 2018, pp. 383-386, doi: 10.1109/INFOCOMMST.2018.8632140.

32. Novoselov S., Nevliudov I. Technology of distributed management of production processes with the use of a service-oriented approach. "Innovative integrated computer systems in strategic project management" : Collective monograph – Riga, 2022 – pp.124-138, DOI: 10.30837/MMP.2022.124.

33. Decoupling Control. Berlin, Heidelberg : Springer Berlin Heidelberg, 2002. URL: https://doi.org/10.1007/3-540-46151-5 (date of access: 02.03.2024).

34. González J. R., Correa G., Montoya O. D. Design of a control with multiple inputs multiple outputs by decoupling. Journal of Physics: Conference Series. 2021.

Vol. 1981, no. 1. P. 012002. URL: https://doi.org/10.1088/1742-6596/1981/1/012002 (date of access: 21.03.2024).

35. Nevlyudov I., Novoselov S., Sychova O. Development and study of the operation of the module for determining the orientation of the manipulator joint. Innovative Technologies and Scientific Solutions for Industries. 2022. No.2(20). P. 86-96. URL: https://doi.org/10.30837/itssi.2022.20.086 (date of access: 24.03.2024).

36. Nevliudov, S. Novoselov, O. Sychova and D. Mospan, "Multithreaded Software Control of Industrial Manipulator Movement," 2022 IEEE 4th International Conference on Modern Electrical and Energy System (MEES), Kremenchuk, Ukraine, 2022, pp. 1-6, doi: 10.1109/MEES58014.2022.10005675.

37. Digital Twins [Електронний ресурс] : IT-Enterprise. – Режим доступу: https://www.it.ua/knowledge-base/technology-innovation/cifrovoj-dvojnik-digital-twin. – Станом на 01.05.2023. – Назва з екрану.

38. Digital Twins for Industrial Applications [Електронний ресурс] : Industrial Internet Consortium, a program of Object Management Group, Inc. (“OMG”). – Режим доступу: https://www.iiconsortium.org/pdf/IIC_Digital_Twins_Industrial_Apps_White_ Paper_2020-02-18.pdf. – Станом на 01.05.2023. – Назва з екрану.

Рекомендуємо інші наші книги:

Застосування цифрових двійників технічних засобів автоматизації для розроблення програмно-технічних комплексів АСУ ТП : Навчальний посібник / І.Ш. Невлюдов, С.П. Новоселов, О.В. Сичова. – Харків: Видавництво Іванченка І. С., 2023. – 267 с.

ISBN 978-617-8059-95-8
DOI: 10.30837/978-617-8059-95-8

У навчальному посібнику розглядаються питання практичного застосовування віртуальних макетів технічних засобів автоматизації, які використовуються для розроблення програмно-технічних комплексів сучасних АСУ ТП. Наведено приклади найпоширеніших технічних засобів автоматизації технологічних процесів і реалізації керування ними за допомогою релейно-контактної логіки.

Технології Інтернету речей в управлінні пристроями на мікроконтролерах: Навчальний посібник [Електронний ресурс] / І.Ш. Невлюдов, В.А. Андрусевич, С.П. Новоселов, О.Г. Резніченко. – Електронне видання. – Харків: ХНУРЕ, 2023. – 214 с. – pdf 2,88 Mb.

ISBN 978-966-659-364-4
DOI: 10.30837/978-966-659-364-4

У навчальному посібнику подані відомості про сучасну концепцію Інтернету речей, що дозволяє здійснювати передачу та обмін даними між фізичним світом і комп'ютерними системами в автоматичному режимі, за допомогою використання стандартних протоколів зв'язку. На реальних прикладах розглянуто процес підключення, моніторингу датчиків, віддаленого управління інтелектуальними виконавчими пристроями через Інтернет.

Новоселов С. П. Основи розробки кросплатформного програмного забезпечення на Avalonia : Навчальний посібник / С. П. Новоселов, О. В. Сичова. – Харків: Видавництво Іванченка І. С., 2024. – 267 с.

ISBN 978-617-8332-57-0
DOI: 10.30837/978-617-8332-57-0

У навчальному посібнику містяться відомості для розробки додатків на Framework Avalonia. Розглядаються різноманітні аспекти, починаючи від налаштування робочого середовища, встановлення .NET на платформі Linux, і завершуючи реалізацією доступу до бази даних та відображенням структурованої інформації.

Посібник також надає вичерпну інформацію щодо створення інтерфейсу користувача засобами Avalonia, взаємодії між візуальними елементами, використання елементів користувача, роботи з XAML та командами Avalonia. Подана інформація щодо прив'язки даних, роботи з властивостями стилю, маршрутизації подій, шаблонів даних і моделей відображення.

Навчальний посібник призначено для підготовки здобувачів освіти першого (бакалаврського), другого (магістерського) рівнів освіти в галузі електроніка, автоматизація та електронні комунікації, а також інформаційні технології. Може бути корисним розробникам програмного забезпечення, які цікавляться конструюванням користувацьких інтерфейсів з використанням Framework Avalonia. Особливо корисним цей посібник буде для тих, хто працює на платформі Linux і хоче використовувати Visual Studio Code або MS Visual Studio як інструменти розробки.

Навчальне видання

НЕВЛЮДОВ Ігор Шакирович

НОВОСЕЛОВ Сергій Павлович

СИЧОВА Оксана Володимирівна

NODE-RED ТА ТЕХНОЛОГІЯ ПРОМИСЛОВОГО ІНТЕРНЕТУ РЕЧЕЙ

Навчальний посібник

Підписано до друку 12.06.2024. Формат 60x84/16
Умовн. друк. арк. 15,5. Наклад 200 прим. Зам. № 12-06.

Видавництво і друк ФОП Іванченко І. С.
пр. Тракторобудівників, 89-а/62, м. Харків, 61135
тел.: +38(050/093)4024350
Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців, виготівників та розповсюджувачів видавничої продукції ДК № 4388 від 15.08.2012 р.
www.monograf.com.ua