

Про авторів

Георгій Землянко (ORCID: 0000-0003-4153-7608), Володимир Пєвнєв (ORCID: 0000-0002-3949-3514) — викладачі Національного аерокосмічного університету «ХАІ». Їхні наукові дослідження в галузі інформаційних технологій, телекомунікацій та кібербезпеки опубліковані в провідних міжнародних базах Scopus та Web of Science.

Як педагоги з багаторічним стажем, вони спеціалізуються на поясненні складних тем простими словами, роблячи акцент на візуалізації та практичних прикладах. Їхній підхід, перевірений у навчальних аудиторіях багатьох українських університетах, допоміг тисячам студентів освоїти основи інтерфейсів та безпеки. Ця книга — результат їхньої місії зробити сучасні технології доступними для кожного, хто бажає розуміти майбутнє цифрового світу.

м. Харків,
Україна

9 787861 785871

Шлях через епохи цифрових технологій

ІНТЕРФЕЙСИ: КЛАСИФІКАЦІЯ ТА ЕВОЛЮЦІЯ

Землянко Георгій
Пєвнєв Володимир



A Journey Through the Epochs of Digital Technologies

INTERFACES – CLASSIFICATION AND EVOLUTION

Zemlianko Heorhii
Pevnev Vladimir

Kharkiv 2026

УДК 004.7:004.3'14 / UDC 004.7:004.3'14
Ш 70

DOI: 10.5281/zenodo.21468107

Рецензенти:
д-р техн. наук, проф. Лукін Володимир Васильович (Volodymyr Lukin);
д-р техн. наук, проф. Кучук Георгій Анатолійович (Heorhii Kuchuk).

Ш 70 Шлях через епохи цифрових технологій: Інтерфейси – класифікація та еволюція: навчальний посібник / Г. А. Землянко, В. Я. Пєвнєв. – Харків : ФОП Бровін О.В., 2026. – 540 с.
ISBN 978-617-8587-66-6

У посібнику систематизовано принципи класифікації, еволюції та практичного застосування дротових і бездротових інтерфейсів у мікроконтролерних системах та Інтернеті речей. Розглянуто фізичні, електричні, логічні та програмні рівні; внутрішньокристальні шини; дротові послідовні протоколи; бездротові технології; прикладні протоколи. Особливу увагу приділено критеріям вибору оптимального інтерфейсу, безпеці на всіх рівнях та практичним кейсам. Для студентів, аспірантів та інженерів у галузях комп'ютерної інженерії, автоматизації та інформаційних технологій.

Ш 70 A Journey Through the Epochs of Digital Technologies: Interfaces – Classification and Evolution: a academic textbook / H. A. Zemlianko, V. Y. Pevnev. – Kharkiv : FOP Brovin O.V., 2026. – 540 p.
ISBN 978-617-8587-66-6

The textbook provides a systematic framework for the classification, evolution, and practical application of wired and wireless interfaces in microcontroller systems and the Internet of Things. Physical, electrical, logical, and software interface layers; on-chip buses; wired serial protocols; wireless technologies; and application protocols are examined. Special attention is paid to criteria for selecting the optimal interface, multi-layer security, and practical case studies. For students, graduate students, and engineers in the fields of computer engineering, automation, and information technology.

ISBN 978-617-8587-66-6

©H.A. Zemlianko, V.Y. Pevnev, 2026
©Землянко Г.А., Пєвнєв В.Я., 2026

ПЕРЕДМОВА

Стрімке ускладнення вбудованих систем та експоненційне зростання кількості пристроїв Інтернету речей (Internet of Things, IoT) вивели інтерфейси взаємодії на перший план як ключовий чинник надійності, енергоефективності та функціональної сумісності цифрових виробів. Сьогодні інженер-розробник має справу не з окремими лініями зв'язку, а з багаторівневим ландшафтом протоколів, що простягається від обміну регістрами всередині кристала до хмарних сервісів, керованих повідомленнями MQTT. Посібник «Шлях через епохи цифрових технологій: Інтерфейси — класифікація та еволюція» покликаний сформувати в студентів цілісне системне мислення, необхідне для свідомого вибору, програмування та діагностування інтерфейсів у проектах з мікроконтролерами та IoT. Матеріал базується на актуальних міжнародних стандартах IEEE, IETF, USB-IF, Bluetooth SIG, LoRa Alliance, технічній документації провідних виробників напівпровідників (STMicroelectronics, Espressif, Nordic Semiconductor, NXP) та наукових публікаціях у галузі комп'ютерної інженерії за період 2020–2026 років, що забезпечує відповідність викладу сучасному стану техніки.

Мета посібника полягає в тому, щоб дати читачеві фундаментальні знання про фізичні, логічні та програмні рівні організації інтерфейсів, навчити зіставляти вимоги конкретної задачі з технічними характеристиками доступних рішень і сформувати практичні навички написання апаратно-залежного коду. Структура книги відображає поступове сходження від базових понять до складних гетерогенних архітектур. Перша частина визначає систему координат — уводить класифікацію за середовищем передавання, топологією, способом синхронізації та типом даних, а також викладає методологію інженерного аналізу, спираючись на критерії пропускної здатності, латентності, енергоспоживання та завадостійкості. У другій частині детально розглянуто дротові інтерфейси: від класичних UART, SPI та I^2C до високошвидкісних USB4, Single Pair Ethernet і промислових шин CAN FD / CAN XL. Третя частина присвячена бездротовим технологіям, охоплюючи як персональні мережі (Bluetooth LE 5.4/6.0, Thread, Zigbee, UWB), так і технології великого радіусу дії (LoRaWAN, NB-IoT, супутникові NTN) та локальні високошвидкісні мережі Wi-Fi 7. Четверта частина переносить фокус на програмні протоколи — MQTT v5.0, CoAP, OPC UA, DDS — які перетворюють сукупність пристроїв на розподілену обчислювальну систему. П'ята частина інтегрує здобуті знання, демонструючи ти-

пові архітектурні шаблони IoT-систем і наскрізні приклади взаємодії, а шоста частина окреслює вектори еволюції, включаючи інтерфейси для периферійного штучного інтелекту та апаратні засоби безпеки. Завершують книгу довідкові додатки з порівняльними таблицями, розпіновками стандартних роз'ємів, прикладами коду та глосарієм, що робить посібник придатним для використання як у навчальній аудиторії, так і в інженерній лабораторії.

Роль інтерфейсів у мікроконтролерних системах і IoT неможливо переоцінити. Саме через інтерфейси мікроконтролер отримує дані від сенсорів, керує виконавчими механізмами, обмінюється повідомленнями з іншими вузлами мережі та синхронізується з глобальними сервісами. Найпростіший датчик температури, під'єднаний по I²C, є лише початком: реальний виріб часто поєднує детерміновану шину CAN FD для критичних до часу контурів керування, енергоефективний BLE для взаємодії з мобільним додатком, резервний канал LoRaWAN для телеметрії на відстані в десятки кілометрів та Ethernet-порт для повноцінного виходу в локальну мережу підприємства. Така різноманітність вимагає від фахівця не механічного запам'ятовування набору команд, а розуміння моделей синхронізації, арбітражу, виявлення колізій, механізмів гарантованої доставки та особливостей апаратних контролерів. Посібник послідовно формує цей понятійний апарат, ілюструючи його діаграмами часових діаграм, структурними схемами периферійних блоків, графіками залежності швидкості від довжини лінії та таблицями порівняльних характеристик. Усі теоретичні положення підкріплено посиланнями на відповідні розділи стандартів — як-от IEEE 802.3cg (10BASE-T1S), IEEE 802.15.4z (UWB), IEC 61158 (типи польових шин) чи специфікації Bluetooth Core 6.0 — що дозволяє за потреби заглибитися в першоджерела.

Для того щоб абстрактні поняття набули практичного втілення, книга побудована навколо трьох наскрізних апаратних платформ, які охоплюють найпоширеніші сценарії застосування. Першою платформою обрано мікроконтролери сімейства STM32 на ядрі ARM Cortex-M, що вирізняються розвиненою периферією, багаторівневим контролером DMA, апаратною підтримкою CAN FD, USB OTG та Ethernet MAC, і які супроводжуються потужним інструментарієм CubeMX та HAL/LL-бібліотеками. Другою платформою є ESP32 виробництва Espressif Systems, що інтегрує бездротові підсистеми Wi-Fi 6 та Bluetooth LE 5.4 із двома ядрами Xtensa або RISC-V, а також містить велику кількість апаратних інтерфейсів, зокрема SPI, I²C, I²S, SDIO, що робить його ідеальним полігоном для вивчення IoT-протоколів. Третьою платформою є Raspberry Pi Pico (чіп RP2040) із

унікальним апаратним блоком PIO (Programmable I/O), який дає змогу програмно реалізовувати довільні цифрові протоколи на зразок однопровідних шин, манчестерського кодування або спеціалізованих відеоінтерфейсів без завантаження основного процесора. У кожному розділі, де це доречно, наведено лістинги ініціалізації та обробників переривань для всіх трьох платформ, а також описано підключення зовнішніх мікросхем фізичного рівня, трансиверів або радіомодулів, що дозволяє відтворити приклади в лабораторії з мінімальними додатковими витратами.

Книга спроектована так, щоб читач міг рухатися як послідовно, від першого до останнього розділу, так і вибірково, звертаючись до конкретного інтерфейсу за потреби. Автор рекомендує студентам молодших курсів зосередитися на частинах I та II, виконуючи запропоновані лабораторні завдання на платформі STM32, тоді як магістранти та інженери-практики знайдуть найбільшу користь у розділах, присвячених бездротовим mesh-мережам, протоколу Matter, програмним інтерфейсам OPC UA та архітектурним шаблонам побудови Edge AI-вузлів. Посібник не претендує на вичерпний переказ специфікацій — він слугує навігатором у сучасному просторі цифрових комунікацій, який допоможе сформулювати правильні запитання та обрати адекватні інженерні рішення. Сподіваємося, що запропонована подорож епохами цифрових технологій стане для читача надійним підґрунтям для подальшої професійної творчості.

ВДЯЧНІСТЬ

Цю книгу автори присвячують світлій пам’яті та науковому доробку свого вчителя, друга і колеги — Плахтеєва Анатолія Павловича, професора Харківського аерокосмічного університету «ХАІ». Його глибокі дослідження в галузі комп’ютерних систем, вбудованих обчислень і методології інженерної освіти стали фундаментом, на якому постав цей посібник. Численні наукові статті, розроблені ним навчальні курси та створені лабораторні стенди перетворили абстрактні інтерфейсні протоколи на зрозумілі й відтворювані інженерні рішення, якими сьогодні користуються сотні випускників університету.

Анатолій Павлович був не лише блискучим ученим, а й мудрим наставником. Саме він прищепив авторам переконання, що справжнє розуміння цифрових комунікацій починається не з бібліотеки викликів, а з усвідомлення фізичних обмежень лінії зв’язку, з уміння читати часові діаграми осцилографа та з поваги до стандартів як до узагальненого досвіду поколінь розробників. Його настанови незмінно спонукали шукати відповіді у першоджерелах — специфікаціях IEEE, технічних нотатках виробників, математичному апараті теорії сигналів, — і водночас ніколи не втрачати з поля зору кінцеву мету: створення надійного, відтворюваного та енергоефективного виробу.

Автори висловлюють безмежну вдячність за багаторічну підтримку, терпляче наставництво та ту високу планку наукової доброчесності, яку Анатолій Павлович задавав власним прикладом. Його вміння бачити красу в лаконічному апаратному дизайні, його віра в студентів та колег і постійна готовність до щирої наукової дискусії залишаються моральним орієнтиром для всіх, хто мав честь із ним працювати. Нехай цей посібник слугуватиме скромним продовженням його просвітницької місії — допомагати новому поколінню інженерів впевнено прокладати шлях через епохи цифрових технологій.

ЧАСТИНА І.
ФУНДАМЕНТАЛЬНІ ЗАСАДИ ІНТЕРФЕЙСІВ ЦИФРОВИХ СИСТЕМ

РОЗДІЛ 1.
БАЗОВІ ПОНЯТТЯ ТА КЛАСИФІКАЦІЯ ІНТЕРФЕЙСІВ

1.1. Визначення інтерфейсу: фізичний, електричний, логічний та програмний рівні

У сучасній технічній літературі термін «інтерфейс» (від англійського interface — поверхня поділу, стик) уживають у надзвичайно широкому діапазоні значень: від фізичного роз'єму на корпусі приладу до абстрактного контракту, якого зобов'язані дотримуватися програмні модулі. Така полісемія часто призводить до непорозумінь, особливо коли в одному проекті зустрічаються фахівці зі схемотехніки, вбудованого програмного забезпечення та хмарних сервісів. Щоб уникнути двозначності, цей посібник спирається на багаторівневу модель, яка розглядає інтерфейс як сукупність узгоджених специфікацій на чотирьох взаємопов'язаних, але функціонально самостійних рівнях: фізичному, електричному, логічному та програмному. Такий підхід корелює з еталонною моделлю взаємодії відкритих систем ISO/IEC 7498-1, однак адаптований саме до особливостей вбудованих мікроконтролерних пристроїв, де рівні не завжди відокремлені формальними межами, а часто реалізуються в межах одного кристала.

Фізичний рівень інтерфейсу визначає матеріальну, дотикову складову з'єднання: геометрію та розпіновку конекторів, тип і хвильовий опір кабелю, кількість провідників, спосіб екранування, масо-габаритні характеристики роз'ємів, а також механічні допуски, зусилля фіксації та допустиму кількість циклів зчленування. Наприклад, фізичний рівень шини CAN визначає використання витої пари з хвильовим опором 120 Ом, тоді як для USB 3.2 Gen 2×2 регламентується застосування спеціального кабелю типу Type-C із вісьмома диференційними парами та суворими обмеженнями на перехресні завади між ними. У контексті мікроконтролерних систем фізичний рівень часто є найпомітнішою частиною інтерфейсу для студента, який уперше бере до рук налагоджувальну плату, адже саме на цьому рівні виникають питання на кшталт «куди під'єднати землю» та «чому сигнал спотворюється без термінуючого резистора». Відповідно до нормативних документів, зокрема IEC 61169 (радіочастотні конектори), IEC 60603 (з'єднувачі для друкованих плат) та фірмових стандартів USB-IF, фізичний рівень завжди описується кресленниками з контрольованими розмірами, які гарантують механічну сумісність виробів різних виробників. У посібнику

фізичний рівень ілюструватиметься фотографіями реальних модулів, а також спрощеними кресленнями розпіновок, що допоможуть читачеві безпомилково здійснити монтаж з'єднань у лабораторії.

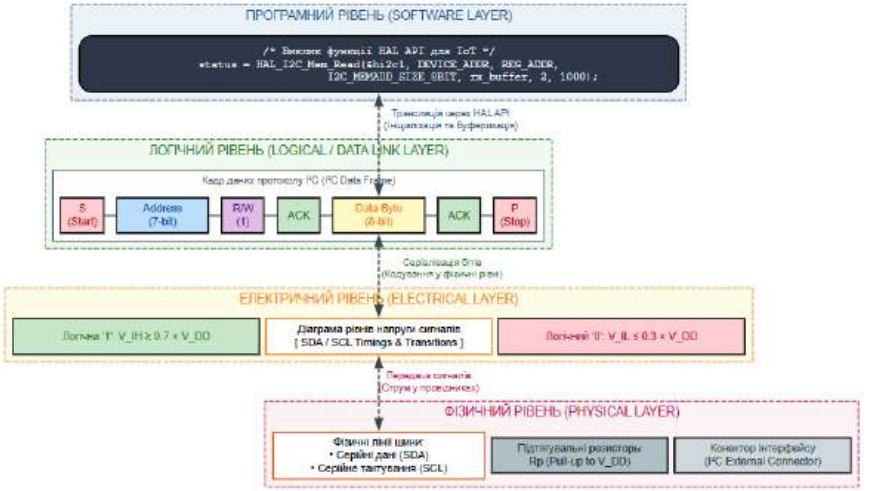


Рисунок 1.1 – Багаторівнева модель інтерфейсу на прикладі шини I²C

Електричний рівень інтерфейсу оперує такими категоріями, як рівні напруги логічного нуля та одиниці, порогова чутливість приймача, вихідний струм передавача, швидкість наростання фронту сигналу, джиттер, очні діаграми та вимоги до цілісності сигналу. Саме на цьому рівні визначається, чи працюватиме конкретний датчик із 1,8-вольтовою логікою на одній шині з 5-вольтовим мікроконтролером, і саме тут стають критичними такі параметри, як максимальна ємність лінії для I²C (400 пФ для швидкого режиму згідно зі специфікацією NXP UM10204) або диференційна напруга порогу для RS-485 (±200 мВ згідно з TIA/EIA-485-A). Електричний рівень тісно пов'язаний із фізичним, оскільки геометрія провідників і діелектричні властивості матеріалу плати безпосередньо впливають на спотворення фронтів імпульсів та перехресні завади, проте логічно він є окремою сутністю: один і той самий фізичний конектор може обслуговувати різні електричні протоколи (наприклад, роз'єм RJ-45 використовується і для 10BASE-T, і для 10GBASE-T, хоча електричні вимоги до сигналів кардинально відрізняються). Для мікроконтролерних систем особливого значення набувають сімейства стандартів JEDEC JESD8 (рівні логіки CMOS і TTL) та специфікації IBIS (Input/Output Buffer Information Specification), які дозволяють моделювати поведінку виводів мікросхем до виготовлення

прототипу. У навчальному контексті електричний рівень часто є джерелом найбільших труднощів, оскільки вимагає від студента водночас володіти навичками роботи з осцилографом і розуміти фізичний зміст явищ, що спостерігаються на екрані. Тому в посібнику для кожного дротового інтерфейсу наводяться типові осцилограми із позначенням критичних часових параметрів (час встановлення, час утримання, тривалість фронту), а також таблиці порогових рівнів, узяті безпосередньо з технічної документації виробників.

Логічний рівень інтерфейсу абстрагується від вольтів і пікосекунд, натомість оперуючи бітами, символами, кадрами, пакетами та правилами, за якими ці абстрактні сутності формуються, передаються, розпізнаються та підтверджуються. Саме на логічному рівні визначаються формат адреси пристрою, механізм арбітражу при одночасній спробі доступу до шини, алгоритм обчислення контрольної суми, структура стартових і стопових умов, а також процедура виявлення та виправлення помилок. Для прикладу, логічний рівень протоколу I²C (Inter-Integrated Circuit) встановлює, що передача починається зі стартової умови S (перехід SDA з високого рівня в низький при високому рівні SCL), після чого слідує 7- або 10-бітна адреса веденого пристрою, біт напрямку R/W, біт підтвердження ACK, байти даних і стопова умова P. Усі ці правила зафіксовані в специфікації I²C-bus specification and user manual компанії NXP Semiconductors (документ UM10204, ревізія 7.0, 2021 р.), і їх порушення призводить до того, що пристрій просто не відгукується на запити. Логічний рівень CAN-шини, своєю чергою, оперує поняттями домінантного та рецесивного бітів, арбітражу на основі ідентифікатора повідомлення, а також чотирма типами кадрів (дані, віддалений запит, помилка, перевантаження), що описані в міжнародному стандарті ISO 11898-1:2015. Варто зауважити, що в сучасних мікроконтролерах переважна більшість логічних функцій реалізована апаратно: наприклад, периферійний блок I²C у кристалах STM32F4 автоматично генерує стартову умову після запису в регістр CR1, відстежує біти підтвердження та виставляє прапорці в регістрі статусу. Таким чином, розробник програмного забезпечення може не замислюватися над мікротактами та фронтами, однак розуміння логічного рівня є необхідною передумовою для ефективного налагодження: коли логічний аналізатор показує відсутність біта ACK на дев’ятому такті, фахівець має миттєво зрозуміти, що ведений пристрій не розпізнав адресу або перебуває в стані скидання.

Таблиця 1.1 – Порівняння рівнів моделі інтерфейсу для найпоширеніших протоколів

Назва протоколу	Характеристика	Значення
UART	*Фізичний рівень*	Провідники TX, RX, GND; апаратне керування RTS/CTS; роз'єми — штирові колодки, DB 9 (для RS 232)
	Електричний рівень	TTL (0–3,3/5 В) або RS 232 (біполярний, ±3…±15 В)
	Логічний рівень	Асинхронний старт-стоповий кадр: старт біт, 5–9 біт даних, біт парності, стоп біти; фіксована швидкість
	Програмний рівень	HAL_UART_Transmit(), termios (Linux), pyserial, стандартний послідовний порт ОС
SPI	*Фізичний рівень*	4 лінії: SCLK, MOSI, MISO, SS (CS); з'єднання штирями, рідше IDC-роз'єми
	Електричний рівень	CMOS/TTL (0 В – Vdd, типово 3,3 В або 5 В)
	Логічний рівень	Синхронний ведучий/ведений; вибір веденого активним рівнем CS; режими CPOL/CPHA, передача старшим/молодшим бітом
	Програмний рівень	HAL_SPI_TransmitReceive(), spidev (Linux), Arduino SPI library
I²C	*Фізичний рівень*	Дві лінії SDA, SCL + GND; обов'язкові підтягувальні резистори до Vdd; штирьове під'єднання
	Електричний рівень	Відкритий колектор/стік; рівні напруги відносно Vdd (1,8/3,3/5 В)
	Логічний рівень	Адресація 7 або 10 біт; старт/стоп-умови; біт підтвердження ACK/NACK; швидкості 100 кбіт/с (Standard), 400 кбіт/с (Fast)
	Програмний рівень	HAL_I2C_Mem_Write(), smbus2 (Python), i2c-tools
CAN	*Фізичний рівень*	Вита пара 120 Ом, DB 9 (CAN_H, CAN_L, GND), термінувальні резистори на кінцях
	Електричний рівень	Диференційний сигнал: CAN_H / CAN_L, домінанта ≈2 В, рецесив ≈0 В
	Логічний рівень	Кадри даних/віддаленого запиту; ідентифікатор 11/29 біт; не руйнівний арбітраж CSMA/CR; CRC та ACK
	Програмний рівень	HAL_CAN_AddTxMessage(), SocketCAN (Linux), can-utils

Кінець таблиці 1.1

Назва протоколу	Характеристика	Значення
USB 2.0	*Фізичний рівень*	4 дроти: VBUS, D+, D−, GND; конектори Type A, Type B, mini/micro USB
	Електричний рівень	Диференціальна пара D+/D−, рівні 3,3 В; LS (1,5 Мбіт/с), FS (12 Мбіт/с), HS (480 Мбіт/с); NRZI-кодування з біт-стафінгом
	Логічний рівень	Кадри 1 мс/125 мкс; токени (IN/OUT/SETUP), пакети даних DATA0/1, handshake; дескриптори пристрою, кінцеві точки (endpoints)
	Програмний рівень	libusb, WinUSB, PyUSB, стек ядра (драйвери usbhid, cdc_acm тощо)
BLE	*Фізичний рівень*	2,4 ГГц ISM, 40 каналів (37 даних + 3 рекламних), GFSK-модуляція, інтегрована/зовнішня антена
	Електричний рівень	Радіочастотний (бездротовий); типова вихідна потужність ≤+10 дБм, чутливість приймача ≤ 70 дБм (для 1 Мбіт/с LE)
	Логічний рівень	Стани Link Layer (Advertising, Scanning, Initiating, Connection); пакети Advertising/Data; адреси (публічні/рандомні); ієрархія GATT (профілі, сервіси, характеристики)
	Програмний рівень	BlueZ D Bus API, NimBLE, ESP IDF BLE GATT API, Android BLE API, Web Bluetooth
MQTT	*Фізичний рівень*	Транспорт TCP/IP (Ethernet, Wi-Fi, стільникова мережа); власного фізичного рівня не визначає
	Електричний рівень	Залежить від використовуваного мережевого стеку та фізичного середовища
	Логічний рівень	Повідомлення CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE; теми з підставними знаками (+/#); QoS 0, 1, 2; збережені повідомлення (Retain)
	Програмний рівень	Paho MQTT (C/C++, Python), mosquitto_pub/sub, ESP MQTT, хмарні SDK (AWS IoT, Azure)

Програмний рівень інтерфейсу є найвищим у запропонованій моделі та описує спосіб, у який прикладне програмне забезпечення звертається до послуг нижчих рівнів. Він охоплює абстракції, реалізовані у вигляді бібліотек, драйверів, програмних інтерфейсів (API) та стеків протоколів. У контексті мікроконтролерів STM32 програмний рівень представлений бібліотеками HAL (Hardware Abstraction Layer) та LL (Low Layer), які надають

функції на кшталт HAL_SPI_TransmitReceive() або HAL_UART_Receive_IT(), приховуючи від користувача деталі запису в регістри керування та обробки прапорців переривань. Для платформи ESP32 компанія Espressif пропонує фреймворк ESP-IDF, що містить програмні драйвери для всіх апаратних периферійних блоків, а також реалізації стеків BLE (Bluedroid), Wi-Fi, Ethernet (LwIP) і MQTT. На програмному рівні інтерфейс перетворюється на набір функцій і структур даних, які програміст викликає у своєму коді; при цьому якість реалізації програмного рівня безпосередньо впливає на такі нефункціональні характеристики системи, як час реакції на переривання, споживання оперативної пам'яті та пропускна здатність каналу зв'язку. Важливим аспектом програмного рівня є модель обробки помилок: якщо на логічному рівні помилка виявляється через контрольну суму або відсутність підтвердження, то програмний рівень повинен коректно передати цю інформацію прикладному коду (наприклад, повернути код HAL_ERROR або викликати callback-функцію з відповідним статусом). У бездротових інтерфейсах програмний рівень ускладнюється додатково через необхідність керувати станами з'єднання, процедурами pairing та bonding (для BLE), а також механізмами повторної передачі пакетів, що передбачені специфікацією Bluetooth Core 5.4.

Доцільно наголосити, що запропонований чотирирівневий поділ не є абсолютно жорстким: на практиці зустрічаються інтерфейси, де межі між рівнями розмиті. Так, в однопровідному протоколі 1-Wire компанії Maxim Integrated фізичний, електричний та логічний рівні настільки тісно інтегровані, що специфікація описує їх у нерозривній єдності — часові слоти читання й запису безпосередньо прив'язані до тривалості імпульсів низького рівня, що формуються шляхом переведення виводу мікроконтролера в стан виходу з відкритим стоком. Аналогічно, у інтерфейсі I²S (Inter-IC Sound) логічний рівень визначає вирівнювання аудіоданих відносно сигналу вибірки слова WS, однак ця логічна специфікація має безпосередні наслідки для конфігурації контролера прямого доступу до пам'яті (DMA), тобто проникає на програмний рівень. Проте саме чотирирівневий аналіз дає змогу студенту системно підходити до вивчення будь-якого нового протоколу, послідовно відповідаючи на запитання: «як фізично з'єднати пристрої?», «які рівні напруги та швидкості допустимі?», «як формуються бітові послідовності та адреси?» і «якими функціями бібліотеки це реалізується?».

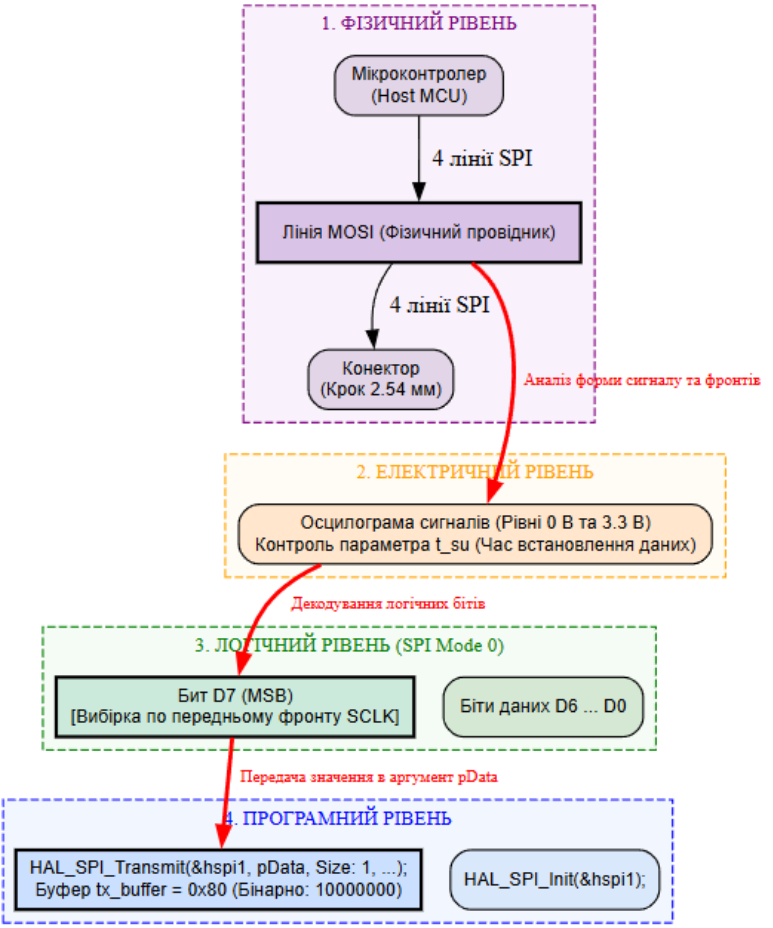


Рисунок 1.2 – Приклад наскрізного аналізу інтерфейсу SPI у чотирирівневій моделі

У міжнародній науково-технічній літературі питання багаторівневого представлення інтерфейсів висвітлено, зокрема, у роботі А. С. Таненбаума «Комп'ютерні мережі» (шостий друк, 2021 р.), де застосовано класичну модель OSI, та в монографії Дж. Акса «Системна інженерія вбудованих пристроїв» (2023 р.), яка адаптує цю модель до мікроконтролерних систем. Виробники мікросхем також приділяють увагу чіткому розмежуванню рівнів: у посібнику з розробки STM32CubeIDE (STMicroelectronics, документ UM2609, ревізія 5.0) явно виділено апаратний, низькорівневий та високорівневий шари абстракції, що відповідає запропонованій у цій книзі моделі.

Схожий підхід демонструє компанія Texas Instruments у технічній нотатці SLVAE57 «Understanding Interface Layer Models for MSP430 MCUs». Отже, студент, який опанує чотирирівневе мислення, отримує універсальний інструмент для декомпозиції будь-якої інтерфейсної задачі — від підключення простого датчика температури до проєктування розподіленої системи керування промисловим роботом.

1.2. Основні критерії класифікації

Розмаїття інтерфейсів, що застосовуються в сучасних мікроконтролерних системах та пристроях Інтернету речей, потребує впорядкування за допомогою чітко визначених критеріїв, які дозволяють інженерові швидко зіставити вимоги проєкту з характеристиками доступних рішень. У цьому посібнику класифікацію побудовано на основі п’яти фундаментальних ознак, які в сукупності дають вичерпне уявлення про будь-який інтерфейс: середовище передавання, топологія, спосіб синхронізації, напрямок передачі та тип даних. Кожен із цих критеріїв розглядається як незалежна вісь багатовимірного простору, що уможливлює точне позиціонування інтерфейсу відносно аналогів. Запропонований підхід узгоджується з методологією, викладеною в міжнародному стандарті ISO/IEC 2382:2015 «Інформаційні технології — Словник», а також із практикою системної інженерії, описаною в роботах Е. Лі та С. Сешіа «Вступ до вбудованих систем» (Lee & Seshia, 2023). Подальші підрозділи послідовно розкривають кожен із названих критеріїв, починаючи з найнаочнішого — фізичного середовища, яким поширюється сигнал.

1.2.1. За середовищем передавання: дротові, бездротові, оптичні

Середовище передавання є тією матеріальною субстанцією або фізичним полем, через яке від передавача до приймача переміщується енергія, що переносить інформацію. Від властивостей середовища безпосередньо залежать досяжна швидкість передавання, максимальна відстань, завадостійкість, складність приймально-передавальної апаратури, енергетичний бюджет та вартість розгортання системи. Історично склалося так, що перші цифрові інтерфейси були виключно дротовими, однак потреби мобільності, розподіленого моніторингу та неможливість прокладання кабелів у важкодоступних місцях зумовили бурхливий розвиток бездротових тех-

нологій, тоді як оптичні лінії зайняли нішу надшвидкісних і повністю гальванічно розв'язаних з'єднань. На сьогодні всі три типи середовищ співіснують, часто доповнюючи одне одного в межах єдиної IoT-системи.

Дротові інтерфейси використовують металеві провідники — переважно мідні — для передавання електричних сигналів. У вбудованих системах застосовуються як прості однопровідні лінії зі зворотним струмом через спільну землю, так і складні багатожильні кабелі з екрануванням. Дротове середовище відзначається високою завадостійкістю за умови коректного узгодження імпедансів і застосування диференційних сигналів, що закріплено в стандартах TIA/EIA-485-A (RS-485), ISO 11898 (CAN) та IEEE 802.3 (Ethernet). Електричний сигнал у металевому провіднику поширюється зі швидкістю, близькою до $2 \cdot 10^8$ м/с, однак неминучі втрати в діелектрику, скін-ефект на високих частотах і паразитна ємність лінії обмежують одночасне досягнення великої дальності та високої пропускної здатності. Це фундаментальне обмеження описане у формі діаграми Шеннона-Хартлі, і його наслідком є те, що, наприклад, інтерфейс USB 4 версії 2.0, який оперує швидкістю до 80 Гбіт/с, вимагає кабелю завдовжки не більше 0,8 м, тоді як низькошвидкісний інтерфейс RS-485 на швидкості 9600 біт/с стійко функціонує на відстанях понад кілометр. Дротові інтерфейси забезпечують також найпростіший спосіб живлення віддалених пристроїв за тією ж лінією зв'язку — технології Power over Ethernet (IEEE 802.3af/at/bt), Power over Data Line в автомобільних мережах та фантомне живлення мікрофонів є тому яскравими прикладами. У контексті мікроконтролерних систем саме дротові інтерфейси формують кістяк внутрішньосхемних з'єднань (SPI, I²C, UART), оскільки на друкованій платі відстані не перевищують десятків сантиметрів, а контроль над фізичним середовищем є абсолютним.

Бездротові інтерфейси використовують як середовище вільний простір, де енергія переноситься електромагнітними хвилями радіочастотного або, в окремих випадках, оптичного діапазону. Відсутність фізичного кабелю радикально спрощує розгортання мереж на великих територіях, у рухомих об'єктах або в агресивних середовищах, де контакти зазнають корозії. Однак платою за цю гнучкість є принципова відкритість каналу, що породжує проблеми електромагнітної сумісності, колізій пакетів, завмирань сигналу через багатопроменеве поширення та вразливість до навмисних перешкод. Усі ці явища детально змодельовані в літературі з бездротової комунікації, зокрема в класичній праці Т. С. Раппапорта «Бездротовий зв'язок: принципи та практика» (Rappaport, 2024), і враховані під час роз-

роблення міжнародних стандартів, якими регулюються бездротові інтерфейси: IEEE 802.11 (Wi-Fi), IEEE 802.15.1 (Bluetooth), IEEE 802.15.4 (Zigbee, Thread) та серія специфікацій 3GPP (LTE-M, NB-IoT, 5G NR).

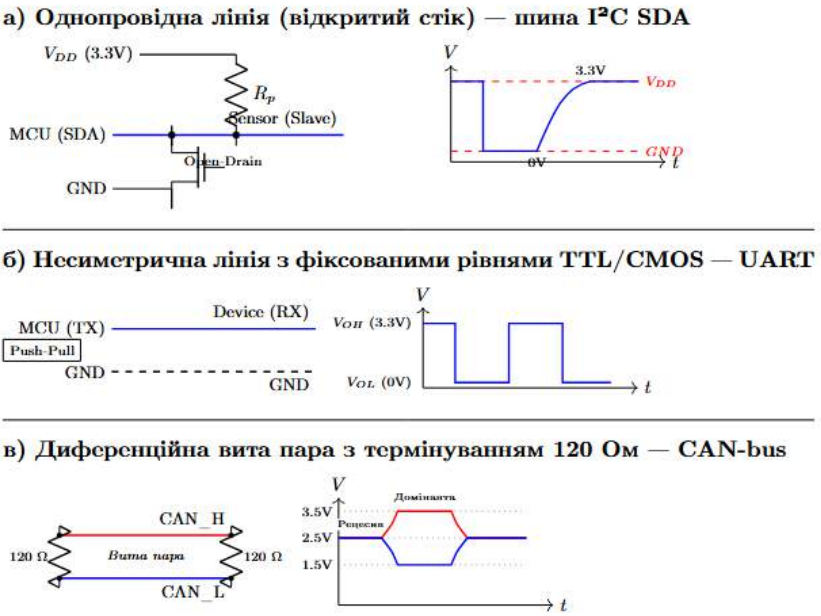


Рисунок 1.3 – Типи дротових ліній у мікроконтролерних системах

Ключовими параметрами, за якими порівнюють бездротові інтерфейси, є робоча частота, ширина смуги каналу, вихідна потужність передавача, чутливість приймача, коефіцієнт підсилення антени та використовувана схема модуляції. Для прикладу, технологія LoRa (Long Range), що регулюється специфікаціями LoRa Alliance, використовує модуляцію CSS (Chirp Spread Spectrum) у неліцензованих діапазонах 868 МГц (Європа) та 915 МГц (Америка), досягаючи чутливості до -148 дБм і дальності до 15 км у сільській місцевості, тоді як Bluetooth LE у діапазоні 2,4 ГГц із модуляцією GFSK забезпечує швидкість до 2 Мбіт/с, але типову дальність у межах 10–50 м у приміщенні. Особливе місце посідають надширокосмугові (UWB) інтерфейси за стандартом IEEE 802.15.4z, що використовують короткі імпульси тривалістю близько 2 нс для досягнення точності визначення відстані в сантиметровому діапазоні. Для мікроконтролерних систем суттєвим є також те, що бездротовий інтерфейс майже завжди реалізується

окремою мікросхемою радіотракту (або інтегрованим у SoC радіомодулем), котра з'єднується з обчислювальним ядром через стандартний дротовий інтерфейс — SPI, UART або SDIO, — що ще раз підкреслює нерозривність обох середовищ у реальних пристроях.

Таблиця 1.2 – Порівняльні характеристики дротових, бездротових та оптичних середовищ передавання

Середовища	Характеристика	Значення
Дротові (мідні)	*Тип носія інформації*	Електричний струм/напруга (симетричні лінії, коаксіал)
	Максимальна швидкість (2026 р.)	10 Гбіт/с (10GBASE-T, IEEE 802.3an) по витій парі; 10 Гбіт/с (DOCSIS 4.0) по коаксіалу; до 25 Гбіт/с на коротких дистанціях (25GBASE-T, IEEE 802.3bq)
	Максимальна дальність без повторювачів	100 м (вита пара Cat6a для 10GBASE-T); 200–500 м (коаксіал RG 6, залежно від частоти); 1–3 км для xDSL по мідному телефонному кабелю
	Завадостійкість	Висока завдяки екрануванню та диференціальній передачі (STP, коаксіал). Сприйнятливе до сильних електромагнітних імпульсів
	Складність приймача / передавача	Низька: трансформаторна розв'язка, лінійний драйвер (PHY Ethernet), дешеві роз'єми RJ45. Прості мікросхеми
	Живлення по лінії	Так, стандарти PoE (IEEE 802.3af/at/bt) до 90 Вт по витій парі; PoDL для однопарного Ethernet (IEEE 802.3bu)
	Гальванічна розв'язка	Забезпечується імпульсними трансформаторами в роз'ємі (наприклад, у Ethernet до 1500 В змінного струму); потрібна в схемі
	Типові представники	Ethernet (Cat5e/6/6a, IEEE 802.3), коаксіальний DOCSIS (кабельне ТБ), xDSL (ITU-T G.992), однопарний Ethernet (10BASE-T1L, IEEE 802.3cg)

Продовження таблиці 1.2

Середовища	Характеристика	Значення
Бездротові (радіо)	*Тип носія інформації*	Електромагнітні хвилі (радіочастотний спектр)
	Максимальна швидкість (2026 р.)	До 46 Гбіт/с теоретично (Wi Fi 7, IEEE 802.11be); до 20 Гбіт/с низхідний канал у 5G-Advanced (3GPP Rel. 18)
	Максимальна дальність без повторювачів	Залежить від частоти та потужності: до 100 м у приміщенні (Wi Fi на 5/6 ГГц); до 10 км (LPWAN, LoRa); стільниковий зв'язок до 15 км (5G NR FR1, сільська місцевість)
	Завадостійкість	Низька в порівнянні з провідними; схильне до завмирань, багатопроменевого поширення, інтерференції та перешкод (навмисних і ненавмисних)
	Складність прийомача/передавача	Висока: радіочастотний підсилювач, синтезатор частоти, модулятор / демодулятор (OFDM, MIMO), антени. Вимагає ретельного узгодження імпедансів
	Живлення по лінії	Ні (за винятком радіочастотного захоплення енергії для надмалопотужних пристроїв, не стандартизовано для передавання даних)
	Гальванічна розв'язка	Властива природі бездротової передачі — автоматична повна гальванічна розв'язка
	Типові представники	Властива природі оптичної передачі — автоматична повна гальванічна розв'язка

Кінець таблиці 1.2

Середовища	Характеристика	Значення
Оптичні (скловолокно)	*Тип носія інформації*	Світлові імпульси (інфрачервоний/видимий діапазон)
	Максимальна швидкість (2026 р.)	400 Гбіт/с на одну довжину хвилі (IEEE 802.3bs); комерційні DWDM-системи передають десятки Тбіт/с по одній жилі (ITU-T G.694.1)
	Максимальна дальність без повторювачів	Багатомодове волокно (OM4/OM5): до 100 м для 100G/400G (IEEE 802.3cm). Одномодове волокно (OS2): 10–40 км для 10G/100G (10GBASE-LR, 100GBASE-LR4) без підсилення; з підсилювачами — сотні км
	Завадостійкість	Абсолютна нечутливість до електромагнітних завад, перехресних наведень та радіочастотних випромінювань; відсутність витоку сигналу через випромінювання
	Складність приймача/передавача	Середня: лазерний діод (VCSEL для MMF, DFB для SMF), фотодіод, волоконно-оптичний трансівер (SFP, QSFP). Потрібне точне суміщення з конектором
	Живлення по лінії	Ні, окрім систем Power-over-Fiber (PoF) для живлення віддалених датчиків (мілівати, не стандартизовано як масовий канал даних)
	Гальванічна розв'язка	Wi Fi 6/6E/7 (IEEE 802.11ax/be), 5G NR (3GPP), Bluetooth LE (5.4), LoRaWAN, Zigbee (IEEE 802.15.4)
	Типові представники	Багатомодове волокно OM4/OM5 (ISO/IEC 11801), одномодове волокно G.652.D, G.657.A2 (ITU-T), з'єднання LC/SC/MPO

Оптичні інтерфейси, у яких носієм виступає світловий потік, посідають особливе місце завдяки поєднанню гальванічної розв'язки, цілковитої нечутливості до електромагнітних завад та потенційно величезної пропускної здатності. У вбудованих системах і IoT оптичні лінії представлені двома основними групами: волоконно-оптичні кабелі, що застосовуються для магістральних з'єднань шлюзів або в промислових мережах із високим

рівнем завад, та безволоконні оптичні з'єднання малої дальності, зокрема IrDA (Infrared Data Association) і видимий світловий зв'язок (VLC, Visible Light Communication), відомий під комерційною назвою Li-Fi. Передавання сигналу скловолокном стандартизоване в документах ITU-T серії G (наприклад, G.652 для одномодового волокна) та IEEE 802.3 (розділи про фізичний рівень 1000BASE-X, 10GBASE-R тощо). В оптичному волокні інформація кодується зміною інтенсивності, фази або поляризації лазерного чи світлодіодного випромінювання, а швидкість передавання в магістральних системах сягає 400 Гбіт/с на одну довжину хвилі. Для мікроконтролерних систем більш характерними є недорогі пластикові оптичні волокна (POF) та трансивери, сумісні за електричним інтерфейсом із UART або SPI, які застосовуються, наприклад, для розв'язки високовольтних кіл у силовій електроніці. Інфрачервоні інтерфейси IrDA, що регулюються специфікаціями IrPHY 1.4, забезпечують швидкості до 4 Мбіт/с на відстані до 1 м, а в режимі малої потужності (IrDA Lite) дозволяють реалізувати безконтактний обмін даними з кишеньковими приладами, хоча поступово витісняються Bluetooth LE. Технологія Li-Fi, стандартизована в IEEE 802.11bb (затверджена 2023 р.), використовує світлодіодні освітлювальні прилади для модуляції світлового потоку на частотах, непомітних для людського ока, чим досягається дуплексна передача даних зі швидкістю понад 1 Гбіт/с у межах освітленої зони. Для розробника мікроконтролерних пристроїв принципово важливим є те, що оптичний канал не накопичує статичної електрики, не створює контурів заземлення і не випромінює радіозавад, що робить його безальтернативним вибором у вибухонебезпечних середовищах (стандарт IEC 60079-28) та в медичному обладнанні, де завади можуть становити загрозу життю пацієнта.

Зіставляючи три середовища, доцільно керуватися енергетичним критерієм та умовами експлуатації. Дротові інтерфейси залишаються на порядок енергоефективнішими за бездротові в перерахунку на біт переданої інформації, що підтверджено порівняльними дослідженнями, опублікованими в журналі IEEE Transactions on Industrial Informatics (том 19, 2023).

Функціональна реалізація описаної взаємодії середовищ на рівні апаратного забезпечення мікроконтролерної техніки наочно демонструється на схемі нижче (рисунок 1.4). На ній детально відображено перетворення сигналів для забезпечення надійного зв'язку між компонентами системи.

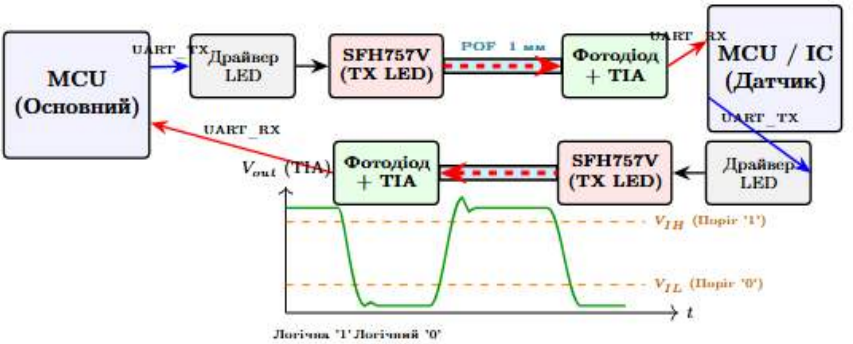


Рисунок 1.4 – Структурна схема оптичного приймально-передавального тракту для мікроконтролера

Бездротові рішення виграють там, де рухомість вузла або відсутність кабельної інфраструктури є визначальними, — наприклад, у системах моніторингу стану залізничних вагонів або в носимих медичних сенсорах. Оптичні канали обирають тоді, коли потрібне поєднання високої швидкості та гарантованої гальванічної розв'язки, що не досягається жодним трансформаторним або ємнісним методом. У реальному IoT-пристрої часто присутні всі три типи середовищ одночасно: датчик збирає інформацію по дротовому I²C, мікроконтролер обробляє її та передає через BLE на смартфон, а шлюз, отримавши пакет, відправляє його в хмару через оптичний Ethernet-канал. Саме тому посібник не протиставляє ці середовища, а розглядає їх як взаємодоповнюючі елементи інженерного інструментарію, вибір між якими визначається конкретним набором вимог.

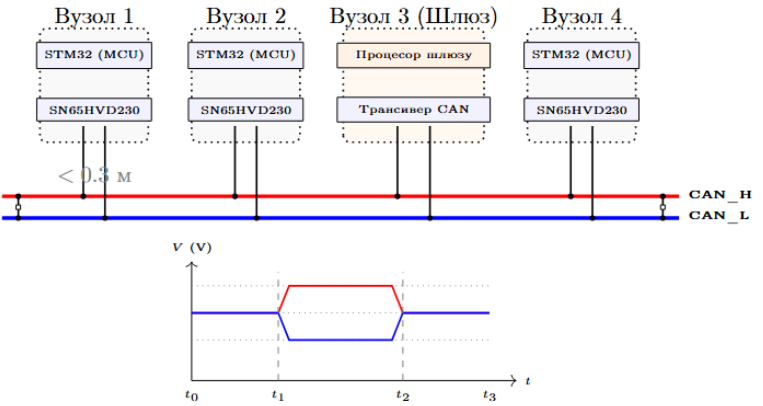
1.2.2. За топологією: точка-точка, шина, зірка, кільце, коміркова мережа

Топологія інтерфейсу визначає просторову конфігурацію з'єднань між вузлами системи, тобто відповідає на запитання, яким чином лінії зв'язку поєднують окремі пристрої в єдину мережу. Цей критерій безпосередньо впливає на надійність, масштабованість, складність кабельного господарства, механізми арбітражу доступу до спільного середовища та затримки передавання. У вбудованих системах і IoT застосовують п'ять основних топологічних схем: точка-точка, шина, зірка, кільце та коміркова мережа, причому в реальних проєктах вони часто комбінуються, утворюючи гібридні архітектури. Вибір топології зумовлюється не лише технічними міркуваннями, а й економічними — вартістю провідників, роз'ємів,

повторювачів або комутаторів, — а також вимогами до живучості системи за відмови окремих вузлів.

Топологія «точка-точка» є найпростішою формою організації зв'язку, за якої існує рівно два учасники обміну, з'єднані виділеним фізичним каналом. Така конфігурація характерна для переважної більшості внутрішньосхемних з'єднань на друкованій платі: UART між мікроконтролером і модулем GPS, SPI між мікроконтролером і зовнішньою флеш-пам'яттю, USB між хостом і периферійним пристроєм. Фізично канал може бути симплексним, напівдуплексним або повнодуплексним, однак ключовою ознакою є відсутність потреби в адресації пакетів на канальному рівні — приймач завжди однозначно визначений. Топологія точка-точка забезпечує максимальну пропускну здатність, оскільки весь ресурс лінії належить одній парі пристроїв, і мінімальну латентність, позбавлену затримок на арбітраж. Саме тому високошвидкісні інтерфейси, як-от PCI Express (PCIe) або MIPI CSI-2, реалізуються саме як набір незалежних ліній точка-точка. Разом із тим, недоліком є квадратичне зростання кількості ліній у разі спроби безпосереднього з'єднання багатьох вузлів, що обмежує застосування цієї топології лише невеликими мережами або змушує вводити комутатори.

Шинна топологія передбачає, що всі пристрої під'єднані до одного спільного фізичного каналу — шини, — по якому сигнал поширюється від передавача до всіх інших вузлів одночасно. Такий підхід мінімізує кількість провідників і роз'ємів, що зробило його панівним у промислових і автомобільних мережах кінця XX століття. Класичними представниками шинної топології у вбудованих системах є I²C, CAN (ISO 11898) та RS-485 у багатоточковому режимі. Через те, що середовище є спільним, виникає потреба в механізмі арбітражу або запобігання колізіям. У шині I²C арбітраж виконується на рівні окремих бітів за принципом монтажного І, у CAN — на основі пріоритету ідентифікатора повідомлення, а в RS-485 (Modbus RTU) колізії уникають за допомогою суворого майстер-веденого протоколу, де ведений вузол відповідає лише після явного запиту. Шинна топологія накладає електричні обмеження на довжину відгалужень (стабів) і вимагає ретельного узгодження хвильового опору на кінцях лінії, що детально описано в стандартах TIA/EIA-485-A (розділ 4.3) та ISO 11898-2. На рисунку 1.5 зображено типову шинну структуру CAN-мережі з двома термінуючими резисторами 120 Ом на протилежних кінцях та короткими відгалуженнями до кожного вузла.



Часовий інтервал	Стан шини CAN	Логічне значення та рівні напруг на лінії
$t_0 \dots t_1$	Рецесивний стан	Логічна '1' ($V_{diff} = 0V$, $CAN\_H = CAN\_L = 2.5V$)
$t_1 \dots t_2$	Домінантний стан	Логічний '0' ($V_{diff} = 2V$, $CAN\_H = 3.5V$, $CAN\_L = 1.5V$)
$t_2 \dots t_3$	Рецесивний стан	Логічна '1' ($V_{diff} = 0V$, $CAN\_H = CAN\_L = 2.5V$)

Рисунок 1.5 – Шинна топологія на прикладі CAN-мережі

Топологія «зірка» концентрує з'єднання навколо центрального вузла, котрий може бути як пасивним концентратором, так і активним комутатором або точкою доступу. У дротових мережах найпоширенішим прикладом є Ethernet із комутатором (IEEE 802.3): кожен кінцевий пристрій з'єднується окремим кабелем із портом світча, а комутатор апаратно маршрутизує кадри лише до порту призначення, забезпечуючи одночасну комунікацію між різними парами вузлів. У бездротових системах топологія зірки реалізується за допомогою точки доступу Wi-Fi (IEEE 802.11) або центрального пристрою Bluetooth piconet, де майстер-вузол керує розкладом передач. Централізація спрощує управління мережею, дозволяє ефективно виявляти несправності та ізолювати некоректну поведінку окремого вузла, проте створює єдину точку відмови: вихід із ладу комутатора або точки доступу паралізує всю мережу. У сучасних IoT-системах зіркова топологія характерна для зв'язку кінцевих пристроїв LoRaWAN зі шлюзом, а також для підключення численних периферійних датчиків до мікроконтролера через окремі лінії SPI з індивідуальними сигналами вибору кристала (CS). Хоча в останньому випадку фізично лінії тактового сигналу SCLK та даних MOSI/MISO можуть бути спільними, логічна структура залишається зірковою, оскільки кожен ведений пристрій комунікує виключно з майстром. Порівняльну характеристику зіркової, шинної та коміркової топологій за критеріями живучості, складності кабельної мережі та детермінованості наведено в таблиці 1.3.

Таблиця 1.3 – Порівняння основних топологій мереж вбудованих систем

Топологія	Характеристика	Значення
Точка-то-чка	*Кількість з'єднань для N вузлів*	1 (тільки два вузли)
	Стійкість до обриву однієї лінії	Повна втрата зв'язку між двома вузлами
	Механізм доступу	Безпосередня виділена передача (без колізій)
	Типова латентність	Дуже низька, детермінована
	Приклад інтерфейсів	UART, SPI, USB (хост-пристрій), I²S
Шина	*Кількість з'єднань для N вузлів*	1 спільне середовище (N відгалужень)
	Стійкість до обриву однієї лінії	Залежить від місця обриву: можлива втрата всієї шини або поділ на сегменти
	Механізм доступу	CSMA/CR (CAN), CSMA/CD (Ethernet), опитування ведучим (Modbus), TDMA
	Типова латентність	Низька для високопріоритетних повідомлень (CAN), середня/висока за колізій (Ethernet)
	Приклади інтерфейсів	CAN, I²C, RS-485 (Modbus RTU), LIN, однопарний Ethernet (10BASE-T1S)
Зірка	*Кількість з'єднань для N вузлів*	N (кожен вузол з'єднаний із центром)
	Стійкість до обриву однієї лінії	Втрачається лише один вузол (обрив його лінії); решта мережі працює
	Механізм доступу	Комутація пакетів (Ethernet-комутатор), опитування або планування центральним вузлом
	Типова латентність	Низька (затримка комутатора), зростає з навантаженням
	Приклади інтерфейсів	Ethernet з комутатором, Wi-Fi інфраструктурний режим, USB з хабом
Кільце	*Кількість з'єднань для N вузлів*	N (кожне з'єднання між сусідами)
	Стійкість до обриву однієї лінії	Мережа розпадається на дві незв'язані частини (без резервування); подвійне кільце зберігає роботу
	Механізм доступу	Передача маркера (Token Ring), часові слоти, спрямована передача по кільцю

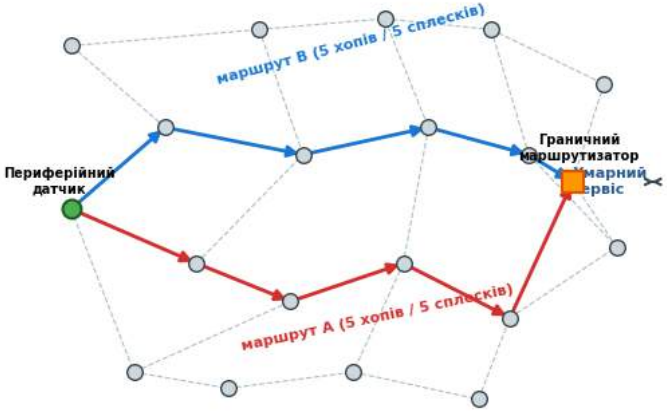
Кінець таблиці 1.3

Топологія	Характеристика	Значення
Кільце	*Типова латентність*	Пропорційна кількості вузлів (очікування маркера)
	Приклади інтерфейсів	Token Ring, Industrial Ethernet (EtherCAT, PROFINET RT у кільцевій конфігурації)
Коміркова мережа	*Кількість з'єднань для N вузлів*	N(N−1)/2 (повнозв'язна), менше за часткового зв'язування
	Стійкість до обриву однієї лінії	Висока стійкість: альтернативні маршрути майже завжди доступні
	Механізм доступу	CSMA/CA з динамічною маршрутизацією (Zigbee), flood mesh (BLE Mesh), децентралізоване керування
	Типова латентність	Змінна, залежить від кількості переходів і щільності мережі
	Приклади інтерфейсів	Zigbee, Thread, BLE Mesh, Wi-Fi Mesh (802.11s), Z-Wave

Кільцева топологія, де кожен вузол з'єднується з двома сусідами, утворюючи замкнений контур, у вбудованих системах трапляється рідше за попередні, однак зберігає важливе значення в окремих нішах. Класичним прикладом є мережі Token Ring (IEEE 802.5), що практично вийшли з ужитку, проте логічне кільце використовується в деяких протоколах промислового Ethernet, зокрема в EtherCAT (IEC 61158 Type 12), де кадр проходить через усі підпорядковані пристрої послідовно. Такий підхід дозволяє досягти надзвичайно низької та детермінованої латентності, оскільки вузол обробляє призначену йому частину кадру «на льоту», без буферизації в комутаторах. Кільцева топологія також забезпечує природну відмовостійкість за наявності двох зустрічних кілець: обрив кабелю перетворює кільце на лінійну шину, не переривaючи зв'язку. У мікроконтролерних системах кільцеві з'єднання іноді реалізують програмно, використовуючи два UART-інтерфейси на кожному вузлі для побудови резервованих каналів телеметрії в безпілотних апаратах. Втім, складність початкового налаштування та реконфігурації кільця після відмови обмежує його застосування тими сферами, де детермінованість є критичною.

Коміркова (mesh) топологія передбачає, що кожен вузол має прямі з'єднання з кількома сусідніми вузлами, а передача даних між довільними точками мережі здійснюється через проміжні ретрансляції (хопи). Така архітектура забезпечує найвищу живучість: вихід із ладу одного чи навіть кі-

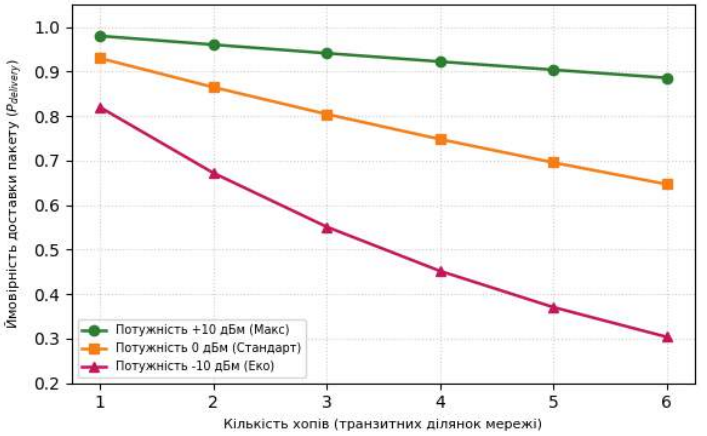
лькох вузлів не розриває мережу, оскільки пакети автоматично маршрутизуються альтернативними шляхами. Коміркові мережі є основою багатьох бездротових протоколів IoT, зокрема Zigbee (IEEE 802.15.4 із мережевим рівнем Zigbee PRO), Thread (IEEE 802.15.4 із IP-маршрутизацією), Bluetooth Mesh (специфікація Bluetooth Mesh Profile 1.0.1) та Z-Wave. Фізичним середовищем у цих випадках виступає радіоефір, тому топологія фактично є повнозв'язною, проте логічно вона регулюється протоколами маршрутизації, що обмежують кількість активних з'єднань для економії енергії. У дротових системах коміркова топологія трапляється в промислових мережах із надлишковістю, наприклад у стандарті HSR (High-availability Seamless Redundancy, IEC 62439-3), а також у внутрішньокристальних мережах на кристалі (NoC), де безліч процесорних ядер з'єднуються між собою матрицею маршрутизаторів. Для мікроконтролерних IoT-пристроїв mesh-мережі цінні тим, що дозволяють охопити велику територію без потужних підсилювачів: кожен недорогий вузол із чутливістю -100 дБм працює лише з найближчими сусідами, а сукупна зона покриття може сягати квадратних кілометрів. Схематичне зображення коміркової мережі з альтернативними маршрутами подано на рисунку 1.6, а.



а)

Рисунок 1.6 – Коміркова топологія бездротової сенсорної мережі:
а) структура коміркової топології мережі

Описана топологічна структура безпосередньо визначає якість функціонування мережі в умовах інтенсивного обміну даними. Залежність імовірності успішної доставки пакетів від довжини маршруту ретрансляції детально відображено на рисунку 1.6, б.



б)

Рисунок 1.6 – Коміркова топологія бездротової сенсорної мережі:
б) надійність доставки залежно від кількості хопів

Підсумовуючи, вибір топології завжди є компромісом між вартістю, простотою, надійністю та пропускною здатністю. Точка-точка залишається неперевершеною для високошвидкісних з'єднань малої кількості компонентів. Шина доцільна там, де мінімізація метражу кабелю важливіша за пікову швидкість, — передусім у автомобільній та промисловій автоматиці. Зірка домінує в офісних і домашніх мережах завдяки зручності адміністрування, а коміркова мережа стала стандартом де-факто для масштабованих бездротових сенсорних систем, де прокладання кабелів фізично неможливе або економічно невигідне. Кільцева топологія й надалі посідає важливе місце в ніші детермінованих промислових мереж із жорсткими часовими обмеженнями, таких як EtherCAT і SERCOS III. Знання сильних сторін та обмежень кожної топології дозволяє інженерові на етапі архітектурного проєктування уникнути дорогих помилок, пов'язаних із недостатньою пропускною здатністю або неприйнятним часом відновлення після збою.

1.2.3. За способом синхронізації: синхронні, асинхронні, ізохронні

Обмін цифровою інформацією між двома пристроями передбачає, що приймач має точно визначити моменти часу, в які слід фіксувати значення сигналу на лінії, тобто моменти дискретизації бітів. Спосіб, у який досягається це узгодження в часі між передавачем і приймачем, становить критерій синхронізації. За цією ознакою всі інтерфейси поділяються на три

великі класи: синхронні, асинхронні та ізохронні. Кожен із них має принципово різний механізм відновлення тактової сітки, що, у свою чергу, впливає на апаратну складність, максимальну швидкість, стійкість до розсинхронізації та сферу застосування.

Синхронні інтерфейси вирізняються наявністю окремого фізичного каналу для тактового сигналу, який виробляється одним із пристроїв (зазвичай майстром) і безперервно або впродовж передачі надходить до всіх інших. Приймач використовує фронти цього тактового сигналу для фіксації даних, що гарантує жорстку прив'язку моменту зчитування до джерела. Завдяки цьому синхронні інтерфейси можуть працювати на дуже високих частотах — десятки мегагерц і більше, — оскільки відпадає потреба в періодичній підстройці внутрішніх генераторів приймача. Типовими представниками синхронних інтерфейсів у мікроконтролерній практиці є SPI, I²C, I²S та SAI. Наприклад, у шині SPI сигнал SCLK генерується майстром і визначає моменти, в які ведений пристрій має виставляти або фіксувати дані на лініях MOSI та MISO. У шині I²C роль тактового сигналу виконує лінія SCL, причому ведений пристрій може тимчасово утримувати її в низькому стані (розтягування такту), щоб уповільнити обмін. Недоліком суто синхронного підходу є необхідність у додатковому проводі під тактовий сигнал та обмеження на довжину лінії, спричинене перекосами фаз між тактом і даними (clock skew), що вимагає ретельного вирівнювання довжин доріжок на платі. Під час проєктування синхронних інтерфейсів визначальним обмеженням виступає перекіс фаз між тактовим сигналом і лініями даних, спричинений різницею в затримках поширення. Виробники мікроконтролерів, зокрема STMicroelectronics у посібнику з проєктування апаратного забезпечення для сімейства STM32H7 (документ AN4938), рекомендують, щоб сумарна затримка поширення сигналу в лініях такту й даних не перевищувала половини періоду тактової частоти. Цю вимогу формалізовано у вигляді формули (1.1):

$$t_{prop} + t_{skew} < \frac{1}{2 \times f_{SCLK}} \ , \qquad (1.1)$$

де t_{prop} — затримка поширення сигналу даних від передавача до приймача; t_{skew} — максимальна різниця затримок між лініями тактового сигналу та даних (перекіс); f_{SCLK} — частота тактового сигналу. Із формули (1.1) безпосередньо випливає, що зі зростанням частоти допустимі геометричні довжини провідників і паразитні ємності швидко зменшуються, що на практиці обмежує частоту SPI у межах типової друкованої плати значеннями 40–80 МГц для стандартних виводів мікроконтролерів.

Крім того, до синхронних часто відносять інтерфейси з вбудованим тактовим сигналом, у яких тактова синхронізація відновлюється з самого потоку даних за допомогою спеціального кодування (манчестерське, 8b/10b), як це реалізовано в USB, Ethernet та CAN. У цих випадках окремий провід такту відсутній, однак логічно система залишається синхронною, оскільки приймач підлаштовує свій тактовий генератор під фазу вхідного сигналу за допомогою схем фазового автопідстроювання (PLL).

Асинхронні інтерфейси не мають ані виділеного тактового проводу, ані вбудованого тактового сигналу; натомість передавач і приймач заздалегідь домовляються про номінальну швидкість передавання (біт/с) і використовують власні незалежні генератори тактових імпульсів. Для того щоб приймач міг визначити початок символу, кожен кадр починається зі стартового біта, а завершується одним або кількома стоповими бітами. Найпоширенішим асинхронним інтерфейсом є UART, формалізований у сімействі стандартів EIA/TIA-232-F (для RS-232) та аналогічних. Після появи стартового біта приймач очікує половину тривалості одного бітового інтервалу, щоб опинитися в центрі першого інформаційного біта, а далі зчитує наступні біти через повні бітові інтервали. Очевидно, що такий метод критично чутливий до неузгодженості частот двох генераторів: якщо за час передачі повного кадру (наприклад, 10 бітів) накопичена фазова помилка перевищить половину бітового інтервалу, станеться помилкове зчитування. Допустиме відхилення частоти зазвичай обмежують на рівні 2–3% для кадрів із 8 бітами даних, що вимагає застосування кварцових або прецизійних RC-генераторів. Перевагою асинхронних інтерфейсів є мінімальна кількість провідників (всього два — TX і RX — плюс спільна земля) та можливість роботи на значних відстанях за умови перетворення рівнів у диференційний сигнал, як у RS-485. У навчальній лабораторній практиці саме UART часто слугує першим інтерфейсом, на якому студент спостерігає явище втрати синхронізації при невідповідності бітрейтів, що наочно демонструє важливість тактової дисципліни.

Ізохронні інтерфейси становлять окремий клас, де визначальною характеристикою є не лише синхронізація бітів, а й гарантована постійність часу доставки даних від джерела до споживача. Термін «ізохронний» (від грецького ἴσος — рівний і χρόνος — час) підкреслює, що інформація має надходити рівномірно, без непередбачуваних затримок і джитеру, що є критичним для потокового аудіо, відео, керування рухом у реальному часі та розподілених вимірювальних систем. Ізохронна передача може спиратися на синхронний або асинхронний фізичний рівень, однак обов’язково міс-

тить механізм резервування пропускної здатності або часових слотів. Наприклад, в універсальній послідовній шині USB специфікація USB 2.0 (розділ 5.6) визначає ізохронний тип передачі, за яким хост гарантує виділення необхідної смуги пропускання на кожному мікрокадрі (125 мкс) і не застосовує повторних передач у разі помилок, жертвуючи надійністю заради своєчасності. Інтерфейс I²S, що широко використовується для цифрового аудіо, є синхронним за фізичною реалізацією (містить лінії такту SCLK і вибірки слова WS), проте за логікою роботи він є ізохронним: дані мають надходити з постійною частотою дискретизації, і будь-яка затримка сприймається як артефакт відтворення. Сучасні промислові мережі на базі Ethernet із підтримкою Time-Sensitive Networking (TSN), що регулюються серією стандартів IEEE 802.1Q (зокрема IEEE 802.1Qbv для планування трафіку та IEEE 802.1AS для синхронізації часу), виводять ізохронність на новий рівень, гарантуючи наскрізну затримку в межах одиниць мікросекунд навіть у мережах зі значним фоновим навантаженням. Це уможливлює заміну традиційних польових шин на єдину Ethernet-магістраль у концепції Industry 4.0.

Для наочного зіставлення синхронних, асинхронних та ізохронних інтерфейсів за ключовими показниками складено таблицю 1.4, де кожен клас охарактеризовано за наявністю окремого тактового сигналу, типовою швидкістю, стійкістю до відхилень частоти, прикладами реалізацій у мікроконтролерних системах та нормативними документами, що визначають відповідні специфікації.

Таблиця 1.4 – Порівняння синхронних, асинхронних та ізохронних інтерфейсів

Клас інтерфейсу	Характеристика	Значення
Синхронний (з окремим тактом)	*Наявність окремого тактового сигналу*	Так, окрема лінія SCLK
	Типова швидкість передавання	До 100 Мбіт/с (SPI)
	Допустиме відхилення частоти генераторів	Визначається вимогами setup/hold (одиниці нс), не накопичується
	Характерна латентність	Низька, детермінована
	Приклади інтерфейсів у вбудованих системах	SPI, I²S, MICROWIRE
	Нормативна база	SPI: де-факто; I²S: NXP (Philips) I²S Bus Specification

Кінець таблиці 1.4

Клас інтерфейсу	Характеристика	Значення
Синхронний (з відновленням такту)	*Наявність окремого тактового сигналу*	Немає окремої лінії; такт вбудовано у дані (кодування NRZI, 8b/10b, 64b/66b)
	Типова швидкість передавання	До десятків Гбіт/с (USB 3.2, PCIe 4.0)
	Допустиме відхилення частоти генераторів	Жорстке, зазвичай ±500 ppm (USB 2.0), ±300 ppm (PCIe); вимагає PLL
	Характерна латентність	Низька, детермінована, але залежить від пакетизації та арбітражу
	Приклади інтерфейсів у вбудованих системах	USB, PCIe, SATA, DisplayPort, Gigabit Ethernet (1000BASE-X)
	Нормативна база	USB 2.0/3.2 Spec, PCI Express Base Spec, IEEE 802.3
Асинхронний	*Наявність окремого тактового сигналу*	Відсутній; синхронізація за старт/стоповими бітами
	Типова швидкість передавання	До ~10 Мбіт/с (RS-485)
	Допустиме відхилення частоти генераторів	±2–3 % на кадр (залежить від довжини слова та стоп-бітів)
	Характерна латентність	Непостійна, змінна від символу до символу
	Приклади інтерфейсів у вбудованих системах	UART, RS-232, RS-485, LIN
	Нормативна база	TIA/EIA-232-F, TIA/EIA-485-A, LIN 2.2a
Ізохронний	*Наявність окремого тактового сигналу*	Може бути присутнім (I²S) або забезпечуватись глобальною синхронізацією (PTP в Ethernet TSN)
	Типова швидкість передавання	До десятків Гбіт/с (Ethernet TSN); фіксована для медіа (I²S)
	Допустиме відхилення частоти генераторів	Жорсткі вимоги до синхронізації (PTP точність < 1 мкс); низький джиттер для чистих тактових
	Характерна латентність	Гарантовано постійна, малий джиттер (QoS)
	Приклади інтерфейсів у вбудованих системах	I²S, USB isochronous transfers, EtherCAT, PROFINET IRT, Ethernet TSN (802.1Qbv)
	Нормативна база	USB 2.0 Spec, IEEE 802.1Qbv, IEC 61158 (EtherCAT)

Механізм синхронізації безпосередньо ілюструється часовими діаграмами. На рисунку 1.7 наведено спрощену осцилограму передачі одного байта даних (0xA5, двійковий вигляд 10100101) через синхронний інтерфейс SPI у режимі 0 (CPOL = 0, CPHA = 0) та через асинхронний UART із конфігурацією 8N1 (8 бітів даних, без паритету, 1 стоп-біт). Для SPI показано лінії SCLK і MOSI, причому стрілками позначено моменти фіксації даних приймачем по передньому фронту тактового сигналу. Для UART показано лінію TX, де низький рівень стартового біта ініціює відлік, а наступні вісім інформаційних бітів і високий рівень стопового біта формують кадр тривалістю 10 бітових інтервалів. Цей рисунок дає змогу студентові наочно порівняти, що в синхронному випадку часове положення кожного біта жорстко задане тактовим сигналом майстра, тоді як в асинхронному — відновлюється локальним генератором приймача лише за стартовим фронтом, і будь-яка похибка накопичується впродовж кадру.

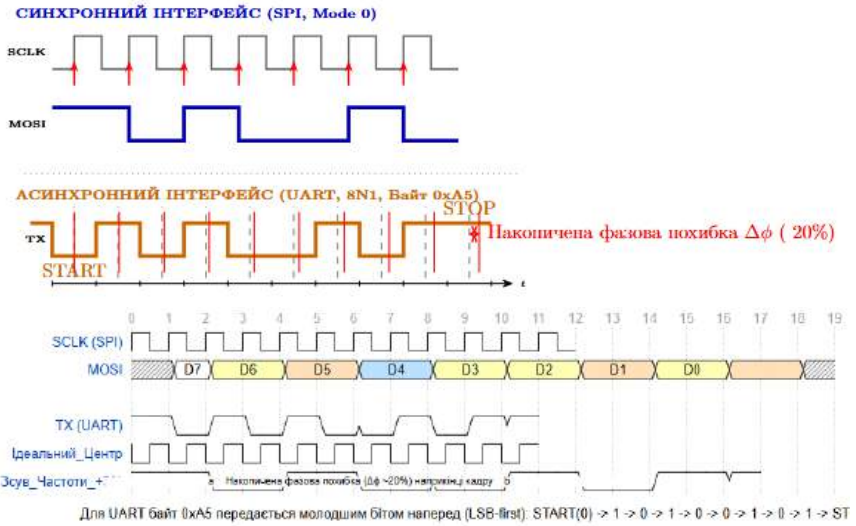


Рисунок 1.7 – Порівняння синхронної (SPI) та асинхронної (UART) передачі на часових діаграмах

Для асинхронної передачі корисним є кількісне розуміння допустимого відхилення частоти. Позначимо номінальну тривалість одного бітового інтервалу як T_{bit}, реальну частоту генератора приймача як $f_{прийм} = f_{ном} * (1 + \varepsilon)$, де ε — відносне відхилення. Приймач фіксує стартовий біт і через половину бітового інтервалу ($T_{bit}/2$) опиняється в центрі першого інформаційного біта. Для N-го біта (N від 1 до 8) момент фіксації

за реальним годинником приймача становитиме $(0{,}5 + N) * T_{bit} / (1 + \varepsilon)$. Відхилення від ідеального моменту зростатиме пропорційно N. Найгірший випадок має місце для останнього, восьмого біта даних перед стоповим бітом. Для гарантовано правильного зчитування абсолютна похибка не повинна перевищувати половини бітового інтервалу $T_{bit}/2$, що дає відому інженерну оцінку максимального допустимого відхилення $\varepsilon_{max} \approx 1 / (2 * (0{,}5 + N_{max}))$. Для типових кадрів із 8 бітами даних ($N_{max} = 8$) це становить близько 5% за умови ідеальної середини стартового біта, проте на практиці з урахуванням шумів, спотворень фронтів і дискретності налаштування бітрейту виробники мікроконтролерів, зокрема Espressif у технічному описі ESP32-S3 (Technical Reference Manual, розділ UART), рекомендують забезпечувати відхилення не більше ±2%, що досягається застосуванням кварцового резонатора або температурно-компенсованого генератора.

Ізохронні системи потребують додаткових механізмів глобальної синхронізації часу. Так, профіль IEEE 802.1AS, що є складовою TSN, запроваджує розподіл прецизійного часу (Precision Time Protocol, PTP) для узгодження годинників усіх вузлів мережі з точністю до сотень наносекунд. На підставі цього глобального часу комутатори TSN виконують планування черг за алгоритмом, визначеним у стандарті IEEE 802.1Qbv (Time-Aware Shaper), виділяючи циклічні часові вікна для ізохронного трафіку. Для студентів, що працюють із мікроконтролерами STM32, практичне знайомство з ізохронністю часто починається з налаштування інтерфейсу I²S у режимі DMA із подвійною буферизацією, де дані з буфера передаються через рівні проміжки часу, задані сигналом WS, і будь-яка затримка в обробці переривання призводить до спотворення звуку. Отже, розуміння відмінностей між синхронною, асинхронною та ізохронною синхронізацією не лише збагачує теоретичний кругозір, а й має безпосередні практичні наслідки під час написання мікропрограмного коду та трасування сигналів на платі.

1.2.4. За напрямком передачі: симплексні, напівдуплексні, повнодуплексні

Напрямок передачі визначає можливість одночасного або почергового пересилання даних між учасниками обміну. Цей критерій безпосередньо пов’язаний із кількістю фізичних провідників, організацією буферів приймально-передавальних блоків та складністю протокольних автоматів. В інженерній практиці розрізняють три принципові режими: симплексний,

напівдуплексний та повнодуплексний, кожен із яких має чітко окреслену сферу застосування.

Симплексний (односпрямований) режим передбачає передавання інформації лише в одному напрямку — від джерела до споживача, без будь-якої можливості зворотного зв'язку тим самим каналом. Фізично симплексний канал може складатися з одного провідника плюс спільна земля (наприклад, лінія TX мікроконтролера, під'єднана до входу RX дисплея, який ніколи не відповідає) або з використанням складніших середовищ — радіомовлення, інфрачервоний передавач на пульті дистанційного керування, оптоволоконна лінія з одним передавачем та одним приймачем. У мікроконтролерних системах симплексна конфігурація UART, де задіяно лише вивід TX, є найпростішим способом виведення налагоджувальної інформації на послідовний термінал. Симплексний режим забезпечує максимальну простоту реалізації, проте обмежує архітектурну гнучкість: відправник не отримує жодних підтверджень, тому протоколи вищих рівнів мусять передбачати надлишкове кодування або циклічне повторення для підвищення ймовірності доставки. Нормативно симплексні лінії зв'язку часто описують у контексті розподілених систем збору даних, що закріплено, зокрема, в IEC 60870-5-101 для телемеханіки, де передбачено режим небалансованої передачі з одностороннім потоком кадрів.

Напівдуплексний режим дозволяє обом пристроям і передавати, і приймати дані, однак не одночасно, а почергово. У певний момент часу середовище передавання зайняте лише в одному напрямку, а зміна напрямку вимагає перемикання трансиверів із режиму передачі в режим прийому. Напівдуплекс є домінуючою схемою для багатоточкових шинних топологій, де кілька вузлів поділяють спільне середовище. Класичним прикладом є двопровідна лінія RS-485, регламентована стандартом TIA/EIA-485-A: диференційний сигнал поширюється по одній витій парі, а кожен вузол активує свій передавач лише тоді, коли отримує право на передачу від майстер-пристрою. У промислових мережах Modbus RTU поверх RS-485 застосовано сувору дисципліну «запит — відповідь», де майстер надсилає кадр запиту, після чого переводить свій трансивер у режим прийому і очікує відповіді від веденого пристрою. Затримка на перемикання трансивера (turnaround time) становить від одиниць до десятків мікросекунд залежно від типу мікросхеми, що має враховуватися під час розрахунку таймаутів у реальному часі. У шині I²C, хоча вона має два окремі проводи SDA та SCL, обмін також є напівдуплексним за логічною суттю: в кожен момент часу лише один пристрій (майстер або ведений після запиту) встановлює рівень на лінії даних, а інші або «слухають», або утримують лінію у високому

стані через підтягувальні резистори. Напівдуплексний режим економить провідники та спектр, але накладає обмеження на максимальну пропускну здатність, оскільки канал мусить ділитися між двома напрямками в часі. Дослідження, опубліковані в IEEE Transactions on Communications (том 70, 2022), показують, що в напівдуплексних системах із фіксованим часом перемикання ефективна пропускна здатність падає обернено пропорційно до суми часу передачі кадру та захисного інтервалу.

Повнодуплексний режим надає можливість одночасної передачі в обох напрямках, що досягається або просторовим рознесенням ліній, або частотним поділом, або застосуванням ехокомпенсації. У мікроконтролерних системах найпоширенішою реалізацією повного дуплексу є використання двох незалежних провідників — TX і RX у UART або MOSI і MISO у SPI. У цих випадках передавач і приймач кожного з пристроїв фізично ізольовані, що уможливлює безперервний двосторонній потік даних без жодних пауз на перемикання. Повнодуплексний UART, реалізований у більшості мікроконтролерів STM32 (документ RM0468 для STM32H723/733, розділ 51), містить окремі зсувні регістри для передачі та прийому, а також незалежні буфери, що дозволяє одночасно відправляти й отримувати байти на повній швидкості без апаратних колізій. У високошвидкісних інтерфейсах, таких як Ethernet, повний дуплекс реалізовано ще радикальніше: у середовищі витої пари 1000BASE-T за стандартом IEEE 802.3ab передавання і прийом відбуваються одночасно по всіх чотирьох парах із застосуванням гібридних схем та ехокомпенсаторів, що прибирають власний переданий сигнал із прийнятого. Такий підхід подвоює ефективну пропускну здатність порівняно з напівдуплексним режимом, однак вимагає значно складнішої апаратури фізичного рівня.

Для унаочнення відмінностей між трьома режимами на рисунку 1.8 наведено спрощені схеми з’єднань із позначенням напрямків потоків даних у кожному випадку.

Кількісне порівняння режимів за такими показниками, як кількість необхідних провідників для двовузлового з’єднання, максимальна сумарна пропускна здатність за наявності ідеального каналу, складність апаратної реалізації та типові представники, наведено в таблиці 1.5.

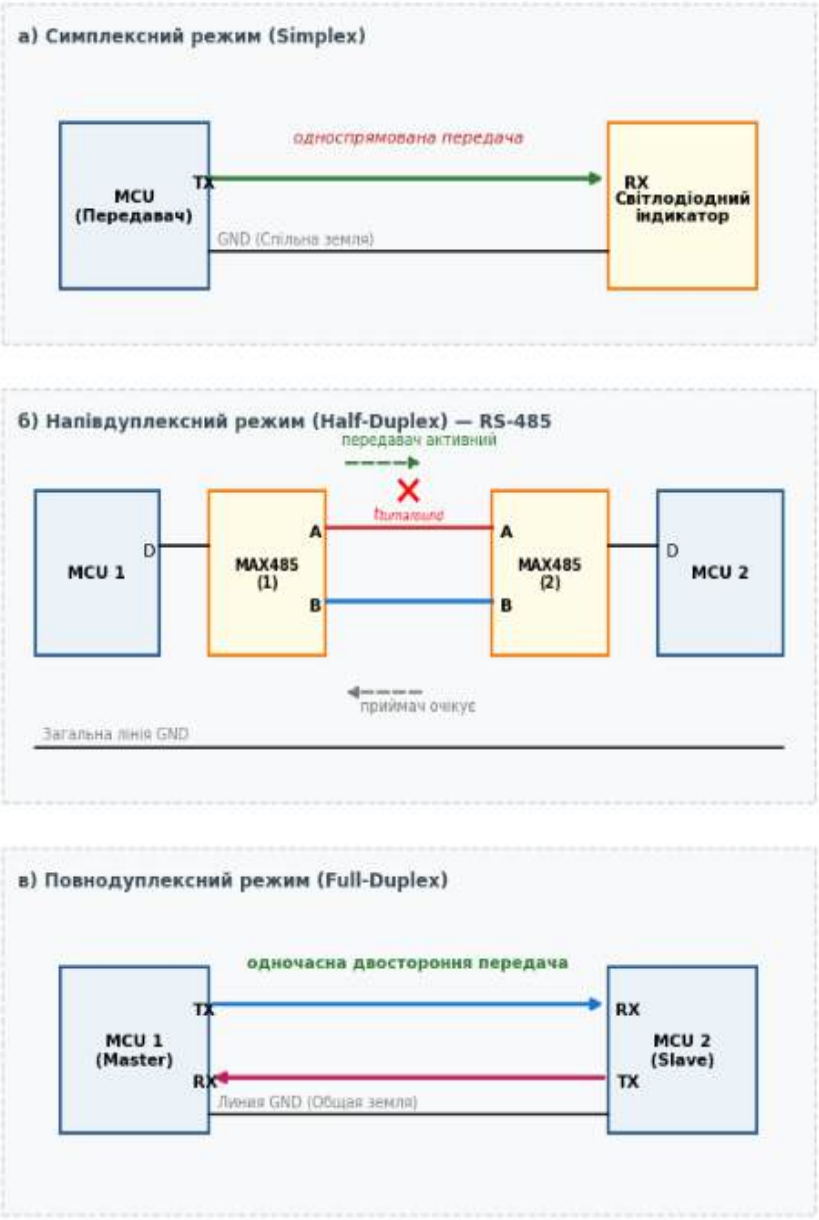


Рисунок 1.8 – Симплексний, напівдуплексний та повнодуплексний режими передавання

Таблиця 1.5 – Порівняння синхронних, асинхронних та ізохронних інтерфейсів

Режим	Характеристика	Значення
Симплексний	*Кількість сигнальни провідників (без GND) для двох вузлів*	1 (наприклад, TX → RX)
	Сумарна пропускна здатність	Дорівнює швидкості в одному напрямку
	Потреба в керуванні трансивером	Не потрібна (передавач постійно активний на одному боці)
	Приклади інтерфейсів	Односторонній UART, радіомовлення, SPI (тільки MOSI)
	Нормативний документ або специфікація	TIA/EIA-232-F (для RS-232)
Напівдуплексний	*Кількість сигнальних провідників (без GND) для двох вузлів*	2 (RS-485), 1 плюс 2 лінії керування за використання DE/RE
	Сумарна пропускна здатність	Пропускна здатність ділиться між напрямками
	Потреба в керуванні трансивером	Обов'язкове
	Приклади інтерфейсів	RS-485 / I²C / CAN
	Нормативний документ або специфікація	TIA/EIA-485-A / ISO 11898-1
Повнодуплексний (роздільні лінії)	*Кількість сигнальних провідників (без GND) для двох вузлів*	2 (UART TX+RX), 4 (SPI MOSI+MISO+SCLK+SS)
	Сумарна пропускна здатність	Подвоєна (одночасна двонапрямлена передача)
	Потреба в керуванні трансивером	Не потрібна
	Приклади інтерфейсів	UART, SPI, RS-422
	Нормативний документ або специфікація	TIA/EIA-422-B, SPI (де-факто)
Повнодуплексний (ехокомпенсація)	*Кількість сигнальних провідників (без GND) для двох вузлів*	1 пара (2 проводи) для 1000BASE-T, 4 пари (8 проводів)
	Сумарна пропускна здатність	Подвоєна без збільшення кількості проводів
	Потреба в керуванні трансивером	Вимагає гібридних схем та цифрової обробки сигналів
	Приклади інтерфейсів	1000BASE-T Ethernet, VDSL2
	Нормативний документ або специфікація	IEEE 802.3ab (1000BASE-T)

Важливо наголосити, що належність інтерфейсу до того чи іншого класу не завжди є абсолютною: деякі протоколи допускають зміну режиму

залежно від конфігурації. Так, інтерфейс SPI за замовчуванням є повнодуплексним, оскільки лінії MOSI та MISO незалежні, однак у чотирипровідному варіанті з однією лінією даних (3-wire SPI, або напівдуплексний SPI) використовується єдиний двонапрямлений вивід, що вимагає програмного або апаратного перемикання напрямку. Подібним чином, специфікація USB 3.2 передбачає одночасне використання кількох диференційних пар, частина з яких працює в симплексному режимі (окремі пари для передачі та прийому), що в сукупності дає повний дуплекс на системному рівні. Під час розроблення вбудованого програмного забезпечення розробник має враховувати не лише фізичний дуплексний режим, а й логічний: навіть маючи повнодуплексний UART, прикладний протокол може бути реалізований у стилі «запит — відповідь», фактично використовуючи канал як напівдуплексний. Така надлишковість іноді є свідомим спрощенням, що полегшує налагодження та підвищує передбачуваність поведінки системи. Отже, вибір режиму за напрямком передачі є не лише схемотехнічним рішенням, а й архітектурним, що впливає на продуктивність, енергоспоживання та стійкість протоколу до помилок синхронізації.

1.2.5. За типом даних: послідовні, паралельні; потокові, пакетні

Критерій типу даних об’єднує два тісно пов’язані, проте концептуально відмінні способи класифікації інтерфейсів: перший стосується фізичної організації бітів під час їхнього транспортування через середовище (послідовна або паралельна передача), другий — логічної структури інформаційного обміну з погляду прикладного рівня (потокова або пакетна передача). Обидва виміри безпосередньо впливають на швидкодію, складність розведення плати, стійкість до завад і вибір програмних абстракцій для роботи з периферійними модулями мікроконтролера. У сучасній інженерній практиці спостерігається виразна тенденція до витіснення паралельних інтерфейсів послідовними на фізичному рівні при одночасному домінуванні пакетної організації на логічному рівні, однак обидва підходи зберігають свої ніші, зумовлені конкретними вимогами до затримок, пропускної здатності та детермінованості.

Послідовна передача даних передбачає, що біти одного інформаційного слова транспортуються через єдиний сигнальний канал один за одним у часі. Саме цей принцип покладено в основу абсолютної більшості сучасних інтерфейсів — UART, SPI, I²C, USB, CAN, Ethernet, PCI Express, SATA, — оскільки він дозволяє мінімізувати кількість провідників, уникнути про-

блеми перекосу фаз між паралельними лініями (clock skew) на високих частотах і спростити конструкцію конекторів. У послідовному каналі приймач мусить відновити не лише значення бітів, а й межі бітових інтервалів, для чого застосовують або окремий тактовий сигнал (синхронні послідовні інтерфейси, як-от SPI), або стартові біти (асинхронні, як-от UART), або вбудоване тактування через надлишкове кодування (8b/10b у PCIe, USB 3.2, Ethernet). Пропускна здатність послідовного каналу визначається добутком символьної швидкості (бод) на кількість бітів, що переносяться одним символом, як це формалізовано в теоремі Найквіста для каналів з обмеженою смугою. Сучасні послідовні інтерфейси, такі як PCIe 6.0 із модуляцією PAM4, досягають швидкостей 64 ГТ/с на одну лінію, що робить паралельну передачу на подібних частотах практично неможливою через неконтрольований clock skew. У мікроконтролерних системах послідовна передача є безальтернативною для зовнішніх з'єднань, тоді як на кристалі або між тісно розташованими мікросхемами пам'яті ще подекуди застосовують паралельний підхід.

Паралельна передача даних використовує кілька сигнальних ліній одночасно для транспортування групи бітів (зазвичай 8, 16, 32 або 64) за один тактовий цикл. Класичними прикладами у вбудованих системах є 8-бітний режим підключення РК-дисплеїв на контролері HD44780, паралельний інтерфейс зовнішньої SRAM або NOR Flash на шині FSMC (Flexible Static Memory Controller) мікроконтролерів STM32, а також паралельний порт Centronics, що домінував у принтерах до появи USB. Перевагою паралельної передачі є потенційно вища пропускна здатність на низьких тактових частотах: наприклад, 8-бітна шина даних на частоті 20 МГц забезпечує ті самі 160 Мбіт/с, що й послідовний канал на частоті 160 МГц. Однак із підвищенням частоти неминучі відмінності в довжинах провідників, ємностях навантаження та затримках у буферах виводів призводять до того, що фронти тактового сигналу і сигналів даних зсуваються одне відносно одного (clock skew). Коли сумарний skew перевищує половину тактового періоду, надійне зчитування стає неможливим. Саме з цієї причини паралельні шини ATA/33 було замінено на послідовні SATA, а паралельний PCI — на послідовний PCI Express, що зафіксовано у відповідних специфікаціях PCI-SIG. У царині мікроконтролерів паралельні інтерфейси доцільні лише за малих відстаней (одиниці сантиметрів на друкованій платі) і помірних частот (до 100 МГц за умови ретельного вирівнювання довжин доріжок). Виробники, зокрема STMicroelectronics у рекомендаціях AN5073 «High-speed layout guidelines for STM32H7 series», наголошують на необхідності узгодження довжин паралельних ліній із точністю до 0,5 мм для

частот понад 50 МГц. Схематичне порівняння послідовного та паралельного способів передавання ілюструється на рисунку 1.9, де показано, як одне й те саме 8-бітне слово (0xA5) транспортується через один послідовний канал за вісім тактових циклів або через вісім паралельних ліній за один цикл.

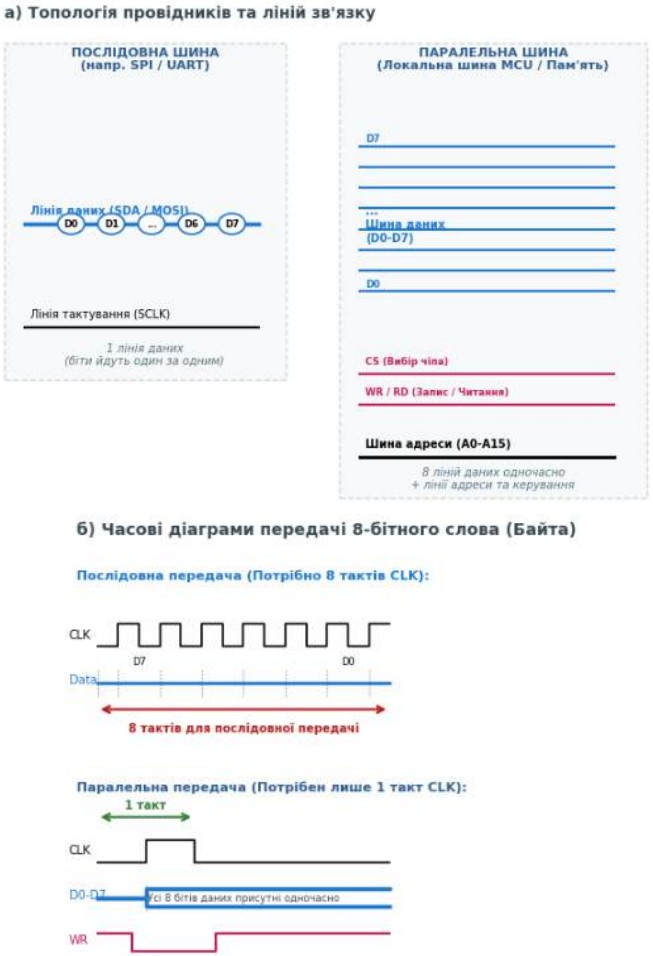


Рисунок 1.9 – Послідовна та паралельна передача 8-бітного слова даних

Потокова передача даних (streaming) характеризується безперервним, неструктурованим із погляду прикладного рівня потоком байтів або символів, де відправник і отримувач заздалегідь узгоджують лише фізичні параметри каналу, але не формат окремих повідомлень. Найпростішим

прикладом є класичний асинхронний UART: після конфігурації бітрейту, кількості стоп-бітів і паритету передавач надсилає послідовність байтів, а приймач відновлює їх один за одним, не маючи жодної інформації про те, де починається і де закінчується осмислене повідомлення. Виділення меж повідомлень у потоці є виключно обов'язком прикладного рівня, що часто спричиняє типову помилку початківців: після скидання мікроконтролера приймач починає зчитувати байти з середини повідомлення, і протокол не відновлюється до наступного стартового байта. Потоковий режим також притаманний послідовним аудіоінтерфейсам I²S, де вибірка за вибіркою безупинно транспортуються від АЦП до процесора без адресації та кадрування. Перевагою потокової передачі є мінімальна службова надлишковість: кожен біт каналу несе корисну інформацію (за винятком старт-стопових бітів в UART), що максимізує ефективну пропускну здатність. Недоліком є вразливість до десинхронізації: якщо приймач пропускає або хибно додає біт, весь подальший потік стає некоректним до ручного відновлення.

Пакетна передача даних організовує інформацію в дискретні блоки (пакети, кадри, фрейми), кожен із яких містить не лише корисні дані, а й службові поля — заголовок із адресою, довжиною та типом пакета, контрольну суму або CRC для виявлення помилок, а також ознаки початку й кінця пакета. Така організація дозволяє однозначно визначати межі повідомлень навіть за наявності пауз у передачі, мультиплексувати трафік від багатьох джерел в одному каналі, адресувати дані конкретному отримувачу в багатоточкових мережах і автоматично виявляти спотворення під час транспортування. Усі сучасні складні інтерфейси є пакетними: USB оперує пакетами (token, data, handshake), CAN — кадрами даних із 11- або 29-бітним ідентифікатором, Ethernet — фреймами з MAC-адресами призначення та джерела, MQTT — повідомленнями з темою та корисним навантаженням. Навіть у порівняно простих внутрішньосхемних інтерфейсах, таких як I²C, обмін структурований на кадри: після стартової умови йде адресний байт, потім один або кілька байтів даних, підтверджуваних бітами ACK, і, нарешті, стопова умова. Пакетна організація неминуче вносить службову надлишковість (overhead), яка зменшує корисну пропускну здатність: наприклад, у мережі CAN зі стандартним 11-бітним ідентифікатором максимальна ефективність використання каналу не перевищує 60% через наявність полів арбітражу, CRC, підтвердження та міжкадрових проміжків. Однак ця плата виправдана радикальним підвищенням надійності та гнучкості.

Порівняльну характеристику потокового та пакетного типів передачі наведено в таблиці 1.6, де зіставлено структуру даних, механізми син-

хронізації повідомлень, способи виявлення помилок, ефективність використання каналу та найтиповіші сфери застосування в мікроконтролерних системах.

Таблиця 1.6 – Порівняння потокової та пакетної організації передавання даних

Характеристика	Потокова передача	Пакетна передача
Структура даних	Неперервний потік байтів/символів	Дискретні блоки із заголовком, даними та контрольною сумою
Виділення меж повідомлення	Відповідальність прикладного рівня (спеціальні старт-стоп символи, таймаути)	Явні маркери початку/кінця або поле довжини
Адресація отримувача	Відсутня	Присутня в заголовку (адреса пристрою, ідентифікатор повідомлення)
Виявлення помилок	Лише на рівні окремого байта (парність, якщо налаштовано)	CRC, контрольна сума в кожному пакеті
Службова надлишковість (overhead)	Мінімальна (старт/стоп біти)	Помітна (заголовки, CRC, міжпакетні інтервали)
Стійкість до десинхронізації	Низька — пропуск одного байта збиває весь потік	Висока — помилка призводить до відкидання лише одного пакета
Типові представники	UART, I²S	I²C, CAN, USB, Ethernet, MQTT
Нормативна база або специфікації	TIA/EIA-232-F, NXP I²S Specification	NXP UM10204, ISO 11898-1, IEEE 802.3, MQTT v5.0 Specification

Важливо розуміти, що поділ на потоковий і пакетний типи не збігається з поділом на послідовний і паралельний: послідовний інтерфейс може бути як потоковим (UART), так і пакетним (CAN, USB), аналогічно, паралельний інтерфейс може працювати в потоці (Centronics у режимі ECP із неперервним потоком даних) або в пакетному режимі (паралельна шина із сигналом адресації та циклом читання-запису, як-от FSMC). Деякі інтерфейси поєднують обидва типи: специфікація USB 2.0 (розділ 5) визначає чотири типи передач — контрольну (control), ізохронну (isochronous), переривання (interrupt) і масову (bulk), — з яких ізохронна є потоковою за

своєю суттю (дані передаються без гарантії доставки, але з гарантованою смугою), тоді як масова і контрольна є пакетними. Схожа гібридність властива і протоколу Bluetooth LE, де рекламні пакети (advertising) та пакети даних у з'єднанні мають чітко визначену структуру з полями довжини та CRC.

Для розробника програмного забезпечення різниця між потоковою і пакетною організацією є наскрізною. У разі потокового UART типовою практикою є застосування кільцевого буфера (ring buffer) та обробника переривань, який лише накопичує байти, а вся логіка виділення повідомлень залишається за прикладним рівнем. Натомість для пакетних інтерфейсів, таких як CAN, апаратний контролер сам фільтрує пакети за ідентифікатором, відкидає некоректні за CRC і сповіщає процесор лише про успішно прийняті повідомлення, що значно розвантажує центральний процесор. У навчальному процесі перехід від потокового мислення (UART із виведенням «Hello, world») до пакетного (I²C із читанням регістрів датчика за адресою, CAN із прийманням повідомлень за фільтром) є одним із ключових етапів становлення інженера-програміста вбудованих систем, оскільки саме тут формується розуміння відмінності між транспортуванням окремих байтів та обміном структурованими даними в розподіленій системі.

1.3. Ієрархія інтерфейсів у вбудованих системах: від внутрішньокристальних шин до глобальних мереж

Сучасна мікроконтролерна система, особливо в контексті Інтернету речей, ніколи не обмежується одним-єдиним інтерфейсом. Навіть у відносно простому пристрої — наприклад, розумному лічильнику води — співіснують щонайменше чотири-п'ять різнорідних інтерфейсів: I²C для зчитування даних із ультразвукового сенсора, SPI для доступу до зовнішньої флеш-пам'яті, UART для обміну AT-командами з NB-IoT модемом, а також внутрішня шина AHB, що зв'язує процесорне ядро з усіма цими периферійними блоками. Ці інтерфейси не є ізольованими; вони утворюють чітку ієрархію, яка відображає фізичну віддаленість компонентів, вимоги до пропускної здатності, енергоспоживання та рівень абстракції, на якому працює розробник програмного забезпечення. Розуміння цієї ієрархії дозволяє інженерові свідомо розподіляти ресурси системи, обирати адекватні протоколи для кожного рівня та уникати архітектурних помилок, коли, скажімо, низькошвидкісний датчик намагаються під'єднати через високошвидкісний PCIe лише тому, що «так зручніше».

Ієрархію інтерфейсів у вбудованих системах доцільно розглядати за аналогією з транспортною інфраструктурою: внутрішньокристальні шини є «ліфтами та коридорами» всередині будівлі (кристала), з'єднання на друкованій платі — «вулицями мікрорайону», промислові мережі — «магістралями між містами», а глобальні бездротові мережі — «міжконтинентальними авіалініями». Кожен рівень характеризується власними обмеженнями за довжиною, швидкістю, кількістю вузлів та енергетичним бюджетом, що закріплено у відповідних стандартах і специфікаціях виробників.

На найнижчому рівні ієрархії розташовані внутрішньокристальні шини, які забезпечують обмін даними між процесорним ядром, оперативною пам'яттю, контролерами периферійних модулів та інтерфейсними блоками в межах одного кристала. Ці шини не мають фізичних конекторів у звичному розумінні; вони реалізовані у вигляді логічних зв'язків у топології системи на кристалі (SoC) і характеризуються надзвичайно високими швидкостями (від сотень мегагерц до одиниць гігагерц), малою розрядністю (32, 64, 128 або 256 бітів) та мінімальною латентністю. Домінуючою архітектурою для мікроконтролерів із ядрами ARM є сімейство шин AMBA (Advanced Microcontroller Bus Architecture), розроблене компанією ARM Holdings. Специфікація AMBA охоплює кілька протоколів, що утворюють власну ієрархію всередині кристала: AHB (Advanced High-performance Bus) для високошвидкісних з'єднань між ядром і пам'яттю, APB (Advanced Peripheral Bus) для низькошвидкісних периферійних блоків, таких як UART, I^2C або таймери, та AXI (Advanced eXtensible Interface) для багатоканальних високопродуктивних потоків, характерних для мікропроцесорів серії Cortex-A та FPGA. У мікроконтролерах STM32 на ядрі Cortex-M4 або Cortex-M7, як зазначено в довідкових посібниках RM0090 та RM0468, матриця шин AHB об'єднує до трьох майстрів (ядро, DMA, Ethernet MAC) із кількома підлеглими пристроями (SRAM, Flash, периферійні шини), причому контролер DMA може паралельно виконувати передачі між пам'яттю та периферією без завантаження центрального процесора. Для мікроконтролерів на архітектурі RISC-V, таких як ESP32-C3 від Espressif, застосовується відкрита внутрішньокристальна шина Wishbone, описана в специфікації Wishbone B4 компанії OpenCores. Внутрішньокристальний рівень є прозорим для прикладного програміста, однак розуміння його структури стає критичним під час налагодження складних сценаріїв із використанням DMA, коли, наприклад, одночасний доступ двох майстрів до одного підлеглого пристрою призводить до стану гонки.

Наступний рівень становлять міжкомпонентні інтерфейси, що з'єднують окремі мікросхеми на друкованій платі. Відстані тут вимірюються

сантиметрами або десятками сантиметрів, швидкості — від десятків кілогерц до десятків мегагерц, а кількість провідників є критичним фактором вартості та технологічності монтажу. До цього рівня належать усі класичні послідовні інтерфейси, розглянуті в першій частині посібника: UART, SPI, I²C, I²S, а також їхні розширені варіанти, такі як QSPI (Quad SPI) для зовнішньої флеш-пам'яті. Типовим представником цього рівня є підключення зовнішнього датчика температури та вологості BME680 до мікроконтролера по шині I²C на відстані 5 см: чотири доріжки на платі (VCC, GND, SDA, SCL), швидкість 400 кбіт/с, адресація 0x76, живлення 3,3 В. На цьому ж рівні працюють і паралельні інтерфейси для підключення дисплеїв (8080-сумісний паралельний інтерфейс на 8 або 16 бітів), зовнішньої SRAM через FSMC, а також спеціалізовані інтерфейси для камер (DCMI — Digital Camera Interface). У технічній документації виробників, зокрема в посібнику з розробки апаратного забезпечення AN4488 для STM32F4, детально регламентовано вимоги до трасування цих ліній: узгодження імпедансів для SPI на частотах понад 30 МГц, обмеження ємності шини I²C до 400 пФ, застосування підтягувальних резисторів із номіналом, розрахованим за формулою, що враховує ємність лінії та напругу живлення.

Третій рівень утворюють системні та локальні мережеві інтерфейси, що з'єднують пристрої в межах одного приміщення, автомобіля, технологічної лінії або будівлі. Відстані зростають до десятків і сотень метрів, кількість вузлів у мережі може сягати десятків або сотень, а вимоги до завадостійкості та надійності стають домінуючими. До цього рівня належать, з одного боку, дротові промислові шини — CAN (ISO 11898), RS-485 із протоколом Modbus RTU, LIN (ISO 17987) для автомобільних підсистем, — а з іншого — локальні бездротові мережі на зразок Bluetooth LE Mesh, Zigbee (IEEE 802.15.4) та Thread. Особливе місце на цьому рівні посідає Ethernet (IEEE 802.3), який історично належав до офісних мереж, але в модифікації Single Pair Ethernet (10BASE-T1S за IEEE 802.3cg-2019) активно проникає у вбудовані системи, замінюючи традиційні польові шини. На цьому рівні принципово змінюється модель адресації: якщо на міжкомпонентному рівні використовується фізична адресація (вивід CS у SPI, 7-бітна адреса в I²C), то на системному з'являється логічна адресація — ідентифікатори повідомлень CAN, IP-адреси в Ethernet, групові адреси в Bluetooth Mesh. Крім того, саме на цьому рівні вперше виникає потреба в мережевих операційних системах або стеках протоколів (LwIP для Ethernet, BLE Mesh Stack від Espressif), оскільки обробка колізій, маршрутизація та повторні передачі вже не можуть бути реалізовані простим опитуванням прапорців у регістрах периферії.

Четвертий рівень охоплює глобальні мережеві інтерфейси, що забезпечують зв'язок пристрою із зовнішньою інфраструктурою — хмарними сервісами, серверами збору даних, іншими географічно віддаленими об'єктами. Тут відстані вимірюються кілометрами або тисячами кілометрів, а кількість потенційних вузлів сягає мільйонів. До цього рівня належать технології LPWAN (Low Power Wide Area Network) — LoRaWAN, NB-IoT, LTE-M (Cat-M1), — які оптимізовані для передавання малих обсягів даних із мінімальним енергоспоживанням, а також стільникові мережі 4G/5G для більш вимогливих до пропускної здатності застосувань. Згідно з дослідженнями, опублікованими в журналі IEEE Internet of Things Journal (том 10, 2023), саме рівень глобальних мереж є найвужчим місцем з погляду енергоспоживання: передача одного кілобайта даних через NB-IoT споживає на три порядки більше енергії, ніж обмін тим самим кілобайтом через I²C, що змушує розробників ретельно планувати розклад пробуджень модему. На цьому рівні до фізичних і канальних протоколів додаються протоколи прикладного рівня — MQTT, CoAP, LwM2M, — які забезпечують сумісність із хмарними платформами (AWS IoT Core, Azure IoT Hub, Google Cloud IoT) і вже не залежать від конкретного фізичного середовища. Таким чином, ієрархія інтерфейсів не лише впорядковує фізичні з'єднання, а й визначає рівні програмних абстракцій, із якими працює розробник.

Для наочного узагальнення описаної ієрархії в таблиці 1.7 зведено ключові характеристики кожного рівня: типові відстані, швидкості, кількість вузлів, представницькі інтерфейси та нормативні документи. Зауважимо, що межі між рівнями не є абсолютно жорсткими: наприклад, SPI з підсилювачами може працювати на відстані кількох метрів, а CAN FD із його швидкістю до 8 Мбіт/с за певних умов наближається до системного рівня, однак запропонована чотирирівнева модель дає студентові зручну систему координат для початкового аналізу будь-якої вбудованої системи. Систематизоване порівняння цих параметрів дозволяє обґрунтовано підійти до проєктування апаратного забезпечення, мінімізуючи ризики надлишковості або недостатності ресурсів на етапі вибору системної архітектури.

Графічно ієрархію інтерфейсів можна представити у вигляді концентричних кіл або піраміди, де кожен наступний рівень охоплює більшу фізичну територію, але, як правило, характеризується нижчою питомою пропускною здатністю у перерахунку на один вузол. Приклад такої візуалізації для реального IoT-пристрою наведено на рисунку 1.10.

Таблиця 1.7 – Ієрархія інтерфейсів у вбудованих системах

Рівень ієрархії	Характеристика	Значення
1. Внутрішньо-кристальний (on-chip)	*Типова відстань*	мікрометри – міліметри (всередині кристала)
	Типова швидкість передавання	до десятків Гбіт/с
	Кількість вузлів	2–10 майстрів/підлеглих
	Енергоспоживання на біт	одиниці пДж/біт
	Приклади інтерфейсів	AMBA (AHB, APB, AXI), Wishbone, NoC
	Нормативна база	ARM AMBA Specification, Wishbone B4
2. Між-компо-нентний (board-level)	*Типова відстань*	сантиметри – десятки сантимет-рів
	Типова швидкість передавання	від кбіт/с до десятків Мбіт/с
	Кількість вузлів	2–10
	Енергоспоживання на біт	десятки–сотні пДж/біт
	Приклади інтерфейсів	UART, SPI, I²C, QSPI, I²S
	Нормативна база	TIA/EIA-232-F, NXP UM10204, специфікації виробників
3. Системний / локальна ме-режа (field/local area)	*Типова відстань*	метри – сотні метрів
	Типова швидкість передавання	від кбіт/с до 10 Гбіт/с (Ethernet)
	Кількість вузлів	десятки – сотні
	Енергоспоживання на біт	нДж/біт – мкДж/біт
	Приклади інтерфейсів	CAN/CAN FD, RS-485/Modbus, Ethernet, BLE Mesh, Zigbee
	Нормативна база	ISO 11898, TIA/EIA-485, IEEE 802.3, IEEE 802.15.4
4. Глобальна мережа (wide area)	*Типова відстань*	кілометри – тисячі кілометрів
	Типова швидкість передавання	від сотень біт/с до сотень Мбіт/с
	Кількість вузлів	тисячі – мільйони
	Енергоспоживання на біт	мкДж/біт – мДж/біт
	Приклади інтерфейсів	LoRaWAN, NB-IoT, LTE-M, 5G NR
	Нормативна база	LoRaWAN Specification, 3GPP TS 36.101, IEEE 802.11 (для Wi-Fi HaLow)

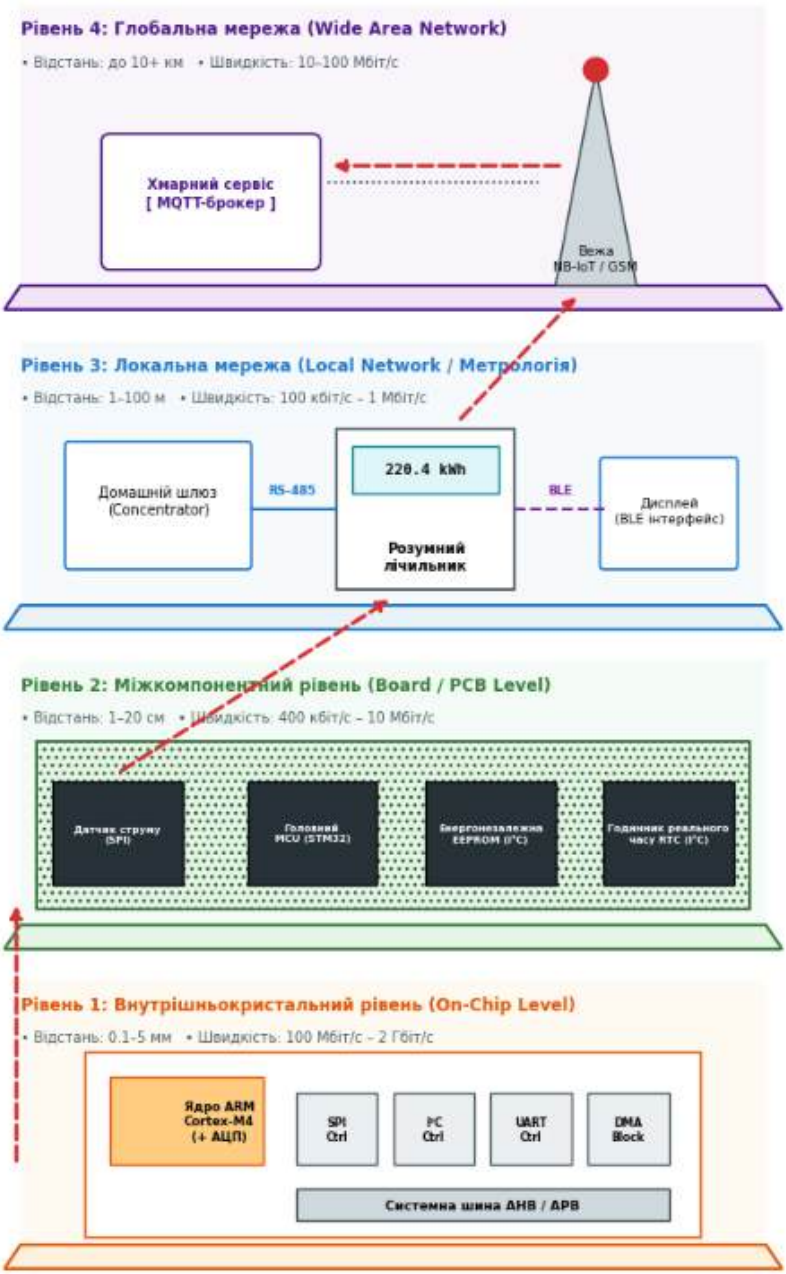


Рисунок 1.10 – Ієрархія інтерфейсів на прикладі розумного лічильника

На цій схемі чітко видно, як дані від внутрішнього АЦП через внутрішню шину AHB потрапляють до ядра Cortex-M, потім через I²C зчитуються додаткові параметри з мікросхеми годинника реального часу, після чого агрегований пакет через UART передається на NB-IoT модем, який, своєю чергою, відправляє його через стільникову мережу на хмарний брокер MQTT.

Практичне значення ієрархічного підходу полягає в тому, що він дозволяє розробникові поставити правильні запитання на етапі проєктування. На внутрішньокристальному рівні головним є питання пропускної здатності шини та пріоритетів доступу до пам'яті; на міжкомпонентному — вибір між кількістю провідників, енергоспоживанням і швидкістю; на системному — топологія, адресація та механізми арбітражу; на глобальному — енергетичний бюджет, тарифи на передачу даних і протоколи безпеки. Студент, який опанує цю ієрархію, отримує системне мислення, що необхідне для роботи над комплексними IoT-проєктами, де один пристрій одночасно є і вузлом внутрішньої шини, і клієнтом локальної mesh-мережі, і віддаленим терміналом глобальної LPWAN-системи.

1.4. Поняття стеку протоколу: фізичний, канальний, мережевий, транспортний та прикладний рівні стосовно інтерфейсів IoT

У найпростішому випадку, коли мікроконтролер з'єднано з датчиком через кілька сантиметрів друкованої плати за допомогою I²C, розробник має справу з фактично однорівневою взаємодією: фізичні рівні напруг, послідовність бітів на лінії SDA та програмна команда читання регістра зливаються в єдиний, нерозривний процес. Однак щойно до системи додається бездротовий канал, необхідність адресації кількох пристроїв, гарантованої доставки даних через ненадійне середовище або сумісності з хмарним сервісом, ця монолітна простота руйнується. На зміну їй приходить багаторівнева архітектура, відома як стек протоколу, де кожен рівень розв'язує чітко окреслену задачу, надаючи вищому рівневі послуги через стандартизований інтерфейс і приховуючи деталі реалізації нижчого рівня. Така декомпозиція є не теоретичною абстракцією, а нагальною інженерною потребою: без неї написати програму, яка одночасно працює з BLE, Wi-Fi та Ethernet, було б неможливо, оскільки довелося б враховувати особливості модуляції, часових слотів і форматів кадрів у кожному окремому драйвері. Саме тому в сучасному Інтернеті речей стек протоколу є центральним поняттям, навколо якого організовано і програмне забезпечення мікроконтролера, і взаємодію з хмарними платформами.

Концепція багаторівневого стеку сягає корінням в еталонну модель взаємодії відкритих систем (OSI, Open Systems Interconnection), стандартизовану в документі ISO/IEC 7498-1:1994. Модель OSI визначає сім рівнів — від фізичного до прикладного, — однак у вбудованих IoT-системах на практиці застосовують спрощену п'ятирівневу модель, що включає фізичний, канальний, мережевий, транспортний і прикладний рівні. Сеансовий рівень і рівень представлення, передбачені OSI, у вбудованих стеках зазвичай розчиняються в прикладному або транспортному рівнях, оскільки жорсткі обмеження за обсягом пам'яті та енергоспоживанням не дозволяють розкіш роздутого стеку. Водночас п'ять рівнів є необхідним і достатнім набором, щоб описати будь-який сучасний IoT-протокол — від простого обміну через UART між мікроконтролером і модулем LoRa до складного з'єднання пристрою з хмарним брокером MQTT через Wi-Fi 7. Відповідність між моделлю OSI та п'ятирівневою моделлю IoT-стеку наведено в таблиці 1.8, яка слугуватиме читачеві дорожньою картою під час подальшого вивчення конкретних протоколів у наступних розділах посібника.

Таблиця 1.8 – Відповідність рівнів моделі OSI та IoT-стеку

Рівень OSI (номер і назва)	Рівень IoT-стеку	Функція в IoT-системі
7. Прикладний	Прикладний (MQTT, CoAP, LwM2M)	Форматування даних застосунків, керування пристроєм, бізнес-логіка
6. Представлення	Об'єднано з прикладним (CBOR, JSON)	Серіалізація/десеріалізація даних, стиснення, кодування
5. Сеансовий	Об'єднано з транспортним (TLS/DTLS handshake)	Встановлення захищеної сесії, узгодження параметрів безпеки
4. Транспортний	Транспортний (TCP, UDP, TLS)	Наскрізна надійність (TCP), контроль перевантаження, безпека (TLS/DTLS)
3. Мережевий	Мережевий (IP, 6LoWPAN, RPL)	Адресація, маршрутизація пакетів, фрагментація, адаптація до мереж з низьким енергоспживанням
2. Канальний	Канальний (MAC-рівень, BLE Link Layer, LoRaWAN MAC)	Керування доступом до середовища, кадрування, виявлення помилок на лінії
1. Фізичний	Фізичний (радіо, Ethernet PHY, UART)	Передавання бітів: модуляція, рівні напруги, частота, синхронізація

Фізичний рівень у контексті IoT-стеку відповідає за передавання окремих бітів або символів через середовище — радіоефір, виту пару, оптоволокно чи навіть ультразвук. У чотирирівневій моделі, розглянутій у підрозділі 1.1, саме фізичний та електричний рівні разом утворюють фізичний рівень стеку. Тут визначаються робоча частота (наприклад, 2,4 ГГц для BLE та Wi-Fi, 868 МГц для LoRa в Європі), схема модуляції (GFSK для Bluetooth LE, CSS для LoRa, OFDM для Wi-Fi 7), вихідна потужність передавача та чутливість приймача, а також швидкість передачі символів. У дротових IoT-інтерфейсах фізичний рівень часто представлений стандартним UART або SPI з'єднанням між хостовим мікроконтролером і периферійним модулем, де біти передаються рівнями TTL/CMOS. Специфікації фізичного рівня завжди жорстко регламентовані, оскільки від них залежить електромагнітна сумісність і можливість спільної роботи пристроїв різних виробників: наприклад, ETSI EN 300 220 визначає вимоги до радіопристроїв малого радіусу дії в Європі, а IEEE 802.15.4-2020 — до фізичного рівня Zigbee та Thread. Для розробника програмного забезпечення фізичний рівень зазвичай прихований за апаратним драйвером, однак такі параметри, як рівень потужності передавача або швидкість передачі символів, часто доступні для конфігурування через API, і їхній неправильний вибір може призвести до повної втрати зв'язку.

Канальний рівень (Data Link Layer) надбудовується над фізичним і відповідає за формування кадрів (фреймів), адресацію пристроїв у межах одного сегмента мережі, виявлення та корекцію помилок, а також керування доступом до спільного середовища (MAC, Medium Access Control). Саме на цьому рівні з'являються такі поняття, як MAC-адреса в Ethernet, адреса пристрою в Bluetooth (BD_ADDR) або ідентифікатор кінцевого пристрою в LoRaWAN (DevEUI). Канальний рівень забезпечує, щоб кадр, відправлений одним вузлом, був отриманий саме тим сусіднім вузлом, якому він призначений, і щоб поодинокі бітові помилки, неминучі на фізичному рівні через шуми та завади, були виявлені за допомогою контрольної суми (CRC). У бездротових мережах канальний рівень також реалізує механізми уникнення колізій — CSMA/CA в Zigbee та Thread, TDMA в LTE-M, адаптивний вибір частоти в BLE. На цьому ж рівні вводиться поняття підтвердження (ACK): після успішного прийому кадру приймач надсилає коротке підтвердження, а відсутність ACK у встановлений таймаут спонукає передавач до повторної спроби. Як приклад, у специфікації LoRaWAN 1.1 (LoRa Alliance, 2022) детально описано MAC-команди для підтвердження, керування швидкістю (ADR) та безпеки на канальному рівні. Для студента практичне знайомство з канальним рівнем часто починається з аналізу

дампів обміну через логічний аналізатор під час передачі даних I²C або SPI, де вже видно кадрову структуру з адресою, даними та бітом підтвердження.

Мережевий рівень (Network Layer) розширює можливості адресації за межі одного фізичного сегмента, уможливлюючи передавання пакетів між пристроями, які не мають безпосереднього з'єднання. У світі TCP/IP, що домінує в IoT, мережевий рівень представлений протоколом IP (Internet Protocol), де кожен пристрій отримує унікальну логічну адресу (IPv4 або IPv6), а маршрутизатори перенаправляють пакети на основі таблиць маршрутизації. У вбудованих системах, особливо в низькошвидкісних бездротових мережах, використовують адаптовані версії: 6LoWPAN (RFC 6282) забезпечує стиснення заголовків IPv6 для передачі через IEEE 802.15.4, а протокол RPL (RFC 6550) реалізує маршрутизацію в mesh-мережах із низьким енергоспоживанням. Водночас існують IoT-стеки, де мережевий рівень не базується на IP: у Bluetooth Mesh застосовано власний механізм керованого лавиноподібного поширення (managed flooding), а в Zigbee — ієрархічну маршрутизацію на основі дерева та AODV. Для розробника мережевий рівень є критичним під час проєктування топології мережі та оцінки масштабованості: саме тут вирішується, чи потрібна система з NAT, чи достатньо локальної адресації, чи підтримується мобільність вузлів і який обсяг службового трафіку генеруватимуть протоколи маршрутизації.

Транспортний рівень (Transport Layer) надає вищим рівням абстракцію надійного (або ненадійного) каналу зв'язку між двома кінцевими точками, приховуючи від прикладної програми деталі маршрутизації, повторних передач і фрагментації. Двома основними транспортними протоколами в IoT є TCP (Transmission Control Protocol), що гарантує доставку та збереження порядку байтів, і UDP (User Datagram Protocol), що забезпечує мінімальну латентність ціною відсутності гарантій. Більшість прикладних протоколів IoT обирають один із цих двох транспортів: MQTT працює поверх TCP, забезпечуючи надійну доставку команд і телеметрії, тоді як CoAP (Constrained Application Protocol) використовує UDP, щоб уникнути накладних витрат на встановлення з'єднання в мережах із високою латентністю. Окреме місце на транспортному рівні посідають протоколи безпеки: TLS (Transport Layer Security) версії 1.3, що працює поверх TCP, і DTLS (Datagram TLS), адаптований для UDP. Саме транспортний рівень для багатьох IoT-пристроїв є межею, за якою починається «великий Інтернет»: мікроконтролер із LwIP-стеком формує TCP-сегменти, встановлює з'єднання з хмарним сервером і обробляє віконне керування потоком, тоді як фізичний і канальний рівні залишаються під керуванням апаратних драйверів.

Прикладний рівень (Application Layer) є найвищим у стеку і безпосередньо взаємодіє з прикладною логікою пристрою. Саме на цьому рівні працюють протоколи, які визначають семантику повідомлень: MQTT v5.0 із темами, рівнями якості обслуговування (QoS 0/1/2) та властивостями пакетів; CoAP із моделлю REST (GET, POST, PUT, DELETE) і форматом корисного навантаження у вигляді JSON або CBOR; LwM2M (Lightweight Machine-to-Machine) для керування парком пристроїв і оновлення прошивок (FOTA). Прикладний рівень не залежить від того, через яке фізичне середовище передаються дані: MQTT-повідомлення, що містить покази датчика, виглядатиме ідентично незалежно від того, чи було його відправлено через Wi-Fi, Ethernet або супутниковий канал. Ця інваріантність і є головною метою багаторівневої архітектури: зміна радіомодуля з BLE на LoRa не повинна вимагати переписування всієї прикладної логіки, достатньо замінити драйвери нижніх рівнів і, можливо, транспортний протокол. У таблиці 1.9 наведено конкретні приклади того, як розподіляються функції за рівнями в трьох найтиповіших IoT-стеках: BLE із GATT-профілем, LoRaWAN із MQTT та Wi-Fi із CoAP.

Таблиця 1.9 – Розподіл протоколів за рівнями в поширених IoT-стеках

Рівень стеку	Стек 1: BLE з GATT (датчик)	Стек 2: LoRaWAN + MQTT (лічильник)	Стек 3: Wi-Fi + CoAP (актуатор)
Прикладний	GATT-профілі (Heart Rate, Environmental Sensing)	MQTT v5.0 (JSON-повідомлення)	CoAP (REST, CBOR)
Транспортний	L2CAP (фрагментація)	TCP/TLS 1.3	UDP/DTLS 1.3
Мережевий	не застосовується (пряме з'єднання)	IPv6 через 6LoWPAN	IPv4/IPv6
Канальний	BLE Link Layer (реклама, з'єднання)	LoRaWAN MAC (ADR, підтвердження)	Wi-Fi MAC (CSMA/CA)
Фізичний	2,4 ГГц GFSK	LoRa CSS 868 МГц	Wi-Fi PHY OFDM 5 ГГц

Наочне уявлення про те, як дані проходять крізь рівні стеку від датчика до хмарного сервісу, дає рисунок 1.11, на якому зображено наскрізний шлях одного пакета телеметрії. На цьому рисунку видно, як на кожному

рівні до корисного навантаження додається відповідний заголовок, утворюючи вкладений конверт, а на приймальному боці заголовки послідовно знімаються знизу вгору.

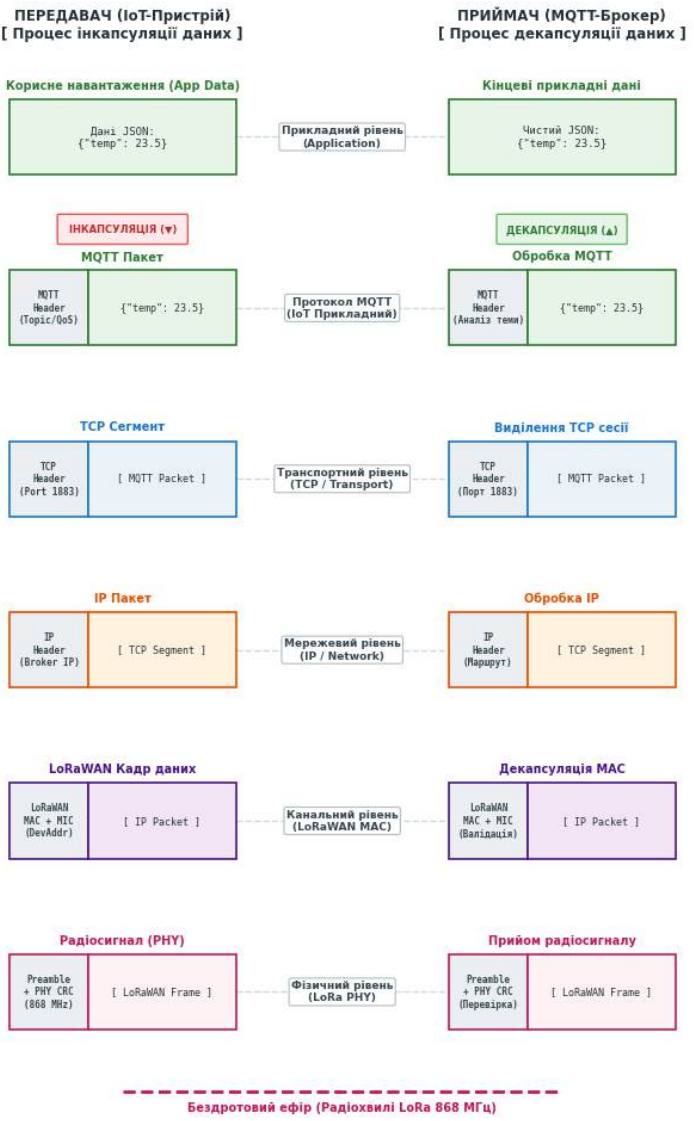


Рисунок 1.11 – Інкапсуляція даних у стеку протоколів IoT на прикладі MQTT поверх LoRaWAN

Слід підкреслити, що в реальних мікроконтролерних реалізаціях стек протоколу не завжди є монолітним програмним блоком; часто він розподілений між апаратним модулем (наприклад, контролер Wi-Fi в ESP32 реалізує фізичний, канальний і частково мережевий рівні апаратно), операційною системою реального часу (FreeRTOS із стеком LwIP) та прикладною бібліотекою (Eclipse Paho для MQTT). Такий розподіл дозволяє оптимізувати продуктивність, однак вимагає від розробника розуміння того, який саме рівень відповідає за ту чи іншу функцію, особливо під час налагодження: наприклад, помилка «Connection refused» на рівні MQTT може бути спричинена не неправильним паролем, а нездатністю TCP встановити з'єднання через втрати пакетів на канальному рівні LoRaWAN. Отже, п'ятирівнева модель стеку протоколу є не просто теоретичним конструктом, а практичним діагностичним інструментом, яким інженер користується щодня. У наступних частинах посібника для кожного інтерфейсу явно зазначатиметься його місце в цій ієрархії, що допоможе читачеві вибудувати стрункy систему знань замість набору розрізнених фактів про окремі протоколи.

РОЗДІЛ 2.
МЕТОДОЛОГІЯ АНАЛІЗУ ТА ВИБОРУ ІНТЕРФЕЙСУ

2.1. Ключові технічні характеристики

Методологія інженерного вибору інтерфейсу для конкретної задачі спирається на кількісне зіставлення характеристик доступних рішень із вимогами, що їх висуває система. Жоден інтерфейс не є універсально кращим; кожен становить компроміс між швидкістю, енергоспоживанням, дальністю, завадостійкістю, складністю реалізації та вартістю компонентів. У цьому підрозділі послідовно розглядаються чотири групи ключових характеристик, які мають бути проаналізовані на етапі проєктування: швидкісні показники, енергетична ефективність, цілісність сигналу та детермінованість. Початок аналізу завжди покладає оцінка того, скільки даних і з якою швидкістю необхідно передавати, — саме це становить предмет першого з розглянутих критеріїв.

2.1.1. Пропускна здатність і швидкість передачі символів

Пропускна здатність каналу зв’язку є тією первинною метрикою, з якої розробник розпочинає аналіз придатності інтерфейсу для конкретного застосування. Під пропускною здатністю розуміють максимальну кількість інформації, що може бути доставлена через канал за одиницю часу в заданих умовах, і вимірюють її в бітах за секунду (біт/с) або кратних одиницях. Проте в технічній документації та на практиці часто фігурують два різні, хоча й пов’язані, поняття: символьна швидкість (Baud rate), що визначає кількість змін стану сигналу в каналі за секунду, і бітова швидкість (bit rate), що визначає кількість переданих бітів за секунду. Співвідношення між ними задається кількістю рівнів модуляції M, де кожен символ переносить $\log_2(M)$ бітів інформації. Для простих двійкових схем, таких як NRZ в UART або SPI, M дорівнює двом, і символьна швидкість чисельно збігається з бітовою. Однак у складніших системах, наприклад, у 1000BASE-T Ethernet із модуляцією PAM-5, один символ несе 2 біти, тож бітова швидкість учетверо перевищує символьну завдяки одночасному використанню чотирьох пар. Формально цей зв’язок виражається через формулу (2.1), за Найквістом для ідеального каналу без шуму:

$$C_{\text{симв}} = 2 * B * \log_2(M) \ , \tag{2.1}$$

де B — смуга пропускання каналу в герцах, M — кількість дискретних рівнів сигналу. У реальних каналах, де присутні теплові шуми та завади, граничну пропускну здатність визначає теорема Шеннона-Хартлі (2.2):

$$C = B * \log_2(1 + S/N) \ , \tag{2.2}$$

де S/N — відношення потужності сигналу до потужності шуму. Ці фундаментальні обмеження, докладно описані в класичних працях із теорії інформації (Shannon, 1948; Cover & Thomas, 2012), є теоретичною стелею, до якої наближаються, але якої ніколи не досягають реальні інтерфейси.

Для інженера, що обирає інтерфейс мікроконтролера, більш практичне значення має не теоретична межа, а номінальна бітова швидкість, яку гарантує стандарт або виробник мікросхеми, та реальна пропускна здатність на рівні корисних даних (goodput). Остання завжди нижча за номінальну бітову швидкість через неминучі службові витрати: стартові та стопові біти в UART, біти підтвердження та адреси в I²C, поля арбітражу, CRC та міжкадрові проміжки в CAN, заголовки пакетів у TCP/IP. Наприклад, для асинхронного UART у конфігурації 8N1 (8 бітів даних, без паритету, 1 стоп-біт) кожен переданий байт корисних даних потребує 10 бітів на лінії — стартовий біт, вісім інформаційних і стоповий біт. Отже, корисна пропускна здатність становить лише 80% від номінального бітрейту. У мережі CAN зі стандартним 11-бітним ідентифікатором для передачі 8 байтів корисних даних витрачається до 125 бітів із урахуванням полів SOF, арбітражу, контролю, CRC, ACK і міжкадрового проміжку, що дає ефективність близько 51%. Для порівняння, в Ethernet-фреймі мінімального розміру (64 байти) корисне навантаження становить лише 46 байтів після вирахування MAC-заголовків і CRC, тобто ефективність близько 72%, і зростає до понад 97% для фреймів максимального розміру завдяки меншій відносній частці заголовків.

Отже, першим кроком під час вибору інтерфейсу є обчислення мінімально необхідної номінальної бітової швидкості, виходячи з обсягу даних, що генеруються за одиницю часу, та допустимого коефіцієнта службових витрат. Якщо, наприклад, датчик формує 100 байтів телеметрії кожні 10 мс, а протокол передбачає 20% накладних витрат, необхідний бітрейт становитиме не менше 100 * 8 * 1,2 / 0,01 = 96 000 біт/с. Жоден UART на швидкості 115 200 біт/с тут не підійде, якщо не врахувати також час на реакцію програмного забезпечення та можливі колізії. Водночас для пристрою, що раз

на годину відправляє 50 байтів показів через LPWAN-мережу, навіть швидкість 300 біт/с є цілком достатньою, і на перший план виходять уже не швидкісні, а енергетичні критерії.

Для полегшення порівняльного аналізу в таблиці 2.1 зведено максимальні символьні та бітові швидкості для найпоширеніших інтерфейсів, що застосовуються у вбудованих системах, разом із характерною ефективною пропускною здатністю корисних даних і посиланнями на відповідні нормативні документи.

Таблиця 2.1 – Швидкісні характеристики поширених інтерфейсів вбудованих систем

Інтерфейс	Максимальна символьна швидкість (бод)	Максимальна бітова швидкість (біт/с)	Типова ефективність використання каналу (%)	Типова ефективна пропускна здатність корисних даних	Нормативний документ або специфікація
UART (асинхронний, 8N1)	10 Мбод (деякі MCU)	10 Мбіт/с	80% (через старт/стоп)	до 8 Мбіт/с	TIA/EIA-232-F
SPI (одинарний)	до 100 МГц тактової → 100 Мбод	до 100 Мбіт/с	~100% (без службових бітів у потоці)	до 100 Мбіт/с	де-факто стандарт виробників
I²C (Fast-mode Plus)	1 Мбод	1 Мбіт/с	~90% (адреса + ACK)	~0,9 Мбіт/с	NXP UM10204
CAN 2.0	1 Мбод	1 Мбіт/с	~57% (для 8-байтових даних)	~0,57 Мбіт/с	ISO 11898-1
CAN FD (дані 64 байти)	8 Мбод (фаза даних)	8 Мбіт/с (фаза даних)	~75% (64 байти)	~6 Мбіт/с	ISO 11898-1:2015
RS-485 (на 10 м)	10 Мбод (UART-подібний)	10 Мбіт/с	80% (якщо UART 8N1)	до 8 Мбіт/с	TIA/EIA-485-A
USB 2.0 Full Speed (Bulk)	12 Мбод (сигнальна)	12 Мбіт/с	~80% (пакетний overhead)	~9,6 Мбіт/с (~1,2 МБ/с)	USB 2.0 Specification
100BASE-TX Ethernet (мінімальний кадр)	125 Мбод (MLT-3)	100 Мбіт/с	~72% (64-байтовий кадр)	~72 Мбіт/с	IEEE 802.3u

Кінець таблиці 2.1

Інтерфейс	Максимальна символьна швидкість (бод)	Максимальна бітова швидкість (біт/с)	Типова ефективність використання каналу (%)	Типова ефективна пропускна здатність корисних даних	Нормативний документ або специфікація
100BASE-TX Ethernet (максимальний кадр)	125 Мбод (MLT-3)	100 Мбіт/с	~97,5% (1518-байтовий кадр)	~97,5 Мбіт/с	IEEE 802.3u
BLE 5.4 (2M PHY)	2 Мсимв/с	2 Мбіт/с	~70% (L2CAP)	до ~1,4 Мбіт/с	Bluetooth Core 5.4
LoRa (SF7, 125 кГц)	залежить від SF	~5 470 біт/с	~90%	~4 920 біт/с	LoRaWAN Specification 1.1
LoRa (SF12, 125 кГц)	залежить від SF	~300 біт/с	~90%	~270 біт/с	LoRaWAN Specification 1.1

Швидкісні можливості інтерфейсу неможливо розглядати у відриві від фізичної довжини лінії та середовища передавання. У дротових інтерфейсах зі збільшенням відстані неминуче зростають втрати в кабелі, спотворення фронтів через дисперсію та вплив зовнішніх завад, що змушує знижувати символьну швидкість для збереження прийнятного рівня бітових помилок. Ця залежність ілюструється рисунком 2.1, на якому наведено графіки максимальної ефективної пропускної здатності для трьох поширених дротових інтерфейсів — RS-485, CAN FD та 100BASE-TX Ethernet — залежно від довжини кабелю. З графіків видно, що RS-485, згідно з рекомендаціями TIA/EIA-485-A, здатний підтримувати швидкість 10 Мбіт/с лише на відстанях до 12 м, тоді як на кілометрових дистанціях швидкість падає до 100 кбіт/с. Натомість CAN FD зі швидкістю фази даних 5 Мбіт/с обмежений приблизно 40 м у стандартній реалізації згідно з ISO 11898-2, тоді як Ethernet 100BASE-TX стабільно забезпечує 100 Мбіт/с на відстані до 100 м завдяки диференційній передачі та суворому нормуванню параметрів кабелю категорії 5e згідно з IEEE 802.3. Для бездротових інтерфейсів аналогічну роль відіграє відстань до приймача, яка визначає бюджет каналу: LoRa на швидкості SF12 здатна працювати на відстані понад 10 км у сільській місцевості, але зі швидкістю лише близько 300 біт/с, тоді як BLE 2M PHY забезпечує 2 Мбіт/с, проте типова дальність у приміщенні не перевищує 20 м. Отже, аналіз пропускної здатності має супроводжуватися оцінкою відповідності системи обмеженням за дальністю, що їх накладають відповідні стандарти та закони фізики.

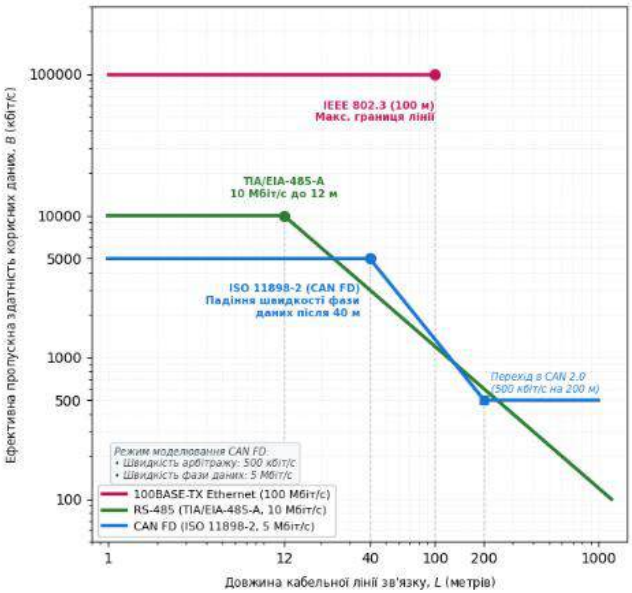


Рисунок 2.1 – Залежність ефективної пропускної здатності від довжини лінії для RS-485, CAN FD та 100BASE-TX Ethernet

Практичне застосування наведених залежностей і табличних даних полягає в тому, що розробник на етапі ескізного проєктування виконує розрахунок за наступною послідовністю: визначає максимальний обсяг даних D, що передаються за один сеанс зв'язку, та максимально допустимий час передачі T_{max}; обчислює мінімальну корисну пропускну здатність $C_{кор} = D / T_{max}$; уводить коефіцієнт запасу k = 1,5–2 для врахування службових витрат і пікових навантажень; обирає інтерфейс, чия номінальна бітова швидкість, помножена на коефіцієнт ефективності з таблиці 2.1, перевищує $C_{кор}$ з урахуванням запасу; перевіряє за рисунком 2.1 або аналогічними даними, чи забезпечується необхідна швидкість на цільовій відстані. Лише після цього має сенс переходити до аналізу інших критеріїв — енергоспоживання, завадостійкості та детермінованості, які розглядатимуться в наступних підпунктах. Важливо пам'ятати, що надлишкова пропускна здатність, не підкріплена реальною потребою, завжди означає перевитрату енергії та ускладнення системи, тому для багатьох IoT-застосувань свідоме обмеження швидкості є не вадою, а ознакою грамотного інженерного рішення.

2.1.2. Енергоспоживання та режими зниженого живлення

Для переважної більшості пристроїв Інтернету речей, особливо тих, що живляться від автономних батарей або збирачів енергії (energy harvesters), енергоспоживання інтерфейсу є не просто однією з характеристик, а визначальним критерієм, що безпосередньо впливає на життєздатність усього виробу. Заміна елемента живлення в датчику, встановленому на висотній опорі мосту, у товщі бетонної конструкції або всередині людського тіла, є або надзвичайно дорогою, або взагалі неможливою операцією, тому інженери прагнуть забезпечити термін автономної роботи від 5 до 15 років від однієї літій-тіонілхлоридної батареї ємністю 2400 мА·год. Досягнення таких показників вимагає глибокого розуміння того, куди саме витрачається кожен мікроджоуль енергії в процесі комунікації, та свідомого вибору інтерфейсу, чий енергетичний профіль відповідає запланованому режиму роботи пристрою. Згідно з порівняльними дослідженнями, опублікованими в IEEE Communications Surveys and Tutorials (том 25, 2023), на бездротову передачу даних у типовому сенсорному вузлі припадає від 60% до 90% загального енергетичного бюджету, тоді як на обчислення та зберігання — значно менше, що робить саме інтерфейс головним споживачем і, відповідно, головною ціллю оптимізації.

Енергоспоживання будь-якого інтерфейсу складається з двох принципово різних компонентів: статичної потужності, що розсіюється безперервно, поки схема увімкнена (струм витоку транзисторів, живлення тактових генераторів, черговий режим приймача), та динамічної потужності, що споживається лише під час активних операцій — передавання пакета, приймання відповіді, перемикання станів автомата протоколу. Статична складова особливо критична для пристроїв, які постійно очікують на вхідний трафік: наприклад, радіоприймач Wi-Fi у режимі прослуховування (idle listen) споживає близько 50–80 мА за напруги 3,3 В, що швидко виснажує батарею. Натомість BLE-приймач у режимі очікування споживає на два порядки менше завдяки коротшим вікнам сканування та енергоефективній архітектурі фізичного рівня. Динамічна складова визначається енергією, необхідною для передавання одного біта або одного пакета даних, і вимірюється в джоулях на біт (Дж/біт) або міліджоулях на пакет (мДж/пакет). Для кількісної оцінки застосовують формулу (2.3):

$$E_{\text{пакет}} = P_{\text{акт}} * T_{\text{акт}} = (I_{TX} * V_{\text{ж}}) * (\frac{L_{\text{пакет}}}{R_{\text{біт}}}) \ , \qquad (2.3)$$

де I_{TX} — струм передавача в активному режимі, $V_{\text{ж}}$ — напруга живлення, $L_{\text{пакет}}$ — довжина пакета в бітах, $R_{\text{біт}}$ — бітова швидкість. Із цієї формули

випливає важливий наслідок: збільшення швидкості передавання за незмінної потужності передавача зменшує час активності, а отже, і загальну енергію на пакет. Саме тому високошвидкісні інтерфейси, як-от BLE 2M PHY або Wi-Fi 7 зі швидкостями понад 1 Гбіт/с, можуть виявитися енергетично вигіднішими за низькошвидкісні, якщо обсяг даних достатньо великий, оскільки пристрій швидше завершує сеанс зв'язку і повертається в режим сну. Це явище, відоме як «race to sleep», є одним із ключових принципів енергоощадного проєктування вбудованих систем.

Однак сама лише енергія на один пакет не дає повної картини без урахування режимів зниженого живлення, які підтримуються конкретним інтерфейсом і мікроконтролером. Сучасні мікроконтролери, зокрема STM32L4/L5 на ядрі ARM Cortex-M4/M33, пропонують ієрархію режимів сну: Sleep (зупинка ядра, периферія працює), Stop (зупинка ядра та більшості периферій, живлення ОЗП зберігається), Standby (повне відімкнення ядра та ОЗП, пробудження лише за зовнішніми сигналами або RTC) та Shutdown (найнижче споживання, пробудження тільки за фронтом на виводі). Споживання в режимі Shutdown може становити лише 30 нА, що дозволяє живити пристрій від крихітного конденсатора, зарядженого фотоелементом. Ключовим питанням стає здатність інтерфейсу пробуджувати мікроконтролер із глибокого сну за наявності вхідних даних. Наприклад, модуль UART у STM32 підтримує функцію Wake-up from Stop mode за фронтом стартового біта, що дозволяє тримати мікроконтролер у режимі Stop (споживання близько 1 мкА) і пробуджувати його лише тоді, коли зовнішній пристрій починає передавання. Аналогічно, інтерфейс I²C може бути налаштований на пробудження за збігом адреси (address match wake-up), що реалізовано в периферійних модулях серій STM32L4 та nRF52. У бездротових інтерфейсах механізми зниженого живлення є ще більш розвиненими: специфікація Bluetooth Core 5.4 визначає режими Advertising Extensions і Periodic Advertising, які дозволяють синхронізувати розклад пробуджень сканера та рекламодавця; протокол LoRaWAN для пристроїв класу A відкриває два короткі вікна прийому після кожної передачі, дозволяючи решту часу тримати радіотракт у сплячому режимі; стільникові модеми NB-IoT підтримують режими PSM (Power Saving Mode) та eDRX (extended Discontinuous Reception), у яких модем від'єднується від мережі на години або навіть дні, зберігаючи при цьому контекст і пробуджуючись лише для періодичного оновлення зони відстеження.

Кількісне зіставлення енергетичної ефективності різних інтерфейсів зручно виконувати за допомогою показника питомої енергії на переданий біт корисних даних, який враховує не лише активний струм передавача, а й

службові витрати протоколу, енергію на приймання підтверджень і пробудження системи. Такий аналіз, результати якого наведено в таблиці 2.2, показує, що найменшу питому енергію для малих пакетів забезпечують BLE та IEEE 802.15.4 (Zigbee/Thread) — близько 10–50 нДж/біт, тоді як LoRaWAN вимагає 1–10 мкДж/біт через триваліший час передачі, а NB-IoT — до 100 мкДж/біт через значне споживання модему під час підключення до мережі. Дротові інтерфейси, такі як I²C або UART, мають найнижчу питому енергію (менше 1 нДж/біт) завдяки малим напругам і відсутності потреби в підсиленні потужності, однак сфера їх застосування обмежена кількома метрами кабелю.

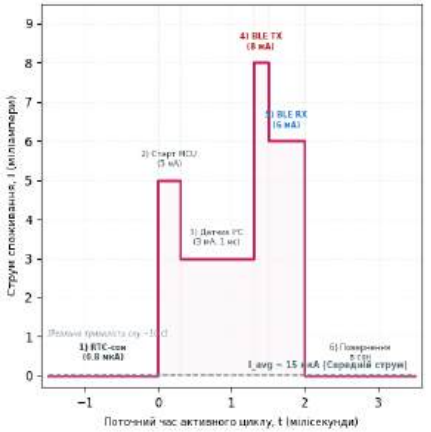
Таблиця 2.2 – Енергетичні характеристики поширених інтерфейсів вбудованих систем

Інтерфейс	Струм передачі (типовий, мА)	Струм прийому (типовий, мА)	Струм у режимі сну/очікування	Питома енергія на корисний біт	Ключові режими зниженого живлення	Нормативний документ або джерело даних
UART (TTL, 3,3 В)	~1–3 мА (залежно від навантаження)	~1–3 мА	<1 мкА (Stop mode із wake-up)	<1 нДж/біт	Wake-up за стартовим бітом, Stop mode	RM0090 STM32F4, AN4621 STM32L4
I²C (Fast-mode, 400 кГц)	~1 мА	~1 мА	<1 мкА	~8 нДж/біт	Wake-up за стартовою умовою, Stop mode	STM32L4xx Reference Manual, NXP UM10204
SPI (10 МГц)	~1–2 мА	~1–2 мА	<1 мкА	~0,5–1 нДж/біт	Тактовий вхід пробудження, Stop mode	RM0090 STM32F4, AN4621 STM32L4
BLE 5.4 (0 дБм, 1M PHY)	~5 мА	~5 мА	~0,5 мкА (sleep із RTC)	~25 нДж/біт	Advertising Extensions, Connection Subrating	Bluetooth Core 5.4, nRF52840 Product Specification
BLE 5.4 (0 дБм, 2M PHY)	~5,5 мА (TX)	~5,5 мА (RX)	~0,5 мкА	~12 нДж/біт	2M PHY (знижує енергію на біт), Connection Subrating	Bluetooth Core 5.4, nRF52840 PS
Zigbee (IEEE 802.15.4, 0 дБм)	~15 мА	~15 мА	<1 мкА	~200 нДж/біт (≈0,2 мкДж/біт)	Sleep з RTC (End Device), Polling	IEEE 802.15.4, CC2652R datasheet

Кінець таблиці 2.2

Інтерфейс	Струм передачі (типовий, мА)	Струм прийому (типовий, мА)	Струм у режимі сну/очікування	Питома енергія на корисний біт	Ключові режими зниженого живлення	Нормативний документ або джерело даних
LoRa (SF7, 14 дБм, 868 МГц)	~40 мА	~10 мА	<1 мкА (з RTC)	~25 мкДж/біт	Class A (два вікна RX після TX)	LoRaWAN 1.1, SX1262 Datasheet
LoRa (SF12, 14 дБм, 868 МГц)	~40 мА	~10 мА	<1 мкА (з RTC)	~450 мкДж/біт (див. прим.) [1]	Class A (два вікна RX після TX)	LoRaWAN 1.1, SX1262 Datasheet
NB-IoT (категорія NB1, 23 дБм)	~200 мА (піковий)	~60 мА	~3 мкА (PSM)	~50–100 мкДж/біт	PSM, eDRX	3GPP TS 36.101, Quectel BC660K-GL
Wi-Fi 6 (20 дБм, MCS7)	~300 мА	~100 мА	~5 мкА (глибокий сон)	~30 нДж/біт (PHY)	DTIM, TWT, Listen Interval	IEEE 802.11ax, ESP32-C6 Datasheet

Наочну ілюстрацію енергетичного профілю типового сенсорного вузла з бездротовим інтерфейсом наведено на рисунку 2.2, де зображено часову діаграму струму споживання впродовж повного циклу «сон — пробудження — вимірювання — передача — очікування підтвердження — повернення до сну». Такі діаграми, отримані за допомогою вимірювального приладу з високою часовою роздільною здатністю (наприклад, Nordic Power Profiler Kit II або Keysight CX3300), є стандартним інструментом налагодження енергоспоживання вбудованих пристроїв. На рисунку чітко видно, що найбільший внесок у загальну енергію циклу робить не власне передача пакета, а фаза очікування підтвердження, під час якої приймач залишається активним. Це спостереження має безпосередній практичний наслідок: протоколи, що мінімізують час активного очікування (BLE з короткими інтервалами з'єднання, LoRaWAN із фіксованими вікнами), виявляються значно енергоефективнішими за ті, що вимагають тривалого прослуховування ефіру.



Фаза / Процес	Формула (I * dt)	Заряд / Енергія
1) RTC-сон	0.8 мкА * 10.0 с	8.00 мкКл
2) Старт MCU	5.0 мА * 0.30 мс	1.50 мкКл
3) Датчик I2C	3.0 мА * 1.00 мс	3.00 мкКл
4) BLE TX	8.0 мА * 0.20 мс	1.60 мкКл
5) BLE RX	**6.0 мА * 0.50 мс**	**3.00 мкКл**
Загальний заряд кадру (Q_cycle):		**17.10 мкКл**
Енергія (E = Q * V) при V=3В		51.30 мкДж

Рисунок 2.2 – Часова діаграма струму споживання IoT-вузла з інтерфейсом BLE під час циклу передавання телеметрії

Для дротових інтерфейсів енергоспоживання хоча й не є настільки критичним, як для бездротових, однак також має значення в пристроях із живленням від мініатюрних батарей або пасивних збирачів енергії. Особливу увагу тут приділяють можливості відключати тактування периферійних блоків, коли вони не використовуються (clock gating), що реалізовано на апаратному рівні в мікроконтролерах STM32 (регістри RCC_AHBxENR, RCC_APBxENR) та ESP32 (Power Management Module). Наприклад, згідно з технічною документацією STM32L476 (документ DS10198), споживання мікроконтролера в режимі Stop 2 з активним RTC та збереженням ОЗП становить лише 0,8 мкА, тоді як увімкнення модуля UART додає до цього близько 5 мкА, а модуля I²C — близько 3 мкА. Ці, здавалося б, незначні мікроампери накопичуються за роки роботи в кулонометри, що скорочують термін служби батареї на місяці. Тому навіть у дротових системах рекомендується керувати живленням периферії явно, вмикаючи її лише на час обміну даними.

Підсумовуючи, аналіз енергоспоживання інтерфейсу не може обмежуватися порівнянням одного числа з таблиці даташита; він вимагає побудови повного енергетичного профілю пристрою з урахуванням робочого циклу, тривалості фаз сну, пробудження, передачі та прийому. Розробник має поставити такі запитання: чи підтримує інтерфейс пробудження мікроконтролера з глибокого сну за вхідною подією; яка тривалість і енергетична вартість процедури підключення до мережі (актуально для стільникових і

Wi-Fi інтерфейсів); чи можна зменшити частоту передач за рахунок агрегації даних на борту; чи виправдане застосування високошвидкісного режиму, якщо обсяг даних малий. Лише після отримання обґрунтованих відповідей на ці запитання, підкріплених вимірюваннями на реальному прототипі, можна вважати енергетичний аналіз завершеним і переходити до наступних критеріїв вибору.

2.1.3. Завадостійкість, цілісність сигналу та максимальна довжина лінії

Навіть найретельніше розрахована пропускна здатність і найоптимальніше налаштовані режими енергозбереження втрачають сенс, якщо фізичний сигнал на вході приймача спотворюється настільки, що біти неможливо розпізнати. У реальному світі жоден канал зв'язку не є ідеальним: електромагнітні завади від сусідніх цифрових схем, силових перетворювачів, електродвигунів та радіопередавачів накладаються на корисний сигнал; неоднорідності хвильового опору вздовж лінії спричиняють відбиття; скін-ефект і діелектричні втрати деградують фронти імпульсів. Тому третім ключовим критерієм аналізу інтерфейсу є його здатність зберігати цілісність сигналу в заданих умовах експлуатації — критерій, що безпосередньо визначає максимальну довжину лінії, вимоги до кабелю, топологію розведення друкованої плати та необхідність додаткових засобів захисту.

Завадостійкість інтерфейсу кількісно характеризують відношенням сигнал/шум (SNR) на вході приймача та запасом за рівнем (noise margin), що становить різницю між гарантованим вихідним рівнем передавача та порогом спрацьовування приймача. Для несиметричних ліній (single-ended), таких як UART TTL або SPI, рівні напруги задаються стандартами логіки: для КМОП-логіки з живленням 3,3 В згідно з JEDEC JESD8C вихідний рівень логічної одиниці V_{OH} становить не менше 2,4 В, а вхідний поріг V_{IH} — не більше 2,0 В, що дає запас завадостійкості лише 0,4 В. Цього може бути достатньо для кількох сантиметрів доріжки на платі, однак за межами плати будь-яка наведена завада амплітудою понад 0,4 В спричинить хибне спрацьовування. Саме тому для зв'язку на відстані понад кілька десятків сантиметрів застосовують диференційні інтерфейси — RS-485, CAN, USB, Ethernet, — де інформація передається різницею напруг між двома проводами витої пари. Завада, що наводиться на обидва проводи синфазно, віднімається диференційним приймачем і практично не впливає на результат. Кількісною мірою цієї здатності є коефіцієнт ослаблення синфазної завади (CMRR, Common-Mode Rejection Ratio), який для сучасних диференційних

приймачів перевищує 60 дБ, тобто синфазна завада послаблюється більш ніж у тисячу разів. Стандарт TIA/EIA-485-A вимагає, щоб приймач розпізнавав диференційний сигнал амплітудою всього 200 мВ, тоді як синфазна напруга може сягати ±7 В відносно землі приймача, що забезпечує надійний зв'язок у промислових цехах із потужними електромагнітними полями.

Цілісність сигналу (signal integrity) — ширше поняття, що охоплює не лише рівень завад, а й спотворення форми сигналу, спричинені властивостями самої лінії передавання. Зі збільшенням частоти або довжини лінії провідник перестає поводитися як звичайний електричний дріт і починає проявляти властивості довгої лінії з розподіленими параметрами: погонною індуктивністю, ємністю та активним опором. Фронти імпульсів деградують через обмежену смугу пропускання кабелю (міжсимвольна інтерференція, ISI), різниця в часі приходу фронтів тактового сигналу та даних (clock skew) призводить до помилок вибірки, а неузгодженість хвильового опору джерела, лінії та навантаження породжує відбиття, які накладаються на основний сигнал і створюють хибні фронти або плато. Для кількісної оцінки цих явищ у високошвидкісних інтерфейсах використовують очну діаграму (eye diagram) — суперпозицію багатьох бітових інтервалів, отриману на осцилографі в режимі нескінченної післясвітності. Розкрив очної діаграми по вертикалі (eye height) показує запас за рівнем із урахуванням шумів і міжсимвольної інтерференції, а розкрив по горизонталі (eye width) — часовий інтервал, упродовж якого сигнал стабільний і може бути коректно зафіксований приймачем. Чим довша лінія або вища швидкість, тим більше «закривається» око, і за певної межі приймач втрачає здатність відновлювати біти з прийнятним рівнем помилок. Для дротових інтерфейсів виробники мікросхем фізичного рівня, як-от Texas Instruments (SN65HVD255 для CAN) або Maxim Integrated (MAX3485 для RS-485), у своїх даташитах наводять гарантовані очні діаграми для різних довжин кабелю та швидкостей, що є основним джерелом інформації під час аналізу цілісності сигналу. Методику моделювання цілісності сигналу для друкованих плат детально викладено в документі AN5073 «High-speed layout guidelines for STM32H7 series» від STMicroelectronics, де рекомендовано виконувати симуляцію в IBIS-моделях для всіх ліній із частотою понад 30 МГц.

Максимальна довжина лінії є інтегральним показником, що випливає з компромісу між завадостійкістю, цілісністю сигналу та швидкістю передавання. Для кожного інтерфейсу існує фундаментальне обмеження, яке можна виразити через критерій довгої лінії: якщо час поширення сигналу в лінії t_{prop} перевищує половину тривалості фронту t_{rise}, лінія вважається еле-

ктрично довгою і потребує узгодження хвильового опору. Оскільки швидкість поширення сигналу в мідному провіднику з типовим діелектриком FR-4 становить близько 15 см/нс, для інтерфейсу SPI на частоті 50 МГц (період 20 нс, тривалість фронту ~2 нс) критична довжина, після якої починаються відбиття, становить лише близько 15 см. Для UART на швидкості 115200 біт/с (тривалість біта ~8,7 мкс, фронти ~100 нс) лінія може сягати кількох метрів без узгодження, однак ємність кабелю почне обмежувати швидкість наростання фронту раніше. Диференційні інтерфейси завдяки безперервному контролю хвильового опору (120 Ом для CAN, 100 Ом для Ethernet) забезпечують передбачувану поведінку на значно більших відстанях. Кількісні значення максимальної довжини для найпоширеніших інтерфейсів згідно з відповідними стандартами наведено в таблиці 2.3.

Таблиця 2.3 – Завадостійкість та максимальна довжина лінії для поширених дротових інтерфейсів

Інтерфейс	Характеристика	Значення
UART TTL (3,3 В)	*Тип сигналу*	несиметричний
	Мінімальний диференційний поріг (мВ) / запас завадостійкості	$V_{IH} - V_{IL} \approx 1{,}0$ В (запас ~0,4 В)
	Вимоги до кабелю (хвильовий опір)	не регламентовано
	Максимальна довжина на максимальній швидкості	0,5 м (на 10 Мбод)
	Максимальна довжина на низькій швидкості	5 м (на 9600 біт/с)
	Нормативний документ	JEDEC JESD8C
UART RS-232	*Тип сигналу*	несиметричний (однополярний з двополярним виходом)
	Мінімальний диференційний поріг (мВ) / запас завадостійкості	±3 В мін. рівень передавача, запас перешкодозахисту ~2 В
	Вимоги до кабелю (хвильовий опір)	не регламентовано (обмеження ємності навантаження 2500 пФ)
	Максимальна довжина на максимальній швидкості	~2 м (на 115200 біт/с)
	Максимальна довжина на низькій швидкості	15 м (на 20 кбіт/с)
	Нормативний документ	TIA/EIA-232-F

Продовження таблиці 2.3

Інтерфейс	Характеристика	Значення
RS-485	*Тип сигналу*	диференційний
	Мінімальний диференційний поріг (мВ) / запас завадостійкості	±200 мВ (CMRR >60 дБ)
	Вимоги до кабелю (хвильовий опір)	вита пара 120 Ом
	Максимальна довжина на максимальній швидкості	12 м (10 Мбіт/с)
	Максимальна довжина на низькій швидкості	1200 м (100 кбіт/с)
	Нормативний документ	TIA/EIA-485-A
CAN 2.0 / CAN FD	*Тип сигналу*	диференційний
	Мінімальний диференційний поріг (мВ) / запас завадостійкості	±500 мВ (домінанта), CMRR >50 дБ
	Вимоги до кабелю (хвильовий опір)	вита пара 120 Ом
	Максимальна довжина на максимальній швидкості	40 м (1 Мбіт/с CAN 2.0) / ~10 м (8 Мбіт/с CAN FD)
	Максимальна довжина на низькій швидкості	500 м (125 кбіт/с)
	Нормативний документ	ISO 11898-2
I²C (Fast-mode, 400 кГц)	*Тип сигналу*	несиметричний, відкритий стік
	Мінімальний диференційний поріг (мВ) / запас завадостійкості	запасу немає (пороги фіксовані 0,3·VDD / 0,7·VDD)
	Вимоги до кабелю (хвильовий опір)	не регламентовано, обмеження за ємністю 400 пФ
	Максимальна довжина на максимальній швидкості	~2 м (за малої ємності кабелю)
	Максимальна довжина на низькій швидкості	~10 м (спеціальні буфери)
	Нормативний документ	NXP UM10204

Продовження таблиці 2.3

Інтерфейс	Характеристика	Значення
SPI (50 МГц, стандартний)	*Тип сигналу*	несиметричний (CMOS/TTL)
	Мінімальний диференційний поріг (мВ) / запас завадостійкості	$V_{IH} - V_{IL} \approx 1{,}3$ В для 3,3 В (високий рівень)/ 0,8 В (низький)
	Вимоги до кабелю (хвильовий опір)	не регламентовано, рекомендовано контрольований імпеданс 50 Ом для високих швидкостей
	Максимальна довжина на максимальній швидкості	<10 см (на 50 МГц, без узгодження)
	Максимальна довжина на низькій швидкості	~1 м (на 1 МГц)
	Нормативний документ	JEDEC JESD8C (рівні), SPI де-факто
USB 2.0 High Speed	*Тип сигналу*	диференційний
	Мінімальний диференційний поріг (мВ) / запас завадостійкості	±200 мВ (за специфікацією чутливість приймача)
	Вимоги до кабелю (хвильовий опір)	екранована вита пара 90 Ом ±15%
	Максимальна довжина на максимальній швидкості	5 м (пасивний кабель)
	Максимальна довжина на низькій швидкості	5 м (швидкість не впливає)
	Нормативний документ	USB 2.0 Specification
100BASE-TX Ethernet	*Тип сигналу*	диференційний (MLT-3)
	Мінімальний диференційний поріг (мВ) / запас завадостійкості	±500 мВ (рівень сигналу), значний запас за рахунок MLT-3 кодування
	Вимоги до кабелю (хвильовий опір)	вита пара категорії 5e, 100 Ом
	Максимальна довжина на максимальній швидкості	100 м
	Максимальна довжина на низькій швидкості	100 м
	Нормативний документ	IEEE 802.3, TIA/EIA-568

Кінець таблиці 2.3

Інтерфейс	Характеристика	Значення
1000BASE-T Ethernet	*Тип сигналу*	диференційний (PAM-5)
	Мінімальний диференційний поріг (мВ) / запас завадостійкості	±1 В (рівень сигналу), захист забезпечується цифровою обробкою (DSP) та ехокомпенсацією
	Вимоги до кабеля (хвильовий опір)	вита пара категорії 5e/6, 100 Ом
	Максимальна довжина на максимальній швидкості	100 м
	Максимальна довжина на низькій швидкості	100 м
	Нормативний документ	IEEE 802.3ab, TIA/EIA-568

Окремої уваги потребує явище відбиттів, яке є чи не найпоширенішою причиною нестабільної роботи дротових інтерфейсів у навчальних лабораторіях. Коли сигнал доходить до кінця лінії, яка не навантажена на узгоджений резистор, енергія імпульсу не поглинається повністю, а частково відбивається назад до джерела. Накладаючись на основний сигнал, відбитий імпульс спотворює фронти, створює додаткові перепади, які можуть бути хибно інтерпретовані як тактові імпульси або біти даних. Коефіцієнт відбиття Γ визначається через відношення різниці хвильових опорів до їх суми (2.4):

$$\Gamma = (Z_L - Z_0)/(Z_L + Z_0) \ , \qquad (2.4)$$

де Z_0 — хвильовий опір лінії, Z_L — опір навантаження. За ідеального узгодження $Z_L = Z_0$ коефіцієнт Γ дорівнює нулю, і відбиття відсутні. Для шини CAN стандарт ISO 11898-2 прямо вимагає встановлення термінуючих резисторів 120 Ом на обох кінцях шини; їхня відсутність — одна з найтиповіших помилок студентів, що призводить до повної втрати зв’язку на швидкостях понад 125 кбіт/с. У шині RS-485 термінування також є обов’язковим для швидкостей понад 200 кбіт/с або довжин понад 50 м, причому резистори мають бути розташовані саме на найвіддаленіших кінцях шини, а не на кожному вузлі. Для SPI та UART окремого термінування стандартами не передбачено, однак послідовний резистор 22–100 Ом на виході передавача часто застосовують для обмеження струму перезаряджання ємності лінії та придушення дзвону.

На рисунку 2.3 (а, б) показано спрощену ілюстрацію впливу електричної довжини лінії на форму сигналу на прикладі SPI з частотою 20 МГц.

Порівнюються дві ситуації: коротка лінія (5 см) без узгодження, де сигнал має чисті фронти і стабільний рівень, та довга лінія (30 см) без терміну-вання, де спостерігаються значні відбиття, дзвін і звуження очної діаграми. Такі осцилограми легко отримати в навчальній лабораторії за допомогою цифрового осцилографа зі смугою пропускання не менше 100 МГц, і вони слугують наочним підтвердженням того, що на високих частотах навіть друкована плата перестає бути прозорим середовищем.

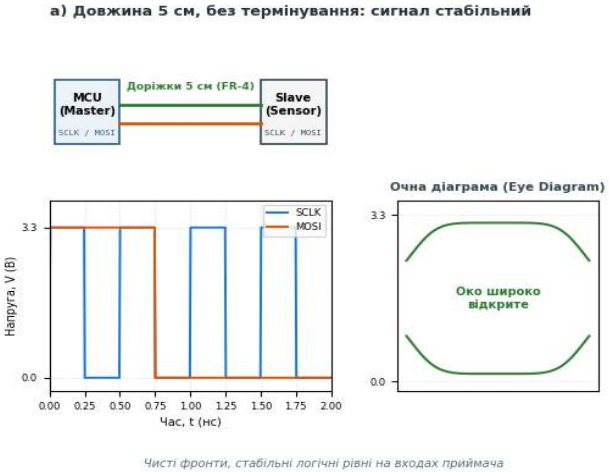


Рисунок 2.3 – Вплив довжини лінії на цілісність сигналу SPI (20 МГц): а) довжина 5 см; б) довжина 30 см

Для бездротових інтерфейсів аналогом завадостійкості та цілісності сигналу виступають показники бітової помилки (BER, Bit Error Rate), чутливості приймача та завадостійкості до сусідніх каналів (adjacent channel rejection). Специфікація Bluetooth Core 5.4 вимагає, щоб приймач забезпечував BER не гірше 0,1% за рівня сигналу -70 дБм, а більшість сучасних трансиверів, як-от Nordic nRF52840, досягають чутливості -103 дБм для BLE 1M PHY, що відповідає запасу бюджету каналу понад 100 дБ порівняно з максимально дозволеною потужністю передавача +20 дБм. Для LoRa чутливість сягає -148 дБм на SF12, що дозволяє приймати сигнал, потужність якого в 40 разів нижча за рівень теплового шуму в смузі каналу, завдяки кореляційному прийманню CSS-символів. Таким чином, аналіз завадостійкості для бездротових інтерфейсів зводиться до розрахунку бюджету каналу (link budget): суми вихідної потужності передавача, підсилень антен та чутливості приймача, із якої віднімаються втрати на поширення в середовищі, що розраховуються за моделлю Окамура-Хата, COST-231 або спрощеною формулою втрат у вільному просторі.

Практичний висновок із викладеного полягає в тому, що вибір інтерфейсу за критерієм завадостійкості є багатокроковою процедурою, яка включає: визначення рівня електромагнітних завад у цільовому середовищі (промисловий цех, автомобіль, житлове приміщення, відкрита місцевість); оцінку необхідної довжини лінії або дальності радіозв'язку; вибір між несиметричним і диференційним фізичним рівнем; розрахунок або моделювання очної діаграми для найгіршого випадку; передбачення засобів захисту — термінуючих резисторів, обмежувальних діодів, TVS-супресорів, екранування кабелю. Для студента важливо засвоїти, що осцилограф є не менш важливим інструментом аналізу інтерфейсів, ніж компілятор, і що проблеми, які неможливо пояснити на рівні коду, часто мають суто фізичну природу — невдало розведену доріжку, відсутній термінатор або занадто довгий кабель. Наступний підрозділ завершує огляд ключових технічних характеристик аналізом детермінованості та латентності — параметрів, які стають критичними в системах реального часу.

2.1.4. Детермінованість, латентність та реальний час

Пропускна здатність, енергоспоживання та завадостійкість, за всієї їхньої важливості, не дають відповіді на ще одне критичне запитання: чи надійде повідомлення вчасно? Для системи, що керує електроприводом промислового робота, антиблокувальною системою автомобіля або хірургічним інструментом із тактильним зворотним зв'язком, затримка навіть в

одну мілісекунду понад допустимий ліміт є не просто деградацією продуктивності, а функціональною відмовою, здатною призвести до аварії чи загрози життю. Саме тому четвертою ключовою групою характеристик інтерфейсу є його поведінка в часовій області, яку описують трьома тісно пов'язаними поняттями: детермінованість, латентність та придатність до систем реального часу.

Детермінованість у контексті цифрових комунікацій означає властивість інтерфейсу гарантувати, що час доставки повідомлення від відправника до отримувача ніколи не перевищить певного, заздалегідь відомого значення, незважаючи на конкуренцію за середовище, фонове навантаження мережі або повторні передачі після помилок. Детермінований інтерфейс характеризується не середньою швидкістю, а максимальним часом виконання (WCET, Worst-Case Execution Time) комунікаційної транзакції. Принципова різниця між «швидким» і «детермінованим» є однією з найважливіших концепцій, які має засвоїти студент: Ethernet зі швидкістю 1 Гбіт/с може мати середню латентність у десятки мікросекунд, однак у моменти перевантаження комутатора затримка може сягати сотень мілісекунд через переповнення черг, що робить стандартний Ethernet непридатним для замкнених контурів керування. Натомість шина CAN зі швидкістю всього 1 Мбіт/с гарантує, що повідомлення з найвищим пріоритетом буде доставлене не пізніше ніж через 130 мкс завдяки механізму побітового арбітражу, який не допускає втрати часу на колізії.

Джерела недетермінованості в інтерфейсах різноманітні. По-перше, це колізії та повторні передачі: у мережах із випадковим доступом до середовища (CSMA/CA у Wi-Fi, класичний Ethernet з концентраторами) час очікування перед повторною спробою є випадковою величиною, що унеможливлює гарантовану верхню межу. По-друге, це програмні черги та переривання: навіть якщо апаратний інтерфейс здатен передати пакет за фіксований час, операційна система реального часу може затримати обробку через пріоритети завдань, витіснення або блокування на семафорах. По-третє, це буферизація в комутаторах і маршрутизаторах, де пакети можуть накопичуватися в чергах, особливо за наявності пріоритезації трафіку. Інтерфейси, спроєктовані спеціально для систем реального часу, усувають або жорстко обмежують ці джерела: CAN використовує детермінований арбітраж на фізичному рівні, EtherCAT обробляє кадри «на льоту» в кожному вузлі без буферизації, а Time-Sensitive Networking (TSN) запроваджує планування трафіку на основі глобального часу згідно з IEEE 802.1Qbv.

Латентність (затримка) — це сумарний час від моменту, коли відправник ініціює передавання повідомлення, до моменту, коли отримувач повністю отримує це повідомлення і воно стає доступним для прикладної програми. У вбудованих системах латентність складається з кількох компонентів, які описуються формулою (2.5):

$$t_{\text{лат}} = t_{\text{підг}} + t_{\text{черга}} + t_{\text{передача}} + t_{\text{пошир}} + t_{\text{обробка}} + t_{\text{відповідь}} \ , \quad (2.5)$$

де $t_{\text{підг}}$ — час підготовки повідомлення прикладною програмою (копіювання в буфер, обчислення CRC); $t_{\text{черга}}$ — час очікування в черзі на вивільнення середовища; $t_{\text{передача}}$ — час безпосереднього виштовхування бітів у канал, що дорівнює довжині кадру, поділеній на бітову швидкість; $t_{\text{пошир}}$ — час поширення сигналу через фізичне середовище (приблизно 5 нс/м для мідного кабелю, 3,3 мкс/км для радіосигналу); $t_{\text{обробка}}$ — час, необхідний апаратному контролеру приймача для перевірки цілісності кадру та генерації переривання; $t_{\text{відповідь}}$ — час реакції програмного забезпечення (латентність переривання, диспетчеризація завдань). У дротових інтерфейсах із короткими лініями домінуючими є $t_{\text{передача}}$ та $t_{\text{відповідь}}$; у бездротових глобальних мережах — $t_{\text{передача}}$ (через низьку бітову швидкість) і $t_{\text{пошир}}$ (через великі відстані). Для прикладу, передача 8-байтового повідомлення через UART на швидкості 115200 біт/с займає близько 870 мкс лише на $t_{\text{передача}}$ (10 бітів/байт × 8 байтів / 115200 біт/с), тоді як через SPI на частоті 10 МГц — лише 6,4 мкс. У мережі LoRaWAN зі швидкістю SF12 (близько 300 біт/с) та сама передача триватиме понад 200 мс, що є абсолютно неприйнятним для керування, але цілком допустимим для щогодинної телеметрії.

Для систем реального часу, які поділяються на жорсткі (hard real-time, де порушення часового ліміту є катастрофічною відмовою), м'які (soft real-time, де порушення знижує якість, але не спричиняє відмови) та тверді (firm real-time, де запізнілі дані просто відкидаються), вибір інтерфейсу визначається виключно його здатністю гарантувати максимальну латентність із заданою ймовірністю. Для жорсткого реального часу застосовують виключно детерміновані інтерфейси — CAN/CAN FD, EtherCAT, Profinet IRT, TSN у конфігурації з плануванням, — тоді як для м'якого реального часу припустимі BLE, Wi-Fi із пріоритезацією (WMM) або навіть стандартний TCP/IP за умови недовантаження мережі. У таблиці 2.4 наведено порівняння характеристик детермінованості та типової латентності для інтерфейсів, що найчастіше застосовуються в системах реального часу.

Таблиця 2.4 – Детермінованість і латентність інтерфейсів для систем реального часу

Інтерфейс	Характеристика	Значення
UART (точка-точка)	*Механізм доступу*	Виділений канал без колізій (повний дуплекс)
	Детермінованість	Повністю детермінований (фіксована затримка на символ)
	Типова мінімальна латентність	~86 мкс (1 байт на 115200 біт/с)
	Максимальна гарантована латентність (за наявності)	Гарантована відсутність затримок понад розрахункову (при використанні DMA / переривань)
	Придатність до жорсткого реального часу	Так
	Нормативний документ	TIA/EIA-232-F
CAN 2.0 (1 Мбіт/с)	*Механізм доступу*	Побітовий арбітраж за пріоритетом
	Детермінованість	Повністю детермінований (WCET розраховується)
	Типова мінімальна латентність	~100 мкс (8 байтів, 1 Мбіт/с)
	Максимальна гарантована латентність (за наявності)	<130 мкс для найвищого пріоритету
	Придатність до жорсткого реального часу	Так
	Нормативний документ	ISO 11898-1
CAN FD (5 Мбіт/с)	*Механізм доступу*	Побітовий арбітраж за пріоритетом (арбітраж на 1 Мбіт/с, дані на 5 Мбіт/с)
	Детермінованість	Повністю детермінований (WCET з урахуванням фаз швидкості)
	Типова мінімальна латентність	~50 мкс (8 байтів, 5 Мбіт/с фаза даних)
	Максимальна гарантована латентність (за наявності)	<70 мкс для найвищого пріоритету (8 байт)
	Придатність до жорсткого реального часу	Так
	Нормативний документ	ISO 11898-1:2015
Ethernet (стандартний, без TSN)	*Механізм доступу*	CSMA/CD (історично) або комутація з чергами
	Детермінованість	Недетермінований (випадкові затримки в чергах)
	Типова мінімальна латентність	~100 мкс (без навантаження)
	Максимальна гарантована латентність (за наявності)	Не гарантується
	Придатність до жорсткого реального часу	Ні
	Нормативний документ	IEEE 802.3

Кінець таблиці 2.4

Інтерфейс	Характеристика	Значення
Ethernet TSN (IEEE 802.1Qbv)	*Механізм доступу*	TDMA (Time-Aware Shaper) + PTP (802.1AS)
	Детермінованість	Детермінований у межах запланованих вікон
	Типова мінімальна латентність	~100 мкс (1 GbE)
	Максимальна гарантована латентність (за наявності)	Залежить від плану, зазвичай <1 мс
	Придатність до жорсткого реального часу	Так
	Нормативний документ	IEEE 802.1Qbv, 802.1AS
EtherCAT	*Механізм доступу*	Master-slave, сумований кадр (on-the-fly processing)
	Детермінованість	Повністю детермінований
	Типова мінімальна латентність	<1 мкс на вузол (апаратна обробка)
	Максимальна гарантована латентність (за наявності)	<100 мкс для циклу з 100 вузлів (типово 50–100 мкс)
	Придатність до жорсткого реального часу	Так
	Нормативний документ	IEC 61158 Type 12, EtherCAT Technology Group
BLE 5.4 (з'єднання, 7,5 мс інтервал)	*Механізм доступу*	Планове опитування (connection events)
	Детермінованість	Частково детермінований (залежить від інтервалу з'єднання)
	Типова мінімальна латентність	~7,5 мс (мінімальний connection interval)
	Максимальна гарантована латентність (за наявності)	~30 мс (на практиці з ретрансляціями)
	Придатність до жорсткого реального часу	Hi (soft real-time)
	Нормативний документ	Bluetooth Core 5.4
LoRaWAN (клас A)	*Механізм доступу*	ALOHA (випадковий доступ)
	Детермінованість	Недетермінований (випадковий доступ + далекість)
	Типова мінімальна латентність	~200 мс (тільки передача, SF12)
	Максимальна гарантована латентність (за наявності)	Не гарантується (години в разі колізій)
	Придатність до жорсткого реального часу	Hi
	Нормативний документ	LoRaWAN Specification 1.1

Для наочного протиставлення детермінованого та недетермінованого інтерфейсів на рисунку 2.4 наведено часові діаграми обміну для шини CAN і для стандартного Ethernet у ситуації конкуренції за середовище. У

випадку CAN три вузли намагаються передати повідомлення одночасно, однак завдяки побітовому арбітражу вузол із найвищим пріоритетом (найменшим ідентифікатором) негайно отримує доступ до шини, а решта відступають і повторюють спробу після завершення поточного кадру. Загальна затримка для найпріоритетнішого повідомлення є сталою і не залежить від кількості конкурентів. У випадку Ethernet комутатор, отримавши одночасно три кадри, буферизує їх і обслуговує в порядку черги, що створює непередбачувану додаткову затримку для двох із трьох відправників. Якщо ж до комутатора надходить потік даних, що перевищує пропускну здатність вихідного порту, черга зростає необмежено аж до втрати кадрів.

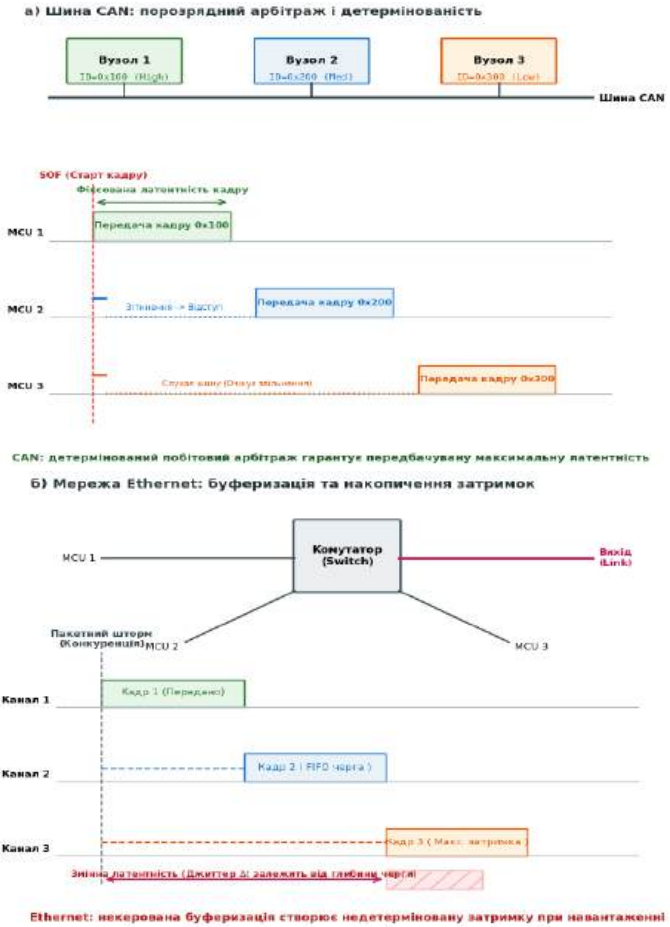


Рисунок 2.4 – Порівняння детермінованості CAN та стандартного Ethernet за конкуренції

Практичний аналіз інтерфейсу на придатність до задач реального часу починають із визначення вимог системи: яка максимально допустима наскрізна затримка (end-to-end latency) від сенсора до актуатора; чи є в системі події, що вимагають негайної реакції (наприклад, сигнал аварійної зупинки); який максимальний джиттер (варіативність латентності) дозволено. Далі обирають інтерфейс, що забезпечує гарантовану верхню межу, і виконують розрахунок WCET для всіх ланок комунікаційного ланцюга. Для CAN цей розрахунок добре формалізований: час передачі кадру обчислюється як функція від довжини даних і швидкості, а найгірший час очікування для повідомлення з пріоритетом m враховує блокування від повідомлень із вищими пріоритетами та максимальну тривалість одного кадру нижчого пріоритету, який міг зайняти шину безпосередньо перед спробою арбітражу. Для Ethernet TSN аналогічний розрахунок потребує моделювання розкладу вікон у Time-Aware Shaper згідно з IEEE 802.1Qbv. Якщо жоден із доступних інтерфейсів не забезпечує потрібної детермінованості, архітектуру системи доводиться змінювати — наприклад, переносити контур керування безпосередньо на мікроконтролер, залишаючи мережі лише функції моніторингу та конфігурування.

На завершення огляду ключових технічних характеристик варто наголосити, що жоден із чотирьох розглянутих критеріїв — пропускна здатність, енергоспоживання, завадостійкість, детермінованість — не є самодостатнім. Реальний вибір інтерфейсу завжди є багатокритеріальною оптимізацією, де покращення одного показника часто досягається ціною погіршення іншого. Підвищення бітової швидкості зменшує час передачі, але збільшує енергоспоживання; застосування диференційної пари підвищує завадостійкість, але вимагає додаткового провідника; впровадження детермінованого планування гарантує латентність, але ускладнює стек протоколу. Мистецтво інженера полягає саме в тому, щоб знайти збалансоване рішення, яке відповідає всім обов'язковим вимогам і водночас не є надлишково складним для конкретної задачі. Наступний підрозділ присвячено саме методиці такого збалансованого вибору, що спирається на нормативну базу та формалізовані критерії.

2.2. Нормативна база: стандарти IEEE, ISO, IEC, ETSI, ITU-T, специфікації USB-IF, Bluetooth SIG, LoRa Alliance

Жоден інтерфейс не існує у вакуумі. За кожним роз'ємом, кожною часовою діаграмою, кожним форматом кадру стоїть нормативний документ, розроблений міжнародною або галузевою організацією, який фіксує

досягнутий консенсус виробників, операторів зв'язку, науковців і регуляторів щодо того, як саме повинен функціонувати даний інтерфейс. Для інженера-розробника вбудованих систем знання цих документів є не менш важливим, ніж володіння мовами програмування чи схемотехнікою, оскільки саме стандарти забезпечують інтероперабельність — здатність пристроїв різних виробників коректно взаємодіяти між собою без додаткових «милиць» і адаптерів. Пристрій, що виходить на ринок без дотримання відповідних стандартів, приречений на ізоляцію, незалежно від його внутрішньої досконалості. З іншого боку, грамотне посилання на стандарт у технічному завданні однозначно визначає вимоги до інтерфейсу без потреби щоразу винаходити власний протокол, що прискорює розроблення, зменшує ризики та полегшує сертифікацію.

У царині інтерфейсів для мікроконтролерних систем та Інтернету речей нормативну базу формують кілька ключових організацій, кожна з яких спеціалізується на певному рівні стеку протоколів або на певній галузі застосування. Інститут інженерів електротехніки та електроніки (IEEE, Institute of Electrical and Electronics Engineers) є найбільшою у світі професійною асоціацією, чий комітет стандартів IEEE-SA (Standards Association) відповідає за переважну більшість стандартів локальних і персональних мереж. Найвідомішим доробком IEEE є родина стандартів IEEE 802, що охоплює Ethernet (IEEE 802.3), Wi-Fi (IEEE 802.11), Bluetooth спільно з Bluetooth SIG (IEEE 802.15.1 історично, хоча сучасні специфікації Bluetooth публікує виключно Bluetooth SIG), а також низькошвидкісні бездротові мережі (IEEE 802.15.4, на якому базуються Zigbee, Thread і 6LoWPAN). Крім того, IEEE розробляє стандарти синхронізації часу (IEEE 1588 Precision Time Protocol), що є фундаментом для TSN, та стандарти для надширокосмугового зв'язку (IEEE 802.15.4z для UWB). Усі ці документи є результатом багаторічної роботи робочих груп за участі інженерів із провідних компаній, проходять кілька раундів рецензування та голосування, після чого публікуються у відкритому доступі (через IEEE Xplore) або можуть бути завантажені безкоштовно після закінчення періоду ембарго. Для розробника посилання на конкретний розділ IEEE 802.11ax або IEEE 802.3cg є незаперечним аргументом у суперечці про сумісність.

Міжнародна організація зі стандартизації (ISO, International Organization for Standardization) спільно з Міжнародною електротехнічною комісією (IEC, International Electrotechnical Commission) опікується стандартами ширшого профілю, що охоплюють не лише електричні, а й механічні, теплові та функціональні аспекти. Для інтерфейсів ключове значення мають стандарти ISO 11898 (CAN), ISO 17987 (LIN), ISO 14229 (UDS —

Unified Diagnostic Services для автомобільної діагностики), а також IEC 61158 (серія стандартів на промислові польові шини, включно з EtherCAT, Profinet, SERCOS). Особливістю стандартів ISO/IEC є їхній статус міждержавних документів, що приймаються національними органами зі стандартизації (в Україні — ДП «УкрНДНЦ») як державні стандарти, часто методом «обкладинки» з перекладом. Таким чином, посилаючись на ISO 11898-1:2015, інженер апелює до документа, визнаного на території України офіційно. Серія стандартів ISO/IEC 7498 (еталонна модель OSI) є концептуальним фундаментом, на якому побудовано всі сучасні стеки протоколів, а ISO/IEC 27000 — основою для оцінки безпеки інтерфейсів.

Європейський інститут телекомунікаційних стандартів (ETSI, European Telecommunications Standards Institute) спеціалізується на стандартах для телекомунікацій і радіозв'язку, причому його документи мають особливу вагу для пристроїв, що експлуатуються в Європі, оскільки саме на ETSI посилається європейське законодавство в частині використання радіочастотного спектра. Для розробника IoT-пристрою з бездротовим інтерфейсом критичними є стандарти ETSI EN 300 220 (пристрої малого радіусу дії, включно з LoRa), ETSI EN 300 328 (широкосмугові пристрої діапазону 2,4 ГГц, зокрема Wi-Fi та Bluetooth), а також ETSI EN 301 908 (стільникові мережі 3G/4G/5G). Сертифікація на відповідність цим стандартам (маркування CE) є обов'язковою умовою для продажу пристрою на європейському ринку, тому знайомство з ними необхідне не лише розробникові, а й менеджерові продукту.

Сектор стандартизації Міжнародного союзу електрозв'язку (ITU-T, International Telecommunication Union — Telecommunication Standardization Sector) розробляє стандарти для глобальних телекомунікаційних мереж, зокрема для оптичних транспортних мереж (OTN), оптоволоконних кабелів (серія G.652–G.657) та протоколів вищих рівнів. У контексті IoT важливим є внесок ITU-T у стандартизацію 5G (спільно з 3GPP) та концепції тактильного Інтернету (серія Y.4000). Крім того, саме ITU-T координує розподіл радіочастотного спектра на міжнародному рівні через Регламент радіозв'язку, що безпосередньо впливає на доступні частоти для бездротових інтерфейсів у різних регіонах світу.

Окрему та надзвичайно впливову категорію становлять галузеві альянси та консорціуми, які не є офіційними органами стандартизації в розумінні ISO, проте де-факто визначають специфікації, що стають загальновизнаними. USB Implementers Forum (USB-IF) публікує специфікації універсальної послідовної шини — від USB 1.1 до USB4 версії 2.0 — а також специфікації USB Type-C, Power Delivery та відповідних класів пристроїв.

Членство в USB-IF і проходження сертифікації дає виробникові право використовувати логотип USB, що є обов'язковою вимогою для великих OEM-замовників. Bluetooth Special Interest Group (Bluetooth SIG) опікується специфікаціями Bluetooth — від класичної BR/EDR до Bluetooth LE Audio та Channel Sounding у Core 6.0 — і веде реєстр ідентифікаторів компаній (Company ID), необхідних для присвоєння адрес пристроїв. LoRa Alliance публікує специфікації LoRaWAN (версії 1.0, 1.1, 1.2) і керує програмою сертифікації пристроїв, що гарантує їхню сумісність із мережевими серверами різних виробників. Подібну роль для стільникового IoT відіграє 3GPP (3rd Generation Partnership Project), чиї релізи (Release 13–18) визначають параметри NB-IoT, LTE-M і 5G RedCap, та організація OPC Foundation, що публікує специфікації OPC UA для промислового IoT.

Узагальнена інформація про розподіл відповідальності між основними організаціями стандартизації стосовно найпоширеніших інтерфейсів вбудованих систем наведена в таблиці 2.5. Ця таблиця допоможе читачеві швидко знайти потрібний нормативний документ за назвою інтерфейсу або рівнем стеку.

Таблиця 2.5 – Основні організації стандартизації та їхня роль у нормативному забезпеченні інтерфейсів

Організація	Характеристика	Значення
IEEE	*Сфера відповідальності*	Локальні та персональні мережі, синхронізація
	Ключові стандарти / специфікації	IEEE 802.3, 802.11, 802.15.4, 802.15.4z, 802.1Q, 802.1AS, 1588
	Інтерфейси, що регулюються	Ethernet, Wi-Fi, Zigbee/Thread (PHY+MAC), UWB, TSN
	Рівень стеку	Фізичний, канальний, частково мережевий
	Доступність документів	IEEE Xplore (частково безкоштовно після ембарго)
ISO/IEC JTC 1	*Сфера відповідальності*	Спільна розробка IT-стандартів, модель OSI, мережі
	Ключові стандарти / специфікації	ISO 11898, ISO 17987, IEC 61158, ISO/IEC 7498, ISO 14229
	Інтерфейси, що регулюються	CAN, LIN, EtherCAT, Profinet
	Рівень стеку	Канальний, прикладний
	Доступність документів	Платні, доступ через національні органи

Продовження таблиці 2.5

Організація	Характеристика	Значення
IEC (окремо)	*Сфера відповідальності*	Промислові мережі, електромагнітна сумісність, функційна безпека
	Ключові стандарти / специфікації	IEC 61158 (Fieldbus), IEC 61784 (профілі), IEC 61000 (EMC), IEC 61508 (безпека)
	Інтерфейси, що регулюються	Profibus, Foundation Fieldbus, SERCOS, EMC вимоги до пристроїв
	Рівень стеку	Фізичний, канальний, прикладний (профілі)
	Доступність документів	Платні, через національні комітети
ETSI	*Сфера відповідальності*	Радіозв'язок, електромагнітна сумісність
	Ключові стандарти специфікації	EN 300 220, EN 300 328, EN 301 908
	Інтерфейси, що регулюються	LoRa (радіо), BLE (радіо), NB-IoT (частково)
	Рівень стеку	Фізичний
	Доступність документів	Безкоштовно на сайті ETSI
ITU-T	*Сфера відповідальності*	Телекомунікаційні мережі, оптичні транспортні системи
	Ключові стандарти / специфікації	ITU-T G.694.1 (DWDM), G.984 (GPON), G.709 (OTN), Y.2060 (IoT)
	Інтерфейси, що регулюються	GPON, DWDM, xDSL
	Рівень стеку	Фізичний, канальний, мережевий (транспортний)
	Доступність документів	Багато безкоштовно після реєстрації, частина платна
USB-IF	*Сфера відповідальності*	Універсальна послідовна шина
	Ключові стандарти / специфікації	USB 2.0, USB 3.2, USB4, USB Type-C, USB PD
	Інтерфейси, що регулюються	USB у всіх режимах
	Рівень стеку	Фізичний, канальний, протокольний
	Доступність документів	Безкоштовно на usb.org

Кінець таблиці 2.5

Організація	Характеристика	Значення
Bluetooth SIG	*Сфера відповідальності*	Бездротовий персональний зв'язок
	Ключові стандарти / специфікації	Bluetooth Core 5.4 / 6.0, Mesh Profile, GATT Specifications
	Інтерфейси, що регулюються	BLE, Bluetooth Mesh
	Рівень стеку	Фізичний, канальний, прикладний
	Доступність документів	Безкоштовно на bluetooth.com
LoRa Alliance	*Сфера відповідальності*	Глобальні LPWAN
	Ключові стандарти / специфікації	LoRaWAN 1.1, Regional Parameters
	Інтерфейси, що регулюються	LoRaWAN
	Рівень стеку	Канальний (MAC), частково прикладний
	Доступність документів	Безкоштовно для членів, частково відкриті
3GPP	*Сфера відповідальності*	Мобільний широкосмуговий зв'язок та масштабний IoT
	Ключові стандарти / специфікації	TS 36.101 (LTE), TS 38.101 (NR), TS 23.401, TS 23.501
	Інтерфейси, що регулюються	LTE-M, NB-IoT, 5G NR
	Рівень стеку	Повний стек (фізичний – прикладний)
	Доступність документів	Безкоштовно на www.3gpp.org
OPC Foundation	*Сфера відповідальності*	Промислова інтероперабельність, обмін даними між пристроями та системами
	Ключові стандарти / специфікації	OPC UA (IEC 62541), OPC UA FX
	Інтерфейси, що регулюються	OPC UA (прикладний протокол поверх TCP/IP)
	Рівень стеку	Прикладний (над транспортним рівнем)
	Доступність документів	Специфікації відкриті, безкоштовні для членів; базові документи у вільному доступі

Окремої згадки потребує ієрархічний характер нормативної бази. Стандарти вищого рівня (наприклад, IEC 61158 для промислових мереж)

часто посилаються на стандарти нижчого рівня (ISO 11898 для CAN, IEEE 802.3 для Ethernet) як на транспортний механізм. Це утворює багатошарову систему залежностей, яку інженер має враховувати під час аналізу сумісності: пристрій, сертифікований за LoRaWAN, автоматично повинен відповідати вимогам ETSI до радіочастотного випромінювання, а пристрій із USB має відповідати не лише специфікації USB-IF, а й нормам електромагнітної сумісності IEC 61000. На рисунку 2.5 цю ієрархію зображено схематично у вигляді піраміди, де на вершині розташовано загальні системні стандарти (ISO/IEC 7498 OSI, IEC 61508 функціональна безпека), на середньому рівні — стандарти на конкретні інтерфейси (ISO 11898, IEEE 802.11), а в основі — стандарти на фізичні компоненти (IEC 61169 для роз'ємів, IEC 60603 для з'єднувачів, JEDEC JESD8 для рівнів логіки).

Важливою практичною навичкою для студента є вміння знаходити, читати та правильно цитувати нормативні документи. Більшість із них мають усталену внутрішню структуру: вступ, область застосування (scope), нормативні посилання, терміни та визначення, основний технічний зміст (часто поділений на розділи, що відповідають рівням моделі OSI) та додатки. Наприклад, стандарт IEEE 802.3-2022 має обсяг понад 7000 сторінок, і було б наївно очікувати, що інженер прочитає його повністю; натомість фахівець звертається до конкретного розділу (Clause), який стосується потрібної швидкості або типу фізичного середовища. Специфікації альянсів часто є більш компактними: документ USB 2.0 Specification налічує близько 650 сторінок, а LoRaWAN 1.1 Specification — близько 100 сторінок. У посібнику в розділах, присвячених конкретним інтерфейсам, наводяться точні посилання на відповідні пункти цих документів, що дозволить студентові виробити звичку перевіряти інформацію за першоджерелом, а не за вторинними компіляціями.

Нормативна база не є застиглою; вона постійно еволюціонує, відображаючи прогрес технологій та потреби ринку. Станом на 2026 рік активними є процеси фіналізації IEEE 802.11be (Wi-Fi 7), розширення TSN-профілів для Industry 4.0 у межах IEEE 802.1, впровадження CAN XL у ISO 11898-1, розвитку Matter як універсального прикладного протоколу під егідою Connectivity Standards Alliance (CSA), а також стандартизації супутникового IoT у 3GPP Release 18 (NTN).





Рисунок 2.5 – Ієрархія нормативної бази інтерфейсів

Для розробника це означає, що знання, отримані сьогодні, потребуватимуть оновлення вже через кілька років, і саме тому посібник робить акцент не на запам'ятовуванні конкретних цифр, а на методології роботи з нормативними документами, що дозволяє самостійно відстежувати зміни та адаптуватися до нових вимог. У цьому контексті таблиця 2.5 слугуватиме читачеві відправною точкою для пошуку актуальних версій стандартів упродовж усієї його професійної кар'єри.

2.3. Критерії вибору для IoT-пристрою: швидкодія vs автономність, вартість vs функціональність, безпека vs простота

Володіння технічними характеристиками десятків інтерфейсів і знання відповідних стандартів становлять необхідний, але недостатній фундамент для прийняття конкретного інженерного рішення. Справжня майстерність розробника виявляється тоді, коли вимоги до виробу є суперечливими за своєю природою, а компроміс між ними неможливо знайти без глибокого розуміння пріоритетів системи. Для Інтернету речей така ситуація є радше правилом, ніж винятком. Пристрій має бути достатньо швидким, щоб вчасно передати дані, і водночас достатньо економним, щоб пропрацювати роки від батареї; він має бути функціонально насиченим, але не виходити за жорсткі рамки собівартості; він мусить бути захищеним від кібератак, але не перетворюватися на лабіринт із криптографічних процедур, недоступний для супроводу. У цьому підрозділі розглядаються три пари таких антагоністичних критеріїв і пропонується формалізований підхід до їх збалансування.

Фундаментальним протиріччям, що пронизує всю індустрію бездротового IoT, є дилема «швидкодія проти автономності». З одного боку, збільшення швидкості передавання скорочує час активності радіотракту, і за принципом «race to sleep» загальна енергія на пакет може навіть зменшитися. З іншого боку, високошвидкісні модуляційні схеми потребують кращого відношення сигнал/шум, що примушує підвищувати вихідну потужність передавача, а складніші протоколи вимагають довшої преамбули, синхронізації, обміну службовими кадрами, що з'їдає виграш. Крім того, автономність визначається не лише енергією на один пакет, а й середнім струмом споживання в режимі очікування, який для деяких інтерфейсів є домінуючим. Наприклад, модуль Wi-Fi, що постійно підтримує асоціацію з точкою доступу, споживає від 10 до 50 мА навіть без передавання даних, тоді як BLE-пристрій у режимі сну з періодичним рекламуванням — менше 1 мкА в середньому. Тому для датчика, що раз на годину відправляє кілька

байтів, BLE або LoRaWAN забезпечать на порядки більшу автономність, ніж Wi-Fi, незважаючи на те що пікова швидкість Wi-Fi на три порядки вища. Кількісно цей компроміс формалізується через коефіцієнт енергетичної ефективності інтерфейсу $\eta = R_{біт} / P_{сер}$, де $R_{біт}$ — бітова швидкість, а $P_{сер}$ — середня потужність споживання за повний цикл «сон — активність — сон». Чим вищий цей коефіцієнт, тим більше бітів можна передати на одиницю витраченої енергії, однак сам по собі він не враховує абсолютного рівня споживання в режимі сну, тому аналіз вимагає побудови енергетичного профілю для конкретного сценарію, як описано в підрозділі 2.1.2.

Не менш гострим є компроміс між вартістю та функціональністю. Поняття вартості в контексті вибору інтерфейсу охоплює не лише ціну мікросхеми фізичного рівня або модуля, а й вартість додаткових компонентів (кварцових резонаторів, антен, роз'ємів, термінуючих резисторів), збільшення площі друкованої плати, ускладнення трасування (наприклад, необхідність контролю хвильового опору для USB або Ethernet), а також витрати на сертифікацію. Так, простий UART-інтерфейс між двома мікроконтролерами коштує практично нуль — два виводи та дві доріжки, — тоді як впровадження Ethernet потребує окремого PHY-трансивера, імпульсного трансформатора, роз'єму RJ-45 із вбудованими магнітиками та ретельного чотиришарового дизайну плати, що разом збільшує собівартість на 3–10 доларів США. Додавання стільникового модему NB-IoT коштує ще дорожче — від 10 до 30 доларів за модуль плюс витрати на SIM-карту або eSIM та сертифікацію у оператора. З іншого боку, функціональність, яку надають ці дорожчі інтерфейси (глобальне покриття, наскрізна IP-адресація, гарантована якість обслуговування), може бути безальтернативною для бізнес-моделі продукту. Інженер має поставити запитання: чи готовий замовник платити за можливість віддаленого оновлення прошивки через стільникову мережу, чи достатньо локального BLE-з'єднання зі смартфоном техніка, який приїжджає раз на рік? Для промислових датчиків із великим радіусом дії додаткові 20 доларів на модем окупаються економією на виїздах персоналу; для масового споживчого трекера домашніх тварин ті самі 20 доларів роблять продукт неконкурентоспроможним. Саме тому вибір інтерфейсу ніколи не є суто технічним — він є техніко-економічним.

Третя пара антагоністів — безпека проти простоти — з розвитком регуляторного тиску набуває дедалі більшої ваги. Сучасний IoT-пристрій, під'єднаний до глобальної мережі, є потенційною точкою входу для атак на всю інфраструктуру, і забезпечення його кібербезпеки перетворюється з бажаної опції на законодавчо обов'язкову вимогу (зокрема, відповідно до європейського регламенту Cyber Resilience Act, що набуває чинності у

2025–2026 рр., та стандарту IEC 62443 для промислової автоматизації). Повноцінний захист передбачає взаємну автентифікацію пристрою та сервера, шифрування даних на транспортному рівні (TLS 1.3 або DTLS 1.3), цілісність програмного коду (secure boot), захищене зберігання ключів (secure element, як-от мікросхема ATECC608B), а також можливість оновлення прошивки із перевіркою цифрового підпису. Усі ці механізми різко підвищують складність системи, вимагають більшого обсягу флеш-пам'яті та ОЗП (лише стек TLS може займати понад 64 КБ), збільшують час виходу на ринок і потребують від розробника компетенцій у криптографії, які виходять за межі типових навичок програміста мікроконтролерів. На противагу цьому, простий UART-міст на двох дротах із фіксованою швидкістю 9600 біт/с не потребує жодної криптографії, однак будь-хто з фізичним доступом до лінії може прослухати трафік або вприснути шкідливі команди. Вибір між безпекою та простотою визначається моделлю загроз: якщо пристрій керує замком вхідних дверей, компроміс неможливий — потрібна максимальна безпека навіть ціною збільшення бюджету та строків; якщо ж це датчик температури в теплиці, що передає дані всередину локальної мережі, достатньо базового симетричного шифрування на канальному рівні, вбудованого в LoRaWAN або Zigbee.

Для систематизації процесу вибору пропонується використовувати матрицю компромісів, наведену в таблиці 2.6, де найпоширеніші інтерфейси IoT оцінено за п'ятибальною шкалою (від 1 — найгірше значення для критерію, до 5 — найкраще) за чотирма ключовими напрямками: пропускна здатність, енергоефективність (автономність), вартість реалізації та рівень вбудованої безпеки. Така таблиця не дає однозначної відповіді, але слугує інструментом для відсіювання явно непридатних варіантів і фокусування на двох-трьох кандидатах для подальшого детального аналізу.

Оцінки є інтегральними (від 1 – найгірше/найнижче до 5 – найкраще/найвище) і відображають баланс характеристик для масових вбудованих IoT-пристроїв, станом на 2026 рік. Вони можуть дещо змінюватися залежно від конкретних чіпів та умов експлуатації. Використано дані з документації виробників (STM32, nRF52/53, ESP32, CC13xx/26xx, Quectel BC660K/EG915, Semtech SX126x) та відкритих стандартів (IEEE 802.11/15.4, Bluetooth SIG, LoRaWAN, 3GPP).

Таблиця 2.6 – Матриця компромісів для вибору інтерфейсу IoT-пристрою

Інтерфейс	Характеристика	Значення
UART (TTL)	*Пропускна здатність (1–5)*	1 (низька, до кількох Мбіт/с)
	Автономність (1–5)	5 (мінімальне споживання, легко вимикається)
	Вартість (1–5)	5 (лише два проводи, без додаткових компонентів)
	Вбудована безпека (1–5)	1 (без захисту, потрібен зовнішній чип шифрування)
	Типові сценарії застосування	Налагоджувальна консоль, з'єднання MCU-MCU, прості датчики всередині плати
I²C	*Пропускна здатність (1–5)*	1 (до 1 Мбіт/с у Fast-mode Plus)
	Автономність (1–5)	5 (дуже низьке споживання, підтримка глибокого сну)
	Вартість (1–5)	5 (дві лінії, внутрішні підтяжки)
	Вбудована безпека (1–5)	1 (відсутній вбудований захист)
	Типові сценарії застосування	Бортові датчики (температура, вологість), EEPROM, PMIC, дрібна периферія
SPI	*Пропускна здатність (1–5)*	3 (одинарний до 100 Мбіт/с)
	Автономність (1–5)	3 (вище споживання на високих частотах, але кероване)
	Вартість (1–5)	5 (мало виводів, майже без зовнішніх компонентів)
	Вбудована безпека (1–5)	1 (без вбудованих механізмів безпеки)
	Типові сценарії застосування	Швидкісні АЦП/ЦАП, дисплеї, зовнішня пам'ять, FPGA
RS-485 (Modbus)	*Пропускна здатність (1–5)*	2 (до 10 Мбіт/с на короткі дистанції)
	Автономність (1–5)	4 (низьке споживання, підходить для живлення від батареї)
	Вартість (1–5)	4 (потрібен трансивер, але недорогий)
	Вбудована безпека (1–5)	1 (протокол не передбачає безпеки, потрібен додатковий рівень)
	Типові сценарії застосування	Промислова мережа (Modbus RTU), лічильники, віддалені датчики, автоматизація будівель

Продовження таблиці 2.6

Інтерфейс	Характеристика	Значення
CAN FD	*Пропускна здатність (1–5)*	3 (до 8 Мбіт/с у фазі даних)
	Автономність (1–5)	3 (помірне споживання, трансивер у сплячому режимі)
	Вартість (1–5)	3 (потрібен CAN-контролер і трансивер)
	Вбудована безпека (1–5)	1 (стандартний CAN не має криптографії, потрібен SecOC)
	Типові сценарії застосування	Автомобільна шина, промислові контролери, робототехніка
USB 2.0 (Device)	*Пропускна здатність (1–5)*	3 (до 480 Мбіт/с)
	Автономність (1–5)	2 (порівняно високе споживання, режим suspend)
	Вартість (1–5)	3 (потрібен USB-контролер, роз'єм)
	Вбудована безпека (1–5)	3 (підтримка захисту на протокольному рівні, але часто не використовується)
	Типові сценарії застосування	Підключення до ПК, реєстратори даних, заряджання/програмування
Ethernet (100BASE-TX)	*Пропускна здатність (1–5)*	4 (100 Мбіт/с)
	Автономність (1–5)	1 (високе споживання, не для батарейного живлення)
	Вартість (1–5)	2 (потрібні PHY, магнітика, роз'єм)
	Вбудована безпека (1–5)	3 (TLS/DTLS на верхніх рівнях, MACsec опційно)
	Типові сценарії застосування	Мережеві шлюзи, IP-камери, промислові контролери з виходом у LAN
BLE 5.4 (2M PHY)	*Пропускна здатність (1–5)*	3 (до 2 Мбіт/с)
	Автономність (1–5)	4 (наднизьке споживання в сні, Connection Subrating)
	Вартість (1–5)	3 (модуль коштує ~2–5 USD)
	Вбудована безпека (1–5)	3 (AES-CCM на канальному рівні, LE Secure Connections)
	Типові сценарії застосування	Носимі пристрої, радіомаяки, домашня автоматика, сенсорні мережі

Кінець таблиці 2.6

Інтерфейс	Характеристика	Значення
Zigbee/Thread	*Пропускна здатність (1–5)*	1 (250 кбіт/с)
	Автономність (1–5)	4 (дуже низьке споживання, роки від батарейки)
	Вартість (1–5)	3 (модуль ~3–6 USD)
	Вбудована безпека (1–5)	4 (AES-128 на MAC, захист на рівні мережі, безпечне приєднання)
	Типові сценарії застосування	Розумний дім, освітлення, вимірювання енергії, комерційні будівлі
LoRaWAN (клас A)	*Пропускна здатність (1–5)*	1 (до 50 кбіт/с, типово <5 кбіт/с)
	Автономність (1–5)	5 (роки автономної роботи від AA-батареї)
	Вартість (1–5)	3 (модуль ~5–10 USD)
	Вбудована безпека (1–5)	4 (AES-128 AppKey/AppSKey, наскрізне шифрування)
	Типові сценарії застосування	Сільське господарство, облік води/газу, віддалений моніторинг
NB-IoT	*Пропускна здатність (1–5)*	2 (до 100 кбіт/с)
	Автономність (1–5)	4 (PSM, eDRX, >10 років від батареї)
	Вартість (1–5)	1 (дорогий модем, потрібна сертифікація оператора)
	Вбудована безпека (1–5)	4 (вбудована SIM, автентифікація мережі, можливість TLS)
	Типові сценарії застосування	Розумне місто, лічильники, трекінг активів, сигналізації
Wi-Fi 6	*Пропускна здатність (1–5)*	5 (сотні Мбіт/с – Гбіт/с)
	Автономність (1–5)	1 (високе споживання, не для батарей)
	Вартість (1–5)	2 (чіп/модуль середньої вартості, але роялті)
	Вбудована безпека (1–5)	5 (WPA3, 802.1X, AES-256, захист на рівні мережі та каналу)
	Типові сценарії застосування	Відеоспостереження, високошвидкісні шлюзи, побутова техніка

Наочне уявлення про те, як різні інтерфейси позиціонуються в просторі «швидкодія — автономність», дає рисунок 2.6, на якому в логарифмічних координатах зіставлено типову швидкість передавання корисних даних та очікуваний термін автономної роботи від стандартної літієвої батареї CR2032 (225 мА·год) для кількох репрезентативних інтерфейсів. Діаграма показує, що високошвидкісні інтерфейси, як-от Wi-Fi, зосереджені у правому нижньому куті (велика швидкість, мала автономність), а енергоефективні LPWAN — у лівому верхньому (мала швидкість, велика автономність). BLE та Zigbee посідають проміжне положення, що робить їх універсальним вибором для багатьох сценаріїв.

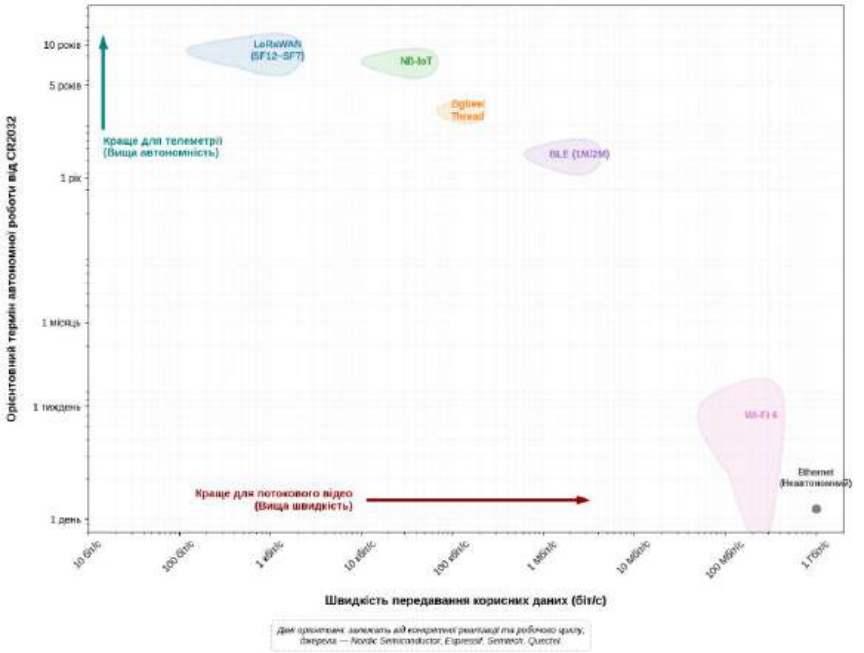


Рисунок 2.6 – Позиціонування інтерфейсів IoT у координатах швидкодія – автономність

2.4. Інженерний кейс: вибір інтерфейсу для автономного датчика вібрації

Розглянемо гіпотетичний, але типовий сценарій, що об’єднує всі описані критерії. Потрібно розробити датчик вібрації для моніторингу промислового насоса. Умови: живлення від батареї (тип AA, 2500 мА·год),

термін роботи не менше 5 років, частота вимірювань — 1 раз на годину, обсяг одного пакета — 100 байт (пікове значення, спектральна ознака), відстань до шлюзу — 50 м через залізобетонні перекриття, навколо — сильні електромагнітні завади від двигуна. Допустима затримка передачі — до 5 секунд. Бюджет пристрою — до 30 USD.

Крок 1. Відсіювання за енергетикою.

За формулою (2.3) та з урахуванням таблиці 2.2 обчислимо середню потужність за цикл. Припустимо, що мікроконтролер споживає 5 мА * 3,3В = 16,5 мВт під час активності (1 секунда) і 5 мкА * 3,3В ≈ 16,5 мкВт у сні (3599 секунд). Енергія на один цикл: $E_{цикл}$ = (0,0165Вт * 1 с.) + (0,0000165Вт * 3599 с.) ≈ 0,0165 + 0,0594 ≈ 0,0759 Дж. За 5 років (≈ 43 800 циклів) сумарна енергія ≈ 3320 Дж. Батарея AA 2500 мА·год при 3,3В забезпечує ≈ 3,3В × 2,5А·год × 3600 = 29 700 Дж, тобто запас є. Але якщо додати Wi-Fi (струм 200 мА в режимі передачі), то $E_{цикл}$ зросте більш ніж у 10 разів, що робить Wi-Fi неприйнятним. Отже, залишаються: BLE, Zigbee, LoRaWAN, NB-IoT, дротові варіанти (RS-485, CAN) — але дріт прокласти неможливо (насос рухомий? умова не уточнена, припустимо, що ні — тоді дротові варіанти відпадають через високу вартість монтажу).

Крок 2. Аналіз дальності та завадостійкості.

Залізобетонні перекриття послаблюють сигнал 2,4 ГГц (BLE, Zigbee) на 20–30 дБ. Типова дальність BLE 5.4 на відкритому просторі — до 100 м, усередині промислової будівлі — не більше 20–30 м. Наші 50 м із перешкодами — на межі, потрібна ретрансляція (mesh). Zigbee має подібний радіоінтерфейс (IEEE 802.15.4), але підтримує mesh-мережі, що дозволяє долати перешкоди через сусідні вузли. LoRaWAN на частоті 868 МГц має кращу проникну здатність крізь стіни (затухання на 10–15 дБ менше, ніж у 2,4 ГГц) і запас за чутливістю, тому 50 м із перекриттями — цілком реально навіть без ретрансляції. NB-IoT покладається на зовнішню вежу, але в глибині цеху сигнал може бути нестабільним, плюс висока вартість модему.

Крок 3. Врахування безпеки та вартості.

Для BLE і Zigbee вбудоване шифрування AES-128 є достатнім для більшості промислових застосувань. LoRaWAN забезпечує наскрізне шифрування між пристроєм і сервером (AppKey). Вартість модулів: BLE/Zigbee ~3–5 USD, LoRaWAN ~8–12 USD, NB-IoT ~15–25 USD. Бюджет 30 USD

дозволяє всі варіанти, але з NB-IoT доведеться економити на інших компонентах.

Крок 4. Остаточне рішення.

За критерієм «надійність зв'язку в умовах перешкод + автономність» найкраще підходить *LoRaWAN класу A*: дальність забезпечена без додаткових вузлів, енергоспоживання низьке, шифрування присутнє. Zigbee+mesh також міг би працювати, але збільшує складність мережі та латентність (декілька хопів). BLE без mesh — не гарантує зв'язку. Отже, для даного кейсу обираємо LoRaWAN.

Варто зазначити, що остаточне рішення на користь LoRaWAN було прийняте не одразу. Попередній аналіз показав, що використання Zigbee з mesh-топологією також теоретично здатне забезпечити зв'язок на відстані п'ятдесяти метрів через залізобетонні перекриття, однак для цього знадобилося б установити принаймні один додатковий вузол-ретранслятор у зоні прямої видимості між датчиком і шлюзом. Такий підхід збільшує складність монтажу, споживання енергії (кожен ретранслятор має постійно перебувати в режимі прийому) і латентність через багаторазову маршрутизацію. Для промислового застосування, де обслуговуючий персонал не завжди має змогу встановлювати додаткове обладнання, перевага надається рішенню з прямою передачею, навіть якщо воно коштує дещо дорожче. Крім того, було розглянуто альтернативу з використанням NB-IoT, яка забезпечує глобальне покриття без потреби в локальному шлюзі. Однак вартість модему та абонентської плати перевищила б бюджет пристрою більш ніж удвічі, а очікуваний термін автономної роботи скоротився б до двох років через більший струм споживання під час реєстрації в мережі. Таким чином, LoRaWAN виявився оптимальним балансом між дальністю, енергоефективністю та вартістю.

Ще одним важливим аспектом, який вплинув на вибір, стала можливість використання готових промислових модулів з інтегрованим стеком LoRaWAN і підтримкою класу A. Такі модулі, наприклад серії RAK Wireless або Murata, мають сертифікацію FCC, CE та відповідають вимогам LoRa Alliance, що значно прискорює вихід продукту на ринок. Крім того, вони забезпечують апаратне шифрування AES-128 на канальному рівні, що вирішує питання безпеки передачі даних без додаткових витрат на мікроконтролер. У результаті обрана конфігурація виявилася не тільки технічно придатною, але й комерційно обґрунтованою, що підтверджує комплексний підхід до вибору інтерфейсу, описаний у цьому розділі.

Отримане рішення, звісно, не є абсолютно універсальним. Якби умови задачі передбачали меншу відстань, наприклад десять метрів у межах одного приміщення без важких перекриттів, перевагу отримав би Zigbee або навіть BLE з мінімальним енергоспоживанням. Якби обсяг даних на годину сягав кількох мегабайт, довелося б відмовитися від LPWAN на користь Wi-Fi або стільникового зв'язку, пожертвувавши автономністю. Отже, запропонована методологія дає змогу свідомо обирати компроміс щоразу, коли виникає нова проєктна задача.

Цей приклад ілюструє, як інтегрувати дані з таблиць 2.2–2.6 та рисунка 2.6 у реальний інженерний аналіз.

Варто підкреслити, що вибір інтерфейсу на основі лише таблиць і графіків є лише першим, ескізним кроком. Остаточне рішення завжди потребує прототипування та вимірювань на реальному залізі, особливо коли йдеться про оцінку автономності чи стійкості до завад. Проте запропонована методологія — від визначення вимог до пропускної здатності та енергетичного бюджету, через аналіз завадостійкості та детермінованості, до зіставлення з нормативною базою та зважування компромісів за допомогою матриці — забезпечує системний, повторюваний і захищений від грубих помилок шлях до обґрунтованого вибору. Саме цей шлях ілюструватиметься в наступних частинах посібника, де кожен конкретний інтерфейс розглядатиметься за єдиною схемою: фізичний рівень — електричний рівень — логічний рівень — програмна реалізація — критерії вибору для IoT. Таким чином, перша частина завершується, сформувавши в читача понятійний апарат, необхідний для свідомого опанування решти матеріалу.

ЧАСТИНА ІІ.
ДРОТОВІ ІНТЕРФЕЙСИ ВБУДОВАНИХ СИСТЕМ

РОЗДІЛ 3.
НАЙПРОСТІШІ ПОСЛІДОВНІ ІНТЕРФЕЙСИ

3.1. UART/USART: асинхронна передача, форматування кадру, генератори швидкості

Універсальний асинхронний приймач-передавач, відомий під абревіатурою UART (Universal Asynchronous Receiver-Transmitter), є найстарішим і водночас найпоширенішим послідовним інтерфейсом у вбудованих системах. Його присутність у складі будь-якого мікроконтролера — від 8-бітних ATmega до багатоядерних STM32H7 та ESP32 — пояснюється граничною простотою реалізації, мінімальною кількістю сигнальних ліній та універсальністю застосувань: від виведення налагоджувальних повідомлень на термінал до обміну даними з модулями GPS, GSM, Bluetooth та іншими периферійними пристроями. У цьому підрозділі розглядаються фізичні та електричні основи UART, тоді як наступні підрозділи будуть присвячені програмній реалізації та особливостям конкретних апаратних модулів.

3.1.1. Електричні рівні TTL/CMOS, перетворювачі RS-232, RS-485

Базовий UART, реалізований у ядрі мікроконтролера, оперує логічними рівнями TTL (Transistor-Transistor Logic) або CMOS (Complementary Metal-Oxide-Semiconductor), які безпосередньо відповідають напрузі живлення мікросхеми. Для 3,3-вольтової логіки, що домінує в сучасних мікроконтролерах STM32, ESP32 та Raspberry Pi Pico, логічна одиниця передається напругою, близькою до 3,3 В (не менше 2,4 В для виходу згідно з JEDEC JESD8C), а логічний нуль — напругою, близькою до 0 В (не більше 0,4 В). У 5-вольтових системах, таких як класичні Arduino Uno на базі ATmega328P, рівні становлять відповідно 5 В і 0 В. Така несиметрична передача є надзвичайно економічною — потрібен лише один сигнальний провідник плюс спільна земля (GND), — однак вона має принципові обмеження. По-перше, запас завадостійкості для 3,3-вольтової логіки становить лише близько 0,4–0,7 В: будь-яка наведена електромагнітна завада амплітудою понад цей поріг здатна спричинити хибне спрацьовування приймача. По-друге, відсутність узгодження хвильового опору лінії призводить до відбиттів і дзвону на фронтах, що обмежує максимальну довжину кабелю кількома десятками сантиметрів на високих швидкостях і кількома

метрами на швидкостях до 9600 біт/с. Цих характеристик цілком достатньо для з'єднань на одній друкованій платі або в межах одного корпусу, однак для зв'язку між пристроями, рознесеними на метри чи десятки метрів, особливо в промисловому середовищі з високим рівнем завад, необхідне перетворення рівнів.

Саме для подолання вказаних обмежень було розроблено стандарт TIA/EIA-232-F, широко відомий під назвою RS-232. Цей стандарт, уперше представлений у 1960-х роках і багаторазово переглянутий, визначає не лише електричні рівні, а й розпіновку стандартного 9- або 25-контактного роз'єму DB-9/DB-25, максимальну швидкість передавання (до 115,2 кбіт/с у базовій версії, хоча сучасні перетворювачі підтримують до 1 Мбіт/с) та максимальну довжину кабелю (до 15 м на швидкості 20 кбіт/с). Ключовою відмінністю RS-232 є використання біполярних рівнів напруги: логічний нуль (space) передається напругою від +3 В до +15 В (типово +12 В), а логічна одиниця (mark) — напругою від -3 В до -15 В (типово -12 В). Діапазон між -3 В і +3 В є зоною нечутливості, що забезпечує значно більший запас завадостійкості порівняно з TTL. Для узгодження мікроконтролерних рівнів TTL/CMOS із рівнями RS-232 застосовують спеціалізовані мікросхеми перетворювачів, найвідомішою з яких є MAX232 (для 5-вольтових систем) та її 3,3-вольтовий аналог MAX3232 виробництва Maxim Integrated (тепер частина Analog Devices). Ці мікросхеми містять вбудований перетворювач напруги на основі конденсаторної помпи (charge pump), що формує необхідні ±10 В із однополярного живлення 3,3 В або 5 В, і чотири драйвери/приймачі: два для перетворення TTL → RS-232 і два для зворотного перетворення RS-232 → TTL. Схема підключення перетворювача MAX3232 до мікроконтролера показана на рисунку 3.1. Для роботи достатньо чотирьох зовнішніх конденсаторів ємністю 0,1 мкФ, які забезпечують роботу charge pump, що робить інтеграцію RS-232 у вбудовану систему надзвичайно простою.

Незважаючи на істотне підвищення завадостійкості, RS-232 залишається несиметричним інтерфейсом (сигнал відносно землі), що обмежує дальність зв'язку та схильність до синфазних завад. Для справді промислових застосувань, де потрібна багатоточкова мережа на відстанях до 1200 метрів, було створено стандарт TIA/EIA-485-A, відомий як RS-485. Фізичний рівень RS-485 використовує диференційну передачу сигналу по витій парі: два проводи, A (неінвертуючий) та B (інвертуючий), переносять взаємно протилежні рівні напруги. Логічна одиниця (mark) відповідає напрузі $V_A - V_B < -200$ мВ, логічний нуль (space) — $V_A - V_B > +200$ мВ.

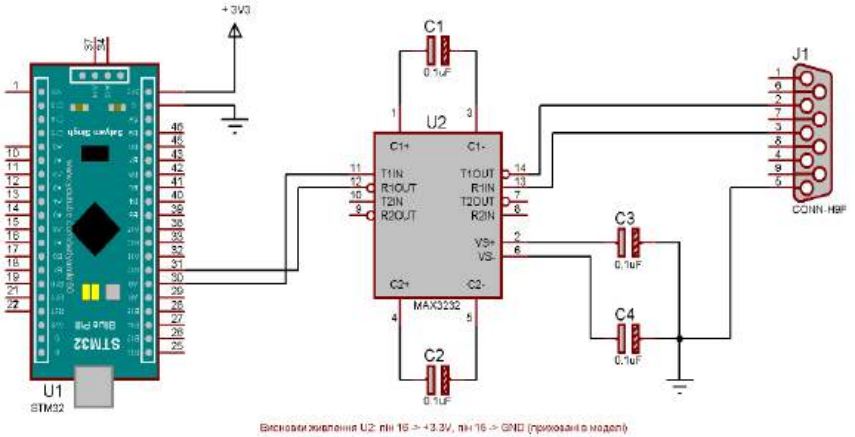


Рисунок 3.1 – Схема підключення перетворювача рівнів MAX3232 між UART мікроконтролера STM32 та роз'ємом DB-9

Стандарт вимагає, щоб передавач забезпечував диференційну вихідну напругу не менше 1,5 В на навантаженні 54 Ом, а приймач мав чутливість ±200 мВ. Синфазна напруга на лінії може перебувати в діапазоні від -7 В до +12 В, що дає змогу працювати в середовищах із високим рівнем індустріальних завад і навіть за наявності різниці потенціалів між землями окремих вузлів. Для перетворення UART-сигналів TTL у диференційний RS-485 використовують трансивери, найпоширенішими серед яких є MAX485 (Maxim Integrated), SN65HVD72 (Texas Instruments) та ADM485 (Analog Devices). Ці мікросхеми містять один диференційний драйвер (D) і один приймач (R), керовані виводами дозволу DE (Driver Enable) та RE (Receiver Enable), що дозволяє реалізувати напівдуплексний обмін по одній витій парі. Оскільки RS-485 є шиною з кількома вузлами, на обох кінцях лінії обов'язково встановлюють термінуючі резистори номіналом 120 Ом, що запобігають відбиттям сигналу, як це було пояснено в підрозділі 2.1.3. Схема підключення трансивера до мікроконтролера наведена на рисунку 3.2.

Для порівняння ключових електричних характеристик трьох розглянутих рівневих стандартів — TTL/CMOS, RS-232 та RS-485 — їх зведено в таблиці 3.1. Таблиця унаочнює, чому для лабораторного макетування достатньо TTL, для з'єднання з ПК зручний RS-232, а для розподілених промислових мереж безальтернативним є RS-485.

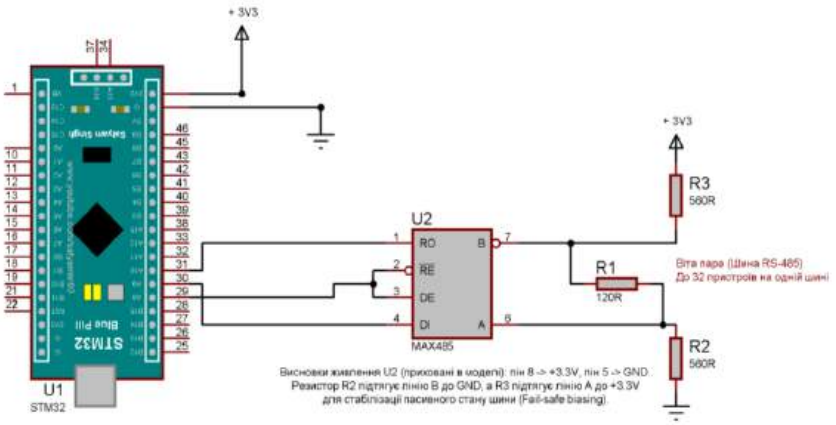


Рисунок 3.2 – Схема підключення трансивера MAX485 до UART мікроконтролера для реалізації напівдуплексного RS-485

Таблиця 3.1 – Порівняння електричних рівнів TTL/CMOS, RS-232 та RS-485 для UART

Рівні	Характеристика	Значення
TTL/CMOS (3,3 В логіка)	*Тип сигналу*	несиметричний
	Рівень логічного «0» на виході передавача	0 – 0,4 В
	Рівень логічної «1» на виході передавача	2,4 – 3,3 В
	Пороги приймача	V_{IL} < 0,8 В, V_{IH} > 2,0 В (запас ~0,4 В)
	Диференційна напруга	не застосовується
	Максимальна кількість вузлів на одній лінії	1 передавач, 1 приймач (точка-точка)
	Максимальна довжина лінії (типова)	0,5 м (50 Мбіт/с) – 5 м (9600 біт/с)
	Завадостійкість	низька (запас ~0,4 В)
	Необхідність узгодження (термінування)	не потрібне
	Типовий трансивер	безпосередньо виводи MCU
	Нормативний документ	JEDEC JESD8C
RS-232	*Тип сигналу*	несиметричний (однополярний з двополярним виходом)
	Рівень логічного «0» на виході передавача	+3…+15 В (вхід приймача сприймає як «0» при > +3 В)
	Рівень логічної «1» на виході передавача	−3…−15 В (вхід приймача сприймає як «1» при < −3 В)

Кінець таблиці 3.1

Рівні	Характеристика	Значення
RS-232	*Пороги приймача*	зона нечутливості −3…+3 В
	Диференційна напруга	не застосовується
	Максимальна кількість вузлів на одній лінії	1 передавач, 1 приймач
	Максимальна довжина лінії (типова)	до 15 м (20 кбіт/с)
	Завадостійкість	середня (запас ~3 В)
	Необхідність узгодження (термінування)	не потрібне
	Типовий трансивер	MAX3232 (3,3 В)
	Нормативний документ	TIA/EIA-232-F
RS-485	*Тип сигналу*	диференційний
	Рівень логічного «0» на виході передавача	$V_A > V_B$ на ≥1,5 В (на навантаженні 54 Ом)
	Рівень логічної «1» на виході передавача	$V_A < V_B$ на ≥1,5 В (на навантаженні 54 Ом)
	Пороги приймача	чутливість ±200 мВ
	Диференційна напруга	≥1,5 В (при навантаженні 54 Ом)
	Максимальна кількість вузлів на одній лінії	до 32 стандартних (до 256 з високоімпедансними приймачами)
	Максимальна довжина лінії (типова)	до 1200 м (100 кбіт/с) – 12 м (10 Мбіт/с)
	Завадостійкість	висока (CMRR >60 дБ)
	Необхідність узгодження (термінування)	обов’язкове (120 Ом на кінцях лінії)
	Типовий трансивер	MAX485, SN65HVD72
	Нормативний документ	TIA/EIA-485-A

Практичний висновок для студента полягає в тому, що проєктування будь-якого UART-з’єднання починається не з написання коду, а з аналізу електричного середовища. Якщо два мікроконтролери знаходяться на одній платі або з’єднані шлейфом довжиною 10 см, достатньо з’єднати їхні виводи TX і RX безпосередньо, не забувши про спільну землю. Якщо відстань становить кілька метрів, а приймачем є COM-порт комп’ютера, необхідно встановити перетворювач MAX3232. Якщо ж система розподілена по цеху, містить десятки датчиків на відстанях у сотні метрів, то безальтернативним вибором є RS-485 з обов’язковим термінуванням і, за потреби, гальванічною розв’язкою через оптрони або ізольовані трансивери, такі як ISO3082 (Texas Instruments). Саме цей рівень аналізу — від фізики сигналу

до стандарту — відрізняє інженера від аматора, і подальші підрозділи, присвячені програмуванню UART, спиратимуться на вже обране апаратне рішення як на незмінний фундамент.

3.1.2. Програмна реалізація опитування, переривання та DMA на STM32

Після того як електричний рівень UART узгоджено з фізичним середовищем, центральним завданням розробника стає організація обміну даними між ядром мікроконтролера та периферійним модулем USART. Мікроконтролери сімейства STM32, незалежно від конкретного ядра (Cortex-M0, M3, M4, M7, M33), надають три принципово різні механізми для цього: опитування прапорців статусу (polling), обробку переривань (interrupt) та прямий доступ до пам'яті (DMA). Вибір одного з них визначає не лише продуктивність і енергоефективність системи, а й архітектуру всього прикладного програмного забезпечення, тому розуміння особливостей кожного методу є обов'язковим для грамотного проєктування.

Найпростішим із погляду реалізації є режим опитування (polling), за якого центральний процесор циклічно перевіряє прапорці в регістрі статусу USART_SR (або ISR для STM32F7/H7) і виконує читання чи запис регістра даних USART_DR (або TDR/RDR) лише тоді, коли периферія готова. Бібліотека HAL пропонує дві основні функції: HAL_UART_Transmit() для відправлення масиву байтів і HAL_UART_Receive() для прийому заданої кількості байтів. Обидві функції є блокуючими: вони не повертають керування доти, доки весь масив не буде переданий або прийнятий, або доки не спливе таймаут, заданий параметром Timeout. Усередині цих функцій відбувається цикл, що очікує встановлення прапорця TXE (Transmit Data Register Empty) для передачі або RXNE (Receive Data Register Not Empty) для прийому, після чого байт записується або зчитується. Такий підхід вимагає від процесора постійної участі в обміні: на швидкості 115200 біт/с один байт передається близько 87 мкс, і весь цей час ядро може бути зайняте очікуванням, що абсолютно неефективно для систем реального часу з кількома задачами. Однак для початкового налагодження, виведення повідомлень на термінал у некритичних до часу ділянках коду або для пристроїв, де UART є єдиною периферією, режим опитування є цілком прийнятним завдяки своїй прозорості та відсутності потреби в конфігурації переривань. Функція HAL_UART_Transmit() у цьому випадку може використовуватися безпосередньо, як показано в лістингу 3.1, де відбувається від-

правлення рядка "Hello, UART!\r\n" на швидкості 115200 біт/с, сконфігурованій за допомогою CubeMX. У лістингу навмисно наведено мінімально необхідний код, щоб продемонструвати простоту першого кроку: після ініціалізації структури UART_HandleTypeDef та виклику HAL_UART_Init() достатньо однієї функції для передавання.

Лістинг 3.1 – Ініціалізація та передавання через UART у режимі опитування (STM32F4, HAL)

```
#include "main.h"
/* Дескриптор для конфігурації UART2 */
UART_HandleTypeDef huart2;
/* Прототипи системних функцій */
void SystemClock_Config(void);
static void MX_USART2_UART_Init(void);

int main(void)
{
  /* Скидання всієї периферії, ініціалізація Flash
інтерфейсу та Systick */
  HAL_Init();
  /* Настройка системного тактування (згенеровано
CubeMX) */
  SystemClock_Config();
 /* Ініціалізація периферії, налаштованої в CubeMX */
  MX_USART2_UART_Init();
/* Оголошення масиву з повідомленням для відправки */
  uint8_t msg[] = "Hello, UART!\r\n";

  while (1)
  {
     /*
      * Передавання даних через UART у режимі опитування
(блокуючий режим).
      * Параметри HAL_UART_Transmit:
      * 1. &huart2 - вказівник на структуру з конфігурацією
UART2.
      * 2. msg - вказівник на буфер із даними для відправки.
      * 3. sizeof(msg)-1 - кількість байтів (виключаючи
нуль-термінатор строки).
      * 4. 1000 - максимальний час очікування (тайм-аут) у
мілісекундах.
      */
     HAL_UART_Transmit(&huart2, msg, sizeof(msg) - 1, 1000);
     /* Затримка 1 секунда між відправками */
     HAL_Delay(1000);
  }
}
/**
  * @brief Функція ініціалізації USART2.
  * @param Немає
  * @retval Немає
  */
static void MX_USART2_UART_Init(void)
```

```
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;                /* Швидкість
передачі: 115200 біт/с */
  huart2.Init.WordLength = UART_WORDLENGTH_8B;  /* Довжина
слова: 8 бітів даних */
  huart2.Init.StopBits = UART_STOPBITS_1;       /*
Кількість стоп-бітів: 1 */
  huart2.Init.Parity = UART_PARITY_NONE;        /*
Перевірка парності: відсутня */
  huart2.Init.Mode = UART_MODE_TX_RX;           /* Режим
роботи: приймач та передавач */
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;  /* Апаратне
керування потоком: відсутнє */
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;

  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    /* Виклик функції обробки помилок у разі збою */
    Error_Handler();
  }
}
```

Логічним розвитком режиму опитування є використання переривань (interrupt mode), яке звільняє процесор від постійного очікування прапорців. Апаратний модуль USART може генерувати переривання за кількома подіями: TXE (регістр передавача порожній), TC (передача завершена), RXNE (регістр приймача не порожній), а також за помилками (ORE, NE, FE, PE). Бібліотека HAL для роботи з перериваннями надає функції HAL_UART_Transmit_IT() та HAL_UART_Receive_IT(), які ініціюють процес і негайно повертають керування. Після цього всі операції виконуються в обробнику переривання USARTx_IRQHandler(), який має викликати HAL_UART_IRQHandler(), а та, своєю чергою, аналізує прапорці й автоматично підставляє наступний байт із буфера або до буфера. Коли весь масив передано або прийнято, HAL викликає callback-функцію HAL_UART_TxCpltCallback() або HAL_UART_RxCpltCallback(), яку розробник має перевизначити у своєму коді для виконання потрібних дій (наприклад, встановлення семафора або обробки прийнятих даних). Передавання в режимі переривання є особливо ефективним, оскільки процесор витрачає час лише на запис одного байта в регістр TDR (одиниці тактів), а решту часу виконує корисну роботу. Приймання також не блокує систему, однак вимагає, щоб callback-функція встигала обробити дані до приходу наступного байта, інакше можливе переповнення апаратного зсувного регістра. Для високих швидкостей (понад 460800 біт/с) навантаження на процесор від переривань може стати надмірним — кожен байт викликає пере-

ривання, що для потоку 1 Мбіт/с означає понад 100 000 переривань на секунду, — і тоді єдиним виходом стає DMA. У лістингу 3.2 наведено приклад організації приймання одного байта в режимі переривання, де після отримання символу він негайно відправляється назад (echo), а також продемонстровано, як увімкнути відповідне переривання через HAL_NVIC_EnableIRQ().

Лістинг 3.2 – Приймання байта в режимі переривання та його echo-відправлення (STM32F4, HAL)

```
#include "main.h"
/* Дескриптор для конфігурації UART2 */
UART_HandleTypeDef huart2;
/* Глоб. буфер для збереження одного прийнятого байта */
uint8_t rx_byte;
/* Прототипи системних функцій */
void SystemClock_Config(void);
static void MX_USART2_UART_Init(void);
static void MX_NVIC_Init(void);

int main(void)
{
  /* Ініціалізація бібліотеки HAL */
  HAL_Init();
  /* Налаштування системного тактування */
  SystemClock_Config();
  /* Ініціалізація периферії UART2 */
  MX_USART2_UART_Init();
 /* Налаштування та увімкнення переривань NVIC для UART2 */
  MX_NVIC_Init();
  /*
   * Активація первинного приймання першого байта в режимі переривання.
   * Функція є неблокуючою: вона лише дозволяє переривання по RXNE (приймання)
   * і миттєво повертає керування. Коли байт надійде, спрацює колбек.
   */
  HAL_UART_Receive_IT(&huart2, &rx_byte, 1);

  while (1)
  {
    /*
     * Основний цикл залишається порожнім або виконує інші фонові задачі.
     * Процесор вільний, оскільки обмін даними повністю перенесено в переривання.
     */
  }
}

/**
  * @brief  Колбек (функція зворотного виклику) завершення приймання через UART.
```

```
  * @param  huart: вказівник на структуру
UART HandleTypeDef, яка викликала переривання.
  * @retval Немає
  */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  /* Перевіряємо, що переривання викликано саме модулем
USART2 */
  if (huart->Instance == USART2)
  {
    /*
     * Ехо-відправлення: негайно надсилаємо прийнятий байт
назад у режимі переривання.
     * Функція HAL UART Transmit IT є неблокуючою.
     */
    HAL_UART_Transmit_IT(&huart2, &rx_byte, 1);

    /*
     * Увага: HAL автоматично вимикає переривання приймання
після зчитування заданої кількості байт.
     * Для безперервного прослуховування лінії необхідно
знову активувати приймання наступного байта.
     */
    HAL_UART_Receive_IT(&huart2, &rx_byte, 1);

    /*
     * Коментар щодо архітектури:
     * Цей лістинг демонструє базовий механізм ехо-
відповіді для одного байта.
     * У реальних промислових додатках пряме викликання
Transmit IT та Receive IT всередині колбека
     * може призвести до втрати даних або перезапису буфера
(Overrun), якщо новий байт надійде до
     * завершення передачі попереднього. Для запобігання
втраті даних у реальних системах
     * обов'язково має використовуватися кільцевий буфер
(Circular Buffer / FIFO).
     */
  }
}

static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;

  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```
102:
103: static void MX_NVIC_Init(void)
104: {
105:   /* Встановлення пріоритету переривання для USART2
106: (Preemption = 0, SubPriority = 0) */
107:   HAL_NVIC_SetPriority(USART2_IRQn, 0, 0);
108:
109: /* Дозвіл переривань USART2 в NVIC */
110:   HAL_NVIC_EnableIRQ(USART2_IRQn);
111: }
```

Режим прямого доступу до пам'яті (DMA) є найпотужнішим і найскладнішим із трьох. Контролер DMA, присутній у всіх STM32 (від простого одноканального в STM32F0 до багатопотокового з підтримкою подвійної буферизації в STM32H7), здатен автоматично переносити дані між периферійним регістром даних UART та оперативною пам'яттю без жодного втручання процесора. Бібліотека HAL пропонує функції HAL_UART_Transmit_DMA() та HAL_UART_Receive_DMA(), які налаштовують дескриптори DMA і запускають передачу. Процесор лише ініціює операцію та отримує сповіщення через callback-функції HAL_UART_TxCpltCallback() / HAL_UART_RxCpltCallback() після завершення всього блоку даних. Для приймання надзвичайно корисним є режим циркулярного DMA (Circular Mode), за якого контролер DMA після досягнення кінця буфера автоматично повертається на його початок, безперервно наповнюючи буфер даними. Це дозволяє реалізувати високоефективний UART-приймач, де програма в будь-який момент може прочитати з буфера найсвіжіші дані, не боячись втратити байти через затримку обробки переривання. HAL підтримує спеціальну функцію HAL_UARTEx_ReceiveToIdle_DMA(), яка приймає дані через DMA доти, доки не настане пауза в прийманні (IDLE line detection), після чого викликається callback, і розробник може обробити саме стільки байтів, скільки надійшло. Цей метод є кращим для протоколів зі змінною довжиною пакетів, оскільки не потребує заздалегідь знати кількість байтів. На рисунку 3.3 схематично показано рух даних у трьох режимах, що допомагає наочно порівняти ступінь участі процесора в обміні.

У таблиці 3.2 наведено порівняння трьох режимів за основними критеріями, якими слід керуватися під час вибору. Слід підкреслити, що на практиці часто комбінують підходи: передавання виконують через DMA, а приймання — через DMA з виявленням IDLE, що дає змогу досягти максимальної продуктивності за мінімального втручання ядра.

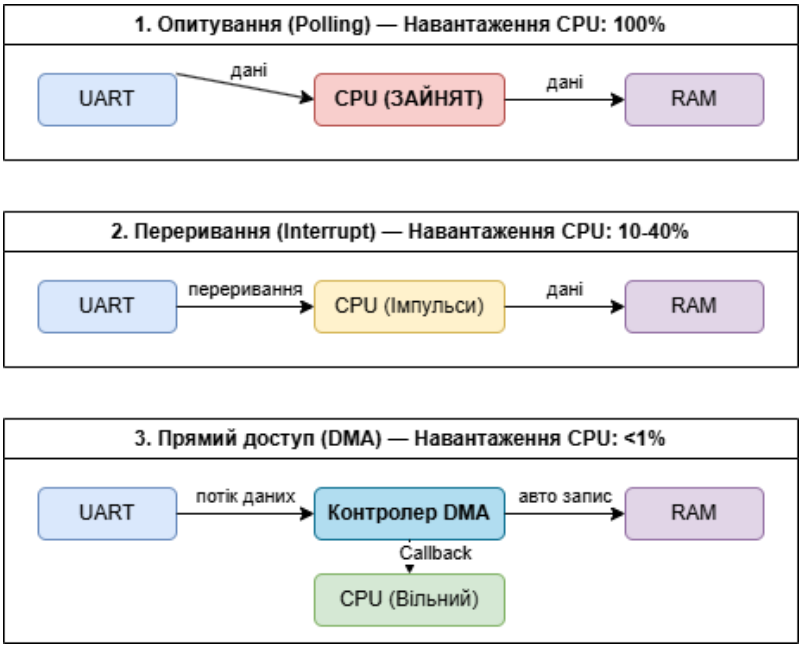


Рисунок 3.3 – Схеми руху даних між UART та пам'яттю в трьох режимах: опитування, переривання, DMA

Таблиця 3.2 – Порівняння методів програмної реалізації UART на STM32

Метод	Характеристика	Значення
Polling (опитування)	*Навантаження CPU*	дуже високе (блокує ядро)
	Пропускна здатність	обмежена швидкість через очікування
	Латентність реакції на прийом	мінімальна (процесор одразу зчитує байт)
	Складність реалізації	найпростіший
	Типове застосування	налагодження, виведення повідомлень
Interrupt (переривання)	*Навантаження CPU*	середнє (один вектор на байт)
	Пропускна здатність	висока, але зі зростанням швидкості зростає ймовірність втрат
	Латентність реакції на прийом	низька (обробка в ISR)
	Складність реалізації	середня
	Типове застосування	інтерактивні протоколи, робота з командами

Кінець таблиці 3.2

Метод	Характеристика	Значення
DMA (прямий доступ до пам'яті)	*Навантаження CPU*	відсутнє (лише обробка завершення блоку)
	Пропускна здатність	максимальна, обмежена лише UART
	Латентність реакції на прийом	може бути вищою через буферизацію (залежить від розміру буфера та порогу)
	Складність реалізації	висока (потрібна конфігурація DMA + IDLE)
	Типове застосування	високошвидкісний обмін, потоки даних, сенсори з безперервним виводом

Примітка: на практиці часто комбінують підходи: передавання виконують через DMA, а приймання — через DMA з виявленням IDLE (за допомогою переривання по холостому стану лінії). Це дозволяє досягти максимальної продуктивності за мінімального втручання ядра. Такий спосіб рекомендовано в багатьох прикладах STM32CubeMX та документації HAL.

Важливою деталлю, яку часто пропускають початківці, є коректне налаштування пріоритетів переривань DMA та UART. У системах із кількома периферійними пристроями не можна допускати, щоб обробник переривання UART витісняв обробник DMA, який обслуговує той самий UART, інакше можливі гонки та спотворення даних. Виробник у документі AN4031 «Using the STM32F2, STM32F4 and STM32F7 Series DMA controller» рекомендує призначати перериванню DMA вищий пріоритет, ніж перериванню UART, якщо використовується режим переривання одночасно з DMA для різних каналів. Крім того, слід пам'ятати, що HAL-функції не є реентерабельними, тому викликати їх одночасно з головного циклу та з переривання не можна — це призводить до непередбачуваної поведінки.

На завершення доцільно наголосити, що гнучкість модуля USART у STM32 не обмежується стандартними функціями HAL. Для найвибагливіших застосувань існує рівень LL (Low Layer), що дає змогу маніпулювати регістрами безпосередньо, а також можливість використання передових режимів, таких як автоматичне керування потоком (RTS/CTS) або синхронний режим SPI. Проте навіть освоєння трьох описаних методів — опитування, переривання, DMA — закладає надійний фундамент для розроб-

лення будь-яких комунікаційних протоколів на базі UART, від найпростішого Modbus RTU до складеного багатозадачного шлюзу. Подальший підрозділ, присвячений аналізу на основі літератури, продемонструє, як еволюція апаратних модулів UART у бік LPUART з автономним пробудженням розширює межі застосування цього, здавалося б, архаїчного інтерфейсу.

3.1.3. Аналіз на основі літератури: еволюція від UART до LPUART з автономним Wake-Up

Класичний універсальний асинхронний приймач-передавач (UART), незважаючи на свою понад півстолітню історію, аж до початку 2010-х років залишався периферійним блоком із незмінною фундаментальною архітектурою. Його ядро становили два зсувні регістри, генератор бітрейту на основі подільника системної тактової частоти, проста логіка формування та перевірки старт-стопових бітів, а також регістри даних і статусу. Така архітектура, детально описана в класичних підручниках з організації вбудованих систем, зокрема в роботі Д. Льюїса «Fundamentals of Embedded Software with the ARM Cortex-M3» (2012), чудово виконувала функцію термінального вводу-виводу, однак виявилася принципово обмеженою для нового класу пристроїв — автономних сенсорних вузлів Інтернету речей, від яких вимагалася багаторічна робота від батареї з періодичним пробудженням для передавання даних. Обмеження ці були зумовлені двома фундаментальними чинниками.

По-перше, генератор бітрейту класичного UART живився від системного тактового сигналу, який у режимах глибокого сну (Stop або Standby) відключався, що унеможливлювало приймання даних без попереднього пробудження ядра та запуску високочастотного генератора (HSE або HSI).

По-друге, для виявлення стартового біта використовувалася схема дискретизації з частотою, кратною бітрейту (зазвичай 16x), яка також вимагала постійного тактування. Як наслідок, мікроконтролер не міг перебувати в режимі низького споживання й одночасно очікувати на вхідний UART-трафік, що змушувало розробників вдаватися до зовнішніх допоміжних схем або періодичного пробудження для опитування, яке марно витрачало енергію. Ця проблема була ґрунтовно проаналізована в оглядовій статті М. Магно та колег «Ultra-Low Power Design: From Dream to Reality» (IEEE Transactions on Circuits and Systems I, том 64, 2017), де показано, що саме постійне тактування комунікаційної периферії є другим за величиною джерелом втрат енергії після радіотракту.

Відповіддю промисловості на цей виклик стала поява модуля LPUART (Low Power Universal Asynchronous Receiver-Transmitter), уперше представленого компанією STMicroelectronics у сімействі наднизькопотужних мікроконтролерів STM32L0 (документація RM0377, 2014 рік) і згодом поширеного на серії STM32L4, STM32L5, STM32G0, STM32G4, а також на окремі представники лінійки STM32WB. Принципова інновація LPUART полягала в тому, що він отримав власний, незалежний від системного тактового генератора, тактовий вузол, здатний працювати від низькошвидкісних джерел — LSE (Low Speed External, 32,768 кГц кварцовий резонатор), LSI (Low Speed Internal, ~32 кГц RC-генератор) або навіть від HSI16 із подільником. Оскільки LSE та LSI залишаються активними в режимах Sleep, Stop 0/1/2 (залежно від моделі) і навіть у деяких конфігураціях Standby, LPUART зберігає здатність детектувати стартовий біт і відновлювати бітову синхронізацію без участі основного процесорного ядра. Як зазначено в технічному посібнику RM0438 для STM32L552 (розділ 42, LPUART), модуль використовує схему дискретизації з програмованою кратністю (від 4x до 32x відносно бітрейту), що дозволяє гнучко балансувати між точністю приймання та енергоспоживанням. Ключовою ж функцією, яка й визначила назву модуля, стала здатність до автономного пробудження (Wake-Up) мікроконтролера з режиму низького споживання за фактом виявлення стартового біта на лінії RX. Цей механізм детально описано в прикладній нотатці AN4621 «Using the LPUART in STOP mode with STM32L4 Series microcontrollers» (STMicroelectronics, 2019), яка й нині залишається основним практичним керівництвом для розробників.

Аналіз літератури та технічної документації виробників дозволяє простежити чітку еволюційну траєкторію. Перше покоління LPUART у STM32L0/L4 підтримувало пробудження за стартовим бітом, однак вимагало, щоб тактування LSE було активним упродовж усього часу очікування, що давало струм споживання близько 1 мкА в режимі Stop 2 із LPUART. Друге покоління, реалізоване в STM32L5 із підтримкою TrustZone, додало можливість апаратного виявлення не лише стартового біта, а й конкретної адреси в першому байті (Address Match Wake-Up), що стало можливим завдяки інтеграції програмованого компаратора адреси безпосередньо в апаратний блок. Це рішення, натхненне аналогічним механізмом у модулях I²C, дозволило LPUART ігнорувати чужий трафік на загальній шині й пробуджувати мікроконтролер лише тоді, коли повідомлення адресоване саме йому, що особливо важливо для багатоточкових RS-485 мереж. Паралельно компанія Texas Instruments у мікроконтролерах MSP430FR із фероелектричною пам’яттю впровадила власний модуль

eUSCI_A з аналогічною функціональністю, а Nordic Semiconductor у nRF52/nRF53 інтегрувала UART із пробудженням у складі загальної системи керування живленням (EasyDMA та PPI), хоча їхній підхід більше спирався на пробудження через програмований контролер периферії, а не на автономний тактовий генератор UART. Порівняльний аналіз цих рішень, виконаний у дисертації К. Ванга «Ultra-Low-Power Communication Interfaces for IoT Sensor Nodes» (MIT, 2021), показує, що саме підхід STMicroelectronics із повністю незалежним тактуванням LPUART від LSE забезпечує найменший середній струм у режимі очікування вхідного трафіку — менше 1 мкА — завдяки тому, що решта системи, включно з високочастотними PLL та flash-пам'яттю, залишаються повністю знеструмленими.

Кількісне порівняння ключових характеристик класичного UART (на прикладі USART1 у STM32F407) та LPUART (на прикладі LPUART1 у STM32L476) наведено в таблиці 3.3, яка унаочнює масштаб досягнутого прогресу в енергоефективності.

Таблиця 3.3 – Порівняння характеристик класичного UART (STM32F4) та LPUART (STM32L4)

Характеристика	***USART (STM32F407)***	***LPUART (STM32L476)***
Джерело тактування	PCLK (APB1/APB2, системний такт)	незалежний тактовий вузол (LSE, LSI, HSI16, APB)
Можливі джерела тактування в режимі Stop	відсутні (USART неактивний у Stop)	LSE (32,768 кГц), LSI (~32 кГц), HSI16 із подільником
Споживання в режимі очікування (Stop 2)	Н/Д (USART вимкнено в Stop)	<1 мкА (LSE + LPUART активний)
Пробудження за стартовим бітом	ні	так
Пробудження за збігом адреси	ні	так (Address Match)
Мінімальна швидкість передавання	~80 біт/с (при PCLK=84 МГц)	100 біт/с
Максимальна Швидкість передавання	до 10,5 Мбіт/с (8х оверсемплінг)	до 5,25 Мбіт/с (при HSI16)
Кратність дискретизації	фіксована 16х або 8х (oversampling)	програмована (4х–32х)

Функціонування механізму Wake-Up схематично зображено на рисунку 3.4, де показано часову діаграму переходу мікроконтролера з режиму Stop 2 до активного приймання кадру UART. На діаграмі видно, що після появи стартового біта LPUART витрачає кілька мікросекунд на детектування фронту та підтвердження, після чого генерує сигнал пробудження для блоку живлення (PWR), який, своєю чергою, запускає HSI16 та ядро. Протягом цього часу перші біти кадру неминуче втрачаються, тому для гарантованого приймання використовують стандартну техніку: передавач надсилає попередній «холостий» байт (dummy byte) або, як це регламентовано в Modbus RTU, перед корисним повідомленням витримується пауза, достатня для пробудження всіх пристроїв на шині.

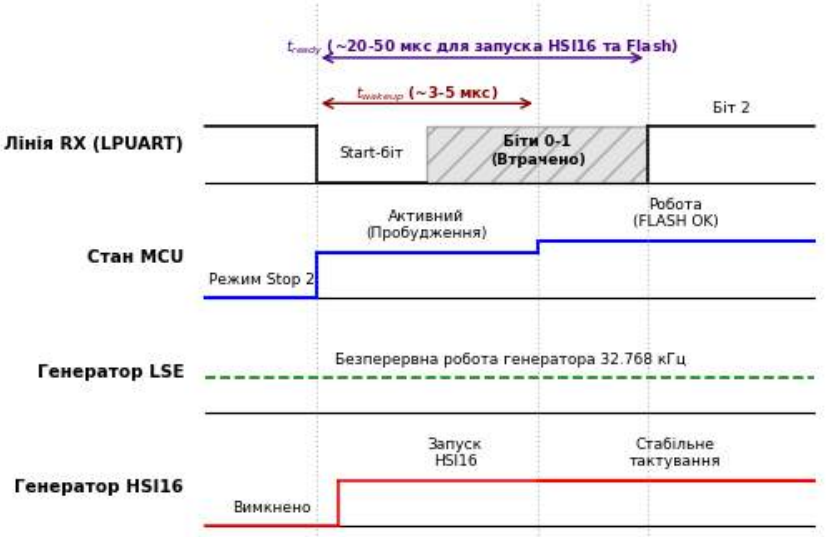


Рисунок 3.4 – Часова діаграма пробудження мікроконтролера з режиму Stop 2 за сигналом LPUART

У лістингу 3.3 наведено мінімальний код конфігурації LPUART для пробудження з режиму Stop 2 на платформі STM32L476 з використанням бібліотеки HAL. Суттєвим є те, що після виклику HAL_UARTEx_ReceiveToIdle_IT() або HAL_UART_Receive_IT() мікроконтролер може бути переведений у режим Stop, і при надходженні стартового біта він автоматично прокинеться, виконає обробник переривання та продовжить роботу головного циклу.

Лістинг 3.3 – Конфігурація LPUART1 для пробудження з режиму Stop 2 (STM32L476, HAL)

```
#include "main.h"
// Глобальні змінні
UART_HandleTypeDef hlpuart1;
uint8_t rx_buffer[20];
volatile uint8_t g_uart_wakeup_flag = 0; // Прапорець успішного пробудження та прийому даних
// Прототипи функцій
void SystemClock_Config(void);
void MX_LPUART1_UART_Init(void);
void Enter_Stop2_And_Resume(void);

int main(void)
{
  // Ініціалізація периферії архітектури HAL
  HAL_Init();
  SystemClock_Config();
  MX_LPUART1_UART_Init();
  /* Готуємо LPUART до прийому даних у режимі переривань перед входом у Stop 2.
     Використовуємо функцію з підтримкою Idle-лінії, щоб зафіксувати закінчення кадру,
     або стандартну HAL UART Receive IT(&hlpuart1, rx buffer, sizeof(rx buffer)). */
  if (HAL_UARTEx_ReceiveToIdle_IT(&hlpuart1, rx_buffer, sizeof(rx_buffer)) != HAL_OK)
  {
    Error_Handler();
  }
  while (1)
  {
    // Головний цикл програми
    if (g_uart_wakeup_flag == 0)
    {
     // Виклик функції переходу в режим глибокого енергозбереження Stop 2
      Enter_Stop2_And_Resume();
    }
   // Якщо прапорець встановлено — MCU успішно прокинувся і обробив дані
    if (g_uart_wakeup_flag)
    {
      // Точка обробки прийнятих даних з буфера rx_buffer
      g_uart_wakeup_flag = 0; // Скидання прапорця
    }
  }
}
/**
  * @brief Ініціалізація LPUART1 з тактуванням від LSE (9600 біт/с, 8N1)
  */
void MX_LPUART1_UART_Init(void)
{
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
  /* НАГОЛОШЕННЯ: Конфігурація джерела тактування для
```

```
55:LPUART1.
56:   Використовуємо безперервно працюючий генератор LSE (32
57:768 Гц) */
58:  PeriphClkInit.PeriphClockSelection = RCC_PERIPH
59:CLK_LPUART1;
60:  PeriphClkInit.Lpuart1ClockSelection =
61:RCC_LPUART1CLKSOURCE_LSE;
62:  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit)!= HAL_OK)
63:  {
64:    Error_Handler();
65:  }
66:  /* Ввімкнення тактування модуля LPUART1 */
67:  __HAL_RCC_LPUART1_CLK_ENABLE();
68:
69:  hlpuart1.Instance = LPUART1;
70:  hlpuart1.Init.BaudRate = 9600;                    // Біт
71:рейт 9600 біт/с
72:  hlpuart1.Init.WordLength = UART_WORDLENGTH_8B; // 8 біт
73:даних
74:  hlpuart1.Init.StopBits = UART_STOPBITS_1; // 1 стоп-біт
75:  hlpuart1.Init.Parity = UART_PARITY_NONE;         // Без
76:перевірки парності (N)
77:  hlpuart1.Init.Mode = UART_MODE_TX_RX;  // Дозволено
78:прийом та передачу
79:  hlpuart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;  // Без
80:апаратного керування потоком
81:  hlpuart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISA
82:BLE;
83:  hlpuart1.AdvancedInit.AdvFeatureInit = UART_ADVFEA
84:TURE_NO_INIT;
85:
86:  if (HAL_UART_Init(&hlpuart1) != HAL_OK)
87:  {
88:    Error_Handler();
89:  }
90:  /* Налаштування LPUART1 для активації пробудження по де
91:тектуванню Старт-біту */
92:  HAL_UARTEx_EnableStopModeWakeUpSource(&hlpuart1,
93:UART_WAKEUP_ON_STARTBIT);
94:  /* Увімкнення периферийного модуля для роботи в режимі
95:Stop */
96:  HAL_UARTEx_EnableLinearToWakeUpMcu(&hlpuart1);
97:}
98:/**
99:  * @brief Функція безпечного переходу в режим Stop 2 та
100:відновлення після нього
101:  */
102:void Enter_Stop2_And_Resume(void)
103:{
104:  /* НАГОЛОШЕННЯ 1: Перед входом у Stop 2 необхідно
105:обов'язково переконатися, що UART не перебуває в стані
106:активної передачі даних, щоб уникнути втрати байтів.
107:    Якщо передача активна — примусово скасовуємо її за до
108:помогою HAL UART Abort IT(). */
109:  if (HAL_UART_GetState(&hlpuart1) ==
110:HAL_UART_STATE_BUSY_TX)
111:  {
```

```
    HAL_UART_Abort_IT(&hlpuart1);
  }
  /* Очищення прапорця пробудження в самому модулі UART
перед засинанням */
  __HAL_UART_CLEAR_FLAG(&hlpuart1, UART_CLEAR_WUF);
  /* Вхід у режим енергозбереження Stop 2 за логікою WFI
(Wait For Interrupt) */
  HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);
  /* ================================================ */
  /* МІКРОКОНТРОЛЕР ПРОКИДАЄТЬСЯ ТУТ ПІСЛЯ ДЕТЕКТУВАННЯ
СИГНАЛУ НА ЛІНІЇ RX    */
  /* ================================================ */
 /* НАГОЛОШЕННЯ 2: Після виходу з режиму Stop 2 ядро авто
матично починає тактуватися
     від внутрішнього генератора MSI або HSI16 (який міг
бути раніше вимкнений).
     Для стабільної роботи периферії необхідно негайно пе
реініціалізувати
     основний системний тактовий генератор системи (напри
клад, PLL від HSE/MSI). */
  SystemClock_Config();
}
/**
  * @brief Обробник переривання завершення прийому даних
(Callback-функція)
  * Автоматично викликається з відповідного ISR після про
будження ядра та зчитування кадру.
  */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if (huart->Instance == LPUART1)
  {
   // Встановлюємо прапорець для головного циклу програми
    g_uart_wakeup_flag = 1;
    // Перезапускаємо процес прийому для наступного циклу
очікування
    HAL_UARTEx_ReceiveToIdle_IT(&hlpuart1, rx_buffer,
sizeof(rx_buffer));
  }
}
```

Еволюція UART не обмежилася лише енергозбереженням. Паралельно з LPUART відбувалося вдосконалення функціональності в напрямку підтримки вищих швидкостей (до 27 Мбіт/с у USART1 деяких STM32H7), апаратного керування потоком (RTS/CTS), синхронного режиму роботи (SPI-подібний USART), можливості емуляції інтерфейсів Smartcard (ISO 7816-3) та IrDA. Однак саме LPUART став тією ланкою, яка поєднала, здавалося б, непоєднуване: сумісність із наймасовішим і найпростішим дротовим протоколом та здатність роками працювати від крихітної батареї, що є необхідною умовою для масового розгортання розподілених сенсорних мереж.

Таким чином, аналіз науково-технічної літератури та первинної документації виробників засвідчує, що трансформація UART від класичного модуля до LPUART є не просто кількісним покращенням параметрів, а якісним стрибком, який перетворив цей інтерфейс із суто термінального засобу на повноцінний енергоефективний комунікаційний вузол, придатний для багаторічної автономної роботи в складі IoT-систем. Студент, який опановує цей матеріал, повинен винести ключовий урок: навіть, здавалося б, «завершена» технологія може зазнати докорінного переосмислення зі зміною вимог ринку, і саме в такі моменти глибоке розуміння фізичних та архітектурних обмежень дозволяє інженерові не просто використовувати готові рішення, але й критично оцінювати їхню еволюцію та обирати оптимальний варіант для конкретного проєкту.

3.2. SPI (Serial Peripheral Interface): архітектура Master-Slave, режими полярності та фази

Послідовний периферійний інтерфейс SPI (Serial Peripheral Interface), розроблений компанією Motorola на початку 1990-х років, нині є одним із трьох базових внутрішньосхемних протоколів, присутніх практично в кожному мікроконтролері. Його фундаментальна відмінність від UART полягає в синхронному характері передавання: окремий тактовий сигнал SCLK, що генерується майстром, жорстко прив'язує моменти виставлення та фіксації даних до фронтів цього сигналу, усуваючи потребу в старт-стопових бітах і високоточних незалежних генераторах на приймальному боці. Завдяки цьому SPI здатен працювати на частотах у десятки мегагерц (до 100 МГц у високошвидкісних периферійних модулях STM32H7 згідно з RM0468), забезпечуючи повнодуплексний обмін із мінімальною службовою надлишковістю. Саме SPI є основним інтерфейсом для підключення зовнішньої флеш-пам'яті (NOR Flash, NAND Flash), SD-карт, дисплеїв, АЦП/ЦАП, цифрових потенціометрів, акселерометрів та багатьох інших периферійних мікросхем, де критичними є висока пропускна здатність і простота реалізації.

Архітектура SPI базується на чіткому розділенні ролей: завжди існує рівно один майстер (Master) і один або кілька ведених (Slave). Майстер одноосібно керує тактовим сигналом SCLK і ініціює всі обміни, тоді як ведений лише відповідає на тактові імпульси, не маючи змоги самостійно розпочати передавання. Фізичне з'єднання передбачає чотири логічні лінії: SCLK (Serial Clock) — тактовий сигнал, що виробляється майстром; MOSI (Master Out Slave In) — лінія даних від майстра до веденого; MISO (Master

In Slave Out) — лінія даних від веденого до майстра; SS (Slave Select) або CS (Chip Select) — сигнал вибору веденого пристрою, який у активному стані (зазвичай низькому) дозволяє веденому брати участь в обміні. Оскільки MISO та MOSI є незалежними, SPI забезпечує повнодуплексний обмін: в той самий момент, коли майстер надсилає біт по MOSI, ведений може надсилати біт по MISO, що робить інтерфейс надзвичайно ефективним для регістрового доступу, де команда читання передається одночасно з отриманням даних. Структурну схему базового з'єднання «один майстер — один ведений» подано на рисунку 3.5, де окремо виділено зсувні регістри обох пристроїв, з'єднані в кільце: вихід зсувного регістра майстра через MOSI потрапляє на вхід регістра веденого, а вихід веденого через MISO — на вхід майстра, утворюючи замкнений контур, у якому за вісім тактів SCLK відбувається обмін байтами.

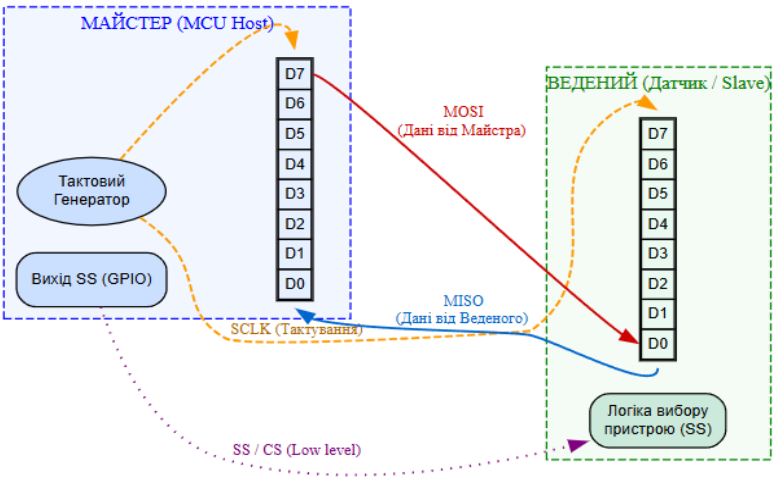


Рисунок 3.5 – Структурна схема SPI-з'єднання зі зсувними регістрами (за кожен такт SCLK відбувається зсув на один біт в обох регістрах одночасно, що забезпечує повнодуплексний обмін)

Ключовою особливістю SPI, яка водночас є джерелом найбільшої плутанини серед початківців, є наявність чотирьох режимів роботи, що визначаються комбінацією двох параметрів: полярності тактового сигналу CPOL (Clock Polarity) та фази CPHA (Clock Phase). Параметр CPOL визначає рівень сигналу SCLK у стані спокою (коли передавання не відбувається): при CPOL = 0 лінія SCLK перебуває в низькому стані, при CPOL = 1 — у високому. Параметр CPHA визначає, за яким фронтом SCLK

відбувається фіксація даних (вибірка) і за яким — їх виставлення (зміна). При CPHA = 0 дані вибірково фіксуються приймачем за першим фронтом після зміни SS (для CPOL=0 це передній фронт, тобто перехід із 0 в 1; для CPOL=1 це задній фронт, перехід із 1 в 0), а виставляються передавачем за протилежним фронтом. При CPHA = 1 дані виставляються за першим фронтом, а фіксуються за другим. Таким чином, чотири можливі комбінації (CPOL=0, CPHA=0; CPOL=0, CPHA=1; CPOL=1, CPHA=0; CPOL=1, CPHA=1) утворюють чотири режими, які часто позначають як Mode 0, Mode 1, Mode 2, Mode 3 відповідно. Часові діаграми для всіх чотирьох режимів на прикладі передачі одного байта показано на рисунку 3.6, де чітко видно моменти виставлення даних (стрілки, спрямовані до фронту) та моменти їх фіксації (вертикальні пунктирні лінії).

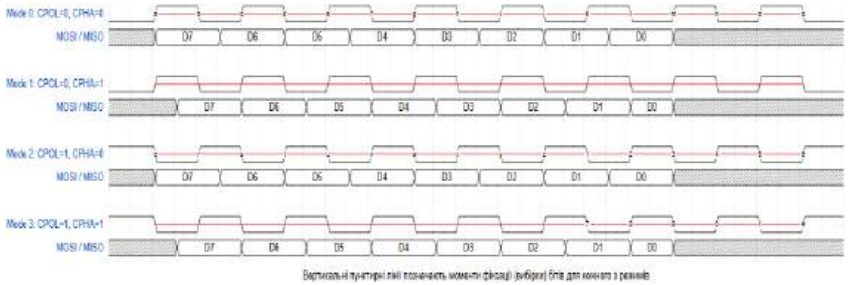


Рисунок 3.6 – Часові діаграми чотирьох режимів SPI

У таблиці 3.4 наведено формалізований опис кожного режиму, що може слугувати швидким довідником під час налаштування периферії.

Таблиця 3.4 – Режими роботи SPI залежно від CPOL та CPHA

Режим	CPOL	CPHA	Рівень SCLK у стані спокою	Фронт фіксації даних (вибірка)	Фронт виставлення даних (зміна)
Mode 0	0	0	низький	передній (0→1)	задній (1→0)
Mode 1	0	1	низький	задній (1→0)	передній (0→1)
Mode 2	1	0	високий	задній (1→0)	передній (0→1)
Mode 3	1	1	високий	передній (0→1)	задній (1→0)

Примітка: вибір конкретного режиму визначається виключно веденим пристроєм і фіксується в його даташиті; майстер має бути налаштований відповідно.

Слід підкреслити, що вибір режиму не є предметом інженерних уподобань; він суворо визначається даташитом веденої мікросхеми. Більшість

сучасних периферійних пристроїв (флеш-пам'ять, датчики) підтримують Mode 0 або Mode 3 як найпоширеніші, однак завжди необхідно перевіряти відповідний розділ документації, оскільки несумісність режиму призведе до того, що пристрій не відповідатиме на запити або повертатиме некоректні дані.

3.2.1. Багатопристроєві конфігурації з апаратним та програмним керуванням CS

На відміну від UART, де адресація ведених пристроїв можлива лише на рівні протоколу (як у Modbus), SPI від самого початку проєктувався з розрахунком на можливість підключення кількох ведених до одного майстра. Ключову роль у цьому відіграє сигнал вибору кристала CS (Chip Select), який у активному стані (низькому рівні, CS#) дозволяє конкретному веденому приймати тактовий сигнал і брати участь в обміні, тоді як решта ведених ігнорують шину, переводячи свої виводи MISO у високоімпедансний стан. Таким чином, для підключення N ведених пристроїв майстер повинен забезпечити N незалежних ліній CS, тоді як лінії SCLK, MOSI та MISO можуть бути спільними для всіх. Це істотно економить кількість виводів мікроконтролера порівняно з окремими SPI-інтерфейсами для кожного пристрою, однак потребує дисциплінованого керування сигналами CS із боку програмного забезпечення.

Практична реалізація керування CS розпадається на два підходи: апаратний та програмний. Апаратне керування CS передбачає використання спеціалізованого виводу NSS (Negated Slave Select), який вбудований до складу периферійного модуля SPI більшості мікроконтролерів. У режимі майстра за відповідного налаштування (біт SSOE у STM32 або аналогічний) апаратний модуль автоматично встановлює низький рівень на виводі NSS перед початком передачі першого біта й повертає високий рівень після завершення останнього біта, без жодного втручання програміста. Такий автоматизм є зручним, однак має суттєве обмеження: апаратний NSS зазвичай жорстко прив'язаний до одного конкретного виводу мікросхеми, що унеможливлює його використання для керування кількома веденими без зовнішнього демультиплексора. Крім того, апаратне керування CS у режимі майстра в STM32 (за винятком окремих моделей) не підтримує повноцінного режиму «один майстер — багато ведених» із автоматичним перемиканням між лініями CS; воно розраховане на конфігурацію «точка-точка». У ESP32, згідно з технічним описом ESP32-S3 Technical Reference Manual (розділ SPI), апаратний модуль SPI підтримує до трьох апаратних

ліній CS (SPI_CS0, SPI_CS1, SPI_CS2) у режимі майстра, що вже дозволяє безпосередньо керувати трьома веденими без додаткових GPIO.

Програмне керування CS (часто зване GPIO-керуванням) є гнучкішим і найуживанішим на практиці. У цьому випадку вивід NSS периферійного модуля SPI не використовується для вибору веденого; замість нього розробник конфігурує довільні виводи загального призначення (GPIO) як цифрові виходи і вручну встановлює низький рівень на потрібному виводі перед викликом функції передавання, а після її завершення повертає високий рівень. Такий підхід надає необмежену гнучкість у кількості ведених (обмежену лише кількістю вільних GPIO) і дозволяє точно контролювати часові інтервали між активізацією CS та початком тактування, що є критичним для дотримання вимог даташитів деяких мікросхем (наприклад, час setup time t_CSS для NOR Flash). Недоліком є необхідність явного написання коду для кожного CS та ризик помилок синхронізації, якщо розробник забуде зняти активний рівень після завершення транзакції. На рисунку 3.7 показано типову багатопристроєву конфігурацію з програмним керуванням CS, де до одного SPI-модуля мікроконтролера під'єднано три ведені пристрої (флеш-пам'ять, акселерометр і ЦАП), а три окремі GPIO виконують роль CS1, CS2, CS3. У таблиці 3.5 порівнюються характеристики апаратного та програмного методів керування CS, що допомагає обрати доцільний варіант для конкретного проєкту.

Таблиця 3.5 – Порівняння методів керування CS у SPI

Критерій	Апаратне керування (NSS)	Програмне керування (GPIO)
Кількість підтримуваних ведених	1(без зовнішнього декодера)	Необмежена (за кількістю GPIO)
Гнучкість вибору виводів	низька (фіксований вивід)	висока
Автоматичне встановлення/зняття CS	так (апаратно)	ні (ручне)
Навантаження на CPU	відсутнє	один запис у порт на транзакцію
Ризик людської помилки	низький (апаратний автомат)	середній (можна забути зняти CS)
Підтримка в STM32	SSOE-режим для одного веденого	стандартний метод
Підтримка в ESP32	до 3 апаратних CS	стандартний метод
Підтримка в RP2040	не підтримується (PIO імітує)	стандартний метод

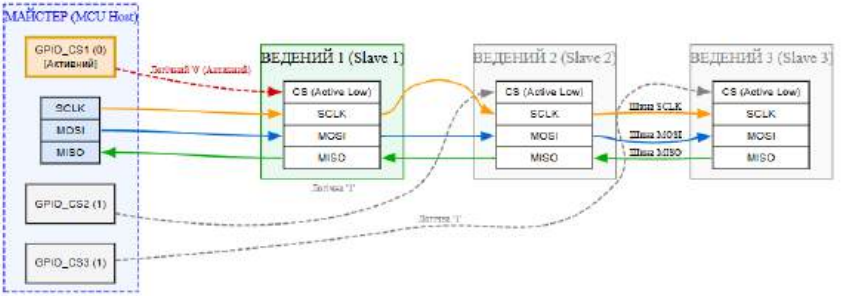


Рисунок 3.7 – Багатопристроєва конфігурація SPI з програмним керуванням CS

Типовий код ініціалізації SPI у бібліотеці HAL для STM32 із програмним керуванням CS наведено в лістингу 3.4. У цьому фрагменті демонструється, як налаштувати SPI у режимі майстра (Mode 0 або Mode 3), після чого перед кожною транзакцією явно скидати відповідний GPIO-вивід у низький рівень, викликати HAL_SPI_Transmit() або HAL_SPI_TransmitReceive(), і після завершення повертати вивід у високий рівень. Такий патерн є універсальним і застосовується на всіх трьох цільових платформах посібника.

Лістинг 3.4 – Програмне керування CS під час SPI-транзакції на STM32 (HAL)

```
#include "main.h"
// Глобальні дескриптори периферії
SPI_HandleTypeDef hspi2;
// Прототипи функцій
void MX_SPI2_Init(void);
void MX_GPIO_Init(void);
void spi_select(uint8_t cs_num);
void spi_deselect(uint8_t cs_num);
void delay_ns_or_nop(void); // Функція для забезпечення
мікрозатримок t CSS/t CSH
int main(void)
{
  // Ініціалізація апаратної абстракції HAL та системного
тактування
  HAL_Init();
  SystemClock_Config();
  // Ініціалізація GPIO (ліній CS) та модуля SPI2
  MX_GPIO_Init();
  MX_SPI2_Init();
  // Буфери для обміну даними з датчиком
  uint8_t tx_buf[2] = {0x80 | 0x3F, 0x00}; // Приклад: ко
манда читання (0x80) регістра 0x3F
  uint8_t rx_buf[2] = {0x00, 0x00}; // Буфер для отримання
```

```
175:відповіді
176:  while (1)
177:  {
178:    /* ============================================== */
179:    /* ПРИКЛАД ЧИТАННЯ РЕГІСТРА ДАТЧИКА ЧЕРЕЗ CS1  */
180:    /* ============================================== */
181:    // 1.Активація веденого пристрою №1(скидання лінії в 0)
182:    spi_select(1);
183:  /* НАГОЛОШЕННЯ 1 (t CSS): Обов'язково витримуємо час очіку
184:вання (Setup Time)між спадним фронтом CS та першим тактовим
185:імпульсом SCLK згідно з даташитом датчика. Для швидких MCU
186:(~80-216 МГц) тут необхідна мікрозатримка на NOP або DWT */
187:    delay_ns_or_nop();
188:    // 2. Повнодуплексна транзакція: одночасна передача ко
189:манди та прийом даних
190:    if (HAL_SPI_TransmitReceive(&hspi2, tx_buf, rx_buf, 2,
191:HAL_MAX_DELAY) != HAL_OK)
192:    {      Error_Handler();    }
193:    /* НАГОЛОШЕННЯ 2 (t CSH): Витримуємо час утримання
194:(Hold Time) після останнього
195:       тактового імпульсу SCLK перед тим, як повернути
196:лінію CS у високий стан. */
197:    delay_ns_or_nop();
198:    // 3. Деактивація веденого пристрою №1 (повернення
199:лінії в 1)
200:    spi_deselect(1);
201:    // Пауза перед наступним опитуванням датчика
202:    HAL_Delay(100);     }  }
203:/**
204:  * @brief Ініціалізація модуля SPI2 (Mode 3, 10 Мбіт/с, 8
205:біт, MSB First)
206:  * Код згенеровано на основі конфігурації CubeMX
207:  */
208:void MX_SPI2_Init(void)  {
209:  hspi2.Instance = SPI2;
210:  hspi2.Init.Mode = SPI_MODE_MASTER;                //
211:STM32 виступає як Master
212:  hspi2.Init.Direction = SPI_DIRECTION_2LINES;       // По
213:внодуплексний режим (MOSI + MISO)
214:  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
215:// Розмір кадру — 8 біт
216:/* НАГОЛОШЕННЯ: Налаштування Mode 3 (CPOL = 1, CPHA = 1) */
217:  hspi2.Init.CLKPolarity = SPI_POLARITY_HIGH;       // SCLK
218:у спокої має високий рівень
219:  hspi2.Init.CLKPhase = SPI_PHASE_2EDGE;            // Фік
220:сація даних за другим фронтом (переднім)
221:  /* Програмне керування NSS (Chip Select) через GPIO */
222:  hspi2.Init.NSS = SPI_NSS_SOFT;
223:  /* Налаштування швидкості: підбирається дільником (Bau
224:dRatePrescaler)
225:     для досягнення цільової швидкості близько 10 Мбіт/с
226:залежно від частоти шини APB */
227:  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
228:  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;           // Пер
229:шим передається старший біт (MSB First)
230:  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;           // Ви
231:мкнено режим TI (Motorola SPI)
```

```
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 7;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {       Error_Handler();   }   }
/**
  * @brief Конфігурація GPIO виходів для трьох ліній CS
(CS1, CS2, CS3)
  */
void MX_GPIO_Init(void)  {
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  // 1. Ввімкнення тактування порту GPIOA
  __HAL_RCC_GPIOA_CLK_ENABLE();
  /* 2. Попереднє виставлення високого рівня (1) на виводах
CS перед їх ініціалізацією,
     щоб ведені пристрої на шині випадково не активувалися
під час старту MCU */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0 | GPIO_PIN_1 |
GPIO_PIN_2, GPIO_PIN_SET);
  // 3. Налаштування параметрів виводів PA0, PA1, PA2 як
виходів Push-Pull
  GPIO_InitStruct.Pin = GPIO_PIN_0 | GPIO_PIN_1 |
GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;        // Дво
тактний вихід (Push-Pull)
  GPIO_InitStruct.Pull = GPIO_NOPULL;                // Без
внутрішніх підтяжок (зовнішні підтяжки бажані)
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;      // Ви
сока швидкість наростання фронтів
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);   }
/**
  * @brief Функція активації вибраного веденого пристрою
(CS -> 0)
  * @param cs_num: Номер пристрою (1, 2 або 3)    */
void spi_select(uint8_t cs_num)
{   switch (cs_num)   {
    case 1: HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0,
GPIO_PIN_RESET); break; // CS1 (PA0) -> 0
    case 2: HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1,
GPIO_PIN_RESET); break; // CS2 (PA1) -> 0
    case 3: HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2,
GPIO_PIN_RESET); break; // CS3 (PA2) -> 0
    default: break;    } }
/**   * @brief Функція деактивації вибраного веденого
пристрою  (CS -> 1)
  * @param cs_num: Номер пристрою (1, 2 або 3)   */
void spi_deselect(uint8_t cs_num)
{  switch (cs_num)  {
    case 1: HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0,
GPIO_PIN_SET); break;   // CS1 (PA0) -> 1
    case 2: HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1,
GPIO_PIN_SET); break;   // CS2 (PA1) -> 1
    case 3: HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2,
GPIO_PIN_SET); break;   // CS3 (PA2) -> 1
    default: break;  } }
/** @brief Програмна затримка для виконання таймінгів t_CSS
та t_CSH.
  * Кількість асемблерних інструкцій NOP залежить від робо
```

```
289:чої частоти ядра MCU.
290:  */
291:void delay_ns_or_nop(void)
292:{   __NOP(); __NOP(); __NOP(); __NOP();   }
```

У підсумку, SPI завдяки простоті, високій швидкості та підтримці багатопристроєвості через гнучке керування CS залишається універсальним інтерфейсом для внутрішньосхемного зв'язку. Наступні підрозділи розглянуть розширені конфігурації SPI, такі як Dual SPI та Quad SPI, що використовуються для значного збільшення пропускної здатності під час роботи із зовнішньою пам'яттю, а також наведуть результати порівняльних випробувань продуктивності на основі даних виробників.

3.2.2. Розширені режими: Dual SPI, Quad SPI (QSPI) для зовнішньої пам'яті

Класичний однобітовий SPI, розглянутий у попередніх підрозділах, оперує єдиною лінією даних у кожному напрямку (MOSI та MISO), що обмежує його пропускну здатність добутком тактової частоти на один біт за такт. Для більшості низькошвидкісних датчиків, АЦП і цифрових потенціометрів такої швидкості цілком достатньо. Однак зі зростанням вимог до обсягу та швидкості доступу до зовнішньої енергонезалежної пам'яті, передусім NOR Flash, що використовується для зберігання програмного коду (Execute-in-Place, XIP), графічних ресурсів, файлових систем та журналів подій, однобітовий SPI перетворюється на вузьке місце. Саме для подолання цього обмеження провідні виробники флеш-пам'яті, зокрема Macronix, Winbond, ISSI, Cypress (Infineon) та Micron, у 2010-х роках запровадили розширені багатобітові режими — Dual SPI (2-бітовий) та Quad SPI (4-бітовий), — які згодом були стандартизовані та отримали апаратну підтримку в периферійних модулях більшості мікроконтролерів.

Фундаментальна ідея Dual SPI та Quad SPI полягає в динамічному перепризначенні функцій сигнальних ліній упродовж однієї транзакції. У класичному SPI лінії MOSI та MISO виконують суворо фіксовані ролі: одна — завжди вихід майстра, друга — завжди вхід. У багатобітових режимах лінії перетворюються на двонапрямлені (half-duplex), і в фазі передавання адреси або даних працюють як паралельна група, передаючи (або приймаючи) по 2 або 4 біти одночасно за один такт SCLK. Це дозволяє збільшити пропускну здатність удвічі (Dual SPI) або вчетверо (Quad SPI) порівняно з класичним SPI на тій самій тактовій частоті без збільшення кількості виво-

дів мікросхеми пам'яті. Важливо, що багатобітові режими не є обов'язковими для всіх транзакцій: команда (opcode) зазвичай передається в однобітовому режимі, щоб мікросхема могла її розпізнати, після чого спеціальними командами (наприклад, 0x38 для переходу в QPI-режим) активується багатобітовий обмін для адреси та даних. Така гнучкість дає змогу зберегти сумісність із класичним SPI на етапі ініціалізації та водночас досягти максимальної продуктивності під час читання чи запису великих блоків.

Фізична організація ліній для трьох режимів — Standard SPI, Dual SPI та Quad SPI — схематично показана на рисунку 3.8. У стандартному режимі задіяно лінії CS#, SCLK, SO (Serial Output, аналог MISO) та SI (Serial Input, аналог MOSI). У Dual SPI лінії SO та SI перетворюються на двонапрямлені IO0 та IO1, які в фазі передавання адреси/даних від майстра працюють як виходи, а в фазі читання — як входи. У Quad SPI додатково залучаються лінії WP# (Write Protect) та HOLD# (Hold), які в цьому режимі функціонують як IO2 та IO3, утворюючи 4-бітову шину даних. Специфікації на конкретні мікросхеми флеш-пам'яті, як-от W25Q128JV від Winbond (документ W25Q128JV Rev. K, 2020) або MX25L6433F від Macronix (документ MX25L6433F Rev. 1.8, 2021), детально описують, які команди підтримують Dual/Quad режими, які біти конфігураційного регістра необхідно встановити для активації цих режимів і які часові параметри слід дотримувати. Зокрема, для переходу в Quad SPI найчастіше використовують команду 0x35 (Quad Enable, QE) для встановлення біта QE в регістрі статусу, після чого стають доступними команди Quad Read (0x6B, 0xEB, 0xE7 залежно від частоти та кількості dummy-циклів).

Кількісне порівняння пропускної здатності для Standard SPI, Dual SPI та Quad SPI на тактовій частоті 100 МГц, що є типовою для сучасних мікросхем флеш-пам'яті, наведено в таблиці 3.6. З неї видно, що перехід до Quad SPI дозволяє досягти швидкості читання 50 МБ/с, що співставно з продуктивністю паралельних шин, але при значно меншій кількості виводів. Саме тому QSPI став де-факто стандартом для зовнішньої пам'яті у вбудованих системах, а згодом еволюціонував у OctoSPI (8-бітовий) для досягнення швидкостей понад 100 МБ/с, що реалізовано, наприклад, у мікроконтролерах STM32L4+, STM32H7 та i.MX RT.

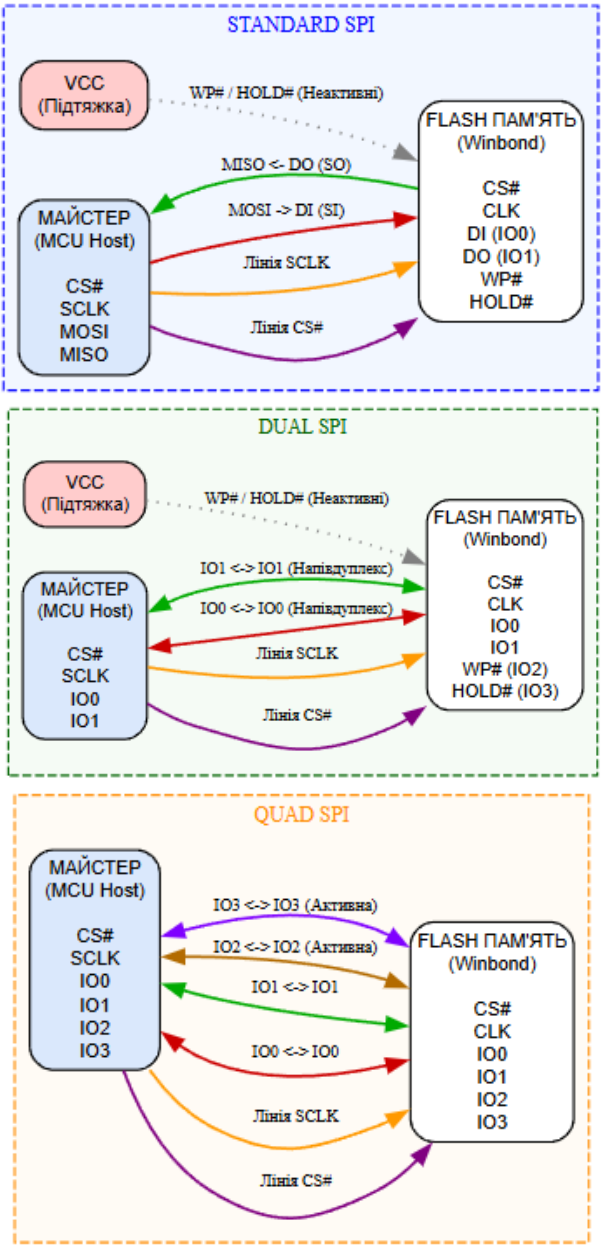


Рисунок 3.8 – Фізична організація ліній Standard SPI, Dual SPI та Quad SPI

Примітка: у багатобітових режимах лінії IO0–IO3 використовуються для одночасної передачі/прийому 2 або 4 бітів за такт.

Таблиця 3.6 – Порівняння пропускної здатності режимів SPI для зовнішньої флеш-пам’яті на частоті 100 МГц

Режим	Характеристика	Значення
Standard SPI	*Кількість ліній даних*	1 (MOSI/MISO)
	Бітів за такт	1
	Максимальна швидкість передавання даних (МБ/с)	12,5 (100 Мбіт/с)
	Типова команда читання	0x03 (Read)
	Примітка	Підтримується всіма мікросхемами
Dual SPI	*Кількість ліній даних*	2 (IO0, IO1)
	Бітів за такт	2
	Максимальна швидкість передавання даних (МБ/с)	25
	Типова команда читання	0xBB (Dual Read)
	Примітка	Потребує встановлення біта QE
Quad SPI	*Кількість ліній даних*	4 (IO0–IO3)
	Бітів за такт	4
	Максимальна швидкість передавання даних (МБ/с)	50
	Типова команда читання	0xEB (Quad Read)
	Примітка	Стандарт для XIP-застосувань
OctoSPI	*Кількість ліній даних*	8 (IO0–IO7)
	Бітів за такт	8
	Максимальна швидкість передавання даних (МБ/с)	100
	Типова команда читання	0xEE (Octo Read)
	Примітка	Підтримується STM32H7, i.MX RT

Примітка: фактична швидкість читання може бути нижчою через наявність dummy-циклів, необхідних після передавання адреси, а також через затримки встановлення режиму та особливості конкретного кристалу пам’яті.

Окремо слід відзначити роль так званих dummy-циклів (режим очікування), які є невід’ємною частиною багатобітових протоколів читання. Після передавання команди та адреси в багатобітовому режимі мікросхема флеш-пам’яті потребує певного часу для доступу до масиву комірок пам’яті (initial latency), упродовж якого майстер продовжує подавати тактові імпульси, але лінії даних перебувають у невизначеному стані. Кількість dummy-циклів залежить від частоти SCLK і конкретної мікросхеми:

для W25Q128JV на частоті 104 МГц потрібно 8 dummy-циклів, на частоті 133 МГц — 10 dummy-циклів. Це означає, що частина тактових імпульсів витрачається не на передавання корисних даних, а на очікування, і цей overhead має бути врахований під час обчислення реальної пропускної здатності для дрібних транзакцій. Для великих послідовних читань (наприклад, потокове зчитування образу прошивки) вплив dummy-циклів є незначним, оскільки вони становлять фіксовану затримку на початку операції, після чого дані надходять безперервно. Цю особливість ілюструє рисунок 3.9, де наведено часову діаграму команди Quad Read (0xEB) для мікросхеми W25Q128JV, із позначенням фаз команди, адреси, dummy-циклів і даних.

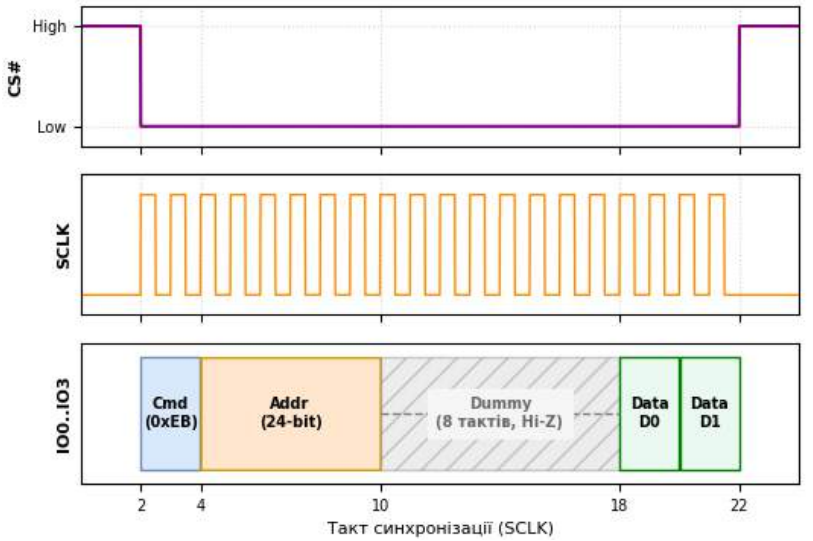


Рисунок 3.9 – Часова діаграма команди Quad Read (0xEB) для W25Q128JV

Апаратна підтримка Quad SPI в мікроконтролерах реалізована через спеціалізовані периферійні модулі, які виходять за межі звичайного SPI. У STM32 сімейств L4+, F7, H7 та інших це модуль QUADSPI, описаний у відповідних довідкових посібниках (наприклад, RM0433 для STM32H723, розділ 21). Він підтримує Standard, Dual та Quad режими, має апаратний скінченний автомат для генерації фаз команди, адреси, альтернативного байта та dummy-циклів, а також здатен працювати в режимі прямого відображення пам’яті (Memory-Mapped Mode). Останній дозволяє процесору звертатися до зовнішньої флеш-пам’яті через звичайні інструкції читання за певною адресою в адресному просторі (наприклад, 0x90000000), а модуль QUADSPI прозоро перетворює ці звернення в Quad SPI-транзакції,

включаючи автоматичну вставку dummy-циклів. Це відкриває можливість виконання коду безпосередньо із зовнішньої пам'яті (XIP) без копіювання в ОЗП, що є критичним для систем із великим обсягом прошивки. Для мікроконтролерів ESP32 компанія Espressif реалізувала аналогічний механізм через модулі SPI0/SPI1 (до 80 МГц, Quad SPI), які використовуються для доступу до зовнішнього flash і PSRAM, а в ESP32-S3 додано модуль SPI2 із підтримкою Octal PSRAM. У Raspberry Pi Pico (RP2040) апаратної підтримки Quad SPI немає, однак гнучкі блоки PIO дозволяють емулювати Dual/Quad протоколи програмно, хоча з обмеженою швидкістю через відсутність апаратного автомата команд.

У лістингу 3.5 наведено фрагмент коду для ініціалізації модуля QUADSPI на STM32H723 у режимі Memory-Mapped для роботи з мікросхемою W25Q128JV, а також демонструється, як після налаштування зовнішня пам'ять стає доступною за вказівником як звичайний масив. Цей код спирається на рекомендації виробника, викладені в прикладній нотатці AN4760 «Quad-SPI interface on STM32 microcontrollers» (STMicroelectronics, 2021).

Лістинг 3.5 – Ініціалізація QUADSPI в режимі Memory-Mapped для W25Q128JV на STM32H723 (HAL)

```
/* --- ВИРІЗКА З ГОЛОВНОГО ФАЙЛУ ПРОГРАМИ (main.c) --- */
int main(void)
{
  /* [Етап 1] Базова ініціалізація системи та тактування */
  HAL_Init();
  SystemClock_Config(); // HCLK = 200 MHz
  // Конфігурація QUADSPI: Prescaler = 1 дає 100 MHz (200 MHz / (1 + 1))
  hqspi.Instance = QUADSPI;
  hqspi.Init.ClockPrescaler = 1;
  hqspi.Init.FlashSize = 23;// 2^24 = 16 Мегабайт (W25Q128JV)
  hqspi.Init.SampleShifting=QSPI_SAMPLE_SHIFTING_HALFCYCLE;
  if (HAL_QSPI_Init(&hqspi) != HAL_OK) { Error_Handler(); }
  /* [Етап 2] Активація Quad-режиму всередині самої мікросхеми Flash.
      Без цього кроку чип Winbond буде ігнорувати 4-бітні команди контролера */
  QSPI_CommandTypeDef cmd = {0};
  uint8_t status_reg2 = 0x02; // Біт QE (Quad Enable) у регістрі статусу-2
  cmd.Instruction = 0x31;//Команда:"Write Status Register-2"
  cmd.InstructionMode = QSPI_INSTRUCTION_1_LINE;
  cmd.DataMode = QSPI_DATA_1_LINE;
  cmd.NbData = 1;
  HAL_QSPI_Command(&hqspi, &cmd, HAL_MAX_DELAY);
  HAL_QSPI_Transmit(&hqspi, &status_reg2, HAL_MAX_DELAY);

  /* [Етап 3] Конфігурація та перехід у безперервний режим
```

```
29:Memory-Mapped. Формуємо команду 0xEB (Fast Read Quad I/O) з
30:8 тактами очікування */
31:  QSPI_MemoryMappedTypeDef cfg = {0};
32:  cmd.Instruction = 0xEB; // Команда швидкого Quad-читання
33:  cmd.InstructionMode = QSPI_INSTRUCTION_1_LINE;
34:  cmd.AddressMode = QSPI_ADDRESS_4_LINES;
35:  cmd.AddressSize = QSPI_ADDRESS_24_BITS;
36:  cmd.DataMode = QSPI_DATA_4_LINES;
37:  cmd.DummyCycles = 8;    // 8 dummy-циклів згідно з даташитом
38:Winbond
39:  cfg.TimeOutActivation = QSPI_TIMEOUT_COUNTER_DISABLE;
40:// Активація апаратного перехоплення запитів до пам'яті (XIP)
41:  if (HAL_QSPI_MemoryMapped(&hqspi, &cmd, &cfg) != HAL_OK) {
42:Error_Handler(); }
43:  /* ================================================== */
44:  /*  ВІДТЕПЕР FLASH ПІДКЛЮЧЕНА НАПРЯМУ ДО АДРЕСНОГО ПРОСТОРУ
45:ЯДРА CORTEX-M7   */
46:  /* ================================================== */
47:  // Покажчик на нульову адресу QUADSPI у STM32H7
48:  uint8_t *ext_flash = (uint8_t *)0x90000000;
49:  // Пряме зчитування байта за допомогою стандартної операції
50:мови C
51:  uint8_t first_byte = ext_flash[0];
52:  while (1)
53:  {
54:    /* НАГОЛОШЕННЯ ДЛЯ СТУДЕНТІВ: Для промислового викори
55:стання цього режиму необхідно налаштувати модуль MPU (Memory
56:Protection Unit). Без увімкнення M_Attr або повної заборони
57:спекулятивного читання (Speculative Read) конвеєр ядра
58:Cortex-M7 може згенерувати помилку BusFault/HardFault. */
59:  }    }
```

Завершуючи розгляд розширених режимів SPI, варто підкреслити, що їх поява є яскравим прикладом того, як тиск з боку прикладних вимог (швидке завантаження графічних інтерфейсів, XIP для великих RTOS, відеопотоки) стимулює еволюцію, здавалося б, усталеного протоколу без зміни його базових принципів. Dual SPI та Quad SPI не замінюють класичний SPI, а доповнюють його в тих нішах, де пропускна здатність стає пріоритетною, зберігаючи при цьому ту саму модель «майстер — ведений» і той самий сигнал CS# для вибору пристрою. Студент, який зрозумів цю еволюційну логіку, зможе не лише грамотно застосовувати QSPI у своїх проєктах, але й передбачити появу нових багатобітових протоколів, таких як OctoSPI або навіть 16-бітовий HyperBus, що вже знаходять застосування в автомобільних системах доповненої реальності та швидкісних реєстраторах даних. Наступний підрозділ, присвячений порівняльним випробуванням продуктивності, надасть кількісне підґрунтя для вибору між цими режимами в конкретних інженерних задачах.

3.2.3. Порівняльні випробування продуктивності за матеріалами виробників (Microchip, ST)

Теоретичний аналіз пропускної здатності SPI, наведений у попередніх підрозділах, дає лише верхню межу, якої можна досягти за ідеальних умов. Реальна продуктивність інтерфейсу в конкретній мікроконтролерній системі залежить від сукупної дії багатьох чинників: архітектури периферійного модуля, продуктивності контролера DMA, пріоритетів шинної матриці, латентності обробки переривань і навіть версії використовуваної бібліотеки (HAL проти LL). З метою надати розробникові об'єктивні орієнтири, провідні виробники мікроконтролерів періодично публікують результати власних порівняльних випробувань у формі прикладних нотаток (Application Notes). У цьому підрозділі систематизовано та проаналізовано дані, отримані з двох авторитетних джерел: прикладної нотатки AN4031 «Using the STM32F2, STM32F4 and STM32F7 Series DMA controller» (STMicroelectronics, 2019) та технічного бюлетеня TB3265 «QSPI Performance on SAM E70/S70/V70/V71 Microcontrollers» (Microchip Technology, 2021). Додатково залучено результати, опубліковані в технічній документації на модуль QUADSPI (AN4760, STMicroelectronics, 2021), що дозволяє провести наскрізне порівняння класичного та розширених режимів SPI на платформах різних виробників.

Методологія випробувань, застосована в AN4031, передбачала вимірювання максимальної пропускної здатності (throughput) у кілобайтах на секунду та відсотка завантаження центрального процесора (CPU load) для трьох режимів обміну через SPI: опитування (polling), переривання (interrupt) і прямого доступу до пам'яті (DMA). Вимірювання виконувалися на мікроконтролері STM32F417 із тактовою частотою ядра 168 МГц і частотою SPI 42 МГц (максимальна для APB1). Обсяг переданих даних становив 8192 байти, що відповідає типовому розміру сторінки зовнішньої флеш-пам'яті. Результати, наведені в оригінальному документі, показують, що режим опитування досягає пропускної здатності близько 4,1 МБ/с, однак при цьому завантаження CPU сягає 100 відсотків, оскільки процесор не виконує жодної корисної роботи, окрім циклічного очікування прапорців статусу. Режим переривання забезпечує дещо нижчу пропускну здатність — близько 3,5 МБ/с, проте CPU завантажений лише на 65–70 відсотків, що вже дозволяє паралельно виконувати інші задачі з низьким пріоритетом. Найкращі показники демонструє режим DMA: пропускна здатність зростає до 5,2 МБ/с (що практично дорівнює теоретичній межі 42/8 = 5,25 МБ/с для однобітового SPI), а завантаження CPU падає нижче 1 відсотка, оскільки

процесор лише ініціює передачу та отримує сповіщення після її завершення. Ці дані узагальнено в таблиці 3.7, яка наочно демонструє, чому DMA є безальтернативним вибором для будь-яких високошвидкісних або фонових обмінів через SPI.

Таблиця 3.7 – Порівняння продуктивності режимів SPI на STM32F417 (SPI на 42 МГц, 8192 байти)

Режим	Характеристика	Значення
Polling	*Пропускна здатність (МБ/с)*	4.1
	Завантаження CPU (%)	100
	Ефективність використання каналу (%)	78
	Примітка	Блокує ядро повністю
Interrupt	*Пропускна здатність (МБ/с)*	3,5
	Завантаження CPU (%)	~68
	Ефективність використання каналу (%)	67
	Примітка	Втрати на обробку переривань
DMA	*Пропускна здатність (МБ/с)*	5,2
	Завантаження CPU (%)	<1
	Ефективність використання каналу (%)	99
	Примітка	Близько до теоретичної межі

Примітка: теоретична максимальна пропускна здатність для однобітового SPI на частоті 42 МГц становить 42/8 = 5,25 МБ/с. Дані адаптовано з AN4031, STMicroelectronics. Ефективність використання каналу розраховано як відношення виміряної пропускної здатності до теоретичної межі.

Проте навіть у режимі DMA класичний однобітовий SPI наштовхується на фундаментальне обмеження: пропускна здатність прямо пропорційна тактовій частоті. Тому закономірним є питання, як змінюється продуктивність зі зростанням частоти SCLK і чи досягається лінійне масштабування. Для відповіді на нього доцільно звернутися до матеріалів Microchip, а саме TB3265, де наведено результати вимірювань пропускної здатності модуля QSPI на мікроконтролері SAM E70 (ядро ARM Cortex-M7, 300 МГц) для різних тактових частот. Випробування проводилися в режимі Quad SPI під час читання зовнішньої флеш-пам’яті SST26VF064B. Результати свідчать, що за частоти 50 МГц досягається швидкість читання 24,8 МБ/с, за 100 МГц — 49,5 МБ/с, а на максимальній частоті 150 МГц (граничній для конкретної мікросхеми пам’яті) —

74,3 МБ/с. Примітно, що масштабування є практично лінійним, а невелике відхилення від ідеалу (75 МБ/с для 150 МГц) пояснюється наявністю dummy-циклів, які не переносять корисних даних. Аналогічні випробування, виконані STMicroelectronics для модуля QUADSPI на STM32H723 і описані в AN4760, показують зіставні результати: 50,1 МБ/с на частоті 100 МГц у режимі Quad SPI для пам'яті W25Q128JV. Залежність пропускної здатності від тактової частоти для обох платформ ілюструється рисунком 3.10, який демонструє, що продуктивність лінійно зростає доти, доки частота не наближається до границь, визначених характеристиками мікросхеми пам'яті або обмеженнями виводів мікроконтролера. Обидві платформи демонструють лінійне масштабування; нахил кривих визначається кількістю бітів за такт (4 для Quad SPI)

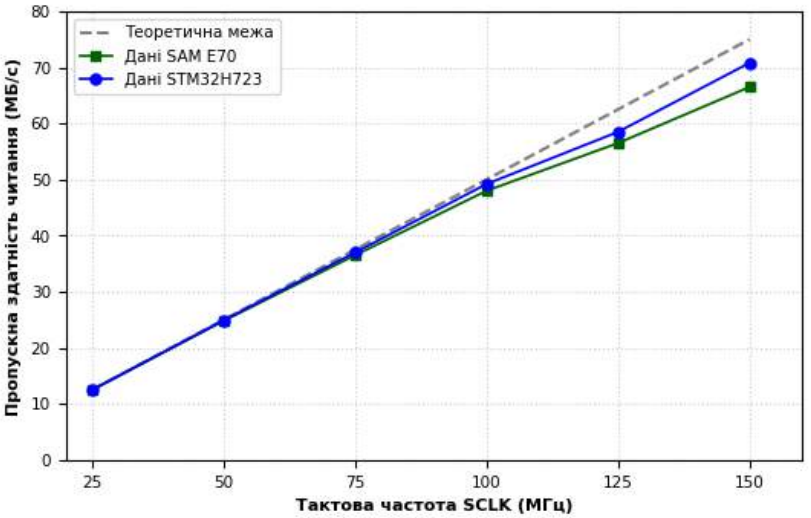


Рисунок 3.10 – Графік залежності пропускної здатності Quad SPI від тактової частоти для STM32H723 (AN4760) та SAM E70 (TB3265)

Окремий інтерес становить порівняння продуктивності не лише в абсолютних цифрах, а й з урахуванням енергоспоживання, що є критичним для портативних IoT-пристроїв. У прикладній нотатці AN4891 «SPI performance and power consumption on STM32L4 Series» (STMicroelectronics, 2020) наведено вимірювання струму споживання мікроконтролера під час передавання через SPI у режимах DMA та переривання. Для STM32L476 на частоті ядра 80 МГц і SPI 40 МГц струм у режимі

DMA становив 4,2 мА, тоді як у режимі переривання — 5,8 мА, що пояснюється додатковими витратами енергії на обробку кожного байта в обробнику переривання. Таким чином, DMA не лише розвантажує процесор, а й знижує енергоспоживання на одиницю переданої інформації, що робить його подвійно вигідним для пристроїв із батарейним живленням.

У лістингу 3.6 наведено спрощений код, який може бути використаний студентом для самостійного відтворення випробувань пропускної здатності SPI на платформі STM32. Він використовує таймер для вимірювання часу передавання великого блоку даних через DMA та обчислює досягнуту швидкість. Подібний підхід дає змогу не лише перевірити заявлені виробником характеристики, а й оцінити вплив власного коду та конфігурації системи на продуктивність інтерфейсу.

Лістинг 3.6 – Вимірювання пропускної здатності SPI у режимі DMA на STM32 (HAL)

```
/* --- ВИРІЗКА З ГОЛОВНОГО ФАЙЛУ ПРОГРАМИ (main.c) --- */
#include "main.h"
// Глобальні змінні для синхронізації DMA переривань
extern SPI_HandleTypeDef hspi1;
extern TIM_HandleTypeDef htim2; // Таймер з роздільною здатністю 1 мкс
volatile uint8_t dma_tx_complete = 0;
/* ... [Тут знаходиться автогенерований код ініціалізації HAL, SPI та TIM] ... */
/**
  * @brief Лістинг 3.6: Вимірювання пропускної здатності SPI у режимі DMA  */
void test_spi_throughput(void)
{
  #define BUFFER_SIZE 8192
  static uint8_t tx_buffer[BUFFER_SIZE]; // Буфер для тестування (виділений у RAM)
  uint32_t start_time = 0, end_time = 0;
  float elapsed_seconds = 0.0f;
  float throughput_mbps = 0.0f;
  // Заповнюємо тестовий буфер паттерном
  for(uint32_t i = 0; i < BUFFER_SIZE; i++) { tx_buffer[i] = (uint8_t)i; }
  dma_tx_complete = 0;
/*[Крок 1]Фіксація початкового часу та запуск транзакції */
  start_time = __HAL_TIM_GET_COUNTER(&htim2); // Старт апаратного таймера (1 мкс = 1 тік)
  if (HAL_SPI_Transmit_DMA(&hspi1, tx_buffer, BUFFER_SIZE) != HAL_OK)
  {    Error_Handler();  }
  /*[Крок 2]Очікування завершення DMA-транзакції (апаратний семафор) */
  while (dma_tx_complete == 0)
  {  // Очікуємо переривання від DMA. В реальних ОС тут використовується osSemaphoreWait  }
```

```
  /*[Крок 3]Фіксація кінцев. часу та обчисл. результатів */
    end_time = __HAL_TIM_GET_COUNTER(&htim2);
    // Розрахунок часу транзакції з урахуванням переповнення
таймера
    uint32_t total_ticks = (end_time >= start_time) ?
(end_time - start_time) : (0xFFFFFFFF - start_time +
end_time);
    elapsed_seconds = (float)total_ticks / 1000000.0f;
// Перевід мікросекунд у секунди
    // Формула: Throughput = (розмір буфера * 8 біт) /
(час_в_секундах * 1 000 000)
    throughput_mbps = ((float)BUFFER_SIZE * 8.0f) /
(elapsed_seconds * 1000000.0f);
    /* НАГОЛОШЕННЯ ДЛЯ СТУДЕНТІВ: Отримане значення є
«чистою» швидкістю передачі даних.
       Для точного аналізу слід враховувати, що реальна
пропускна здатність шини трохи нижча,
       оскільки апаратний таймер також фіксує накладні витрати
(Overhead): час на виклик
       функції HAL SPI Transmit DMA(), ініціалізацію регістрів
контролера DMA, а також
       вхід/вихід з ISR-обробника переривань. */
    printf("SPI DMA Throughput: %.2f Mbit/s\r\n",
throughput_mbps);
}
/**  * @brief Переривання: викликається автоматично після
завершення передачі всього буфера  */
void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *huart)
{
  if (huart->Instance == SPI1)
  {
    dma_tx_complete = 1; // Звільняємо семафор/прапорець
очікування
  }}

int main(void)
{  HAL_Init();  SystemClock_Config();
  /* ... [Ініціалізація MX GPIO Init, MX DMA Init,
MX_SPI1_Init, MX_TIM2_Init] ... */
  HAL_TIM_Base_Start(&htim2); // Запуск таймера метрик
  // Виклик вимірювання пропускної здатності
  test_spi_throughput();
  while (1)
  {    // Основний цикл програми  }   }
```

Підсумовуючи наведені порівняльні дані, можна зробити кілька практичних висновків.

По-перше, режим DMA є обов'язковим для будь-яких застосувань, де обсяг переданих даних перевищує кілька десятків байтів або де потрібна фонова передача без блокування ядра.

По-друге, перехід від класичного SPI до Quad SPI дає чотирикратний виграш у пропускній здатності на тій самій тактовій частоті, що є критичним для роботи із зовнішньою пам'яттю у складі графічних інтерфейсів або реєстраторів даних.

По-третє, продуктивність SPI-модулів на сучасних мікроконтролерах від STMicroelectronics та Microchip є зіставною і досягає теоретичної межі за умови використання DMA, тому вибір між платформами має визначатися радше екосистемою, вартістю та додатковими периферійними модулями, ніж суто швидкістю SPI.

Ці результати слугуватимуть надійним орієнтиром під час вибору конфігурації SPI для конкретних задач, які розглядатимуться в наступних розділах посібника.

3.3. I²C (Inter-Integrated Circuit): адресація, арбітраж, розтягування такту

Шина I²C (Inter-Integrated Circuit), розроблена компанією Philips Semiconductors на початку 1980-х років і нині підтримувана NXP Semiconductors, посідає унікальне місце серед послідовних інтерфейсів завдяки поєднанню надзвичайної простоти фізичного рівня — лише два провідники із загальним зворотним проводом — та розвиненої багатомайстерної архітектури з апаратною адресацією і детектуванням колізій. Фізично шина складається з двох ліній: SDA (Serial Data) для даних і SCL (Serial Clock) для тактового сигналу, обидві підтягнуті до напруги живлення через резистори, що забезпечує логіку «монтажного І» (wired-AND). Кожен пристрій на шині має унікальну адресу, яка може бути 7- або 10-бітною, і саме за цією адресою майстер ініціює обмін, передаючи її після стартової умови. Специфікація I²C-bus specification and user manual (NXP UM10204, rev. 7.0, 2021) є нормативним документом, що визначає всі електричні та часові параметри, протокольні стани і процедури. На відміну від SPI, де майстер одноосібно керує тактовим сигналом, у I²C ведений пристрій може утримувати лінію SCL у низькому стані після отримання байта, щоб виграти час на обробку даних — цей механізм зветься розтягуванням такту (clock stretching) і є обов'язковим для підтримки в усіх сумісних пристроях. Арбітраж у I²C відбувається на рівні бітів: якщо два майстри одночасно починають передавання, той із них, хто першим виставить низький рівень на SDA тоді, коли інший намагається втримати високий, програє арбітраж і негайно припиняє передавання, не спотворюючи сигнал переможця. Саме ці властивості — мала кількість виводів, апаратна адресація, багатомайстерність і

стійкий до колізій арбітраж — зробили I²C домінуючим інтерфейсом для підключення датчиків, годинників реального часу, мікросхем пам'яті EEPROM, розширювачів портів та іншої низькошвидкісної периферії, особливо в мобільних, автомобільних і IoT-застосуваннях.

3.3.1. Стандартний (100 кбіт/с), швидкий (400 кбіт/с), Fast+ (1 Мбіт/с) та Ultra Fast (5 Мбіт/с) режими

Однією з найважливіших характеристик I²C є наявність кількох стандартизованих швидкісних режимів, які дають розробникові змогу масштабувати пропускну здатність шини відповідно до вимог системи, не змінюючи базовий протокол. Початкова версія специфікації визначала лише Standard-mode з максимальною тактовою частотою 100 кбіт/с, що було більш ніж достатньо для керування телевізійними тюнерами та аудіопроцесорами, для яких власне і створювався інтерфейс. Проте вже у 1992 році з'явився Fast-mode (400 кбіт/с), потім Fast-mode Plus (1 Мбіт/с), а згодом і Ultra Fast-mode (5 Мбіт/с), призначений для специфічних застосувань із однонапрямленим передаванням. Усі ці режими, включно з High-speed mode (3,4 Мбіт/с), детально описані в документі NXP UM10204, на який і спиратиметься подальший виклад.

Standard-mode (Sm) характеризується тактовою частотою SCL до 100 кГц. Часові параметри цього режиму є найбільш поблажливими: мінімальна тривалість низького рівня SCL становить 4,7 мкс, високого — 4,0 мкс, час наростання сигналу може сягати 1000 нс, а максимальна ємність шини — 400 пФ. Такі допуски дозволяють використовувати порівняно високоомні підтягувальні резистори (типово 4,7 кОм для 5-вольтової логіки) і не пред'являють жорстких вимог до трасування плати, що робить Standard-mode ідеальним для макетування, навчальних лабораторних робіт та простих пристроїв із невисокими вимогами до швидкості обміну. Переважна більшість мікросхем I²C-периферії підтримує Standard-mode, що гарантує сумісність у будь-якій конфігурації.

Fast-mode (Fm) підвищує максимальну частоту SCL до 400 кГц, скорочуючи мінімальну тривалість низького рівня до 1,3 мкс, а високого — до 0,6 мкс. Час наростання сигналу обмежено 300 нс, що вимагає зниження опору підтягувальних резисторів або застосування активних підтяжок (current source pull-up) для дотримання фронтів. Ємність шини при цьому не повинна перевищувати 400 пФ, але за меншої ємності можна використ-

товувати слабшу підтяжку. Саме Fast-mode є робочим стандартом для більшості сучасних датчиків (інерційних, тиску, температури), оскільки забезпечує прийнятний баланс між швидкістю та енергоспоживанням.

Fast-mode Plus (Fm+) став відповіддю на потребу в ще вищій пропускній здатності без переходу до складнішого High-speed режиму. Його ключові параметри: частота SCL до 1 МГц, мінімальна тривалість низького рівня 0,5 мкс, час наростання до 120 нс, а найголовніше — збільшена до 30 мА сила струму драйвера (порівняно з 3 мА для Standard і Fast), що дозволяє заряджати ємність шини до 550 пФ без надмірного сповільнення фронтів. Завдяки цьому Fm+ придатний для багатовузлових мереж на одній платі або для зв'язку через короткі кабелі. Варто зауважити, що Fm+ є електрично сумісним зі Standard і Fast згори донизу: пристрій Fm+ може працювати на шині з меншою швидкістю, але для досягнення 1 Мбіт/с усі пристрої на шині повинні підтримувати цей режим.

Ultra Fast-mode (UFm) є найменш поширеним, однак важливим для повноти огляду. Його специфікація передбачає однонапрямлене передавання від майстра до ведених із частотою SCL до 5 МГц, причому лінія SCL є push-pull виходом, а не відкритим стоком, що унеможливлює розтягування такту та багатомайстерний арбітраж. Цей режим використовується переважно в моніторах, телевізорах та інших застосуваннях, де потрібне швидке конфігурування великої кількості периферійних мікросхем без зворотного зв'язку. Оскільки UFm не сумісний із класичною двоспрямованою архітектурою I²C, у цьому посібнику він згадується як приклад спеціалізованого розширення стандарту.

Порівняння основних параметрів усіх чотирьох режимів наведено в таблиці 3.8, яка унаочнює поступове зростання швидкості ціною жорсткіших вимог до фронтів і драйверів. Дані взято безпосередньо з офіційної специфікації NXP UM10204 (rev. 7.0, таблиці 5, 6, 7, 8).

Вибір конкретного швидкісного режиму в реальному проєкті визначається не лише бажаною пропускною здатністю, а й фізичними обмеженнями конкретної плати. Головним ворогом високих швидкостей є паразитна ємність доріжок і виводів мікросхем, яка в сумі з опором підтягувального резистора утворює RC-коло, що обмежує швидкість наростання фронту.

Для наближеного розрахунку максимального опору підтягувального резистора R_p, який забезпечує необхідний час наростання t_r, застосовують формулу (3.1):

$$R_p(max) = t_r/(0{,}8473 * C_b) \ , \qquad (3.1)$$

Таблиця 3.8 – Параметри швидкісних режимів I²C згідно з NXP UM10204 rev. 7.0

Режим	Характеристика	Значення
Standard-mode (Sm)	*Максимальна частота SCL*	100 кГц
	Мінімальна тривалість низького рівня SCL	4,7 мкс
	Мінімальна тривалість високого рівня SCL	4,0 мкс
	Максимальний час наростання сигналу	1000 нс
	Максимальна ємність шини	400 пФ
	Сила струму драйвера (типова)	3 мА
	Сумісність із clock stretching	так
	Тип виходу	відкритий стік
Fast-mode (Fm)	*Максимальна частота SCL*	400 кГц
	Мінімальна тривалість низького рівня SCL	1,3 мкс
	Мінімальна тривалість високого рівня SCL	0,6 мкс
	Максимальний час наростання сигналу	300 нс
	Максимальна ємність шини	400 пФ
	Сила струму драйвера (типова)	3 мА
	Сумісність із clock stretching	так
	Тип виходу	відкритий стік
Fast-mode Plus (Fm+)	*Максимальна частота SCL*	1 МГц
	Мінімальна тривалість низького рівня SCL	0,5 мкс
	Мінімальна тривалість високого рівня SCL	0,26 мкс
	Максимальний час наростання сигналу	120 нс
	Максимальна ємність шини	550 пФ
	Сила струму драйвера (типова)	30 мА
	Сумісність із clock stretching	так
	Тип виходу	відкритий стік
Ultra Fast-mode (UFm)	*Максимальна частота SCL*	5 МГц
	Мінімальна тривалість низького рівня SCL	0,1 мкс
	Мінімальна тривалість високого рівня SCL	0,07 мкс
	Максимальний час наростання сигналу	25 нс
	Максимальна ємність шини	Визначається драйвером
	Сила струму драйвера (типова)	30 мА
	Сумісність із clock stretching	ні
	Тип виходу	push-pull

де C_b — сумарна ємність шини, а коефіцієнт 0,8473 випливає з визначення часу наростання між рівнями $0{,}3 \cdot V_{DD}$ та $0{,}7 \cdot V_{DD}$ для експоненційного перехідного процесу. Якщо обчислене значення R_p перевищує максимально допустимий опір, що гарантує стабільний низький рівень при заданому

струмі витоку (згідно з V_{ol} та I_{ol}), слід зменшувати ємність шини (скороченням доріжок, вибором мікросхем із меншою вхідною ємністю) або переходити до активних підтяжок.

Для ілюстрації впливу швидкісного режиму на форму сигналів на рисунку 3.11 наведено типові осцилограми сигналів SCL і SDA для Standard-mode (100 кГц) та Fast-mode (400 кГц) на одній і тій самій шині з підтягувальним резистором 4,7 кОм та ємністю близько 120 пФ. Видно, що в режимі 400 кГц фронти сигналів стають пологішими, проте все ще вкладаються в допуск, тоді як спроба роботи на 1 МГц із тим самим резистором призвела б до порушення часових параметрів. Це наочно демонструє, чому розробник не може просто збільшити частоту в регістрі конфігурації, не перевіривши фізику шини.

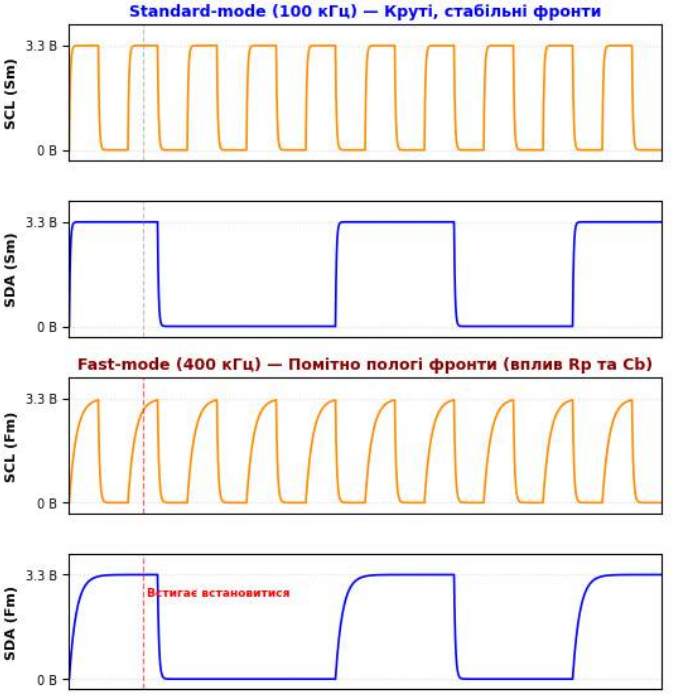


Рисунок 3.11 – Осцилограми сигналів I²C для Standard-mode (100 кГц) та Fast-mode (400 кГц) на шині з Rp=4,7 кОм, Cb≈120 пФ

Примітка: на частоті 400 кГц із Rp=4,7 кОм шина ще працює стабільно; для Fm+ необхідно зменшити Rp до 1–2 кОм або застосувати активну підтяжку.

Слід також зазначити, що виробники мікроконтролерів часто вказують підтримувані швидкісні режими в документації на периферійний модуль I²C. Так, у STM32F4 (RM0090, розділ 27) апаратний модуль підтримує Standard та Fast режими без додаткових налаштувань, тоді як для Fast+ може знадобитися налаштування спеціальних бітів у регістрах керування та використання виводів із підвищеною здатністю навантаження (SYSCFG). У ESP32 (ESP32-S3 Technical Reference Manual, розділ I²C) підтримуються Standard, Fast та Fast+, причому максимальна частота SCL обмежена 1,25 МГц, однак реальна досяжна швидкість залежить від якості плати та навантаження. Мікроконтролери RP2040 (Raspberry Pi Pico) також підтримують Standard, Fast і Fast+ із максимальною частотою 1 Мбіт/с безпосередньо через апаратний модуль I²C.

Ієрархія швидкісних режимів I²C надає розробникові гнучкий інструментарій: від максимально прощального Standard-mode для макетів до швидкісного Fast-mode Plus для обміну з високопродуктивними сенсорами. Наступні підрозділи, присвячені SMBus та I3C, продемонструють, як принципи I²C були розвинені в більш спеціалізовані протоколи, що зберігають сумісність із базовою архітектурою, але додають нові функції, такі як гаряче підключення та динамічна адресація.

3.3.2. SMBus та PMBus як похідні для систем живлення

Універсальність I²C, доведена десятиліттями успішного використання, закономірно спонукала інженерів до створення на його основі спеціалізованих протоколів, які, не змінюючи базову фізичну архітектуру, додають функції, критично необхідні для конкретних галузей. Двома найяскравішими прикладами такого підходу є SMBus (System Management Bus) та його нащадок PMBus (Power Management Bus), які перетворили I²C із універсальної периферійної шини на стандартизований кістяк для моніторингу, керування та діагностики систем електроживлення. Поява цих протоколів була зумовлена гострою потребою в єдиному інтерфейсі для взаємодії між батареями, зарядними пристроями, регуляторами напруги, контролерами вентиляторів і центральним процесором — компонентами, від злагодженої роботи яких безпосередньо залежить надійність серверів, ноутбуків, телекомунікаційного обладнання та вбудованих систем.

Системна шина керування SMBus була вперше специфікована компанією Intel у 1995 році як засіб для низькошвидкісного обміну між інтелектуальними акумуляторними батареями (Smart Battery) та материнською

платою. Згодом розвиток специфікації було передано незалежному консорціуму SBS Implementers Forum (SBS-IF), який опублікував версію SMBus 3.2 у 2022 році. Фізичний рівень SMBus майже ідентичний I²C: дві лінії SMBCLK (тактовий сигнал) і SMBDAT (дані) з відкритим стоком і підтягувальними резисторами. Відмінність полягає в жорсткіших електричних специфікаціях: номінальна напруга живлення може становити 1,8 В, 2,5 В або 3,3 В (на відміну від переважно 5-вольтового I²C), що відповідає сучасній низьковольтній логіці. Проте ключова інновація SMBus лежить не у фізиці, а в чітко визначеному наборі протокольних правил, які роблять поведінку пристроїв передбачуваною в будь-яких, навіть позаштатних, ситуаціях.

Найважливішим доповненням SMBus до базового I²C є система таймаутів. Специфікація SMBus встановлює мінімальну частоту SMBCLK 10 кГц (на відміну від нульової в I²C, де допускається повна зупинка такту) та жорсткі часові обмеження на різні фази транзакції. Якщо ведений пристрій утримує лінію SMBCLK у низькому стані довше дозволеного часу T_TIMEOUT (зазвичай 25 або 35 мс), майстер має право примусово завершити транзакцію, скинувши шину. Це запобігає повному зависанню системи через несправний периферійний модуль — ситуації, яка для стандартного I²C могла б призвести до фатального блокування. Крім того, SMBus запроваджує спеціальний протокол періодичного опитування аварій (ARA — Alert Response Address), за яким пристрій, що потребує негайної уваги (наприклад, батарея, яка перегрілася), сигналізує про це через вивід SMBALERT#, а майстер у відповідь надсилає зарезервованою адресою 0x0C запит, на який пристрій відповідає своєю адресою. Таким чином, SMBus перетворює пасивну шину на систему з підтримкою асинхронних сповіщень, що є критичним для керування живленням, де реакція на перегрів чи коротке замикання має вимірюватися мілісекундами.

SMBus також суворо стандартизував формат команд і даних. Якщо в I²C інтерпретація байтів, що слідують за адресою, повністю залишається на розсуд виробника мікросхеми, то в SMBus визначено близько десятка обов'язкових протоколів — Send Byte, Receive Byte, Write Byte/Word, Read Byte/Word, Block Read/Write тощо. Кожен із цих протоколів передбачає, що перший байт після адреси є кодом команди, а подальші байти — даними визначеної довжини. Це дозволило створити єдиний програмний інтерфейс для різнорідних пристроїв, уніфікувавши драйверний рівень.

Подальшим розвитком ідей SMBus став протокол PMBus (Power Management Bus), специфікація якого підтримується консорціумом PMBus Implementers Forum (нині частина System Management Interface Forum,

SMIF). Перша версія PMBus з'явилася у 2005 році, а актуальна версія 1.3.1 датується 2015 роком. PMBus повністю сумісний із SMBus на фізичному та канальному рівнях, але додає до нього розгалужену систему команд, спеціалізованих саме для керування перетворювачами енергії — DC/DC-конверторами, AC/DC-блоками, контролерами батарей, моніторами струму. Специфікація PMBus визначає понад 200 стандартних команд, згрупованих за функціональним призначенням: встановлення вихідної напруги (VOUT_COMMAND), обмеження струму (IOUT_OC_FAULT_LIMIT), контроль температури (OT_FAULT_LIMIT), моніторинг статусу (STATUS_WORD), вимірювання реальних значень (READ_VOUT, READ_IOUT, READ_TEMPERATURE_1) тощо. Формат даних у PMBus, відомий як LINEAR11, використовує 16-бітні слова, де 5 старших бітів відводяться під експоненту (степінь двійки), а 11 молодших — під мантису. Це забезпечує компактне представлення величин, що змінюються в широкому діапазоні (від мікровольт до сотень вольт), із достатньою для керування точністю.

Використання PMBus дозволяє реалізувати концепцію цифрового керування живленням, за якої центральний мікроконтролер (або спеціалізований PMBus-контролер) динамічно регулює напруги, відстежує енергоспоживання, прогнозує відмови та оптимізує ККД усієї системи. Типову архітектуру PMBus-системи зображено на рисунку 3.12: мікроконтролер виконує роль майстра, до шини під'єднано кілька ведених пристроїв — батарейний контролер, модуль гарячої заміни (hot swap), багатоканальний DC/DC-конвертор із PMBus-інтерфейсом, а також монітор струму. Кожен із цих пристроїв має вивід ALERT#, під'єднаний до лінії SMBALERT#, що забезпечує негайне сповіщення майстра про аварійні події.

Порівняння ключових характеристик I²C, SMBus та PMBus наведено в таблиці 3.9, яка унаочнює еволюцію від загальної периферійної шини до спеціалізованого протоколу керування живленням. Дані для таблиці взято з NXP UM10204 (I²C), SMBus Specification 3.2 (SBS-IF, 2022) та PMBus Specification 1.3.1 (SMIF, 2015).

Важливою практичною перевагою SMBus і PMBus для розробника є те, що вони позбавляють необхідності вивчати унікальний набір регістрів кожного нового мікроконтролера живлення. Завдяки стандартизованим командам PMBus код для зчитування вихідної напруги чи встановлення порогу струмового захисту виглядає ідентично незалежно від того, чи це мікросхема від Texas Instruments (наприклад, TPS544C25), чи від Infineon (IR38060), чи від Maxim Integrated (MAX15301).

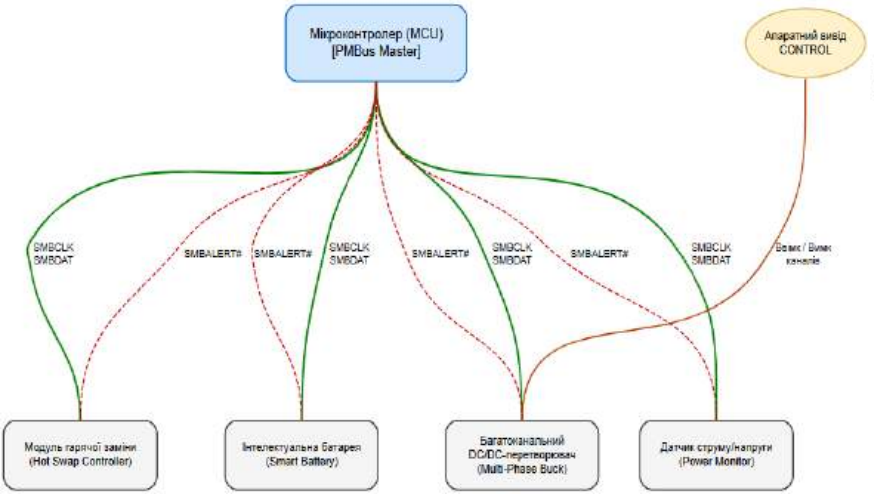


Рисунок 3.12 – Архітектура системи цифрового керування живленням на базі PMBus

Таблиця 3.9 – Порівняння характеристик I²C, SMBus та PMBus

Режим	Характеристика	Значення
I²C (Standard/Fast)	*Фізичний рівень*	2 дроти, відкритий стік
	Мінімальна тактова частота	0 Гц (немає)
	Таймаути	відсутні
	Сповіщення про події	немає
	Стандартизовані команди	немає
	Адреса для аварійних сповіщень	немає
	Формат даних	визначається виробником
	Типове застосування	датчики, EEPROM, RTC
SMBus 3.2	*Фізичний рівень*	2 дроти (SMBCLK, SMBDAT), 1.8/2.5/3.3 В
	Мінімальна тактова частота	10 кГц
	Таймаути	T_TIMEOUT (25–35 мс)
	Сповіщення про події	SMBALERT# + ARA (адреса 0x0C)
	Стандартизовані команди	~10 обов'язкових протоколів (Write Byte тощо)
	Адреса для аварійних сповіщень	0x0C (ARA)
	Формат даних	Фіксований (команда + дані)
	Типове застосування	Smart Battery, моніторинг температури

Кінець таблиці 3.9

Режим	Характеристика	Значення
PMBus 1.3.1	*Фізичний рівень*	2 дроти, сумісний з SMBus 1.1/2.0
	Мінімальна тактова частота	10 кГц (як SMBus)
	Таймаути	успадковано від SMBus
	Сповіщення про події	SMBALERT# + CONTROL#
	Стандартизовані команди	>200 стандартних команд (VOUT_COMMAND тощо)
	Адреса для аварійних сповіщень	успадковано від SMBus
	Формат даних	LINEAR11 (16 біт)
	Типове застосування	цифрове керування живленням, DC/DC, AC/DC

Така уніфікація прискорює розроблення, зменшує кількість помилок і дозволяє створювати бібліотеки повторно використовуваного коду. У лістингу 3.7 наведено фрагмент коду для STM32, який виконує стандартну PMBus-транзакцію Read Word для отримання виміряної вихідної напруги (команда 0x8B, READ_VOUT) від DC/DC-конвертора за адресою 0x40. Код використовує HAL-функції I²C, оскільки фізично PMBus є тим самим I²C.

Лістинг 3.7 – Читання вихідної напруги через PMBus командою READ_VOUT (0x8B) на STM32 (HAL)

```
/* --- ВИРІЗКА З ГОЛОВНОГО ФАЙЛУ ПРОГРАМИ (main.c) --- */
#include "main.h"
#include <math.h> // Необхідно для функції powf()
// Глобальні змінні
extern I2C HandleTypeDef hi2c1;
// Налаштування PMBus згідно з документацією
#define PMBUS_ADDR          (0x40 << 1) // 7-бітний адрес,
зсунутий для HAL (8-біт)
#define PMBUS READ VOUT     0x8B // Команда читання вихідної
напруги
// Прототипи функцій
HAL_StatusTypeDef pmbus_read_word(uint8_t cmd, uint16_t
*data);

int main(void)    {
  /* ... [Базова ініціалізація системи HAL_Init(),
SystemClock_Config() та MX_I2C1_Init()] ... */
  uint16_t raw_data = 0;
  float vout = 0.0f;
  while (1)      {
    /* [Крок 1] Безпечне читання даних через PMBus */
    // НАГОЛОШЕННЯ ДЛЯ СТУДЕНТІВ: Перед кожним читанням у
```

```
реальних проектах слід перевіряти
    // готовність пристрою за допомогою
HAL I2C IsDeviceReady(), щоб уникнути зависання шини.
    if (HAL_I2C_IsDeviceReady(&hi2c1, PMBUS_ADDR, 3, 100) ==
HAL_OK)
    {
      if (pmbus_read_word(PMBUS_READ_VOUT, &raw_data) ==
HAL_OK)      {
        /* [Крок 2] Декодування формату PMBus LINEAR11 (2
байти)
           Бортові обчислення: дані приходять як signed
мантиса (11 біт) та signed експонента (5 біт).
           Структура raw data: [ EEEE EMMM MMMM MMMM ] */
        // Витягуємо 5-бітну експоненту (старші біти) та
приводимо до знакового 8-бітного типу
        int8_t exponent = (int8_t)(raw_data >> 11);
        if (exponent > 15) { exponent -= 32; } // Обробка
знаку для 5-бітного числа (двійковий доповнюючий код)
        // Витягуємо 11-бітну мантису (молодші біти) та при
водимо до знакового 16-бітного типу
        int16_t mantissa = (int16_t)(raw_data & 0x07FF);
        if (mantissa > 1023) { mantissa -= 2048; } // Обробка
знаку для 11-бітного числа
        /* [Крок 3] Обчислення фізичної величини у Вольтах
    Формула стандарту PMBus: V = Mantissa * 2^(Exponent) */
    vout = (float)mantissa * powf(2.0f, (float)exponent);
        // Результат готовий для виведення або телеметрії
      printf("Вихідна напруга VOUT: %.3f V\r\n", vout);  } }
    HAL_Delay(500); // Опитування кожні 500 мс  }    }
/**   * @brief Функція читання 16-бітного слова (двох байт)
з регістру PMBus пристрою   */
HAL_StatusTypeDef pmbus_read_word(uint8_t cmd, uint16_t
*data)   {   uint8_t buf[2] = {0};
  // PMBus базується на протоколі SMBus, де транзакція
читання слова (Read Word)
  // спочатку надсилає код команди (cmd), а потім зчитує 2
байти даних
  HAL_StatusTypeDef status = HAL_I2C_Mem_Read(&hi2c1,
PMBUS_ADDR, cmd,
                          I2C_MEMADD_SIZE_8BIT, buf, 2, 100);
  if (status == HAL_OK)
  {    // Об'єднуємо два зчитані байти у єдине 16-бітне слово
(Little-Endian формат)
    *data = (uint16_t)(buf[0] | (buf[1] << 8));  }
  return status;   }
```

SMBus і PMBus є прикладами того, як галузеві консорціуми, спираючись на перевірену часом фізичну основу I²C, створили повноцінні спеціалізовані протоколи, що вирішують конкретні інженерні проблеми — передбачуване відновлення після збоїв, асинхронне сповіщення про події та уніфіковане керування енергоперетворювальною апаратурою. Знання цих протоколів є обов'язковим для будь-якого розробника вбудованих систем,

який працює з батарейним живленням, серверними платформами чи промисловою автоматикою. Наступний підрозділ, присвячений еволюції I²C до I3C, покаже, як навіть найвдаліші стандарти переосмислюються з появою нових вимог, таких як ультранизьке енергоспоживання та динамічне гаряче підключення.

3.3.3. Модернізовані специфікації I3C (MIPI I3C) з гарячим підключенням та динамічною адресацією

Незважаючи на десятиліття успішного застосування I²C у вбудованих системах, на початку 2010-х років стала очевидною його фундаментальна обмеженість для нового покоління мобільних і сенсорних пристроїв. Фіксована 7-бітна адресація спричиняла конфлікти між однаковими мікросхемами на одній шині, відсутність підтримки гарячого підключення ускладнювала модульні конструкції, а виділений вивід переривання (як у SMBALERT#) збільшував кількість доріжок на платі. У відповідь на ці виклики альянс MIPI (Mobile Industry Processor Interface) у 2017 році опублікував специфікацію MIPI I3C v1.0, яка, зберігаючи двопровідну сумісність із I²C, переосмислює протокол на всіх рівнях. I3C (Improved Inter-Integrated Circuit) поєднує в собі енергоефективність I²C зі швидкістю SPI, додаючи механізми, необхідні для сучасних сенсорних хабів у смартфонах, носимих пристроях, автомобільних системах і розподілених IoT-мережах. Подальша версія I3C Basic v1.1.1 (MIPI, 2021) спростила вимоги для широкого впровадження, зробивши специфікацію доступнішою для виробників мікроконтролерів. Сьогодні I3C підтримується на апаратному рівні в сімействах STM32L5, STM32U5, nRF5340, а також у складі платформ Snapdragon та інших SoC, що робить його обов'язковим для вивчення.

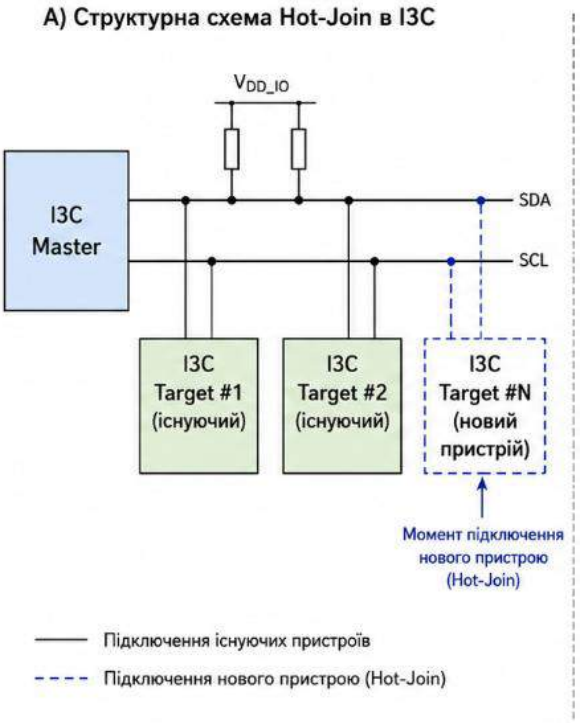
Фізичний рівень I3C використовує ті самі дві лінії — SCL (Serial Clock) і SDA (Serial Data), — проте на відміну від I²C, де обидві є відкритим стоком, у I3C лінія SCL керується майстром у двотактному (push-pull) режимі, що дозволяє досягати крутіших фронтів і вищих частот. Лінія SDA залишається відкритим стоком для забезпечення сумісності з I²C та можливості арбітражу, однак під час високошвидкісних передач вона також може переходити в push-pull. Базовий швидкісний режим SDR (Single Data Rate) забезпечує тактову частоту до 12,5 МГц, що з урахуванням формату даних дає корисну пропускну здатність близько 10 Мбіт/с — у 25 разів більше, ніж Fast-mode I²C. Розширені режими HDR (High Data Rate), зокрема HDR-DDR (Double Data Rate) і HDR-TSP/TSP, піднімають швидкість до 33 Мбіт/с, використовуючи складніші схеми модуляції. Таким чином, I3C

закриває розрив між «повільною» периферією I²C та «швидкою» SPI, лишаючись при цьому на двопровідній шині.

Проте головні інновації I3C зосереджені не на швидкості, а на протокольному рівні, і саме вони становлять предмет цього підрозділу. Найважливішою з них є динамічна адресація (Dynamic Address Assignment, DAA). У класичному I²C кожен ведений пристрій має фіксовану апаратну адресу, що призводить до конфліктів при спробі встановити два однакові датчики на одну шину. I3C вирішує цю проблему радикально: після ввімкнення або скидання шини майстер виконує процедуру призначення адрес, під час якої кожен ведений пристрій отримує унікальну 7-бітну динамічну адресу. Процес спирається на унікальний 48-бітний ідентифікатор Provisional ID (PID), який зберігається у пристрої і надається виробником. Майстер опитує шину, використовуючи команду ENTDAA (Enter Dynamic Address Assignment), і пристрої по черзі повідомляють свої PID, на основі яких майстер призначає адреси, уникнувши дублювання. Після цього пристрій відгукується лише на динамічну адресу, а його фіксована I²C-адреса деактивується (якщо пристрій не є суто I²C-сумісним). Цей механізм робить шину самоконфігурованою і усуває потребу в ручному встановленні адресних перемичок чи конфігураційних регістрів.

Прямим наслідком динамічної адресації є підтримка гарячого підключення (Hot-Join). Пристрій, під'єднаний до вже працюючої шини I3C, може сповістити про свою появу, потягнувши лінію SDA в низький рівень під час стартової умови (так званий Hot-Join Request). Майстер, виявивши це, ініціює процедуру DAA для нового пристрою, не перериваючи обмін із рештою вузлів. Це докорінно відрізняє I3C від I²C, де будь-яка зміна конфігурації шини вимагала повного скидання. Схематичне зображення процедури Hot-Join із часовими діаграмами подано на рисунку 3.13.

Ще однією визначною рисою I3C є вбудована підтримка переривань (In-Band Interrupt, IBI). У I²C для аварійних сповіщень потрібен окремий провід (SMBALERT#), що збільшує кількість виводів. I3C використовує той самий канал SDA для передачі переривання: ведений пристрій, якому потрібна увага майстра, подає запит під час вільного періоду шини, після чого майстер опитує пристрої, щоб визначити джерело.



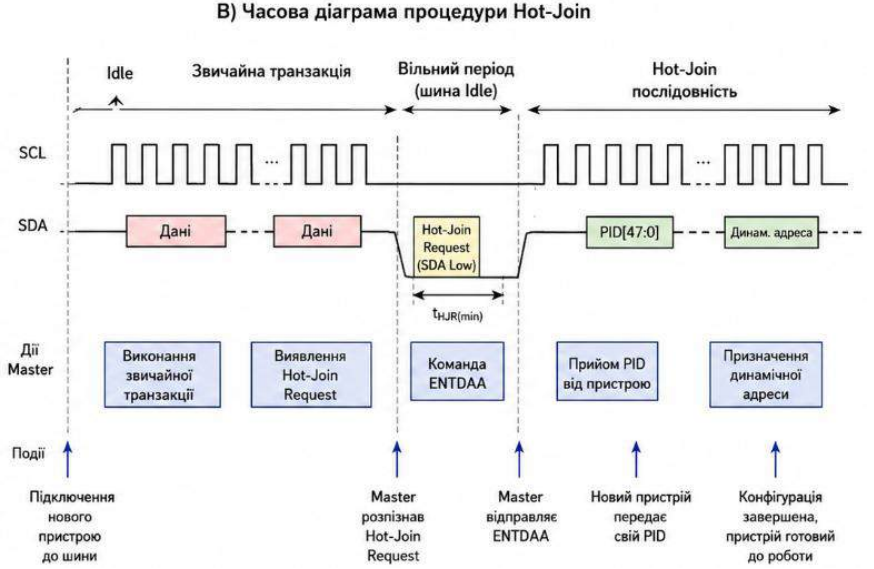


Рисунок 3.13 – Процедура Hot-Join в I3C

Це зменшує кількість доріжок і спрощує розведення плати. Крім того, I3C запроваджує апаратний механізм виявлення помилок за допомогою CRC (Cyclic Redundancy Check) на канальному рівні, що відсутній у стандартному I²C. Разом із вбудованими таймаутами та відновленням після збоїв це робить шину значно стійкішою до завад, що є критичним для автомобільних і промислових застосувань.

Важливо підкреслити, що I3C зберігає зворотну сумісність із I²C на рівні електричних сигналів. Майстер I3C може обслуговувати класичні I²C-пристрої на тій самій шині, але з обмеженнями: для I²C-слейвів не працюють I3C-специфічні функції, вони повинні підтримувати швидкість не нижче Fast-mode, і не повинні використовувати розтягування такту, оскільки майстер I3C не зобов'язаний його обробляти. Таким чином, розробник може поступово мігрувати з I²C на I3C, замінюючи лише ті компоненти, які потребують нових можливостей.

Для кількісного порівняння основних характеристик I²C та I3C складено таблицю 3.10, що спирається на специфікації NXP UM10204 (I²C) та MIPI I3C v1.0 (MIPI, 2017) і I3C Basic v1.1.1 (MIPI, 2021).

Таблиця 3.10 – Порівняння I²C та I3C

Інтерфейс	Характеристика	Значення
I²C (Fast / Fast+)	*Максимальна тактова частота*	1 МГц (Fm+)
	Корисна пропускна здатність	~0,8 Мбіт/с
	Тип виходу SCL	відкритий стік
	Адресація	фіксована (7/10 біт), можливі конфлікти
	Вирішення конфліктів адрес	потребує ручного налаштування
	Гаряче підключення	потребує ручного налаштування
	Переривання від веденого	відсутнє
	Виявлення помилок	парність (опціонально)
	Сумісність із I²C	так (базовий)
	Типове енергоспоживання на біт	~0,5 нДж/біт

Кінець таблиці 3.10

Інтерфейс	Характеристика	Значення
I3C (SDR)	*Максимальна тактова частота*	12,5 МГц
	Корисна пропускна здатність	~10 Мбіт/с
	Тип виходу SCL	push-pull (SCL)
	Адресація	динамічна (DAA), унікальний PID
	Вирішення конфліктів адрес	автоматичне вирішення
	Гаряче підключення	підтримується (Hot-Join)
	Переривання від веденого	вбудоване (IBI)
	Виявлення помилок	CRC, таймаути
	Сумісність із I²C	так (обмежена)
	Типове енергоспоживання на біт	~0,2 нДж/біт

Мікроконтролери STM32, починаючи з лінійок L5 та U5, містять апаратний модуль I3C, налаштування якого в середовищі CubeMX подібне до I²C, але потребує додаткових параметрів. У лістингу 3.8 наведено спрощений приклад ініціалізації I3C-майстра на STM32L5 у HAL (HAL_I3C_Init), хоча слід зауважити, що повна реалізація динамічної адресації та Hot-Join потребує виклику спеціальних функцій стеку I3C, наданих STMicroelectronics у складі пакету X-CUBE-I3C.

Лістинг 3.8 – Ініціалізація I3C-майстра та базове передавання на STM32L5 (HAL)

```
/* --- ВИРІЗКА З ГОЛОВНОГО ФАЙЛУ ПРОГРАМИ (main.c) --- */
#include "main.h"
// Глобальний дескриптор периферії I3C
I3C_HandleTypeDef hi3c1;
/* ... [Тут знаходиться автогенерований код ініціалізації
системи HAL_Init()] ... */
int main(void)
{
  SystemClock_Config(); // Налаштування тактування ядра та
периферії
  /* [Етап 1] Конфігурація I3C-майстра (12.5 МГц, Режим Push-
Pull) */
  hi3c1.Instance = I3C1;
  hi3c1.Init.Mode = HAL_I3C_MODE_MASTER;   // Мікроконтролер
виступає як Майстер шини
  hi3c1.Init.CtrlBusFreq.SclBusFreq = 12500000;   // Частота
тактування SCL = 12.5 МГц (максимум для I3C SDR)
  hi3c1.Init.CtrlBusFreq.SclHighTime = 40;  // Таймінги
високого рівня SCL у наносекундах
  hi3c1.Init.TxFifoThreshold = HAL_I3C_TXFIFO_THRESHOLD_1;
```

```
  hi3c1.Init.RxFifoThreshold = HAL_I3C_RXFIFO_THRESHOLD_1;
  // Виклик базової апаратної ініціалізації контролера I3C у
STM32L5
  if (HAL_I3C_Init(&hi3c1) != HAL_OK)
  {    Error_Handler();  }
  /*[Етап 2]Підготовка даних та базова транзакція на шині */
  uint8_t tx_buffer[] = {0x40, 0x1A, 0x2B}; // Приклад пакета
даних (команда + параметри)
  uint8_t target_dynamic_addr = 0x0A;       // Тимчасова або
фіксована адреса веденого
  /* Виклик функції для прямого передавання даних веденому
пристрою. Останній параметр (100) — таймаут операції в
мілісекундах. */
  if (HAL_I3C_Master_Transmit(&hi3c1, target_dynamic_addr,
tx_buffer, sizeof(tx_buffer), 100) != HAL_OK)
  {
    Error_Handler();
  }
  /* ======================================================= */
  /*  ШИНА ПЕРЕЙШЛА В СТАН ОЧІКУВАННЯ (IDLE) ПІСЛЯ УСПІШНОЇ
ТРАНЗАКЦІЇ         */
  /* ======================================================= */
  while (1)
  {
 /* НАГОЛОШЕННЯ ДЛЯ СТУДЕНТІВ: Цей лістинг демонструє лише
базовий обмін за готовою адресою. Для повноцінного
використання технології I3C (Hot-Join, Dynamic Address
Allocation) у реальних лабораторних роботах необхідно:
1. Спочатку зареєструвати пристрій через HAL_I3C_AddSlave()
за його статічною адресою. 2. Викликати процедуру апаратної
генерації адрес HAL I3C Master ProcessDAA() (команда
ENTDAA), яка зчитає 48-бітний PID веденого та призначить йому
динамічний адрес. */
  }   }
```

Підсумовуючи, I3C уособлює еволюційний стрибок від класичної послідовної шини до інтелектуальної сенсорної мережі, що самоналаштовується. Завдяки динамічній адресації, гарячому підключенню та вбудованим перериванням він усуває найболючіші проблеми I²C, зберігаючи при цьому зворотну сумісність. Для студента-розробника це означає, що знання I²C є надійним фундаментом, але в нових проєктах слід оцінити доцільність переходу на I3C, особливо коли йдеться про мультисенсорні платформи з частим гарячим підключенням модулів. Третій розділ на цьому завершується, і наступна частина посібника перейде до вивчення промислових та автомобільних шин, які, подібно до I²C, використовують принцип спільної шини, але оперують значно більшими відстанями та жорсткішими вимогами до надійності.

РОЗДІЛ 4.
ПРОМИСЛОВІ ТА АВТОМОБІЛЬНІ ПОСЛІДОВНІ ШИНИ

4.1. RS-485 та Modbus RTU: диференційна передача, термінування, топології багатоточкових мереж

У промисловій автоматизації та системах збору даних, де пристрої рознесені на десятки й сотні метрів, а рівень електромагнітних завад часто сягає десятків вольт, класичні несиметричні інтерфейси (TTL, RS-232) стають непридатними. Натомість диференційна передача сигналу забезпечує високу завадостійкість і дозволяє будувати багатоточкові шини з десятками вузлів. Саме на цих засадах ґрунтується фізичний рівень RS-485, стандартизований у TIA/EIA-485-A, який став основою для наймасовішого промислового протоколу прикладного рівня — Modbus. Поєднання простоти монтажу (вита пара, термінування на кінцях) і зрілої специфікації Modbus, що охоплює як бінарний RTU-режим, так і текстові ASCII-кадри, дозволяє інтегрувати різнорідне обладнання від десятків виробників без потреби в дорогих шлюзах. Водночас експлуатація таких мереж у цехах, на відкритих майданчиках або в транспортній інфраструктурі висуває жорсткі вимоги до захисту від імпульсних перенапруг, електростатичних розрядів і кондуктивних завад, які регламентуються серією стандартів IEC 61000. Цей розділ послідовно розкриває логіку Modbus-кадрів, методи обчислення контрольної суми та обробки винятків, а також надає практичні схеми захисту портів RS-485, що дозволяють забезпечити багаторічну безвідмовну роботу в умовах реального виробництва.

4.1.1. Протокол Modbus: кадри RTU/ASCII, виняткові ситуації

Modbus є одним із найстаріших і водночас найпоширеніших протоколів прикладного рівня для промислових мереж. Розроблений компанією Modicon (нині частина Schneider Electric) у 1979 році для зв’язку програмованих логічних контролерів, він пережив кілька десятиліть і залишається актуальним завдяки своїй відкритості, простоті реалізації та незалежності від фізичного середовища. Специфікації Modbus підтримуються некомерційною організацією Modbus Organization і є вільно доступними на її вебсайті, що зробило протокол стандартом де-факто для інтеграції різнорідного обладнання — від датчиків температури до частотних перетворюва-

чів. У цьому підрозділі розглядаються два основні режими передавання даних через послідовні лінії (RTU та ASCII), структура кадрів, методи контролю цілісності та механізми обробки виняткових ситуацій, як це визначено в документах Modbus Application Protocol Specification V1.1b3 (Modbus Organization, 2012) та Modbus over Serial Line Specification and Implementation Guide V1.02 (Modbus Organization, 2006). Фізичний рівень, заснований на RS-485, буде детально описаний у наступних підрозділах, однак його ключові характеристики — диференційна передача, необхідність термінування та багатоточкова топологія — є невід'ємним контекстом для розуміння того, чому саме RTU-режим став домінуючим у промисловості.

Протокол Modbus на послідовних лініях передбачає два режими передавання: RTU (Remote Terminal Unit) та ASCII (American Standard Code for Information Interchange). Режим RTU є обов'язковим для всіх пристроїв, що претендують на відповідність специфікації, тоді як ASCII є опціональним і використовується переважно в застарілих системах або там, де потрібна можливість налагодження через звичайний термінал. Основна відмінність між режимами полягає у форматі кадру та способі обчислення контрольної суми, тоді як логічна структура повідомлень (адреса, код функції, дані) залишається ідентичною.

Кадр Modbus RTU, структуру якого наведено на рисунку 4.1, починається з байта адреси веденого пристрою (діапазон 1–247, адреса 0 зарезервована для широкомовних повідомлень), за яким слідує байт коду функції. Поле даних має змінну довжину, яка залежить від конкретної функції; воно може містити адреси регістрів, кількість регістрів, лічильники байтів і власне дані для запису. Завершується кадр двома байтами циклічного надлишкового коду CRC-16, причому молодший байт передається першим, що є характерною особливістю Modbus. Критичним часовим параметром RTU-режиму є міжкадрова пауза (inter-frame gap), яка має бути не меншою за 3,5 символьних інтервали на заданій швидкості передавання. Ведений пристрій використовує цю паузу для детектування кінця кадру; якщо після останнього байта пауза менша за 1,5 символьних інтервали, кадр вважається неповним і відкидається. Такий підхід уможливлює синхронізацію без використання спеціальних старт-стопових символів, однак вимагає від приймача точного вимірювання часу між байтами.

Кадр Modbus ASCII, на відміну від бінарного RTU, використовує виключно друковані символи шістнадцяткової системи числення (0–9, A–F).



Рисунок 4.1 – Структура кадрів Modbus RTU та Modbus ASCII

Кожен байт оригінального повідомлення перетворюється на два ASCII-символи, що робить трафік читабельним для людини, але збільшує обсяг переданих даних удвічі. Кадр починається з символу двокрапки (':', код 0x3A) і завершується парою символів повернення каретки та переведення рядка (CRLF, 0x0D 0x0A). Контрольна сума в ASCII-режимі обчислюється за алгоритмом LRC (Longitudinal Redundancy Check) — простою сумою всіх байтів повідомлення, взятою з оберненням (доповненням до двох). Між символами всередині кадру допускається пауза до 1 секунди, що робить ASCII більш стійким до нестабільних ліній зв'язку, однак менш ефективним за швидкістю. Порівняльна структура обох форматів кадру показана на рисунку 4.1, де чітко видно різницю в кодуванні та методах обрамлення.

Функціональність Modbus визначається набором стандартних кодів функцій, які зведено в таблиці 4.1. Кожна функція ініціює певну операцію над даними веденого пристрою, що зберігаються в чотирьох адресних просторах: дискретні виходи (Coils, адреси 0xxxx), дискретні входи (Discrete Inputs, 1xxxx), регістри вводу (Input Registers, 3xxxx) та регістри зберігання (Holding Registers, 4xxxx). Найуживанішими функціями є 03 (Read Holding Registers) для читання значень аналогових датчиків і 06 (Write Single Register) для керування виконавчими механізмами. Кожен запит майстра містить код функції, початкову адресу та кількість елементів, а ведений формує відповідь, що або повторює код функції з даними, або сигналізує про помилку.

Таблиця 4.1 – Стандартні коди функцій Modbus (найпоширеніші)

Код (hex)	*Назва функції*	*Призначення*	*Тип даних*
0x01	Read Coils	Читання дискретних виходів (стану котушок/реле)	Coils (0xxxx)
0x02	Read Discrete Inputs	Читання дискретних входів (стану кнопок, перемикачів тощо)	Discrete Inputs (1xxxx)
0x03	Read Holding Registers	Читання 16-бітних регістрів зберігання (параметрів, уставок)	Holding Registers (4xxxx)
0x04	Read Input Registers	Читання 16-бітних вхідних регістрів (аналогових сигналів)	Input Registers (3xxxx)
0x05	Write Single Coil	Встановлення (запис) одного дискретного виходу	Coils
0x06	Write Single Register	Запис одного регістра зберігання	Holding Registers
0x0F	Write Multiple Coils	Запис кількох дискретних виходів	Coils
0x10	Write Multiple Registers	Запис кількох регістрів зберігання	Holding Registers

Контроль цілісності в RTU-режимі забезпечується алгоритмом CRC-16, що використовує породжувальний поліном 0xA001 (обернений стандартний CRC-16-IBM). Алгоритм обчислює 16-бітний залишок від ділення бітової послідовності всього кадру (без самого CRC) на поліном, причому обчислення виконується над байтами в порядку їх слідування. Приклад програмної реалізації цього алгоритму на мові C для мікроконтролерів STM32 наведено в лістингу 4.1, де демонструється ітеративний підхід із попередньо обчисленою таблицею або без неї (для економії пам’яті). Важливо, що приймач, отримавши кадр, обчислює CRC самостійно і порівнює з прийнятим значенням; розбіжність призводить до ігнорування кадру без формування відповіді про помилку, оскільки пошкоджений кадр міг містити некоректну адресу.

Лістинг 4.1 – Обчислення CRC-16 для Modbus RTU на мові C (табличний метод)

```
1:// Сумісність: MODBUS over Serial Line Specifica-tion V1.02,
2:Appendix B
3: #include <stdint.h>
4:// Таблиця CRC-16 для полінома 0xA001 (реверсивна форма
5: 0x8005)
6: // Попередньо обчислені значення для всіх 256 можливих
7: байт
8: static const uint16_t crc16_table[256] = {
9: 0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280,
```

```
0xC241,
0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481,
0x0440,
// (повна таблиця з 256 елементів опущена для стислості)
0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81,
0x0E40,
0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880,
0xC841};
// ------------------------------------------------------
// @brief Обчислення CRC-16 для Modbus RTU
// @param buf Покажчик на масив байт даних
// @param len Довжина даних у байтах
// @return 16-бітове CRC-значення
// @note - Початкове значення CRC-реєстру: 0xFFFF
- Фінальний XOR: 0x0000
// - Поліном: 0xA001 (реверсивний від 0x8005)
- Порядок передачі: молодший байт вперед (little-
endian) [7†L4-L5]
// ------------------------------------------------------
uint16_t modbus_crc16(uint8_t *buf, uint16_t len) {
uint16_t crc = 0xFFFF; // 1. Ініціалізація
CRC-реєстру [1†L5]
for (uint16_t i = 0; i < len; i++) {
// 2. XOR байта даних з молодшим байтом CRC-реєстру
[9†L4-L5]
/ / 3. Індекс у таблиці: (crc ^ buf [i]) & 0xFF
// 4. Оновлення CRC: зсув праворуч на 8 біт + XOR з
табличним значенням
crc = (crc >> 8) ^ crc16_table[(crc ^ buf[i]) & 0xFF];
}
return crc; // 5. Фінальний XOR 0x0000 (без змін)
}
```

Виняткові ситуації (exceptions) є невід'ємною частиною Modbus і забезпечують інформування майстра про неможливість виконати коректний запит. Коли ведений пристрій виявляє помилку, він формує спеціальний кадр відповіді, у якому старший біт коду функції встановлюється в одиницю (тобто до коду функції додається 0x80), а поле даних містить один байт — код винятку. Майстер, отримавши таку відповідь, може ідентифікувати тип помилки та вжити відповідних заходів. Специфікація Modbus визначає вісім стандартних кодів винятків, перелічених у таблиці 4.2. На практиці найчастіше зустрічаються коди 0x01 (непідтримувана функція), 0x02 (некоректна адреса даних) та 0x03 (неприпустиме значення даних, наприклад, спроба записати значення поза діапазоном). Важливо, що на виняткові ситуації не поширюється правило мовчання при помилці CRC: якщо кадр має коректну структуру та CRC, але запитувана функція не реалізована, ведений пристрій зобов'язаний відповісти винятком, а не мовчати.

Таблиця 4.2 – Коди виняткових ситуацій Modbus

Код (hex)	Назва	Опис
0x01	Illegal Function	Код функції не підтримується веденим пристроєм (сервером).
0x02	Illegal Data Address	Запитувана адреса даних (регістр, дискретний вхід або котушка) не існує або виходить за межі дозволеного діапазону.
0x03	Illegal Data Value	Значення в полі даних є неприпустимим для даної функції (наприклад, неправильна кількість елементів для читання/запису).
0x04	Slave Device Failure	Невідновна помилка веденого пристрою (сервера) під час виконання запиту.
0x05	Acknowledge	Запит прийнято, але виконання триває (не помилка, а статус). Використовується для довготривалих команд програмування.
0x06	Slave Device Busy	Ведений пристрій (сервер) зайнятий обробкою довготривалої команди. Клієнту слід повторити запит пізніше.
0x08	Memory Parity Error	Помилка парності пам'яті веденого пристрою (сервера). Використовується з функціями 20 і 21 для доступу до розширеної пам'яті.

Для ілюстрації обміну розглянемо типовий запит на читання трьох послідовних регістрів зберігання, починаючи з адреси 0x006B (107), від веденого з адресою 0x01. Майстер формує кадр RTU, який складається з байтів [0x01, 0x03, 0x00, 0x6B, 0x00, 0x03, CRC Low, CRC High]. Ведений пристрій, отримавши цей кадр, перевіряє CRC, виконує читання регістрів і відповідає кадром [0x01, 0x03, 0x06, дані_H, дані_L, ..., CRC]. Якщо запит адресує неіснуючий регістр, ведений відповість кадром винятку [0x01, 0x83, 0x02, CRC], де 0x83 = 0x80 + 0x03, а 0x02 — код Illegal Data Address. У лістингу 4.2 наведено фрагмент коду на STM32 із використанням HAL, який формує подібний запит у режимі RTU та обробляє відповідь з урахуванням можливого винятку.

Лістинг 4.2 – Формування Modbus RTU запиту читання регістрів та обробка винятку на STM32 (HAL)

```
#include "main.h"
#include <stdint.h>
#include <string.h>
// ------------------ Константи -------------------------
#define SLAVE_ADDR          0x01    // Адреса веденого пристрою
#define FC_READ_HOLDING     0x03    // Код функції: читання
Holding Registers
#define FC_EXCEPTION_MASK   0x80    // Маска старшого біта для
коду винятку[reference:1]
#define RESPONSE_TIMEOUT_MS 100     // Таймаут очікування
```

```
відповіді (мс)
#define BUF_SIZE           256     // Розмір буфера для кадру
extern UART_HandleTypeDef huart1;  // UART для Modbus RTU
extern TIM_HandleTypeDef htim3;  // Таймер для міжсимвольного
таймауту 3.5T
static uint8_t tx_buf[BUF_SIZE];   // Буфер передачі
static uint8_t rx_buf[BUF_SIZE];   // Буфер прийому
static uint16_t rx_len = 0;        // Кількість прийнятих
байтів
static uint8_t frame_ready = 0;    // Флаг готовності кадру
// Прототип функції CRC з лістингу 4.1
extern uint16_t modbus_crc16(uint8_t *buf, uint16_t len);
// ----------------------------------------------------
// @brief   Формування Modbus RTU запиту читання Holding
Registers (FC=0x03)
// @param   addr       Адреса веденого пристрою (1-247)
// @param   start_reg  Початкова адреса регістра (0-65535)
// @param   num   Кількість регістрів для читання (1-125)
// @param   out  Буфер для збереження отриманих даних (16-
бітних значень)
// @return  0 - успішно, >0 - код винятку, -1 - помилка
таймауту/CRC
// @note    Відповідно до MODBUS over Serial Line V1.02[ref
erence:2]
// ----------------------------------------------------
int modbus_rtu_read_regs(uint8_t addr, uint16_t start_reg,
uint16_t num, uint16_t *out)
{
    uint16_t crc;
    uint8_t resp_addr, resp_fc, resp_len;
    uint16_t resp_crc, calc_crc;
    int idx = 0;
    // ---- 1. Формування кадру запиту (RTU ADU) --------
    tx_buf[0] = addr;                        // Адреса веденого
    tx_buf[1] = FC_READ_HOLDING;  // Код функції 0x03
    tx_buf[2] = (start_reg >> 8) & 0xFF;   // Старший байт
адреси регістра
    tx_buf[3] = start_reg & 0xFF;  // Молодший байт адреси
регістра
    tx_buf[4] = (num >> 8) & 0xFF;  // Старший байт кількості
регістрів
    tx_buf[5] = num & 0xFF;  // Молодший байт кількості
регістрів
    // Обчислення CRC для 6 байтів (без CRC-поля)
    crc = modbus_crc16(tx_buf, 6);
    tx_buf[6] = crc & 0xFF;   // Молодший байт CRC
(передається першим)[reference:3]
    tx_buf[7] = (crc >> 8) & 0xFF;  // Старший байт CRC
    // - 2. Відправка запиту через HAL UART Transmit() --
    HAL_UART_Transmit(&huart1, tx_buf, 8, HAL_MAX_DELAY);
    // - 3. Очікування відповіді з таймаутом ----
    frame_ready = 0;
    rx_len = 0;
    __HAL_TIM_SET_COUNTER(&htim3, 0); // Скидання лічильника
таймера
    HAL_TIM_Base_Start_IT(&htim3);// Запуск таймера для 3.5T
    uint32_t start_tick = HAL_GetTick();
```

```
    while (!frame_ready) {
        if ((HAL_GetTick() - start_tick) > RE
SPONSE_TIMEOUT_MS) {
            HAL_TIM_Base_Stop_IT(&htim3);
            return -1; // Помилка таймауту        }    }
    HAL_TIM_Base_Stop_IT(&htim3);
    // ---- 4. Перевірка мінімальної довжини кадру -----
    if (rx_len < 5) return -1;  // Занадто короткий кадр
(адреса + FC + ... + CRC)
    // ------- 5. Перевірка CRC отриманого кадру -------
    calc_crc = modbus_crc16(rx_buf, rx_len - 2);
    resp_crc = ((uint16_t)rx_buf[rx_len - 1] << 8) |
rx_buf[rx_len - 2];
    if (calc_crc != resp_crc) return -1;   // Помилка CRC
    // ---- 6. Аналіз коду функції у відповіді ------
    resp_addr = rx_buf[0];
    resp_fc  = rx_buf[1];
    if (resp_addr != addr) return -1; // Неправильна адреса
    // Якщо старший біт коду функції встановлено – це
виняток[reference:4]
    if (resp_fc & FC_EXCEPTION_MASK) {
        // Код винятку передається в наступному байті
        return rx_buf[2];//0x01,0x02,0x03,0x04
тощо[reference:5]
    }
    if (resp_fc != FC_READ_HOLDING) return -1; // Невідомий
код функції
    // ---- 7. Розбір даних (Holding Registers) -------
    resp_len = rx_buf[2];    // Кількість байтів даних
    if (rx_len != (resp_len + 5)) return -1; // Перевірка
довжини кадру
    if (resp_len != num * 2) return -1;    // Перевірка
кількості регістрів
    for (int i = 0; i < num; i++) {
        out[i] = ((uint16_t)rx_buf[3 + i*2] << 8) | rx_buf[4
+ i*2];
    }    return 0;  // Успішно   }
// ============================================================
// ОБРОБНИК ПЕРИВАННЯ UART ДЛЯ ПРИЙОМУ З МІЖСИМВОЛЬНИМ
ТАЙМАУТОМ (3.5T)
// ============================================================
// @brief   Переривання від UART – прийом одного байта
// @note    Кожен прийнятий байт зберігається в буфері, таймер
3.5T скидається
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{    if (huart->Instance == USART1) {
        if (rx_len < BUF_SIZE) {
            rx_buf[rx_len++] = rx_byte; // Зберегти байт
        }
        // Скидання таймера 3.5T – кожен новий байт подовжує
очікування кінця кадру[reference:6]
        __HAL_TIM_SET_COUNTER(&htim3, 0);
        // Запуск прийому наступного байта
        HAL_UART_Receive_IT(&huart1, &rx_byte, 1);    }   }
// ------------------------------------------------------------
// @brief Переривання від таймера – міжсимвольний таймаут
3.5T
```

```
125:// @note  Якщо за 3.5 символьних інтервали не надійшов новий
126:байт,
127://    кадр вважається завершеним[reference:7][reference:8]
128:// ----------------------------------------------------
129:void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
130:{    if (htim->Instance == TIM3) {
131:         HAL_TIM_Base_Stop_IT(htim);     // Зупинити таймер
132:         frame_ready = 1;       // Позначити кадр як готовий
 133:    }    }
```

Мікроконтролерна реалізація Modbus-протоколу часто спирається на готові бібліотеки, такі як FreeMODBUS, що доступна для більшості популярних платформ. Однак у ресурсообмежених системах розробники нерідко пишуть власний мінімальний стек, обмежуючись підтримкою кількох ключових функцій (03, 06, 16). Незалежно від підходу, критичним залишається дотримання часових вимог RTU: міжсимвольні та міжкадрові паузи повинні жорстко контролюватися апаратними таймерами, оскільки їх порушення призводить до десинхронізації шини. Сучасні мікроконтролери, такі як STM32F4 або ESP32, мають достатньо апаратних таймерів і DMA, щоб повністю автоматизувати цей процес, звільняючи центральний процесор для виконання основної логіки керування.

Протокол Modbus, незважаючи на свій поважний вік, залишається надійним і гнучким інструментом для побудови розподілених промислових систем. Його стійкість до помилок, забезпечена CRC-16 або LRC, та розвинений механізм виняткових ситуацій роблять його придатним навіть для критичних застосувань. У поєднанні з диференційним фізичним рівнем RS-485, який розглядатиметься далі, Modbus RTU утворює перевірений часом фундамент для недорогих і надійних багатоточкових мереж, що експлуатуються в цехах, на електростанціях і в транспортній інфраструктурі. Наступні підрозділи детально розкриють питання захисту від перенапруг, електромагнітної сумісності та практичних рекомендацій щодо прокладання кабелів згідно з IEC 61000.

4.1.2. Захист від перенапруг та електромагнітних завад згідно IEC 61000

Надійність роботи промислових мереж на основі RS-485 і Modbus RTU визначається не лише коректністю протокольної реалізації, а й здатністю апаратного забезпечення витримувати жорсткі електромагнітні впливи. У цехових умовах сигнальні лінії прокладаються поблизу силових кабелів, зварювальних апаратів, частотних перетворювачів, комутаційної

апаратури, тому навіть найдосконаліший диференційний приймач може виявитися безсилим перед імпульсними перенапругами або кондуктивними завадами, що на кілька порядків перевищують рівень корисного сигналу. Міжнародна серія стандартів IEC 61000, зокрема її частини 4-2, 4-4, 4-5 та 4-6, встановлює методи випробувань та критерії стійкості обладнання до таких впливів, що робить її обов'язковою нормативною базою для будь-якого проекту, орієнтованого на промислову експлуатацію. У цьому підрозділі систематизовано основні види загроз і схемотехнічні засоби захисту, які дозволяють привести вузол RS-485 у відповідність до вимог зазначених стандартів.

Перенапруги, викликані розрядами блискавки або комутацією потужного обладнання, нормуються стандартом IEC 61000-4-5 (Surge immunity test). Випробувальний імпульс має форму 1,2/50 мкс за напругою та 8/20 мкс за струмом, імітуючи енергію, здатну пошкодити напівпровідникові компоненти. Для захисту входів диференційного трансивера застосовують каскадну схему, де першим рубежем слугує газорозрядний розрядник (Gas Discharge Tube, GDT), здатний відвести струми порядку кілоамперів, але з відносно повільним спрацьовуванням. Другим каскадом встановлюється швидкодіючий TVS-діод (Transient Voltage Suppressor), який обмежує напругу до безпечного для мікросхеми рівня (зазвичай 6–12 В) за одиниці наносекунд. Між GDT і TVS необхідно розмістити струмообмежувальний резистор або феритовий дріт, який узгоджує енергетичні характеристики обох каскадів. Таку триланкову структуру, що стала галузевим стандартом для RS-485, наведено на рисунку 4.2 із позначенням конкретних компонентів і шляхів протікання струму під час дії імпульсу.

Швидкі перехідні завади (Electrical Fast Transient, EFT), спричинені комутацією індуктивних навантажень, стандартизовано в IEC 61000-4-4. Вони являють собою пачки імпульсів тривалістю одиниці наносекунд із частотою повторення до 100 кГц, які наводяться на сигнальні лінії через паразитну ємність. На відміну від поодинокого потужного імпульсу Surge, EFT-завада проникає навіть через невеликі паразитні зв'язки, тому для її придушення недостатньо лише обмежувачів напруги; потрібне частотне фільтрування. Найефективнішим засобом є синфазний дросель (common-mode choke), який створює високий імпеданс для синфазних шумів, не впливаючи на диференційний сигнал. Додатково на входи приймача встановлюють керамічні конденсатори невеликої ємності (47–100 пФ) для шунтування високочастотних складових. Виробники трансиверів, зокрема Analog Devices у технічній нотатці AN-960 «RS-485/RS-422 Circuit

Implementation Guide», рекомендують розташовувати фільтруючі компоненти якомога ближче до виводів мікросхеми, щоб мінімізувати площу паразитної антени.

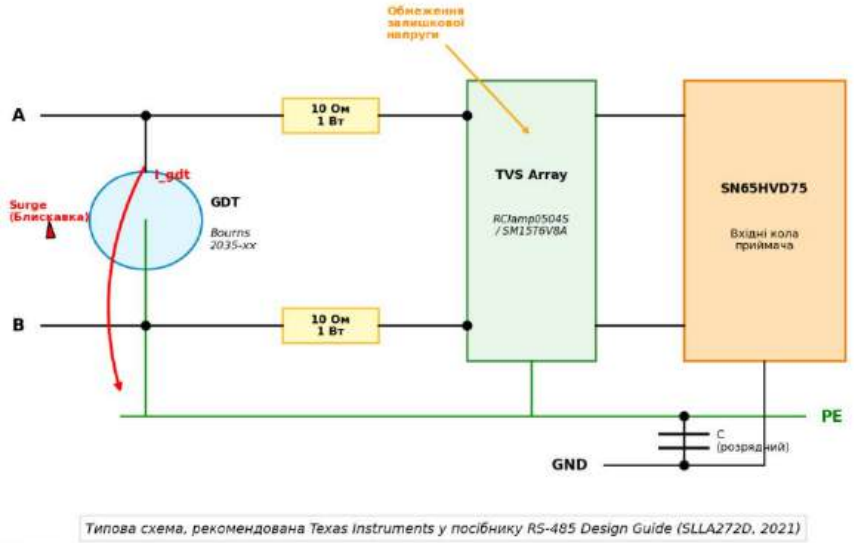


Рисунок 4.2 – Триланкова схема захисту порту RS-485 від перенапруг згідно IEC 61000-4-5

Електростатичні розряди (ESD), що виникають при дотику обслуговуючого персоналу до конекторів або при підключенні кабелів, згідно з IEC 61000-4-2 досягають напруги 8 кВ (контактний розряд) і 15 кВ (повітряний). Сучасні інтерфейсні мікросхеми часто мають вбудований ESD-захист рівня ±15 кВ, однак за жорстких промислових вимог доцільно встановлювати зовнішні малоємнісні TVS-діодні збірки (наприклад, Nexperia PESD5V0L2BT), які гарантовано витримують багаторазові розряди. Ці діоди мають типову ємність менше 2 пФ, що не спотворює сигнал навіть на частотах передавання 10 Мбіт/с.

Кондуктивні завади, наведені радіочастотними полями, регламентуються IEC 61000-4-6 і охоплюють діапазон частот від 150 кГц до 80 МГц. Їх дія призводить до паразитної демодуляції на нелінійностях захисних діодів, що може спричинити хибні спрацьовування приймача. Основним способом боротьби з кондуктивними завадами є гальванічна розв’язка шини за допомогою ізольованих трансиверів (наприклад, ISO1412 від Texas Instruments або ADM2483 від Analog Devices), які поєднують у собі дифе-

ренційний драйвер, ізоляційний бар'єр і DC/DC-перетворювач для живлення ізольованого боку. Ізоляція руйнує контури протікання синфазних струмів, що радикально підвищує стійкість до завад. Якщо ж з економічних міркувань застосовується неізольований трансивер, слід передбачити синфазний дросель та розділовий конденсатор між локальною землею і захисною землею (PE).

Узагальнену схему комплексного захисту вузла RS-485, яка забезпечує відповідність усім згаданим стандартам, подано в таблиці 4.3, де кожному типу загрози зіставлено рекомендований компонент і його типовий представник. Слід зауважити, що в реальному проектуванні необхідно не лише обрати правильні компоненти, а й забезпечити низькоімпедансний шлях для струмів розряду на корпусну землю (PE), уникаючи проходження цих струмів через сигнальний шар плати. Для цього на платі виділяють окремий полігон захисної землі, з'єднаний із контуром заземлення шафи через стійку або гвинт.

Практичні рекомендації з монтажу, сформульовані в прикладній нотатці AN4312 «Designing robust RS-485 interfaces» від STMicroelectronics (2020), наголошують, що всі захисні компоненти мають розташовуватися на відстані не більше 5 мм від роз'єму, а доріжки до них повинні бути максимально короткими й широкими. Екранування кабелю слід заземлювати з обох кінців для високочастотних завад і з одного кінця — для низькочастотних, уникаючи протікання зрівнювальних струмів через екран. Крім того, за специфікацією TIA/EIA-485-A на обох кінцях шини обов'язково встановлюють термінуючі резистори 120 Ом, які також виконують роль елемента узгодження, зменшуючи відбиття, що можуть помилково сприйматися як імпульсні завади.

Захист відповідно до IEC 61000 є не окремою опцією, а невід'ємною складовою проектування промислових інтерфейсів. Розглянута триланкова схема захисту, доповнена синфазними фільтрами та, за потреби, гальванічною ізоляцією, дозволяє забезпечити надійну багаторічну роботу мереж RS-485/Modbus у найсуворіших умовах експлуатації. Студент, який опанував цей матеріал, має усвідомити, що навіть найкращий програмний протокол безсилий без грамотного апаратного захисту, і що інженерна культура вимагає приділяти питанням електромагнітної сумісності не менше уваги, ніж написанню коду. Наступний розділ посібника, присвячений CAN-шинам, продемонструє, як аналогічні принципи диференційної передачі та захисту реалізовані в ще більш завадостійкому і детермінованому протоколі.

Таблиця 4.3 – Компоненти комплексного захисту порту RS-485

Тип загрози	Характеристика	Значення
Перенапруга (Surge)	*Стандарт IEC*	IEC 61000-4-5
	Захисний компонент	Триланкова схема (GDT + резистор + TVS)
	Приклад компонента	Bourns 2035-xx-SM + CDSOT23-SM712
	Примітка	GDT серії 2035 витримує імпульсний струм до 5000 А (5 кА); TVS CDSOT23-SM712 розроблено спеціально для RS-485
Швидкі перехідні (EFT/Burst)	*Стандарт IEC*	IEC 61000-4-4
	Захисний компонент	Синфазний дросель + Феритові намиста
	Приклад компонента	Würth Elektronik 744227 (WE-FC серія) + Murata BLM18
	Примітка	744227: 51 мкГн, 5.5 кОм @ 10 МГц, 1 А; придушує синфазні завади в діапазоні 1–100 МГц
Електростатичний розряд (ESD)	*Стандарт IEC*	IEC 61000-4-2
	Захисний компонент	Малоємнісна TVS-діодна збірка
	Приклад компонента	Nexperia PESD5V0L2BT (SOT23)
	Примітка	Захист двох ліній; ємність < 2 пФ (тип. 75 пФ); ESD захист до ±30 кВ (IEC 61000-4-2, рівень 4)
Кондуктивні завади (RF)	*Стандарт IEC*	IEC 61000-4-6
	Захисний компонент	Гальванічна розв'язка або синфазний дросель + конденсатор
	Приклад компонента	ISO1412 (Texas Instruments) або ADM2483 (Analog Devices)
	Примітка	ISO1412 – ізольований RS-485 приймач/передавач; ADM2483 з технологією iCoupler® забезпечує ізоляцію 2.5 кВ; усуває контури завад та гальванічну розв'язку

4.2. CAN (Controller Area Network) 2.0 та CAN FD (Flexible Data Rate)

Серед усіх польових шин, що застосовуються в автомобільній та промисловій електроніці, Controller Area Network (CAN) посідає особливе місце завдяки унікальному поєднанню надійності, детермінованості та масштабованості. Розроблений для жорстких умов транспортних засобів,

CAN забезпечує пріоритетний доступ до шини без втрати часу на колізії, що робить його ідеальним для систем реального часу, де затримка в мілісекунди може коштувати безпеки. Класичний протокол CAN 2.0, формалізований у ISO 11898, став галузевим стандартом для електронних блоків керування двигуном, гальмівних систем і датчиків. Проте зростання обсягів даних (оновлення прошивок, передавання конфігурацій, потоки з камер і радарів) спонукало до появи CAN FD (Flexible Data Rate), який зберігає арбітражний механізм, але збільшує швидкість фази даних до 8 Мбіт/с і розширює корисне навантаження до 64 байтів. Сьогодні наступне покоління — CAN XL, специфіковане в CiA 610-1, — уже виходить на ринок, пропонуючи гігабітні швидкості та тунелювання Ethernet-кадрів, що відкриває шлях до уніфікованої архітектури транспортних засобів. У цьому розділі детально аналізуються апаратна реалізація CAN-контролерів, фізичні стани шини, побітовий арбітраж, а також особливості впровадження CAN FD і перспективи CAN XL, спираючись на чинні міжнародні стандарти та технічну документацію провідних виробників.

4.2.1. Апаратний контролер, домінантні/рецесивні стани, арбітраж

Протокол CAN (Controller Area Network), розроблений компанією Robert Bosch GmbH у середині 1980-х років і вперше представлений у серійному автомобілі Mercedes-Benz W140 у 1991 році, нині є не лише стандартом де-факто для автомобільної електроніки, а й фундаментом для численних промислових протоколів верхнього рівня — CANopen, DeviceNet, J1939. Міжнародний стандарт ISO 11898-1:2015 визначає рівень каналу передавання даних (Data Link Layer, DLL) та фізичний рівень для швидкостей до 1 Мбіт/с, а у 2012 році Bosch представив специфікацію CAN FD (Flexible Data Rate), яка була інтегрована в ISO 11898-1:2015 як додаткова можливість. Фундаментальною відмінністю CAN від усіх розглянутих раніше інтерфейсів є його орієнтація на повідомлення, а не на адреси вузлів: кожен кадр містить унікальний ідентифікатор, який одночасно є пріоритетом у разі конкуренції за шину. Ця властивість у поєднанні з неразрушуючим побітовим арбітражем забезпечує детермінований доступ до середовища передавання, що критично важливо для систем безпеки, керування двигуном, антиблокувальних гальм та інших застосувань реального часу. У цьому підрозділі детально розглядаються апаратна організація CAN-вузла, електричні стани шини та механізм арбітражу, як це визначено в оригінальній специфікації Bosch CAN Specification 2.0 (1991), стандарті ISO 11898-1:2015 та додаткових рекомендаціях організації CAN in Automation (CiA).

Апаратний CAN-контролер у сучасній мікроконтролерній системі є спеціалізованим периферійним блоком, що реалізує повний протокольний автомат канального рівня. До його складу входять: блок синхронізації та відновлення тактової частоти (Bit Timing Logic), який налаштовується під конкретну швидкість передавання; апаратний генератор та перевірник CRC; фільтри ідентифікаторів, що дозволяють приймати лише потрібні повідомлення, не завантажуючи центральний процесор; а також буферна пам'ять, організована у вигляді поштових скриньок (mailboxes) або FIFO-черг. У мікроконтролерах сімейства STM32F4/F7 застосовується модуль bxCAN (Basic Extended CAN), що підтримує протокол CAN 2.0B з 11- та 29-бітними ідентифікаторами. Старші сімейства STM32H7, STM32G4, STM32L5 обладнані модулем FDCAN, який відповідає ISO 11898-1:2015 і додатково підтримує CAN FD. Документація RM0468 (для STM32H723/733) описує FDCAN як апаратний блок із 64 виділеними буферами прийому/передачі, гнучкими фільтрами на 128 елементів і підтримкою швидкостей до 8 Мбіт/с у фазі даних. Для мікроконтролерів, що не мають вбудованого CAN-контролера (наприклад, Raspberry Pi Pico на RP2040), застосовують зовнішні контролери, такі як MCP2515 (Microchip) або MCP2517FD (CAN FD), що підключаються через SPI. Структурна схема типового CAN-вузла з внутрішнім контролером та зовнішнім трансивером показана на рисунку 4.3.

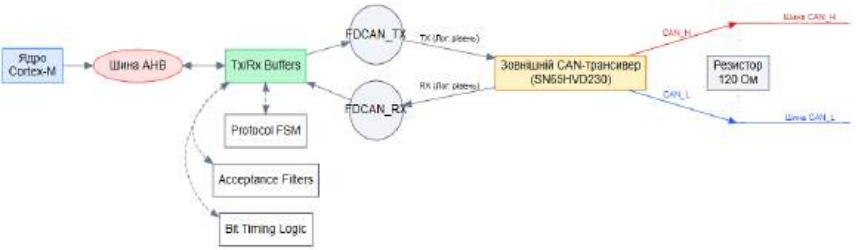


Рисунок 4.3 – Структурна схема CAN-вузла на базі мікроконтролера STM32 із вбудова-ним FDCAN

Фізичний рівень CAN базується на двох станах шини: домінантному (Dominant) та рецесивному (Recessive), що є ключем до розуміння як передавання даних, так і механізму арбітражу. Рецесивний стан відповідає логічній одиниці: обидва проводи CAN_H та CAN_L перебувають приблизно на одному потенціалі (типово 2,5 В), диференційна напруга V_diff = V(CAN_H) – V(CAN_L) близька до нуля. Жоден вузол активно не

керує шиною в цьому стані; рівень підтримується пасивно за рахунок резисторів зсуву всередині трансиверів. Домінантний стан відповідає логічному нулю: трансивер, що передає, активно піднімає напругу CAN_H до 3,5 В і опускає CAN_L до 1,5 В, створюючи диференційну напругу близько 2 В. Якщо хоча б один вузол встановлює домінантний рівень, шина переходить у домінантний стан, незалежно від того, скільки інших вузлів намагаються залишити її в рецесивному. Така поведінка реалізує логіку «монтажного І» (wired-AND): домінантний біт (0) «перемагає» рецесивний (1). Саме на цій властивості ґрунтується арбітраж.

Арбітраж у CAN є неразрушуючим (non-destructive) і побітовим: усі вузли, що бажають передавати, починають це робити одночасно після закінчення міжкадрового проміжку, виставляючи свої ідентифікатори біт за бітом. Кожен передавач одночасно слухає шину. Якщо вузол передає рецесивний біт, але виявляє на шині домінантний рівень, це означає, що інший вузол передав домінантний біт (логічний 0), тобто його ідентифікатор має вищий пріоритет (менше числове значення). Вузол, що програв арбітраж, негайно припиняє передавання і переходить у режим приймача, не спотворюючи повідомлення переможця. Арбітраж триває протягом усього поля ідентифікатора та біта RTR/SRR. Оскільки ідентифікатор є унікальним для кожного типу повідомлень, саме він, а не адреса вузла, визначає пріоритетність: повідомлення від датчика швидкості обертання коліс (ABS) матиме менший ідентифікатор, ніж команда регулювання сидіння, і гарантовано виграє арбітраж. Часову діаграму процесу арбітражу між трьома вузлами з ідентифікаторами 0x100, 0x200 та 0x300 показано на рисунку 4.4. Як видно з діаграми, вузол із найменшим ідентифікатором безперешкодно завершує кадр, тоді як інші відступають без втрати часу на повторні спроби.

У CAN FD процедура арбітражу залишається ідентичною класичному CAN: вона відбувається на стандартній швидкості (до 1 Мбіт/с) і охоплює поля SOF, ідентифікатора, RRS (Remote Request Substitution, аналог RTR) та біта FDF (FD Format indicator), який сигналізує, що далі кадр може перейти в режим FD. Після завершення арбітражу переможець може підвищити швидкість передавання для фази даних (до 8 Мбіт/с і вище), оскільки подальші поля (BRS, ESI, DLC, Data, CRC) уже не потребують одночасного моніторингу шини іншими вузлами.

Поле BRS (Bit Rate Switch) є ключовим: якщо воно встановлене в рецесивний стан, швидкість перемикається на заздалегідь узгоджену більш високу. У таблиці 4.4 зіставлено основні параметри класичного CAN 2.0 та CAN FD, включно з максимальною довжиною даних і пропускною здатністю.

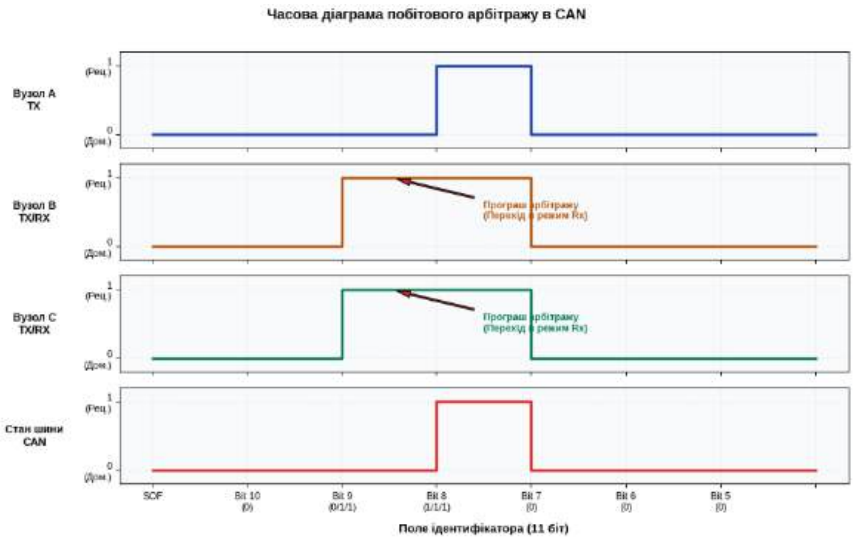


Рисунок 4.4 – Часова діаграма побітового арбітражу в CAN

Таблиця 4.4 – Порівняння CAN 2.0 та CAN FD

Параметр	*CAN 2.0 (ISO 11898-1:2003)*	*CAN FD (ISO 11898-1:2015)*
Максимальна швидкість арбітражу	1 Мбіт/с	1 Мбіт/с (арбітраж)
Максимальна швидкість даних	1 Мбіт/с	до 8 Мбіт/с (фаза даних)
Максимальна довжина поля даних	8 байт	до 64 байт
Контрольна сума	CRC-15	CRC-17 або CRC-21 (залежно від довжини)
Типи кадрів	Data, Remote	Data (кадри CAN FD)
Сумісність	повністю сумісний з CAN FD у режимі 2.0	CAN FD-вузли можуть приймати кадри 2.0

Для програмування CAN на STM32 з використанням HAL достатньо налаштувати структуру FDCAN_HandleTypeDef, визначити часові параметри (номінальний бітрейт для арбітражу та, за потреби, бітрейт даних для FD), активувати фільтри і потім використовувати функції HAL_FDCAN_AddMessageToTxFifoQ() та HAL_FDCAN_GetRxMessage(). У лістингу 4.3 наведено фрагмент коду ініціалізації FDCAN на STM32H723 у режимі CAN FD зі швидкістю арбітражу 500 кбіт/с та швидкістю даних 5 Мбіт/с, що є типовою конфігурацією для автомобільних та промислових мереж.

Лістинг 4.3 – Ініціалізація FDCAN на STM32H723 у ре-жимі CAN FD (HAL)

```
// Відповідно до рекомендацій CiA 601-3 та STM32H7 Reference Manual
#include "fdcan.h"
// ----------- Глобальний дескриптор FDCAN --------------
FDCAN_HandleTypeDef hfdcan1;
// 1. ІНІЦІАЛІЗАЦІЯ FDCAN З БІТОВИМИ СЕГМЕНТАМИ
/** * @brief Ініціалізація FDCAN1 у режимі CAN FD з BRS (Bit Rate Switch)
 * @note  Тактування FDCAN: 80 МГц
 *        Арбітражна фаза (номінальна): 500 кбіт/с
 *        Фаза даних: 5 Мбіт/с
 * @note  Розрахунок бітових сегментів згідно з CiA 601-3:
 *        Tq = Prescaler / fCLK
 *        Номінальний час біта = (TSEG1 + TSEG2 + 3) * Tq
 *        Точка вибірки = (TSEG1 + 1) / (TSEG1 + TSEG2 + 2) */
void MX_FDCAN1_Init(void)
{
    hfdcan1.Instance = FDCAN1;
    // ---- Номінальна фаза (арбітраж) – 500 кбіт/с -----
    hfdcan1.Init.NominalPrescaler = 8;  // Дільник: 80 МГц / 8 = 10 МГц (Tq = 100 нс)
    hfdcan1.Init.NominalSyncJumpWidth = 1; // SJW = 1 Tq
    hfdcan1.Init.NominalTimeSeg1 = 7;  // TSEG1 = 7 Tq (фаза 1)
    hfdcan1.Init.NominalTimeSeg2 = 2;     // TSEG2 = 2 Tq (фаза 2)
  // Час біта = (7 + 2 + 3) * 100 нс = 12 * 100 нс = 1200 нс → 500 кбіт/с
    // Точка вибірки = (7 + 1) / (7 + 2 + 2) = 8 / 11 ≈ 72.7%
    // ---------- Фаза даних (CAN FD) – 5 Мбіт/с ----------
    hfdcan1.Init.DataPrescaler      = 2; // Дільник: 80 МГц / 2 = 40 МГц (Tq = 25 нс)
    hfdcan1.Init.DataSyncJumpWidth  = 1;     // SJW = 1 Tq
    hfdcan1.Init.DataTimeSeg1 = 5;     // TSEG1 = 5 Tq
    hfdcan1.Init.DataTimeSeg2 = 2;     // TSEG2 = 2 Tq
    // Час біта = (5 + 2 + 3) * 25 нс = 10 * 25 нс = 250 нс
 5 Мбіт/с
    // Точка вибірки = (5 + 1) / (5 + 2 + 2) = 6 / 9 ≈ 66.7%
    // ---------- Загальні параметри ----------
    hfdcan1.Init.FrameFormat = FDCAN_FRAME_FD_BRS;  // CAN FD з BRS [0†L11-L12]
    hfdcan1.Init.Mode = FDCAN_MODE_NORMAL;   // Нормальний режим
    hfdcan1.Init.AutoRetransmission = DISABLE;
    hfdcan1.Init.TransmitPause = DISABLE;
    hfdcan1.Init.ProtocolException = DISABLE;
    // ---------- Налаштування Message RAM ----------
    hfdcan1.Init.MessageRAMOffset = 0;
    hfdcan1.Init.StdFiltersNbr = 1; // 1 стандартний фільтр
    hfdcan1.Init.ExtFiltersNbr = 0;
    hfdcan1.Init.RxFifo0ElmtsNbr = 2;
    hfdcan1.Init.RxFifo0ElmtSize = FDCAN_DATA_BYTES_64; // 64 байти для CAN FD
    hfdcan1.Init.TxEventsNbr = 1;
```

```
    hfdcan1.Init.TxBuffersNbr = 1;
    hfdcan1.Init.TxFifoQueueElmtsNbr = 2;
    hfdcan1.Init.TxFifoQueueMode = FDCAN_TX_FIFO_OPERATION;
    hfdcan1.Init.TxElmtSize = FDCAN_DATA_BYTES_64;
    // Ініціалізація FDCAN
    if (HAL_FDCAN_Init(&hfdcan1) != HAL_OK) {
        Error_Handler();    }   }
// 2. НАЛАШТУВАННЯ ФІЛЬТРА ПРИЙОМУ
// =======================================================
/**
 * @brief Налаштування фільтра для прийому повідомлень з
конкретним ID
 * @note  Фільтр налаштовано в режимі маски (
FDCAN FILTER MASK) */
void MX_FDCAN1_Filter_Config(void)
{
    FDCAN_FilterTypeDef sFilterConfig;
    sFilterConfig.IdType= FDCAN_STANDARD_ID; //11-бітний ID
    sFilterConfig.FilterIndex = 0;
    sFilterConfig.FilterType=FDCAN_FILTER_MASK; //Масковий
режим [8↑L8]
    sFilterConfig.FilterConfig = FDCAN_FILTER_TO_RXFIFO0; //
Направити до RX FIFO 0
    sFilterConfig.FilterID1 = 0x7E8; // ID для прийому
    sFilterConfig.FilterID2 = 0x7E8; // Маска (0x7E8 = про
пускати тільки цей ID)
    sFilterConfig.RxBufferIndex     = 0;
    if (HAL_FDCAN_ConfigFilter(&hfdcan1, &sFilterConfig) !=
HAL_OK) {        Error_Handler();    }   }
// 3. АКТИВАЦІЯ ПЕРИВАНЬ
// =======================================================
/** * @brief Активація сповіщень про прийом та завершення
передачі */
void MX_FDCAN1_Activate_Interrupts(void)
{
    // Активація переривань: прийом у RX FIFO 0 та завершення
передачі
    if (HAL_FDCAN_ActivateNotification(&hfdcan1,
FDCAN_IT_RX_FIFO0_NEW_MESSAGE |
FDCAN_IT_TX_COMPLETE,
0) != HAL_OK) {        Error_Handler();    }   }
// 4. ВІДПРАВЛЕННЯ ПОВІДОМЛЕННЯ З 64-БАЙТНИМ ПОЛЕМ ДАНИХ
// =======================================================
/** * @brief Відправлення CAN FD повідомлення з 64 байтами даних
 * @param id           Ідентифікатор повідомлення
 * @param pData        Вказівник на масив даних (64 байти)
 * @param len          Довжина даних (максимум 64)
 * @retval HAL_OK      Успішно
 * @retval HAL_ERROR   Помилка */
HAL_StatusTypeDef FDCAN_Send_64Bytes(uint32_t id, uint8_t
*pData, uint8_t len)
{    FDCAN_TxBufferTypeDef TxBuffer;
    TxBuffer.Identifier          = id;
    TxBuffer.IdType              = FDCAN_STANDARD_ID;
    TxBuffer.TxFrameType         = FDCAN_DATA_FRAME;
    TxBuffer.DataLength          = len;
    TxBuffer.ErrorStateIndicator = DISABLE;
```

```
    TxBuffer.BitRateSwitch       = ENABLE;      // BRS = 1 → перехід на 5 Мбіт/с у фазі даних
    TxBuffer.FDFormat            = ENABLE;      // CAN FD формат
    TxBuffer.TxEventFifoControl  = DISABLE;
    TxBuffer.MessageMarker       = 0;
    // Копіювання до 64 байт даних
    for (uint8_t i = 0; i < len && i < 64; i++) {
        TxBuffer.Data[i] = pData[i];
    }
    // Відправлення через буфер передачі
    return HAL_FDCAN_AddMessageToTxBuffer(&hfdcan1, &TxBuffer, 0);
}
// 5. ОБРОБНИК ПЕРИВАНЬ (приклад)
// ================================================
/** * @brief Callback при отриманні нового повідомлення в RX FIFO 0 */
void HAL_FDCAN_RxFifo0Callback(FDCAN_HandleTypeDef *hfdcan,
                               uint32_t RxFifo0Index)
{    FDCAN_RxBufferTypeDef RxBuffer;
    if (HAL_FDCAN_GetRxMessage(hfdcan, RxFifo0Index, &RxBuffer) == HAL_OK) {
        // Обробка отриманих даних (до 64 байт)
        // RxBuffer.Data[0..63]    }  }
```

Апаратний контролер CAN у поєднанні з логікою домінантних/рецесивних станів та неразрушуючим арбітражем утворюють основу для надзвичайно надійної та детермінованої багатомайстерної шини. Саме ці властивості зробили CAN безальтернативним вибором для автомобільної електроніки та критично важливих систем керування. Наступні підрозділи детальніше розкриють особливості CAN FD — збільшену пропускну здатність і сумісність із класичними вузлами — а також окреслять перспективи CAN XL для гігабітних швидкостей у рамках специфікації CiA 610-1.

4.2.2. CAN FD: збільшена швидкість фази даних до 8 Мбіт/с та довжина поля даних 64 байти

Класичний протокол CAN 2.0, що протягом трьох десятиліть залишався незмінним фундаментом автомобільної та промислової автоматизації, на початку 2010-х років почав вичерпувати свій потенціал. Основні обмеження — максимальна швидкість передавання 1 Мбіт/с та корисне навантаження лише 8 байтів на кадр — перетворилися на критичне вузьке місце для нових застосувань, таких як оновлення програмного забезпечення електронних блоків керування через діагностичний роз'єм (FOTA/SOTA), передавання даних від систем допомоги водієві (ADAS), мультимедійних потоків і злиття сенсорної інформації. Пропускна здатність класичного CAN,

що в найкращому разі сягає 500–600 кбіт/с корисних даних, виявилася недостатньою для передавання багатомегабайтних образів прошивки або потокового відео, а збільшення швидкості арбітражу понад 1 Мбіт/с неможливе через фундаментальні обмеження, пов'язані з часом поширення сигналу в мережі. У відповідь на ці виклики компанія Robert Bosch GmbH у 2012 році представила специфікацію CAN з гнучкою швидкістю передавання даних (CAN FD — Flexible Data Rate), яка була інтегрована в міжнародний стандарт ISO 11898-1:2015. CAN FD зберігає механізм арбітражу, фізичний рівень і базовий формат кадру класичного CAN, але додає можливість збільшити обсяг корисних даних до 64 байтів і підвищити бітову швидкість під час передавання фази даних до значень, що на практиці сягають 5–8 Мбіт/с, а в перспективних реалізаціях — до 10–12 Мбіт/с.

Ключовою архітектурною інновацією CAN FD є концепція двох бітових швидкостей у межах одного кадру. Арбітраж, поля підтвердження (ACK) і завершення кадру (EOF) передаються на стандартній номінальній швидкості (Arbitration Bit Rate), яка зазвичай становить від 250 кбіт/с до 1 Мбіт/с і визначається вимогами до довжини шини та кількості вузлів. Після успішного завершення арбітражу та розпізнавання кадру як FD-кадру, вузол-переможець має право перемкнутися на вищу швидкість передавання даних (Data Bit Rate) для полів BRS (Bit Rate Switch), ESI (Error State Indicator), DLC, власне даних і CRC. Перемикання швидкості відбувається синхронно на всіх вузлах мережі, які підтримують FD, за фронтом біта BRS: якщо цей біт встановлено в рецесивний стан, одразу після його зчитування приймачі та передавач перемикають свої генератори бітової синхронізації на заздалегідь сконфігуровані параметри. Після завершення CRC delimitier швидкість повертається до номінальної для полів ACK, EOF та міжкадрового проміжку. Таким чином, в одному кадрі поєднуються сумісність із класичною топологією на етапі арбітражу та висока пропускна здатність на етапі передавання корисних даних. Структуру кадру CAN FD з позначенням швидкісних режимів наведено на рисунку 4.5.



Рисунок 4.5 – Структура кадру CAN FD (стандартний ідентифікатор, 11 біт) з позначен-ням швидкостей

Збільшення максимальної довжини поля даних із 8 до 64 байтів досягається завдяки новій схемі кодування поля DLC (Data Length Code), яка використовує ті самі 4 біти, що й у класичному CAN, але інтерпретує їх інакше. У класичному CAN значення DLC від 0 до 8 відповідають кількості байтів даних, а значення 9–15 заборонені або інтерпретуються як 8. Специфікація CAN FD вводить розширене кодування, наведене в таблиці 4.5: значення 0–8, як і раніше, позначають від 0 до 8 байтів, а значення 9–15 кодують більші обсяги — 12, 16, 20, 24, 32, 48 та 64 байти відповідно.

Таблиця 4.5 – Кодування DLC у CAN FD згідно з ISO 11898-1:2015

Значення DLC (десяткове)	Значення DLC (двійкове)	Кількість байтів даних
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	12
10	1010	16
11	1011	20
12	1100	24
13	1101	32
14	1110	48
15	1111	64

Примітка: значення DLC 0–8 відповідають класичному CAN (відображення 1:1), а розширений діапазон 9–15 розпізнається лише CAN FD-контролерами. Для класичних CAN-контролерів усі значення DLC 9–15 інтерпретуються як 8 байт даних.

Такий підхід дозволив зберегти сумісність на рівні формату кадру та водночас уможливив передавання значних блоків даних, що критично важливо для завантаження прошивок, діагностичних сесій UDS (Unified Diagnostic Services, ISO 14229) та обміну великими масивами калібрувальних параметрів.

Підвищення швидкості у фазі даних до 8 Мбіт/с і вище потребує перегляду не лише протокольного автомата, а й вимог до фізичного рівня. У

класичному CAN бітова синхронізація спирається на жорсткі допуски щодо фронтів сигналу, які забезпечуються відносно повільними переходами між домінантним і рецесивним станами. Для швидкостей понад 1 Мбіт/с такі фронти стають неприйнятними, тому стандарт ISO 11898-2:2016 (частина фізичного рівня для CAN FD) встановлює нові вимоги до трансиверів: зменшений час затримки в петлі (loop delay) до 75 нс (порівняно з 255 нс для класичного), симетричніші характеристики драйверів, а також рекомендовану топологію із суттєво обмеженою довжиною відгалужень. На практиці швидкість 8 Мбіт/с досягається лише в мережах із малою кількістю вузлів (до 10), короткою магістраллю (до 20 м) і трансиверами, що відповідають вимогам CiA 601-1. Для типових автомобільних мереж зі швидкостями 2–5 Мбіт/с допускаються довжини до 40 м за умови ретельного узгодження імпедансів та мінімізації відгалужень. Кількісну залежність максимальної швидкості даних від довжини шини для різних типів трансиверів наведено на рисунку 4.6.

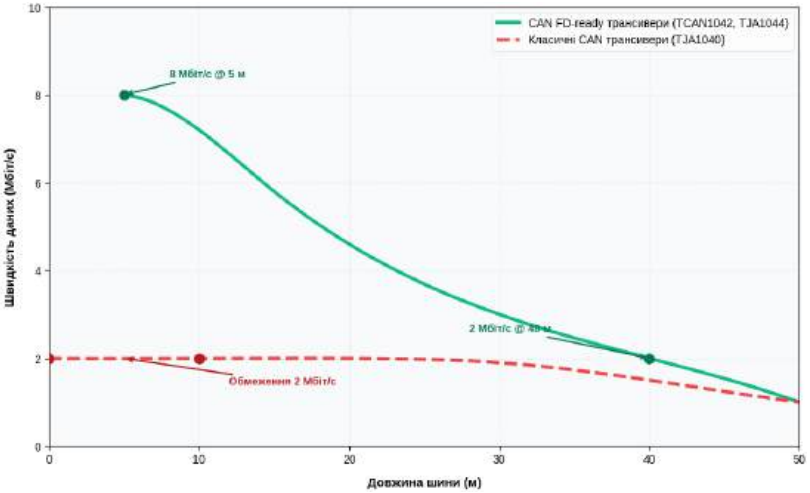


Рисунок 4.6 – Залежність максимальної швидкості даних CAN FD від довжини шини

Пропускна здатність CAN FD суттєво перевищує класичний CAN не лише завдяки вищій швидкості, а й завдяки кращому співвідношенню корисних даних до службових бітів. Для класичного CAN під час передавання 8 байтів даних частка протокольних накладних витрат (ідентифікатор, CRC, ACK, stuffing bits) становить близько 50%, що дає ефективну пропускну здатність близько 500 кбіт/с при швидкості арбітражу 1 Мбіт/с. У CAN FD при передаванні 64 байтів даних на швидкості 5 Мбіт/с (за номінальної

швидкості 500 кбіт/с) корисна швидкість сягає 3,5–4 Мбіт/с, тобто зростає у 7–8 разів. Наближене обчислення ефективної швидкості R_{eff} виконується за формулою (4.1):

$$R_{eff} = (L_{data} \times 8)/(T_{arb} + T_{data}) \ , \qquad (4.1)$$

де L_{data} — кількість байтів даних, T_{arb} — тривалість номінальної частини кадру (арбітраж, контрольні поля до BRS, ACK, EOF), T_{data} — тривалість фази даних, яка залежить від Data Bit Rate. Для ілюстрації порівняльні значення ефективної пропускної здатності для різних конфігурацій зведено в таблицю 4.6. Дані розраховано з урахуванням середньої кількості stuffing-бітів (приблизно 1 stuffing-біт на 10 бітів даних) згідно з методикою, описаною в CiA 601-1.

Таблиця 4.6 – Порівняння ефективної пропускної здатності CAN 2.0 та CAN FD

Конфігурація	Номінальна швидкість (кбіт/с)	Швидкість даних (кбіт/с)	Довжина даних (байт)	Ефективна пропускна здатність (кбіт/с)
CAN 2.0 (8 байт)	1000	—	8	**~580**
CAN FD (8 байт)	500	5000	8	**~450**
CAN FD (64 байти)	500	5000	64	**~3850**
CAN FD (64 байти, 8 Мбіт/с)	1000	8000	64	**~6200**

Зворотна сумісність із класичним CAN є важливою перевагою FD, оскільки дозволяє поступове впровадження нових вузлів у існуючі мережі. CAN FD-контролер повністю підтримує приймання та передавання класичних кадрів CAN 2.0. Класичний CAN-контролер, не розпізнаючи формат FD, сприймає FD-кадр як помилковий і генерує Error Frame, тому в мережі зі змішаними вузлами необхідно або використовувати окремі сегменти, або активувати FD лише для тих вузлів, які гарантовано його підтримують. Розпізнавання FD-кадру відбувається за бітом FDF (FD Format indicator) у полі керування: якщо він рецесивний, кадр є FD, якщо домінантний — класичним. Це дозволяє FD-контролеру в реальному часі перемикатися між форматами без попереднього узгодження.

Практична реалізація CAN FD на мікроконтролерах STM32 вимагає не лише коректного налаштування бітових сегментів, а й вибору трансивера, здатного працювати на цільовій швидкості. У лістингу 4.4 наведено функцію розрахунку параметрів бітової синхронізації для FDCAN з використанням HAL, яка дозволяє отримати номінальний бітрейт 500 кбіт/с та

бітрейт даних 5 Мбіт/с при тактовій частоті модуля 80 МГц. Розрахунок виконується згідно з рекомендаціями CiA 301 (CANopen) та AN5341 (STMicroelectronics).

Лістинг 4.4 – Розрахунок і налаштування бітрейтів CAN FD на STM32H7 (HAL)

```
// Згідно з рекомендаціями CiA 601-3 та STM32H7 Reference Manual (RM0433)
#include "fdcan.h"
// -------- Глобальний дескриптор FDCAN ------------
FDCAN_HandleTypeDef hfdcan1;
// ====================================================
// 1. ФУНКЦІЯ РОЗРАХУНКУ БІТОВИХ СЕГМЕНТІВ
// ====================================================
/**
 * @brief Підбір параметрів бітового таймінгу для досягнення цільового бітрейту
 * @param target_rate  Цільова швидкість (біт/с)
 * @param clk          Тактова частота FDCAN (Гц)
 * @param prescaler  Вказівник на дільник (вихідний пара метр)
 * @param tseg1        Вказівник на TSEG1 (вихідний параметр)
 * @param tseg2        Вказівник на TSEG2 (вихідний параметр)
 * @param sjw    Вказівник на Sync Jump Width (вихідний параметр)
 * @retval 1-успішно,0 - помилка (неможливо досягти бітрейту)
 * @note   Обмеження: tseg2 >= sjw, tseg1 >= tseg2 + sjw
 *         Час біта = (tseg1 + tseg2 + 3) * (prescaler / clk)
 */
uint8_t calculate_bit_timing(uint32_t target_rate, uint32_t clk,
                             uint32_t *prescaler, uint32_t *tseg1,
                             uint32_t *tseg2, uint32_t *sjw)
{
    // Мінімальна кількість Time Quanta на біт (згідно зі специфікацією)
    const uint32_t MIN_TQ = 4;   // CAN FD допускає від 4 до 81 Tq на біт [3†L14-L15]
    const uint32_t MAX_TQ = 81;
    for (uint32_t p = 1; p <= 32; p++) { // Prescaler: 1..32
        for (uint32_t s = 1; s <= 4; s++) { // SJW: 1..4
            for (uint32_t seg2 = 1; seg2 <= 16; seg2++) {
                for (uint32_t seg1 = seg2 + s; seg1 <= 64; seg1++) {
                    // Перевірка обмежень
                    if (seg2 < s) continue;
                    if (seg1 < seg2 + s) continue;
                    uint32_t tq_per_bit = seg1 + seg2 + 3;
                    if (tq_per_bit < MIN_TQ || tq_per_bit > MAX_TQ) continue;
                    uint32_t rate = clk / (p * tq_per_bit);
                    if (rate == target_rate) {
                        *prescaler = p;
                        *tseg1 = seg1;
                        *tseg2 = seg2;
                        *sjw = s;
```

```
                        return 1;  // Успішно
                    }               }           }
        }    }
    return 0;  // Неможливо досягти цільового бітрейту   }
// ======================================================
// 2. НАЛАШТУВАННЯ БІТРЕЙТІВ (НОМІНАЛЬНИЙ ТА ДАНИХ)
// ======================================================
void MX_FDCAN1_Init(void)
{
    uint32_t prescaler, tseg1, tseg2, sjw;
    hfdcan1.Instance = FDCAN1;
    // -- Розрахунок для номінальної швидкості (арбітраж) -
500 кбіт/с --
    if (!calculate_bit_timing(500000, 80000000, &prescaler,
&tseg1, &tseg2, &sjw)) {
        Error_Handler();  // Помилка розрахунку
    }
    hfdcan1.Init.NominalPrescaler     = prescaler;
    hfdcan1.Init.NominalSyncJumpWidth = sjw;
    hfdcan1.Init.NominalTimeSeg1      = tseg1;
    hfdcan1.Init.NominalTimeSeg2      = tseg2;
    // Для 500 кбіт/с при 80 МГц типовий результат:
prescaler=8, tseg1=7, tseg2=2, sjw=1
    // ---------- Розрахунок для швидкості даних (CAN FD) -
5 Мбіт/с ----------
    if (!calculate_bit_timing(5000000, 80000000, &prescaler,
&tseg1, &tseg2, &sjw)) {
        Error_Handler();    }
    hfdcan1.Init.DataPrescaler        = prescaler;
    hfdcan1.Init.DataSyncJumpWidth    = sjw;
    hfdcan1.Init.DataTimeSeg1         = tseg1;
    hfdcan1.Init.DataTimeSeg2         = tseg2;
    // Для 5 Мбіт/с при 80 МГц: prescaler=1, tseg1=13,
tseg2=2, sjw=1 → 16 Tq на біт [8†L19-L22]
    // ---------- Загальні параметри ----------
  hfdcan1.Init.FrameFormat=FDCAN_FRAME_FD_BRS;//CAN FD з BRS
    hfdcan1.Init.Mode                 = FDCAN_MODE_NORMAL;
    hfdcan1.Init.AutoRetransmission   = DISABLE;
    hfdcan1.Init.TransmitPause        = DISABLE;
    hfdcan1.Init.ProtocolException    = DISABLE;
    // ---------- Налаштування Message RAM ----------
    hfdcan1.Init.StdFiltersNbr        = 1;
    hfdcan1.Init.RxFifo0ElmtsNbr      = 2;
    hfdcan1.Init.RxFifo0ElmtSize      = FDCAN_DATA_BYTES_64;
    hfdcan1.Init.TxBuffersNbr         = 1;
    hfdcan1.Init.TxFifoQueueElmtsNbr  = 2;
    hfdcan1.Init.TxElmtSize           = FDCAN_DATA_BYTES_64;
    // Ініціалізація FDCAN
    if (HAL_FDCAN_Init(&hfdcan1) != HAL_OK) {
        Error_Handler();    }   }
```

CAN FD є не просто еволюційним розширенням, а якісним стрибком, що забезпечує безперервність використання перевірених механізмів

арбітражу та виявлення помилок у поєднанні з різко збільшеною пропускною здатністю. Це дозволило CAN залишитися основним стандартом для автомобільних і промислових мереж навіть в епоху гігабітного Ethernet, закривши нішу детермінованих і безпечних комунікацій із помірними швидкостями. Наступний підрозділ, присвячений CAN XL, окреслить подальший розвиток протоколу в напрямку гігабітних швидкостей і використання Ethernet-подібних методів кодування.

4.2.3. Впровадження CAN XL для гігабітних швидкостей у 2024–2026 рр. за специфікацією CiA 610-1

Подальшим еволюційним кроком після CAN FD, що відкриває шлях до гігабітних швидкостей передавання даних у вбудованих мережах реального часу, став протокол CAN XL (eXtended Length), специфікований міжнародною асоціацією CAN in Automation (CiA) у документі CiA 610-1 (CAN XL data link layer and physical coding sublayer, версія 1.0, 2022, з подальшими уточненнями у 2024 р.). Поштовхом до його розроблення стали насамперед потреби автомобільної галузі у високошвидкісному обміні великими обсягами даних для програмного оновлення електронних блоків керування «по повітрю» (SOTA/FOTA), передавання картографічної інформації для систем автономного водіння, потокового відео з камер кругового огляду, а також тунелювання кадрів Ethernet (наприклад, SOME/IP або DoIP) через існуючу CAN-інфраструктуру. Якщо CAN FD забезпечив стрибок до 8 Мбіт/с і 64 байтів даних, то CAN XL робить наступний квантовий перехід: швидкість фази даних зростає до 10–20 Мбіт/с, а максимальний розмір поля корисних даних розширюється до 2048 байтів (2 КБ). Такі параметри дозволяють CAN XL конкурувати зі 100BASE-T1 Ethernet у сегменті детермінованих комунікацій, зберігаючи при цьому ключові переваги екосистеми CAN — неразрушуючий арбітраж за пріоритетом повідомлень, жорстку детермінованість, надзвичайно низьку вартість впровадження та сумісність із попередніми поколіннями на рівні топології та кабельної інфраструктури. Як зазначається у дослідженні, опублікованому в журналі IEEE Access (том 10, 2022), саме гібридні мережі, що поєднують класичний CAN/CAN FD для критичних до часу керуючих сигналів і CAN XL для великих масивів даних, розглядаються як оптимальна архітектура для централізованих обчислювальних платформ транспортних засобів наступного покоління.

Ключовою інновацією CAN XL на фізичному рівні є перехід від традиційної несиметричної диференційної передачі з відкритим стоком до симетричного двотактного (push-pull) режиму роботи трансивера під час фази даних. Це докорінно відрізняє XL від класичного CAN та FD, де і арбітраж, і дані використовують спільну схему «домінанта/рецесив». Для забезпечення сумісності CAN XL вводить дві фази: фаза арбітражу, яка повністю відповідає CAN FD (домінантні/рецесивні рівні, номінальна швидкість до 1 Мбіт/с), і фаза даних, де трансивер перемикається у високошвидкісний push-pull режим із рівнями напруги ±1,2 В диференційно (CAN SIC XL transceiver). Специфікація CiA 610-1 визначає також фізичне кодування (PCS — Physical Coding Sublayer), яке використовує модуляцію PWM (Pulse Width Modulation) для даних на швидкостях 10 Мбіт/с і вище, що дозволяє ефективно відновлювати тактову частоту на приймальному боці без потреби у високоточних кварцових генераторах, притаманних Ethernet. Такий підхід здешевлює реалізацію, оскільки допускає більші похибки тактування.

Формат кадру CAN XL, схематично зображений на рисунку 4.7, значно розширений порівняно з CAN FD. Після стандартного поля арбітражу (11- або 29-бітний ідентифікатор) та бітів FDF (рецесивний) і XLF (eXtended Length Format), що ідентифікують XL-кадр, вводиться нове поле Priority ID (до 32 бітів), яке може використовуватися прикладним рівнем для додаткової внутрішньої пріоритезації або адресації в межах одного ідентифікатора. Далі слідує поле SDT (Service Data Unit Type), що вказує на тип вмісту (наприклад, класичний CAN-трафік, тунельований Ethernet-фрейм, діагностичні дані), поле DLC (розширене до 11 бітів, що й дає змогу адресувати до 2048 байтів), саме поле даних (0–2048 байтів), поле CRC (покращений поліном для виявлення помилок у великих пакетах) та завершальна частина з підтвердженням ACK на номінальній швидкості. Важливо, що біт BRS (Bit Rate Switch) успадкований від CAN FD і вказує на момент перемикання на високу швидкість даних. Таким чином, CAN XL зберігає сумісність із CAN FD на рівні арбітражу, що дозволяє FD- та XL-вузлам співіснувати в одній мережі, хоча для повноцінного використання XL-можливостей необхідні XL-трансивери та контролери з обох боків.



Рисунок 4.7 – Структура кадру CAN XL (стандартний ідентифікатор) згідно з CiA 610-1

У таблиці 4.7 наведено порівняння ключових параметрів трьох поколінь протоколу CAN. З неї видно, що CAN XL не лише збільшує швидкість та обсяг даних, а й додає гнучкість через введення SDT, що робить протокол придатним для мультисервісних мереж, де в одному фізичному сегменті співіснують критичні до часу команди керування, діагностичні сесії та потоки мультимедіа.

Таблиця 4.7 – Порівняння поколінь протоколу CAN

Параметр	***CAN 2.0***	***CAN FD***	***CAN XL***
Нормативний документ	ISO 11898-1:2003	ISO 11898-1:2015	CiA 610-1 (передано до ISO 11898-1)
Максимальна швидкість даних	1 Мбіт/с	до 8 Мбіт/с	до 20 Мбіт/с
Максимальна довжина даних	8 байт	64 байти	2048 байт
Тип трансивера у фазі даних	відкритий стік	відкритий стік	push-pull (SIC XL)
Кодування у фазі даних	NRZ	NRZ (з overclocking)	PWM (імпульсно-широтна модуляція)
Поле ідентифікації	11/29 біт	11/29 біт	11/29 біт + Priority ID
Поле DLC	4 біти (0–8)	4 біти (розширене)	11 біт (0–2048)
Контрольна сума	CRC-15	CRC-17/21	CRC-32 (два поля CRC)
Зворотна сумісність	так	з CAN 2.0 (обмежено)	з CAN FD (спільна шина)
Основне призначення	керування, датчики	FOTA, діагностика	тунелювання Ethernet, відео, радари

Період 2024–2026 років став визначальним для практичного впровадження CAN XL. Провідні виробники напівпровідників анонсували та випустили мікроконтролери з інтегрованими контролерами CAN XL. Зокрема, компанія STMicroelectronics у сімействі Stellar (для автомобільних шлюзів і контролерів доменів) реалізувала модулі CAN XL, що відповідають CiA 610-1, а також заявила про підтримку XL у нових лінійках STM32H5 та STM32MP2 (прес-реліз ST, 2023). Компанія Infineon Technologies інтегрувала CAN XL у мікроконтролери AURIX TC4x (згідно з документацією AURIX TC4xx User Manual, 2024), позиціонуючи їх як основу для центральних обчислювальних платформ транспортних засобів. NXP Semiconductors представила трансивер TJA1463, сумісний із CAN SIC XL, який підтримує як класичний CAN/CAN FD, так і XL-режим із автоматичним визначенням формату кадру. Texas Instruments у 2025 році анонсував трансивер TCAN1044A-Q1 з аналогічною функціональністю. Таким чином, екосистема компонентів для CAN XL швидко розширюється, що дає підстави прогнозувати його широке застосування в автомобілях модельного ряду 2027+ та в промислових системах, що потребують конвергенції детермінованого керування й високошвидкісної передавання даних.

У лістингу 4.5 наведено фрагмент конфігурації CAN XL-контролера на гіпотетичному мікроконтролері сімейства STM32H5 за допомогою HAL (з урахуванням, що на момент написання посібника цей API перебуває в стадії активної розробки, але принципи залишаються незмінними). Приклад демонструє, як задати номінальну швидкість арбітражу 500 кбіт/с та швидкість даних 10 Мбіт/с, а також як виконати передавання кадру з 2048 байтами корисних даних.

Лістинг 4.5 – Ініціалізація CAN XL на STM32H5 (HAL, прогнозований API)

```
// Відповідно до специфікацій CAN XL(CiA 610) та рекомендацій STM32H5
#include "fdcan.h"
// -------- Глобальний дескриптор FDCAN -------
FDCAN_HandleTypeDef hfdcan1;
// ======================================================
// 1. НАЛАШТУВАННЯ НОМІНАЛЬНОЇ ФАЗИ (АРБІТРАЖ) ТА ФАЗИ ДАНИХ (CAN XL)
// ======================================================
/**
 * @brief Ініціалізація FDCAN1 у режимі CAN XL
 * @note  Номінальна швидкість (арбітраж): 500 кбіт/с (як у CAN FD)
 *        Швидкість даних (CAN XL): 10 Мбіт/с
 * @note  Для роботи CAN XL необхідний трансивер із підтримкою CAN SIC,
```

```
 *         наприклад TJA1463 або аналогічний
 * @note  API є прогнозованим і може змінитися з виходом
фінальної версії CubeMX */
void MX_FDCAN1_Init(void)
{
    hfdcan1.Instance = FDCAN1;
    // - Номінальна фаза (арбітраж) - 500 кбіт/с -
    hfdcan1.Init.NominalPrescaler = 8; // Дільник для 80 МГц
    hfdcan1.Init.NominalSyncJumpWidth = 1;
    hfdcan1.Init.NominalTimeSeg1    = 7;
    hfdcan1.Init.NominalTimeSeg2    = 2;
 // - Фаза даних (CAN XL) - 10 Мбіт/с -
 // Використання структури XL DataPhaseConfig для параметрів
фази даних
    XL_DataPhaseConfig.DataPrescaler = 1; // Дільник: 80 МГц
/ 1 = 80 МГц
    XL_DataPhaseConfig.DataSyncJumpWidth  = 1;
    XL_DataPhaseConfig.DataTimeSeg1       = 6;   // 6 Tq
    XL_DataPhaseConfig.DataTimeSeg2       = 1;   // 1 Tq
    // Час біта=(6+1+3)*(1/80 МГц)=10*12.5нс=125нс → 8 Мбіт/с
    // (для 10 Мбіт/с необхідно підібрати інші параметри)
    // - Загальні параметри -
    hfdcan1.Init.FrameFormat = FDCAN_FRAME_XL;//Режим CAN XL
    hfdcan1.Init.Mode   = FDCAN_MODE_XL;     // Режим XL
    hfdcan1.Init.AutoRetransmission   = DISABLE;
    hfdcan1.Init.TransmitPause        = DISABLE;
    hfdcan1.Init.ProtocolException    = DISABLE;
    // - Налаштування Message RAM (для XL - до 2048 байт) -
    hfdcan1.Init.StdFiltersNbr        = 1;
    hfdcan1.Init.RxFifo0ElmtsNbr      = 2;
    hfdcan1.Init.RxFifo0ElmtSize = FDCAN_DATA_BYTES_2048; //
2048 байт для XL
    hfdcan1.Init.TxBuffersNbr         = 1;
    hfdcan1.Init.TxFifoQueueElmtsNbr  = 2;
    hfdcan1.Init.TxElmtSize           = FDCAN_DATA_BYTES_2048;
    // Ініціалізація FDCAN
    if (HAL_FDCAN_Init(&hfdcan1) != HAL_OK) {
        Error_Handler();
    }
}
// ====================================
// 2. НАЛАШТУВАННЯ ФІЛЬТРІВ
// ====================================
/**
 * @brief Налаштування фільтра для прийому повідомлень CAN
XL   */
void MX_FDCAN1_Filter_Config(void)
{
    FDCAN_FilterTypeDef sFilterConfig;
    sFilterConfig.IdType = FDCAN_EXTENDED_ID; // 29-бітний
ID для XL
    sFilterConfig.FilterIndex = 0;
    sFilterConfig.FilterType = FDCAN_FILTER_MASK;
    sFilterConfig.FilterConfig = FDCAN_FILTER_TO_RXFIFO0;
    sFilterConfig.FilterID1 = 0x1A2B3C4D; // Приклад ID
    sFilterConfig.FilterID2 = 0x1FFFFFFF; // Маска
    sFilterConfig.RxBufferIndex  = 0;
```

```
    if (HAL_FDCAN_ConfigFilter(&hfdcan1, &sFilterConfig) !=
HAL_OK) {
        Error_Handler();    }   }
// ======================================================
// 3. ВІДПРАВЛЕННЯ ПОВІДОМЛЕННЯ З 2048 БАЙТАМИ ДАНИХ
// ======================================================
/**
 * @brief Відправлення CAN XL повідомлення з 2048 байтами
даних
 * @param id          Ідентифікатор повідомлення
 * @param pData       Вказівник на масив даних (до 2048 байт)
 * @param len         Довжина даних (максимум 2048)
 * @retval HAL_OK     Успішно
 * @retval HAL_ERROR  Помилка   */
HAL_StatusTypeDef FDCAN_Send_XL_2048Bytes(uint32_t id,
uint8_t *pData, uint16_t len)
{
    FDCAN_TxBufferTypeDef TxBuffer;
    TxBuffer.Identifier  = id;
    TxBuffer.IdType = FDCAN_EXTENDED_ID; // 29-бітний ID
    TxBuffer.TxFrameType = FDCAN_DATA_FRAME;
    TxBuffer.DataLength  = len;  // до 2048 байт
    TxBuffer.ErrorStateIndicator = DISABLE;
    TxBuffer.BitRateSwitch  = ENABLE;   // BRS = 1
    TxBuffer.FDFormat = ENABLE; // CAN XL формат
    TxBuffer.TxEventFifoControl = DISABLE;
    TxBuffer.MessageMarker = 0;
    // Копіювання до 2048 байт даних
    for (uint16_t i = 0; i < len && i < 2048; i++) {
        TxBuffer.Data[i] = pData[i];
    }
    // Відправлення через Tx FIFO/Queue
   return HAL_FDCAN_AddMessageToTxFifoQ(&hfdcan1, &TxBuffer,
0);      }
```

CAN XL є не просто черговою ітерацією, а стратегічним мостом між традиційними CAN-мережами та світом високошвидкісних IP-протоколів. Він усуває останні бар'єри, пов'язані з обмеженою пропускною здатністю, дозволяючи використовувати перевірену часом парадигму пріоритетного арбітражу для передавання великих даних, що раніше потребувало б окремої Ethernet-лінії. Для інженера це означає можливість будувати однорідні, легкі в обслуговуванні мережі транспортного засобу або промислової установки, де один і той самий протокол обслуговує і екстрене гальмування (короткий кадр із найвищим пріоритетом), і завантаження оновленої прошивки в усі блоки за кілька секунд. Саме тому вивчення CAN XL у курсі програмування мікроконтролерів є інвестицією в розуміння майбутнього вбудованих комунікацій, яке вже починається сьогодні.

4.3. LIN (Local Interconnect Network): дешева субшина для автомобільних датчиків та актуаторів

Локальна мережа взаємоз'єднань LIN (Local Interconnect Network) була розроблена консорціумом європейських автовиробників та виробників напівпровідників на початку 2000-х років з єдиною метою — створити максимально дешеву послідовну шину для заміни дискретної проводки до простих датчиків та виконавчих механізмів, таких як датчики дощу, освітлення салону, приводи дзеркал, склопідйомники, блоки клімат-контролю. Якщо CAN є високошвидкісною магістраллю, що поєднує критично важливі електронні блоки, то LIN виконує роль допоміжної субшини, яка агрегує численні низькошвидкісні пристрої в локальному сегменті, під'єднаному до основної CAN-мережі через шлюз (майстер LIN). Специфікація LIN була стандартизована Міжнародною організацією зі стандартизації як серія ISO 17987 (частини 1–7, 2016 рік), що охоплює фізичний рівень, протокол, API та опис конфігураційної мови. Додатково широко використовується специфікація LIN Specification Package 2.2A (LIN Consortium, 2010), яка залишається актуальною для більшості впроваджень. Фізичний рівень LIN базується на одному дроті (single-wire) з напругою живлення від автомобільної бортової мережі 12 В, доповненому спільною землею, що зменшує вагу та вартість кабельної мережі порівняно з диференційними шинами. Максимальна швидкість передавання обмежена 20 кбіт/с, чого цілком достатньо для більшості невибагливих до часу застосувань, а низька швидкість дозволяє відмовитися від кварцових резонаторів на ведених вузлах і використовувати дешеві RC-генератори з автокалібруванням за синхропосилкою майстра.

В основі протоколу LIN лежить архітектура «майстер — ведений» із чітким розподілом ролей. Майстер LIN зазвичай реалізується на потужнішому мікроконтролері, який також виконує роль CAN-шлюзу, і містить точний кварцовий генератор. Усі обміни на шині ініціюються виключно майстром: він розсилає заголовки (headers) кадрів за суворим розкладом (schedule table), а ведені вузли у відповідь на відомий їм ідентифікатор або передають дані, або приймають їх, або ігнорують повідомлення. Така централізована дисципліна гарантує детермінованість і відсутність колізій без потреби в складному арбітражі, що є ключовою перевагою для дешевих 8-бітних мікроконтролерів із мінімальною периферією. Кількість ведених вузлів в одному сегменті LIN не перевищує 16, а загальна довжина шини — 40 метрів, що відповідає масштабу окремої зони автомобіля (двері, дах, передня панель). Типову топологію LIN-сегмента, під'єднаного до CAN-

магістралі через шлюз, наведено на рисунку 4.8, де показано розподіл функцій між магістраллю та субшиною.

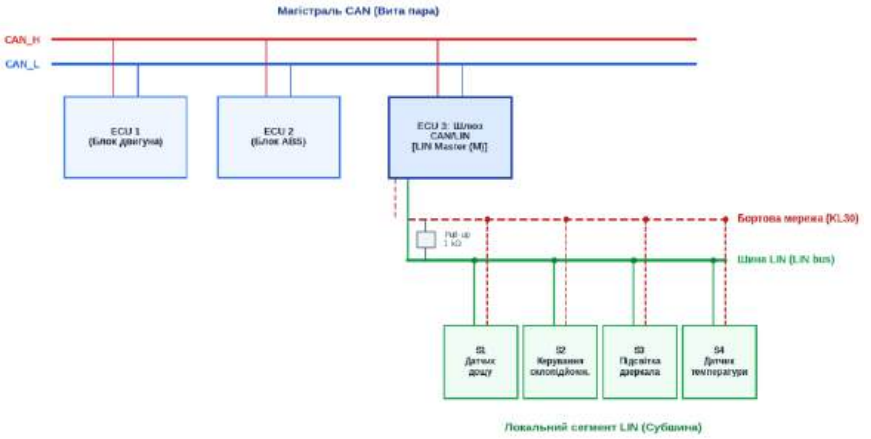


Рисунок 4.8 – Топологія LIN-субшини як доповнення до CAN-магістралі

Формат кадру LIN є надзвичайно простим і нагадує звичайний UART з додаванням спеціальної послідовності синхронізації. Кожен кадр складається з двох частин: заголовка (header), що передається майстром, та відповіді (response), що передається або майстром, або одним із ведених пристроїв. Заголовок починається з Break-поля — це навмисне порушення формату UART, коли лінія утримується в низькому стані протягом щонайменше 13 бітових інтервалів (на швидкості 19,2 кбіт/с це близько 680 мкс), після чого слідує роздільник (delimiter). Break-поле гарантує, що всі ведені вузли, незалежно від стану їхніх RC-генераторів, розпізнають початок нового кадру. Після Break майстер надсилає Sync-байт (0x55, що дає характерну послідовність бітів 01010101), за яким ведені вузли вимірюють реальну швидкість передавання і калібрують свої внутрішні генератори. Далі передається байт PID (Protected Identifier), який містить 6-бітний ідентифікатор повідомлення та два біти парності для захисту. Ідентифікатори 0x00–0x3B призначені для звичайних даних, 0x3C–0x3D — для діагностики, а 0x3E–0x3F — зарезервовані. Відповідь містить від 1 до 8 байтів даних (визначається конфігурацією для даного ідентифікатора) та завершальний байт контрольної суми (Checksum), яка обчислюється або лише над полем даних (Classic Checksum, для старих версій), або над PID і даними разом (Enhanced Checksum, рекомендований для LIN 2.x). Структуру кадру із позначенням полів наведено на рисунку 4.9.



Рисунок 4.9 – Структура кадру LIN 2.2A

Апаратна реалізація LIN-вузла вимагає лише стандартного модуля UART, здатного генерувати Break-поле тривалістю 13 бітів, та зовнішнього трансивера (наприклад, TJA1021 від NXP або MCP2003 від Microchip), який перетворює логічні рівні TX/RX у однопровідний сигнал із рівнями, сумісними з бортовою мережею 12 В. Більшість сучасних мікроконтролерів мають вбудовану підтримку LIN-режиму в модулях UART: наприклад, у STM32 (серії F3, F4, G0, G4) біт LINMEN у регістрі USART_CR2 активує автоматичне виявлення Break-поля приймачем і дозволяє формувати 13-бітовий Break передавачем через встановлення біта SBKRQ в регістрі USART_RQR. Згідно з посібником з програмування RM0360 (STM32F3) та прикладною нотаткою AN4032 «LIN communication using STM32 USART» (STMicroelectronics, 2019), процедура передавання заголовка кадру включає: увімкнення режиму LIN (LINMEN=1), встановлення необхідної швидкості (до 20 кбіт/с), генерування Break через SBKRQ, після прапорця TXE надсилання Sync-байта 0x55, а потім — байта PID з обчисленими бітами парності. Приймальний бік автоматично фіксує переривання за подією виявлення Break (прапорець LBDF). Критичним для роботи без зовнішнього кварцу є автокалібрування: приймач вимірює тривалість Sync-байта і обчислює поправку для свого генератора, що дозволяє використовувати внутрішній HSI або LSI генератор, здешевлюючи систему. У лістингу 4.6 наведено фрагмент коду для STM32F303 (HAL), який реалізує функцію майстра LIN для відправлення заголовка кадру з використанням апаратного режиму LIN.

Лістинг 4.6 – Відправлення заголовка кадру LIN у режимі майстра на STM32F303 (HAL, UART у режимі LIN)

```
// Відповідно до ISO 17987-2 та STM32F303 (RM0316)
#include "stm32f3xx_hal.h"
UART_HandleTypeDef huart2;
// ====================================
// 1. ІНІЦІАЛІЗАЦІЯ UART У РЕЖИМІ LIN
// ====================================
void MX_USART2_UART_Init(void)
{    huart2.Instance = USART2;
    huart2.Init.BaudRate = 19200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B; // LIN вимагає
8 біт даних
    huart2.Init.StopBits = UART_STOPBITS_1; // 1 стоп-біт
    huart2.Init.Parity = UART_PARITY_NONE; // Без parity
(окремі біти PID)
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    huart2.Init.LINMode = UART_LIN_ENABLE; // Активація LIN
  if (HAL_UART_Init(&huart2) != HAL_OK) { Error_Handler();  } }
// ====================================================
// 2. ФУНКЦІЯ ВІДПРАВЛЕННЯ ЗАГОЛОВКА (BREAK + SYNC + PID)
// ====================================================
/** * @brief Відправлення заголовка LIN кадру (майстер)
* @param huart  Дескриптор UART у режимі LIN
* @param pid    Захищений ідентифікатор (6 біт ID + 2 біти
парності) * @retval HAL_OK / HAL_ERROR
* @note Після Break (13 біт) передається Sync = 0x55, потім PID
 * @note  Парність PID згідно з ISO 17987-2:
 *        P0 = ID0 XOR ID1 XOR ID2 XOR ID4
 *        P1 = NOT (ID1 XOR ID3 XOR ID4 XOR ID5) */
HAL_StatusTypeDef LIN_SendHeader(UART_HandleTypeDef *huart,
uint8_t pid)
{    // 1. Генерація Break (13 біт '0') через апаратний
механізм LIN [5†L13]
  if (HAL_LIN_SendBreak(huart) != HAL_OK) { return HAL_ERROR; }
  // 2. Очікування готовності TX (після Break)
  while (!(huart->Instance->ISR & UART_FLAG_TXE));
  // 3. Відправлення Sync байта 0x55
  huart->Instance->TDR = 0x55;
  while (!(huart->Instance->ISR & UART_FLAG_TC));
  // 4. Відправлення Protected Identifier (PID)
  huart->Instance->TDR = pid;
  while (!(huart->Instance->ISR & UART_FLAG_TC));
  return HAL_OK;   }
// ============================================
// 3. ОБРОБНИК ПЕРИВАННЯ UART (ДЛЯ ПРИЙМАЧА)
// ============================================
void USART2_IRQHandler(void){    HAL_UART_IRQHandler(&huart2);
    // Перевірка прапорця виявлення Break (LBDF)
    if (huart2.Instance->ISR & UART_FLAG_LBDF) {
        __HAL_UART_CLEAR_FLAG(&huart2, UART_FLAG_LBDF);
        // Обробка виявленого Break (початок нового кадру)
    }   }
```

Порівняння LIN з CAN доцільно підсумувати в таблиці 4.8, яка наочно демонструє, чому LIN є не замінником, а доповненням CAN у сучасному автомобілі. Дані взято зі стандартів ISO 11898, ISO 17987 та специфікації LIN 2.2A.

LIN є яскравим прикладом спеціалізованого інтерфейсу, який, поступаючись за швидкістю та функціональністю, виграє за вартістю та простотою інтеграції.

Таблиця 4.8 – Порівняння характеристик LIN та CAN

Характеристика	**LIN (ISO 17987)**	**CAN 2.0 / CAN FD (ISO 11898)**
Середовище передавання	однопровідне + GND (single-wire)	диференційна вита пара (twisted pair)
Максимальна швидкість	20 кбіт/с	1 Мбіт/с (CAN FD до 8 Мбіт/с)
Кількість провідників	1 (+ живлення)	2 (+ живлення)
Максимальна кількість вузлів	до 16 (1 майстер + до 15 ведених)	до 32 (теоретично більше)
Арбітраж	майстер-ведений, за розкладом (schedule table)	побітовий неруйнівний за ідентифікатором
Типова вартість трансивера	дуже низька (TJA1021 ~$0.5)	середня (TJA1040 ~$0.8–1.2)
Стабільність Тактування ведених	RC-генератор із калібруванням (синхронізація за SYNCH byte)	кварцовий резонатор обов'язковий
Детермінованість	гарантована розкладом (frame slots)	гарантована пріоритетом (ID)
Основне застосування	датчики, освітлення, дверні модулі	керування двигуном, гальма, кермо

У навчальному контексті освоєння LIN дає студенту розуміння того, що інженерія — це мистецтво знаходження балансу між вимогами, і що для десятків простих функцій в автомобілі прокладати окремий CAN-кабель так само недоцільно, як використовувати гігабітний Ethernet для ввімкнення лампочки. Наступна частина посібника перейде до розгляду високошвидкісних універсальних інтерфейсів, які, навпаки, прагнуть досягти максимальної універсальності та швидкості, таких як USB та Ethernet.

РОЗДІЛ 5.
ВИСОКОШВИДКІСНІ УНІВЕРСАЛЬНІ ІНТЕРФЕЙСИ

5.1. USB у вбудованих системах

Універсальна послідовна шина (USB) за три десятиліття свого існування перетворилася з простого інтерфейсу для підключення клавіатур і мишок на домінуючий засіб зв'язку периферійних пристроїв із комп'ютерами, а також на універсальний порт живлення. Для вбудованих систем USB є незамінним інструментом: він забезпечує швидкий і надійний обмін даними з ПК, дає змогу реалізувати віртуальні COM-порти (CDC/ACM), емулювати клавіатури та миші (HID), підключати зовнішні накопичувачі (MSD) або навіть передавати відеопотоки. Сучасні мікроконтролери, як-от STM32H7 або ESP32-S3, оснащені контролерами USB OTG, що дозволяє пристрою виступати не лише як периферія (Device), але і як хост, під'єднуючи флешки, клавіатури чи інші USB-гаджети. Розвиток стандарту від USB 1.1 до USB4 версії 2.0 із пропускною здатністю 80 Гбіт/с, а також впровадження роз'єму Type-C з підтримкою Power Delivery (до 240 Вт) відкривають нові можливості для гетерогенних систем, де через один кабель передаються дані, відео та живлення. У цьому розділі розглянуто еволюцію USB, ключові класи пристроїв, режими OTG і хост-функціональність на прикладах популярних платформ, а також надано практичні рекомендації щодо інтеграції USB у проєкти IoT та промислової автоматики.

5.1.1. Еволюція від USB 1.1 до USB4 версії 2.0 (80 Гбіт/с) та USB Type-C Power Delivery 3.2

Універсальна послідовна шина USB (Universal Serial Bus), розроблена консорціумом USB Implementers Forum (USB-IF) у середині 1990-х років, пройшла шлях від скромного інтерфейсу для підключення клавіатур і мишок до уніфікованої високошвидкісної магістралі, здатної передавати відео, дані та потужність через єдиний кабель. Еволюція USB є яскравою ілюстрацією того, як стандарт, керований потребами ринку та технічним прогресом, може масштабуватися на кілька порядків за швидкістю, зберігаючи зворотну сумісність. У цьому підрозділі простежено ключові етапи цієї еволюції — від USB 1.1 до USB4 версії 2.0 — та розглянуто супутній

стандарт USB Type-C з підсистемою Power Delivery, яка перетворила USB з інтерфейсу передавання даних на універсальний інтерфейс живлення.

Перше масове впровадження USB розпочалося з версії USB 1.1 (1998 рік), яка визначила дві швидкості: Low Speed (1,5 Мбіт/с) для клавіатур і мишок та Full Speed (12 Мбіт/с) для принтерів, сканерів і аудіопристроїв. Фізичний рівень використовував несиметричну диференційну пару з напругою 3,3 В, а топологія була жорстко ієрархічною «зіркою» з хостом у центрі. Справжній прорив стався з виходом USB 2.0 у 2000 році, який додав режим High Speed (480 Мбіт/с), що зробило інтерфейс придатним для зовнішніх накопичувачів і відеокамер. Специфікація USB 2.0 (USB-IF, 2000, з пізнішими доповненнями) залишається базовою для більшості вбудованих мікроконтролерів і сьогодні.

Революційний стрибок відбувся з USB 3.0 (згодом перейменованим на USB 3.1 Gen 1, а потім на USB 3.2 Gen 1×1), який вперше представив додаткову диференційну пару для передавання даних на швидкості 5 Гбіт/с (SuperSpeed). На відміну від напівдуплексного USB 2.0, SuperSpeed використовує окремі лінії для передачі та прийому, реалізуючи повний дуплекс. USB 3.1 Gen 2 (10 Гбіт/с) запровадив ефективніше кодування 128b/132b. USB 3.2 (2017 рік) використав обидві диференційні пари роз'єму Type-C для одночасної двонаправленої передачі (x2 lanes), досягнувши 20 Гбіт/с. Усі ці версії детально описані в сімействі специфікацій USB 3.2 (USB-IF, 2017).

USB4 став тектонічним зсувом в архітектурі, оскільки він базується не на розвитку власного протоколу USB 3.2, а на специфікації Thunderbolt 3, переданій компанією Intel до USB-IF. Специфікація USB4 версії 1.0 (2019) визначила швидкості 20 Гбіт/с (Gen 2×2) та 40 Гбіт/с (Gen 3×2). USB4 версії 2.0, опублікована USB-IF у 2022 році, підняла швидкість до 80 Гбіт/с завдяки новому фізичному рівню PAM-3 та можливості асиметричного режиму (120 Гбіт/с в одному напрямку, 40 Гбіт/с у зворотному). Архітектура USB4 базується на тунелюванні протоколів: USB 3.2, DisplayPort і PCI Express (PCIe) пакуються в спільний транспортний рівень і маршрутизуються через єдиний кабель. Для вбудованих систем це означає можливість під'єднання високошвидкісної периферії, включно із зовнішніми GPU або NVMe-накопичувачами, через єдиний порт Type-C. Архітектуру протокольного стеку USB4 показано на рисунку 5.1.

Паралельно зі зростанням швидкості змінювався фізичний конектор. Стандарти USB Type-C (USB-IF, 2014) та супутній Power Delivery (PD) радикально розширили функціональність.

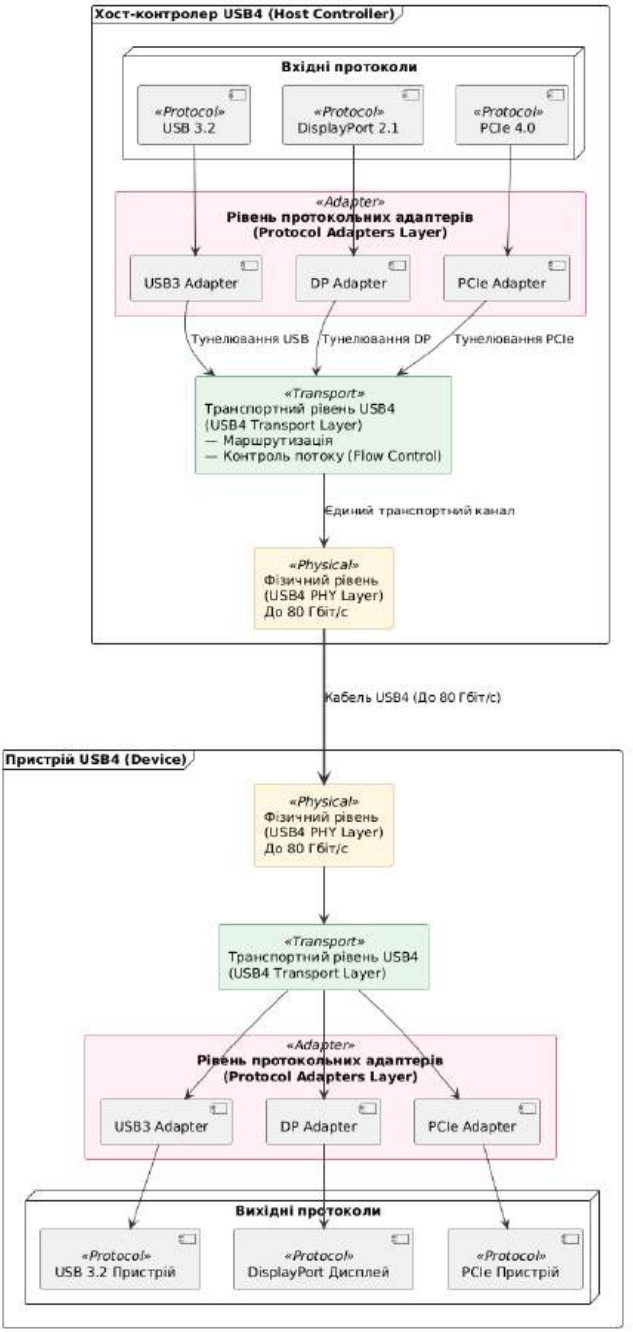


Рисунок 5.1 – Архітектура протокольного стеку USB4

Роз'єм Type-C має 24 контакти, симетричний дизайн (обидві сторони однакові) та спеціальний контакт CC (Configuration Channel) для узгодження ролей і потужності. Специфікація USB Power Delivery 3.0 (2019) запровадила програмоване джерело живлення (Programmable Power Supply, PPS), що дозволяє регулювати напругу з кроком 20 мВ. Версія PD 3.1 (2021) розширила діапазон потужності до 240 Вт (48 В / 5 А) завдяки режиму Extended Power Range (EPR). Станом на 2026 рік актуальною є версія PD 3.2, яка уточнює вимоги до кабелів і додає функції для автомобільних застосувань. У таблиці 5.1 — рівні потужності Power Delivery, а в таблиці 5.2 зведено ключові характеристики поколінь USB.

Таблиця 5.1 – Рівні потужності USB Power Delivery

Режим	Максимальна потужність	Напруга	Струм	Версія PD
Standard Power Range (SPR)	до 100 Вт	5 / 9 / 15 / 20 В	до 5 А	PD 2.0 / 3.0
Extended Power Range (EPR)	до 240 Вт	28 / 36 / 48 В	до 5 А	PD 3.1 / 3.2

Таблиця 5.2 – Еволюція поколінь USB

Версія	Рік	Максимальна швидкість	Кодування	Режим дуплексу	Типовий конектор
USB 1.1	1998	12 Мбіт/с (Full Speed)	NRZI	напівдуплекс	Type-A, Type-B
USB 2.0	2000	480 Мбіт/с (High Speed)	NRZI	напівдуплекс	Type-A, Type-B, Mini, Micro
USB 3.2 Gen 1×1	2017	5 Гбіт/с	8b/10b	повний	Type-A, Type-B, Type-C
USB 3.2 Gen 2×1	2017	10 Гбіт/с	128b/132b	повний	Type-A, Type-B, Type-C
USB 3.2 Gen 2×2	2017	20 Гбіт/с	128b/132b	повний	Type-C
USB4 Gen 3×2	2019	40 Гбіт/с	NRZ	повний	Type-C
USB4 v2.0	2022	80 Гбіт/с (симетр.) / 120/40 Гбіт/с (асиметр.)	PAM-3	повний	Type-C (обов'язково)

У вбудованих системах підтримка USB забезпечується апаратними контролерами, інтегрованими в мікроконтролери. Наприклад,

STM32F4/F7/H7 містять контролери USB 2.0 OTG (On-The-Go), що дозволяють пристрою виконувати роль як хоста, так і периферії. STM32H723 підтримує як USB 2.0 High Speed із зовнішнім PHY (через ULPI), так і вбудований PHY для Full Speed. ESP32-S3 підтримує USB 1.1 Full Speed (12 Мбіт/с) у режимі пристрою (Device), а через зовнішній USB-UART конвертор (CP210x) реалізує програмування. Raspberry Pi Pico (RP2040) має контролер USB 1.1, який може працювати як Host або Device, що активно використовується в проєктах на базі TinyUSB. Підтримка USB4 у світі мікроконтролерів поки відсутня через надзвичайну складність стеку, але доступна в системах на кристалі (SoC) класу i.MX 8/9, Rockchip RK3588 та інших, які використовують Linux.

Еволюція USB демонструє унікальну в історії техніки траєкторію: від простого інтерфейсу введення-виведення до універсального конвергентного порту, що передає дані, відео та потужність. Для розробника вбудованих систем це означає, що навіть у простих мікроконтролерних проєктах доступний надійний і швидкий канал зв'язку з ПК чи смартфоном, а з переходом на Linux-платформи відкриваються можливості підключення високошвидкісної периферії через один стандартизований кабель. Наступний підрозділ детальніше розкриє аспекти практичної реалізації USB у вбудованих системах, зокрема класи пристроїв (CDC, HID, MSD) та програмування OTG.

5.1.2. Класи пристроїв: CDC/ACM (віртуальний COM), HID, MSD, відео

Однією з найбільших переваг USB перед спеціалізованими послідовними інтерфейсами є те, що хостова операційна система не потребує встановлення специфічних драйверів для кожного нового пристрою, якщо цей пристрій відповідає одному зі стандартних класів, визначених USB-IF. Клас пристрою (Device Class) — це набір специфікацій, який уніфікує формат дескрипторів, типи кінцевих точок (endpoints), протокол обміну та формат даних для цілої категорії периферії. Завдяки цьому, наприклад, будь-яка USB-клавіатура працюватиме з будь-яким комп'ютером без інсталяції драйверів, оскільки всі вони використовують клас HID, вбудований драйвер для якого є в ядрі кожної операційної системи. Для розробника вбудованих систем використання стандартних класів — це шлях до швидкої інтеграції пристрою в екосистему ПК, планшетів і смартфонів без необхідності створювати та підтримувати власні драйвери. У цьому підрозділі розглянуто чотири найпоширеніші у вбудованих системах класи: CDC/ACM

(Communications Device Class / Abstract Control Model), HID (Human Interface Device), MSD (Mass Storage Device) та Video (USB Video Class), з акцентом на практичні аспекти їх реалізації на мікроконтролерах STM32, ESP32 та RP2040.

У лістингу 5.1 наведено приклад реалізації простої exo-програми на базі CDC/ACM з використанням TinyUSB на платформі RP2040.

Лістинг 5.1 – Exo-програма через CDC/ACM на Raspberry Pi Pico (TinyUSB)

```
1:// Джерело: TinyUSB (https://github.com/hathach/tinyusb)
2:#include "tusb.h"   // Підключення бібліотеки TinyUSB
3:// Ініціалізація USB-стеку (порт 0 для RP2040)
4:void main(void) {
5:    tud_init(BOARD_TUD_RHPORT); // BOARD_TUD_RHPORT = 0
6:    while (1) {
7:        tud_task();// Обробка USB-подій (обов'язково в циклі)
8: //Якщо є дані від хоста, читаємо та надсилаємо їх назад (exo)
9:        if (tud_cdc_available()) {
10:           uint8_t buf[64];
11:           uint32_t count = tud_cdc_read(buf, sizeof(buf));
12:           if (count > 0) {
13:               tud_cdc_write(buf, count);  // Exo-відповідь
14:               tud_cdc_write_flush(); // Примусове надсилання
15:           }       }    }   }
```

Клас CDC/ACM є найуживанішим у вбудованих системах завдяки своїй здатності емулювати віртуальний COM-порт (Virtual COM Port, VCP). Пристрій, що реалізує CDC/ACM, визначається специфікацією USB Class Definitions for Communication Devices (USB-IF, версія 2.0, 2010) та використовує, як правило, дві кінцеві точки типу Bulk (Bulk IN та Bulk OUT) для передавання даних і одну Interrupt IN для сповіщень. Коли такий пристрій під'єднується до ПК, операційна система автоматично завантажує стандартний драйвер usbser.sys (Windows) або cdc_acm.ko (Linux) і створює віртуальний COM-порт (наприклад, COM3 або /dev/ttyACM0). Для прикладної програми на ПК цей порт нічим не відрізняється від фізичного UART: його можна відкрити, читати та записувати байти. На стороні мікроконтролера це означає, що функції HAL_UART_Transmit() / HAL_UART_Receive() замінюються на відповідні функції USB-стека, які оперують не регістрами UART, а буферами кінцевих точок. У мікроконтролерах STM32 для реалізації CDC використовується стек USB Device Library (або сучасніший Azure RTOS USBX), де потрібно налаштувати дескриптори, створити буфери та реалізувати callback-функції для обробки прийнятих пакетів. Для ESP32-S3 компанія Espressif у фреймворку

ESP-IDF надає компонент usb_serial_jtag, який апаратно реалізує CDC через JTAG-інтерфейс, а також програмний стек TinyUSB, що підтримує CDC на звичайному USB-контролері. Для Raspberry Pi Pico SDK використовується бібліотека TinyUSB, яка надає простий API для створення CDC-пристрою буквально в кілька рядків коду.

Клас HID (Human Interface Device) призначений для пристроїв, якими керує людина: клавіатур, мишок, джойстиків, геймпадів, сенсорних панелей. Однак у вбудованому світі HID часто використовується не за прямим призначенням, а як зручний спосіб обміну невеликими пакетами даних без необхідності створення віртуального COM-порту та без потреби у спеціальних драйверах. Специфікація Device Class Definition for HID (USB-IF, версія 1.11) визначає, що пристрій HID використовує одну Interrupt IN кінцеву точку для передавання звітів (reports) і опціональну Interrupt OUT для отримання. Формат звітів описується спеціальним HID Report Descriptor, який є декларативним описом полів даних та їхнього призначення. Завдяки цьому дескриптору операційна система одразу розуміє, як інтерпретувати отримані байти (наприклад, перший байт — це біти модифікаторів клавіатури, другий — зарезервований, третій — код клавіші). Для вбудованого розробника, що створює нестандартний пристрій (наприклад, кастомний пульт керування або датчик), HID є найпростішим шляхом, оскільки його драйвер є вбудованим у всі ОС, він не потребує встановлення INF-файлів у Windows, і дозволяє передавати до 64 КБ/с через Interrupt-передачі. У лістингу 5.2 наведено фрагмент налаштування HID-дескриптора для кастомного пристрою з 8-байтовими звітами на STM32F4 з використанням бібліотеки ST USB Device Library.

Клас MSD (Mass Storage Device) дозволяє вбудованому пристрою представляти себе для хоста як зовнішній накопичувач (флешка, жорсткий диск, кардридер). Специфікація Mass Storage Class Overview (USB-IF, версія 1.4) та Transport Protocol (Bulk-Only Transport, BOT) описують, як хост надсилає команди SCSI (Small Computer System Interface) через кінцеві точки Bulk, а пристрій їх виконує, повертаючи дані або статус. У вбудованих системах MSD часто використовується для зручного доступу до файлів на SD-карті або зовнішній флеш-пам’яті без виймання носія. Наприклад, реєстратор даних може накопичувати файли CSV на SD-картці, а при підключенні до ПК через USB представляти цю картку як зовнішній диск, з якого можна просто скопіювати файли. Реалізація MSD потребує значного обсягу коду, оскільки потрібно емулювати файлову систему (FAT) або працювати з секторами. На STM32 для цього часто використовують стек USB Host/Device Library із компонентом MSC (Mass Storage Class) у поєднанні з

бібліотекою FatFS. На ESP32-S3 фреймворк ESP-IDF надає готовий приклад storage/sd_card через USB MSC, що дозволяє підключити SD-карту як USB-диск.

Лістинг 5.2 – Конфігурація HID Report Descriptor для 8-байтового пакета (STM32F4, ST USB Library)

```
// Відповідно до USB HID Specification v1.11 та STM32 USB Device Library
#include "usbd_customhid.h"
// ----- HID Report Descriptor (8 байт) ------
/** @brief Масив дескриптора звіту для Custom HID
 *  @note  Usage Page: Vendor Defined (0xFF00)
 *         Collection: Application
 *         Report Size: 8 біт (1 байт)
 *         Report Count: 8 (загалом 8 байт)
 *         Input: Data, Variable, Absolute */
  ALIGN_BEGIN static uint8_t CUSTOM_HID_ReportDesc[USBD_CUSTOM_HID_REPORT_DESC_SIZE] __ALIGN_END = {
    0x06, 0x00, 0xFF,//USAGE_PAGE(Vendor Defined)
    0x09, 0x01, // USAGE (Vendor Usage 1)
    0xA1, 0x01, // COLLECTION (Application)
    0x15, 0x00,         // LOGICAL_MINIMUM (0)
    0x26, 0xFF, 0x00,   // LOGICAL_MAXIMUM (255)
    0x75, 0x08,         // REPORT_SIZE (8 біт)
    0x95, 0x08,         // REPORT_COUNT (8 полів)
    0x81, 0x02,         // INPUT (Data, Var, Abs)
    0x91, 0x02,         // OUTPUT (Data, Var, Abs)
    0xC0                // END COLLECTION
};
// ---- Налаштування інтерфейсу USBD InterfaceTypeDef -----
/** @brief Структура інтерфейсу з вказівниками на функції
класу HID
 *  @note  Init – ініціалізація інтерфейсу
 *         DeInit – деініціалізація
 *         Receive – обробник отриманих даних  */
USBD_InterfaceTypeDef USBD_Interface_fops_FS = {
    .Init     = USBD_CUSTOM_HID_Init,
    .DeInit   = USBD_CUSTOM_HID_DeInit,
    .Receive  = USBD_CUSTOM_HID_Receive,
    // ... інші поля
};
```

Клас Video (USB Video Class, UVC) є найскладнішим із розглянутих, але надзвичайно актуальним із розвитком комп'ютерного зору на краю (Edge AI). Специфікація USB Device Class Definition for Video Devices (USB-IF, версія 1.5, 2012) описує, як пристрій може передавати відеопотоки (MJPEG, H.264, YUV тощо) через ізохронні кінцеві точки. Найбільша перевага UVC полягає в тому, що операційна система має вбудований драйвер (uvcvideo в Linux, стандартний драйвер у Windows 8+), тому будь-яка

програма, що працює з вебкамерами (Zoom, OpenCV, VLC), може безпосередньо отримувати відео з мікроконтролерної камери без написання драйверів. Для вбудованих систем це відкриває шлях до створення USB-камер на базі STM32, ESP32-S3 або Raspberry Pi Pico. Проте реалізація UVC є значно складнішою за CDC або HID, оскільки потребує коректного формування відеодескрипторів, підтримки ізохронного трафіку з гарантованою смугою пропускання та стиснення відео (або передавання сирого YUV, що потребує високої швидкості). На сьогодні існують відкриті реалізації UVC для STM32 на базі бібліотек TinyUSB, а також комерційні рішення. Порівняння характеристик розглянутих класів наведено в таблиці 5.3.

Таблиця 5.3 – Порівняння стандартних USB-класів для вбудованих систем

Клас	Характеристика	Значення
CDC/ACM (Virtual COM)	*Тип кінцевих точок*	Bulk IN/OUT, Interrupt IN
	Типова швидкість обміну	до 1 МБ/с (Full Speed)
	Необхідність драйверів	стандартний драйвер ОС (VCP)
	Складність реалізації	низька
	Типове застосування	консоль, програмування, телеметрія
HID (Custom)	*Тип кінцевих точок*	Interrupt IN/OUT (OUT – опціонально)
	Типова швидкість обміну	до 64 КБ/с (Interrupt)
	Необхідність драйверів	стандартний драйвер ОС (HID)
	Складність реалізації	низька
	Типове застосування	датчики, кнопки, ігрові контролери
MSD (Mass Storage)	*Тип кінцевих точок*	Bulk IN/OUT
	Типова швидкість обміну	до 1 МБ/с (Full Speed)
	Необхідність драйверів	стандартний драйвер ОС
	Складність реалізації	середня
	Типове застосування	реєстратори, зовнішні диски
Video (UVC)	*Тип кінцевих точок*	Isochronous IN (або Bulk IN)
	Типова швидкість обміну	до десятків МБ/с (High Speed+)
	Необхідність драйверів	стандартний драйвер ОС
	Складність реалізації	висока
	Типове застосування	камери, відеозахоплення

Варто підкреслити, що в реальних проектах мікроконтролер може одночасно підтримувати кілька класів у складі одного пристрою (Composite Device). Наприклад, поширеною є конфігурація, коли пристрій представляється як CDC/ACM (для налагоджувальної консолі) плюс MSD

(для доступу до файлів налаштувань). Або поєднання HID (для кнопок) та CDC (для передавання даних). Сучасні стеки, такі як TinyUSB, роблять створення композитних пристроїв відносно простим завданням, дозволяючи описати кілька інтерфейсів у конфігураційному дескрипторі та призначити їм різні кінцеві точки.

Використання стандартних USB-класів є потужним інструментом вбудованого розробника, що дозволяє реалізувати plug-and-play сумісність із будь-яким ПК або мобільним пристроєм без написання драйверів. Вибір конкретного класу визначається характером даних: для потокової телеметрії зручний CDC/ACM, для дискретного керування — HID, для файлового обміну — MSD, а для відеопотоків — UVC. Наступний підрозділ, присвячений USB On-The-Go та хост-режиму, покаже, як мікроконтролер сам може виступати в ролі хоста, керуючи підключеними до нього USB-пристроями.

5.1.3. USB On-The-Go (OTG) та хост-режим на мікроконтролерах (ESP32-S3, STM32H7)

Традиційна архітектура USB передбачає чітке розділення ролей: хост (Host) ініціює всі транзакції, керує живленням шини та координує підключення пристроїв, тоді як периферійний пристрій (Device) лише відповідає на запити хоста. У світі персональних комп'ютерів ця модель працює бездоганно: ПК завжди є хостом, а клавіатура чи флешка — пристроєм. Проте у вбудованих системах ситуація складніша: мікроконтролер, що керує промисловим датчиком, може потребувати зчитування даних із USB-флешки (виступаючи хостом), а за кілька хвилин бути підключеним до ПК для вивантаження накопичених логів (виступаючи пристроєм). Саме для таких сценаріїв було розроблено доповнення до специфікації USB 2.0 під назвою On-The-Go (OTG), яке дозволяє пристрою динамічно перемикатися між роллю хоста (Host) та периферії (Device) без перезавантаження. Специфікація USB OTG and Embedded Host Supplement (USB-IF, версія 2.0, 2011) визначає механізми узгодження ролей через контакт ID роз'єму Mini-AB або Micro-AB, протокол HNP (Host Negotiation Protocol) для програмного перемикання та SRP (Session Request Protocol) для пробудження сплячої шини.

У сучасних мікроконтролерах OTG-контролери інтегруються апаратно, значно спрощуючи реалізацію подвійної ролі. Сімейство STM32F2/F4/F7/H7, а також окремі представники STM32G4, містять мо-

дуль USB OTG (High Speed або Full Speed), який апаратно підтримує режими Host, Device та OTG. У документації RM0468 (для STM32H723/733, розділ 47) детально описано архітектуру цього модуля, який включає власний DMA-контролер, підтримку кінцевих точок (до 16 IN та 16 OUT у режимі хоста), апаратне виявлення підключення/відключення пристрою та генерацію SOF (Start of Frame). Для підключення зовнішнього High Speed PHY використовується інтерфейс ULPI (UTMI+ Low Pin Interface). На рисунку 5.2 показано структурну схему STM32H7 з модулем OTG та зовнішнім PHY для High Speed (480 Мбіт/с).

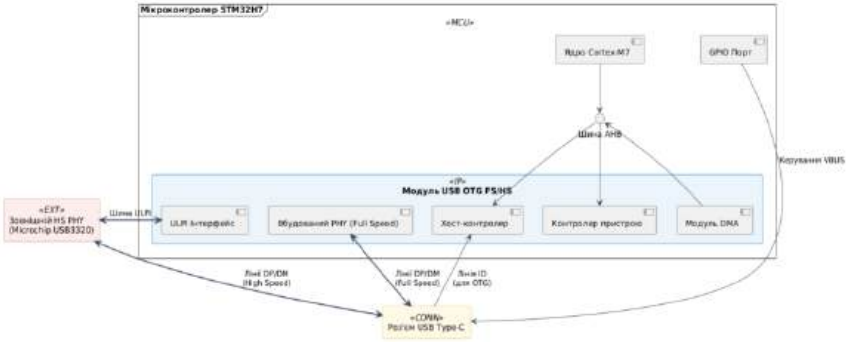


Рисунок 5.2 – Структурна схема USB OTG High Speed на STM32H7

ESP32-S3, згідно з технічним описом (ESP32-S3 Technical Reference Manual, розділ 33), має вбудований USB 1.1 Full Speed OTG-контролер із підтримкою як Host, так і Device режимів. Фреймворк ESP-IDF надає компонент USB Host Library, який використовує апаратний контролер для роботи з пристроями класів HID, MSC, CDC. У режимі хоста ESP32-S3 може, наприклад, зчитувати дані з USB-флешки (через MSC) або приймати введення з USB-клавіатури (через HID). Raspberry Pi Pico (RP2040) також має вбудований контролер USB 1.1, який у поєднанні з бібліотекою TinyUSB підтримує як Host, так і Device режими, хоча продуктивність у хост-режимі обмежена швидкістю Full Speed (12 Мбіт/с).

Практична реалізація OTG вимагає не лише програмного налаштування контролера, а й коректної апаратної підтримки. По-перше, роз'єм повинен бути типу Micro-AB або Type-C з підтримкою Dual Role Port (DRP). У Type-C роль визначається рівнями напруги на контактах CC (Configuration Channel) або програмним перемиканням. По-друге, мікроконтролер повинен мати можливість керувати подачею живлення VBUS (5 В) у режимі хоста, для чого зазвичай використовується зовнішній

MOSFET-ключ, керований GPIO, як-от мікросхема STMPS2151. Коли пристрій переходить у режим хоста, він активує VBUS, після чого під'єднаний периферійний пристрій отримує живлення і починає процес нумерації (enumeration). Якщо ж пристрій сам підключається до іншого хоста, він вимикає власний VBUS і живиться від шини. У лістингу 5.3 наведено фрагмент ініціалізації USB Host на STM32H723 із використанням Azure RTOS USBX (або еквівалентного HAL), що демонструє процедуру виявлення та монтування USB-флешки.

Лістинг 5.3 – Ініціалізація USB Host для роботи з MSC (флешкою) на STM32H723 (Azure RTOS USBX)

```
// Джерело: STM32 USBX examples, Azure RTOS Documentation
#include "ux api.h"
#include "ux host class storage.h"
#include "fx api.h"
// ---- Глобальні дескриптори -----
UX_HOST_CLASS_STORAGE storage_instance;  // Екземпляр класу MSC
FX_MEDIA      media;     // Медіа для FileX
FX_FILE       file;      // Файл для операцій
// ================================================
// 1. ІНІЦІАЛІЗАЦІЯ HCD (Host Controller Driver)
// ================================================
/** * @brief Ініціалізація USB Host через HAL та USBX
 * @note  Попередньо необхідно налаштувати DMA, виділити стек
для USBX та виконати конфігурацію Pinout для VBUS-ключа  */
void MX_USB_HOST_Init(void) {
// 1. Ініціалізація HCD (HAL, налаштовує тактування, піни, DMA
    HAL_HCD_Init(&hhcd_USB_OTG_FS);
    // 2. Реєстрація HCD у USBX
    ux_host_stack_hcd_register(_ux_system_host_hcd_stm32_name,
                        _ux_hcd_stm32_initialize, 0x200, 0x200);
    // 3. Активація класу MSC (Mass Storage Class)
    ux_host_class_storage_register();  }
// ================================================
// 2. СТВОРЕННЯ КЛАСІВ (MSC) ТА АКТИВАЦІЯ
// ================================================
/** * @brief Реєстрація класу MSC в USBX
* @note Клас реєструється через ux_host_class_storage_register()
Після підключ. При-ою виклик. UX_HOST_CLASS_STORAGE_ACTIVATE */
void USBX_Host_Class_Init(void) {
    // Реєстрація класу зберігання
    ux_host_class_storage_register();  }
// Callback при активації класу (викликається після підключення
флешки)
UINT storage_activate(UX_HOST_CLASS_STORAGE *storage) {
    // Ініціалізація медіа для FileX
    fx_media_open(&media, "usb_disk",
                  storage->ux_host_class_storage_media,
                  FX_NULL, FX_NULL);
    return UX_SUCCESS;  }
// ================================================
// 3. ГОЛОВНИЙ ЦИКЛ — ОБРОБКА ПОДІЙ USBX
```

```
// ================================================
void main(void) {
    MX_USB_HOST_Init();//Ініціалізація HCD та реєстрація класів
    while (1) {
    // Обробка всіх подій USBX (enumeration, передача даних)
        ux_host_stack_tasks_run();
        // Якщо флешка підключена та відмонтована — виконуємо операції з файлом
        if (media.fx_media_opened) {
            Read_File_From_USB();
        }    }   }
// ========================================
// 4. ЧИТАННЯ ФАЙЛУ З ФЛЕШКИ (FileX)
// ========================================
/** * @brief Демонстрація читання файлу з флешки через FileX
 * @note  Використовуються fx_media_open() та fx_file_read() */
void Read_File_From_USB(void) {
    UINT status;
    ULONG bytes_read;
    CHAR buffer[256];
    // 1. Відкриття медіа (файлової системи)
    status = fx_media_open(&media, "usb_disk",
                storage_instance.ux_host_class_storage_media,
                           FX_NULL, FX_NULL);
    if (status != FX_SUCCESS) return;
    // 2. Відкриття файлу для читання
    status = fx_file_open(&media, &file, "test.txt",
FX_OPEN_FOR_READ);
    if (status != FX_SUCCESS) return;
    // 3. Читання даних з файлу
    status = fx_file_read(&file, buffer, sizeof(buffer),
&bytes_read);
    if (status == FX_SUCCESS) {
        // Обробка прочитаних даних
    }
    // 4. Закриття файлу та медіа
    fx_file_close(&file);
    fx_media_close(&media);
}
```

Окремої уваги заслуговує концепція Embedded Host, яка є спрощеним варіантом хоста, призначеним для вбудованих систем із обмеженими ресурсами. На відміну від повноцінного хоста ПК, Embedded Host не зобов'язаний підтримувати всі можливі класи пристроїв; він реалізує лише ті, які потрібні для конкретного застосування (наприклад, лише MSC або лише HID). Це значно зменшує обсяг коду та споживання пам'яті. Саме в цьому режимі найчастіше працюють мікроконтролери з обмеженими ресурсами, такі як STM32F4 або ESP32-S3. Специфікація USB OTG дозволяє також використовувати режим SRP (Session Request Protocol), за яким периферійний пристрій може запросити у хоста ввімкнення VBUS, що важливо для збереження енергії в батарейних системах.

Порівняння можливостей OTG/хост-режиму для трьох цільових платформ посібника наведено в таблиці 5.4, яка узагальнює дані з офіційної документації виробників.

Таблиця 5.4 – Порівняння USB OTG/хост-можливостей платформ STM32H7, ESP32-S3 та RP2040

Характеристика	STM32H7	ESP32-S3	RP2040 (Raspberry Pi Pico)
Версія USB	USB 2.0 OTG (High-Speed + Full-Speed)	USB 1.1 OTG (Full-Speed)	USB 1.1 (Full-Speed)
Максимальна швидкість	High Speed (480 Мбіт/с) через зовнішній ULPI PHY	Full Speed (12 Мбіт/с)	Full Speed (12 Мбіт/с)
Режими	Host, Device, OTG	Host, Device (OTG)	Host, Device
Підтримка ULPI	так (для OTG_HS1)	ні	ні
Типовий програмний стек	Azure RTOS USBX, ST USB Library	ESP-IDF USB Host Library, TinyUSB	TinyUSB
Підтримувані хост-класи (типово)	MSC, HID, CDC	MSC, HID, CDC (ACM)	MSC, HID, CDC, MIDI

USB OTG та хост-режим на мікроконтролерах відкривають шлях до створення автономних вбудованих систем, здатних безпосередньо взаємодіяти зі стандартною USB-периферією — флешками, клавіатурами, WiFi-адаптерами, камерами — без посередництва ПК. Це є критично важливим для IoT-шлюзів, промислових реєстраторів, медичних приладів та автомобільних інформаційно-розважальних систем. Наступний підрозділ, присвячений Ethernet у вбудованих системах, продемонструє, як доповнити USB мережевим інтерфейсом, що забезпечує вихід у локальну мережу та Інтернет.

5.2. Ethernet для вбудованих систем

Упродовж останніх десятиліть Ethernet міцно асоціювався з офісними та корпоративними мережами, тоді як у світі мікроконтролерів і польових пристроїв панували спеціалізовані послідовні шини — RS-485, CAN, Modbus. Однак зниження вартості апаратних компонентів, поява компактних стеків TCP/IP і, головне, потреба в наскрізній IP-адресації від датчика до хмарного сервісу докорінно змінили цю ситуацію. Сьогодні

Ethernet дедалі впевненіше проникає на найнижчі рівні автоматизації, витісняючи застарілі шлюзи трансляції протоколів і спрощуючи архітектуру промислових IoT-систем. Особливого значення цей процес набуває зі стандартизацією однопарного Ethernet (Single Pair Ethernet, SPE) у вигляді 10BASE-T1S, що дозволяє будувати багатоточкові польові мережі з живленням по одній витій парі та детермінованим доступом до середовища. Водночас сучасні мікроконтролери, як-от STM32H7, оснащуються вбудованими MAC-контролерами з підтримкою DMA, а також інтерфейсами RMII/MII для підключення зовнішніх PHY, що разом із легковаговими стеками LwIP або FreeRTOS+TCP перетворює навіть недорогий пристрій на повноцінний TCP/IP-вузол. Нарешті, для застосувань із жорсткими вимогами до реального часу впровадження профілів Time-Sensitive Networking (TSN, IEEE 802.1Q) робить Ethernet придатним для заміни традиційних польових шин у замкнених контурах керування, забезпечуючи гарантовану латентність і синхронізацію часу в масштабах усієї мережі. Цей розділ послідовно розкриває ці три ключові напрямки: фізичний рівень SPE для польових мереж, апаратну та програмну реалізацію Ethernet-вузла на базі MAC+PHY та стеків TCP/IP, а також детерміновані механізми TSN, що відкривають шлях до єдиної конвергентної мережі промислового Інтернету речей.

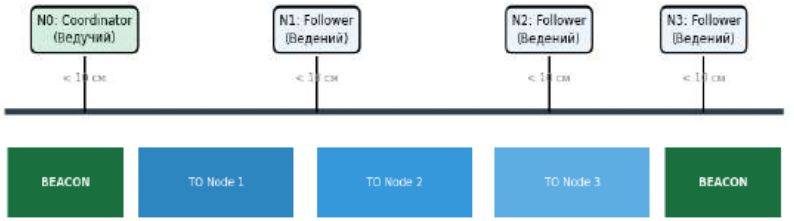
5.2.1. 10BASE-T1S (Single Pair Ethernet) як новий стандарт для поля IoT згідно IEEE 802.3cg

Протягом десятиліть Ethernet залишався магістральною технологією офісних і промислових мереж, однак його проникнення на найнижчий рівень автоматизації — до окремих датчиків, виконавчих механізмів і польових пристроїв — стримувалося двома фундаментальними обмеженнями класичних реалізацій 10BASE-T/100BASE-TX. По-перше, для з'єднання вимагалося щонайменше дві виті пари (або чотири для 1000BASE-T), що збільшувало масу, вартість і габарити кабельної мережі, роблячи її неприйнятною для сотень розподілених сенсорів. По-друге, топологія «зірка» з активними комутаторами була надлишково дорогою для багатоточкових польових шин, де історично домінували RS-485 і CAN. У відповідь на ці виклики робоча група IEEE 802.3cg у 2019 році затвердила поправку до стандарту IEEE 802.3, яка ввела два нових типи фізичного рівня для Single Pair Ethernet (SPE) — 10BASE-T1S (Short Reach, до 25 метрів) та 10BASE-T1L (Long Reach, до 1000 метрів). Саме 10BASE-T1S, розроблений для багатоточкових польових мереж із живленням по тій самій парі,

став каталізатором переходу промислового Інтернету речей на наскрізний IP-протокол — від датчика до хмари. Як зазначено в дослідженні, опублікованому в IEEE Transactions on Industrial Informatics (том 18, 2022), впровадження SPE-мереж дозволяє усунути шлюзи трансляції протоколів (наприклад, Modbus TCP/RTU) і спростити архітектуру системи, підвищивши її надійність і зменшивши латентність.

Фізичний рівень 10BASE-T1S забезпечує швидкість передавання 10 Мбіт/с через одну неекрановану виту пару (UTP) або екрановану (STP), використовуючи диференційну передачу з рівнями напруги, сумісними з автомобільними та промисловими вимогами. Завдяки цьому один і той самий двожильний кабель може одночасно слугувати для передавання даних і живлення відповідно до стандарту IEEE 802.3bu (Power over Data Line, PoDL), що є аналогом Power over Ethernet, адаптованим для однопарних ліній. PoDL забезпечує потужність до 50 Вт (клас 12) у режимі точка-точка, чого достатньо для живлення більшості польових пристроїв, включно з датчиками, клапанами та невеликими контролерами. На відміну від 10BASE-T1L, який оптимізований для з'єднань точка-точка на великі відстані, 10BASE-T1S підтримує як точку-точку, так і багатоточкову шинну топологію (multidrop) із підключенням до 8 вузлів на одному сегменті завдовжки до 25 метрів. Саме ця багатоточковість у поєднанні з механізмом PLCA (Physical Layer Collision Avoidance) робить 10BASE-T1S прямою заміною RS-485 і CAN у польових мережах.

PLCA є ключовою інновацією 10BASE-T1S, що вирішує проблему колізій, притаманну класичному напівдуплексному Ethernet із CSMA/CD. У мережі PLCA один із вузлів призначається координатором (PLCA Coordinator), який циклічно надає право передавання кожному вузлу, надсилаючи спеціальний сигнал BEACON. Вузол, що отримав дозвіл (Transmit Opportunity), може передати кадр, після чого координатор переходить до наступного вузла. Якщо вузол не має даних для передавання, він мовчить, і координатор після короткого таймауту переходить далі. Такий механізм гарантує детерміновану латентність без колізій, що є критичним для систем реального часу. Крім того, PLCA зберігає сумісність із існуючими MAC-рівнями Ethernet, оскільки реалізується на фізичному рівні (PCS — Physical Coding Sublayer) і не вимагає змін у MAC-контролері мікроконтролера. Завдяки цьому будь-який стандартний MAC 10/100 Мбіт/с може працювати з PHY 10BASE-T1S, що підтримує PLCA, без модифікації програмного стеку TCP/IP. Схематичне зображення багатоточкової топології 10BASE-T1S із PLCA-координатором наведено на рисунку 5.3.



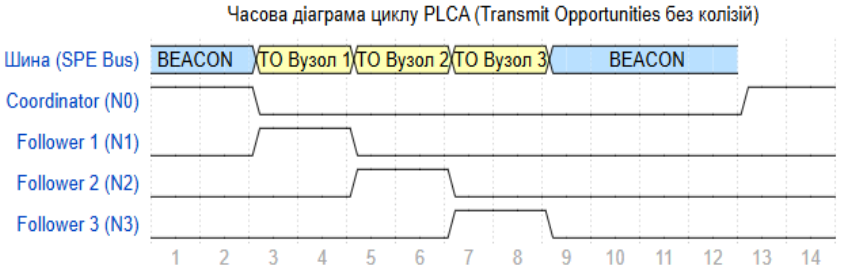


Рисунок 5.3 – Багатоточкова топологія 10BASE-T1S із PLCA

Для порівняння з іншими польовими інтерфейсами в таблиці 5.5 наведено ключові характеристики 10BASE-T1S, CAN FD та RS-485. З таблиці видно, що 10BASE-T1S поєднує переваги IP-мереж (наскрізна адресація, підтримка TCP/IP) із простотою двопровідної шини, яка раніше була доступна лише для спеціалізованих промислових протоколів.

Практичне впровадження 10BASE-T1S у вбудовані системи потребує мікроконтролера з вбудованим MAC-рівнем Ethernet (стандартним 10/100 Мбіт/с) та зовнішнім PHY-трансивером, сумісним із 10BASE-T1S. На ринку доступні PHY-мікросхеми від провідних виробників: LAN8670/LAN8671 від Microchip Technology, DP83TD510E від Texas Instruments, TJA1101S від NXP Semiconductors. Ці PHY підключаються до MAC через стандартний інтерфейс RMII або MII і підтримують PLCA на апаратному рівні.

Для мікроконтролерів STM32, що мають вбудований Ethernet MAC (серії F4, F7, H7, H5), підключення здійснюється через RMII, причому замість стандартного PHY 100BASE-TX (наприклад, LAN8720) встановлюється PHY 10BASE-T1S.

Таблиця 5.5 – Порівняння 10BASE-T1S, CAN FD та RS-485 для польових мереж IoT

Характеристика	10BASE-T1S (IEEE 802.3cg)	CAN FD (ISO 11898-1)	RS-485 (TIA/EIA-485)
Швидкість	10 Мбіт/с	до 8 Мбіт/с (фаза даних)	до 10 Мбіт/с (на 12 м)
Топологія	багатоточкова шина (multidrop) або точка-точка	багатоточкова шина (multidrop)	багатоточкова шина (multidrop)
Максимальна кількість вузлів	до 8 (на сегмент, згідно зі стандартом)	до 30–60 (залежить від трансиверів та швидкості)	до 32 (стандарт)
Максимальна довжина	25 м (Mixing Segment)	40 м (на 1 Мбіт/с) – 500 м (на 125 кбіт/с)	1200 м (на 100 кбіт/с)
Детерміно-ваність	гарантована PLCA (Physical Layer Collision Avoidance)	гарантована арбітражем (побітовий неруйнівний арбітраж за ідентифікатором)	залежить від протоколу верхнього рівня (наприклад, Modbus RTU — запит-відповідь)
Живлення по лінії	PoDL (IEEE 802.3bu) – для Link Segment; для Mixing Segment не специфіковано	ні (живлення окремим провідником)	ні (живлення окремим провідником)
Протокольний стек	TCP/IP, UDP, MQTT, HTTP (native Ethernet)	CANopen, J1939, власні протоколи	Modbus RTU, Profibus DP
Основний стандарт	IEEE 802.3cg-2019	ISO 11898-1:2015	TIA/EIA-485 (RS-485)

У лістингу 5.4 наведено фрагмент коду ініціалізації PHY LAN8670 через MDIO з використанням HAL на STM32H723, де ключовою відмінністю є запис у регістри PLCA-конфігурації (PLCA Control/Status) для активації режиму координатора або веденого.

Програмна підтримка 10BASE-T1S на рівні стеку TCP/IP не потребує модифікацій, оскільки MAC-рівень залишається стандартним Ethernet. Стек LwIP (Lightweight IP), що широко використовується у вбудованих системах, може без змін працювати поверх драйвера MAC STM32, який взаємодіє з PHY 10BASE-T1S.

Лістинг 5.4 – Ініціалізація PHY 10BASE-T1S LAN8670 через MDIO на STM32H723 (HAL)

```
//Відповідно до IEEE802.3cg(Clause 147) та Microchip LAN8670 DS
#include "stm32h7xx_hal.h"
//Адреси регістрів PHY(відпов. до даташиту LAN8670)
#define PHY_BCR          0x00   // Basic Control Register
#define PHY_BMSR         0x01   // Basic Status Register
#define PHY_PLCA_CTRL0   0xCA00 // PLCA Control 0
#define PHY_PLCA_CTRL1  0xCA02 // PLCA Control 1 (ID+Node Count)
// Біти керування
#define BCR_RESET  0x8000 // Скидання PHY
#define BCR_DUPLEX_MODE  0x0100 // Повний дуплекс (0 - Half для
multidrop)
#define BCR_AN_ENABLE  0x1000 // Автоузгодження (вимкнено для
10BASE-T1S)
#define PLCA_EN   0x8000 // Біт активації PLCA
#define PLCA_COORDINATOR 0x00 // ID=0 - координатор
#define PLCA_FOLLOWER    0x01 // ID=1 - ведений
extern ETH_HandleTypeDef heth;//Дескриптор Ethernet (STM32H723)
// -------------------------------------------------
// @brief  Ініціалізація PHY LAN8670 у режимі 10BASE-T1S із PLCA
// @note Швидкість фіксована, 10 Мбіт/с (без автоузгодження)
//   Дуплекс - Half (обов'язково для multidrop)
//   PLCA - увімкнено, координатор (ID=0) або ведений (ID=1)
// -------------------------------------------------
void ETH_PHY_Init(void) {
    uint32_t reg;
// 1. Скидання PHY (Basic Control Register, bit 15)
    HAL_ETH_WritePHYRegister(&heth, PHY_BCR, BCR_RESET);
    HAL_Delay(100);   // Час на вихід зі скидання
    // 2. Налаштування базового режиму (BCR):
    //  - швидкість 10 Мбіт/с (за замовчуванням)
    //  - Half Duplex (для multidrop)
    //  - вимкнути автоузгодження (10BASE-T1S не підтримує)
    reg = 0;   // BCR = 0 → 10 Мбіт/с, Half Duplex, AN вимкнено
    HAL_ETH_WritePHYRegister(&heth, PHY_BCR, reg);
    // 3. Активація PLCA (Physical Layer Collision Avoidance)
   //Запис у PLCA Control 0: встановити біт PLCA EN
    HAL_ETH_WritePHYRegister(&heth, PHY_PLCA_CTRL0, PLCA_EN);
   //4.Налаштування PLCA Control 1 (ID та Node Count)
    //  - ID=0 → координатор (генерує BEACON)
    //  - ID=1 → ведений (очікує свою чергу)
    //  - Node Count = 8 (максимум вузлів у мережі)
    uint32_t node_id = PLCA_COORDINATOR;   // або PLCA_FOLLOWER
    uint32_t node_count = 8;
    reg = (node_count << 8) | node_id; // NCNT[15:8] + ID[7:0]
    HAL_ETH_WritePHYRegister(&heth, PHY_PLCA_CTRL1, reg);
//5.(Опціонально) Перевірка статусу PHY (Basic Status Register)
    // HAL_ETH_ReadPHYRegister(&heth, PHY_BMSR, &reg);
}
```

Це означає, що всі протоколи вищого рівня — MQTT, HTTP, Modbus TCP, OPC UA — стають доступними безпосередньо на польовому пристрої без проміжних шлюзів. За даними дослідження, опублікованого в

Proceedings of the IEEE (том 110, 2024), саме конвергенція польових комунікацій на основі SPE та TSN дозволить створити повністю детерміновані IP-мережі для Industry 4.0 з наскрізною латентністю менше 1 мс.

10BASE-T1S є не просто черговою варіацією Ethernet, а стратегічним мостом між традиційними польовими шинами та світом Інтернету речей. Для розробника це означає можливість використовувати єдиний наскрізний протокол IP для всіх рівнів системи, відмовившись від трансляції через шлюзи та спростивши архітектуру. Наступний підрозділ, присвячений MAC+PHY контролерам та стекам TCP/IP, розгляне практичні аспекти інтеграції Ethernet у проекти на STM32 з використанням RMII/MII та FreeRTOS+LwIP.

5.2.2. MAC + PHY контролери: RMII/MII, вбудовані стеки TCP/IP (FreeRTOS+TCP, LwIP)

Реалізація Ethernet у вбудованій системі вимагає чіткого розуміння розподілу функцій між апаратними рівнями, оскільки, на відміну від простих послідовних інтерфейсів, підключення до мережі ніколи не обмежується лише мікроконтролером. Модель OSI та стандарт IEEE 802.3 визначають, що канальний рівень (Data Link Layer) поділяється на два підрівні: MAC (Media Access Control) та LLC (Logical Link Control). Саме MAC-підрівень відповідає за формування кадрів Ethernet, обчислення та перевірку CRC-32, дотримання міжкадрових інтервалів, обробку колізій (у напівдуплексному режимі) та фільтрацію за MAC-адресою. Фізичний рівень (PHY) відповідає за кодування бітів у сигнали для конкретного середовища передавання (вита пара, оптоволокно, SPE), автоперемовляння швидкості та дуплексу, формування лінійних сигналів, гальванічну розв'язку через імпульсні трансформатори. У сучасних мікроконтролерах, таких як STM32F4/F7/H7, SAM E70, i.MX RT, MAC-рівень інтегровано безпосередньо в кристал у вигляді периферійного модуля Ethernet MAC (часто з підтримкою DMA для дескрипторів кадрів), тоді як PHY майже завжди є зовнішньою мікросхемою. Такий поділ зумовлений тим, що PHY потребує прецизійних аналогових кіл, які технологічно складніше інтегрувати в чисто цифровий кристал, а також необхідністю гнучкості для підтримки різних фізичних середовищ (100BASE-TX, 10BASE-T1S, 1000BASE-T). Інтерфейсом між MAC і PHY слугують стандартизовані шини MII (Media Independent Interface) та його похідні, серед яких домінуюче місце у вбудованих системах посідає RMII (Reduced MII).

Стандартний інтерфейс MII, визначений у IEEE 802.3 Clause 22, використовує 16 сигнальних ліній для даних (4-бітна передача TXD[3:0] та 4-бітний прийом RXD[3:0]), тактові сигнали TX_CLK та RX_CLK (25 МГц для 100 Мбіт/с), сигнали валідності даних, помилок і керування. Така кількість виводів часто надлишкова для мікроконтролерів із обмеженою кількістю GPIO. RMII, специфікований у RMII Specification Rev. 1.2 (RMII Consortium, 1998), скорочує кількість ліній до 7 завдяки двом ключовим спрощенням. По-перше, замість двох окремих тактових сигналів TX_CLK та RX_CLK використовується єдиний опорний тактовий сигнал REF_CLK частотою 50 МГц, який подається на обидва пристрої. По-друге, ширина шини даних зменшується до 2 біт у кожному напрямку (TXD[1:0], RXD[1:0]), при цьому бітова швидкість 100 Мбіт/с досягається завдяки тому, що за один такт REF_CLK передається 2 біти (50 МГц × 2 = 100 Мбіт/с). Крім того, сигнал CRS_DV об'єднує функції Carrier Sense та RX Data Valid. Більшість PHY для вбудованих систем (LAN8720, LAN8742, DP83848) підтримують обидва режими, але за замовчуванням працюють у RMII. Структурну схему підключення MAC до PHY через RMII із зазначенням усіх сигналів показано на рисунку 5.4.

Керування PHY здійснюється через двопровідний інтерфейс MDIO/MDC, який функціонально подібний до I²C і описаний у IEEE 802.3 Clause 22/45. Через MDIO майстер (MAC) може читати та записувати 16-бітні регістри PHY, зокрема регістр базового статусу (BMSR, адреса 0x01), регістр керування (BMCR, 0x00), регістри автоузгодження, ідентифікатор виробника. Стандартний стек, як-от LwIP, використовує ці функції для визначення статусу з'єднання (link up/down) та узгодженої швидкості. Слід зазначити, що 10BASE-T1S PHY, розглянутий у попередньому підрозділі, також підключається через RMII і керується через MDIO, що робить апаратну частину незмінною для розробника при переході між різними фізичними середовищами.

Порівняння характеристик MII та RMII наведено в таблиці 5.6. Дані взято з IEEE 802.3, RMII Specification та технічних посібників STM32.

Переходячи від апаратного до програмного рівня, слід підкреслити, що сам по собі MAC-контролер не робить пристрій мережевим. Потрібен стек протоколів TCP/IP, який реалізує мережевий і транспортний рівні (IP, TCP, UDP), а також прикладні протоколи (MQTT, HTTP, DHCP, DNS).

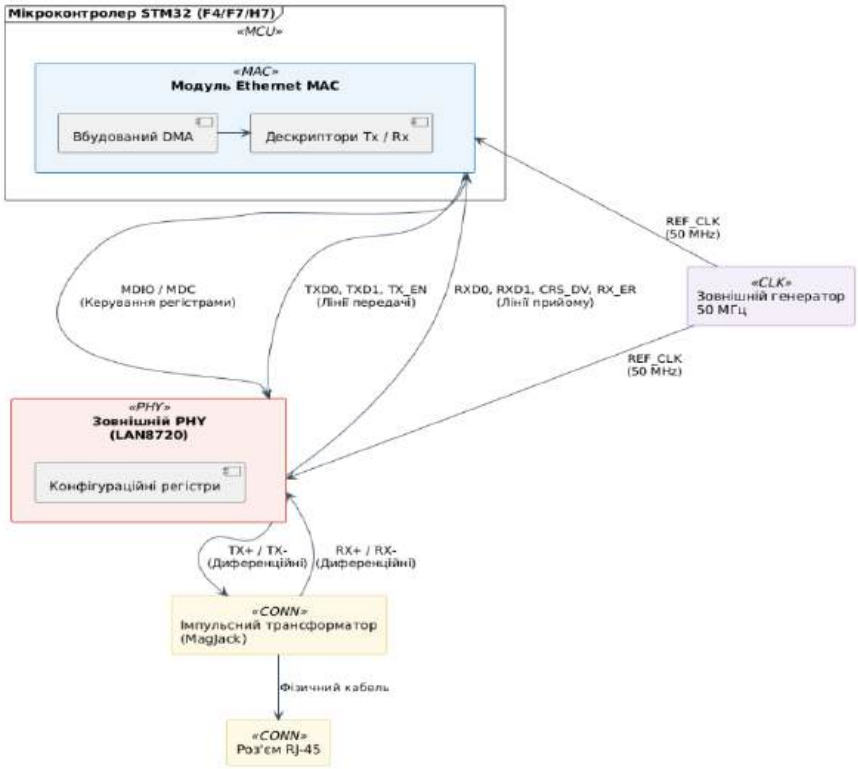


Рисунок 5.4 – Структурна схема з’єднання MAC STM32 з PHY через RMII

Таблиця 5.6 – Порівняння інтерфейсів MII та RMII

Параметр	MII	RMII
Кількість сигнальних ліній	~16	~7–8
Розрядність даних	4 біти (TXD[3:0] / RXD[3:0])	2 біти (TXD[1:0] / RXD[1:0])
Тактовий сигнал	окремі TX_CLK, RX_CLK	єдиний REF_CLK (спільний для MAC та PHY)
Частота для 100 Мбіт/с	25 МГц	50 МГц
Частота для 10 Мбіт/с	2,5 МГц	50 МГц (дані передаються кожен 10-й такт)
Максимальна швидкість PHY	100 Мбіт/с	100 Мбіт/с
Підтримка в STM32	у старших серіях (F4/F7/H7)	у переважній більшості
Типовий PHY	DP83848 (TI)	LAN8720, LAN8742 (Microchip)

У світі вбудованих систем абсолютне лідерство належить двом відкритим стекам: LwIP (Lightweight IP), що розробляється спільнотою і підтримується авторами на чолі з Адамом Дункельсом, та FreeRTOS+TCP, який поширюється Amazon Web Services як частина вільної операційної системи реального часу FreeRTOS.

LwIP є де-факто стандартом для невеликих мікроконтролерів завдяки своїй модульності, малому обсягу оперативної пам'яті (від 15 КБ ОЗП для мінімальної конфігурації UDP) і широкій підтримці виробниками. Компанія STMicroelectronics офіційно інтегрувала LwIP у пакети STM32Cube (документація UM1713, User Manual for STM32Cube Ethernet IAP example), і більшість прикладів для STM32F4/F7/H7 використовують саме цей стек. Архітектура LwIP передбачає кілька рівнів абстракції: драйвер PHY (через MDIO), драйвер MAC (низькорівневий обробник переривань, що копіює кадри з дескрипторів DMA в pbuf-структури LwIP), ядро TCP/IP (обробка IP-пакетів, фрагментація, TCP-автомати, UDP) та сокетний API. Розробник може працювати або через Raw API (зворотні виклики без операційної системи, мінімальне ОЗП), або через Netconn/Socket API (сумісний із POSIX, зручний для перенесення коду, потребує FreeRTOS). Типову архітектуру програмного стеку Ethernet на базі STM32 із FreeRTOS та LwIP показано на рисунку 5.5.

FreeRTOS+TCP, своєю чергою, є нативним для екосистеми FreeRTOS і тісно інтегрується з її примітивами (семафори, черги, таймери). Він використовує оригінальну реалізацію TCP/UDP, що пройшла незалежне тестування, і надає сокетний API, максимально наближений до Berkeley Sockets. Згідно з посібником FreeRTOS+TCP User Guide (AWS, 2023), цей стек може працювати як із FreeRTOS, так і в однозадачному режимі без ОС.

Практична реалізація будь-якого Ethernet-проєкту на STM32 розпочинається з конфігурації CubeMX, де вмикають модуль ETH (MAC), обирають RMII, налаштовують виводи, активують DMA та налаштовують контролер переривань. Після генерації коду наступним кроком є увімкнення LwIP у Middleware, де задаються ключові параметри в lwipopts.h: розміри пулів pbuf, кількість TCP-сегментів, MSS, пріоритети потоків. У головному циклі викликається MX_LWIP_Process(), яка обслуговує таймери ARP, DHCP, TCP-ретрансмісій.

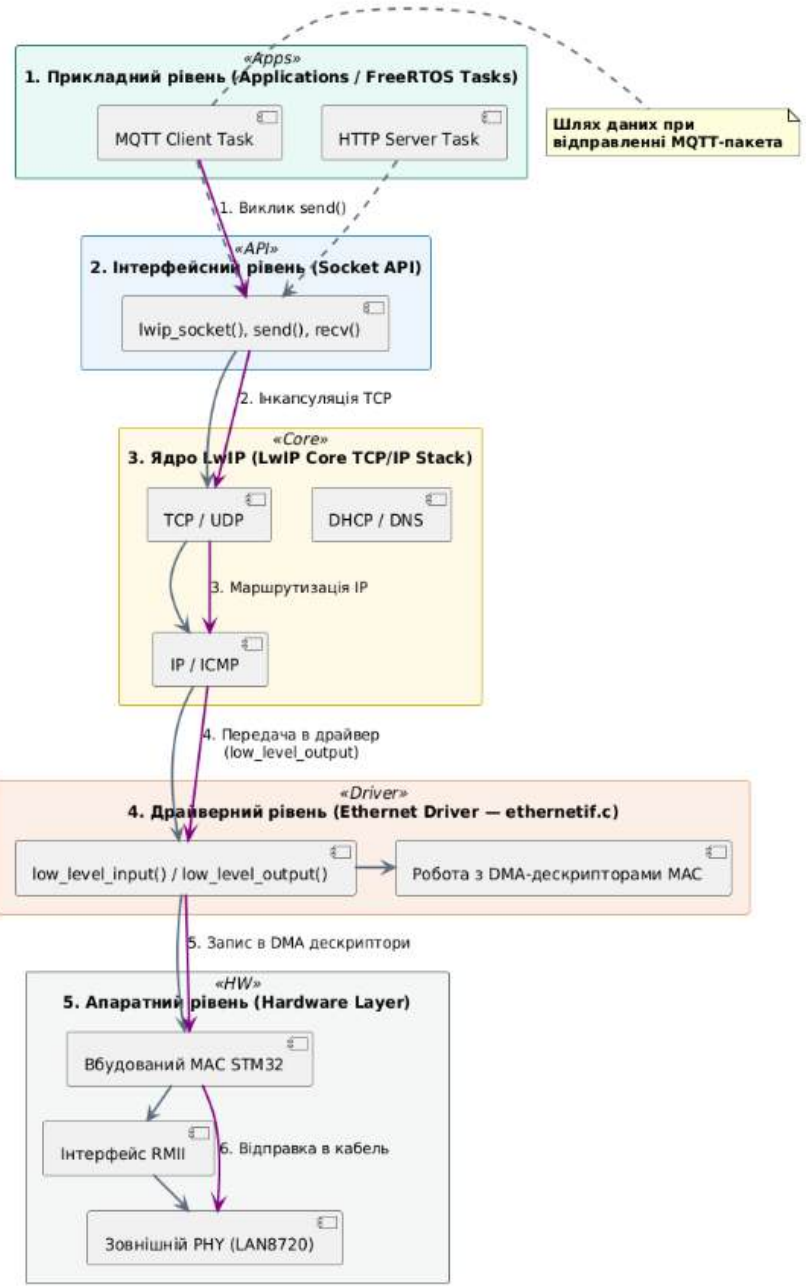


Рисунок 5.5 – Архітектура програмного стеку Ethernet на STM32 із FreeRTOS та LwIP

Типовий код створення простого TCP-сервера або MQTT-клієнта стає майже ідентичним до коду на ПК завдяки сокетам, що ілюструє лістинг 5.5, де наведено приклад створення MQTT-клієнта на STM32H723 з використанням LwIP та бібліотеки coreMQTT.

Лістинг 5.5 – Створення MQTT-клієнта на STM32H723 (FreeRTOS + LwIP + coreMQTT)

```
// Джерела: coreMQTT (FreeRTOS), LwIP (ST), STM32CubeH7
#include "main.h"
#include "lwip.h"
#include "lwip/sockets.h"
#include "core_mqtt.h"
// ------- Конфігурація MQTT -------
#define MQTT_BROKER_IP   "192.168.1.100"   // IP-адреса брокера
#define MQTT_BROKER_PORT 1883              // Порт брокера
#define MQTT_CLIENT_ID   "STM32H723_Client"
MQTTContext_t mqtt_ctx;     // Контекст MQTT-клієнта
// ============================================
// 1. ІНІЦІАЛІЗАЦІЯ МЕРЕЖІ (Ethernet + LwIP)
// ============================================
/** * @brief Ініціалізація Ethernet та LwIP (генерується CubeMX)
 * @note  Функції MX_ETH_Init() та MX_LWIP_Init() створюються
CubeMX * @note Перед MQTT-підключенням обов'язково
перевірити DHCP-статус */
void MX_ETH_Init(void) { //  ...налаштування Ethernet (HAL) ...}
void MX_LWIP_Init(void) {// ...ініціалізація LwIP (IP, маска,
шлюз, DHCP)...}
// ============================================
// 2. ПЕРЕВІРКА DHCP-СТАТУСУ
// ============================================
/** @brief Очікування отримання IP-адреси від DHCP-сервера
*@note Необхідно перед підключенням до MQTT-брокера */
uint8_t IsDHCPReady(void) {
    extern struct netif gnetif;
    return (gnetif.ip_addr.addr != 0) ? 1 : 0;//IP != 0.0.0.0
}
// ============================================
// 3. ЗАДАЧА FREERTOS ДЛЯ MQTT-КЛІЄНТА
// ============================================
/** * @brief Задача FreeRTOS, яка реалізує MQTT-клієнта
 * @param argument  Параметр задачі (не використовується)
 * @note  Використовує BSD-сокети (lwip_socket)
 * @note  Послідовність: сокет → підключення → MQTT_Init →
MQTT_Connect → MQTT_Publish */
void mqtt_task(void *argument) {
    int sock;
    struct sockaddr_in server_addr;
    MQTTStatus_t status;
    // Очікування отримання IP від DHCP
    while (!IsDHCPReady()) {
        vTaskDelay(pdMS_TO_TICKS(500));    }
    // ----- 1. Створення TCP-сокета -----
    sock = lwip_socket(AF_INET, SOCK_STREAM, 0);// IPv4, TCP
    if (sock < 0) { // Обробка помилки  vTaskDelete(NULL);    }
```

```
    // ---- 2. Налаштування адреси брокера ---
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(MQTT_BROKER_PORT);
    server_addr.sin_addr.s_addr = inet_addr(MQTT_BROKER_IP);
    // ----- 3. Підключення до брокера ---
    if (lwip_connect(sock, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0) {  // Обробка помилки
    lwip_close(sock);    vTaskDelete(NULL);    }
    // ---- 4. Ініціалізація MQTT-контексту (coreMQTT) ---
    MQTT_Init(&mqtt_ctx,                // Контекст
              transport_send,    // Функція відправлення
              transport_recv,    // Функція прийому
              get_time_ms);     // Функція часу (мс)
    // ---- 5. Підключення до MQTT-брокера ----
    MQTTConnectInfo_t connect_info = {
        .cleanSession = true,
        .clientId = MQTT_CLIENT_ID,
        .clientIdLength = sizeof(MQTT_CLIENT_ID) - 1,
        .keepAliveSeconds = 60,
    };
    status = MQTT_Connect(&mqtt_ctx, &connect_info, NULL,
1000);
    if (status != MQTTSuccess) {   // Обробка помилки
    lwip_close(sock);        vTaskDelete(NULL);    }
    // ---- 6. Публікація повідомлення ----
    const char *topic = "test/topic";
    const char *payload = "Hello from STM32H723!";
    MQTTPublishInfo_t publish_info = {
        .qos = MQTTQoS0,
        .topicName = topic,
        .topicNameLength = strlen(topic),
        .payload = payload,
        .payloadLength = strlen(payload),
    };
    status = MQTT_Publish(&mqtt_ctx, &publish_info, 0);
    if (status != MQTTSuccess) {        // Обробка помилки    }
    // Основний цикл обробки MQTT-подій
    while (1) {
        MQTT_ProcessLoop(&mqtt_ctx, 100);   // Обробка вхідних
повідомлень
        vTaskDelay(pdMS_TO_TICKS(10));    }   }
// ========================================
// 4. ЗАПУСК СИСТЕМИ
// ========================================
int main(void) {
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_ETH_Init();          // Ініціалізація Ethernet
    MX_LWIP_Init();         // Ініціалізація LwIP
    MX_FREERTOS_Init();     // Створення задачі mqtt_task
    osKernelStart();        // Запуск FreeRTOS
    while (1) {}
}
```

Порівняльний аналіз ключових параметрів LwIP та FreeRTOS+TCP, необхідний для свідомого вибору стеку, наведено в таблиці 5.7. Дані ґрунтуються на офіційній документації обох стеків та досвіді їх застосування, описаному в журналі IEEE Embedded Systems Letters (том 15, 2023).

Таблиця 5.7 – Порівняння стеків LwIP та FreeRTOS+TCP для STM32

Критерій	LwIP (2.1.2)	FreeRTOS+TCP (3.1)
Обсяг Flash (приблизно)	~40–100 КБ	~50–80 КБ
Мінімальний обсяг ОЗП	~15 КБ (UDP) – 40 КБ (TCP)	~20 КБ (TCP+UDP)
Підтримка IPv6	так (LwIP 2.x)	обмежена (експериментальна у v3.1; повна у v4.0+)
Сокетний API	через Netconn або POSIX (Berkeley-alike)	так (Berkeley Sockets)
Raw API (без ОС)	так (NO_SYS=1)	ні (потребує FreeRTOS)
Інтеграція з AWS IoT	через адаптери (наприклад, CoreMQTT-Agent)	нативна (Amazon FreeRTOS / X-CUBE-AWS)
Ліцензія	BSD-style	MIT (з версії 2.0.0)
Типова застосованість у STM32Cube	базова (приклади NUCLEO)	рекомендована для AWS IoT Core

Поєднання інтегрованого MAC, стандартного інтерфейсу RMII до зовнішнього PHY та потужного, але компактного стеку LwIP або FreeRTOS+TCP перетворює мікроконтролер на повноцінний мережевий вузол, здатний обмінюватися даними через Інтернет на швидкостях, обмежених лише фізичним середовищем. Наступний підрозділ, присвячений Time-Sensitive Networking, покаже, як детермінований Ethernet (TSN) розширює межі реального часу, дозволяючи замінити спеціалізовані польові шини єдиною мережею підприємства.

5.2.3. Time-Sensitive Networking (TSN) профілі IEEE 802.1Q для детермінованого Ethernet

Стандартний Ethernet, незважаючи на свою величезну пропускну здатність і універсальність, має фундаментальний недолік, що тривалий час унеможливлював його застосування в критичних до часу системах керування: комутатори Ethernet обробляють трафік за принципом «найкращих зусиль» (best effort), буферизуючи кадри в чергах і обслуговуючи їх у порядку надходження або за пріоритетами, але без жодних гарантій щодо максимальної затримки або її варіативності. У моменти перевантаження кадр

може бути затриманий на невизначений час або навіть відкинутий, що є неприйнятним для замкнених контурів керування рухом, промислових роботів, синхронізованих вимірювальних систем і автомобільних мереж. Усвідомлюючи цю проблему, робоча група IEEE 802.1 розробила комплекс доповнень до стандартів IEEE 802.1Q та IEEE 802.3, об'єднаних під загальною назвою Time-Sensitive Networking (TSN). TSN не є окремим протоколом; це набір взаємодоповнюючих механізмів, які, працюючи разом, трансформують стандартний Ethernet із недетермінованої мережі в мережу з гарантованою латентністю, мінімальним джиттером, синхронізацією часу з субмікросекундною точністю та резервуванням каналів зв'язку. Саме TSN розглядається як ключова технологія для конвергенції інформаційних (IT) та операційних (OT) мереж у рамках Industry 4.0, а також як основа для магістральних мереж транспортних засобів наступного покоління. Як зазначено в оглядовій статті Л. Ло Белло та співавторів (IEEE Communications Surveys and Tutorials, том 21, 2020), TSN є однією з найвпливовіших інновацій у галузі промислових комунікацій після появи самого Ethernet.

Синхронізація часу є фундаментом, на якому будуються всі інші механізми TSN. Для цього використовується протокол IEEE 802.1AS, відомий як Generalized Precision Time Protocol (gPTP), що є профілем стандарту IEEE 1588-2008 (PTPv2), оптимізованим для мостів (комутаторів) Ethernet. gPTP забезпечує розподіл точного часу від одного Grandmaster Clock до всіх вузлів мережі з точністю близько 100 наносекунд для повністю gPTP-сумісного обладнання. Механізм спирається на апаратне штемпелювання (timestamping) моментів проходження пакетів PTP через PHY-рівень, що усуває програмні затримки. На основі глобального часу кожен комутатор може виконувати планування черг із прив'язкою до заздалегідь узгодженого розкладу.

Другим критичним компонентом є механізм планування трафіку IEEE 802.1Qbv (Time-Aware Shaper, TAS). Він дозволяє розділити час роботи вихідного порту комутатора на циклічні вікна, в яких дозволено передавання лише кадрам певних класів. Критичний до часу трафік (клас Scheduled Traffic, ST) отримує виділені вікна, під час яких інші класи (наприклад, фоновий TCP/IP) не мають права займати лінію. Завдяки цьому кадр Scheduled Traffic гарантовано буде відправлений без затримки на очікування в черзі, що забезпечує жорстку детермінованість із максимальною латентністю, яка обчислюється на етапі проектування мережі. Доповнює TAS механізм IEEE 802.3br / IEEE 802.1Qbu (Frame Preemption), який дозволяє перервати передавання довгого не-критичного кадру (express) для

негайного відправлення критичного, якщо той прибув під час незапланованого вікна. Це мінімізує латентність навіть за неоптимального розкладу.

Для підвищення надійності застосовують резервування кадрів згідно з IEEE 802.1CB (Frame Replication and Elimination for Reliability). Відправник дублює критичний кадр і передає його через два незалежні шляхи в мережі; отримувач приймає перший примірник, а другий відкидає. Якщо один із шляхів виходить із ладу, дані все одно доходять без втрати жодного кадру, що є обов'язковою вимогою для систем функціональної безпеки (IEC 61508 SIL3). Крім того, IEEE 802.1Qca (Path Control and Reservation) надає засоби для явного конфігурування маршрутів, відмінних від стандартного Spanning Tree Protocol, що дозволяє оптимізувати шляхи для критичного трафіку.

Сукупність згаданих стандартів формує так звані TSN-профілі — добірки обов'язкових та опціональних функцій, орієнтовані на конкретні галузі. Профіль IEC/IEEE 60802 для промислової автоматизації є спільною розробкою IEC та IEEE і визначає вимоги до комутаторів, топологій і параметрів для конвергентних мереж Industry 4.0. Профіль IEEE 802.1DG для автомобільних мереж адаптує TSN для заміни традиційних CAN/LIN-магістралей на єдину Ethernet-основу. В аерокосмічній галузі розробляються профілі TSN для авіоніки, що забезпечують детерміновану передавання критичних даних польоту через стандартний Ethernet. Ключові TSN-стандарти зведено в таблиці 5.8.

Таблиця 5.8 – Ключові стандарти TSN (IEEE 802.1)

Стандарт	Назва	Функція
IEEE 802.1AS-2020	Timing and Synchronization	синхронізація часу (gPTP – generalized Precision Time Protocol), точність ~100 нс. Визначає домен gPTP, у якому системний час є узгодженим.
IEEE 802.1Qbv	Time-Aware Shaper (TAS)	планування черг (Gate Control Lists), гарантовані часові вікна для критичного трафіку (Time-Sensitive Traffic).
IEEE 802.1Qbu / 802.3br	Frame Preemption	переривання передачі некритичних кадрів для зменшення латентності критичних кадрів.
IEEE 802.1CB	Frame Replication and Elimination for Reliability (FRER)	резервування шляхів: реплікація кадрів для надлишкової передачі та виявлення/видалення дублікатів для підвищення надійності.

Кінець таблиці 5.8

Стандарт	Назва	Функція
IEEE 802.1Qcc	Stream Reservation Protocol (SRP) Enhancements	конфігурування потоків (централізоване, з підтримкою UNI); розширення параметрів резервування для додатків з високими вимогами до надійності та латентності.
IEEE 802.1Qci	Per-Stream Filtering and Policing (PSFP)	захист від помилкових/шкідливих кадрів: фільтрація, полісинг та вибір класу обслуговування для кожного потоку на основі розкладу.

З апаратного погляду впровадження TSN потребує комутаторів і кінцевих пристроїв із підтримкою відповідних функцій. Провідні виробники напівпровідників уже пропонують TSN-сумісні MAC/комутатори: NXP SJA1110 (4-портовий TSN-комутатор), Microchip LAN9668, Texas Instruments DP83640 (PHY із підтримкою IEEE 1588), Intel i210 (контролер із TSN). У світі мікроконтролерів STMicroelectronics інтегрує підтримку IEEE 802.1AS (gPTP) у модулі Ethernet MAC у нових сімействах STM32MP2, а також пропонує рішення на базі FPGA (через партнерство з TT Tech). На практиці невеликий вбудований вузол на STM32H7 може бути TSN-кінцевою точкою за умови використання відповідного PHY з підтримкою штемпелювання (наприклад, DP83640) і стеку TSN (наприклад, комерційні стеки від TTTech Industrial, RT-Labs або відкриті реалізації OpenTSN на базі Linux). Архітектуру мережі TSN із різними класами трафіку та Time-Aware Shaper у комутаторі показано на рисунку 5.6.

Для студента-розробника вбудованих систем важливо розуміти, що TSN не є «чарівною паличкою», яка робить будь-який Ethernet детермінованим. Впровадження TSN вимагає, щоб усі комутатори та кінцеві вузли на шляху критичного трафіку підтримували необхідний набір стандартів, а також ретельного проектування розкладу вікон (scheduling) на етапі інтеграції мережі. Проте переваги — наскрізна IP-адресація, висока швидкість, єдина інфраструктура для керування й передавання даних, відкриті стандарти — роблять TSN безальтернативним вибором для нових проектів у промисловому IoT, автомобілебудуванні та авіоніці. Як і CAN XL, що був розглянутий раніше, TSN є свідченням того, що магістральні технології продовжують еволюціонувати, відповідаючи на виклики реального часу, що раніше були прерогативою спеціалізованих польових шин.

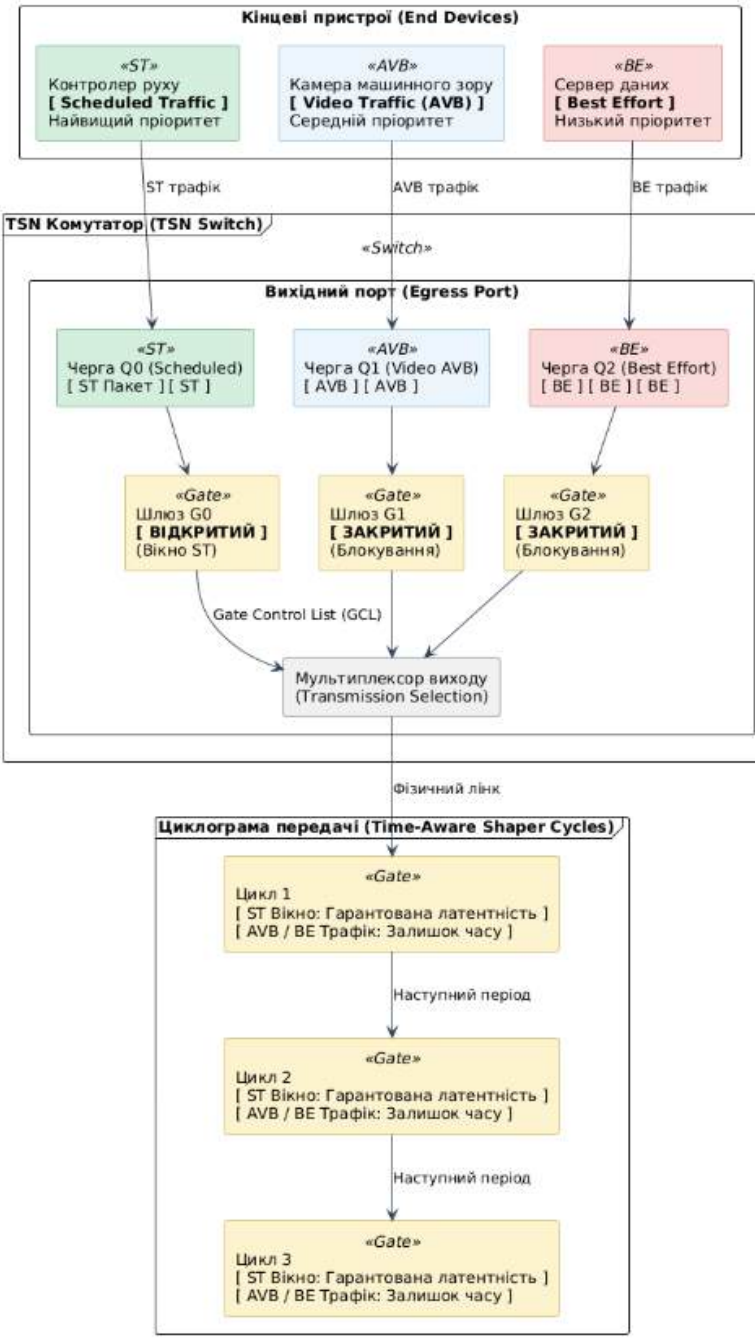


Рисунок 5.6 – Архітектура TSN-мережі з Time-Aware Shaper

5.3. PCI Express (PCIe) у гетерогенних системах: зв’язок мікроконтролера з FPGA або GPU-модулем

Інтерфейс PCI Express (PCIe), стандартизований консорціумом PCI-SIG (Peripheral Component Interconnect Special Interest Group), є на сьогодні найпоширенішою високошвидкісною послідовною шиною для з’єднання процесорів із периферійними пристроями як у персональних комп’ютерах, так і в серверних, вбудованих та гетерогенних системах. Його фундаментальна відмінність від класичних паралельних шин (PCI, PCI-X) полягає в переході до пакетної, послідовної, двоточкової топології з використанням високошвидкісних диференційних пар (ліній, lanes), що забезпечує масштабовану пропускну здатність від 2,5 ГТ/с (гігатранзакцій за секунду) на одну лінію в поколінні Gen 1 до 64 ГТ/с у Gen 6. У вбудованих системах PCIe застосовується для підключення високопродуктивних периферійних модулів — твердотільних накопичувачів NVMe, мережевих адаптерів, графічних процесорів (GPU) та, що особливо важливо для гетерогенних обчислень, програмованих логічних інтегральних схем (FPGA). У цьому підрозділі розглядаються архітектурні особливості PCIe, його еволюція, практичні аспекти використання для зв’язку мікроконтролера (або вбудованого процесора) з FPGA чи GPU, а також типові топології гетерогенних систем, що будуються на цій основі.

Архітектура PCIe базується на багаторівневій моделі, яка включає фізичний, канальний (Data Link) та транзакційний (Transaction) рівні. На фізичному рівні використовується одна або кілька ліній (lane), кожна з яких складається з двох диференційних пар — одна для передачі, друга для прийому, що реалізує повнодуплексний зв’язок. Лінії групуються в лінки (links) шириною x1, x2, x4, x8, x16 або x32, що дозволяє гнучко масштабувати пропускну здатність пропорційно до кількості ліній. Канальний рівень забезпечує надійну доставку пакетів між двома пристроями за допомогою механізму підтверджень (ACK/NACK) та повторних передач, а також керування потоком. Транзакційний рівень оперує транзакціями читання/запису в адресному просторі пам’яті, вводу-виводу або конфігураційних регістрів, використовуючи адресну маршрутизацію (для пристроїв) або маршрутизацію за ідентифікатором (ID routing для мостів). Конфігурація системи, ініціалізація шини та виявлення пристроїв виконуються програмним забезпеченням системи (BIOS/UEFI або завантажувачем) у тісній взаємодії з кореневим комплексом (Root Complex) — центральним вузлом ієрархії PCIe, який інтегрується в процесор або чіпсет.

Еволюція PCIe демонструє постійне подвоєння швидкості передавання з кожним новим поколінням завдяки вдосконаленню методів модуляції та кодування. У таблиці 5.9 наведено основні характеристики поколінь від Gen 1 до Gen 6, згідно з офіційними специфікаціями PCI-SIG (PCI Express Base Specification, ревізії 1.0a – 6.0.1). Для вбудованих гетерогенних систем найбільш поширеними на сьогодні є покоління Gen 3 (8 ГТ/с) та Gen 4 (16 ГТ/с), тоді як Gen 5 та Gen 6 починають застосовуватися у високопродуктивних FPGA і GPU-модулях для периферійного штучного інтелекту та систем реального часу.

Таблиця 5.9 – Основні характеристики поколінь PCIe

Покоління	Символьна швидкість на лінію (ГТ/с)	Пропускна здатність x1 (Гбіт/с)	Пропускна здатність x16 (Гбіт/с)	Кодування	Рік затвердження
PCIe 1.0/1.1	2,5 ГТ/с	2,5 Гбіт/с	40 Гбіт/с	8b/10b	2003 / 2005
PCIe 2.0/2.1	5 ГТ/с	5 Гбіт/с	80 Гбіт/с	8b/10b	2007
PCIe 3.0/3.1	8 ГТ/с	8 Гбіт/с	128 Гбіт/с	128b/130b	2010 / 2014
PCIe 4.0	16 ГТ/с	16 Гбіт/с	256 Гбіт/с	128b/130b	2017
PCIe 5.0	32 ГТ/с	32 Гбіт/с	512 Гбіт/с	128b/130b	2019
PCIe 6.0	64 ГТ/с	64 Гбіт/с	1024 Гбіт/с	PAM-4 + FEC (FLIT)	2022

У гетерогенних вбудованих системах, особливо орієнтованих на задачі штучного інтелекту, машинного зору, програмно-визначеного радіо або високошвидкісного збору даних, мікроконтролер або вбудований процесор виступає в ролі центрального обчислювального вузла (хоста), тоді як FPGA або GPU підключаються через PCIe як прискорювачі або периферійні пристрої. Така архітектура дозволяє поєднати низьку латентність детермінованого керування на ядрі реального часу (наприклад, ARM Cortex-R або Cortex-M, інтегрованому в SoC) з масовою паралельною обробкою потоків даних на FPGA або GPU, об’єднаних високошвидкісним каналом PCIe.

Типовий представник платформи, що реалізує цю концепцію — сімейство Xilinx Zynq UltraScale+ MPSoC, яке поєднує чотириядерний ARM Cortex-A53, два ядра Cortex-R5 для реального часу, програмовану логіку (FPGA) та контролер PCIe Gen 2/3/4 (залежно від моделі). Через PCIe до такого SoC можна під’єднати зовнішній GPU (наприклад, дискретний

модуль на базі AMD або NVIDIA) або додаткову FPGA більшої ємності. Інший приклад — NXP i.MX 8M Plus, який містить ARM Cortex-A53, нейронний процесор (NPU) та PCIe Gen 3 для підключення зовнішніх прискорювачів або NVMe-накопичувачів. Навіть деякі високопродуктивні мікроконтролери з ядром ARM Cortex-M7, такі як STM32H7, не мають власного PCIe, але можуть взаємодіяти з FPGA через зовнішній міст PCIe-to-SPI (наприклад, MAX14850) або через власний інтерфейс FMC (Flexible Memory Controller) для паралельного підключення FPGA, хоча це й не є справжнім PCIe.

У повноцінних SoC підтримка PCIe реалізується апаратним контролером, сумісним зі специфікаціями PCI-SIG. Програмна модель для розробника вбудованих Linux-систем спирається на стандартну підсистему ядра Linux: драйвер контролера PCIe (наприклад, pcie-xilinx, pci-imx6), який виконує перерахування шини, конфігурує BAR (Base Address Registers) та створює в sysfs файли пристроїв. Прикладний розробник може взаємодіяти з FPGA через символьний пристрій /dev/xdma0, використовуючи стандартні read()/write() або mmap(), як це реалізовано у фреймворку Xilinx DMA/Bridge Subsystem for PCI Express (XDMA). Для bare-metal систем на Cortex-R5 існують компактні бібліотеки (наприклад, pcie-lib від Xilinx), які реалізують мінімально необхідний набір функцій для ініціалізації PCIe без операційної системи. Схематичний приклад коду для ініціалізації PCIe-з'єднання з FPGA на Zynq MPSoC у середовищі Vitis (bare-metal) наведено в лістингу 5.6, де демонструється виявлення пристрою та налаштування DMA-трансферів.

Лістинг 5.6 – Ініціалізація PCIe Root Complex та налаштування DMA до FPGA на Zynq UltraScale+ (bare-metal, Xilinx pcie-lib)

```
1:// Платформа: Zynq UltraScale+ (bare-metal, Xilinx pcie-lib /
2:xdmapcie)
3:// Джерела: Xilinx Vitis drivers(xdmapcie, pciepsu),UG1085,
4:PG195
5:// ============================================
6:#include "xparameters.h"
7:#include "xdmapcie.h"
8:#include "xdmapcie_rc.h"  // API для Root Complex режиму
9:// ------ Дескриптори PCIe -----
10:#define PCIE DEVICE ID  XPAR XDMAPCIE 0 DEVICE ID //ID пристрою
11:з xparameters.h
12:XDmaPcie Pcie;              // Екземпляр драйвера XDMA PCIe
13:XDmaPcie_Config *PcieConfig;   // Конфігурація драйвера
14:// Буфер даних у DDR (джерело) та BRAM FPGA (призначення)
15:#define BUFFER_SIZE 1024
16:uint8_t TxBuffer[BUFFER_SIZE] __attribute__((aligned(64)));
17:// 64-байтове вирівнювання для DMA
```

```
18:volatile uint8_t *FpgaBramPtr;    // Вказівник на BRAM FPGA
19:(після BAR-відображення)
20:// ------------------------------------------------
21:// @brief  Ініціалізація PCIe Root Complex та налаштування DMA
22:// @note   Після успішної ініціалізації FPGA відображається в
23:адресний простір процесора через BAR
24:// @note   Доступ до регістрів FPGA здійснюється як до звичайної
25:пам'яті
26:// ------------------------------------------------
27:int main(void) {
28:    int Status;
29:    XDmaPcie_Config *ConfigPtr;
30:    u32 BarIndex = 0;          // Використовуємо BAR0
31:    u64 BarSize;
32:    u64 BarAddress;
33:    // ---- 1. Ініціалізація драйвера PCIe (Root Complex) ----
34:    // Ініціалізація екземпляра XDmaPcie
35:    ConfigPtr = XDmaPcie_LookupConfig(PCIE_DEVICE_ID);
36:    if (!ConfigPtr) {        return XST_FAILURE;    }
37:    Status = XDmaPcie_CfgInitialize(&Pcie, ConfigPtr,
38:ConfigPtr->BaseAddress);
39:    if (Status != XST_SUCCESS) {       return XST_FAILURE;    }
40:    // -- 2. Перелік (Enumerate) PCIe-пристроїв у системі --
41:    // Виявлення підключених кінцевих пристроїв (Endpoints)
42:    Status = XDmaPcie_RC_Enumerate(&Pcie);
43:    if (Status != XST_SUCCESS) {       return XST_FAILURE;    }
44:    // -- 3. Виділення BAR для кінцевого пристрою (FPGA) --
45:  // Призначення BAR (Base Address Register) для доступу до FPGA
46:    Status = XDmaPcie_RC_AllocBar(&Pcie, 0 /*Endpoint Index */,
47:                                  BarIndex /* BAR0 */,
48:                                  &BarAddress, &BarSize);
49:    if (Status != XST_SUCCESS) {       return XST_FAILURE;    }
50:// Отримання вказівника на область пам'яті FPGA (BRAM через BAR)
51:    FpgaBramPtr = (volatile uint8_t *)(uintptr_t)BarAddress;
52:    // --- 4. Налаштування DMA-дескрипторів ---
53:// Ініціалізація DMA-каналу для передачі даних з DDR у BRAM FPGA
54:    // (Приклад для каналу H2C - Host to Card)
55:    // Заповнення буфера тестовими даними
56:    for (int i = 0; i < BUFFER_SIZE; i++) {
57:        TxBuffer[i] = (uint8_t)(i & 0xFF);
58:    }
59:    // Налаштування DMA-дескриптора (спрощено)
60:    // - Адреса джерела: TxBuffer (DDR)
61:    // - Адреса призначення: FpgaBramPtr (BRAM FPGA)
62:    // - Розмір: BUFFER_SIZE
63:    // - Напрямок: H2C (Host to Card)
64:    // Скидання DMA-каналу
65:    XDmaPcie_DmaReset(&Pcie, XDMAPCIE_DMA_H2C_MASK);
66:    // Налаштування дескриптора (залежить від реалізації DMA-
67:контролера)
68:    // У спрощеному вигляді - пряме копіювання через пам'ять
69:(Для повноцінного DMA використовуються XDmaPcie_DmaTransfer()
70:або робота з дескрипторними списками scatter-gather)
71:    // ---------- 5. Запуск DMA-передачі ----------
72:    // Виконання передачі даних з DDR у BRAM FPGA
73:    // (Приклад використання блочного DMA)
74:    Status = XDmaPcie_DmaTransfer(&Pcie,
```

```
            XDMAPCIE_DMA_H2C, // Напрямок: Host → Card
            (UINTPTR)TxBuffer, // Адреса джерела (DDR)
            (UINTPTR)FpgaBramPtr, // Адреса призначення (BRAM)
            BUFFER_SIZE,   // Розмір
            0);  // Таймаут
    if (Status != XST_SUCCESS) {        return XST_FAILURE;    }
    // --- 6. Доступ до FPGA як до звичайної пам'яті ---
    // Після ініціалізації та відображення BAR, доступ до
регістрів FPGA здійснюється безпосередньо через вказівник
    uint32_t reg_value=FpgaBramPtr[0];//Читання першого байта
BRAM
    // Запис у регістр FPGA
    FpgaBramPtr[0] = 0xAA;      // Запис у BRAM
    while (1) {         // Основний цикл     }
    return XST_SUCCESS;
}
```

Топологію гетерогенної системи з PCIe-з'єднанням показано на рисунку 5.7. На прикладі SoC, що виконує роль Root Complex, через лінії PCIe під'єднано FPGA-акселератор (наприклад, для попередньої обробки даних із швидкісних АЦП) та GPU-модуль (для інференції нейронної мережі). Обидва пристрої перебувають у власному адресному просторі PCIe, а обмін даними відбувається через DMA з мінімальним завантаженням центрального процесора.

Таким чином, PCI Express є ключовою технологією для побудови гетерогенних вбудованих систем, де поєднання мікроконтролерної (або мікропроцесорної) платформи з FPGA та GPU дозволяє досягти безпрецедентної продуктивності та гнучкості. Завдяки відкритості стандарту, широкому спектру доступних IP-блоків та підтримці виробниками, PCIe залишатиметься магістральним інтерфейсом для високошвидкісної міжчіпової взаємодії ще багато років. Цим завершується огляд високошвидкісних універсальних інтерфейсів; наступна частина посібника буде присвячена бездротовим технологіям, які виводять IoT-пристрої у світ глобальної зв'язності.

Окрім очевидних переваг високої пропускної здатності та масштабованості, практичне застосування PCIe у вбудованих гетерогенних системах вимагає врахування низки архітектурних нюансів, що безпосередньо впливають на продуктивність і детермінованість.

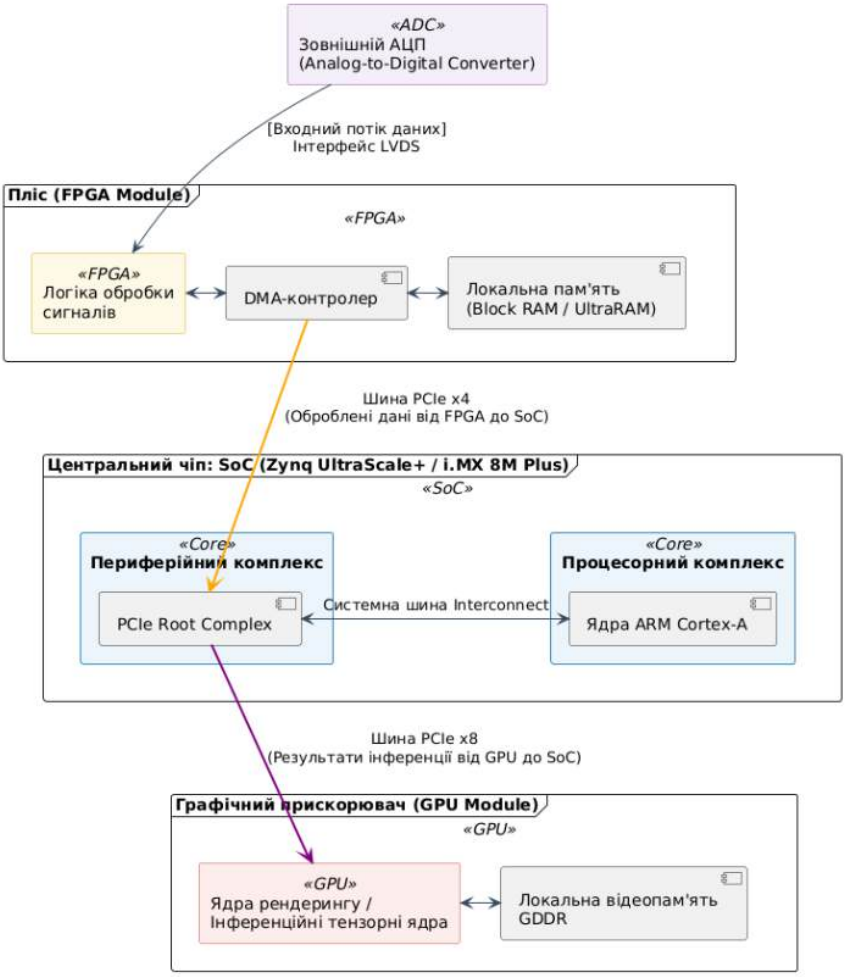


Рисунок 5.7 – Гетерогенна система на базі SoC із підключенням FPGA та GPU через PCIe

На програмному рівні критичне значення має підсистема переривань: сучасні контролери PCIe підтримують MSI/MSI-X, які замінюють традиційні апаратні переривання на передачу повідомлень через шину. Це унеможливлює конфлікти за лінії IRQ і забезпечує масштабовану обробку подій від FPGA-акселераторів, що генерують інтенсивний потік переривань. У системах реального часу MSI-X дозволяє призначати різним функціям пристрою окремі вектори переривань із власними пріоритетами, що дає змогу гнучко налаштовувати диспетчеризацію завдань.

Не менш важливим є ефективне управління прямим доступом до пам'яті (DMA). FPGA або GPU виступають як bus masters, ініціюючи DMA-транзакції без участі центрального процесора. Критичними параметрами при цьому є вирівнювання буферів (64/128 байт), розмір транзакційних пакетів (MPS) та налаштування кешування — їх некоректна конфігурація спричиняє фрагментацію шини, падіння пропускної здатності або помилки когерентності в багатоядерних системах. Для спрощення розробки виробники надають спеціалізовані драйверні бібліотеки, які інкапсулюють роботу з дескрипторами DMA, підтримують scatter-gather і автоматизують обробку завершення транзакцій, проте їх використання все одно вимагає глибокого розуміння моделей пам'яті та порядкових моделей обміну, щоб уникнути станів гонки.

Окремою важливою нішею є підключення високошвидкісних NVMe-накопичувачів. Завдяки низькій латентності та пропускній здатності до кількох гігабайт за секунду, вони дозволяють реалізувати потокове зберігання великих обсягів даних у реальному часі (відеоспостереження, радіолокація, реєстратори польотів). SoC із PCIe виступає хостом, а обмін даними відбувається через стандартний драйвер Linux або компактні реалізації для RTOS, що спрощує архітектуру та знижує вартість систем зберігання.

Попри появу альтернативних високошвидкісних інтерфейсів (CXL, Gen-Z), PCIe залишається фундаментом для більшості вбудованих гетерогенних платформ завдяки зрілій екосистемі, широкому вибору IP-блоків, доступності недорогих FPGA-модулів із вбудованим контролером PCIe та великому обсягу готових інженерних напрацювань. Зростання продуктивності периферійних обчислювальних модулів (GPU, NPU, TPU) у поєднанні з PCIe Gen 4/5 забезпечує достатню пропускну здатність для виконання ресурсомістких задач штучного інтелекту безпосередньо на межі мережі (Edge AI), що підтверджує домінуюче становище PCIe як магістрального інтерфейсу для міжчіпової взаємодії в найближчій перспективі.

ЧАСТИНА III.
БЕЗДРОТОВІ ІНТЕРФЕЙСИ ДЛЯ ІНТЕРНЕТУ РЕЧЕЙ

РОЗДІЛ 6.
ТЕХНОЛОГІЇ МАЛОГО РАДІУСУ ДІЇ (WPAN)

6.1. Bluetooth LE (BLE) за специфікацією Bluetooth Core 5.4 та 6.0

Bluetooth Low Energy (BLE), представлений комерційно у 2010 році як ключова інновація специфікації Bluetooth Core 4.0, докорінно змінив ландшафт бездротових персональних мереж (WPAN). На відміну від класичного Bluetooth (BR/EDR), орієнтованого на безперервне потокове передавання (аудіо, передача файлів), BLE створювався з нуля для ультранизького енергоспоживання, періодичного передавання невеликих обсягів даних та максимально швидкого встановлення з'єднання. Сучасні специфікації Bluetooth Core 5.4 (2023) та 6.0 (2024) розширюють ці можливості, додаючи функції двонапрямленого рекламування (Periodic Advertising with Responses — PAwR), покращеного визначення місцезнаходження (Channel Sounding), аудіо через LE (LE Audio з кодеком LC3) та ще більшої енергоефективності.

6.1.1. Структура стеку, GATT профілі, рекламні пакети

Для розробника вбудованих систем BLE є основним бездротовим інтерфейсом для зв'язку зі смартфонами, планшетами, персональними комп'ютерами та іншими мобільними пристроями, що робить його вивчення обов'язковим етапом опанування IoT-технологій.

Структура стеку BLE, стандартизована в документі Bluetooth Core Specification (розділ 2, Architecture), має чітко виражену ієрархію з трьох основних блоків: Controller (контролер), Host (хост) та Application (прикладний рівень). Контролер охоплює фізичний рівень (PHY) та рівень зв'язку (Link Layer) і зазвичай реалізується апаратно в окремому радіотракті (наприклад, nRF52840) або як частина SoC (ESP32-C3). PHY визначає робочі частоти в діапазоні 2,4000–2,4835 ГГц, розбитому на 40 каналів по 2 МГц, із яких 3 канали (37, 38, 39) є первинними рекламними (Primary Advertising Channels), а решта 37 — каналами даних (Data Channels). Link Layer керує станами пристрою — Standby, Advertising, Scanning, Initiating, Connection (Master/Slave), — формує пакети, обчислює CRC, виконує шифрування AES-CCM та реалізує алгоритм адаптивного стрибкоподібного перемикання частоти (Adaptive Frequency Hopping). Хост містить верхні рівні

стеку: L2CAP (Logical Link Control and Adaptation Protocol) для мультиплексування даних, ATT (Attribute Protocol) для доступу до атрибутів, GATT (Generic Attribute Profile) — надбудову над ATT, яка структурує дані в сервіси та характеристики, SM (Security Manager) для сполучення та шифрування, GAP (Generic Access Profile) для визначення ролей (Broadcaster, Observer, Peripheral, Central) та процедур виявлення. На рисунку 6.1 показано цю архітектуру з розділенням на контролер і хост, а також взаємодію через HCI (Host Controller Interface).

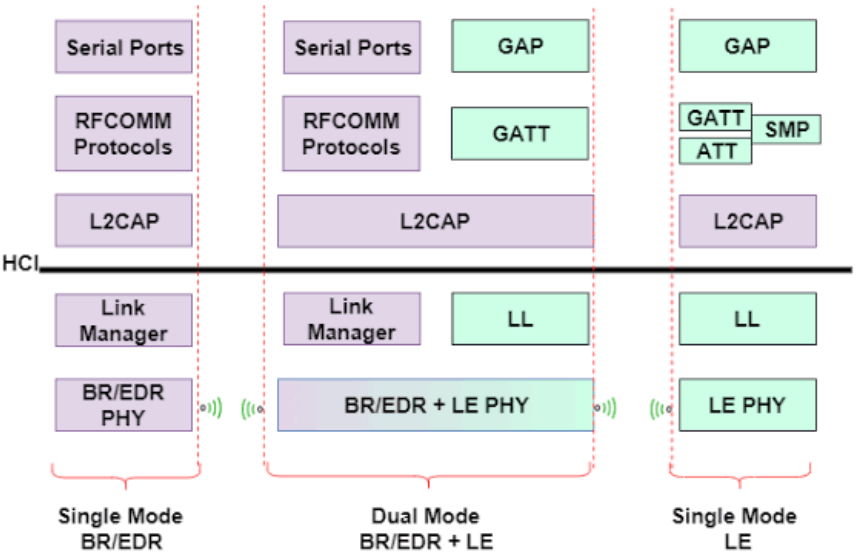


Рисунок 6.1 – Архітектура стеку BLE (Bluetooth Core 5.4)

GATT (Generic Attribute Profile) є серцем прикладної взаємодії BLE. Він визначає ієрархічну модель даних, за якою всі дані організовані як атрибути, згруповані в сервіси (Services), кожен із яких містить характеристики (Characteristics) та опціональні дескриптори (Descriptors). Сервіс ідентифікується 16- або 128-бітним UUID; стандартні UUID зарезервовані Bluetooth SIG (наприклад, 0x180F для Battery Service, 0x180D для Heart Rate Service), а виробники можуть створювати власні сервіси. Характеристика є атомарною одиницею даних, що складається з властивостей (Read, Write, Notify, Indicate тощо), значення та UUID. Дескриптор надає додаткову інформацію про характеристику; обов'язковим є дескриптор 0x2901 (Characteristic User Description) та 0x2902 (Client Characteristic Configuration Descriptor, CCCD), який дозволяє клієнту вмикати сповіщення

(notifications/indications). Взаємодія за GATT відбувається за моделлю «клієнт – сервер»: GATT-сервер (зазвичай периферійний пристрій, датчик) зберігає атрибути, а GATT-клієнт (зазвичай смартфон або центральний пристрій) надсилає запити на читання, запис, підписку на сповіщення. Прикладом стандартного профілю є Heart Rate Profile (HRP), який вимагає обов'язкової підтримки Heart Rate Service (UUID 0x180D) з характеристикою Heart Rate Measurement (0x2A37) та опціональною Body Sensor Location (0x2A38). У таблиці 6.1 наведено фрагмент стандартних сервісів та характеристик BLE.

Таблиця 6.1 – Приклади стандартних GATT-сервісів та характеристик

Сервіс (UUID)	Характеристика (UUID)	Властивості	Призначення
Heart Rate Service (0x180D)	Heart Rate Measurement (0x2A37)	Notify	частота пульсу в BPM (ударах за хвилину)
Battery Service (0x180F)	Battery Level (0x2A19)	Read, Notify	рівень заряду батареї у відсотках (0–100%)
Device Information (0x180A)	Manufacturer Name String (0x2A29)	Read	назва виробника пристрою
Environmental Sensing (0x181A)	Temperature (0x2A6E)	Read, Notify	температура в °C з роздільною здатністю 0.01 °C

Рекламні пакети (Advertising Packets) є ключовим механізмом, за допомогою якого пристрій BLE заявляє про свою присутність у просторі, передає інформацію про свої можливості або запрошує до з'єднання. Реклама передається на трьох первинних рекламних каналах (37, 38, 39), які спеціально обрані так, щоб мінімізувати інтерференцію з каналами Wi-Fi. Специфікація визначає кілька типів рекламних PDU (Protocol Data Unit), із яких найчастіше використовуються ADV_IND (connectable undirected advertising — загальна реклама з можливістю підключення), ADV_NONCONN_IND (non-connectable undirected — реклама без можливості підключення, для маячків), ADV_SCAN_IND (scannable undirected — дозволяє сканеру запитати додаткову інформацію через Scan Response). Кожен рекламний пакет містить поле AdvA (адреса рекламодавця, 6 байт) та поле AdvData (рекламні дані, до 31 байта в Core 5.4). AdvData структуроване як послідовність AD-структур, кожна з яких містить довжину, тип (AD Type) та значення. Типи AD визначають вміст: Type 0x01 — Flags (прапорці режиму), Type 0x02/0x03 — 16/128-бітні UUID сервісів, Type 0x09 — повна локальна

назва (Complete Local Name), Type 0x0A — рівень потужності передавача (Tx Power Level). Наприклад, маячок iBeacon використовує ADV_NONCONN_IND із Manufacturer Specific Data (AD Type 0xFF), що містить UUID, Major і Minor для ідентифікації.

Важливою інновацією Bluetooth Core 5.0 стало введення Extended Advertising, яке знімає обмеження в 31 байт завдяки використанню вторинних рекламних каналів (Secondary Advertising Channels, ті самі 37 каналів даних). Первинний пакет ADV_EXT_IND на первинному каналі містить лише вказівник (AuxPtr) на пакет AUX_ADV_IND на вторинному каналі, який може переносити до 255 байтів даних і підтримує ланцюжки пакетів (Advertising Chains). Core 5.4 додав режим PAwR (Periodic Advertising with Responses), де сканер може відповідати на періодичну рекламу, що відкриває шлях для масового двонапрямленого обміну даними з сотнями пристроїв одночасно (наприклад, поличні цінники в супермаркетах). У лістингу 6.1 наведено приклад налаштування рекламних даних на ESP32-C3 за допомогою ESP-IDF та Bluedroid-стеку, де встановлюються прапорці, локальна назва та UUID сервісу.

Лістинг 6.1 – Налаштування рекламного пакета BLE на ESP32-C3 (ESP-IDF, Bluedroid)

```
// Відповідно до Supplement to the Bluetooth Core Specification
#include "esp_bt.h"
#include "esp_gap_ble_api.h"
// UUID Battery Service (0x180F)
static uint8_t service_uuid[16] = { /* UUID 0x180F */ };
// ---- Налаштування рекламних даних ----
/**
 * @brief Структура рекламних даних
 * @note  set_scan_rsp = false – конфігурація рекламного пакета
(не scan response)
* include name = true – додавати ім'я пристрою
* include txpower = false – не додавати потужність передавача
* min interval, max interval – інтервали підключення(0x20,0x40)
 * appearance = 0 – зовнішній вигляд (не вказано)
 * service uuid = 0x180F – Battery Service */
esp_ble_adv_data_t ble_adv_data = {
   .set_scan_rsp = false, // Рекламний пакет
   .include_name = true, // Включити ім'я пристрою
   .include_txpower = false, // Не включати Tx потужність
   .min_interval = 0x20, // Мін. інтервал підключення
   .max_interval = 0x40, // Макс. інтервал підключення
   .appearance = 0, // Зовнішній вигляд (не вказано)
   .manufacturer_len = 0, // Немає даних виробника
   .service_uuid_len = sizeof(service_uuid), // Довжина UUID
   .p_service_uuid = service_uuid, // Вказівник на UUID (0x180F)
   .flag = (ESP_BLE_ADV_FLAG_GEN_DISC | // Загальне виявлення
     ESP_BLE_ADV_FLAG_BREDR_NOT_SPT)//Без Classic BT
};
```

```
// ---- Застосування конфігурації ----
void app_main(void) {
    // ... ініціалізація Bluetooth ...
    // Налаштування рекламних даних
    esp_ble_gap_config_adv_data(&ble_adv_data);
}
```

Рекламний процес тісно пов'язаний зі скануванням. Сканер (Scanner) прослуховує рекламні канали, і при виявленні пакета може або запитати Scan Response, або ініціювати з'єднання, надіславши CONNECT_IND. Часові параметри реклами — інтервал реклами (від 20 мс до 10,24 с) та затримка (advDelay, псевдовипадкова добавка до 10 мс для уникнення постійних колізій) — визначають баланс між швидкістю виявлення та енергоспоживанням. Після встановлення з'єднання обмін переходить на канали даних, де використовується механізм hopping за псевдовипадковою послідовністю, узгодженою під час з'єднання.

Засвоєння структури стеку, GATT-моделі даних та механізмів рекламування є фундаментом для розробки будь-якого BLE-пристрою. Наступний підрозділ, присвячений розширеним функціям (Audio LE, Channel Sounding), покаже, як сучасні специфікації розширюють можливості BLE далеко за межі простої передавання даних, роблячи його платформою для позиціонування, аудіо та масової комунікації.

6.1.2. Розширені функції: Audio LE, Channel Sounding для точного позиціювання (~10 см)

Bluetooth Low Energy, замислений спочатку виключно як інтерфейс для передавання невеликих пакетів даних із низьким енергоспоживанням, у версіях Core 5.2–5.4 та 6.0 зазнав фундаментального розширення функціональності, яке вивело його далеко за межі початкової концепції. Двома найвагомішими інноваціями, що знаменують перехід BLE в нові прикладні сфери, стали LE Audio (Low Energy Audio), який вперше дозволив передавати синхронні аудіопотоки через низькоенергетичний радіотракт, та Channel Sounding, який уперше забезпечив точне вимірювання відстані між пристроями BLE з точністю до 10 сантиметрів без потреби в додатковому обладнанні. Ці розширення суттєво змінюють уявлення про BLE як про «простий протокол для фітнес-браслетів» і відкривають шлях до його застосування в системах об'ємного звуку, слухових апаратах, безключовому доступі до автомобілів, цифрових ключах, навігації всередині приміщень

та промислових системах позиціонування. Обидві функції детально специфіковані в документах Bluetooth Core Specification 5.4 (2023) та Core Specification 6.0 (2024), на які спирається подальший виклад.

LE Audio являє собою не окремий протокол, а цілісний набір специфікацій, що охоплює новий аудіокодек, транспортний рівень для синхронних потоків і архітектуру для багатопотокових і широкомовних аудіо застосувань. Ключовим елементом є кодек LC3 (Low Complexity Communications Codec), стандартизований Bluetooth SIG як обов'язковий для LE Audio. LC3 забезпечує значно вищу якість звуку порівняно з класичним SBC (Subband Codec) при тих самих або навіть нижчих бітрейтах: згідно з технічним звітом Bluetooth SIG, LC3 на бітрейті 160 кбіт/с досягає суб'єктивної оцінки якості, еквівалентної SBC на 345 кбіт/с. Це дозволяє передавати високоякісний стереозвук через низькоенергетичний канал BLE зі значно меншим енергоспоживанням, що критично для слухових апаратів і справжніх бездротових навушників (True Wireless Stereo, TWS). Транспортним механізмом для аудіопотоків слугують ізохронні канали (Isochronous Channels), введені в Core 5.2 і розширені в 5.4. Ізохронний канал гарантує синхронізоване в часі передавання пакетів із фіксованим інтервалом (ISO Interval), що усуває джиттер, неприйнятний для аудіо. Розрізняють Connected Isochronous Stream (CIS) для двоточкового з'єднання (наприклад, лівий/правий навушник до смартфону) та Broadcast Isochronous Stream (BIS) для широкомовної передачі на необмежену кількість приймачів без попереднього сполучення.

На основі BIS будується революційна функція Auracast (раніше відома як Audio Sharing), яка перетворює BLE-передавач на мініатюрну радіостанцію. Один пристрій-джерело (наприклад, смартфон, телевізор, публічна інформаційна панель) транслює аудіопотік, який можуть одночасно приймати десятки або сотні пристроїв-приймачів (навушників, слухових апаратів) у радіусі дії. При цьому приймачі не потребують сполучення, а вибирають потік за метаданими (назва, мова). Auracast відкриває сценарії, раніше неможливі для Bluetooth: прослуховування оголошень в аеропорту безпосередньо в навушниках, перегляд фільму в кінотеатрі з отриманням аудіодоріжки власною мовою, підключення до публічних телевізорів у спортивних барах. Технічно BIS базується на періодичній рекламі (Periodic Advertising), яка оголошує розклад ізохронних потоків; клієнт синхронізується з цим розкладом і відкриває вікна прийому у визначені моменти, заощаджуючи енергію. Архітектуру LE Audio з розподілом на CIS та BIS ілюструє рисунок 6.2.

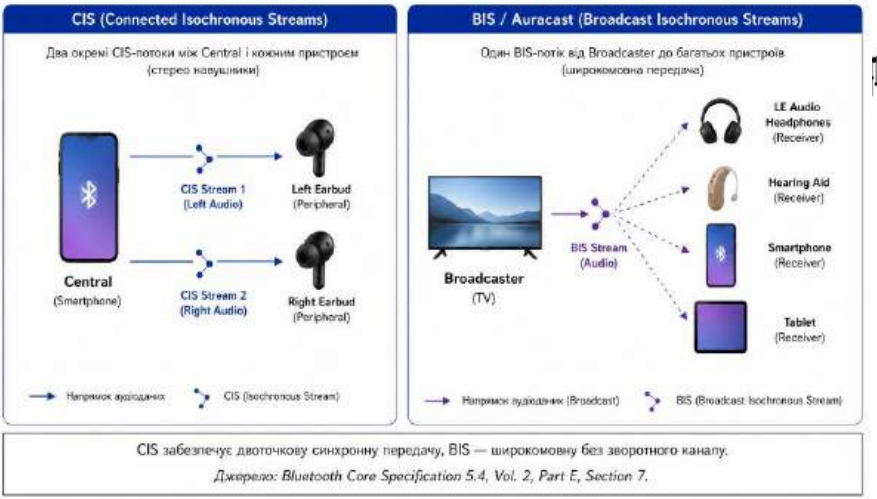


Рисунок 6.2 – Архітектура LE Audio з CIS та BIS

Другою визначальною інновацією є Channel Sounding, уперше представлений у Bluetooth Core 6.0. Якщо попередні підходи до позиціонування (RSSI, HADM/ AoA/AoD) давали точність від 1 до 10 метрів, залежну від середовища та калібрування, то Channel Sounding забезпечує точність вимірювання відстані порядку 10–50 сантиметрів у реальних умовах і до 10 сантиметрів у контрольованому середовищі. Основою методу є вимірювання часу польоту радіосигналу (ToF, Time of Flight) з використанням фазового далекоміру (Phase-Based Ranging, PBR). Два пристрої, що вимірюють відстань, виконують двонапрямлений обмін тонами на кількох частотах у діапазоні 2,4 ГГц. За різницею фаз прийнятих і переданих сигналів на різних частотах обчислюється відстань, причому багаточастотний підхід усуває неоднозначність вимірювання на одній частоті. Важливо, що Channel Sounding не залежить від антенних решіток (як AoA/AoD), що робить його доступним навіть для пристроїв із однією антеною. Криптографічний захист процедури (вбудований у протокол обміну) унеможливлює атаки підміни відстані (man-in-the-middle distance spoofing), що критично для безпеки автомобільних безключових систем. Технологія Channel Sounding розглядається консорціумом Car Connectivity Consortium (CCC) як основа для стандарту Digital Key 3.0, що дозволить відмикати автомобіль із смартфона з точним визначенням, чи власник знаходиться біля дверей.

У таблиці 6.2 зіставлено основні характеристики традиційних методів позиціонування BLE (RSSI, AoA/AoD) та Channel Sounding, що демонструє якісний стрибок у точності та безпеці.

Таблиця 6.2 – Порівняння методів позиціонування BLE

Метод	Точність (типова)	Залежність від середовища	Потреба в антенній решітці	Криптографічний захист	Сумісність із Core
RSSI (за рівнем сигналу)	3–10 м	сильна (завади, багатопроменевість, тіло)	ні	ні	4.0+
AoA/AoD (Angle of Arrival / Departure)	0,5–2 м	середня (багатопроменевість)	так (мінімум 2 антени)	ні	5.1+
Channel Sounding (PBR)	0,1–0,5 м (до сантиметрів)	слабка (багаточастотна компенсація)	ні (може працювати з однією антеною)	так (захист від спуфінгу)	6.0+

Практична реалізація LE Audio та Channel Sounding потребує підтримки на рівні апаратного забезпечення та стеку. Для LE Audio провідні виробники SoC, зокрема Nordic Semiconductor (nRF5340 Audio DK), Qualcomm (QCC514x) та Espressif (ESP32-S3 із зовнішнім кодеком), надають готові SDK із підтримкою ізохронних каналів та кодека LC3. Для Channel Sounding перші комерційно доступні мікросхеми з'явилися у 2025 році (наприклад, nRF54L15 від Nordic, що має апаратний прискорювач PBR). У лістингу 6.2 наведено гіпотетичний код налаштування ізохронного CIS-потоку на nRF5340 з використанням nRF Connect SDK (Zephyr), що демонструє ключові параметри — інтервал ISO, максимальний розмір SDU, затримку транспорту та тип каналу.

Лістинг 6.2 – Налаштування Connected Isochronous Stream (CIS) на nRF5340 (Zephyr, nRF Connect SDK)

```
// Відповідно до Zephyr Bluetooth ISO API та nRF Connect SDK
#include <zephyr/bluetooth/iso.h>
// ---- Параметри якості обслуговування (QoS) ----
/** * @brief Структура QoS для CIS-каналу
 * @note  sdu = 100 байт – максимальний розмір SDU
 * framing = BT_ISO_FRAMING_UNFRAMED – нефреймований режим
 * interval= 10000 мкс (10 мс) – інтервал ISO
 * latency = 20 мс – допустима затримка
 * rtn = 2 – кількість повторних передач */
```

```
static struct bt_iso_chan_io_qos tx_qos = {
    .sdu = 100,  // 100 байт
    .framing = BT_ISO_FRAMING_UNFRAMED, // Нефреймований режим
    .interval = 10000,  // 10 мс
    .latency = 20,   // 20 мс
    .rtn = 2,  // 2 повторні передачі   };
// ---- Callback-функція для отриманих даних ----
/** * @brief Обробка отриманих SDU (Service Data Units)
 * @param chan   Вказівник на ISO-канал
 * @param buf    Вказівник на отриманий буфер даних
 * @param user_data Користувацькі дані */
static void cis_data_recv(struct bt_iso_chan *chan, struct
net_buf *buf,
                          void *user_data) {
    // Обробка отриманих даних (наприклад, декодування LC3)   }
// ---- Підключення CIS-каналу ----
/** * @brief Встановлення CIS-з'єднання
 * @note  Використовується bt_iso_chan_connect() для створення
CIS */
void connect_cis(void) {
    struct bt_iso_chan chan = {
    .qos.tx = &tx_qos,      // QoS для передачі
    .qos.rx = &tx_qos,      // QoS для прийому
    .recv   = cis_data_recv, // Callback для прийому даних   };
    struct bt_iso_connect_param param = {
        .iso_chan = &chan,
        // ... ACL-з'єднання та інші параметри ...
    };
    bt_iso_chan_connect(&param, 1); // Підключення CIS
    }
```

Audio LE та Channel Sounding символізують якісну еволюцію BLE від простого каналу передавання телеметрії до повноцінної мультимедійної та просторово-вимірювальної платформи. Для розробника вбудованих систем це відкриває можливості створення інноваційних продуктів — від розумних навушників із підтримкою Auracast до систем точного позиціонування персоналу на виробництві чи безпечного доступу до автомобіля. Наступний підрозділ, присвячений енергоощадним алгоритмам та Bluetooth Mesh, покаже, як BLE масштабується на сотні вузлів у розподілених мережах із багаторічною автономністю.

6.1.3. Енергоощадні алгоритми та співставлення з Bluetooth Mesh

Фундаментальною перевагою Bluetooth LE, яка визначає його домінування в ніші автономних IoT-пристроїв, є не лише низьке енергоспоживання радіотракту в активному режимі, а й глибоко продумана система алгоритмів, що дозволяють пристрою перебувати в режимі сну понад 99,9%

часу, прокидаючись лише на кілька мілісекунд для обміну даними. Ці алгоритми є невід'ємною частиною специфікації Bluetooth Core 5.4 і реалізуються на рівні Link Layer, тісно взаємодіючи з апаратними можливостями сучасних SoC, такими як Nordic nRF54L15, ESP32-C3 або STM32WB. Водночас, коли вимоги до мережі виходять за межі простої топології «зірка», на сцену виходить Bluetooth Mesh — мережева надбудова, що дозволяє тисячам пристроїв формувати самовідновлювані коміркові мережі, зберігаючи при цьому енергоефективність, притаманну BLE. У цьому підрозділі розглядаються ключові енергоощадні механізми BLE та проводиться порівняльний аналіз із Bluetooth Mesh, що дозволить розробникові свідомо обрати архітектуру для конкретного застосування.

Основним механізмом, що забезпечує ультранизьке середнє споживання BLE, є концепція Connection Events — коротких синхронізованих вікон, під час яких відбувається обмін пакетами між майстром (Central) і веденим (Peripheral). Поза цими вікнами обидва пристрої можуть вимикати радіотракт і переходити в режим глибокого сну. Інтервал з'єднання (Connection Interval) може бути налаштований у діапазоні від 7,5 мс до 4 секунд. Для датчика, що передає покази раз на хвилину, інтервал 1 секунда є абсолютно прийнятним, і при тривалості одного Connection Event близько 1–2 мс радіотракт активний лише 0,1–0,2% часу. Наступним рівнем оптимізації є параметр Slave Latency — кількість послідовних Connection Events, які ведений пристрій може пропустити без втрати з'єднання. Якщо датчику немає що передавати, він не прокидається для порожніх подій, тоді як майстер продовжує надсилати пакети, даючи веденому змогу повернутися в будь-який момент. Максимальне значення Slave Latency обмежується наглядовим таймаутом (Supervision Timeout), який зазвичай становить від 100 мс до 32 секунд. Правильний вибір цих параметрів дозволяє знизити середній струм споживання до одиниць мікроампер, що робить можливим живлення BLE-пристрою від батареї CR2032 протягом кількох років.

Специфікація Bluetooth Core 5.4 запровадила новий механізм Connection Subrating, який став значним кроком уперед в енергозбереженні для пристроїв із асиметричним трафіком. Традиційно Connection Interval був єдиним для обох напрямків, що створювало дилему: малий інтервал забезпечував низьку латентність, але збільшував споживання; великий — економив енергію, але погіршував чуйність. Connection Subrating дозволяє призначити швидкий під-інтервал (sub-interval) для критичних до часу даних і повільний основний інтервал для фонового обміну. Наприклад, кла-

віатура може використовувати основний інтервал 100 мс для сплячого режиму, але миттєво перемикатися на суб-інтервал 7,5 мс при натисканні клавіші, забезпечуючи суб'єктивно нульову затримку при наборі тексту і мінімальне споживання в паузах.

Окремо слід відзначити механізм періодичної реклами з відповідями PAwR (Periodic Advertising with Responses), доданий у Core 5.4, який по суті є енергоефективною альтернативою з'єднанню для масового обміну даними. У цьому режимі центральний пристрій транслює синхронізований потік рекламних пакетів, а сотні периферійних пристроїв у визначені моменти часу прокидаються, слухають пакет і, якщо їм адресовано запит, відповідають у відведеному слоті. Такий підхід ідеально підходить для систем електронних поличних цінників (ESL), де сотні дисплеїв мають оновлюватися раз на добу, але при цьому повинні миттєво реагувати на запити інвентаризації. Енергетична ефективність досягається завдяки тому, що пристрої точно знають, коли саме мають бути активними, і решту часу можуть не вмикати радіоприймач.

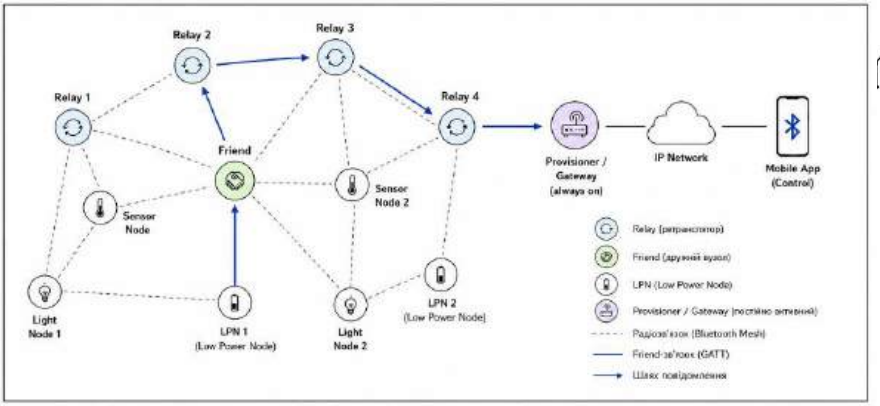
Якщо розглянуті алгоритми оптимізують споживання в межах топології «зірка» (один Central — кілька Peripheral), то Bluetooth Mesh принципово змінює топологію, масштабуючи мережу до тисяч вузлів на великих площах. Bluetooth Mesh, специфікований у документі Bluetooth Mesh Profile Specification 1.0.1 (2017, із доповненнями 2019–2023), реалізує коміркову (mesh) топологію на основі керованого лавиноподібного поширення (managed flooding). У такій мережі кожен вузол може бути одночасно і джерелом, і приймачем, і ретранслятором повідомлень. Повідомлення, надіслане одним вузлом, поширюється через усю мережу стрибками (hops) із контролем TTL, не потребуючи центрального маршрутизатора. Для забезпечення надзвичайно низького споживання батарейних вузлів у Mesh-мережі введено концепцію Low Power Node (LPN) та Friend Node. LPN — це, як правило, датчик на батареї, який більшу частину часу перебуває в глибокому сні. Він встановлює дружні стосунки з одним із сусідніх вузлів (Friend), який живиться від мережі та постійно активний. Friend зберігає всі повідомлення, адресовані LPN, і доставляє їх лише тоді, коли LPN прокидається за розкладом і надсилає запит (Friend Poll). У свою чергу, LPN може передати свої дані через Friend. Така архітектура дозволяє датчику на батареї споживати мікроампери, залишаючись повноцінним учасником великої mesh-мережі. Співставлення традиційного BLE-з'єднання та Bluetooth Mesh наведено в таблиці 6.3.

Таблиця 6.3 – Співставлення стандартного BLE (зірка) та Bluetooth Mesh

Характеристика	***BLE (з'єднання точка-точка / зірка)***	***Bluetooth Mesh***
Топологія	зірка (Central — один або кілька Peripheral)	коміркова (mesh), кожен вузол може спілкуватися з кожним
Максимальна кількість вузлів	обмежена ресурсами Central (практично до 20–30; теоретично до тисяч завдяки 24-бітній адресації)	до 32767 (теоретично); на практиці — сотні
Механізм маршрутизації	пряме з'єднання, маршрутизація відсутня	managed flooding (кероване лавиноподібне поширення)
Енергоспоживання батарейного вузла	дуже низьке (мікроампери)	дуже низьке для LPN (Low Power Node); високе для Relay/Friend
Типова Латентність доставки	одиниці–десятки мс	від ~4,5 мс (1 хоп) до сотень мс (багато хопів)
Надійність (стійкість до відмови вузла)	низька (відмова Central руйнує мережу)	висока (самовідновлення через множинні шляхи; немає єдиної точки відмови)
Сумісність зі смартфонами	повна (через GATT)	обмежена (потребує Proxy-вузла для перетворення GATT ↔ Mesh)
Типове застосування	фітнес-трекери, клавіатури, навушники	розумне освітлення, сенсорні мережі в будівлях, промисловий моніторинг
Специфікація	Bluetooth Core 5.4	Bluetooth Mesh Profile 1.0.1

Наочне уявлення про структуру Bluetooth Mesh-мережі дає рисунок 6.3, де показано розподіл ролей між вузлами та шлях повідомлення від LPN до шлюзу.

Практична реалізація Bluetooth Mesh на мікроконтролерах потребує програмного стеку, сертифікованого Bluetooth SIG. Для платформи ESP32 компанія Espressif надає стек ESP-BLE-MESH, який підтримує як Provisioner, так і звичайні вузли. Для nRF52/nRF53/nRF54 компанія Nordic Semiconductor пропонує nRF5 SDK for Mesh (застарілий, але популярний) та новий стек на основі Zephyr RTOS.



Friend-вузол буферизує повідомлення для сплячого LPN і доставляє їх за запитом.

Джерело: Bluetooth Mesh Profile Specification 1.0.1, розділ 3.4.

Рисунок 6.3 – Структура Bluetooth Mesh-мережі з LPN та Friend

У лістингу 6.3 наведено приклад конфігурації LPN-вузла на ESP32 із використанням ESP-BLE-MESH, де встановлюються параметри дружнього опитування (Friend Poll).

Лістинг 6.3 – Конфігурація Low Power Node у Bluetooth Mesh на ESP32 (ESP-BLE-MESH)

```
// Відповідно до ESP-IDF Bluetooth Mesh API та специфікації Mesh Profile
#include "esp_ble_mesh_low_power_api.h"
// ---- Реєстрація callback-ів ----
/** @brief Реєстрація callback-ів для конфігур-ого клієнта */
esp_ble_mesh_register_config_client_callback(esp_ble_mesh_config_client_cb);
// ---- Параметри LPN (Low Power Node) ----
/** * @brief Структура конфігурації LPN
 * @note  rssi_factor = 2 - коефіцієнт для RSSI при виборі Friend
 * receive_window = 150 мс - вікно прийому після запиту
 * poll_timeout = 30 с - таймаут між опитуваннями
* friend_poll_interval=1000 мс - інтервал опитування Friend  */
esp_ble_mesh_lpn_config_t lpn_config = {
.rssi_factor = 2, // Коефіцієнт RSSI для вибору Friend
.receive_window = 150, // Вікно прийому (мс)
.poll_timeout = 30, // Таймаут опитування (с)
.friend_poll_interval = 1000, // Інтервал опитування Friend(мс)
    };
// ---- Активація LPN ----
/** * @brief Активація режиму LPN після provisioning
* @note Після виклику вузол автоматично шукає Friend*/
esp_ble_mesh_lpn_enable(true);        // Увімкнення LPN
// ---- Головний цикл ----
```

```
26:void app_main(void) {
27:     // ... provisioning ...
28:   while (1) {
29:   // Запит даних у Friend (polling)
30:   esp_ble_mesh_lpn_poll();// Отримання накопичених повідомлень
 31:         vTaskDelay(pdMS_TO_TICKS(100));     }    }
```

BLE пропонує цілий спектр енергоощадних механізмів — від простого збільшення інтервалу з'єднання до складних схем із суб-інтервалами (Connection Subrating) та періодичною рекламою з відповідями (PAwR). Bluetooth Mesh доповнює цей арсенал можливістю масштабування на сотні вузлів зі збереженням ультранизького споживання для батарейних пристроїв завдяки архітектурі Low Power Node — Friend. Вибір між звичайним BLE та Bluetooth Mesh визначається передусім топологією та масштабом: для персональних пристроїв навколо смартфону ідеально підходить «зірка», тоді як для розумного будинку, промислового моніторингу або освітлення великих приміщень Mesh є безальтернативним. Наступний підрозділ, присвячений IEEE 802.15.4, продемонструє альтернативний фізичний і канальний рівень, на якому базуються Zigbee та Thread, порівнюючи їх із Bluetooth Mesh.

6.2. IEEE 802.15.4 як основа для багатьох протоколів

Стандарт IEEE 802.15.4, уперше затверджений Інститутом інженерів електротехніки та електроніки у 2003 році й оновлений у версії IEEE 802.15.4-2020, займає унікальне місце в ієрархії бездротових інтерфейсів. Він не є завершеним протоколом на кшталт Bluetooth чи Wi-Fi, а визначає лише два найнижчі рівні моделі OSI — фізичний (PHY) та канальний (MAC — Medium Access Control), — створюючи універсальний фундамент, на якому надбудовуються верхні рівні різноманітних мережевих протоколів. Саме на цьому фундаменті базуються Zigbee (з надбудовою NWK та APS від Connectivity Standards Alliance), Thread (з IP-маршрутизацією поверх 6LoWPAN), WirelessHART (промисловий стандарт IEC 62591), ISA100.11a, а також численні пропрієтарні мережі з ультранизьким енергоспоживанням. Така архітектурна гнучкість стала можливою завдяки тому, що IEEE 802.15.4 від початку проєктувався для низькошвидкісних бездротових персональних мереж (LR-WPAN) із критичними вимогами до енергоефективності, простоти реалізації та підтримки багатьох топологій.

6.2.1. Фізичний та MAC рівні; підтримка на радіотрактах CC2538, nRF52840

У цьому підрозділі детально розглядаються фізичний та MAC-рівні стандарту, а також апаратна підтримка на прикладі двох знакових радіотрактів — Texas Instruments CC2538 та Nordic Semiconductor nRF52840.

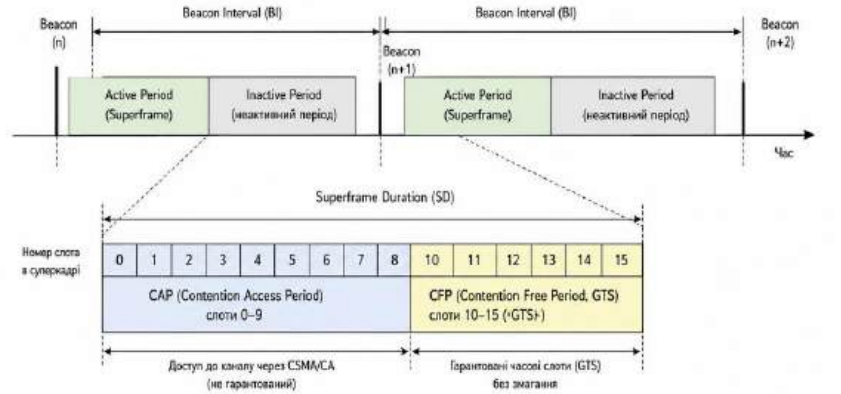
Фізичний рівень IEEE 802.15.4 визначає кілька діапазонів частот і відповідних схем модуляції, що дозволяє адаптувати пристрій до регуляторних вимог конкретного регіону світу. Найпоширенішим є глобальний діапазон 2,400–2,4835 ГГц, у якому стандарт виділяє 16 каналів із кроком 5 МГц. Швидкість передавання в цьому діапазоні становить 250 кбіт/с із використанням модуляції O-QPSK (Offset Quadrature Phase Shift Keying) із DSSS (Direct Sequence Spread Spectrum). DSSS забезпечує підвищену завадостійкість завдяки розширенню спектра, перетворюючи кожні 4 біти даних на 32-чіпову послідовність. Для регіональних діапазонів нижче 1 ГГц стандарт пропонує кілька варіантів: 868,0–868,6 МГц (Європа, 1 канал, до 100 кбіт/с), 902–928 МГц (Північна Америка, 10 каналів, до 250 кбіт/с), а також численні субгігагерцові смуги в Азії.

Оновлення IEEE 802.15.4g-2012 та IEEE 802.15.4-2020 додали фізичні рівні SUN (Smart Utility Network) із підтримкою модуляцій FSK, O-QPSK та OFDM для збільшення дальності та пропускної здатності, що особливо важливо для розумних енергомереж. Ключові параметри фізичного рівня для найпоширеніших конфігурацій зведено в таблиці 6.4.

Таблиця 6.4 – Параметри фізичного рівня IEEE 802.15.4 (основні конфігурації)

Діапазон	Частоти (МГц)	Кількість каналів	Крок каналів (МГц)	Модуляція	Чіпова швидкість (кчіп/с)	Бітова швидкість (кбіт/с)
868 МГц (EU)	868,0 – 868,6	1	—	BPSK	300	20
868 МГц (EU, alt)	868,0 – 868,6	1	—	O-QPSK	400	100
915 МГц (US)	902 – 928	10	2	O-QPSK	1000	250
2,4 ГГц (Global)	2400 – 2483,5	16	5	O-QPSK	2000	250

MAC-рівень IEEE 802.15.4 є значно складнішим за фізичний і саме він реалізує функції, що визначають енергоефективність, надійність та гнучкість топології. Стандарт визначає два режими роботи MAC: не-маяковий (non-beacon) та маяковий (beacon-enabled). У не-маяковому режимі вузли використовують протокол CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance) без часової синхронізації. Перед передаванням вузол виконує процедуру CCA (Clear Channel Assessment) для перевірки зайнятості каналу; якщо канал вільний, починається передача, інакше вузол відступає на випадковий час (backoff). Такий підхід простий у реалізації, але не забезпечує гарантованої латентності, оскільки затримка залежить від навантаження мережі. У маяковому режимі координатор мережі періодично розсилає кадри-маяки (beacons), які синхронізують годинники всіх вузлів і оголошують структуру суперкадру (superframe). Суперкадр обмежений маяками і поділяється на активний період (Superframe Duration, SD) та опціональний неактивний період. Активний період складається з 16 рівних слотів, які можуть бути розділені на CAP (Contention Access Period) — для випадкового доступу через CSMA-CA — та CFP (Contention-Free Period), що містить гарантовані часові слоти (GTS — Guaranteed Time Slots) для критичного до часу трафіку. Структуру суперкадру IEEE 802.15.4 наведено на рисунку 6.4. Саме маяковий режим із суперкадром є основою для роботи Zigbee та Thread, забезпечуючи передбачувану латентність і можливість синхронізованого сну вузлів для економії енергії.



Маяковий режим дозволяє синхронізувати вузли та виділяти гарантовані слоти для реального часу.
Джерело: IEEE 802.15.4-2020, Figure 6-1.

Рисунок 6.4 – Структура суперкадру IEEE 802.15.4 у маяковому режимі

MAC-рівень також відповідає за адресацію, яка може бути короткою (16 біт) або розширеною (64 біти, EUI-64), підтвердження доставки (ACK) із можливістю повторних передач (до 3 спроб), а також за безпеку на канальному рівні через AES-CCM-128 із забезпеченням конфіденційності, цілісності та автентичності. Саме вбудована криптографія на MAC-рівні є однією з ключових переваг IEEE 802.15.4 перед простими пропрієтарними радіопротоколами.

Апаратна підтримка IEEE 802.15.4 реалізується у вигляді радіотрактів (трансиверів), які можуть бути або окремими мікросхемами, або частиною системи на кристалі (SoC). Історично знаковим представником першого покоління є CC2538 від Texas Instruments — потужна SoC на ядрі ARM Cortex-M3 із вбудованим радіотрактом IEEE 802.15.4. CC2538, випущений у 2013 році, став галузевим стандартом для Zigbee-мереж завдяки підтримці повного стеку Z-Stack від Texas Instruments. Його радіочастина підтримує діапазон 2,4 ГГц із чутливістю -97 дБм, вихідною потужністю до +7 дБм (без підсилювача) та апаратним прискорювачем AES-128, що відповідає вимогам IEEE 802.15.4-2006. Архітектура CC2538 базується на розділенні між хостовим процесором (Cortex-M3) і радіомодулем, які об'єднані внутрішньою шиною, однак усі функції MAC (CSMA-CA, автоматичне підтвердження, фільтрація адрес) реалізовані апаратно, що значно розвантажує ядро.

Другим яскравим представником є nRF52840 від Nordic Semiconductor — багатопротокольний SoC на ядрі ARM Cortex-M4, який підтримує одночасно BLE 5.4, IEEE 802.15.4 (Thread, Zigbee), ANT та пропрієтарні протоколи. На відміну від CC2538, де підтримка 802.15.4 є статичною, nRF52840 використовує гнучку архітектуру програмованого радіо (Software Defined Radio, SDR), де параметри модуляції, частоти та формату кадру конфігуруються програмно через API-рівень SoftDevice або Zephyr RTOS. Це дозволяє одному й тому самому апаратному забезпеченню підтримувати IEEE 802.15.4 для Thread або Zigbee, а також BLE, динамічно перемикаючись між ними за допомогою механізму Timeslot API. Радіотракт nRF52840 забезпечує чутливість -103 дБм для 802.15.4 (250 кбіт/с), вихідну потужність до +8 дБм та струм споживання 6 мА в режимі прийому, що робить його одним із найенергоефективніших рішень на ринку. Порівняльні характеристики CC2538 та nRF52840 як апаратних платформ для IEEE 802.15.4 наведено в таблиці 6.5.

Таблиця 6.5 – Порівняння радіотрактів CC2538 та nRF52840 для IEEE 802.15.4

Характеристика	**CC2538 (TI)**	**nRF52840 (Nordic)**
Ядро	ARM Cortex-M3	ARM Cortex-M4F
Максимальна частота ядра	32 МГц	64 МГц
Підтримувані Протоколи (окрім 802.15.4)	немає (тільки 802.15.4/Zigbee)	BLE 5.4, ANT, Thread, Zigbee, пропрієтарні
Чутливість (802.15.4, 250 кбіт/с)	-97 дБм	-100 дБм
Максимальна вихідна потужність	+7 дБм	+8 дБм
Струм прийому (802.15.4)	20 мА (CPU idle)	6,26 мА (при 1 Мбіт/с)
Апаратний AES	так (AES-128/256)	так (AES-128)
Типовий програмний стек	Z-Stack (TI RTOS)	OpenThread, Zephyr, nRF5 SDK for Thread/Zigbee

Для розробника програмного забезпечення практичне використання IEEE 802.15.4 майже завжди відбувається не через пряме програмування регістрів радіотракту, а через стек вищого рівня. Для nRF52840 у середовищі Zephyr RTOS існує драйвер ieee802154_nrf5, який реалізує стандартний API IEEE 802.15.4, сумісний із OpenThread. У лістингу 6.4 наведено мінімальний код для ініціалізації інтерфейсу IEEE 802.15.4 на nRF52840 та створення сокету для передавання даних через OpenThread.

Лістинг 6.4 – Ініціалізація IEEE 802.15.4 та створення UDP-сокету на nRF52840 (Zephyr + OpenThread)

```
// Відповідно до Zephyr Networking API та OpenThread API
#include <zephyr/kernel.h>
#include <zephyr/net/socket.h>
#include <zephyr/net/openthread.h>
// ---- Ініціалізація OpenThread ----
/** * @brief Ініціалізація драйвера IEEE 802.15.4 та OpenThread
 * @note  ieee802154_init() ініціалізує L2-стек для заданого
інтерфейсу
 * @note  otInstanceInitSingle() створює єдиний екземпляр Open
Thread */
void network_init(void) {
    // Ініціалізація драйвера IEEE 802.15.4 (виконується
автоматично в Zephyr)
    // ...
    // Створення екземпляра OpenThread
    otInstance *instance = otInstanceInitSingle();
    // Увімкнення IPv6-інтерфейсу
    otIp6SetEnabled(instance, true);   }
// ---- Створення UDP-сокету та відправлення даних ----
```

```
/** * @brief Створення UDP-сокету та відправлення даних на
задану IPv6-адресу
 * @note  Використання BSD-сокетів у Zephyr
 * @note  OpenThread автоматично обробляє MAC-рівень 802.15.4,
CSMA-CA та 6LoWPAN-компресію */
void send_udp_data(void) {
 int sock;
 struct sockaddr_in6 dest_addr;
 char *message = "Hello from nRF52840!";
 // 1. Створення UDP-сокету (IPv6) [0†L9-L11]
 sock = socket(AF_INET6, SOCK_DGRAM, IPPROTO_UDP);
 if (sock < 0) {
     // Обробка помилки
     return;
 }
 // 2. Налаштування адреси призначення (IPv6)
 dest_addr.sin6_family = AF_INET6;
 dest_addr.sin6_port = htons(1234);
 inet_pton(AF_INET6, "ff03::1", &dest_addr.sin6_addr);
// Multicast адреса
    // 3. Відправлення даних через UDP
    int ret = sendto(sock, message, strlen(message), 0,
              (struct sockaddr *)&dest_addr, sizeof(dest_addr));
    if (ret < 0) {         // Обробка помилки     }
    // 4. Закриття сокету     close(sock);      }
```

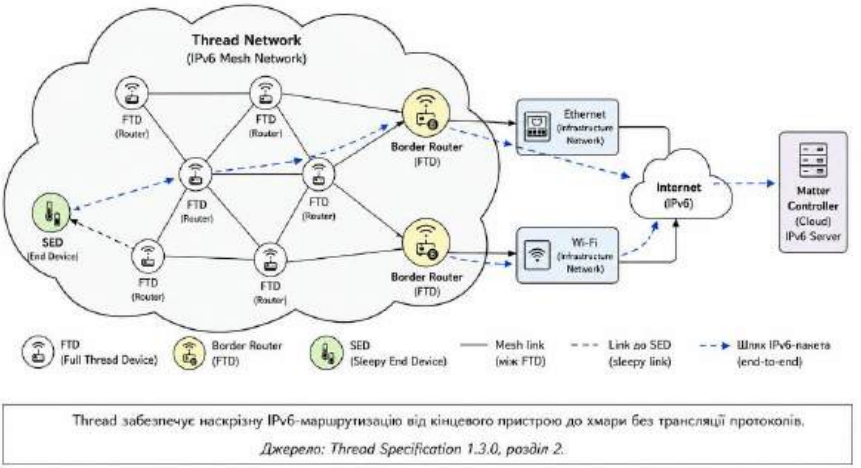
IEEE 802.15.4 є тим непомітним, але критично важливим фундаментом, на якому тримається ціла екосистема бездротових протоколів для IoT. Розуміння його фізичного та MAC-рівнів є необхідним для свідомого вибору між Zigbee, Thread або власною мережевою надбудовою, а також для налагодження проблем на нижніх рівнях, таких як колізії, втрати пакетів або недостатня чутливість. Наступні підрозділи, присвячені Thread та Zigbee, продемонструють, як різні мережеві рівні використовують цей фундамент для вирішення кардинально відмінних завдань — IP-маршрутизації в mesh-мережі та кластерної моделі для розумного дому.

6.2.2. Thread: IP-орієнтована mesh-мережа з безпекою на рівні мережі

Стандарт IEEE 802.15.4, розглянутий у попередньому підрозділі, забезпечує фізичний та канальний рівні, проте сам по собі не визначає, яким чином пакети мають знаходити шлях через мережу з десятків або сотень вузлів, як має виконуватися адресація та як забезпечується безпека зв'язку. Саме цю прогалину заповнює протокол Thread, розроблений консорціумом Thread Group (заснованим у 2014 році компаніями Google Nest, ARM, Silicon Labs, Samsung та іншими) і опублікований у вигляді відкритої спе-

цифікації Thread Specification 1.3.0 (Thread Group, 2023). Thread є низькопотужною, самовідновлюваною mesh-мережею, яка базується на IPv6 і забезпечує наскрізну IP-адресацію кожного вузла — від кінцевого датчика до хмарного сервісу — без проміжних шлюзів трансляції протоколів. Це принципово відрізняє Thread від Zigbee чи Z-Wave, де для виходу в Інтернет потрібен шлюз, що перепаковує повідомлення з пропрієтарного формату в IP. Крім того, Thread був обраний як одна з двох базових мережевих технологій (поряд із Wi-Fi) для протоколу Matter (раніше Project CHIP), що робить його одним із найперспективніших стандартів для розумного дому та комерційних IoT-рішень. Як зазначено в оглядовій статті К. А. Брента та співавторів (IEEE Internet of Things Journal, том 11, 2024), поєднання Thread із Matter створює уніфіковану IP-екосистему, що усуває фрагментацію ринку розумного дому.

Архітектура Thread-мережі спирається на чіткий розподіл ролей між вузлами. Повнофункціональний пристрій (Full Thread Device, FTD) має постійне живлення (мережеве), завжди активний радіотракт і виконує функції маршрутизатора (Router), ретранслюючи пакети через mesh-мережу. Пристрої з обмеженим функціоналом (Minimal Thread Device, MTD) є малопотужними батарейними вузлами, які не беруть участі в маршрутизації, а лише спілкуються з одним із FTD-маршрутизаторів, найчастіше через механізм синхронізованого сну (Sleepy End Device, SED). Особливу роль відіграє граничний маршрутизатор (Thread Border Router, TBR), який фізично з'єднує Thread-мережу з зовнішньою IP-інфраструктурою (Ethernet, Wi-Fi, стільниковий зв'язок) і виконує функції трансляції між 6LoWPAN-доменом усередині Thread та IPv6-мережею зовні. Важливо, що TBR не є єдиною точкою відмови: мережа Thread може мати кілька Border Router-ів, і в разі виходу з ладу одного з них трафік автоматично перенаправляється через інші. Крім того, TBR не виконує жодної трансляції прикладних протоколів — він працює виключно на мережевому рівні, маршрутизуючи IPv6-пакети. Це означає, що будь-який IP-додаток (CoAP, MQTT-SN, HTTP) може працювати наскрізно без модифікацій. Типову архітектуру Thread-мережі з Border Router та різними типами вузлів показано на рисунку 6.5.



Thread забезпечує наскрізну IPv6-маршрутизацію від кінцевого пристрою до хмари без трансляції протоколів.
Джерело: Thread Specification 1.3.0, розділ 2.

Рисунок 6.5 – Архітектура Thread-мережі з Border Router

На відміну від Zigbee, який використовує ієрархічну маршрутизацію на основі дерева (tree routing) із можливістю AODV для оптимізації, Thread застосовує повністю розподілену маршрутизацію, де кожен FTD-маршрутизатор підтримує таблицю маршрутизації до всіх інших маршрутизаторів мережі та обмінюється з ними векторами відстаней. Це забезпечує швидку конвергенцію після відмови вузла та автоматичне знаходження найкоротшого шляху. Кожен вузол Thread має кілька IPv6-адрес, із яких Mesh-Local Address (MLA) використовується для внутрішньої маршрутизації в мережі, а Global Unicast Address (GUA) — для комунікації за межами Thread-мережі через Border Router.

Безпека в Thread інтегрована на всіх рівнях і є обов'язковою, а не опціональною. На відміну від багатьох інших протоколів IoT, де безпеку часто «додають» на прикладному рівні (наприклад, через TLS), Thread реалізує безпеку на мережевому рівні за допомогою IPsec (IP Security) та IKEv2 (Internet Key Exchange версії 2) для автентифікації та обміну ключами. Кожен пристрій перед приєднанням до мережі проходить процедуру введення (Commissioning), під час якої він отримує мережевий ключ (PSKc — Pre-Shared Key for the Commissioner), захищений криптографією на еліптичних кривих (ECC). Після успішної автентифікації всі дані, що передаються через mesh, шифруються на мережевому рівні за допомогою AES-CCM-128, що гарантує конфіденційність і цілісність трафіку незалежно від використовуваного прикладного протоколу. Крім того, Thread підтримує ротацію ключів, що унеможливлює компрометацію мережі в разі

довготривалого прослуховування. Для захисту від атак на канальному рівні використовується фреймова безпека IEEE 802.15.4 (MAC-рівень AES-CCM), яка діє на кожному хопі.

Співставлення Thread з іншими mesh-протоколами, що базуються на IEEE 802.15.4, наведено в таблиці 6.6, яка узагальнює ключові архітектурні відмінності.

Таблиця 6.6 – Порівняння Thread, Zigbee та Bluetooth Mesh

Характеристика	**Thread**	**Zigbee 3.0**	**Bluetooth Mesh**
Мережевий рівень	IPv6 + 6LoWPAN	Zigbee NWK (пропрієтарний)	Bluetooth Mesh Bearer Layer (рекламний)
Адресація	IPv6 (Mesh-Local, Global)	16-бітна мережева адреса (коротка адреса NWK)	UUID (128 біт), адреси вузлів 16 біт (Unicast, Group, Virtual)
Наскрізний IP	так (без шлюзу трансляції)	Потребує Zigbee-to-IP шлюзу	потребує BLE Mesh-to-IP шлюзу
Безпека	IPsec/IKEv2 + AES-CCM на MAC-рівні	AES-CCM-128 на мережевому рівні (Trust Center)	AES-CCM на мережевому рівні (подвійний захист)
Маршрутизація	дистанційно-векторна, розподілена (Mesh Routing)	ієрархічна (tree) + AODV	managed flooding (кероване лавиноподібне поширення)
Сумісність із Matter	так (обов'язкова)	через Matter Bridge	через Matter Bridge
Типова застосовність	розумний дім, комерційні IoT-мережі	освітлення, сенсорні мережі	освітлення, сенсорні мережі

Практична реалізація Thread-вузла найчастіше спирається на відкритий стек OpenThread, що підтримується консорціумом Thread Group та компанією Google. OpenThread є компактною, переносною реалізацією всіх функцій Thread (Router, Leader, Commissioner, Border Router), доступною у вихідних кодах на GitHub і ліцензованою під BSD. Він може працювати як на bare-metal мікроконтролерах (у складі Zephyr RTOS), так і на Linux-платформах (Raspberry Pi, BeagleBone) для створення Border Router. На апаратному рівні OpenThread підтримує трансивери, сумісні з IEEE 802.15.4, зокрема nRF52840 (Nordic), CC1352 (Texas Instruments), EFR32 (Silicon Labs). У лістингу 6.5 наведено приклад ініціалізації Thread-вузла на nRF52840 із

використанням OpenThread API в середовищі Zephyr, де демонструється створення UDP-сокету та відправлення повідомлення.

Лістинг 6.5 – Створення Thread-вузла та відправлення UDP-повідомлення на nRF52840 (Zephyr + OpenThread)

```
//Платформа: nRF52840 (Zephyr + OpenThread)
//Відповідно до OpenThread API Reference
#include <openthread/instance.h>
#include <openthread/thread.h>
#include <openthread/ip6.h>
#include <openthread/udp.h>
#include <string.h>
// ---- Глобальні змінні ----
static otInstance *sInstance=NULL;//Екземпляр OpenThread
static otUdpSocket sUdpSocket; // UDP-сокет
// ---- Callback для отримання даних ----
/** * @brief Обробник отриманих UDP-даних (відповідь) */
static void udp_receive_handler(void *aContext, otMessage
*aMessage, const otMessageInfo *aMessageInfo) {
    // Обробка отриманого повідомлення    }
// ---- Ініціалізація OpenThread ----
/** * @brief Ініціалізація OpenThread та створення вузла
 * @note  otInstanceInitSingle() створює єдиний екземпляр
OpenThread
 * @note  otIp6SetEnabled() увімкнює IPv6-інтерфейс
 * @note  otThreadSetEnabled() запускає Thread-стек  */
void thread_init(void) {
//1. Ініціалізація екземпляра OpenThread
 sInstance = otInstanceInitSingle();
 // 2. Увімкнення IPv6-інтерфейсу
 otIp6SetEnabled(sInstance, true);
 // 3. Запуск Thread-стека
 otThreadSetEnabled(sInstance, true);
 // 4. Відкриття UDP-сокету
 otUdpOpen(sInstance, &sUdpSocket, udp_receive_handler, NULL);
}
// ---- Відправлення UDP-повідомлення ----
/** * @brief Відправлення даних на Mesh-Local адресу через UDP
*@note otUdpSend() надсилає UDP-дейтаграму
* @note Після приєднання вузол автоматично отримує IPv6-адресу*/
void send_udp_message(void) {
    otMessage *message;
    otMessageInfo messageInfo;
    const char *payload = "Hello from Thread node!";
    // 1. Створення повідомлення
    message = otUdpNewMessage(sInstance, NULL);
    // 2. Додавання даних
    otMessageAppend(message, payload, strlen(payload));
  // 3. Налаштування адреси призначення (Mesh-Local)
  memset(&messageInfo, 0, sizeof(messageInfo));
  messageInfo.mPeerPort = 1234;
  // Заповнення mPeerAddr Mesh-Local IPv6-адресою
  // 4. Відправлення повідомлення
  otUdpSend(sInstance, &sUdpSocket, message, &messageInfo);   }
```

Thread є еволюційним кроком у розвитку mesh-мереж, що поєднує переваги IP-адресації (наскрізна маршрутизація, відсутність трансляції), вбудовану безпеку мережевого рівня (IPsec), самовідновлювану маршрутизацію та енергоефективність, успадковану від IEEE 802.15.4. Саме ці властивості зробили Thread одним із наріжних каменів протоколу Matter, який розглядатиметься в наступному підрозділі й який знаменує перехід до справді інтероперабельних екосистем розумного дому.

6.2.3. Zigbee 3.0 та Zigbee PRO: кластерна модель, сумісність з Matter

Zigbee є одним із найстаріших і найпоширеніших протоколів для бездротових mesh-мереж малого радіусу дії, що базуються на фізичному та канальному рівнях IEEE 802.15.4. Його специфікація підтримується альянсом Connectivity Standards Alliance (CSA, раніше Zigbee Alliance) і пройшла довгий шлях еволюції від пропрієтарних профілів Zigbee 2004/2006 через Zigbee PRO (2007) до уніфікованого стандарту Zigbee 3.0 (2016), який об'єднав раніше розрізнені профілі Zigbee Home Automation (ZHA), Zigbee Light Link (ZLL), Zigbee Smart Energy (ZSE) та інші в єдиний сумісний екосистемний простір. Якщо Thread, розглянутий у попередньому підрозділі, робить ставку на IP-адресацію та наскрізну маршрутизацію, то Zigbee пропонує власний потужний мережевий рівень із підтримкою сотень вузлів, надзвичайно низьким енергоспоживанням батарейних пристроїв та розвиненою кластерною моделлю прикладного рівня, яка забезпечує інтероперабельність пристроїв різних виробників без жодного центрального хабу. Специфікація Zigbee 3.0 (Zigbee Specification, документ 05-3474-21, CSA, 2016, з пізнішими доповненнями) є основним нормативним документом для розробників, а прийняття Zigbee як однієї з технологій, сумісних із Matter (через Zigbee Bridge), відкрило шлях до інтеграції мільярдів уже встановлених Zigbee-пристроїв у новітню уніфіковану екосистему розумного дому.

Архітектура Zigbee спирається на власний мережевий рівень (NWK), який надбудовується над MAC IEEE 802.15.4, і підрівень підтримки застосувань (APS — Application Support Sublayer), що забезпечує фільтрацію за адресами, надійну доставку з підтвердженням і фрагментацію великих пакетів. Мережа Zigbee має три типи вузлів: координатор (Zigbee Coordinator, ZC) — один на мережу, ініціює її створення, керує ключами безпеки та виступає Trust Center; маршрутизатор (Zigbee Router, ZR) — постійно активний вузол, що ретранслює пакети та дозволяє приєднання нових пристроїв; кінцевий пристрій (Zigbee End Device, ZED) — батарейний вузол, який не

бере участі в маршрутизації і більшу частину часу проводить у режимі сну, прокидаючись лише для опитування свого батьківського маршрутизатора. Маршрутизація в Zigbee PRO базується на комбінації ієрархічної маршрутизації за деревом (Tree Routing) та реактивного протоколу AODV (Ad-hoc On-demand Distance Vector), який знаходить оптимальний шлях у відповідь на запит маршрутизатора, коли tree-шлях є неефективним. Zigbee PRO також запровадив стохастичне адресування (Stochastic Addressing), частотну ажильність (Frequency Agility) для уникнення інтерференції з Wi-Fi та симетричне керування ключами. На відміну від Thread, мережа Zigbee не є нативно IP-сумісною; для виходу в Інтернет використовується шлюз (Zigbee Gateway), який транслює Zigbee-трафік у TCP/IP та навпаки. Архітектуру Zigbee-мережі з різними типами вузлів показано на рисунку 6.6.

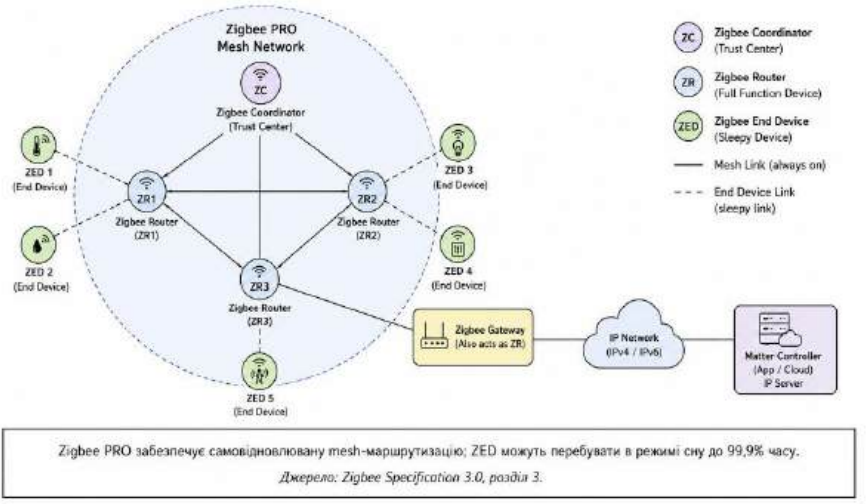


Рисунок 6.6 – Архітектура Zigbee 3.0 мережі з координатором, маршрутизаторами та кінцевими пристроями

Ключовою інновацією, що забезпечує інтероперабельність пристроїв різних виробників у Zigbee, є кластерна модель прикладного рівня, реалізована через Zigbee Cluster Library (ZCL). Кластер — це логічна одиниця функціональності, що складається з набору атрибутів (даних, що зберігаються на пристрої) та команд (дій, які можна виконати над атрибутами). Кластери згруповані за функціональним призначенням: наприклад, кластер On/Off (ID 0x0006) має атрибут OnOff (Boolean, 0 — вимкнено, 1 — увімкнено) та команди On, Off, Toggle; кластер Level Control (ID 0x0008) має

атрибут CurrentLevel (0–254) та команди MoveToLevel, Step, Stop; кластер Temperature Measurement (ID 0x0402) має атрибут MeasuredValue (у сотих градусах Цельсія). Кожен пристрій Zigbee описується набором підтримуваних кластерів, які вказуються в дескрипторах кінцевих точок (Endpoint Descriptors), і саме цей опис дозволяє контролеру (наприклад, Zigbee-координатору або шлюзу) автоматично виявити функціональність пристрою та керувати ним без жодних драйверів. У таблиці 6.7 наведено приклади стандартних кластерів Zigbee, що використовуються в Zigbee 3.0.

Таблиця 6.7 – Приклади стандартних кластерів Zigbee (ZCL)

Кластер (ID)	Призначення	Ключові атрибути	Ключові команди
On/Off (0x0006)	керування ввімкненням / вимкненням пристрою (перемикання між станами «увімкнено» та «вимкнено»)	OnOff (Boolean) – поточний стан (0x01 – увімкнено, 0x00 – вимкнено)	On (0x01), Off (0x00), Toggle (0x02)
Level Control (0x0008)	регулювання рівня (яскравості, потужності тощо)	CurrentLevel (0–254) – поточний рівень	MoveToLevel, Step, Stop
Color Control (0x0300)	керування кольором у колірному просторі CIE	CurrentHue – поточний відтінок; CurrentSaturation – поточна насиченість	MoveToHue, MoveToSaturation
Temperature Measurement (0x0402)	вимірювання температури	MeasuredValue (×0,01 °C) – виміряне значення температури у градусах Цельсія	(лише звітування; команди не визначено)
IAS Zone (0x0500)	охоронні датчики (рух, відкриття, задимлення тощо)	ZoneState – статус enroll-менту в IAS-системі; ZoneStatus (16-бітова бітова мапа) – статус кожного тригера сповіщення	Enroll, Arm
Identify (0x0003)	візуальна ідентифікація пристрою (наприклад, блиманням світлодіода)	IdentifyTime (секунди) – час, що залишився в режимі ідентифікації	Identify (0x00), TriggerEffect

Прикладний рівень Zigbee 3.0 вимагає, щоб пристрій обов’язково підтримував базовий набір функцій, зокрема Commissioning Cluster для спрощеного введення в мережу (Find&Bind, Touchlink), OTA Upgrade Cluster

для оновлення прошивки, та Alarms Cluster для діагностики. Завдяки цьому Zigbee-лампочка виробника A може бути безпосередньо керована Zigbee-вимикачем виробника B без жодної конфігурації — цей сценарій, відомий як Zigbee Interoperability, був головною метою створення Zigbee 3.0 і досягається через уніфікацію всіх профілів у єдиний Application Layer.

Сумісність Zigbee з Matter є важливим аспектом для майбутнього екосистеми розумного дому. Matter (розроблений CSA, раніше Project CHIP) є універсальним прикладним протоколом, що працює поверх IP-транспорту (Thread, Wi-Fi) і забезпечує крос-платформенну сумісність. Zigbee-пристрої не можуть безпосередньо спілкуватися з Matter-контролерами, оскільки використовують власний мережевий рівень. Однак специфікація Matter Bridge дозволяє пристроям, що не є нативно Matter-сумісними, бути представленими в Matter-екосистемі через спеціальний шлюз. Zigbee-шлюз, що реалізує Matter Bridge, транслює Zigbee-кластери та атрибути у відповідні Matter-кластери (наприклад, Zigbee On/Off Cluster транслюється в Matter On/Off Cluster), роблячи Zigbee-пристрої видимими для Matter-контролерів (Apple Home, Google Home, Amazon Alexa). Це дозволяє зберегти величезну встановлену базу Zigbee-пристроїв (понад 1 мільярд одиниць, за оцінками CSA на 2025 рік) і водночас інтегрувати їх у новітню уніфіковану екосистему.

У практичній реалізації Zigbee на мікроконтролерах найчастіше використовують стеки, надані виробниками SoC. Для nRF52840 компанія Nordic Semiconductor пропонує стек Zigbee, сумісний із Zephyr RTOS, який реалізує Zigbee PRO та Zigbee 3.0. Для CC2652 (Texas Instruments) доступний стек Z-Stack 3.x, що підтримує Zigbee 3.0 та Zigbee PRO. У лістингу 6.6 наведено приклад створення Zigbee-пристрою (лампочки) на базі nRF52840 у Zephyr, де конфігуруються кінцева точка та кластери On/Off, Level Control.

Лістинг 6.6 – Створення Zigbee-пристрою з кластерами On/Off та Level Control на nRF52840 (Zephyr + Zigbee Stack)

```
//Платформа: nRF52840 (Zephyr + ZBOSS)
//Відповідно до ZBOSS API
#include "zboss_api.h"
#include "zcl/zb_zcl_on_off.h"
#include "zcl/zb_zcl_level_control.h"
// ---- Оголошення атрибутів кластерів ----
/** @brief Список атрибутів для кластера On/Off */
ZB_ZCL_DECLARE_ON_OFF_ATTRIB_LIST(on_off_attr_list,
&on_off_attr);
/** @brief Список атрибутів для кластера Level Control */
ZB_ZCL_DECLARE_LEVEL_CONTROL_ATTRIB_LIST
(level_control_attr_list, &level_control_attr);
```

```
// ---- Список кластерів (вхідні) ----
/** @brief Список вхідних кластерів (серверна частина) */
ZB_ZCL_DECLARE_CLUSTER_LIST(
  zb_zcl_cluster_desc_t,   // Тип дескриптора кластера
  input_clusters,          // Ім'я списку
  ZB_ZCL_CLUSTER_ID_ON_OFF, //ID кластера On/Off
  ZB_ZCL_ARRAY_SIZE(on_off_attr_list, on_off_attr_list),
  on_off_attr_list,            // Список атрибутів
  ZB_ZCL_CLUSTER_ROLE_SERVER, // Роль: сервер
  NULL,                        // Дескриптор команд
  NULL,                        // Глобальні команди
  NULL,                        // Специфічні команди
  ZB_ZCL_CLUSTER_ID_LEVEL_CONTROL, // ID кластера Level Control
    ZB_ZCL_ARRAY_SIZE(level_control_attr_list,
level_control_attr_list),
    level_control_attr_list,             // Список атрибутів
    ZB_ZCL_CLUSTER_ROLE_SERVER,          // Роль: сервер
    NULL, NULL, NULL
);
// ---- Дескриптор кінцевої точки ----
/** @brief Дескриптор кінцевої точки(Simple Descriptor) */
ZB_DECLARE_SIMPLE_DESC(
    endpoint_desc,          // Ім'я дескриптора
    1,                      // Номер кінцевої точки (1-240)
    0x0104,                 // Профіль: Home Automation
    0x0101,                 // Пристрій: Dimmable Light
    0x01,                   // Версія пристрою
    0,                      // Кількість вихідних кластерів
    input_clusters,         // Вхідні кластери
    NULL                    // Вихідні кластери (немає)
);
// ----Callback для обробки команд ----
/** @brief Обробник команд ZCL(On/Off, Level Control) */
void zboss_zcl_cb(zb_bufid_t bufid) {
    // Обробка отриманих команд     }
// ---- Ініціалізація стеку та запуск ----
void zigbee_init(void) {
    // 1. Ініціалізація стеку ZBOSS
    ZB_INIT("nRF52840_Zigbee");
    // 2. Реєстрація кінцевої точки
    ZB_AF_REGISTER_ENDPOINT_DESC(&endpoint_desc);
    // 3. Реєстрація callback-функції
    ZB_ZCL_REGISTER_DEVICE_CB(zboss_zcl_cb);
    // 4. Запуск commissioning (Find&Bind)
    bdb_start_top_level_commissioning(ZB_BDB_FINDING_BINDING);
}
```

Підсумовуючи, Zigbee 3.0 та Zigbee PRO пропонують розробникові потужний і гнучкий інструментарій для побудови бездротових mesh-мереж із багаторічною автономністю вузлів, гарантованою інтероперабельністю та шляхом до інтеграції з новітнім стандартом Matter через Bridge. Вибір між Zigbee та Thread визначається архітектурними вимогами: якщо потрібна наскрізна IP-адресація та прямий зв'язок із хмарними сервісами без

шлюзу трансляції — Thread є кращим вибором; якщо ж критичними є широка підтримка вже існуючих пристроїв, мінімальне споживання батарейних вузлів та розвинена кластерна модель із сотнями стандартизованих функцій — Zigbee залишається неперевершеним. Наступний підрозділ, присвячений Matter як уніфікованому прикладному протоколу, розкриє, як цей протокол об'єднує Thread, Wi-Fi та Zigbee в єдину екосистему.

6.3. Matter (Project CHIP) – уніфікований прикладний протокол для розумного дому поверх Thread, Wi-Fi та Ethernet

Фрагментація екосистем розумного дому, що тривала понад десятиліття, призвела до ситуації, коли пристрої різних виробників — розумні лампочки, замки, термостати, датчики — були замкнені в ізольованих пропрієтарних екосистемах і не могли взаємодіяти між собою без складних хмарних інтеграцій або спеціальних хабів. Споживач був змушений обирати продукти одного бренду або миритися з необхідністю використання кількох додатків. У відповідь на цю проблему в грудні 2019 року гіганти індустрії — Apple, Google, Amazon, Samsung SmartThings та Zigbee Alliance (нині Connectivity Standards Alliance, CSA) — оголосили про створення проєкту Connected Home over IP (Project CHIP), метою якого була розробка єдиного, відкритого, вільного від роялті прикладного протоколу, що забезпечує інтероперабельність пристроїв розумного дому незалежно від виробника, платформи або голосового асистента. Результатом цієї роботи став протокол Matter, версія 1.0 якого була опублікована CSA у жовтні 2022 року, а подальші оновлення (Matter 1.1, 1.2, 1.3) розширили підтримувані категорії пристроїв та функції. Станом на 2026 рік Matter є обов'язковим для підтримки в нових пристроях розумного дому від основних виробників і розглядається як ключовий чинник подолання фрагментації ринку. Як зазначено в технічному огляді К. Тоффеля (CSA, 2023), Matter не створює нових мережевих протоколів, а працює як універсальний прикладний рівень поверх існуючих IP-транспортів — Thread, Wi-Fi та Ethernet, — забезпечуючи єдину модель даних, процедури введення в експлуатацію (commissioning) та механізми безпеки для всіх категорій пристроїв.

Архітектура Matter спирається на принцип наскрізної IP-зв'язності. Пристрої Matter спілкуються між собою через IPv6, використовуючи TCP або UDP як транспортний протокол. Це докорінно відрізняє Matter від Zigbee або Z-Wave, де потрібен шлюз трансляції протоколів у IP. У екосистемі Matter будь-який пристрій — чи то Thread-датчик на батареї, чи то Wi-Fi-розетка, чи то Ethernet-контролер — має унікальну IPv6-адресу і

може безпосередньо взаємодіяти з іншими пристроями в локальній мережі без посередництва хмари. Хмара використовується опціонально для віддаленого доступу та голосових асистентів, але локальна взаємодія не залежить від інтернет-з'єднання. Структура протокольного стеку Matter, показана на рисунку 6.7, має три основні рівні: транспортний (TCP/UDP + IPv6), рівень безпеки та повідомлень (Matter Message Layer із захистом через DTLS/TLS) та прикладний рівень (Matter Data Model, Interaction Model). Така багаторівнева архітектура дозволяє використовувати єдиний прикладний код незалежно від того, чи пристрій підключено через Wi-Fi, Thread або Ethernet.

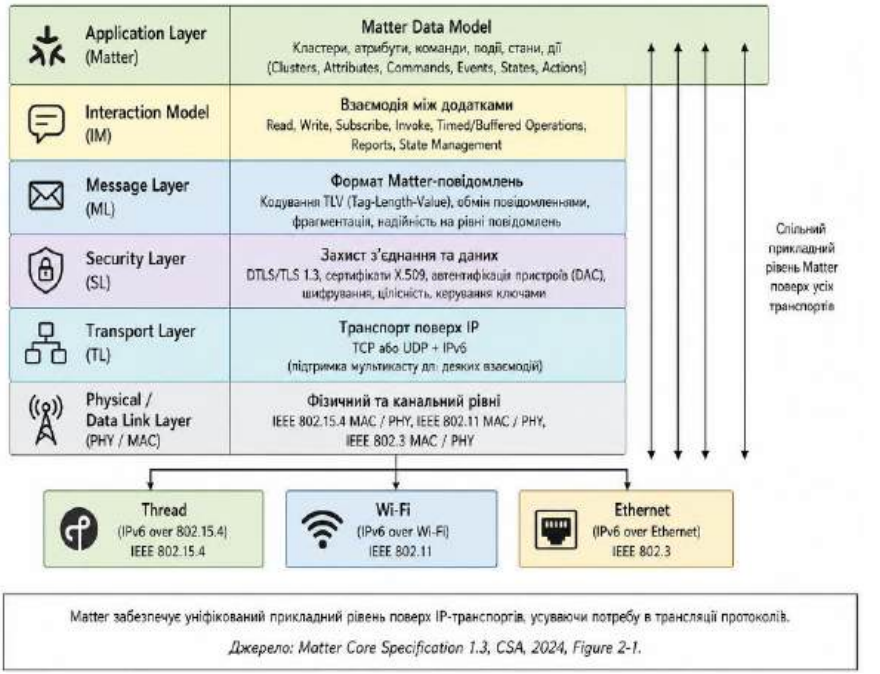


Рисунок 6.7 – Протокольний стек Matter

Ключовим елементом Matter є універсальна модель даних (Data Model), що базується на ієрархії: Вузол (Node) містить одну або кілька Кінцевих точок (Endpoints), кожна з яких реалізує певну функціональність через Кластери (Clusters). Кластер у Matter, подібно до Zigbee ZCL, є логічною одиницею, що містить атрибути (дані), команди (дії) та події (сповіщення про зміну стану). Наприклад, кластер On/Off (ID 0x0006) має атрибут OnOff (тип Boolean), команди On, Off, Toggle; кластер Level Control

(ID 0x0008) — атрибут CurrentLevel (0–254) та команди MoveToLevel, Step; кластер Temperature Measurement (ID 0x0402) — атрибут MeasuredValue. Matter визначає понад 50 стандартних кластерів, згрупованих за доменами (освітлення, HVAC, безпека, датчики тощо). Завдяки цьому будь-який Matter-сертифікований контролер (Apple Home, Google Home, Amazon Alexa, Samsung SmartThings) може автоматично розпізнати функціональність пристрою та керувати ним без встановлення драйверів. Модель взаємодії (Interaction Model) визначає чотири основні типи транзакцій: Read (читання атрибутів), Write (запис атрибутів), Invoke (виклик команд) та Subscribe (підписка на періодичні звіти про зміну атрибутів, що реалізує енергоефективне опитування без постійних запитів).

Безпека Matter є обов'язковою і багаторівневою. Кожен пристрій Matter має унікальний ідентифікатор і сертифікат пристрою (Device Attestation Certificate, DAC), що видається на заводі CSA-акредитованим центром сертифікації (Product Attestation Authority, PAA). Під час введення в експлуатацію (Commissioning) контролер перевіряє DAC пристрою через онлайн-базу Distributed Compliance Ledger (DCL) на основі блокчейну, що гарантує автентичність пристрою та його відповідність сертифікаційним вимогам. Після успішної перевірки встановлюється захищений сеанс із використанням DTLS 1.3 (для UDP) або TLS 1.3 (для TCP), із взаємною автентифікацією на основі сертифікатів. Усі дані між пристроями шифруються та підписуються, що запобігає прослуховуванню та підміні команд. Крім того, Matter підтримує концепцію Multi-Admin, яка дозволяє одному пристрою бути одночасно підконтрольним кільком екосистемам (наприклад, Apple Home та Google Home), що є унікальною перевагою порівняно з пропрієтарними протоколами.

Сумісність Matter із існуючими технологіями забезпечується через спеціальні механізми. Пристрої, що нативно підтримують IP-транспорт (Thread, Wi-Fi, Ethernet), можуть бути повноцінними Matter-вузлами без додаткового обладнання. Пристрої на базі Zigbee, Z-Wave або Bluetooth Mesh, які не є нативно IP-сумісними, можуть бути інтегровані в екосистему Matter через Matter Bridge — шлюз, який транслює непідтримуваний протокол у Matter-модель даних. Наприклад, Zigbee-лампочка через Zigbee-Matter Bridge стає видимою як стандартний Matter-пристрій із кластером On/Off. Крім того, Matter підтримує безпосереднє керування через BLE під час процедури Commissioning (для пристроїв без дисплея). Порівняння Matter із Zigbee та Thread наведено в таблиці 6.8.

Таблиця 6.8 – Порівняння Matter, Zigbee 3.0 та Thread

Характеристика	**Matter**	**Zigbee 3.0**	**Thread**
Призначення	уніфікований прикладний протокол для забезпечення інтероперабельності між різними екосистемами розумного дому	повний вертикальний стек для локального бездротового зв'язку пристроїв з низьким енергоспоживанням	IP-мережевий протокол для низькопотужних бездротових mesh-мереж
Мережевий рівень	IP (IPv6) поверх Thread, Wi-Fi або Ethernet	Zigbee NWK (пропрієтарний) поверх IEEE 802.15.4	IP (IPv6 + 6LoWPAN) поверх IEEE 802.15.4
Транспортний рівень	TCP, UDP	Zigbee APS (Application Support Sub-layer)	UDP
Прикладний рівень	Matter Data Model (кластери, атрибути)	Zigbee Cluster Library (ZCL)	не визначений (Thread забезпечує лише транспорт)
Наскрізна IP-адресація	так (кожен пристрій має IPv6-адресу)	ні (потребує шлюзу для трансляції в IP)	так (native IPv6 через 6LoWPAN)
Безпека	DAC (Distributed Compliance Ledger) + DTLS/TLS 1.3, DCL (Distributed Compliance Ledger)	AES-CCM-128 на мережевому рівні, Trust Center	IPsec/IKEv2 + AES-CCM на MAC-рівні
Інтероперабельність	гарантована між виробниками (сертифікація CSA)	у межах Zigbee-екосистеми (сертифікація Zigbee)	залежить від прикладного протоколу (наприклад, Matter)
Сумісність із голосовими асистентами	нативна (Apple, Google, Amazon)	через Zigbee-Matter Bridge	через Matter (Thread Border Router)

Практична реалізація Matter на мікроконтролерах спирається на відкритий стек Matter, що підтримується CSA та компаніями-учасниками (Apple, Google, Nordic, Silicon Labs, Espressif). Для nRF52840 та nRF5340 компанія Nordic Semiconductor надає nRF Connect SDK із інтеграцією Matter поверх OpenThread (Thread + Matter). Для ESP32-C3/C6 компанія

Espressif пропонує ESP-Matter SDK, який дозволяє створювати Matter-пристрої поверх Wi-Fi. У лістингу 6.7 наведено приклад конфігурації Matter-пристрою з кластером On/Off на ESP32-C6 за допомогою ESP-Matter SDK.

Лістинг 6.7 – Створення Matter-пристрою (лампочки) з кластером On/Off на ESP32-C6 (ESP-Matter SDK)

```
// Платформа: ESP32-C6 (ESP-Matter SDK)
// Відповідно до ESP-Matter Programming Guide
#include "esp_matter.h"
#include "esp_matter_endpoint.h"
using namespace esp_matter;
// ---- Callback для обробки команд On/Off ----
/** * @brief Обробник змін атрибута On/Off (вмикання/вимикання лампочки)
 * @note Виклик. при отриманні команди від Matter-контролера */
static esp_err_t on_off_callback(attribute::
callback_type_t type, cluster_t *cluster,
          attribute_t *attribute, esp_matter_attr_val_t *val) {
    if (type == attribute::PRE_UPDATE) {
        bool state = val->val.b;  // Отримання нового стану (true/false)
      // Керування GPIO для вмикання/вимикання лампочки
      // gpio_set_level(LED_GPIO, state);
      // ESP_LOGI(TAG, "Light turned %s", state ? "ON" : "OFF");
    }    return ESP_OK;    }
// ---- Створення вузла та кінцевої точки ----
/** * @brief Створення Matter-вузла з кінцевою точкою лампочки (On/Off)
 * @note  node::create() створює кореневий вузол
 * @note  on_off::create() додає кластер On/Off до кінцевої точки  */
void matter_init(void) {
    // 1. Створення конфігурації вузла
    node::config_t node_config;
    node_t *node = node::create(&node_config, on_off_callback, nullptr);
    if (!node) {        // Обробка помилки        return;    }
    // 2. Створення кінцевої точки (Endpoint) для лампочки
    on_off::config_t on_off_config;
    on_off_config.on_off = true;  // Початковий стан: увімкнено
    endpoint_t    *endpoint    =    on_off_light::create(node, &on_off_config, ENDPOINT_FLAG_NONE, nullptr);
    if (!endpoint) {  // Обробка помилки     return;    }
    // Додаткові кластери (наприклад, Level Control для регулювання яскравості)
    // cluster_t *cluster = on_off::create(endpoint, &on_off_config, CLUSTER_FLAG_SERVER)
}
// ---- Запуск Matter стеку та Commissioning ----
/** * @brief Ініціалізація стеку та запуск Commissioning
 * @note  esp_matter::start() ініціалізує Matter-стек і починає рекламу
 * @note  Після комісіонування пристрій можна додати до Apple Home / Google Home через QR-код  */
```

```
48:void matter_start(void) {
49:     // 1. Запуск Matter-стека
50:     esp_err_t err = esp_matter::start(nullptr);
51:     if (err != ESP_OK) {    // Обробка помилки      return;    }
52:     // 2. (Опціонально) Налаштування Commissioning
53:     // Пристрій автоматично стає доступним для комісіонування
 54:    // QR-код генерується та виводиться в консоль
 55:    }
```

Matter є не просто ще одним протоколом, а архітектурним рішенням, що об'єднує розрізнені бездротові технології під єдиним прикладним рівнем. Він звільняє розробника від необхідності створювати окремі версії продукту для кожної екосистеми та гарантує споживачеві, що пристрій із логотипом Matter працюватиме з будь-яким контролером. Наступний підрозділ, присвячений UWB, розгляне надширокосмугову технологію, яка виводить точність позиціонування на якісно новий рівень, доповнюючи можливості розглянутих мережевих протоколів.

6.4. UWB (Ultra-Wideband) IEEE 802.15.4z: точне визначення відстані (до 5 см), застосування в безключовому доступі автомобілів

Технологія надширокосмугового зв'язку Ultra-Wideband (UWB), стандартизована в поправці IEEE 802.15.4z-2020 до базового стандарту IEEE 802.15.4, посідає унікальне місце серед бездротових інтерфейсів малої дальності завдяки своїй здатності вимірювати відстань між пристроями з безпрецедентною для масового сегменту точністю — до 5 сантиметрів у реальних умовах експлуатації. На відміну від вузькосмугових технологій, таких як BLE або Wi-Fi, які модулюють неперервну несучу, UWB передає дані за допомогою надзвичайно коротких імпульсів тривалістю близько 2 наносекунд, займаючи смугу частот не менше 500 МГц у діапазонах від 3,1 до 10,6 ГГц. Саме ця імпульсна природа в поєднанні з широкою смугою забезпечує високу часову роздільну здатність приймача, необхідну для точного вимірювання часу польоту (Time of Flight, ToF) радіосигналу між двома пристроями. Крім того, коротка тривалість імпульсів дозволяє легко відокремлювати прямий сигнал від відбитих копій (багатопроменеве поширення), що робить UWB надзвичайно стійким до завмирань і спотворень у складних середовищах — в автомобільних салонах, промислових цехах, складських приміщеннях із металевими конструкціями. Згідно з дослідженням, опублікованим у журналі IEEE Transactions on Instrumentation and Measurement (том 72, 2023), UWB-системи демонструють середню похибку

вимірювання відстані менше 10 см у 95% випадків навіть за наявності значних багатопроменевих завад, що недосяжно для жодної іншої технології порівнянної вартості.

Фізичний рівень IEEE 802.15.4z визначає два основних типи передавачів: HRP (High Rate Pulse) для швидкостей передавання даних від 110 кбіт/с до 27 Мбіт/с та LRP (Low Rate Pulse) для низькошвидкісних, але енергоефективних застосувань із бітрейтами до 1 Мбіт/с. Обидва типи підтримують режими точного вимірювання відстані (ranging), однак саме HRP-режим забезпечує найвищу точність завдяки ширшій смузі каналу. Для ranging найчастіше використовують канал 5 (центр 6,4896 ГГц, смуга 499,2 МГц) або канал 9 (центр 7,9872 ГГц), які є глобально доступними без ліцензування. Центральним механізмом визначення відстані є двостороннє вимірювання часу обігу (Two-Way Ranging, TWR), за якого пристрій-ініціатор надсилає запит, пристрій-відповідач після фіксованої або виміряної затримки відповідає, а ініціатор фіксує час приходу відповіді. Відстань обчислюється за формулою (6.1):

$$d = c \times {}^{t_{prop}}/_{2} \ , \tag{6.1}$$

де c — швидкість світла (приблизно $3 \cdot 10^8$ м/с), t_{prop} — час поширення сигналу в один бік, який визначається як $(t_{round} - t_{reply})$ / 2 для базового одностороннього TWR (SS-TWR). Проте через розбіжність тактових генераторів двох пристроїв SS-TWR схильне до похибок у кілька наносекунд, що на дистанції 10 м може давати помилку до 1 м. Тому на практиці застосовують симетричне двостороннє TWR (DS-TWR), яке компенсує дрейф генераторів шляхом додаткового обміну повідомленнями і обчислює t_{prop} через розширену формулу, що враховує чотири виміряні інтервали. Висока часова роздільна здатність UWB-приймача (до 15 пс) досягається завдяки широкій смузі сигналу: чим ширша смуга, тим вужчий кореляційний пік, і тим точніше можна визначити момент приходу імпульсу. На рисунку 6.8 наведено типову часову діаграму сеансу DS-TWR між двома UWB-вузлами з позначенням вимірюваних інтервалів.

Ключовою інновацією IEEE 802.15.4z, що вивела UWB на рівень, придатний для захищених фінансових та автомобільних транзакцій, стало запровадження захищеної структури ranging — Scrambled Timestamp Sequence (STS).

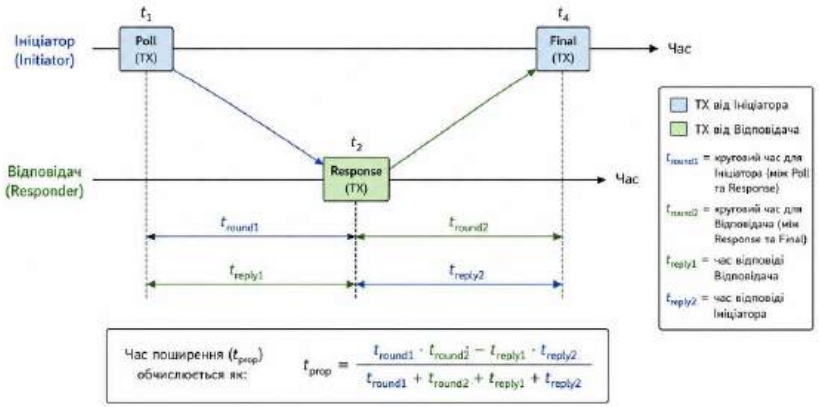


Рисунок 6.8 – Часова діаграма двостороннього симетричного ranging (DS-TWR) за IEEE 802.15.4z

У режимі STS у повідомлення ranging вставляється криптографічно захищена послідовність, яка унеможливлює підробку або зсув міток часу зловмисником. Це принципово важливо для протидії relay-атакам, коли сигнал перехоплюється, підсилюється та ретранслюється з метою обману системи доступу. Оскільки UWB вимірює фізичний час польоту, який неможливо зменшити (швидкість світла є абсолютною верхньою межею), ретрансльований сигнал завжди матиме більшу затримку, ніж прямий, і система виявить атаку. Саме ця властивість зробила UWB безальтернативним вибором для наступного покоління автомобільних цифрових ключів — Car Connectivity Consortium (CCC) Digital Key Release 3.0, який використовує UWB як основний засіб точного позиціонування смартфона відносно автомобіля. Система здатна визначити, чи знаходиться власник біля дверей, і відімкнути лише відповідну сторону, запобігаючи relay-атакам, що були можливі з попередніми системами на основі BLE RSSI.

Апаратна підтримка UWB забезпечується спеціалізованими мікросхемами, лідерами серед яких є Qorvo (яка поглинула піонера UWB — компанію Decawave) із серією DW3000 (DW3110, DW3120), NXP Semiconductors із лінійкою Trimension (NCJ29D5 для автомобільного сегменту, SR040/SR150 для IoT та смартфонів), а також Apple із власним процесором U1/U2, інтегрованим у iPhone 11 і новіші моделі. Для вбудованих розробників типовим рішенням є модуль DWM3000 (Qorvo), який підключається до мікроконтролера через SPI і підтримує всі режими ranging

згідно з IEEE 802.15.4z. У лістингу 6.8 наведено фрагмент коду ініціалізації ranging на платформі STM32 із використанням бібліотеки Decawave Driver.

Лістинг 6.8 – Ініціалізація UWB ranging на DWM3000 (STM32, Qorvo Decawave Driver)

```
// Відповідно до DWM3000 API Software and API Guide (Qorvo)
#include "dwt_uwb_driver.h"
#include "deca_device_api.h"
// ---- Глобальні змінні ----
static dwt_config_t config = {
    .chan = 5,          // Канал 5 (6.5 ГГц)
    .prf = DWT_PRF_64M,      // PRF 64 МГц
    .txPreambLength = DWT_PLEN_128, // Довжина преамбули 128 символів
    .dataRate = DWT_BR_6M8, // Швидкість даних 6.8 Мбіт/с
    // ... інші параметри (код преамбули, PAC тощо) ...
};
// ---- Ініціаліалізація UWB ----
/** * @brief Ініціалізація DWM3000 для ranging
 * @note  dwt_initialise(DWT_LOADUCODE) завантажує LDE мікрокод
 * @note  dwt_configure() застосовує параметри каналу
 * @note  Попередньо необхідно налаштувати антену та затримки TX/RX через
 *     dwt settxantennadelay() та dwt setrxantennadelay()  */
void uwb_init(void) {
    // 1. Завантаження LDE мікрокоду (необхідно для ranging)
    if (dwt_initialise(DWT_LOADUCODE) != DWT_SUCCESS) {
      // Обробка помилки        return;    }
    // 2. Налаштування затримок антени (калібрувальні значення)
    dwt_settxantennadelay(TX_ANT_DLY);
    dwt_setrxantennadelay(RX_ANT_DLY);
    // 3. Конфігурація каналу (Channel 5, PRF 64 МГц, преамбула 128, 6.8 Мбіт/с)
    dwt_configure(&config);
}
// ---- Запуск передачі (ranging) ----
/** * @brief Ініціювання передачі для ranging
 * @note dwt_starttx(DWT_START_TX_IMMEDIATE)-негайна передача*/
void uwb_start_tx(void) {
    // Підготовка буфера з даними ranging
    // ...
    // Запуск передачі
    dwt_starttx(DWT_START_TX_IMMEDIATE);    // Негайна передача
}
// ---- Callback для обробки прийому ----
/** * @brief Обробник подій прийому (RX) для ranging
 * @param  event  Тип події (RX OK, RX ERROR, RX TIMEOUT тощо)
 * @note  Реєструється через dwt_setcallbacks() */
void dwt_rxcallback(uint32_t event) {
    if (event & DWT_RX_GOOD) {
        // Обробка отриманого ranging-пакета
    }
}
```

Порівняння UWB з альтернативними технологіями точного позиціонування, зокрема з BLE Channel Sounding (розглянутим у підрозділі 6.1.2), наведено в таблиці 6.9. Хоча BLE Channel Sounding забезпечує зіставну точність у сприятливих умовах, UWB демонструє кращу стабільність у складних багатопроменевих середовищах, вищий рівень фізичної безпеки та меншу чутливість до інтерференції від Wi-Fi.

Таблиця 6.9 – Порівняння UWB (IEEE 802.15.4z) та BLE Channel Sounding (Bluetooth Core 6.0)

Характеристика	**UWB**	**BLE Channel Sounding**
Типова точність	< 10 см (до 5–10 см)	від 10–50 см до 1 м
Стійкістьдо багатопроменевості	дуже висока (широка смуга, імпульсна передача)	середня (вузькосмуговий сигнал, чутливість до відбиттів)
Криптографічний захист ranging	так (STS – Scrambled Timestamp Sequence, захист від relay-атак)	так (RTT-верифікація, NADM, захист від спуфінгу та relay-атак)
Максимальна дальність ranging	до 100 м (типова), до 125 м (оптимальні умови)	до 150 м
Енергоспоживання на ranging	20–50 мА (активний режим)	5–10 мА (нижча пікова потужність)
Типове застосування	автомобільні цифрові ключі (CCC DK 3.0), безпечні платежі, RTLS	смарт-замки, позиціонування в приміщеннях, трекери
Нормативна база	IEEE 802.15.4z, CCC DK 3.0	Bluetooth Core 6.0

Узагальнюючи, UWB займає чітко окреслену нішу високоточного та захищеного бездротового позиціонування, недосяжну для інших масових технологій. Його впровадження в автомобільній індустрії через стандарт Digital Key 3.0 та присутність у флагманських смартфонах робить UWB обов'язковим для вивчення в курсі IoT, оскільки саме на перетині точного позиціонування, безпеки та міжплатформенної сумісності будуть створюватися найінноваційніші продукти наступного десятиліття. Наступна частина посібника перейде до розгляду технологій великого радіусу дії (LPWAN), які вирішують протилежну задачу — забезпечення зв'язку на кілометри з мінімальним енергоспоживанням.

РОЗДІЛ 7.
ТЕХНОЛОГІЇ ВЕЛИКОГО РАДІУСУ ДІЇ (LPWAN)

7.1. LoRa та LoRaWAN

Технологія LoRa (Long Range), розроблена компанією Cycleo SAS (з 2012 року — частина Semtech Corporation), становить фізичний рівень, що забезпечує безпрецедентний для малопотужних пристроїв баланс між дальністю зв'язку, енергоспоживанням та завадостійкістю.

7.1.1. Символьна модуляція CSS, параметри spreading factor

На відміну від традиційних схем модуляції, таких як FSK або O-QPSK, які домінують у вузькосмугових системах, LoRa використовує запатентовану техніку розширення спектра — символьну модуляцію CSS (Chirp Spread Spectrum). Саме CSS є ключем до досягнення чутливості приймача до -148 дБм, що дозволяє приймати сигнал, рівень якого значно нижчий за рівень теплового шуму. Ця властивість, поєднана з високою стійкістю до частотного зсуву (доплерівського ефекту), багатопроменевого поширення та інтерференції, робить LoRa ідеальним фізичним рівнем для мереж LoRaWAN, що розгортаються на величезних територіях — від сільськогосподарських угідь до промислових зон. Специфікація фізичного рівня LoRa детально викладена в технічній документації на трансивери сімейства Semtech SX127x (SX1276/77/78/79) та SX126x, а також у LoRaWAN Regional Parameters (LoRa Alliance, версія RP002-1.0.3, 2023). Подальший виклад спирається на ці першоджерела, а також на фундаментальні принципи CSS, описані в класичних роботах з теорії зв'язку (Skolnik, Radar Handbook, 2008).

Основою CSS-модуляції є лінійно-частотно-модульований імпульс, або чірп (chirp). У найпростішому випадку, який використовується в LoRa, частота сигналу лінійно зростає (up-chirp) або лінійно спадає (down-chirp) упродовж символьного інтервалу. Математично, комплексна обвідна базосмугового сигналу для up-chirp описується як експонента з квадратичною фазою, де миттєва частота $f(t) = f_c + \left({B}/{T_s}\right) * t$, а B — смуга сигналу, T_s — тривалість символу. Добуток $B * T_s$, відомий як коефіцієнт розширення спектра (Spreading Factor, SF), визначає, у скільки разів енергія сигналу розподіляється в частотній області порівняно з вихідною швидкістю

передавання символів. У LoRa значення SF безпосередньо пов'язане з кількістю бітів, що переносяться одним символом: $SF = \log_2(M)$, де M — кількість можливих символів (рівнів квантування початкової частоти чірпа). Таким чином, при SF=7 один символ несе 7 бітів інформації, а при SF=12 — 12 бітів. Загальна кількість символів у алфавіті становить 2^{SF}, що при SF=12 дає 4096 різних символів.

Власне модуляція в CSS полягає в циклічному зсуві початкової частоти чірпа на величину, пропорційну значенню символу. Уявімо собі базовий up-chirp, частота якого зростає від -B/2 до +B/2 упродовж T_s. Якщо ми «розріжемо» цей чірп у точці, де частота дорівнює нулю, і поміняємо отримані половинки місцями, ми отримаємо так званий «символ нуля» (symbol 0). Якщо ж ми зсунемо момент початку лінійної розгортки, то отримаємо чірп, що починається з деякої початкової частоти f_0, лінійно зростає до +B/2, потім стрибком переходить на -B/2 і продовжує зростати до f_0. Це і є модульований символ. Циклічний зсув, а отже, і значення f_0, квантується з кроком $\Delta f = {}^{B}/_{2^{SF}}$. Приймач LoRa виконує зворотну операцію: вхідний сигнал множиться на комплексно-спряжений базовий down-chirp, після чого виконується швидке перетворення Фур'є (FFT) для визначення піку, що відповідає початковій частоті символу. Саме завдяки цій обробці приймач здатен «витягувати» сигнал із шумів: енергія символу концентрується в одному FFT-біні, тоді як шум розподіляється рівномірно. Виграш в обробці (Processing Gain, G_p) виражається формулою (7.1):

$$G_p = 10 * \log_{10}(2^{SF}) * \text{дБ} \ . \qquad (7.1)$$

Для SF=7 виграш становить близько 21 дБ, для SF=12 — близько 36 дБ. Саме цей виграш забезпечує здатність LoRa-приймача працювати з сигналами, що на 20 дБ нижчі за рівень шумів. На рисунку 7.1 показано часово-частотне представлення кількох символів CSS для SF=7, де видно різні початкові частоти та циклічний характер чірпів.

Крім SF, ключовими параметрами фізичного рівня LoRa є смуга пропускання (Bandwidth, BW) та швидкість кодування (Coding Rate, CR). BW може набувати значень 125 кГц, 250 кГц або 500 кГц (а також 7,8–62,5 кГц для деяких чіпів у спеціальних режимах). Тривалість символу T_s однозначно визначається SF та BW за формулою (7.2):

$$T_s = {}^{2^{SF}}/_{BW} \ . \qquad (7.2)$$

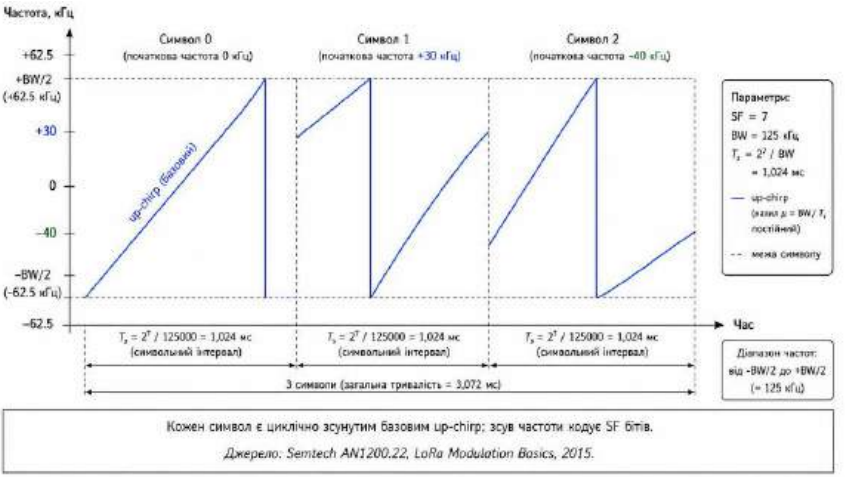


Рисунок 7.1 – Часово-частотне представлення символів CSS LoRa (SF=7, BW=125 кГц)

Збільшення BW скорочує тривалість символу, підвищуючи швидкість передавання, але зменшує чутливість приймача через ширшу смугу шуму. Збільшення SF, навпаки, експоненційно збільшує тривалість символу, підвищуючи чутливість, але знижуючи швидкість. CR визначає частку надлишкових бітів, що додаються для прямого виправлення помилок (FEC): 4/5, 4/6, 4/7 або 4/8. Номінальна бітова швидкість R_b обчислюється за формулою (7.3):

$$R_b = SF * \left({BW}/{2^{SF}}\right) * CR \quad . \tag{7.3}$$

Для прикладу, при SF=7, BW=125 кГц, CR =4/5 швидкість становить $7 \cdot (125000 / 128) \cdot (4/5) \approx 5469$ біт/с. При SF=12, BW=125 кГц, CR=4/5 швидкість падає до $12 \cdot (125000 / 4096) \cdot (4/5) \approx 293$ біт/с. Це ілюструє фундаментальний компроміс LoRa: дальність (чутливість) досягається ціною швидкості, що є прийнятним для більшості сенсорних застосувань IoT. У таблиці 7.1 наведено залежність чутливості приймача та максимальної дальності (для умов прямої видимості та типової міської забудови) від SF для трансивера Semtech SX1276, згідно з даташитом (DS_SX1276-7-8-9_V7, 2020).

Практична реалізація LoRa-модему на мікроконтролері виконується через зовнішній трансивер (SX1276, SX1262), що підключається по SPI.

Таблиця 7.1 – Залежність швидкості, чутливості та дальності від SF (SX1276, BW=125 кГц, CR=4/5)

SF	Номінальна бітова швидкість (біт/с)	Чутливість приймача (дБм)	Максимальна дальність, пряма видимість (км)	Максимальна дальність, міська забудова (км)
7	5469	-124	3,0	0,8
8	3125	-127	4,5	1,2
9	1758	-130	6,0	1,6
10	977	-133	8,0	2,2
11	537	-136	11,0	3,0
12	293	-139	15,0	4,0

Мікроконтролер записує в регістри трансивера значення SF, BW, CR, частоту каналу та потужність передавача, після чого передавання ініціюється записом даних у буфер FIFO. У лістингу 7.1 наведено приклад ініціалізації LoRa-трансивера SX1276 на STM32 із встановленням параметрів SF=12, BW=125 кГц, CR=4/5.

Лістинг 7.1 – Ініціалізація SX1276 для LoRa (SF12, BW125, CR4/5) через SPI на STM32 (HAL)

```
1:// Джерело: SX1276/77/78/79 Datasheet, Rev. 4
2:
3:#include "sx1276.h"
4:#include "spi.h"
5:// ---- Визначення регістрів та бітових масок ----
6:#define REG_OPMODE    0x01   // Режим роботи
7:#define REG_FRF_MSB   0x06   // Частота (старші біти)
8:#define REG_FRF_MID   0x07   // Частота (середні біти)
9:#define REG_FRF_LSB   0x08   // Частота (молодші біти)
10:#define REG_MODEM_CONFIG1 0x1D//Конфігурація модему 1
11:#define REG_MODEM_CONFIG2 0x1E//Конфігурація модему 2
12:// Біти RegOpMode
13:#define OPMODE_LORA     0x80   // Біт 7: LoRa режим
14:#define OPMODE_MASK     0x07   // Біти 2-0: режим роботи
15:#define OPMODE_SLEEP    0x00   // Режим сну
16:#define OPMODE_STANDBY  0x01   // Режим очікування
17:#define OPMODE_TX       0x03   // Режим передачі
18:// Біти RegModemConfig1
19:#define MC1_BW_MASK     0xF0  // Біти 7-4: смуга пропускання
20:#define MC1_BW_125KHZ   0x70  // 125 кГц (0b0111)
21:#define MC1_CR_MASK     0x0E  // Біти 3-1: швидкість кодування
22:#define MC1_CR_4_5      0x02  // CR = 4/5 (0b001)
23:// Біти RegModemConfig2
24:#define MC2_SF_MASK     0xF0  // Біти 7-4: spreading factor
25:#define MC2_SF_12       0xC0  // SF = 12 (0b1100)
26:// ------------------------
27:// @brief  Ініціалізація SX1276 у режимі LoRa з параметрами:
28://         - Частота: 868.1 МГц
```

```
//          - Смуга: 125 кГц
//          - Швидкість кодування: 4/5
//          - Spreading Factor: 12
//@note Послідовність налаштувань згідно з даташитом
// -------------------------
void SX1276_Init(void) {
//1. Перехід у режим сну (для зміни режиму на LoRa)
    uint8_t opmode = OPMODE_LORA | OPMODE_SLEEP;
    SX1276_WriteReg(REG_OPMODE, opmode);
// 2. Встановлення частоти 868.1 МГц
    //  Frf = (868.1e6 / 61.03515625) = 0xE4C000
    SX1276_WriteReg(REG_FRF_MSB, 0xE4);
    SX1276_WriteReg(REG_FRF_MID, 0xC0);
    SX1276_WriteReg(REG_FRF_LSB, 0x00);
// 3. Налаштування ModemConfig1: BW=125 кГц, CR=4/5
    uint8_t mc1 = MC1_BW_125KHZ | MC1_CR_4_5;
    SX1276_WriteReg(REG_MODEM_CONFIG1, mc1);
// 4. Налаштування ModemConfig2: SF=12
    uint8_t mc2 = MC2_SF_12;
    SX1276_WriteReg(REG_MODEM_CONFIG2, mc2);
// 5. Перехід у режим очікування (готовність до роботи)
    opmode = OPMODE_LORA | OPMODE_STANDBY;
    SX1276_WriteReg(REG_OPMODE, opmode);
}
// -------------------------
// @brief  Запуск передачі (перехід у режим TX)
// -------------------------
void SX1276_StartTx(void) {
    uint8_t opmode = OPMODE_LORA | OPMODE_TX;
    SX1276_WriteReg(REG_OPMODE, opmode);
}
```

CSS-модуляція у поєднанні з гнучкими налаштуваннями SF, BW та CR забезпечує LoRa виняткову адаптивність до вимог конкретного застосування — від швидкого обміну на коротких дистанціях (SF7, BW500) до надійного зв'язку на межі чутливості (SF12, BW125). Наступний підрозділ, присвячений архітектурі LoRaWAN, покаже, як цей фізичний рівень використовується для побудови масштабованих глобальних мереж із сотнями тисяч кінцевих пристроїв.

7.1.2. Архітектура LoRaWAN: кінцеві пристрої, шлюзи, мережевий сервер

Якщо LoRa, розглянута в попередньому підрозділі, є виключно фізичним рівнем, що визначає метод модуляції та формування радіосигналу, то LoRaWAN (Long Range Wide Area Network) являє собою мережевий протокол канального та частково мережевого рівнів, який регламентує доступ до середовища, адресацію, безпеку, адаптацію швидкості та взаємодію з магістральною інфраструктурою. Специфікація LoRaWAN публікується та

підтримується міжнародним альянсом LoRa Alliance, який об'єднує понад 500 компаній, включаючи операторів зв'язку, виробників обладнання та розробників програмного забезпечення. Актуальними на 2026 рік є версії LoRaWAN 1.0.4, 1.1.0 та 1.2.0, що описані в документах LoRaWAN Specification (LoRa Alliance, 2023), причому версія 1.0.4 залишається найпоширенішою в існуючих мережах завдяки своїй простоті. Архітектура LoRaWAN побудована за топологією «зірка зірок» (star-of-stars), де кінцеві пристрої (End Devices) безпосередньо не з'єднуються між собою, а передають дані на шлюзи (Gateways), які, своєю чергою, через стандартні IP-з'єднання (Ethernet, Wi-Fi, стільниковий зв'язок) підключаються до централізованого мережевого сервера (Network Server). Така архітектура дозволяє розгортати мережі на величезних територіях з мінімальними витратами на інфраструктуру, оскільки один шлюз може обслуговувати тисячі кінцевих пристроїв у радіусі до 15 км у сільській місцевості.

Кінцеві пристрої LoRaWAN поділяються на три класи — A, B та C, — які визначають режим роботи приймача після завершення передавання. Клас A є обов'язковим для всіх пристроїв і забезпечує максимальну енергоефективність за рахунок того, що приймач відкривається лише на два короткі вікна після кожної вихідної передачі (uplink). Перше вікно прийому (RX1) відкривається через фіксований час після завершення передавання і використовує той самий частотний канал та швидкість, що й uplink; друге вікно (RX2) відкривається через більший інтервал на спеціально виділеному каналі з фіксованою низькою швидкістю (зазвичай SF12 BW125). Поза цими вікнами пристрій може повністю вимикати радіотракт, досягаючи середнього струму споживання в одиниці мікроампер. Клас B додає до класу A синхронізовані пінг-слоти (ping slots), які відкриваються за розкладом, що транслюється шлюзом через періодичні маяки (beacons). Це дозволяє мережевому серверу ініціювати передавання до пристрою (downlink) із передбачуваною затримкою, що є компромісом між автономністю та чуйністю. Клас C призначений для пристроїв із постійним живленням, у яких приймач відкритий безперервно, крім моментів власної передачі, що забезпечує мінімальну латентність downlink, але робить клас C непридатним для батарейного живлення. У таблиці 7.2 зіставлено характеристики трьох класів за ключовими параметрами, що дозволяє розробникові свідомо обрати необхідний режим.

Шлюз LoRaWAN (Gateway) є радіочастотним концентратором, який не виконує жодної обробки протокольних даних, а лише демодулює прийняті LoRa-пакети від кінцевих пристроїв, додає до них метадані (час прийому, рівень RSSI, відношення SNR, частотний канал, SF) та тунелює

ці пакети через IP-мережу на мережевий сервер за допомогою протоколу Semtech UDP Packet Forwarder або його сучасніших альтернатив — Basics Station (LoRa Alliance, 2022).

Таблиця 7.2 – Характеристики класів кінцевих пристроїв LoRaWAN

Параметр	Клас A	Клас B	Клас C
Обов'язковість	обов'язковий для всіх кінцевих пристроїв	опціональний	опціональний
Вікна прийому після uplink	два (RX1, RX2) після кожного аплінку	два (RX1, RX2) після кожного аплінку	два (RX1, RX2) після кожного аплінку
Додаткові вікна прийому	немає	пінг-слоти за розкладом (синхронізовані за маяками)	безперервно (крім часу передачі)
Латентність downlink	найвища (лише після uplink)	середня (до кількох секунд)	мінімальна
Енергоспоживання	мінімальне (приймач активний <1% часу)	середнє	максимальне (приймач завжди увімкнений)
Типове застосування	датчики, лічильники, моніторинг довкілля	пристрої з періодичним керуванням (лічильники комунальних послуг, вуличне освітлення)	виконавчі механізми з постійним живленням

У зворотному напрямку шлюз приймає від мережевого сервера пакети для передавання (downlink) у точно визначені моменти часу (оскільки кінцевий пристрій класу A очікує відповідь у строго фіксовані вікна) і модулює їх у LoRa-сигнал. Шлюз може бути одно- або багатоканальним: сучасні чіпсети, такі як Semtech SX1302/SX1303, здатні одночасно приймати до 8 каналів з різними SF (LoRa) та один канал FSK/GFSK, що дозволяє обробляти сотні пакетів за секунду. Ключовою особливістю архітектури LoRaWAN є те, що один кінцевий пристрій може бути почутий кількома шлюзами одночасно; мережевий сервер виконує дедуплікацію таких пакетів, залишаючи лише одну копію для передачі на сервер додатків. Це під-

вищує надійність мережі та дозволяє реалізовувати грубе визначення місцеположення пристрою методом тріангуляції за різницею часу прибуття (TDoA) на основі метаданих від різних шлюзів.

Мережевий сервер (Network Server, NS) є центральним елементом LoRaWAN-інфраструктури, що реалізує всю логіку протоколу. Він виконує перевірку цілісності пакетів (MIC — Message Integrity Code), дешифрування даних на мережевому рівні, керування сесіями кінцевих пристроїв, адаптацію швидкості передавання (Adaptive Data Rate, ADR), вибір найкращого шлюзу для downlink-передачі та маршрутизацію пакетів до відповідного сервера додатків (Application Server, AS). Сервер додатків, своєю чергою, дешифрує дані на прикладному рівні, обробляє корисне навантаження та надсилає команди керування. У версії LoRaWAN 1.1.0 було додано окремий сервер приєднання (Join Server, JS), який зберігає кореневі ключі пристроїв і бере участь у процедурі активації Over-The-Air Activation (OTAA). Процедура активації може виконуватися двома способами: OTAA, за якої пристрій відправляє запит на приєднання (Join Request), що містить унікальний ідентифікатор DevEUI, і після обміну з Join Server отримує сеансові ключі (NwkSKey, AppSKey), та Activation By Personalization (ABP), за якої ключі жорстко прошиваються в пристрій на етапі виробництва. OTAA є безпечнішою та рекомендованою, оскільки дозволяє змінювати ключі та параметри сесії. Архітектуру LoRaWAN-мережі з основними компонентами показано на рисунку 7.2.

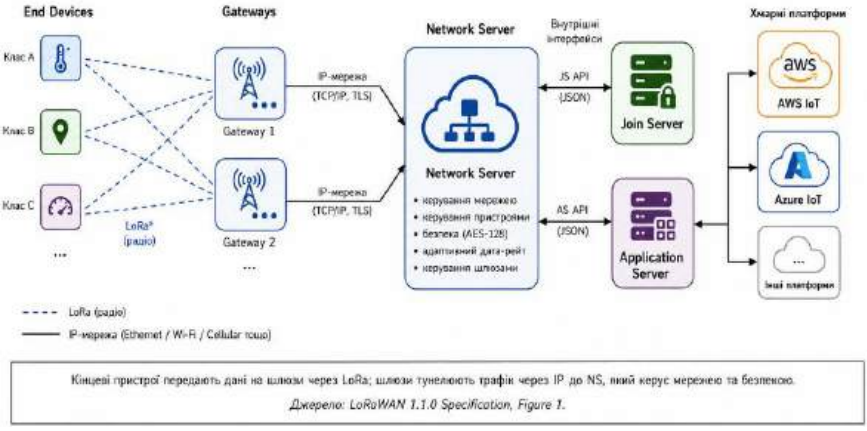


Рисунок 7.2 – Архітектура LoRaWAN (зірка зірок)

Практична реалізація LoRaWAN-пристрою на мікроконтролері потребує стеку, сумісного зі специфікацією LoRa Alliance. Найпоширенішим є

стек LMIC (LoraMAC-in-C), спочатку розроблений IBM, а нині підтримуваний спільнотою у версіях MCCI LMIC та Arduino-LMIC. Для STM32 існує офіційний пакет I-CUBE-LRWAN (STMicroelectronics), який містить повну реалізацію LoRaWAN 1.0.3/1.0.4 для роботи з трансиверами Semtech SX1272/SX1276/SX1262. У лістингу 7.2 наведено фрагмент коду, що демонструє процедуру OTAA-активації з використанням LMIC на умовній платформі.

Лістинг 7.2 – OTAA-активація LoRaWAN-пристрою з використанням LMIC (Arduino / MCCI LMIC)

```
// Відповідно до MCCI LoRaWAN LMIC Library та LoRaWAN L2 1.0.4 Specification
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
// ------ Pin Mapping ------
/** * @brief Визначення пінів для підключення LoRa-модуля
 * @note  nss  - Chip Select (SS)
 *   rxtx - не використовується (LMIC UNUSED PIN)
 *   rst  - Reset
 *   dio  - масив DIO0, DIO1, DIO2 (DIO2 необов'язковий) */
static const lmic_pinmap lmic_pins = {
    .nss = 10,                  // Chip Select
    .rxtx = LMIC_UNUSED_PIN,    // RX/TX (не використовується)
    .rst = 9,                   // Reset
    .dio = {2, 6, 7},           // DIO0, DIO1, DIO2
};
// ------ OTAA Keys ------
/** * @brief Ключі для OTAA-активації
 * @note  DevEUI - унікальний ідентифікатор пристрою (8 байт)
 *        AppEUI - ідентифікатор додатку (8 байт)
 *        AppKey - ключ додатку (16 байт) - використовується для генерації сеансових ключів під час OTAA */
static const u1_t PROGMEM DEVEUI[8]  = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
static const u1_t PROGMEM APPEUI[8]  = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
static const u1_t PROGMEM APPKEY[16] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
// ------ Event Handler -----
/** * @brief Callback-функція для обробки подій LMIC
 * @param ev  Тип події
 * @note  EV_JOINED генерується після успішного приєднання до мережі */
static void onEvent(ev_t ev) {
  switch (ev) {
    case EV_JOINED: // Успішне приєднання до мережі
       // Збереження сеансових ключів (NwkSKey, AppSKey)
     break;         // ... обробка інших подій    }    }
// ------ Setup -----
```

```
43:void setup() {
44:    // 1. Ініціалізація LMIC
45:    os_init();
46:    // 2. Скидання MAC-стану
47:    LMIC_reset();
48:    // 3. Встановлення OTAA-ключів
49:    LMIC_setDevEui(DEVEUI);          // DevEUI
50:    LMIC_setAppEui(APPEUI);          // AppEUI
51:    LMIC_setAppKey(APPKEY);          // AppKey
52:// 4. Компенсація похибки тактування (для ATmega328 та інших)
53:    LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);
54:    // 5. Реєстрація callback-функції
55:    LMIC_registerEventCb(onEvent);
56:    // 6. Запуск OTAA-активації (автоматично)
57:}
58:// ------ Main Loop ------
59:void loop() {
60:    // Обробка подій LMIC (обов'язково викликати в циклі)
61:    os_runloop_once();
62:}
```

Архітектура LoRaWAN є вдалим прикладом розподіленої системи, де прості та дешеві кінцеві пристрої з мінімальною логікою покладаються на потужну централізовану інфраструктуру шлюзів та мережевих серверів. Такий підхід дозволив LoRaWAN стати домінуючою технологією LPWAN для приватних, публічних та гібридних мереж IoT по всьому світу. Наступний підрозділ, присвячений еволюції LoRa до діапазону 2,4 ГГц та технології LR-FHSS, покаже, як ця екосистема адаптується до викликів масового масштабування та глобальної сумісності.

7.1.3. Еволюція до LoRa 2.4 ГГц та LR-FHSS для гібридних мереж

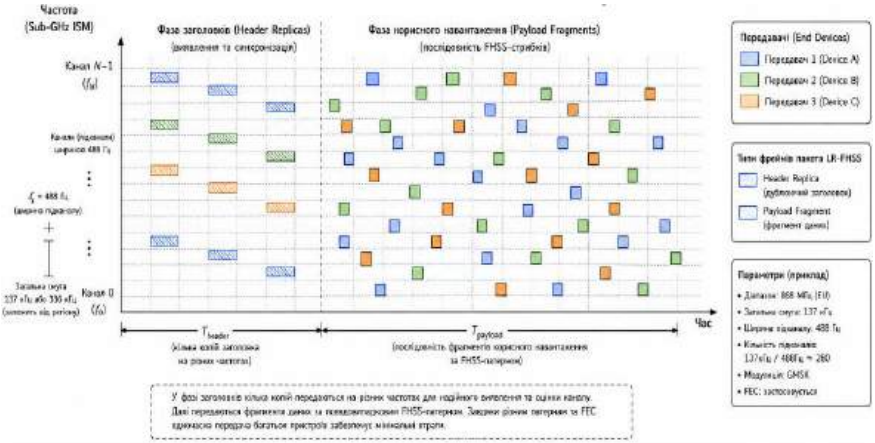
Історично LoRa асоціювалася переважно з регіональними субгігагерцовими діапазонами — 868 МГц у Європі, 915 МГц у Північній Америці, 433 МГц та 470 МГц в Азії. Ці частоти забезпечують чудову дальність поширення завдяки меншим втратам у вільному просторі та кращому проникненню крізь перешкоди. Проте субгігагерцові діапазони мають фундаментальні обмеження: вони не є глобально уніфікованими, що змушує виробників випускати регіональні варіанти пристроїв; доступна смуга частот обмежена (зазвичай 125–250 кГц на канал), що стримує масштабування мереж із сотнями тисяч вузлів; а максимальний робочий цикл (duty cycle) жорстко регулюється (1% або 0,1% у Європі згідно з ETSI EN 300 220), що обмежує кількість повідомлень, які один пристрій може передати за одиницю часу. У відповідь на ці виклики компанія Semtech у 2018 році випустила трансивер SX1280, який уперше реалізував фізичний рівень LoRa в

глобальному неліцензованому діапазоні 2,4 ГГц (2,400–2,4835 ГГц), а у 2021 році представила нову модуляційну схему LR-FHSS (Long Range Frequency Hopping Spread Spectrum), що стала частиною специфікації LoRaWAN починаючи з версії протоколу, адаптованої для гібридних мереж. Ці дві інновації радикально розширили межі застосування LoRa, зробивши її придатною для глобальних логістичних трекерів, масового розгортання розумних лічильників із високою щільністю вузлів та супутникових IoT-мереж (Satellite-IoT), де один супутник на низькій навколоземній орбіті виконує роль шлюзу LoRaWAN. Як зазначено в технічному огляді Semtech «LR-FHSS: The Next Frontier for LoRaWAN» (Semtech White Paper, 2022) та публікації в IEEE Internet of Things Magazine (том 5, 2022), LR-FHSS дозволяє підтримувати до 1,5 мільйона кінцевих пристроїв на один шлюз, що на порядок перевищує можливості традиційної CSS-модуляції.

Перехід у діапазон 2,4 ГГц надає кілька принципових переваг. По-перше, це глобальна уніфікація: один і той самий пристрій може легально працювати в будь-якій країні світу без модифікації апаратного забезпечення, що критично для логістичних трекерів, які перетинають кордони. По-друге, доступна смуга частот є значно ширшою — 83,5 МГц, що дозволяє використовувати більшу кількість каналів і зменшувати ймовірність колізій. По-третє, регуляторні обмеження в діапазоні 2,4 ГГц (згідно з ETSI EN 300 440 у Європі та FCC Part 15.247 у США) допускають вищі швидкості передавання та відсутність жорсткого обмеження duty cycle, натомість застосовуючи обмеження за спектральною щільністю потужності та вимоги до розширення спектра. Трансивер SX1280 підтримує LoRa-модуляцію (CSS) у діапазоні 2,4 ГГц з SF від 5 до 12 та BW від 203 кГц до 1625 кГц, а також FLRC (Fast LoRa) та GFSK для ще вищих швидкостей (до 2 Мбіт/с). Максимальна дальність LoRa на 2,4 ГГц є меншою, ніж на 868 МГц, через більші втрати поширення (приблизно на 8–10 дБ за формулою Фріїса), однак у міській забудові це частково компенсується меншим рівнем інтерференції та можливістю використання компактніших антен.

Справжнім проривом для масштабованих мереж стала модуляція LR-FHSS, яка принципово відрізняється від CSS і спрямована не на максимальну дальність, а на максимальну ємність мережі при збереженні хорошої чутливості. LR-FHSS використовує комбінацію вузькосмугової модуляції GMSK (Gaussian Minimum Shift Keying) зі швидким псевдовипадковим стрибкоподібним перемиканням частоти (Frequency Hopping). Кожен пакет даних розбивається на фрагменти, які передаються на різних частотних каналах у межах доступної смуги відповідно до псевдовипадкової по-

слідовності, відомої як шлюзу, так і кінцевому пристрою. Шлюз, оснащений чіпсетом SX1303 (або SX1302 із відповідною прошивкою), одночасно прослуховує всі канали і виконує кореляційний пошук преамбул LR-FHSS-пакетів у частотно-часовій області. Після виявлення преамбули шлюз слідує за стрибками частоти цього конкретного передавача і демодулює фрагменти даних. Оскільки різні кінцеві пристрої використовують різні псевдовипадкові послідовності стрибків, їхні пакети можуть частково перекриватися в часі та частоті без втрати даних — шлюз розрізняє їх за ортогональними сітками стрибків. Це забезпечує ефект, подібний до CDMA (Code Division Multiple Access), де множина передавачів може працювати одночасно, не заважаючи один одному. Теоретична ємність LR-FHSS-мережі сягає 1,5 мільйона пристроїв на шлюз, тоді як для класичного LoRa CSS цей показник становить близько 50 тисяч. Схематичне представлення частотно-часової діаграми LR-FHSS із кількома передавачами наведено на рисунку 7.3.



LR-FHSS забезпечує масовий паралельний доступ через ортогональні частотно-часові патерни стрибків.

Джерела: Semtech LR-FHSS White Paper, 2022, Figure 4.

Рисунок 7.3 – Частотно-часова діаграма LR-FHSS із трьома передавачами

Гібридні мережі LoRaWAN, що поєднують CSS у субгігагерцовому діапазоні, CSS на 2,4 ГГц та LR-FHSS на 2,4 ГГц, дозволяють операторам мереж гнучко балансувати між дальністю, пропускною здатністю та ємністю. Типовий сценарій передбачає використання CSS на 868/915 МГц для пристроїв, що потребують максимальної дальності та глибокого проникнення в приміщення (підвальні лічильники, сільськогосподарські датчики),

та LR-FHSS на 2,4 ГГц для пристроїв із високою щільністю розташування (логістичні трекери на складах, розумні лічильники у багатоквартирних будинках). Шлюзи LoRaWAN поступово стають мультипротокольними, об'єднуючи SX1303 (для субгігагерцового CSS) та SX1280/LC11xx (для 2,4 ГГц CSS/LR-FHSS) в одному корпусі. Порівняльні характеристики класичної LoRa CSS (868 МГц), LoRa CSS (2,4 ГГц) та LR-FHSS (2,4 ГГц) зведено в таблиці 7.3.

Таблиця 7.3 – Порівняння LoRa CSS (868 МГц), LoRa CSS (2,4 ГГц) та LR-FHSS (2,4 ГГц)

Параметр	LoRa CSS (868 МГц, SF12, BW125)	LoRa CSS (2,4 ГГц, SF12, BW203)	LR-FHSS (2,4 ГГц)
Модуляція	CSS (Chirp Spread Spectrum)	CSS (Chirp Spread Spectrum)	GMSK + FHSS (Frequency Hopping Spread Spectrum)
Бітова швидкість (кбіт/с, тип.)	~0,3	~0,6	~0,3–1,6 (залежить від конфігурації)
Чутливість (дБм, тип.)	-137 … -148	-130 … -132	-127
Максимальна дальність (сільська, км)	до 15	~5	~3–5
Ємність мережі (пристроїв на шлюз)	~50 тис.	~50 тис.	~1,5 млн. (до 140× більше, ніж LoRa SF12)
Стійкість до інтерференції	висока (DSSS)	висока (DSSS)	дуже висока (FHSS + CDMA-подібний поділ)
Регуляторні обмеження	Duty cycle 1% (ETSI)	без duty cycle (ISM 2,4 ГГц)	без duty cycle (ISM 2,4 ГГц)

Практична підтримка LR-FHSS вже реалізована в мікросхемах Semtech LR1121 (трансивер з підтримкою субгігагерцового LoRa, 2,4 ГГц LoRa, LR-FHSS, Wi-Fi passive scanning та GNSS sniffing), який є флагманським рішенням для глобальних IoT-пристроїв. У лістингу 7.3 наведено фрагмент коду ініціалізації LR-FHSS на LR1121 за допомогою драйвера Semtech, де конфігуруються параметри стрибкоподібної сітки (Grid) та встановлюється режим передавання.

Лістинг 7.3 – Ініціалізація LR-FHSS на трансивері Semtech LR1121 (C, драйвер Semtech)

```c
// Відповідно до Semtech LR1121 User Manual Rev 1.1
(UM.LR1121.W.APP)
#include "lr1121_driver.h"
// ------ Параметри LR-FHSS ------
/** * @brief Структура параметрів LR-FHSS
 * @note  Частота: 2.404 ГГц (2.4 ГГц ISM-діапазон)
 *  BW: 1523 кГц - смуга пропускання [16†L29]
 *  Grid: 4 - кількість сіток (визначає крок стрибків)
 * Режим: Uplink, DR8 (швидкість даних 488.28125 біт/с) */
lr_fhss_params_t fhss_params = {
    .frequency = 2404000000,   // 2.404 ГГц
    .bandwidth = 1523000,      // 1523 кГц
    .grid      = 4,         // Кількість сіток (4)
    .mode      = LR_FHSS_UPLINK,
    .data_rate = LR_FHSS_DR8,  // 488.28125 біт/с
    .header_count = 1,     // Кількість заголовків
    .coding_rate = CR_5_6, // Швидкість кодування 5/6
};
// -----Ініціалізація та передача ------
void lr_fhss_init_and_send(uint8_t *payload, uint8_t len) {
// 1. Встановлення параметрів LR-FHSS
    lr_fhss_set_parameters(&fhss_params);
// 2. Побудова кадру (кодування, таблиця стрибків)
    lr_fhss_build_frame(payload, len);
// 3. Запуск передачі
    radio_tx();
}
```

Еволюція LoRa у бік 2,4 ГГц та впровадження LR-FHSS демонструють, як зріла технологія може адаптуватися до нових викликів масштабування та глобалізації. Для розробника IoT-пристрою це означає можливість обрати оптимальний фізичний рівень залежно від вимог: CSS на 868 МГц для максимальної дальності, CSS на 2,4 ГГц для глобальної сумісності, або LR-FHSS для мереж із надвисокою щільністю вузлів та супутникового зв'язку. Наступний підрозділ, присвячений технологіям NB-IoT та LTE-M, продемонструє, як альтернативний підхід до LPWAN, що використовує ліцензований спектр стільникових мереж, вирішує подібні задачі, але з іншим набором компромісів.

7.2. NB-IoT та LTE-M (Cat-M1/M2)

На відміну від технологій бездротового зв'язку, що функціонують у неліцензованих діапазонах частот (ISM-смуги на кшталт 868 МГц, 2,4 ГГц для LoRa та Wi-Fi), вузькосмуговий Інтернет речей NB-IoT (Narrowband IoT) та LTE-M (Long Term Evolution for Machines, категорії

Cat-M1 та Cat-M2) є стільниковими технологіями, які працюють у ліцензованому спектрі. Ліцензований спектр — це частина радіочастотного ресурсу, на яку оператор мобільного зв'язку отримує ексклюзивне право використання від державного регулятора. Така ексклюзивність дає дві фундаментальні переваги, які визначають позиціонування NB-IoT та LTE-M у ландшафті LPWAN. По-перше, відсутність неконтрольованої інтерференції з боку інших пристроїв та мереж гарантує передбачувану якість обслуговування (Quality of Service, QoS), стабільну латентність та надійність доставки повідомлень. По-друге, оператор зв'язку може виділяти гарантовані смуги частот, застосовувати ефективніші схеми модуляції та керування радіоресурсами, що в сукупності забезпечує кращу спектральну ефективність та вищий рівень безпеки завдяки вбудованим механізмам автентифікації стільникової мережі (SIM-карта, взаємна автентифікація, шифрування на рівні радіоканалу). Саме тому NB-IoT та LTE-M є безальтернативним вибором для застосувань, де втрата пакета або несанкціоноване втручання неприпустимі: інтелектуальні електролічильники з дистанційним відключенням, медичні тривожні кнопки, системи безпеки, банкомати, POS-термінали, датчики критичної інфраструктури.

7.2.1. Ліцензований спектр, з'єднання як частина 5G NR згідно 3GPP Release 18

NB-IoT (стандартизований 3GPP як Cat-NB1 у Release 13, 2016 рік, та Cat-NB2 у Release 14, 2017 рік) та LTE-M (Cat-M1 у Release 13, Cat-M2 у Release 14) були розроблені як частина екосистеми LTE (4G). Проте вже в Release 15 (2018 рік) 3GPP заклав основу для інтеграції цих технологій у нове радіо 5G NR (New Radio), а в Release 16 (2020 рік) додав підтримку підключення пристроїв NB-IoT та LTE-M до ядра мережі 5G Core (5GC). Ключовим етапом стала специфікація 3GPP Release 17 (2022 рік), яка ввела концепцію NR NTN (Non-Terrestrial Networks) для супутникового IoT та остаточно закріпила статус NB-IoT і LTE-M як повноправних компонентів 5G, що відповідають вимогам IMT-2020 (International Mobile Telecommunications-2020) для масового машинного зв'язку (mMTC — massive Machine Type Communications). Нарешті, 3GPP Release 18 (запланований до завершення у 2024 році, із впровадженням у 2025–2026 роках) ще більше посилює цю інтеграцію, додаючи функції енергозбереження (Extended Discontinuous Reception, eDRX, Power Saving Mode, PSM), покращеної мобільності та спрощеного повторного використання частотного спектра (in-band, guard-band, standalone deployments) у контексті 5G NR.

Відповідно до дослідження, опублікованого в IEEE Communications Magazine (том 62, 2024), Release 18 робить NB-IoT та LTE-M не просто сумісними з 5G, а повністю інтегрованими, дозволяючи операторам розгортати ці технології в межах єдиної мережевої архітектури 5G NR поряд з eMBB (enhanced Mobile Broadband) та URLLC (Ultra-Reliable Low-Latency Communications).

Фізичний рівень NB-IoT базується на прямому розширенні LTE, але зі значними спрощеннями для зменшення вартості пристроїв та енергоспоживання. Один несучий канал NB-IoT займає смугу лише 180 кГц (один ресурсний блок LTE), що дозволяє розміщувати його в трьох режимах: in-band (всередині смуги LTE-каналу), guard-band (у захисному інтервалі між каналами LTE) або standalone (в окремому каналі, наприклад, на місці GSM-несучої). Використовується модуляція QPSK (для даних) та π/2-BPSK або π/4-QPSK, а максимальна швидкість передавання даних становить близько 127 кбіт/с для downlink та 159 кбіт/с для uplink (Cat-NB2). NB-IoT не підтримує хендовер (передавання з'єднання між стільниками) у з'єднаному режимі, але може виконувати повторний вибір стільника (cell reselection) в режимі очікування, що робить його придатним для стаціонарних або маломобільних пристроїв. LTE-M (Cat-M1), навпаки, підтримує смугу 1,4 МГц, модуляцію до 16QAM, швидкість до 1 Мбіт/с (Cat-M1) та до 4 Мбіт/с (Cat-M2), повноцінний хендовер і навіть передавання голосу (VoLTE). Це робить LTE-M кращим вибором для пристроїв, які потребують мобільності (автомобільні трекери, носимі пристрої) або вищої пропускної здатності. Архітектуру інтеграції NB-IoT/LTE-M у 5G NR показано на рисунку 7.4.

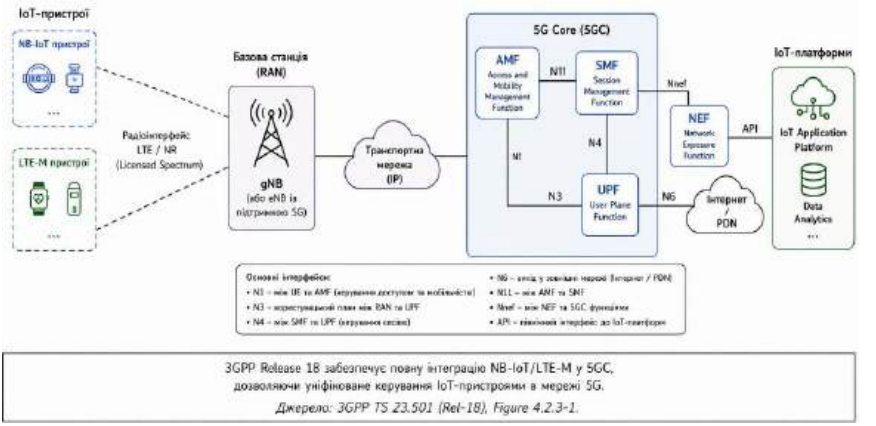


Рисунок 7.4 – Інтеграція NB-IoT/LTE-M у мережу 5G NR згідно 3GPP Release 18

У таблиці 7.4 наведено порівняння ключових характеристик NB-IoT (Cat-NB2) та LTE-M (Cat-M1/M2) згідно зі специфікаціями 3GPP TS 36.101, TS 36.201 та TS 36.300.

Таблиця 7.4 – Порівняння NB-IoT Cat-NB2 та LTE-M Cat-M1/Cat-M2

Характеристика	NB-IoT (Cat-NB2)	LTE-M (Cat-M1)	LTE-M (Cat-M2)
Смуга каналу	180 кГц (UE bandwidth 200 кГц)	1,4 МГц (6 PRB, ~1,08 МГц)	5 МГц (24 PRB)
Модуляція	DL: QPSK; UL: π/2-BPSK (single-tone), QPSK (multi-tone)	QPSK, 16QAM (DL/UL)	QPSK, 16QAM, 64QAM
Максимальна швидкість DL/UL	~127 кбіт/с / ~159 кбіт/с	1 Мбіт/с / 1 Мбіт/с	~4 Мбіт/с / ~7 Мбіт/с
Підтримка голосу (VoLTE)	ні	так	так
Мобільність (хендовер)	ні (тільки cell reselection)	так (повний хендовер)	так
Чутливість приймача	-135 дБм (тип.)	-135 дБм (тип.)	-135 дБм (тип.)
Споживання в режимі PSM	~3 мкА (тип.)	~3 мкА (тип.)	~3 мкА (тип.)
Типове застосування	стаціонарні лічильники, сільське господарство	трекери, носимі пристрої, банкомати	відеоспостереження, автомобільна телеметрія

Практична реалізація NB-IoT або LTE-M на мікроконтролері майже завжди виконується через зовнішній стільниковий модуль (модем), який підключається до хостового MCU через UART (AT-команди) або SPI/QSPI. Типовими представниками таких модулів є Quectel BC660K-GL (NB-IoT, надкомпактний), Quectel EG21-G (LTE-M/NB-IoT/GNSS), u-blox SARA-R5 (LTE-M/NB-IoT), SIMCom SIM7070G (LTE-M/NB-IoT). Ці модулі вже містять вбудований стек стільникового протоколу, включаючи Non-Access Stratum (NAS), RRC, PDCP, RLC, MAC та підтримку SIM/eSIM. Від хостового мікроконтролера вимагається лише відправляти AT-команди для конфігурації PDP-контексту (AT+CGDCONT), активації мережі (AT+COPS, AT+CFUN), створення сокету (AT+USOCR) та передавання да-

них (AT+USOWR). Для NB-IoT критичним є налаштування режимів енергозбереження PSM та eDRX за допомогою AT-команд AT+CPSMS та AT+CEDRXS, оскільки саме ці режими визначають, як часто пристрій прокидатиметься для перевірки вхідних повідомлень і скільки енергії споживатиме в режимі очікування. У лістингу 7.4 наведено приклад ініціалізації NB-IoT модуля Quectel BC660K-GL через UART на STM32 з використанням HAL.

Лістинг 7.4 – Ініціалізація NB-IoT модему Quectel BC660K-GL через UART на STM32 (HAL)

```
// Відповідно до Quectel BC660K-GL AT Commands Manual та 3GPP TS 27.007
#include "main.h"
#include "string.h"
UART_HandleTypeDef huart2;  // UART для зв'язку з модемом
#define RX_BUFFER_SIZE 256
char rx_buffer[RX_BUFFER_SIZE];
char tx_buffer[128];
// ------------------------------
//@brief  Надсилання AT-команди та очікування відповіді OK
//@param  cmd    Рядок AT-команди (без завершального \r\n)
//@param  timeout  Час очікування відповіді (мс)
//@retval 1 - успішно (отримано OK), 0 - помилка
//@note Кожна AT-команда має завершуватися <CR> (0x0D)
//Наступна команда виконується лише після завершення попередньої
// ------------------------------
uint8_t send_at_command(char *cmd, uint32_t timeout) {
  // Формування команди з завершальним \r\n
  sprintf(tx_buffer, "%s\r\n", cmd);
  HAL_UART_Transmit(&huart2, (uint8_t*)tx_buffer, strlen(tx_buffer),timeout);
   // Очікування відповіді
   uint32_t tick = HAL_GetTick();
   while (HAL_GetTick() - tick < timeout) {
    if (HAL_UART_Receive(&huart2, (uint8_t*)rx_buffer, 1, 100) == HAL_OK) {
            // Пошук "OK" або "ERROR" у буфері (спрощено)
            if (strstr(rx_buffer, "OK") != NULL) return 1;
            if (strstr(rx_buffer, "ERROR") != NULL) return 0;
        }    }    return 0;   }
// ----------------------
// @brief  Ініціалізація NB-IoT модему BC660K-GL
// @note   Послідовність команд згідно з Quectel AT Commands Manual
//Базова швидкість UART за замовчуванням 115200 бод
// ----------------------
void nbiot_init(void) {
    // 1. Перевірка зв'язку з модемом
    // Якщо відповіді немає - повторювати AT кожні 500 мс до 10 разів
    while (!send_at_command("AT", 1000)) {
```

```
    HAL_Delay(500);     }
// 2. Отримання IMEI (міжнародний ідентифікатор обладнання)
send_at_command("AT+CGSN=1", 2000);
// 3. Отримання IMSI (міжнародний ідентифікатор абонента)
send_at_command("AT+CIMI", 2000);
// 4. Встановлення APN (Access Point Name)
//    Формат: AT+CGDCONT=<cid>,"<PDP_type>","<APN>"
send_at_command("AT+CGDCONT=1,\"IP\",\"iot.operator.com\"",
5000);
// 5. Вибір оператора мережі
//    AT+COPS=<mode>,<format>,<operator>
send_at_command("AT+COPS=1,2,\"24001\"", 10000);
// 6. Активація радіоінтерфейсу (повна функціональність)
//    AT+CFUN=1 - повний функціональний режим
 send_at_command("AT+CFUN=1", 5000);
 // 7. Налаштування PSM (Power Saving Mode)
 // Формат: AT+CPSMS=<mode>,<requested_periodic_TAU>,
<requested_active_time>
    //    "00100010" - періодичне оновлення TAU кожні 10 хв
(0x22 = 34 децисекунди)
    //    "00000010" - активний час 2 хв (0x02 = 2 децисекунди)
    send_at_command("AT+CPSMS=1,,,\"00100010\",\"00000010\"",
5000);
  // 8. Перевірка реєстрації в мережі
  send_at_command("AT+CEREG?", 3000);  // Статус реєстрації EPS
}
// ----------------------------
// @brief  Створення TCP-сокету та відправлення даних
// @note   AT+USOCR=6 - створення TCP-сокету (6 = TCP)
// AT+USOWR=<socket>,<length>,<data> - запис даних у сокет
// ----------------------------
void nbiot_send_tcp(char *data, uint16_t len) {
    // 1. Створення TCP-сокету
    send_at_command("AT+USOCR=6", 5000); // Відповідь: +USOCR:
<socket_id>
    // 2. Відправлення даних через сокет
    char cmd[64];
    sprintf(cmd, "AT+USOWR=0,%d,\"%s\"", len, data);
    send_at_command(cmd, 5000);     }
```

NB-IoT та LTE-M, спираючись на ліцензований спектр та повну інтеграцію з архітектурою 5G NR згідно з 3GPP Release 18, пропонують розробникові IoT-пристрою неперевершену комбінацію надійності, безпеки, гарантованої якості обслуговування та глобального покриття. Вибір між ними визначається вимогами до мобільності, швидкості та енергоспоживання, проте обидві технології забезпечують багаторічну автономну роботу від батареї, що робить їх основними претендентами на роль комунікаційного ядра в масових розподілених системах Інтернету речей. Наступний підрозділ детальніше розкриє механізми енергозбереження (PSM, eDRX) та особливості інтеграції стільникових модулів із мікроконтролерами.

7.2.2. Режими PSM та eDRX, глибока автономність (>10 років від батареї)

Фундаментальною перевагою стільникових технологій LPWAN, таких як NB-IoT та LTE-M, перед класичним стільниковим зв'язком є не лише оптимізований фізичний рівень, а й наявність спеціалізованих механізмів енергозбереження, які дозволяють кінцевому пристрою перебувати в стані надзвичайно низького споживання протягом 99,9% часу, зберігаючи при цьому реєстрацію в мережі. Двома ключовими механізмами, що забезпечують заявлену виробниками та операторами глибоку автономність понад 10 років від однієї літій-тіонілхлоридної батареї ємністю 2,5–3,6 А·год, є режим енергозбереження PSM (Power Saving Mode) та розширений режим переривчастого приймання eDRX (extended Discontinuous Reception). Обидва механізми стандартизовані 3GPP (TS 24.301, TS 23.682, TS 36.304) та є обов'язковими для підтримки в мережах, що претендують на відповідність вимогам 5G NR згідно з Release 18. Їхнє розуміння є критичним для розробника автономного IoT-пристрою, оскільки правильний вибір і налаштування цих режимів визначають не лише термін служби батареї, а й доступність пристрою для вхідних команд (downlink reachability) та затримку доставки повідомлень.

Режим Power Saving Mode (PSM), вперше введений у 3GPP Release 12, є найрадикальнішим механізмом енергозбереження, що дозволяє пристрою фактично повністю відключати радіотракт на тривалий час — від кількох хвилин до кількох днів або навіть тижнів. Після завершення активної сесії передавання даних пристрій переходить у стан PSM, в якому він не прослуховує пейджинговий канал (paging channel) і є недосяжним для мережі. З точки зору мережі такий пристрій виглядає як вимкнений, хоча насправді він залишається зареєстрованим у мережі (зберігається контекст PDN-з'єднання, IP-адреса тощо). Тривалість перебування в PSM визначається двома таймерами: T3324 (Active Timer), який задає час, протягом якого пристрій залишається доступним для пейджингу після завершення передавання (зазвичай від 0 до 186 хвилин), та T3412 (Extended TAU Timer), який визначає періодичність процедури відстеження зони (Tracking Area Update, TAU), що виконується для підтвердження присутності пристрою в мережі. Таймер T3412 може бути встановлений у діапазоні від 2 секунд до 413 днів. Протягом часу, коли пристрій перебуває в PSM (тобто після закінчення Active Timer і до наступного TAU або наступної вихідної передачі), струм споживання модему падає до рівня струму витоку (leakage current), який для сучасних модулів, таких як

Quectel BC660K-GL або u-blox SARA-R5, становить від 2 до 5 мкА. Саме цей ультранизький струм у поєднанні з великою ємністю літієвої батареї дозволяє досягти терміну автономної роботи понад 10 років для сценаріїв із рідкісним передаванням (наприклад, одне повідомлення на добу). Часову діаграму струму споживання для пристрою, що використовує PSM, наведено на рисунку 7.5.

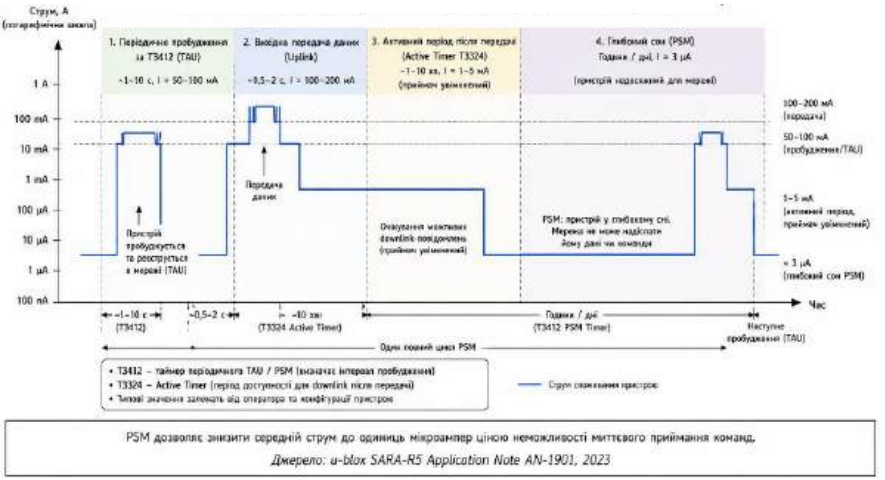


Рисунок 7.5 – Часова діаграма струму споживання NB-IoT пристрою в режимі PSM

Extended Discontinuous Reception (eDRX), специфікований у 3GPP Release 13, є менш радикальним, але гнучкішим механізмом, призначеним для пристроїв, які повинні залишатися досяжними для мережі з відносно невеликою затримкою, але при цьому значно економити енергію. У звичайному режимі DRX (Discontinuous Reception) пристрій прокидається для прослуховування пейджингового каналу кожні 1,28 або 2,56 секунди, що призводить до середнього струму споживання порядку 1–5 мА. eDRX дозволяє збільшити цей інтервал до десятків хвилин (максимальне значення для NB-IoT становить 2 години 54 хвилини, для LTE-M — близько 40 хвилин), причому пристрій домовляється про конкретне значення eDRX-циклу з мережею під час процедури Attach або TAU. У проміжках між пробудженнями пристрій перебуває в режимі сну (Sleep) зі струмом споживання 0,1–0,5 мА, що значно менше, ніж у режимі безперервного приймання, але більше, ніж у PSM. На відміну від PSM, пристрій в eDRX залишається досяжним для мережі; максимальна затримка доставки вхідного

повідомлення дорівнює тривалості eDRX-циклу. Це робить eDRX ідеальним для застосувань, де потрібне періодичне віддалене керування — наприклад, клапани в системах зрошення, освітлювальні прилади, замки, які мають відкриватися за командою зі смартфона із затримкою не більше кількох хвилин.

Кількісне зіставлення трьох режимів — безперервного приймання (No Power Saving), eDRX та PSM — за критеріями середнього струму споживання, досяжності та максимальної затримки наведено в таблиці 7.5, що дозволяє обрати оптимальний режим для конкретної прикладної задачі.

Таблиця 7.5 – Порівняння режимів енергозбереження NB-IoT/LTE-M

Параметр	Без збереження	eDRX	PSM
Середній струм споживання (типовий)	~1–5 мА (активний режим)	38–100 мкА (залежно від циклу eDRX)	0.8–3 мкА
Досяжність для мережі	постійно (завжди на зв'язку)	періодично (у вікнах пейджингу)	ні (лише після вихідної передачі або TAU)
Максимальна затримка downlink	<1 с	до тривалості eDRX-циклу (секунди – десятки хвилин)	години/дні (до наступного TAU або uplink)
Типове застосування	живлення від мережі, критичні команди	керовані клапани, освітлення	лічильники, датчики з періодичним звітуванням

Практичне налаштування цих режимів виконується через AT-команди. Для PSM пристрій надсилає команду AT+CPSMS з параметрами режиму, таймерів T3324 та T3412. Для eDRX використовується AT+CEDRXS (для LTE-M) або AT+NPTWEDRXS (для NB-IoT). У лістингу 7.5 наведено приклад конфігурації PSM для лічильника, що прокидається раз на добу.

Лістинг 7.5 – Конфігурація PSM для NB-IoT модему

```
// Відповідно до 3GPP TS 27.007 та Quectel AT Commands Manual
// ------ Налаштування PSM (Power Saving Mode) -----
/** * @brief Встановлення параметрів PSM
 * @note AT+CPSMS=<mode>,<requested_periodic_TAU>,
<requested active time>
 * Формат: 8-бітний рядок (8 октетів, кожен — 2 hex-цифри)
 * T3412 — періодичне оновлення TAU (час сну)
 * T3324 — активний час після TAU */
// Приклад 1: Тестовий режим (T3324 = 1 хв, T3412 = 5 хв)
// "00000001" → T3324 ≈ 1 хв, "00000101" → T3412 ≈ 5 хв
```

```
AT+CPSMS=1,,,"00000001","00000101"
// Приклад 2: Реальна робота (T3412 ≈ 1 година)
// "00000001" → T3324 ≈ 1 хв, "00101000" → T3412 ≈ 1 година
AT+CPSMS=1,,,"00000001","00101000"
// Приклад 3: Максимальний час сну (T3412 ≈ 320 годин)
// "11001110" → T3412 ≈ 320 годин
AT+CPSMS=1,,,"00000001","11001110"
// ----- Перехід у мінімальний режим перед сном -----
/** * @brief Перехід у мінімальний функціональний режим
 * @note  AT+CFUN=0 - вимкнення радіо, збереження енергії
 * AT+CFUN=1 - повна функціональність */
AT+CFUN=0    // Мінімальний режим (перед глибоким сном)
```

Окремо слід наголосити, що досягнення 10-річної автономності є не лише результатом низького споживання модему в PSM, а й наслідком системного підходу до проектування всього пристрою. Загальний термін служби визначається за формулою (7.4):

$$T_{\text{життя}} = E_{\text{бат}} / (I_{PSM} * t_{PSM} + I_{\text{акт}} * t_{\text{акт}} + I_{MCU} * t_{MCU} + I_{\text{вит}} * t_{\text{заг}}) / 8760, \quad (7.4)$$

де $E_{\text{бат}}$ — енергія батареї в мА·год, I_{PSM} — струм модему в PSM, t_{PSM} — частка часу в PSM, $I_{\text{акт}}$ — струм модему під час передавання, $t_{\text{акт}}$ — частка часу передавання, I_{MCU} — середній струм мікроконтролера з урахуванням його власних режимів сну, $I_{\text{вит}}$ — струм витоку інших компонентів (датчиків, DC/DC-перетворювача). Для реального досягнення 10 років необхідно, щоб середній струм усього пристрою не перевищував 25–30 мкА (для батареї 2400 мА·год). Тому розробник має з особливою ретельністю ставитися до вибору DC/DC-перетворювача з малим власним споживанням (менше 1 мкА в режимі холостого ходу), вимикати живлення датчиків між вимірюваннями за допомогою MOSFET-ключів та максимізувати час перебування мікроконтролера в режимі глибокого сну (Stop або Standby). Як зазначено в дослідженні, опублікованому в IEEE Access (том 12, 2024), саме невраховані струми витоку допоміжних компонентів, а не споживання модему, є найчастішою причиною недосягнення цільових показників автономності.

Режими PSM та eDRX у поєднанні з ретельним системним дизайном дозволяють стільниковим IoT-пристроям досягти безпрецедентної автономності, що робить NB-IoT та LTE-M придатними для розгортання у важкодоступних місцях без необхідності заміни батареї протягом усього життєвого циклу виробу. Наступний підрозділ, присвячений інтеграції модулів з мікроконтролерами, покаже практичні аспекти програмування Quectel та SIMCom на базі STM32.

7.2.3. Інтеграція модемів (Quectel, SIMCom) з мікроконтролерами через UART/QSPI

Практичне втілення стільникового зв'язку у вбудованому пристрої майже ніколи не передбачає реалізації стеку NB-IoT або LTE-M безпосередньо на прикладному мікроконтролері. Натомість загальноприйнятою архітектурною моделлю є зв'язка «хостовий мікроконтролер (Host MCU) — стільниковий модем (Cellular Module)», де модем виконує роль спеціалізованого співпроцесора, що містить власний потужний процесор (зазвичай ARM Cortex-A або Cortex-M), радіочастотний тракт, повний стек стільникового протоколу від PHY до NAS (Non-Access Stratum), підтримку SIM/eSIM, GPS/GNSS (опціонально) та захищене середовище для криптографічних операцій. Хостовий мікроконтролер, своєю чергою, виконує виключно прикладні задачі — збір даних із датчиків, керування виконавчими механізмами, реалізацію бізнес-логіки, — а до модему звертається через стандартизований інтерфейс за допомогою AT-команд, які є текстовим протоколом керування модемами, успадкованим від Hayes-сумісних пристроїв і розширеним 3GPP (TS 27.007, TS 27.005) для потреб стільникових мереж. У цьому підрозділі розглянуто апаратні інтерфейси (UART, QSPI) та програмну модель взаємодії між хостом і модемом на прикладі модулів Quectel (BC660K-GL, EG21-G) та SIMCom (SIM7070G, SIM7090G), які є найтиповішими представниками на ринку вбудованих LPWAN-рішень.

Фізичне з'єднання між мікроконтролером і модемом найчастіше реалізується через асинхронний UART, оскільки він підтримується абсолютно всіма мікроконтролерами та модулями, потребує мінімальної кількості виводів (TX, RX, GND) і є достатньо швидким для передавання AT-команд та користувацьких даних на швидкостях до 921600 бод, а в деяких модулях — до 3,6 Мбод (HS-UART). Для пристроїв, що передають значні обсяги даних (наприклад, відеофрагменти з камери або пакетне оновлення прошивки), стандартний UART може стати вузьким місцем, тому сучасні модулі (наприклад, Quectel EG25-G, SIMCom SIM7600) додатково підтримують інтерфейс QSPI (Quad SPI) для високошвидкісного передавання даних між процесором модему та хостом. QSPI забезпечує швидкість до 50 Мбіт/с, використовуючи 4-бітову шину даних, і часто застосовується для передавання TCP/IP-пакетів безпосередньо в оперативну пам'ять модему з мінімальним завантаженням основного каналу UART, який залишається для службових AT-команд. Також слід зауважити, що окрім ліній даних, до обов'язкових сигналів належать PWRKEY (кнопка або сигнал увімкнення/вимкнення модему), RESET, STATUS (індикація стану мережі) та, для деяких модулів, DTR (Data Terminal Ready) для пробудження модему з

режиму сну. Схема підключення модему Quectel BC660K-GL до мікроконтролера STM32 через UART показана на рисунку 7.6, де також позначено кола живлення з обов'язковим низькоспоживним LDO-регулятором.

Програмна модель взаємодії базується на парадигмі «запит — відповідь» із використанням AT-команд. Після подачі живлення та апаратного ввімкнення (через PWRKEY) модем виконує внутрішнє завантаження, реєстрацію в мережі (процедура Attach) і після цього готовий приймати команди від хосту.

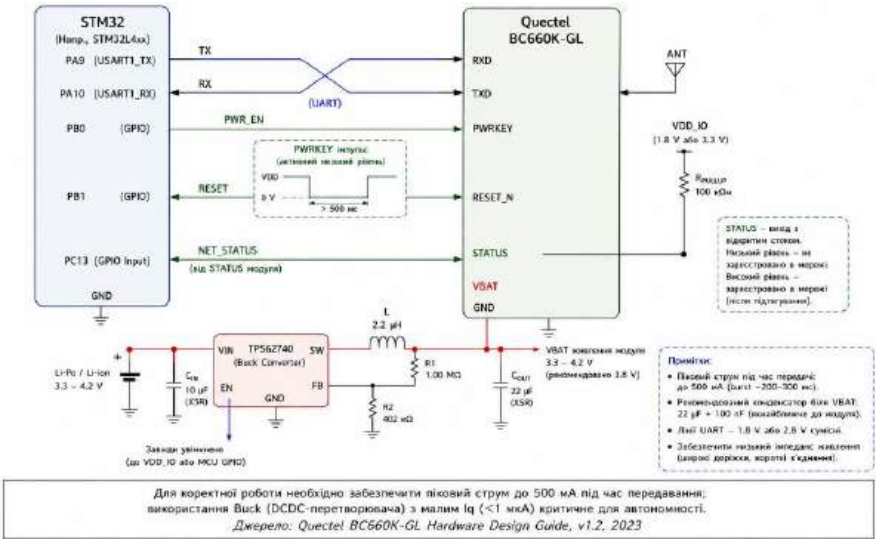


Рисунок 7.6 – Схема підключення NB-IoT модуля Quectel BC660K-GL до STM32 через UART

Кожна AT-команда починається з префікса "AT", за яким слідує основний текст команди, і завершується символом повернення каретки (\r). Модем обробляє команду і повертає відповідь, яка складається з можливого ехо-повторення команди, рядка з даними (якщо команда передбачає відповідь) та фінального рядка "OK" (успіх) або "ERROR" (помилка). Для команд, що ініціюють тривалі операції (наприклад, сканування мережі), відповідь може надходити асинхронно у вигляді URC (Unsolicited Result Code) — непрошених повідомлень, які модем надсилає з власної ініціативи (наприклад, "+CEREG: 1" при успішній реєстрації). Хостовий мікроконтролер повинен реалізувати парсер AT-відповідей, здатний відрізняти синхронні відповіді від URC, обробляти таймаути (якщо модем не відповідає протягом заданого часу) та коректно виділяти рядки даних із відповідей,

що містять інформаційні префікси (наприклад, "+COPS: 0,0,"Vodafone UA",9"). У лістингу 7.6 наведено фрагмент коду для STM32, що реалізує функцію надсилання AT-команди з очікуванням відповіді та базовим парсингом, використовуючи кільцевий буфер для прийому через DMA.

Лістинг 7.6 – Надсилання AT-команди з очікуванням відповіді через UART DMA на STM32 (HAL)

```
// Відповідно до STM32 HAL Driver та 3GPP TS 27.007
#include "main.h"
#include "cmsis_os.h"
// ------ Дескриптори та буфери ------
#define UART_RX_BUF_SIZE  1024  // Розмір DMA-буфера
#define AT_RESPONSE_MAX   256  // Максимальний розмір відповіді
UART_HandleTypeDef huart2;
static uint8_t rx_dma_buffer[UART_RX_BUF_SIZE]; // DMA-буфер
static uint8_t rx_ring_buffer[UART_RX_BUF_SIZE]; // Кільцевий буфер (КБ)
static uint16_t ring_head = 0, ring_tail = 0; // Покажчики кільцевого буфера
static osSemaphoreId_t at_semaphore;//Семафор для синхронізації
static uint8_t at_response[AT_RESPONSE_MAX]; // Буфер для відповіді
static uint16_t at_response_len = 0;
// ------ Callback UART (DMA + IDLE) -----
/** * @brief Callback при отриманні даних через UART DMA з IDLE-детекцією
 * @note  Викликається при завершенні прийому або виявленні IDLE
 * @note  IDLE-переривання дозволяє визначити кінець кадру */
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size) {
    if (huart->Instance == USART2) {
 // Копіювання даних з DMA-буфера в кільцевий буфер
    for (uint16_t i = 0; i < Size; i++) {
        rx_ring_buffer[ring_head++] = rx_dma_buffer[i];
        if (ring_head >= UART_RX_BUF_SIZE) ring_head = 0;      }
 // Перевірка наявності "OK\r\n" або "ERROR\r\n" у КБ
        if (ring_head >= ring_tail) {
            // Пошук маркерів (спрощено)
            if (strstr((char*)&rx_ring_buffer[ring_tail], "OK\r\n") != NULL || strstr((char*)&rx_ring_buffer[ring_tail], "ERROR\r\n")!= NULL) {
      // Копіювання відповіді у вихідний буфер
      // ...
     osSemaphoreRelease(at_semaphore); // Сигнал про завершення
            }        }
      // Перезапуск DMA-прийому (циркулярний режим)
      HAL_UARTEx_ReceiveToIdle_DMA(huart, rx_dma_buffer, UART_RX_BUF_SIZE);    }   }
// ----- Функція надсилання AT-команди -----
/** * @brief Надсилання AT-команди та очікування відповіді
 * @param  cmd      Рядок AT-команди (без завершального \r\n)
 * @param  response Вказівник на буфер для відповіді
 * @param  timeout  Час очікування відповіді (мс)
 * @retval 1 - успішно, 0 - помилка/таймаут
```

```
 * @note  Використовує DMA для передачі та прийому
 * @note  Для продакшен-коду слід передбачити обробку URC
(наприклад, "+CEREG") поза основною функцією очікування */
uint8_t at_send_command(const char *cmd, char *response,
uint32_t timeout) {
    char tx_buffer[128];
    // 1. Формування команди з завершальним \r\n
    sprintf(tx_buffer, "%s\r\n", cmd);
    // 2. Відправлення через DMA
    if (HAL_UART_Transmit_DMA(&huart2, (uint8_t*)tx_buffer,
strlen(tx_buffer)) != HAL_OK) {
        return 0;    }
    // 3. Очікування на семафор (встановлюється в callback)
    if (osSemaphoreAcquire(at_semaphore, timeout) != osOK) {
        return 0;   // Таймаут    }
    // 4. Копіювання відповіді у вихідний буфер
    strncpy(response, (char*)at_response, at_response_len);
    response[at_response_len] = '\0';
    return 1;   }
// ------ Ініціалізація --------
void uart_dma_init(void) {
    // Створення семафору
    at_semaphore = osSemaphoreNew(1, 0, NULL);
    // Запуск DMA-прийому з IDLE-детекцією
    HAL_UARTEx_ReceiveToIdle_DMA(&huart2, rx_dma_buffer,
UART_RX_BUF_SIZE);    }
```

Типова послідовність ініціалізації модему та встановлення TCP-з'єднання для передавання даних через NB-IoT наведена в лістингу 7.7. Вона включає перевірку зв'язку (AT), отримання ідентифікаторів (AT+CGSN, AT+CIMI), налаштування APN (AT+CGDCONT), реєстрацію в мережі (AT+COPS, AT+CEREG), налаштування режиму енергозбереження (AT+CPSMS), створення сокету (AT+USOCR), підключення до сервера (AT+USOCO) та власне передавання даних (AT+USOWR).

Лістинг 7.7 – Послідовність AT-команд для ініціалізації NB-IoT модему та відправлен-ня даних через TCP

```
// Відповідно до 3GPP TS 27.007 та Quectel AT Commands Manual
// ---- 1. Базове налаштування мережі -----
at_send_command("AT\r\n",...); // Перевірка зв'язку
at_send_command("AT+CGDCONT=1,\"IP\",\"iot.operator.com\"\r\n"
, ...); // Встановлення APN
at_send_command("AT+COPS=1,2,\"24001\"\r\n", ...); // Вибір
оператора (MCC/MNC)
// Очікування реєстрації в мережі (URC: +CEREG: 1)
// +CEREG: 1 – пристрій зареєстровано в домашній мережі
// ----- 2. Створення TCP-сокету -----
at_send_command("AT+USOCR=6\r\n", ...); // Створення TCP-сокету
(6 = TCP)
// Відповідь: +USOCR: <socket id> (наприклад, 0)
// ----- 3. Підключення до сервера -----
```

```
at_send_command("AT+USOCO=0,\"93.184.216.34\",80\r\n", ...);
// Підключення до сервера
// ----- 4. Відправлення даних -----
at_send_command("AT+USOWR=0,5,\"Hello\"\r\n", ...); // Запис
даних у сокет
```

Інтеграція стільникових модулів Quectel та SIMCom із мікроконтролерами через UART або QSPI є добре відпрацьованою інженерною практикою, яка дозволяє абстрагуватися від складнощів стільникового стеку й зосередитися на прикладній логіці. Для подальшого підвищення надійності та зменшення енергоспоживання рекомендується використовувати спеціалізовані бібліотеки (наприклад, Quectel QuecOpen SDK, який дозволяє перенести частину прикладної логіки безпосередньо в процесор модему), однак розуміння низькорівневого AT-протоколу залишається обов'язковим для налагодження та діагностики. Наступний підрозділ, присвячений супутниковим IoT-інтерфейсам, покаже, як стільникова модель зв'язку розширюється на глобальне покриття за допомогою угруповань на низькій навколоземній орбіті.

7.3. Супутникові IoT-інтерфейси: 3GPP NTN (Non-Terrestrial Networks), прямі з'єднання з супутниками Starlink Direct-to-Cell, протокол Iridium SBD

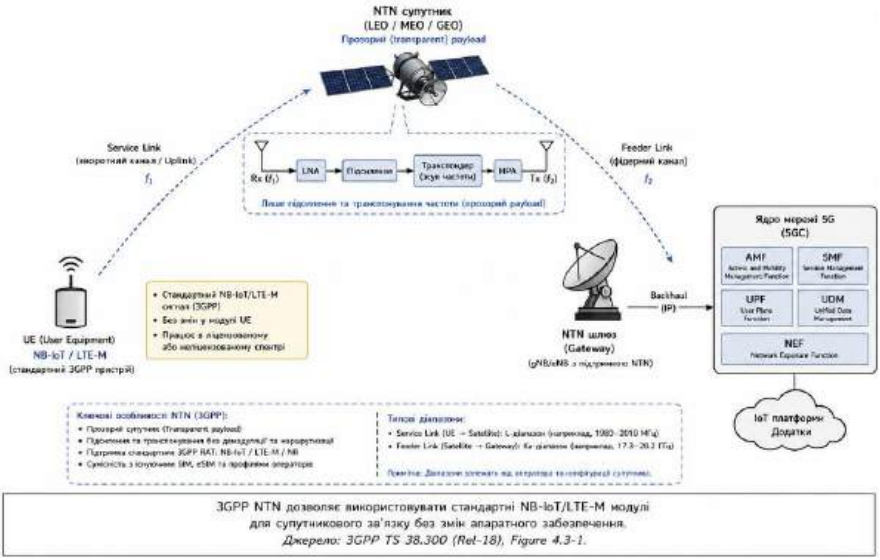
Наземні технології LPWAN, такі як LoRaWAN, NB-IoT та LTE-M, забезпечують надійне покриття на територіях, охоплених наземними шлюзами або базовими станціями стільникового зв'язку. Однак понад 80% поверхні Землі, включаючи океани, пустелі, гірські масиви, полярні регіони та віддалені сільські райони, не мають наземної інфраструктури зв'язку. Саме для цих зон супутниковий IoT є безальтернативним рішенням. Традиційно супутниковий зв'язок для телеметрії асоціювався з дорогими та енергоємними терміналами, проте в останні роки відбулися тектонічні зрушення: стандартизація неназемних мереж (NTN) консорціумом 3GPP, поява угруповань низькоорбітальних супутників (LEO) із прямою взаємодією з масовими пристроями (Direct-to-Cell) та мініатюризація існуючих супутникових модемів. Ці три напрями — 3GPP NTN, Starlink Direct-to-Cell та Iridium SBD (Short Burst Data) — формують сучасний ландшафт супутникового IoT, дозволяючи розробникам обирати рішення залежно від вимог до глобальності покриття, латентності, енергоспоживання та вартості.

Історично найпоширенішим протоколом для супутникової передачі коротких телеметричних повідомлень був Iridium Short Burst Data (SBD), що функціонує в угрупованні Iridium NEXT (66 активних супутників на низькій навколоземній орбіті, ~780 км). Мережа Iridium забезпечує справді глобальне покриття, включно з полюсами, завдяки міжсупутниковим лініям зв'язку (ISL), що утворюють самодостатню mesh-мережу в космосі без необхідності в наземних шлюзах у зоні прямої видимості терміналу. SBD-повідомлення є пакетами фіксованої максимальної довжини: до 340 байт у напрямку від терміналу (uplink) та до 270 байт до терміналу (downlink). Передавання виконується в L-діапазоні (1,616–1,626 ГГц), що забезпечує надійний зв'язок навіть за відсутності прямої видимості супутника (наприклад, під легким дахом або в лісі), хоча й з невисокою швидкістю — близько 2,4 кбіт/с. Ключовими перевагами Iridium SBD є повна глобальність, низька латентність (десятки секунд завдяки LEO-орбіті) та наявність компактних і відносно недорогих модулів (наприклад, RockBLOCK 9603 від Ground Control, що базується на чіпсеті Iridium 9603). Основним недоліком є висока вартість трафіку порівняно з наземними LPWAN та необхідність окремого модуля, несумісного з масовими стільниковими чи LoRa-мережами. Попри це, Iridium SBD залишається стандартом для віддаленого моніторингу в екстремальних середовищах — океанографічних буїв, арктичних метеостанцій, трекерів контейнерів у транстихоокеанських перевезеннях.

Революційним кроком до демократизації супутникового IoT стала стандартизація неназемних мереж NTN (Non-Terrestrial Networks) консорціумом 3GPP у Release 17 (2022 рік) із подальшим розширенням у Release 18. NTN дозволяє супутникам виступати в ролі «прозорих ретрансляторів» (transparent payload) для стандартних сигналів NB-IoT або LTE-M. На практиці це означає, що існуючі модулі NB-IoT/LTE-M (наприклад, Quectel BC660K-GL або u-blox SARA-R5) після незначного оновлення прошивки можуть підключатися до супутника, що пролітає над ними, без жодних апаратних модифікацій. Супутник ретранслює сигнал на наземний шлюз (feeder link), під'єднаний до ядра мережі 5G. Головна технічна складність, яку долає NTN, — це компенсація великого доплерівського зсуву частоти (десятки кГц) та значної затримки поширення (десятки мілісекунд), що виходять за межі стандартних допусків наземного NB-IoT. Для цього Release 17 вводить допоміжну інформацію (ephemeris data) та механізми попередньої компенсації частоти й часу на боці терміналу. Очікується, що NTN-сумісні модулі з'являться в масовому сегменті в 2025–2026

роках, а їхня вартість поступово знизиться до рівня звичайних LPWAN-модулів, що відкриє еру глобального стільникового IoT без «білих плям». Архітектуру мережі 3GPP NTN з прозорим корисним навантаженням показано на рисунку 7.7.

Найамбітнішим проектом у сфері Direct-to-Cell є ініціатива компанії SpaceX, відома як Starlink Direct-to-Cell. На відміну від 3GPP NTN, яка розрахована на поступове оновлення спеціалізованих IoT-модулів, Starlink Direct-to-Cell має на меті надати можливість звичайним, немодифікованим смартфонам (спочатку LTE Cat-1, згодом і Cat-M) безпосередньо підключатися до супутників Starlink v2.5 (Gen2) для передавання SMS, голосу та даних. Технічно це досягається завдяки тому, що супутники Starlink Gen2 обладнані потужною фазованою антенною решіткою (phased array) площею 25 м², яка формує на поверхні Землі стільникові промені, що емулюють стандартну базову станцію eNodeB. Супутники функціонують на частотах, ліцензованих у партнерстві з наземними операторами (зокрема, T-Mobile у США, Rogers у Канаді, Optus в Австралії), і використовують ті самі смуги PCS (Personal Communications Service, 1,9 ГГц), що й наземні мережі. Це означає, що користувач перебуває в роумінгу між наземною мережею та супутниковою без жодних дій зі свого боку — телефон просто бачить супутник як віддалену базову станцію.



3GPP NTN дозволяє використовувати стандартні NB-IoT/LTE-M модулі для супутникового зв'язку без змін апаратного забезпечення.
Джерело: 3GPP TS 38.300 (Rel-18), Figure 4.3-1.

Рисунок 7.7 – Архітектура 3GPP NTN (Non-Terrestrial Network) з прозорим супутником

Станом на 2026 рік сервіс Direct-to-Cell перебуває на етапі комерційного розгортання з підтримкою SMS-повідомлень, а передавання голосу та даних очікується з розширенням угруповання. Для вбудованих IoT-розробників це означає, що в майбутньому стандартний LTE-M модуль (наприклад, u-blox SARA-R5) зможе отримати глобальне покриття через Starlink без будь-яких змін, окрім оновлення прошивки для роботи з розширеними часовими затримками.

Порівняльний аналіз трьох технологій супутникового IoT наведено в таблиці 7.6. Дані базуються на специфікаціях 3GPP TS 38.300, документації Iridium SBD (Iridium 9603 Transceiver Product Specification, 2020) та публічних матеріалах SpaceX Starlink (Starlink Direct-to-Cell Overview, 2024).

Практична реалізація супутникового IoT-пристрою найчастіше передбачає використання готового модуля, що взаємодіє з мікроконтролером через UART (аналогічно до стільникових модемів, розглянутих у попередньому підрозділі).

Таблиця 7.6 – Порівняння технологій супутникового IoT

Характеристика	Iridium SBD (Iridium 9603)	3GPP NTN (NB-IoT, Rel-18)	Starlink Direct-to-Cell (LTE)
Орбіта супутників	LEO (~780 км)	LEO / MEO / GEO	LEO (~550 км / ~340 миль)
Глобальність покриття	повна (включаючи полюси)	залежить від угруповання	залежить від угод з операторами (650+ супутників D2C)
Типовий термінал	Iridium 9603/9604 модуль	стандартний NB-IoT/LTE-M модуль із NTN-прошивкою	стандартний смартфон / LTE-модуль
Максимальна довжина повідомлення (uplink)	340 байт	до 1600 байт (NB-IoT)	обмежена SMS (~140 байт), надалі IP-пакети
Латентність (середня)	~20–60 с	~5–10 с (залежить від орбіти)	~20–40 мс (сигнал) / 10–20 с (SMS)
Швидкість передавання	~2,4 кбіт/с	до 127 кбіт/с (DL)	до 3–4 Мбіт/с (DL, плановано)
Споживання терміналу (активне)	~1,5 Вт (пік)	~0,5–1 Вт (пік, як наземний NB-IoT)	~0,2–0,5 Вт (пік, як наземний LTE)
Стандартизація	пропрієтарна (Iridium)	3GPP Rel-17/18 (відкрита)	пропрієтарна (SpaceX) + 3GPP (LTE)

Для Iridium SBD це модулі RockBLOCK 9603 або безпосередньо Iridium 9603/9604, керовані AT-командами. У лістингу 7.8 наведено фрагмент коду для відправлення SBD-повідомлення через модуль Iridium 9603.

Лістинг 7.8 – Відправлення SBD-повідомлення через Iridium 9603 (AT-команди)

```
// Відповідно до Iridium AT Command Reference та RockBLOCK
Developer's Manual
// ----- 1. Перевірка рівня сигналу -----
/** * @brief Запит рівня сигналу перед передачею
 * @note  Відповідь: +CSQ: <rssi>,<ber>
 *    rssi: 0-5 (0 - немає сигналу, 5 - відмінний) */
AT+CSQ\r
// Відповідь: +CSQ: 3,99
// ------- 2. Завантаження даних у буфер ------
/** * @brief Запис текстового повідомлення у вихідний буфер (MO)
 * @note AT+SBDWT=<текст> - для ASCII-повідомлень
 *   AT+SBDWB=<довжина><дані> - для бінарних даних
 *        Ліміт: 128 байт для тексту  */
AT+SBDWT=Hello from Iridium!\r
// Відповідь: OK\r
// ------ 3. Ініціювання SBD-сеансу -----
/** * @brief Запуск передачі даних через супутникову мережу
 * @note  AT+SBDIX - ініціювання Extended SBD сеансу
 *        Тривалість сеансу може сягати 60 секунд
 *        Потрібна пряма видимість супутника  */
AT+SBDIX\r
// Відповідь: +SBDIX: <MO status>, <MOMSN>, <MT status>,
<MTMSN>, <MT length>, <MT queued>\r
// +SBDIX: 0, 1, 0, 0, 0, 0 - успішна передача
```

Супутниковий IoT перестає бути нішевим і дорогим рішенням, стрімко перетворюючись на невід'ємний компонент глобальної інфраструктури Інтернету речей. Конвергенція наземних і супутникових мереж через 3GPP NTN, а також амбітні проекти Direct-to-Cell, стирають межі між «цивілізацією» та «автономкою», дозволяючи єдиному пристрою безперервно передавати дані — від підвалу міського будинку через наземний NB-IoT до відкритого океану через супутник. Цим завершується розгляд технологій великого радіусу дії; наступна частина посібника буде присвячена локальним високошвидкісним бездротовим мережам, які, на противагу LPWAN, орієнтовані на максимальну пропускну здатність.

РОЗДІЛ 8.
ЛОКАЛЬНІ ТА СТІЛЬНИКОВІ ВИСОКОШВИДКІСНІ БЕЗДРОТОВІ МЕРЕЖІ

8.1. Wi-Fi 7 (IEEE 802.11be) у вбудованих пристроях

Стандарт IEEE 802.11be, відомий під маркетинговою назвою Wi-Fi 7, знаменує собою черговий еволюційний стрибок у продуктивності локальних бездротових мереж, орієнтований на задоволення потреб додатків із надвисокою пропускною здатністю та гранично низькими затримками. Якщо Wi-Fi 6 (802.11ax) фокусувався на ефективності в щільних середовищах, то Wi-Fi 7 націлений на досягнення максимальної теоретичної швидкості до 46 Гбіт/с, що майже вп’ятеро перевищує показники попереднього покоління. Це досягається завдяки сукупності безпрецедентних фізичних і MAC-інновацій, серед яких ключовими є Multi-Link Operation (MLO), модуляція 4096-QAM (4K QAM) та розширення ширини каналу до 320 МГц. Для вбудованих систем ці нововведення відкривають можливості, що раніше були доступні лише дротовим інтерфейсам: бездротове підключення дисплеїв із роздільною здатністю 8K, передавання нестисненого відео в системах машинного зору, створення бездротових магістралей для промислових контролерів, а також організація надшвидкісних точок доступу для шлюзів IoT, що агрегують дані з тисяч сенсорів. Специфікація IEEE 802.11be була офіційно ратифікована у 2024 році, а масове впровадження сумісних чіпсетів у сегменті вбудованих систем розпочалося у 2025–2026 роках, що робить її актуальним предметом вивчення.

8.1.1. Multi-Link Operation (MLO), 4K QAM, ширина каналу 320 МГц

Multi-Link Operation (MLO) є, без перебільшення, найреволюційнішою функцією Wi-Fi 7, яка докорінно змінює парадигму використання радіочастотного спектра. У всіх попередніх поколіннях Wi-Fi пристрій міг одночасно обмінюватися даними лише в одному частотному діапазоні (2,4 ГГц, 5 ГГц або 6 ГГц), навіть якщо апаратно підтримував кілька діапазонів. MLO дозволяє точці доступу та клієнтському пристрою встановлювати кілька незалежних з’єднань (лінків) на різних частотних каналах, які можуть належати до одного або різних діапазонів, і використовувати їх одночасно для передавання даних одного потоку або різних потоків. Розрізняють два основні режими MLO: STR (Simultaneous Transmit and Receive),

коли пристрій може одночасно передавати на одному лінку та приймати на іншому, що вимагає складнішої радіочастотної розв'язки, та NSTR (Non-Simultaneous Transmit and Receive), простіший у реалізації, де одночасна передача і прийом на різних лінках не допускаються, але можливе чергування. MLO забезпечує три ключові переваги: агрегацію пропускної здатності (сумарна швидкість дорівнює сумі швидкостей окремих лінків), підвищену надійність (якщо один лінк зазнає інтерференції, дані автоматично дублюються або перенаправляються через інший) та знижену латентність (пакет може бути відправлений через той лінк, який звільнився першим). Саме остання властивість є критичною для систем реального часу, таких як бездротове керування роботами або VR/AR-шоломи, де затримка не повинна перевищувати 5 мс. Архітектуру MLO з двома лінками в діапазонах 5 ГГц та 6 ГГц ілюструє рисунок 8.1.

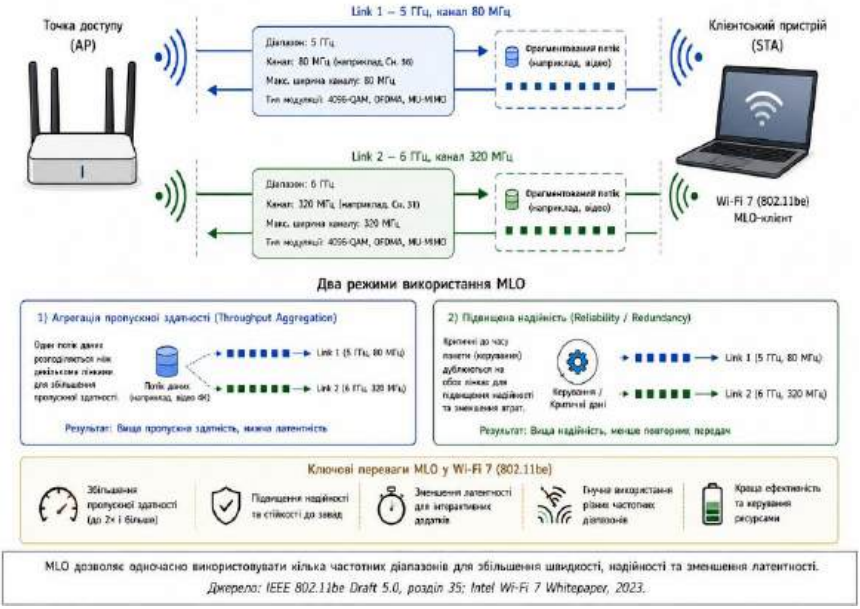


Рисунок 8.1 – Архітектура Multi-Link Operation (MLO) у Wi-Fi 7 з двома лінками

Модуляція 4096-QAM (4K QAM) є наступним кроком у підвищенні спектральної ефективності після 1024-QAM, впровадженої в Wi-Fi 6. У системі з 4096-QAM кожен символ несе 12 бітів інформації (оскільки $\log_2(4096) = 12$), порівняно з 10 бітами для 1024-QAM. Це дає приріст швидкості передавання даних приблизно на 20% за тієї самої ширини каналу та

кількості просторових потоків. Однак перехід до 4K QAM висуває надзвичайно високі вимоги до відношення сигнал/шум (SNR) та лінійності підсилювачів потужності: якщо для 1024-QAM потрібне SNR близько 35 дБ, то для 4096-QAM необхідно понад 40 дБ для досягнення прийнятної частоти бітових помилок. Це означає, що 4K QAM буде використовуватися переважно на коротких відстанях (в межах однієї кімнати) та за відсутності значних завад. У вбудованих системах ця технологія знайде застосування в бездротових дисплеях, док-станціях для гарнітур віртуальної реальності та високошвидкісних лініях зв'язку між промисловими контролерами, де відстань не перевищує кількох метрів. Порівняння модуляційних схем різних поколінь Wi-Fi наведено в таблиці 8.1.

Таблиця 8.1 – Еволюція квадратурної амплітудної модуляції (QAM) у стандартах Wi-Fi

Покоління Wi-Fi	Стандарт IEEE	Максимальна модуляція	Бітів на символ	Типове необхідне SNR (дБ)	Рік впровадження
Wi-Fi 4	802.11n	64-QAM	6	~25	2009
Wi-Fi 5	802.11ac	256-QAM	8	~32	2013
Wi-Fi 6	802.11ax	1024-QAM	10	~35	2019
Wi-Fi 7	802.11be	4096-QAM	12	~38–42	2024

Розширення ширини каналу до 320 МГц стало можливим завдяки освоєнню нового, відносно вільного від завад діапазону 6 ГГц (5,925–7,125 ГГц), відкритого для неліцензованого використання регуляторами багатьох країн у 2020–2024 роках. Якщо максимальна ширина каналу в Wi-Fi 6 становила 160 МГц, то подвоєння цього значення в Wi-Fi 7 дозволяє вдвічі збільшити пікову швидкість передавання даних. Канал 320 МГц може бути сформований як один безперервний блок спектра або як агрегація двох рознесених блоків по 160 МГц (Carrier Aggregation). Важливим доповненням є механізм Preamble Puncturing, який дозволяє пристрою використовувати неперервний широкий канал, навіть якщо частина його смуги зайнята вузькосмуговою інтерференцією (наприклад, від радара). Замість того щоб зменшувати ширину всього каналу, пристрій «вирізає» уражену ділянку спектра (punctured subchannel), продовжуючи передавати дані на решті смуги, що підвищує стійкість і середню пропускну здатність у реальних умовах. Використання каналів 320 МГц у вбудованих системах вима-

гає наявності високопродуктивних RF-компонентів і є доцільним для пристроїв, що обробляють величезні потоки даних — бездротових базових станцій, відеосерверів, шлюзів доповненої реальності.

Рисунок 8.2 ілюструє концепцію Preamble Puncturing для каналу 320 МГц.

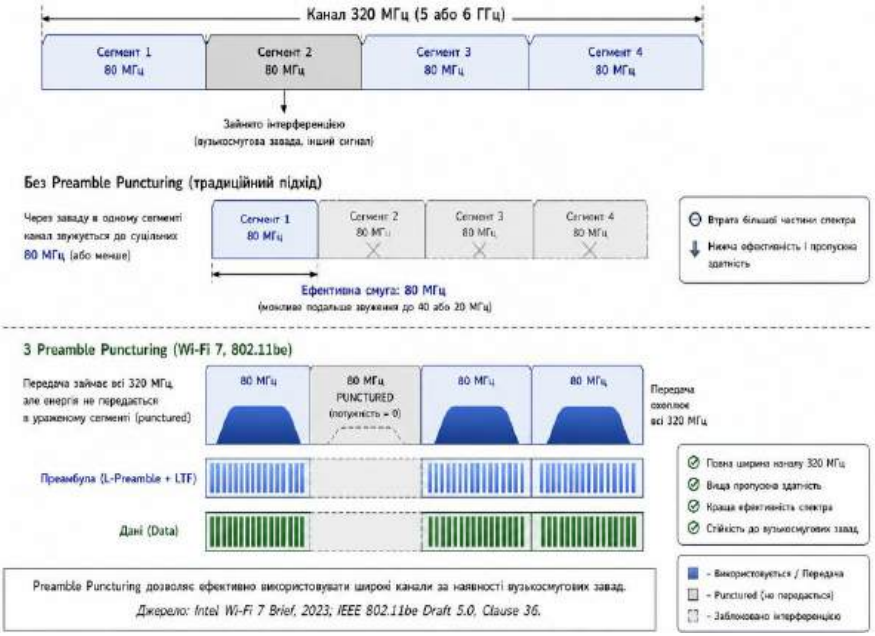


Рисунок 8.2 – Preamble Puncturing у каналі 320 МГц (Wi-Fi 7)

Сукупність MLO, 4K QAM та каналів 320 МГц у поєднанні з 16 просторовими потоками (MU-MIMO) та вдосконаленим OFDMA забезпечує безпрецедентну продуктивність, що перетворює Wi-Fi 7 на повноцінну заміну дротових інтерфейсів у багатьох сценаріях. Для вбудованих систем це означає можливість створення повністю бездротових виробничих ліній, медичних томографів із миттєвою передачею зображень на сервер обробки або автомобільних інформаційно-розважальних систем із потоковим відео 8K. Хоча на момент 2026 року повна інтеграція Wi-Fi 7 у мікроконтролери початкового рівня ще попереду, чіпсети для вбудованих застосувань (наприклад, Qualcomm QCA6696, MediaTek Filogic 380, Infineon AIROC CYW5557x) вже доступні й підключаються до хост-процесорів через PCIe 4.0 або USB 3.2, дозволяючи розробникам скористатися перевагами нового стандарту вже сьогодні.

8.1.2. Енергоефективні мікроконтролери з Wi-Fi 6/6E (ESP32-C6, Pico W2)

Якщо Wi-Fi 7, розглянутий у попередньому підрозділі, є вершиною швидкості та пропускної здатності для найвимогливіших застосувань, то для масового сегменту вбудованих IoT-пристроїв на перший план виходить баланс між достатньою продуктивністю, енергоефективністю та вартістю. У цій ніші Wi-Fi 6 (IEEE 802.11ax) та його розширення Wi-Fi 6E (із підтримкою діапазону 6 ГГц) стали «золотою серединою», що принесла у світ мікроконтролерів технології, раніше доступні лише висококласним SoC. Два знакові представники цього нового покоління енергоефективних мікроконтролерів — Espressif ESP32-C6 та Raspberry Pi Pico W2 (базований на чіпі RP2350 зі співпроцесором бездротового зв'язку CYW43455 або аналогічним, що підтримує Wi-Fi 6) — уособлюють перехід від простого «підключення до Wi-Fi» до інтелектуального керування радіоресурсами, що дозволяє батарейним пристроям роками працювати в мережах із високою щільністю клієнтів. У цьому підрозділі аналізуються ключові енергоощадні механізми Wi-Fi 6, реалізовані в цих платформах, та особливості їх практичного застосування.

Головною інновацією Wi-Fi 6, що безпосередньо впливає на енергоспоживання кінцевих пристроїв, є механізм Target Wake Time (TWT). На відміну від класичного режиму енергозбереження Power Save Mode (PSM), де клієнт періодично прокидається для прослуховування маяків (beacon) згідно з DTIM-інтервалом, TWT дозволяє клієнту та точці доступу заздалегідь узгодити індивідуальний розклад пробуджень. Клієнт повідомляє точці доступу, через які проміжки часу він прокидатиметься для обміну даними, і в решту часу може повністю вимикати радіотракт, знижуючи струм споживання до рівня витоку (одиниці мікроампер). Це принципово відрізняється від PSM, де інтервал пробудження жорстко прив'язаний до періоду DTIM (зазвичай 100–300 мс), і дозволяє збільшити періоди сну до секунд або навіть хвилин, що є критичним для датчиків із рідкісним передаванням. Крім TWT, Wi-Fi 6 запровадив OFDMA (Orthogonal Frequency Division Multiple Access) для висхідного каналу, що дозволяє точці доступу виділяти окремі ресурсні блоки (Resource Units, RU) різним клієнтам у межах одного каналу. Для малопотужного пристрою це означає, що він може передати невеликий пакет даних у виділеному вузькому піддіапазоні, не конкуруючи за середовище з іншими пристроями, що скорочує час активної передачі та зменшує ймовірність колізій. Додатково, підтримка діапазону

6 ГГц у Wi-Fi 6E (доступна в ESP32-C6, а також у чіпах, що використовуються в Pico W2) відкриває доступ до ширших і менш завантажених каналів, що також сприяє зниженню енерговитрат завдяки меншій кількості повторних передач.

Мікроконтролер Espressif ESP32-C6, анонсований у 2021 році та доступний у масових обсягах з 2023 року, є одним із перших представників сімейства ESP32 із ядром RISC-V (32-бітне, до 160 МГц) та вбудованим радіотрактом Wi-Fi 6 (802.11ax, 20 МГц, 2,4 ГГц) і BLE 5.3. Згідно з офіційним технічним описом (ESP32-C6 Technical Reference Manual, версія 1.0, Espressif, 2023), радіопідсистема підтримує режими TWT, OFDMA (uplink/downlink), MU-MIMO (downlink), а також усі обов'язкові механізми енергозбереження, передбачені стандартом. Споживання в режимі глибокого сну (Deep Sleep) із вимкненим радіо та працюючим RTC-таймером становить близько 7 мкА, а з активованим Wi-Fi TWT-режимом середній струм за цикл «сон – передача – прийом» може бути знижений до десятків мікроампер залежно від інтервалу пробудження. Espressif надає для ESP32-C6 фреймворк ESP-IDF, який містить високорівневий API для конфігурації TWT (функції з префіксом esp_wifi_set_twt()), що дозволяє розробнику задати інтервал пробудження та тривалість активного вікна без заглиблення в низькорівневі деталі протоколу.

Другим знаковим пристроєм є Raspberry Pi Pico W2 — логічне продовження лінійки Pico W, випущене у 2025 році. Pico W2 базується на новому мікроконтролері власної розробки Raspberry Pi — RP2350 (двоядерний ARM Cortex-M33 або RISC-V Hazard3, до 150 МГц), доповненому окремим бездротовим співпроцесором Infineon CYW43455 (або аналогічним) із підтримкою Wi-Fi 6 (802.11ax) у діапазонах 2,4 ГГц та 5 ГГц (Wi-Fi 6E не підтримується, оскільки CYW43455 обмежений 2,4/5 ГГц, проте майбутні ревізії можуть додати 6 ГГц). Особливістю Pico W2 є те, що стек Wi-Fi працює на співпроцесорі, звільняючи основні ядра RP2350 для прикладних задач. Ця архітектурна особливість, подібна до тієї, що використовується в оригінальному Pico W, дозволяє досягти дуже низького енергоспоживання в режимі сну, оскільки основні ядра можуть бути повністю вимкнені, а співпроцесор періодично прокидається за розкладом TWT для перевірки вхідних пакетів. Програмування Pico W2 здійснюється через C/C++ SDK або MicroPython, причому обидва середовища надають простий API для підключення до Wi-Fi (network.WLAN у MicroPython, cyw43_arch_wifi_connect у C SDK) та конфігурації режимів енергозбереження. У таблиці 8.2 наведено порівняння ключових характеристик ESP32-C6 та Pico W2, що дозволяє зробити обґрунтований вибір платформи.

Таблиця 8.2 – Порівняння енергоефективних мікроконтролерів із Wi-Fi 6

Характеристика	ESP32-C6	Raspberry Pi Pico W2
Ядро	RISC-V 32-біт (два ядра: HP та LP)	RP2350 (Cortex-M33 / RISC-V)
Частота	до 160 МГц	до 150 МГц
Wi-Fi	802.11ax (Wi-Fi 6), 20 МГц	802.11ax (Wi-Fi 6), 20/40 МГц
Діапазони	2,4 ГГц	2,4 ГГц + 5 ГГц
BLE	5.3	5.2 (через співпроцесор)
TWT	підтримується	підтримується (через стек співпроцесора)
Струм Deep Sleep (Wi-Fi вимк.)	~7 мкА	~10 мкА (RP2350 вимкнено, CYW активний)
Струм у режимі TWT (середній, залежно від інтервалу)	~30–50 мкА (при інтервалі пробудження 1 с)	~40–80 мкА (при інтервалі пробудження 1 с)
Підтримка 6E	ні (2,4 ГГц)	ні (обмежено 5 ГГц, без 6E)
Фреймворк	ESP-IDF, Arduino	C/C++ SDK, MicroPython
Типова ціна модуля	~$2,5	~$6,0 (цільова)

Для ілюстрації простоти налаштування TWT на ESP32-C6 у лістингу 8.1 наведено фрагмент коду на ESP-IDF, який конфігурує індивідуальний TWT-розклад із пробудженням кожні 5 секунд.

Лістинг 8.1 – Конфігурація Target Wake Time (TWT) на ESP32-C6 (ESP-IDF)

```
// Відповідно до ESP-IDF Programming Guide та Wi-Fi Driver API
#include "esp_wifi.h"
#include "esp_wifi_he.h" // HE (Wi-Fi 6) розширення для TWT
// -------- Підключення до Wi-Fi мережі --------
/** * @brief Підключення до точки доступу в режимі станції
 * @note  Перед налаштуванням TWT необхідно встановити Wi-Fi
з'єднання */
esp_wifi_connect();  // Підключення до AP
// ------- Отримання можливостей TWT точки доступу -----
/** * @brief Запит підтримки TWT з боку точки доступу
 * @note  Функція esp_wifi_get_twt_capability() перевіряє, чи
AP підтримує TWT
 * Якщо AP не підтримує TWT, подальше налаштування неможливе */
bool twt_capable;
```

```
esp_wifi_get_twt_capability(&twt_capable); // Отримання можливостей TWT
// ------ Налаштування структури TWT -------
/** * @brief Структура конфігурації TWT
 * @note  twt_id - ідентифікатор угоди TWT (0-7)
 *     wake_interval  - інтервал пробудження (мкс)
 *     wake_duration  - тривалість пробудження (мкс)
 * setup_command - тип команди: REQUEST, SUGGEST, DEMAND
 */
esp_twt_config_t twt_config = {
    .twt_id         = 0,    // Ідентифікатор угоди (0-7)
    .wake_interval  = 5000000,    // 5 секунд (5000000 мкс)
    .wake_duration  = 20480,    // 20 мс (20480 мкс)
    .setup_command  = TWT_REQUEST, // Запит на встановлення угоди
};
// ------ Застосування конфігурації TWT -------
/** * @brief Встановлення угоди TWT з точкою доступу
 * @note  Після успішної конфігурації модуль автоматично входитиме в сон
 * між вікнами TWT, пробуджуючись лише в зазначені інтервали */
esp_wifi_sta_config_twt(&twt_config); // Застосування TWT конфігурації
```

Слід зазначити, що енергоефективність Wi-Fi 6 на мікроконтролерах визначається не лише апаратними можливостями, а й коректною інтеграцією з точкою доступу. Як зазначено в дослідженні, опублікованому в IEEE Internet of Things Journal (том 11, 2024), ефективність TWT суттєво залежить від реалізації планувальника на боці точки доступу: деякі точки доступу можуть ігнорувати запити TWT або виділяти вікна пробудження, що перекриваються з іншими клієнтами, нівелюючи енергозбереження. Тому під час проектування системи рекомендується тестувати сумісність із цільовими моделями точок доступу та за можливості використовувати режим Broadcast TWT, де розклад визначається точкою доступу централізовано. Крім того, ESP32-C6 підтримує додатковий механізм Dynamic Multi-Polling (DMP), який дозволяє точці доступу ефективно опитувати кілька малопотужних клієнтів у межах одного TXOP (Transmission Opportunity), що ще більше знижує накладні витрати на доступ до середовища.

Поява мікроконтролерів на кшталт ESP32-C6 та Pico W2 з підтримкою Wi-Fi 6 та TWT знаменує перехід масових IoT-пристроїв від постійно активного Wi-Fi-з'єднання до імпульсного, суворо регламентованого обміну даними, що робить технологію Wi-Fi життєздатною навіть для пристроїв із батарейним живленням. Наступний підрозділ, присвячений 5G RedCap, продовжить тему балансу між продуктивністю та енергоефективністю в контексті стільникових мереж п'ятого покоління.

8.2. 5G RedCap (Reduced Capability) для промислового IoT: баланс між продуктивністю та складністю

Розвиток стільникових технологій для Інтернету речей тривалий час відбувався в межах чітко окресленої дихотомії: з одного боку — наднизькопотужні та повільні NB-IoT/LTE-M для телеметрії, з іншого — високошвидкісні та складні eMBB (enhanced Mobile Broadband) та URLLC (Ultra-Reliable Low-Latency Communications) для потокового відео, автономного керування та критичних комунікацій. Однак значна кількість промислових застосувань не вкладається в жодну з цих крайнощів. Розумні камери відеоспостереження, носимі пристрої доповненої реальності для технічного обслуговування, бездротові програмовані логічні контролери (PLC), автономні транспортні засоби на виробництві (AGV), датчики вібрації з високою частотою опитування — усі вони потребують пропускної здатності, значно вищої за можливості LPWAN, але водночас не можуть дозволити собі вартості, енергоспоживання та складності повноцінного 5G-модему для смартфонів чи автомобілів. Для заповнення цієї «середньої» ніші консорціум 3GPP у Release 17 (завершеному в 2022 році) ввів специфікації NR RedCap (Reduced Capability), формально відомі як NR Light. Подальший розвиток у Release 18 (завершеному в 2024 році) додав ще більш спрощену категорію eRedCap (evolved Reduced Capability), яка наближається за складністю до LTE Cat-1, але працює в екосистемі 5G NR. RedCap розглядається як ключовий елемент трансформації промислового IoT, оскільки дозволяє уніфікувати всю комунікаційну інфраструктуру підприємства на базі єдиного стандарту 5G, від польових датчиків до хмарних серверів, без шлюзів трансляції протоколів. Як зазначено в оглядовій статті С. Парка та співавторів (IEEE Communications Surveys & Tutorials, том 26, 2024), RedCap забезпечує «безшовну» інтеграцію пристроїв середньої складності в архітектуру 5G Core (5GC) із підтримкою нарізки мережі (network slicing), що є критичним для розмежування трафіку різних класів у межах одного підприємства.

Технічна сутність RedCap полягає в наборі цілеспрямованих обмежень, накладених на фізичний та MAC-рівні стандартного 5G NR (New Radio) згідно з TS 38.101-1, TS 38.300 та TS 38.211. По-перше, максимальна ширина каналу для RedCap-пристрою в діапазоні FR1 (нижче 7,125 ГГц) обмежується 20 МГц (порівняно зі 100 МГц для звичайного NR). Для eRedCap у Release 18 це обмеження додатково знижено до 5 МГц для деяких частотних конфігурацій, що наближає його до смуги LTE. По-друге, кількість антен скорочується до конфігурації 1T2R (один передавач, два

приймачі) для RedCap, тоді як звичайний NR передбачає щонайменше 2T4R. Це суттєво зменшує кількість радіочастотних трактів, вартість компонентів і складність обробки сигналів, але обмежує максимальну кількість просторових MIMO-шарів до одного в низхідному каналі. По-третє, RedCap підтримує напівдуплексний режим FDD (HD-FDD), за якого пристрій не здатен одночасно передавати та приймати, що дозволяє відмовитися від дорогого дуплексного фільтра. По-четверте, максимальна модуляція обмежена 64QAM (для eRedCap) або 256QAM (для базового RedCap), що зменшує вимоги до лінійності підсилювача. Наслідком цих обмежень є зниження пікової швидкості до приблизно 150 Мбіт/с у низхідному каналі та 50 Мбіт/с у висхідному для базового RedCap (порівняно з гігабітами для eMBB), що, однак, у кілька разів перевищує можливості LTE Cat-4. Для eRedCap пікова швидкість становить близько 10–25 Мбіт/с, що вже зіставно з LTE Cat-1, але з перевагою повної інтеграції в ядро 5G. Порівняння ключових характеристик RedCap, eRedCap, NB-IoT та LTE-M наведено в таблиці 8.3.

Таблиця 8.3 – Порівняння технологій стільникового IoT середньої та низької складності

Характеристика	NB-IoT (Cat-NB2)	LTE-M (Cat-M1)	NR RedCap (Rel-17)	NR eRedCap (Rel-18)
Смуга каналу	180 кГц	1,4 МГц	до 20 МГц (FR1)	до 5 МГц (FR1)
Антенна конфігурація	1T1R	1T1R (опціонально 1T2R)	1T1R або 1T2R	1T1R
Дуплекс	HD-FDD (half-duplex)	HD-FDD, FD-FDD, TDD	FD/HD-FDD, TDD	HD-FDD, TDD
Максимальна модуляція	QPSK	QPSK, 16QAM	64QAM (DL/UL)	64QAM (DL/UL)
Піков швидкість DL/UL	~127/159 кбіт/с	1/1 Мбіт/с	~150/50 Мбіт/с	~10/10 Мбіт/с
Підтримка URLLC	ні	обмежена	обмежена (Rel-17)	планується (Rel-18)
Режими енергозбереження	PSM, eDRX	PSM, eDRX	eDRX, WUS, RRM-релаксація	eDRX, WUS
Ядро мережі	EPC (LTE)	EPC (LTE)	5GC	5GC

З точки зору енергоспоживання, RedCap успадковує всі механізми, розроблені для 5G NR, а також адаптує перевірені підходи з LTE. Підтримка розширеного переривчастого приймання (eDRX) дозволяє пристрою узгоджувати тривалі цикли сну (до хвилин) із мережею, а спеціально введений у Release 18 сигнал пробудження Wake-Up Signal (WUS) дає змогу пристрою вмикати основний приймач лише тоді, коли мережа дійсно має для нього дані, а не за фіксованим розкладом. Крім того, напівдуплексний режим та менша кількість антен самі по собі знижують споживання енергії на етапі приймання. Хоча RedCap не досягає мікроамперних рівнів NB-IoT у режимі PSM, він здатен забезпечити багатоденну або багатомісячну автономність для пристроїв із помірним трафіком (наприклад, камера, що передає знімок раз на хвилину), що є значно кращим показником, ніж у традиційних широкосмугових модемах.

У промисловому IoT RedCap розглядається як основа для побудови приватних 5G-мереж (Private 5G) на території заводів, портів, складів і шахт. Єдина інфраструктура 5G NR може одночасно обслуговувати критичний до затримок трафік URLLC (екстрене відключення верстата), масовий сенсорний трафік (mMTC через NB-IoT/LTE-M у складі 5GC) та середньошвидкісний трафік RedCap (відеоспостереження, телеметрія AGV, доповнена реальність). Мережева нарізка (network slicing) дозволяє логічно ізолювати ці потоки, гарантуючи якість обслуговування для кожного. Практична реалізація RedCap-пристрою виконується на базі спеціалізованих модулів, які, подібно до NB-IoT, підключаються до хостового мікроконтролера або процесора через стандартні інтерфейси. Першими комерційними модулями стали Quectel RG620U (на чіпсеті Qualcomm Snapdragon X35) та SIMCom SIM8230, представлені у 2023–2024 роках. Ці модулі підтримують як автономний (SA) режим 5G NR, так і fallback до LTE, і можуть інтегруватися через USB 3.0, PCIe або SDIO, надаючи хосту стандартний IP-інтерфейс. У лістингу 8.2 наведено умовний фрагмент коду на Python для Raspberry Pi (як хост-процесора), що демонструє активацію контексту даних через ModemManager API (D-Bus) для модуля Quectel RG620U, підключеного через USB.

5G RedCap являє собою стратегічно важливу ланку, що усуває розрив між наднизькопотужними LPWAN та високошвидкісними широкосмуговими технологіями. Для промислового IoT це відкриває можливість консолідації всіх комунікацій на єдиній мережевій архітектурі 5G, що спрощує розгортання, підвищує безпеку та знижує сукупну вартість володіння. Наступна частина посібника перейде до програмних інтерфейсів та протоко-

лів обміну, які формують верхні рівні стеку і забезпечують інтероперабельність між пристроями незалежно від використовуваного фізичного каналу.

Лістинг 8.2 – Активація 5G RedCap-модему Quectel RG620U через ModemManager (Python, dbus)

```
# Відповідно до ModemManager D-Bus API та 3GPP 5G RedCap специфікацій
import dbus
# ------ Пошук модему за IMEI -----
# Підключення до системної шини D-Bus та пошук модему за IMEI
bus = dbus.SystemBus()
mm_service = bus.get_object('org.freedesktop.ModemManager1',
                            '/org/freedesktop/ModemManager')
mm = dbus.Interface(mm_service,
'org.freedesktop.DBus.ObjectManager')
# ... пошук модему з потрібним IMEI ...
# -----Підключення до мережі 5G SA -------
""" @brief Підключення до мережі 5G SA через ModemManager
@note  mm_modem.Simple.Connect() створює та активує Bearer
@note  ModemManager автоматично виконує атач до мережі 5G SA
@note  Для RedCap-пристроїв необхідно переконатися, що модуль підтримує відповідні capabilities (наприклад, обмежену ширину смуги) """
bearer = mm_modem.Simple.Connect({
    'apn': 'private5g.local',      # Точка доступу до мережі
    'ip-type': 4,     # IPv4 (4) або IPv4V6 (6)
})
# Відповідь: об'єкт Bearer з IP-адресою
# ------- Використання IP-адреси для сокетів -------
# Після отримання Bearer, IP-адреса може бути використана для TCP/UDP
```

ЧАСТИНА IV.
ПРОГРАМНІ ІНТЕРФЕЙСИ ТА ПРОТОКОЛИ ОБМІНУ

РОЗДІЛ 9.
ІНТЕРФЕЙСИ ПРОГРАМНОГО РІВНЯ В IOT

9.1. MQTT v5.0: публікація-підписка, властивості пакетів, причини завершення сесії

MQTT (Message Queuing Telemetry Transport) є де-факто стандартним протоколом прикладного рівня для Інтернету речей, який реалізує модель публікації-підписки (publish-subscribe), докорінно відмінну від клієнт-серверної парадигми HTTP чи «запит-відповідь» CoAP. У центрі цієї моделі перебуває посередник — MQTT-брокер, який виконує роль маршрутизатора повідомлень, звільняючи кінцеві пристрої від необхідності знати мережеву адресу одне одного.

9.1.1. Брокери (Mosquitto, EMQX), якість обслуговування (QoS 0/1/2)

Клієнт, що публікує дані (наприклад, датчик температури), відправляє повідомлення на брокер, вказуючи тему (topic) — ієрархічно структурований рядок, наприклад, "building/room1/temperature". Брокер зіставляє цю тему зі списком підписок, раніше зареєстрованих іншими клієнтами (споживачами даних), і доставляє повідомлення кожному з них. Така архітектура забезпечує просторову та часову розв'язку: видавець і підписник не мусять бути онлайн одночасно, а брокер може зберігати останнє повідомлення (retained message) для доставки новим підписникам. Специфікація MQTT версії 5.0, прийнята OASIS у 2019 році, суттєво розширила базовий протокол, додавши властивості пакетів (Properties), коди причин завершення сесії (Reason Codes), механізми узгодження можливостей (Capabilities), покращений контроль потоку та гнучке керування сесіями. Цей підрозділ зосереджується на центральному елементі екосистеми — брокерах — та фундаментальному механізмі забезпечення надійності доставки — рівнях якості обслуговування (Quality of Service, QoS).

Брокер є не пасивним ретранслятором, а активним вузлом, який виконує автентифікацію клієнтів, авторизацію доступу до тем, керування сесіями (включаючи збереження непрочитаних повідомлень для відключених клієнтів), перетворення протоколів (наприклад, MQTT-SN у MQTT) та інтеграцію з зовнішніми системами через плагіни. У світі відкритого програмного забезпечення домінують два брокери, кожен із яких орієнтований на власний клас завдань. Eclipse Mosquitto, що розробляється під егідою

Eclipse Foundation, є легковаговим, написаним на мові C брокером, який може працювати навіть на одноплатних комп'ютерах із обмеженими ресурсами, таких як Raspberry Pi. Mosquitto підтримує всі версії протоколу MQTT (3.1, 3.1.1, 5.0), забезпечує роботу з TLS, мостами (bridge) для об'єднання кількох брокерів та динамічним контролем доступу через плагіни. Його архітектура оптимізована для сценаріїв із сотнями або тисячами клієнтів, де критичною є мінімальна затримка та передбачуване споживання пам'яті, що робить Mosquitto ідеальним вибором для локальних шлюзів IoT, промислових контролерів і невеликих приватних мереж. На протилежному кінці спектру знаходиться EMQX (раніше EMQ X), розроблений китайською компанією EMQ Technologies на мові Erlang/OTP, яка відома своєю здатністю підтримувати мільйони одночасних з'єднань. EMQX є хмарно-орієнтованим, горизонтально масштабованим брокером, що підтримує кластеризацію, інтеграцію з Apache Kafka, PostgreSQL, Redis, MongoDB, а також розширену підтримку MQTT 5.0, включаючи всі властивості пакетів. EMQX часто використовується операторами зв'язку, великими IoT-платформами та в промислових хмарних рішеннях, де кількість пристроїв вимірюється десятками або сотнями тисяч. Порівняння ключових характеристик цих брокерів наведено в таблиці 9.1.

Таблиця 9.1 – Порівняння брокерів Mosquitto та EMQX

Характеристика	Eclipse Mosquitto (2.x)	EMQX (5.x)
Мова програмування	C / C++	Erlang/OTP
Продуктивність (повідомлень / с на одне ядро)	~20 000 (типово)	~100 000+
Максимальна кількість з'єднань	~100 000 (один вузол)	~1 000 000 (один вузол)
Підтримка MQTT 5.0	повна	повна
Масштабування	вертикальне, міст (bridge)	горизонтальне (кластер)
Підтримка правил/плагінів	плагіни на C/Lua	вбудований Rule Engine, WebHook, Kafka
Ліцензія	Eclipse Public License 2.0	Apache 2.0 (Open Source Edition)

Центральним механізмом, що визначає поведінку MQTT-брокера та мережевий трафік, є рівень якості обслуговування QoS, який встановлюється для кожного окремого повідомлення. Специфікація MQTT 5.0 визначає три рівні QoS — 0, 1 та 2, — кожен із яких пропонує різний баланс між

надійністю доставки, накладними витратами та латентністю. QoS 0 (At most once, не більше одного разу) є найпростішим і найшвидшим режимом: відправник публікує повідомлення рівно один раз, без жодного підтвердження від брокера. Брокер доставляє повідомлення підписникам і не зберігає його копію, якщо підписник відключений (якщо тільки це не retained-повідомлення). Цей рівень мінімізує службовий трафік (немає ACK-пакетів) і затримку, однак абсолютно не гарантує доставку: якщо повідомлення втрачається через переповнення буфера, обрив TCP-з'єднання або збій брокера, воно зникає безслідно. QoS 0 застосовується для періодичних телеметричних даних, де втрата одного показу на загальному тренді є некритичною — наприклад, датчик, що передає температуру в теплиці кожні 10 секунд.

QoS 1 (At least once, щонайменше один раз) гарантує, що повідомлення буде доставлене до брокера (або від брокера до підписника) незалежно від можливих втрат пакетів, однак допускає дублювання. Механізм спирається на ідентифікатор пакета (Packet Identifier) та пакет підтвердження PUBACK. Відправник зберігає повідомлення у своєму буфері доти, доки не отримає PUBACK з відповідним Packet Identifier. Якщо PUBACK не надходить протягом встановленого таймауту, відправник повторює публікацію з тим самим Packet Identifier. Саме повторна передача є джерелом можливих дублікатів: якщо PUBACK було втрачено в мережі, але брокер вже отримав і обробив повідомлення, повторна публікація створить дублікат. Специфікація MQTT 5.0 вимагає, щоб брокери та клієнти обробляли дублікати на прикладному рівні, якщо це необхідно (наприклад, через ідемпотентні операції або додавання унікального ідентифікатора повідомлення в payload). QoS 1 є найпоширенішим вибором для більшості IoT-застосувань, оскільки забезпечує надійну доставку без значних накладних витрат, а періодичні дублікати, що виникають лише за умов мережевих збоїв, для багатьох сценаріїв є прийнятними.

QoS 2 (Exactly once, рівно один раз) пропонує найвищий рівень гарантії, за якого повідомлення доставляється одержувачу рівно один раз, без втрат і без дублікатів. Це досягається за допомогою чотирикрокового рукостискання: PUBLISH (QoS 2), PUBREC (Publish Received), PUBREL (Publish Release), PUBCOMP (Publish Complete). Отримавши PUBLISH із QoS 2, брокер запам'ятовує його ідентифікатор, але не доставляє далі, доки не отримає PUBREL, який підтверджує, що клієнт не скасував передавання. Лише після цього брокер публікує повідомлення для підписників і надсилає клієнту PUBCOMP. Такий багатокроковий обмін гарантує, що повідо-

млення не буде втрачене на жодному етапі, однак подвоює затримку доставки (2 round-trip time) і збільшує обсяг службового трафіку. QoS 2 застосовується в критичних до надійності застосуваннях: фінансові транзакції, команди аварійної зупинки промислового обладнання, підтвердження прийому ліків у медичних інфузійних насосах.

Важливо розуміти, що рівень QoS узгоджується між видавцем, брокером і підписником незалежно на кожному сегменті доставки. Клієнт-видавець може встановити QoS 2 для публікації, але якщо підписник оформив підписку з максимально допустимим QoS 1, брокер доставить йому повідомлення саме з QoS 1 (знизивши рівень — QoS downgrade). Це дозволяє гнучко адаптувати надійність доставки до можливостей і вимог кожного клієнта. Схематичне зображення потоків пакетів для трьох рівнів QoS на всіх сегментах мережі наведено на рисунку 9.1.

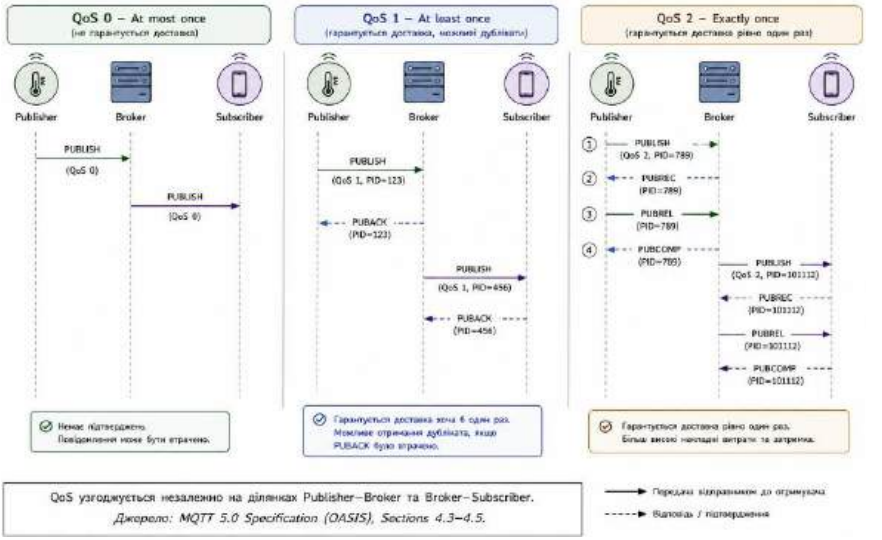


Рисунок 9.1 – Потоки пакетів для QoS 0, 1, 2 у MQTT 5.0

Практичне використання QoS на вбудованих системах реалізується через клієнтські бібліотеки. Для платформи STM32 з використанням стеку LwIP та FreeRTOS широко застосовується бібліотека coreMQTT (AWS), яка надає компактний, написаний на C клієнт MQTT 5.0, оптимізований для роботи в багатозадачному середовищі. У лістингу 9.1 наведено фрагмент коду, що демонструє публікацію повідомлення з QoS 1 та підписку на тему з QoS 2.

Лістинг 9.1 – Публікація з QoS 1 та підписка з QoS 2 на STM32 (coreMQTT + LwIP)

```
// Відповідно до FreeRTOS coreMQTT API
#include "core_mqtt.h"
// ------ Ініціалізація та підключення -------
/** * @brief Підключення до брокера
 * @note keepAliveSeconds = 60 - підтримка з'єднання
 *       cleanSession = true - початок нової сесії */
MQTTContext_t mqtt_ctx;
MQTTConnectInfo_t connect_info = {
    .keepAliveSeconds = 60,
    .cleanSession = true,
};
MQTT_Connect(&mqtt_ctx, &connect_info, NULL, 1000);
// ------ Публікація з QoS 1 ------
/** * @brief Публікація повідомлення з QoS 1
 * @note  Для QoS > 0 обов'язковий packetId
 * packetId генерується через MQTT GetPacketId()  */
MQTTPublishInfo_t pub_info = {
   .qos = MQTT_QOS_1, // Доставка щонайменше один раз
   .topicName = "test/topic",
   .payload = "Hello",    };
uint16_t packet_id =MQTT_GetPacketId(&mqtt_ctx); //Отримання ID
MQTT_Publish(&mqtt_ctx, &pub_info, packet_id); // Відправлення
// ------ Підписка з QoS 2 --------
/** * @brief Підписка на тему з QoS 2
 * @note  QoS 2 - точна доставка один раз
 *       Масив тем може містити кілька фільтрів  */
MQTTSubscribeInfo_t sub_list[] = {
  { .qos = MQTT_QOS_2, .topicFilter = "test/topic" } // QoS 2
};
packet_id = MQTT_GetPacketId(&mqtt_ctx);//Новий ID для підписки
MQTT_Subscribe(&mqtt_ctx, sub_list, 1, packet_id); // Відправка
SUBSCRIBE
// ----- Обробка вхідних пакетів ------
/** * @brief Основний цикл обробки MQTT-пакетів
 * @note MQTT_ProcessLoop() обробляє PUBACK, PUBREC, SUBACK
тощо */
while (1) {
   MQTT_ProcessLoop(&mqtt_ctx, 100); // Обробка вхідних пакетів
}
```

Вибір брокера та рівня QoS є двома ключовими архітектурними рішеннями під час проектування MQTT-системи. Mosquitto забезпечує легку та швидку основу для локальних розгортань, тоді як EMQX пропонує масштабованість хмарного рівня. Водночас рівень QoS 1 є розумним компромісом для більшості IoT-застосувань, тоді як QoS 0 і QoS 2 слід застосовувати виважено, пам'ятаючи про їхні обмеження та накладні витрати. Наступний підрозділ, присвячений захищеному з'єднанню через TLS/DTLS, розкриє, як забезпечується конфіденційність і цілісність MQTT-трафіку поверх транспортного рівня безпеки.

9.1.2. Захищене з'єднання через TLS/DTLS

Модель публікації-підписки MQTT, розглянута в попередньому підрозділі, за замовчуванням передає всі дані, включно з темами, ідентифікаторами клієнтів, паролями та корисним навантаженням, у відкритому вигляді через TCP-з'єднання. В умовах промислового IoT, де через мережу передаються команди керування критичною інфраструктурою, персональні медичні дані або фінансова телеметрія, така незахищеність є категорично неприйнятною. Для забезпечення конфіденційності, цілісності та автентичності MQTT-трафіку специфікація MQTT 5.0 (OASIS, 2019) передбачає обов'язкове використання транспортного рівня безпеки TLS (Transport Layer Security) для з'єднань TCP або DTLS (Datagram Transport Layer Security) для з'єднань UDP, якщо використовується MQTT-SN (MQTT for Sensor Networks). Вибір між TLS і DTLS визначається транспортним протоколом: практично всі реалізації MQTT для вбудованих систем використовують TCP і, відповідно, TLS, тоді як DTLS знаходить застосування в сценаріях із обмеженими ресурсами та високою латентністю, де використовується UDP (наприклад, CoAP+DTLS). Сучасні стандарти TLS 1.3 (RFC 8446, 2018) та DTLS 1.3 (RFC 9147, 2022) забезпечують не лише надійний захист, а й значне скорочення кількості обмінів під час рукостискання (1-RTT, а з PSK — 0-RTT), що особливо важливо для пробудження батарейних IoT-пристроїв, які прагнуть мінімізувати час активної радіопередачі. Як зазначено в дослідженні Н. А. Алурі та співавторів (IEEE Transactions on Industrial Informatics, том 19, 2023), впровадження TLS 1.3 навіть на мікроконтролерах класу ARM Cortex-M4 дозволяє досягти часу рукостискання менше 100 мс та енерговитрат на рівні одиниць міліджоулів, що робить захищене MQTT-з'єднання практично досяжним для масових IoT-пристроїв.

TLS 1.3 став результатом ґрунтовного переосмислення попередніх версій (1.2, 1.1) з метою усунення вразливостей (таких як POODLE, BEAST, Lucky13) та радикального спрощення рукостискання. У TLS 1.3 було вилучено підтримку статичних ключів RSA (замінено на ефемерний обмін Діффі-Хеллмана, DHE або ECDHE), видалено застарілі алгоритми симетричного шифрування (RC4, 3DES, CBC) та хешування (MD5, SHA-1), а також скорочено кількість обмінів повідомленнями з двох (2-RTT) до одного (1-RTT). Клієнт у першому ж повідомленні ClientHello надсилає свої підтримувані набори шифрів (cipher suites), ключі для ECDHE та, за необхідності, параметри PSK (Pre-Shared Key). Сервер відпо-

відає повідомленням ServerHello, яке містить обраний набір шифрів, власний ключ ECDHE, сертифікат (або підтвердження PSK), і вже на цьому етапі обидві сторони мають достатньо інформації для обчислення спільного секрету (master secret). Після цього сервер надсилає зашифроване повідомлення Finished, і клієнт відповідає своїм Finished. Весь подальший трафік шифрується та автентифікується. Для пристроїв, що використовують попередньо розподілені ключі (PSK), TLS 1.3 допускає режим 0-RTT, за якого клієнт може надіслати зашифровані дані додатків уже в першому пакеті після ClientHello, що є унікальною перевагою для датчиків, які прокидаються раз на годину для передавання одного вимірювання. DTLS 1.3 адаптує ці механізми для UDP, додаючи обробку втрат і перевпорядкування пакетів, що є природними для ненадійного транспорту, проте в контексті MQTT (яке спирається на TCP) DTLS зустрічається рідше — переважно в шлюзах MQTT-SN. Структуру рукостискання TLS 1.3 з використанням сертифікатів X.509 наведено на рисунку 9.2.

Основним викликом для вбудованої реалізації TLS є обмежені обчислювальні ресурси та обсяг пам'яті. Повноцінне рукостискання з використанням сертифікатів X.509 (асиметрична криптографія ECDSA на кривій secp256r1, обмін ECDHE, обчислення HMAC-SHA256) вимагає близько 64–128 КБ оперативної пам'яті для стеку TLS та буферів, а також до 256 КБ флеш-пам'яті для бібліотеки (mbedTLS, wolfSSL). Для мікроконтролерів із обмеженим ОЗП, таких як ESP32-C3 (400 КБ SRAM) або STM32L476 (128 КБ SRAM), це цілком прийнятно, однак для найменших пристроїв (STM32F0, ATmega) TLS із сертифікатами є недосяжним, тому в таких випадках використовують спрощену автентифікацію на основі PSK (Pre-Shared Key), яка не потребує операцій з асиметричною криптографією. Режим PSK є надзвичайно легким: клієнт і сервер заздалегідь мають спільний симетричний ключ, який використовується для генерації сеансових ключів, а рукостискання зводиться до перевірки, чи обидві сторони знають цей ключ. MQTT-брокери, такі як Mosquitto, підтримують PSK через плагін автентифікації, хоча більш поширеною є комбінація TLS із зовнішньою автентифікацією (сертифікати або пароль) на рівні протоколу.

Основними відкритими бібліотеками TLS, адаптованими для вбудованих систем, є mbedTLS (розробляється ARM, ліцензія Apache 2.0) та wolfSSL (розробляється wolfSSL Inc., ліцензія GPLv2/комерційна). Обидві підтримують TLS 1.3, DTLS 1.3, PSK, апаратне прискорення криптографії (AES, SHA) на мікроконтролерах STM32 (CRYP, HASH) та ESP32 (вбудований апаратний AES/SHA), а також легко інтегруються зі стеками TCP/IP (LwIP, FreeRTOS+TCP). mbedTLS є стандартом де-факто для STM32,

оскільки постачається у складі пакетів STM32Cube та забезпечує оптимізовані драйвери для апаратних криптографічних модулів.

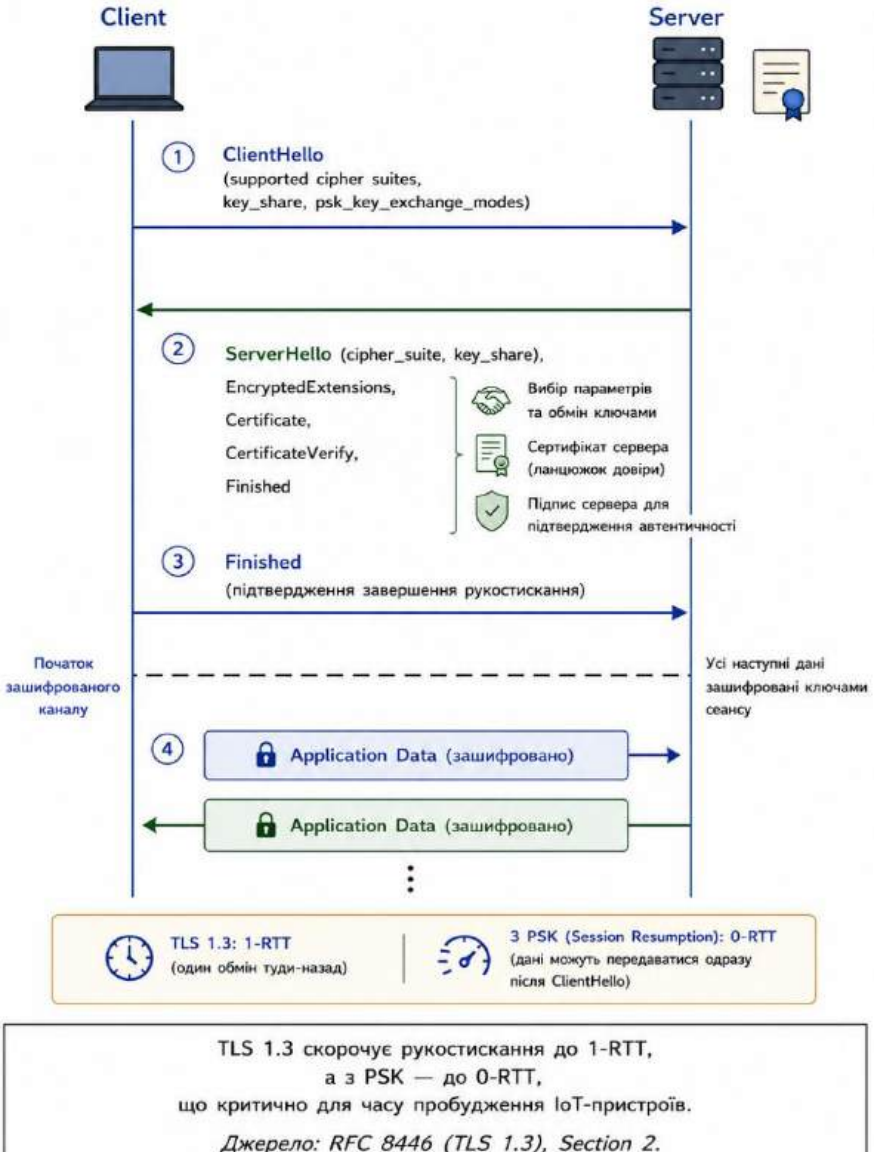


Рисунок 9.2 – Рукостискання TLS 1.3 (1-RTT) з автентифікацією сертифікатами

Для ESP32, який використовує стек LwIP, бібліотека mbedTLS також попередньо інтегрована в ESP-IDF. Порівняння характеристик mbedTLS та wolfSSL наведено в таблиці 9.2.

Таблиця 9.2 – Порівняння бібліотек mbedTLS та wolfSSL для вбудованих систем

Характеристика	mbedTLS (3.x)	wolfSSL (5.x)
Підтримка TLS 1.3	так (повна підтримка з версії 3.6.0)	так
DTLS 1.3	у дорожній карті («Future»), найближчих планів немає	так
PSK	так	так (WOLFSSL_STATIC_PSK)
Апаратне прискорення	так (STM32, ESP32, Silicon Labs)	так (STM32, ESP32-C3/C6/S2, NXP CAAM)
Типовий обсяг Flash (мінімальний)	~60 КБ (типова конфігурація)	20–100 КБ (залежно від конфігурації)
Ліцензія	Apache 2.0 / GPL-2.0-or-later (подвійна)	GPLv2 / комерційна (подвійна)
Інтеграція з MQTT	coreMQTT, Paho Embedded	wolfMQTT, Paho Embedded
Платформи	STM32, ESP32, Pico W, Silicon Labs	STM32, ESP32, Pico W, NXP i.MX

Практична інтеграція TLS з MQTT на STM32 вимагає налаштування контексту TLS, завантаження кореневих сертифікатів (CA certificates) або PSK, та передачі цього контексту в MQTT-клієнт. У лістингу 9.2 наведено фрагмент коду для STM32F4 з використанням бібліотек coreMQTT, mbedTLS та LwIP, де TLS-контекст створюється на основі сертифіката ЦС, а потім використовується для захищеного підключення до брокера з портом 8883 (стандартний порт MQTT+TLS).

Лістинг 9.2 – Налаштування TLS 1.3 для MQTT-клієнта на STM32 (coreMQTT + mbedTLS + LwIP)

```
// Відповідно до mbedTLS API
#include "mbedtls/ssl.h"
#include "mbedtls/net sockets.h"
#include "lwip/sockets.h"
// ------ Кореневий сертифікат (PEM) --------
extern const char root_ca_pem[];   // Масив із кореневим
сертифікатом CA
// ------ 1. Ініціалізація TLS-конфігурації ------
/** * @brief Налаштування TLS 1.3 контексту
 * @note  mbedtls_ssl_conf_authmode() — обов'язкова перевірка
сертифіката
```

```
 * @note  mbedtls_ssl_conf_ca_chain() — завантаження кореневого
CA */
mbedtls_ssl_context ssl;
mbedtls_ssl_config conf;
mbedtls_x509_crt cacert;
mbedtls_ssl_init(&ssl);
mbedtls_ssl_config_init(&conf);
mbedtls_x509_crt_init(&cacert);
// Завантаження кореневого сертифіката
mbedtls_x509_crt_parse(&cacert, (const unsigned char
*)root_ca_pem,
                       strlen(root_ca_pem) + 1);
// Налаштування конфігурації
mbedtls_ssl_config_defaults(&conf,
                            MBEDTLS_SSL_IS_CLIENT,
                            MBEDTLS_SSL_TRANSPORT_STREAM,
                            MBEDTLS_SSL_PRESET_DEFAULT);
mbedtls_ssl_conf_authmode(&conf, MBEDTLS_SSL_VERIFY_REQUIRED);
// Обов'язкова перевірка
mbedtls_ssl_conf_ca_chain(&conf, &cacert, NULL); // Кореневий
CA
mbedtls_ssl_setup(&ssl, &conf);
// ------ 2. Прив'язка до сокету LwIP -------
/** * @brief Підключення TLS до транспортного рівня (LwIP
socket)
 * @note  mbedtls_ssl_set_bio() — прив'язка callback-ів
send/recv до сокету  */
int sock = lwip_socket(AF_INET, SOCK_STREAM, 0);
// ... підключення до сервера ...
mbedtls_ssl_set_bio(&ssl, &sock, // Контекст і сокет
          mbedtls_net_send, // Callback для відправлення
          mbedtls_net_recv, // Callback для прийому
          NULL); // Таймаут (не використовується)
// ----- 3. Виконання TLS-рукопоскання ------
/** * @brief Встановлення захищеного каналу
 * @note mbedtls_ssl_handshake() — виконує TLS 1.3 handshake */
int ret = mbedtls_ssl_handshake(&ssl);
if (ret != 0) {    // Обробка помилки    }
// ------- 4. Використання з coreMQTT ------
/** * @brief Ініціалізація MQTT-контексту з TLS-транспортом
 * @note  Функції MQTT_Connect / MQTT_Publish використовують
mbedtls ssl write/read */
MQTTContext_t mqtt_ctx;
MQTT_Init(&mqtt_ctx, transport_send, transport_recv,
get_time_ms);
// Функції transport send/recv викликають mbedtls ssl write() /
mbedtls_ssl_read()
```

Слід підкреслити, що впровадження TLS на транспортному рівні не звільняє розробника від необхідності захисту на прикладному рівні — автентифікації клієнта, авторизації доступу до тем, валідації корисного навантаження. Однак саме TLS/DTLS забезпечують першу, найважливішу лінію оборони, яка унеможливлює пасивне прослуховування та активні атаки «людина посередині» (MITM) на всьому шляху між клієнтом і брокером.

З урахуванням вимог європейського регламенту Cyber Resilience Act (2025), що набуває чинності для IoT-пристроїв, використання TLS 1.3 із сертифікатами або PSK стає не просто рекомендованим, а юридично обов'язковим для більшості категорій продуктів. Таким чином, освоєння практичного налаштування TLS для MQTT є обов'язковою компетенцією сучасного розробника вбудованих систем. Наступний підрозділ, присвячений CoAP, продемонструє альтернативний прикладний протокол, що працює через UDP і нативно підтримує DTLS, пропонуючи інший баланс між легкістю та безпекою.

9.2. CoAP (Constrained Application Protocol) згідно RFC 7252

Протокол CoAP (Constrained Application Protocol), специфікований IETF у документі RFC 7252 (2014), був створений спеціально для подолання фундаментальних обмежень вбудованих пристроїв, які унеможливлюють пряме використання HTTP. Якщо HTTP спирається на TCP, що потребує встановлення з'єднання, складного керування потоком, великих заголовків і текстового формату повідомлень, то CoAP побудований поверх UDP — легкого, дейтаграмного протоколу без з'єднання, — і використовує компактний бінарний формат повідомлень, придатний для передавання навіть у найбільш обмежених LPWAN-мережах.

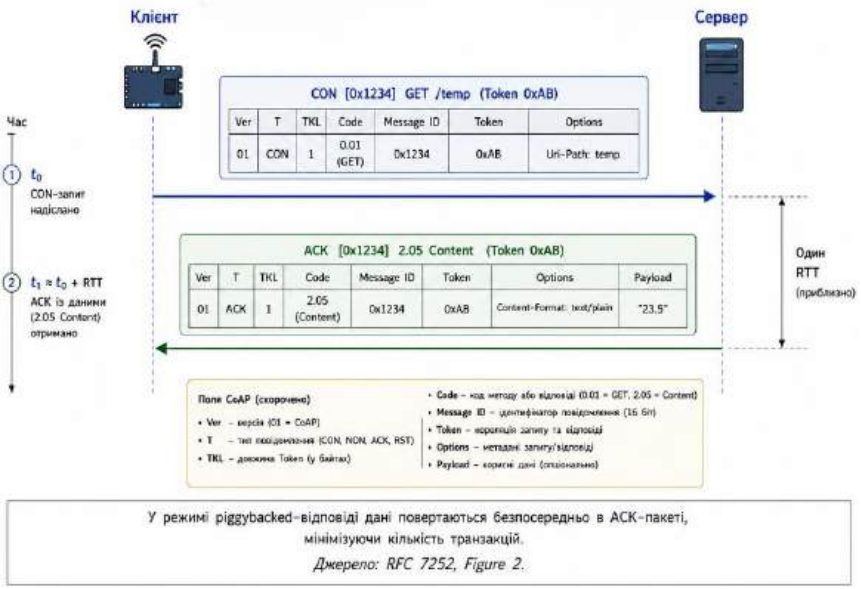
9.2.1. REST-архітектура, модель запит/відповідь поверх UDP

Справжньою інновацією CoAP є не просто заміна транспорту, а перенесення архітектурного стилю REST (Representational State Transfer) у світ мікроконтролерів. Кожен фізичний ресурс пристрою — показання датчика, стан актуатора, конфігураційний параметр — представляється у вигляді веб-подібного ресурсу, ідентифікованого URI (Uniform Resource Identifier), наприклад, coap://[2001:db8::1]/sensors/temperature. Взаємодія з ресурсами здійснюється через стандартний набір методів, знайомий кожному веб-розробнику: GET для читання, POST для створення, PUT для оновлення, DELETE для видалення. Сервер (вбудований пристрій) обробляє ці запити та повертає коди відповідей, аналогічні HTTP-статусам: 2.05 Content при успішному GET, 4.04 Not Found при спробі доступу до неіснуючого ресурсу, 4.00 Bad Request при некоректному запиті. Таким чином, CoAP надає природний міст між обмеженими IoT-мережами та повноцінним веб-середовищем, дозволяючи безпосередньо відображати вбудо-

вані пристрої в простір REST API без важких шлюзів трансляції протоколів. Як зазначено в огляді К. Борманна та співавторів (IEEE Communications Magazine, том 52, 2014), CoAP є ключовим елементом архітектури «Web of Things», що поширює принципи Всесвітньої павутини на фізичний світ.

Модель запит/відповідь у CoAP, хоча й зовні нагадує HTTP, глибоко адаптована до ненадійної природи UDP. Оскільки UDP не гарантує доставки, порядку та відсутності дублікатів, CoAP вводить дворівневу систему надійності на прикладному рівні. Повідомлення поділяються на два типи: Confirmable (CON), які вимагають підтвердження від отримувача, та Non-confirmable (NON), які передаються без підтвердження. Якщо клієнт надсилає CON-запит (наприклад, GET температури), він очікує у відповідь ACK-пакет, що містить або безпосередньо дані (piggybacked response), або сигнал про те, що дані будуть надіслані пізніше окремим CON-повідомленням (separate response). Якщо ACK не отримано протягом таймауту (зазвичай 2–3 секунди, з експоненційним збільшенням), клієнт повторює передавання. Для NON-повідомлень підтвердження не надсилається, що мінімізує трафік для даних, де втрата одного вимірювання не є критичною. Такий механізм дозволяє досягти рівня надійності, зіставного з TCP, але без накладних витрат на постійне з'єднання, що особливо важливо для пристроїв, які більшу частину часу перебувають у режимі сну. Кожне повідомлення CoAP містить 16-бітний Message ID для виявлення дублікатів, а запити та відповіді пов'язуються за допомогою Token — байтової послідовності змінної довжини, яка дозволяє мультиплексувати кілька незалежних запитів в одному UDP-потоці. Формат базового CoAP-повідомлення надзвичайно компактний: фіксований 4-байтовий заголовок (Version, Type, Token Length, Code, Message ID) та опціональні поля Token, Options і Payload. Для порівняння, мінімальний HTTP-запит містить сотні байтів текстових заголовків, тоді як CoAP-запит на читання одного значення може вкластися в 20–30 байтів. Це робить CoAP ідеальним для LPWAN-мереж, де кожен байт на вагу золота. Схематичне зображення обміну CON-запитом із piggybacked-відповіддю наведено на рисунку 9.3.

Хоча обидва протоколи є лідерами прикладного рівня IoT, вони принципово відрізняються за архітектурною моделлю: MQTT є подієво-орієнтованим (event-driven), де брокер активно проштовхує повідомлення підписникам, тоді як CoAP є ресурсно-орієнтованим (resource-driven), де клієнт явно запитує стан ресурсу. Це робить CoAP кращим вибором для сценаріїв, де потрібна інтеграція з існуючими веб-сервісами (REST API), а MQTT — для систем, де критичною є швидка доставка сповіщень багатьом споживачам.



У режимі piggybacked-відповіді дані повертаються безпосередньо в ACK-пакеті, мінімізуючи кількість транзакцій.

Джерело: RFC 7252, Figure 2.

Рисунок 9.3 – Модель запит/відповідь CoAP: CON-запит із piggybacked-відповіддю

Порівняння CoAP із MQTT, яке дозволяє обрати протокол під конкретну задачу, наведено в таблиці 9.3.

Таблиця 9.3 – Порівняння CoAP та MQTT як прикладних протоколів

Характеристика	CoAP (RFC 7252)	MQTT 5.0 (OASIS)
Транспортний протокол	UDP (опціонально DTLS)	TCP (опціонально TLS)
Архітектурна модель	REST (ресурси, URI)	публікація-підписка (брокер)
Модель взаємодії	запит/відповідь (синхронна, асинхронна)	подієво-орієнтована (асинхронна)
Формат повідомлень	бінарний (компактний)	бінарний (MQTT packet)
Механізм надійності	CON/NON + повторні передачі (retransmissions)	QoS 0/1/2 на рівні протоколу
Підтримка QoS	через CON/NON	так (0, 1, 2)
Безпека	DTLS 1.3 (через coaps)	TLS 1.3
Інтеграція з веб	природна (coap:// → http:// через proxy)	через MQTT Bridge або HTTP-to-MQTT плагіни
Типова сфера застосування	сенсорні мережі, розумні лічильники, керування актуаторами	телеметрія, сповіщення, домашня автоматизація

Практичне використання CoAP на вбудованих системах спирається на легковагові бібліотеки, найвідомішою з яких є libcoap (C, ліцензія BSD), що підтримує весь набір функцій RFC 7252, включаючи Observe, Block-wise Transfer та DTLS. Для мікроконтролерів із обмеженими ресурсами часто використовують microcoap або власні реалізації, що вписуються в кілька кілобайт флеш-пам'яті. У лістингу 9.3 наведено приклад створення CoAP-клієнта на ESP32 із використанням бібліотеки libcoap для відправлення GET-запиту на публічний CoAP-сервер.

Лістинг 9.3 – Відправлення CoAP GET-запиту з використанням libcoap на ESP32 (ESP-IDF)

```
// Відповідно до libcoap API
#include "coap3/coap.h"
// ------ Глобальний контекст ------
static coap_context_t *coap_context = NULL;
static coap_session_t *coap_session = NULL;
// ----- Callback для обробки відповіді ------
/** * @brief Обробник відповіді на CoAP-запит
 * @param context  Контекст CoAP
 * @param session  Сесія CoAP
 * @param sent     Надісланий PDU
 * @param received Отриманий PDU (відповідь)
 * @param id       ID повідомлення
 * @note Реєструється через coap_register_response_handler() */
static void response_handler(coap_context_t *context,
coap_session_t *session,
                             coap_pdu_t *sent, coap_pdu_t *received,
                             const coap_tid_t id) {
    if (received) {
        unsigned char *data = NULL;
        size_t data_len = 0;
        coap_get_data(received, &data_len, &data); // Отримання
payload
        // Обробка даних    }   }
// ------ Відправлення GET-запиту ------
/** * @brief Ініціалізація CoAP та відправлення GET-запиту
 * @note  coap_new_context() — створення контексту
 *        coap new session() — створення сесії до сервера
coap://coap.me:5683
 *       coap new pdu() — створення PDU з типом COAP MESSAGE CON
та кодом COAP REQUEST GET
 *        coap add option() — додавання опції URI-PATH
coap send() — відправлення запиту */
void send_coap_get(void) {
    // 1. Створення контексту
    coap_context = coap_new_context(NULL);
    // 2. Створення сесії до сервера coap.me:5683
    coap_address_t server_addr;
    coap_address_init(&server_addr);
    server_addr.addr.sin.sin_family = AF_INET;
    server_addr.addr.sin.sin_port = htons(5683);
    inet_pton(AF_INET, "104.21.4.231", &server_addr.addr.sin.
```

```
42:sin_addr); // IP coap.me
43:    coap_session = coap_new_client_session(coap_context, NULL,
44:&server_addr, COAP_PROTO_UDP);
45:  // 3. Створення PDU (Confirmable, GET)
46:  coap_pdu_t *pdu = coap_new_pdu(coap_session);
47:  pdu->type = COAP_MESSAGE_CON;
48:  pdu->code = COAP_REQUEST_GET;
49:  // 4. Додавання URI-Path: /sensors/temperature
50:  coap_add_option(pdu, COAP_OPTION_URI_PATH, 7, (const uint8_t
51:*)"sensors");
52:  coap_add_option(pdu, COAP_OPTION_URI_PATH, 11, (const uint8_t
53:*)"temperature");
54:  // 5. Реєстрація обробника відповіді
55:  coap_register_response_handler(coap_context,
56:response_handler);
57:  // 6. Відправлення запиту    coap_send(coap_session, pdu);  }
58:// ------ Головний цикл ------
59:void app_main(void) {
60:    // ... ініціалізація Wi-Fi ...
61:    send_coap_get();
62:    while (1) {
63:        coap_io_process(coap_context, 1000); // Обробка вхідних
64:повідомлень
65:        vTaskDelay(pdMS_TO_TICKS(10));    }   }
```

CoAP є елегантним поєднанням простоти UDP, компактності бінарних протоколів та універсальності REST-архітектури, що робить його ідеальним кандидатом для безпосередньої інтеграції обмежених IoT-пристроїв у глобальну веб-екосистему.

9.2.2. Розширення Observe, Block-wise Transfer, Group Communication

Базова модель запит/відповідь CoAP, розглянута в попередньому підрозділі, забезпечує компактний і надійний механізм для точкового опитування ресурсів обмежених пристроїв. Проте реальні IoT-застосування часто вимагають складніших патернів взаємодії, які виходять за межі простого синхронного GET/POST. Датчик температури, що безперервно моніторить середовище, не повинен очікувати періодичного опитування від сервера — він має проактивно сповіщати про зміни стану щойно вони відбуваються, мінімізуючи як трафік, так і латентність. Пристрій, що передає файл прошивки або журнал подій, мусить розбивати великий обсяг даних на частини, придатні для передавання через дейтаграми UDP розміром у кілька сотень байтів. А система освітлення, що одночасно вмикає десятки ламп, потребує групової адресації, аби уникнути лавинної розсилки індивідуальних команд. Для вирішення цих завдань IETF розробив три ключові

розширення CoAP, специфіковані в окремих RFC: Observe (RFC 7641, 2015), Block-wise Transfer (RFC 7959, 2016) та Group Communication for CoAP (RFC 7390, 2014). Ці розширення не змінюють ядро протоколу, а додають опціональні механізми, які можуть бути реалізовані навіть на пристроях із обмеженими ресурсами завдяки компактному бінарному формату опцій.

Розширення Observe трансформує CoAP із протоколу опитування (polling) у протокол сповіщення (push notification). У класичному режимі клієнт, якому потрібно відстежувати зміну ресурсу (наприклад, рівень заряду батареї), змушений періодично надсилати GET-запити, навіть якщо значення не змінилося. Кожен такий запит генерує мережевий трафік, витрачає енергію та затримує виявлення змін на величину періоду опитування. Observe змінює цю парадигму: клієнт надсилає GET-запит із доданою опцією Observe (значення 0), яка сигналізує серверу про бажання підписатися на сповіщення. Сервер негайно відповідає поточним значенням ресурсу в ACK-пакеті (якщо запит був CON) або окремим NON-повідомленням, але на додачу зберігає клієнта у списку спостерігачів (observers). Кожного разу, коли стан ресурсу змінюється, сервер надсилає клієнту нове NON-повідомлення (або CON, залежно від конфігурації) з оновленим представленням ресурсу та тією самою опцією Observe, яка тепер містить порядковий номер сповіщення. Клієнт, отримуючи ці повідомлення, відстежує послідовність і може виявити пропущені оновлення. Для припинення підписки клієнт або надсилає GET без опції Observe, або відповідає RST на отримане сповіщення. Завдяки Observe кількість переданих пакетів скорочується на порядок, оскільки повідомлення генеруються лише за фактом зміни даних, а не за розкладом. Це має подвійний позитивний ефект: зменшується енергоспоживання клієнта (якщо він є спостерігачем і міг би ініціювати опитування) та розвантажується мережа. У поєднанні з CON-повідомленнями Observe забезпечує надійну доставку сповіщень із підтвердженням. Часову діаграму роботи Observe наведено на рисунку 9.4.

Розширення Block-wise Transfer вирішує проблему передавання великих обсягів даних через протокол, який спирається на UDP із обмеженим максимальним розміром дейтаграми (MTU). Типове значення MTU для 6LoWPAN-мереж становить 1280 байт, а для деяких LPWAN — значно менше. Водночас IoT-пристрої потребують передавання файлів прошивки (FOTA), журналів подій, сертифікатів або навіть зображень. RFC 7959 вводить дві нові опції: Block1 (для тіла запиту) та Block2 (для тіла відповіді),

які дозволяють фрагментувати велике тіло повідомлення на блоки фіксованого розміру (16, 32, 64, 128, 256, 512 або 1024 байти) і передавати їх послідовно.

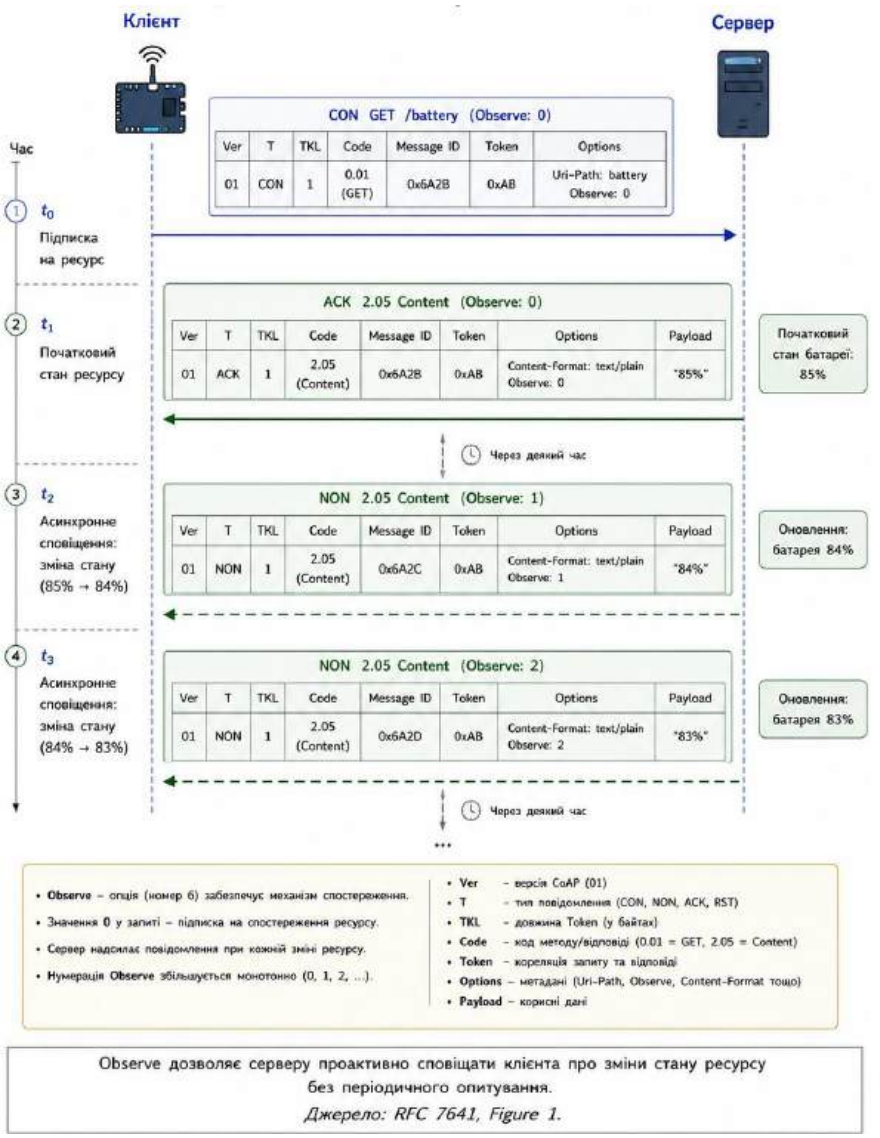


Рисунок 9.4 – Розширення CoAP Observe: підписка та асинхронні сповіщення

Процес є адаптивним: клієнт і сервер можуть узгоджувати розмір блоку (SZX — Size eXponent) під час обміну, зменшуючи його, якщо проміжні маршрутизатори фрагментують або втрачають великі дейтаграми. Опція Block2, яка використовується для великих відповідей, має три підполя: NUM (номер блоку), M (More — прапорець, що вказує на наявність наступних блоків) та SZX (експонента розміру блоку). Клієнт надсилає GET-запит із порожньою опцією Block2 (NUM=0, SZX=пропонований).

Сервер у відповіді повертає перший блок даних, заповнюючи Block2 конкретними значеннями NUM=0, M=1 (якщо є продовження) та SZX=фактичний. Клієнт потім надсилає наступний GET із Block2, де NUM=1, і так далі, допоки не отримає блок із M=0. Аналогічно Block1 використовується для великих PUT/POST-запитів, де клієнт передає дані частинами. Цей механізм є ідемпотентним: клієнт може повторно запитати конкретний блок, якщо попередня спроба була невдалою. Для спрощення навігації CoAP також підтримує опцію Size1/Size2, яка вказує загальний очікуваний розмір тіла запиту або відповіді, дозволяючи приймачу заздалегідь оцінити кількість блоків і виділити буфер. У таблиці 9.4 наведено порівняльний аналіз трьох розширень CoAP.

Розширення Group Communication (RFC 7390) дозволяє CoAP-клієнту адресувати запит одразу групі пристроїв, використовуючи IP-мультикаст. У стандартному REST-стилі URI ідентифікує конкретний ресурс на конкретному хості. Групова комунікація розширює це поняття: URI може вказувати на групу пристроїв, наприклад, *coap://[ff02::1]/lighting/switch, де адреса ff02::1* є мультикаст-адресою (all nodes link-local). Клієнт надсилає один мультикаст-запит (зазвичай NON, оскільки підтвердження від десятків пристроїв призвели б до імплозії), і всі пристрої, що підписані на цю мультикаст-адресу та порт, обробляють запит.

Для уникнення колізій відповідей RFC 7390 рекомендує використовувати NON-повідомлення з опцією Multicast-Signaling, що інкапсулює групові параметри, або взагалі не надсилати відповідей (fire-and-forget). Цей механізм надзвичайно ефективний для керування групами однотипних пристроїв — освітленням, жалюзі, вентиляцією в розумному будинку, — де одна команда «увімкнути все» має бути доставлена десяткам вузлів одночасно, без індивідуального опитування.

Таблиця 9.4 – Порівняння розширень CoAP: Observe, Block-wise Transfer, Group Communication

Характеристика	Observe (RFC 7641)	Block-wise Transfer (RFC 7959)	Group Communication (RFC 7390)
Призначення	асинхронне сповіщення клієнтів про зміни стану ресурсу (підписка на спостереження)	передавання великих тіл повідомлень, що не вміщуються в один дейтаграмний пакет (наприклад, FOTA, журнали, сертифікати)	одночасна адресація та керування групою пристроїв (наприклад, вмикання / вимикання всіх світильників у кімнаті)
Механізм	клієнт надсилає GET-запит із опцією Observe; сервер надсилає початкову відповідь і наступні сповіщення (NON/CON) зі зростаючим номером послідовності для виявлення перевпорядкування	фрагментація представлення ресурсу на блоки фіксованого розміру; обмін відбувається в кількох парах запит-відповідь з використанням опцій Block1 / Block2; сервер може бути повністю безстатевим	використання UDP/IP-мультикасту як основного транспорту; клієнт надсилає multicast-запит, усі учасники групи отримують його та можуть відповідати (за потреби)
Опції CoAP	Observe (номер опції 6)	Block1 (27) – для тіла запиту (PUT/POST); Block2 (23) – для тіла відповіді (GET); додатково Size1 / Size2	використання стандартних опцій CoAP у поєднанні з multicast-адресацією; групова адреса визначається URI; опціональне multicast-сигналізування
Типовий сценарій	моніторинг датчиків у реальному часі (температура, вологість, рівень заряду тощо)	оновлення прошивки (FOTA), завантаження журналів подій, обмін сертифікатами	керування освітленням, синхронізація стану, масова конфігурація пристроїв у будівельній автоматизації

Кінець таблиці 9.4

Характеристика	Observe (RFC 7641)	Block-wise Transfer (RFC 7959)	Group Communication (RFC 7390)
Обмеження	сповіщення надсилаються за принципом "best effort"; затримка залежить від частоти змін ресурсу; клієнт повинен обробляти можливу втрату або перевпорядкування повідомлень	збільшення кількості запитів (N блоків = N запитів); додаткові накладні витрати на кожен блок; потребує підтримки з боку сервера	відсутність гарантії доставки всім учасникам (UDP-мультикаст не є надійним); обмеження на розмір відповідей через можливе перевантаження мережі

Практичне використання Observe та Block-wise Transfer часто поєднується. Типовим прикладом є процедура оновлення прошивки (FOTA) через CoAP: клієнт спочатку підписується через Observe на ресурс /firmware/status, щоб отримувати сповіщення про доступність нової версії; отримавши таке сповіщення, він ініціює завантаження образу через GET /firmware/image із використанням Block2, отримуючи прошивку блоками по 1024 байти, перевіряючи цілісність кожного блоку та, за потреби, запитуючи повторно. У лістингу 9.4 наведено фрагмент коду на libcoap, що демонструє реєстрацію Observe-спостерігача на ресурсі.

Лістинг 9.4 – Реєстрація Observe-спостерігача на CoAP-сервері (libcoap, C)

```
1:// Відповідно до RFC 7641 та libcoap API
2:#include <coap3/coap.h>
3:// ------ Глобальний контекст та сесія ------
4:static coap_context_t *coap_context = NULL;
5:static coap_session_t *coap_session = NULL;
6:// ------ Обробник відповідей (Observe) ------
7:/** * @brief Callback для обробки відповідей на Observe-запит
8: * @param context Контекст CoAP
9: * @param session  Сесія CoAP
10: * @param sent     Надісланий PDU
11: * @param received Отриманий PDU (відповідь/сповіщення)
12: * @param id       ID повідомлення
13: * @note  Реєструється через coap_register_response_handler()
14:* При отриманні сповіщення анал-ся токен та опція Observe */
15:static void observe_response_handler(coap_context_t *context,
16:coap_session_t *session,
17:                    coap_pdu_t *sent, coap_pdu_t *received,
18:                                    const coap_tid_t id) {
```

```
    if (!received) return;
    coap_opt_iterator_t opt_iter;
    coap_opt_t *observe_opt;
// 1. Перевірка наявності опції Observe у відповіді
 observe_opt = coap_check_option(received, COAP_OPTION_OBSERVE,
&opt_iter);
    if (observe_opt) {
   // 2. Отримання значення Observe (послідовний номер)
        uint32_t observe_value = coap_opt_value(observe_opt);
   // 3. Отримання токена для ідентифікації підписки
        coap_bin_const_t token = coap_pdu_get_token(received);
   // 4. Оновлення локального значення ресурсу
        unsigned char *data = NULL;        size_t data_len = 0;
        coap_get_data(received, &data_len, &data);
        // Обробка отриманих даних (оновлення локального стану)
        // ...     }   }
// ------ Реєстрація Observe-спостерігача ------
/** * @brief Встановлення Observe-підписки на ресурс
 * @note  Створення GET-запиту з опцією COAP_OPTION_OBSERVE
 *   Значення опції: COAP OBSERVE ESTABLISH (0) для встановлення
Підписки. Реєстрація обробника відповідей через
coap_register_response_handler() * Після успішної реєстрації
сервер надсилатиме сповіщення при зміні ресурсу */
void coap_observe_resource(void) {
    // 1. Створення контексту та сесії (як у лістингу 9.3)
    coap_context = coap_new_context(NULL);
    // ... налаштування сесії ...
    // 2. Створення PDU (Confirmable, GET)
    coap_pdu_t *pdu = coap_new_pdu(coap_session);
    pdu->type = COAP_MESSAGE_CON;
    pdu->code = COAP_REQUEST_GET;
    // 3. Додавання опції Observe (встановлення підписки)
    uint8_t observe_value = COAP_OBSERVE_ESTABLISH; // 0 -
встановити підписку
    coap_add_option(pdu, COAP_OPTION_OBSERVE,
            coap_encode_var_bytes((uint8_t *)&observe_value,
1),
                    (const uint8_t *)&observe_value);
     // 4. Додавання URI-Path (наприклад, /sensors/temperature)
      coap_add_option(pdu, COAP_OPTION_URI_PATH, 7, (const
uint8_t *)"sensors");
      coap_add_option(pdu, COAP_OPTION_URI_PATH, 11, (const
uint8_t *)"temperature");
       // 5. Реєстрація обробника відповідей
       coap_register_response_handler(coap_context, observe_
response_handler);
  // 6. Відправлення запиту    coap_send(coap_session, pdu);  }
// ------ Головний цикл ------
void app_main(void) {
    coap_observe_resource();
    while (1) {
        coap_io_process(coap_context, 1000); // Обробка вхідних
сповіщень    }   }
```

Розширення Observe, Block-wise Transfer та Group Communication перетворюють CoAP із простого протоколу «запит-відповідь» на повнофункціональний прикладний фреймворк, здатний вирішувати широкий спектр задач IoT — від потокового моніторингу до масової конфігурації та надійного передавання великих файлів. Їхня легка інтеграція в існуючі CoAP-бібліотеки та відповідність REST-архітектурі роблять CoAP одним із найгнучкіших протоколів для обмежених середовищ. Наступний підрозділ, присвячений OPC UA, продемонструє контрастний підхід до програмних інтерфейсів, орієнтований на промислову інтероперабельність та складне інформаційне моделювання.

9.3. OPC UA для Industry 4.0

У той час як MQTT та CoAP, розглянуті в попередніх підрозділах, орієнтовані на легковагову, горизонтальну комунікацію між обмеженими пристроями та хмарними сервісами, промисловий Інтернет речей (IIoT) висуває значно жорсткіші вимоги до інтероперабельності, семантичної насиченості даних та безпеки. Концепція Industry 4.0 передбачає безперешкодну взаємодію між датчиками, контролерами, SCADA-системами, MES (Manufacturing Execution Systems), ERP (Enterprise Resource Planning) та хмарними аналітичними платформами. Для цього недостатньо просто передати число з датчика — необхідно передати його у контексті: одиниці вимірювання, часову мітку, якість даних, діапазон допустимих значень, зв’язок із фізичним процесом (наприклад, що це температура підшипника конкретного двигуна в конкретному цеху). Саме для вирішення цього класу задач консорціумом OPC Foundation було розроблено стандарт OPC UA (Open Platform Communications Unified Architecture), специфікований у серії стандартів IEC 62541.

9.3.1. Клієнт-серверна модель та публікація-підписка (PubSub)

OPC UA не є просто ще одним протоколом передавання даних — це цілісна платформа для міжмашинної взаємодії, яка поєднує дві взаємодоповнюючі комунікаційні моделі: традиційну клієнт-серверну (Client-Server) для структурованого доступу до поточних та історичних даних, а також модель публікації-підписки (Publish-Subscribe, PubSub), введену в оновленні IEC 62541-14 (2018), для масштабованої потокової передавання телеметрії від багатьох джерел до багатьох споживачів. Таке поєднання дозволяє одному пристрою одночасно виступати в ролі сервера, що

надає детальний структурований доступ до свого адресного простору, і в ролі видавця, що потоково публікує критичні до часу дані для сотень підписників без накладних витрат на встановлення сесій. Як зазначено в монографії В. Манке та співавторів «OPC Unified Architecture» (Springer, 2021), саме цей дуалізм робить OPC UA єдиним стандартом, здатним забезпечити як вертикальну інтеграцію (від датчика до хмари), так і горизонтальну (між виробничими лініями) в рамках Industry 4.0.

Клієнт-серверна модель OPC UA є історично першою та найбільш фундаментальною. Вона базується на TCP/IP (або HTTPS) і передбачає встановлення сеансу (Session) між клієнтом і сервером, у межах якого виконуються аутентифікація, авторизація, шифрування та обмін повідомленнями. Сервер OPC UA надає клієнтам доступ до свого адресного простору (Address Space) — графу вузлів, який моделює всі дані, події, методи та їхні взаємозв'язки. Кожен вузол має унікальний ідентифікатор (NodeId), належить до певного класу (Object, Variable, Method, ObjectType тощо) та містить атрибути (значення, мітка часу, якість, опис). Клієнт може виконувати читання (Read), запис (Write), виклик методів (Call), перегляд ієрархії (Browse), а також підписуватися на моніторинг змін окремих змінних через механізм Subscriptions та Monitored Items. Підписка дозволяє клієнту не опитувати сервер періодично, а отримувати сповіщення у разі зміни значення або за таймером, із гарантованою доставкою та відновленням після розриву з'єднання. Ця модель ідеально підходить для SCADA-систем, HMI-панелей та інструментів конфігурування, де оператору потрібен інтерактивний доступ до поточних параметрів обладнання.

Модель публікації-підписки (PubSub), додана в OPC UA для задоволення потреб масової міжмашинної комунікації (M2M), принципово відрізняється від клієнт-серверної. PubSub розриває жорсткий зв'язок «один клієнт — один сервер» і дозволяє одному видавцю (Publisher) надсилати повідомлення одразу багатьом підписникам (Subscribers) без встановлення сеансів, використовуючи легковагові транспортні протоколи — UDP (UADP, OPC UA UDP Multicast) або MQTT (JSON або бінарний кодування). Видавець формує DataSetMessage — повідомлення, що містить значення однієї або кількох змінних із часовими мітками та статусом якості, — і публікує його в мережу або на брокер MQTT. Підписники, знаючи структуру DataSet (DataSetMetaData), можуть приймати ці повідомлення та оновлювати свої локальні копії даних. Ключовою перевагою PubSub є відсутність необхідності для видавця знати, хто саме підписаний на його дані; розподіл виконується мережевою інфраструктурою (для UDP Multicast) або брокером MQTT. Крім того, PubSub підтримує різні рівні якості обслуговування,

включаючи циклічну передачу з фіксованим інтервалом (циклічний PubSub), що є критичним для замкнених контурів керування. Схематичне зображення обох моделей у єдиній системі наведено на рисунку 9.5.

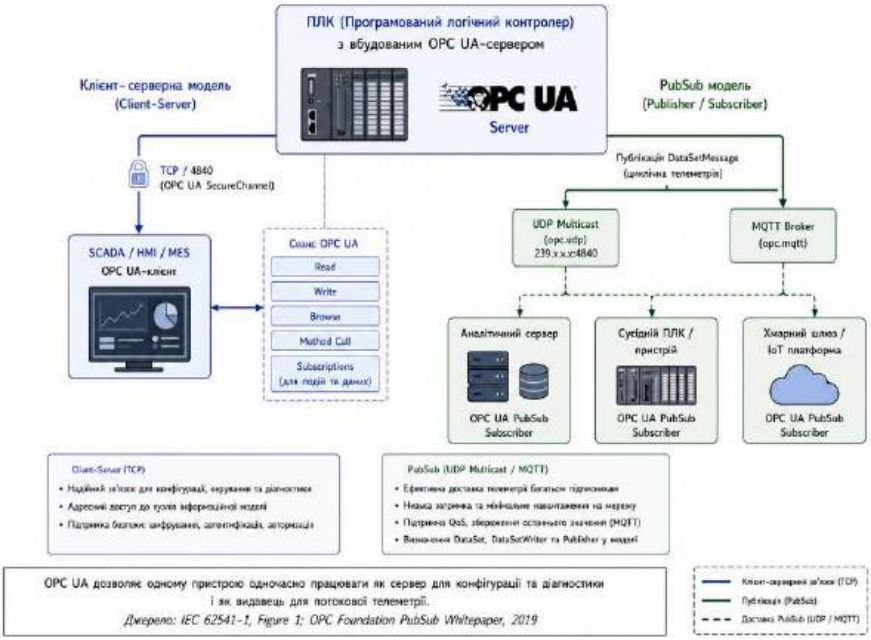


Рисунок 9.5 – Клієнт-серверна та PubSub моделі OPC UA в єдиній системі

Порівняння двох моделей за ключовими характеристиками наведено в таблиці 9.5, яка дозволяє обрати потрібний механізм залежно від задачі.

Практичне впровадження OPC UA PubSub часто спирається на брокер MQTT як транспортний рівень, що дозволяє використовувати всю потужність екосистеми MQTT (масштабовані брокери, хмарні інтеграції) у поєднанні з семантичною насиченістю OPC UA. Наприклад, видавець на ПЛК публікує DataSetMessage у форматі UADP (бінарний, компактний) на тему MQTT, а хмарний підписник або інший ПЛК підписується на цю тему та десеріалізує дані, використовуючи заздалегідь відомі DataSetMetaData. У лістингу 9.5 наведено фрагмент коду на Python із використанням бібліотеки opcua-asyncio, який демонструє налаштування PubSub-видавця, що публікує значення змінної «Temperature» через MQTT-брокер.

Таблиця 9.5 – Порівняння клієнт-серверної та PubSub

Характеристика	Client-Server (IEC 62541-4)	PubSub (IEC 62541-14)
Зв'язок між вузлами	один-до-одного (point-to-point) між клієнтом і сервером	один-до-багатьох (one-to-many) між видавцем і підписниками
Транспорт	TCP (opc.tcp://), HTTPS (opc.https://)	UDP (UADP), MQTT, AMQP; підтримує broker-less (UDP/IP, Ethernet multicast) та broker-based (MQTT, AMQP) моделі
Модель даних	повний адресний простір (Address Space), сеанси (Sessions), підписки (Subscriptions), контрольовані елементи (MonitoredItems)	DataSet (фіксований набір змінних), DataSetMetaData для опису вмісту та семантики
Встановлення з'єднання	так (встановлення сеансу, автентифікація)	ні (повідомлення передаються без сеансу; видавець не знає про підписників)
Типове застосування	SCADA, конфігурування, діагностика, історичні дані, керування та доступ за запитом	масова телеметрія, M2M, зам. контурів керування, хмарна аналітика, потокові дані в реальному часі

Лістинг 9.5 – Налаштування OPC UA PubSub видавця через MQTT (Python, opcua-asyncio)

```
# Відповідно до OPC UA Part 14 (PubSub) та asyncua v1.1+
import asyncio
from asyncua import Server, ua
from asyncua.pubsub.publisher import Publisher
from asyncua.pubsub.information_model import (
    PubSubConnectionConfig,
    WriterGroupConfig,
    DataSetWriterConfig,
    PublishedDataSetConfig,
    PublishedVariableConfig,    )
async def main():
# 1. Створення OPC UA сервера та додавання змінної "Temperature"
    server = Server()
    await server.init()
    server.set_endpoint("opc.tcp://0.0.0.0:4840/freeopcua/
server/")
    ns = await server.register_namespace("urn:example:line1")
    folder = await server.nodes.objects.add_folder(ns, "Line1")
    temp = await folder.add_variable(ns, "Temperature", 72.4)
    await temp.set_writable()
# 2. PublishedDataSet — визначення даних для публікації
    pds = PublishedDataSetConfig(
        name="Line1Telemetry",
```

```
        published_variables=[
            PublishedVariableConfig(
                published_variable=temp.nodeid,
                attribute_id=ua.AttributeIds.Value,
                field_name="Temperature",
            ),        ],    )
# 3. PubSubConnection — підключення до MQTT-брокера
    conn_cfg = PubSubConnectionConfig(
        name="mqtt-broker-1",
     publisher_id=ua.Variant("plc-01", ua.VariantType.String),
        transport_profile_uri=(
"http://opcfoundation.org/UA-Profile/Transport/pubsub-mqtt-
json"        ),
    address_url="mqtt://broker.hivemq.com:1883", # MQTT-брокер
    )
# 4. WriterGroup — цикл публікації 1 секунда
    wg_cfg = WriterGroupConfig(
        name="Line1Group",
        writer_group_id=1,
        publishing_interval=1000.0,  # 1 секунда
        keep_alive_time=5000.0,
        message_settings={
            "NetworkMessageContentMask": 0x0000003F,
            "DataSetMessageContentMask": 0x00000007,    },   )
# 5. DataSetWriter — публікація на тему opcua/temperature
    dsw_cfg = DataSetWriterConfig(
        name="Line1Writer",
        data_set_writer_id=1,
        data_set_name="Line1Telemetry",
        queue_name="opcua/temperature",  # MQTT-тема
        data_set_field_content_mask=(
            ua.DataSetFieldContentMask.StatusCode |
            ua.DataSetFieldContentMask.SourceTimestamp   ),   )
# 6. Збірка та запуск Publisher
    pubsub = Publisher(server)
    await pubsub.add_published_data_set(pds)
    connection = await pubsub.add_connection(conn_cfg)
    writer_group = await connection.add_writer_group(wg_cfg)
    await            writer_group.add_data_set_writer(dsw_cfg,
pds_name="Line1Telemetry")
    async with server:
        await pubsub.start()
# 7. Оновлення значення змінної в головному циклі
        i = 0
        while True:
            await temp.write_value(72.4 + (i % 10) * 0.1)
            i += 1
            await asyncio.sleep(1.0)
if __name__ == "__main__":
    asyncio.run(main())
```

Поєднання клієнт-серверної та PubSub моделей в OPC UA забезпечує унікальну гнучкість, якої бракує окремим протоколам. Виробнича лінія може використовувати PubSub для швидкої потокової передавання даних

між контролерами, одночасно надаючи SCADA-системі детальний клієнт-серверний доступ для візуалізації та ручного керування. Наступний підрозділ, присвячений інформаційному моделюванню в OPC UA, розкриє, як стандартизовані типи даних (Companion Specifications) перетворюють розрізнені сигнали на уніфіковану, машинно-читабельну модель виробничого процесу.

9.3.2. Інформаційне моделювання, адресний простір, підтримка TSN

Сила OPC UA як комунікаційної платформи для Industry 4.0 полягає не лише в гнучких транспортних моделях, розглянутих у попередньому підрозділі, але насамперед у його здатності до інформаційного моделювання — формального, машинно-читабельного опису не просто даних, а їхньої семантики, структури та взаємозв'язків. У той час як MQTT передає непрозорі бінарні блоки (payload), чия інтерпретація повністю покладається на прикладний рівень, а CoAP оперує ресурсами без вбудованого механізму опису їхнього типу, OPC UA надає потужний метамодельний апарат, який дозволяє виробникові обладнання один раз формально описати свій пристрій (його змінні, методи, події, їхні типи, діапазони, одиниці вимірювання, ієрархічні зв'язки), а споживачеві даних — автоматично виявити цей опис (discovery) та інтерпретувати дані без жодної попередньої домовленості, окрім спільного розуміння стандарту OPC UA. Це досягається через адресний простір (Address Space) — центральне поняття специфікації IEC 62541-3, — який є не просто списком змінних, а повнозв'язним орієнтованим графом вузлів (Nodes), з'єднаних посиланнями (References). Кожен вузол у цьому графі належить до одного з базових класів (Object, Variable, Method, View, DataType, ReferenceType) та має набір атрибутів, включно з унікальним ідентифікатором (NodeId), символьною назвою (BrowseName), відображуваною назвою (DisplayName), описом (Description) та, для змінних, значенням (Value), типом даних (DataType) і міткою часу (SourceTimestamp). Завдяки посиланням вузли можуть організовуватися в ієрархії (HasComponent, Organizes), спеціалізуватися через наслідування типів (HasSubtype), посилатися на інші стандарти (HasDictionaryEntry) або описувати фізичні зв'язки між компонентами (HasConnector). Така насиченість робить адресний простір OPC UA не просто інтерфейсом для читання значень, а цифровим двійником (Digital Twin) фізичного активу, придатним для автоматизованого аналізу, симуляції та діагностики.

Приклад фрагменту адресного простору для простого датчика температури, змодельованого відповідно до OPC UA for Devices, наведено на рисунку 9.6.

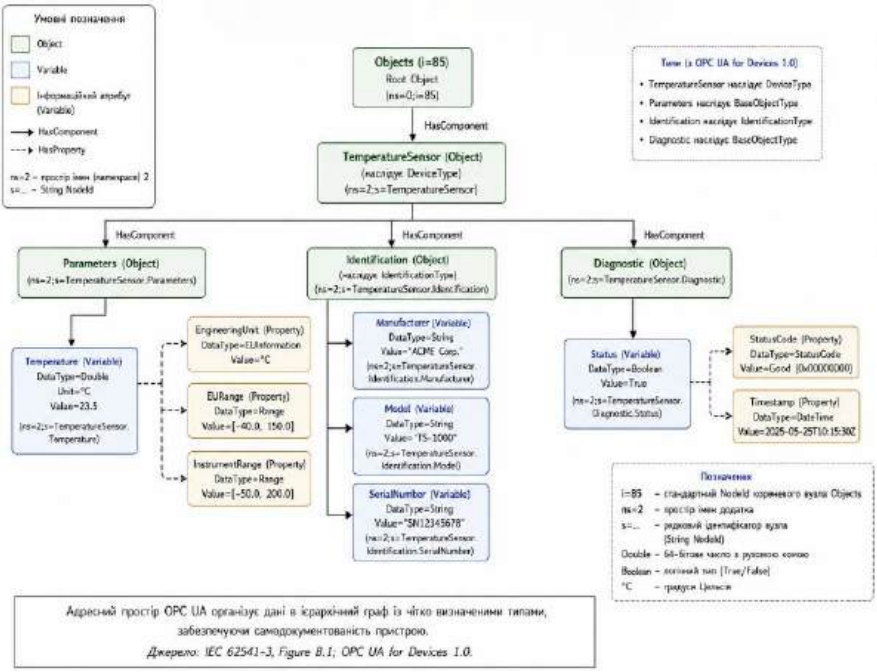


Рисунок 9.6 – Фрагмент адресного простору OPC UA для датчика температури (спрощено)

Ключову роль у забезпеченні інтероперабельності між пристроями різних виробників відіграють стандартизовані інформаційні моделі — Companion Specifications, що розробляються OPC Foundation спільно з галузевими консорціумами. Companion Specification визначає набір типів об'єктів (ObjectTypes), типів змінних (VariableTypes), типів даних (DataTypes) та подій (EventTypes), які є спільними для цілого класу пристроїв. Наприклад, специфікація OPC UA for Robotics (OPC 40010-1) описує стандартний інтерфейс для роботів: об'єкт Robot містить змінні CurrentJointPosition, CurrentTool, методи MoveToHome, EmergencyStop, події CollisionWarning. Специфікація OPC UA for Machine Tools (OPC 40001-1) уніфікує представлення верстатів із ЧПК, включаючи шпиндель, інструментальний магазин, систему охолодження. OPC UA for Devices (IEC 62541-100) надає базову модель для будь-якого польового

пристрою, включаючи ідентифікацію (Manufacturer, Model, SerialNumber), параметри комунікації та діагностичні дані. Коли пристрій, що реалізує певну Companion Specification, підключається до мережі, клієнт (наприклад, SCADA або MES) може через Browse автоматично виявити всі його вузли та однозначно інтерпретувати їх, знаючи, що змінна з BrowseName "CurrentTool" типу ToolIdentifierType в об'єкті, інстанційованому з RobotType, містить саме ідентифікатор поточного інструменту робота.

У таблиці 9.6 наведено приклади найважливіших Companion Specifications станом на 2026 рік, які формують фундамент інтероперабельності в різних галузях промисловості.

Інформаційна насиченість OPC UA висуває підвищені вимоги до транспортного рівня, особливо в сценаріях, де потрібна одночасна передача великих обсягів семантично збагачених даних із гарантованою латентністю. Саме тому OPC UA було обрано як один із ключових прикладних протоколів, що інтегрується з Time-Sensitive Networking (TSN), розглянутим у підрозділі 5.2.3. Специфікація OPC UA Part 14 (PubSub) та технічний звіт OPC UA over TSN (OPC Foundation, 2020) описують, як OPC UA PubSub може використовувати TSN-мережу як детермінований транспорт для потоків даних реального часу. У такій конфігурації видавець (наприклад, ПЛК) публікує DataSetMessage, яке через UDP/IP потрапляє на TSN-комутатор. Time-Aware Shaper (IEEE 802.1Qbv) гарантує, що цей кадр буде відправлений у строго визначеному часовому вікні, без конкуренції з фоновим трафіком, а синхронізація gPTP (IEEE 802.1AS) забезпечує, що всі вузли мережі мають єдиний час із точністю до сотень наносекунд. Це дозволяє OPC UA PubSub досягати наскрізної латентності менше 1 мс із джиттером менше 10 мкс, що є достатнім для замикання контурів керування рухом (Motion Control) через стандартну Ethernet-мережу.

Таблиця 9.6 – Ключові Companion Specifications OPC UA для Industry 4.0

Специфікація	Характеристика	Значення
OPC UA for Devices (IEC 62541-100)	*Галузь*	універсальна (базова модель для будь-якого пристрою)
	Ключові типи об'єктів	DeviceType, IdentificationType, DiagnosticType, DeviceHealth
	Призначення	забезпечує уніфіковане представлення пристроїв, їх апаратних та програмних компонентів, незалежно від базового протоколу; стандартизує ідентифікацію, діагностику та конфігурацію

Кінець таблиці 9.6

Специфікація	Характеристика	Значення
OPC UA for Robotics (OPC 40010-1)	*Галузь*	робототехніка
	Ключові типи об'єктів	RobotType, ControllerType, AxisType, PowerTrainType
	Призначення	стандартизований інформаційний інтерфейс для промислових та мобільних роботів, додаткових осей та контролерів; AxisType описує вісь рухового пристрою; ControllerType описує блок керування
OPC UA for Machine Tools (OPC 40001-1)	*Галузь*	металообробка, верстатобудування
	Ключові типи об'єктів	MachineToolType, SpindleType, ToolMagazineType, MachineToolIdentificationType
	Призначення	уніфіковане представлення верстатів з ЧПК; MachineToolType є точкою входу до інтерфейсу верстата; забезпечує обмін інформацією між верстатом та MES/SCADA/ERP-системами
OPC UA for Process Automation (PA-DIM)	*Галузь*	процесна автоматизація (хімічна, нафтогазова промисловість)
	Ключові типи об'єктів	FlowMeterType, PressureTransmitterType, TemperatureTransmitterType, LevelTransmitterType
	Призначення	модель інформації для процесних вимірювальних приладів (витратоміри, датчики тиску, температури, рівня, позиціонери клапанів); забезпечує протокольно-незалежний доступ до параметрів; реалізує вимоги NAMUR NE 131
OPC UA for Energy Management (IEC 62541-8 / OPC 30141)	*Галузь*	енергетика, енергоменеджмент
	Ключові типи об'єктів	EnergySourceType, BatteryType, EnergyMeasurementType
	Призначення	моніторинг та керування енергоспоживанням; EnergySourceType визначає джерело енергії (фотовольтаїка, вітер тощо); базується на PROFIenergy

Практична реалізація OPC UA over TSN потребує спеціалізованого стеку, який доступний у вигляді комерційних бібліотек (наприклад, Softing, Unified Automation) або відкритих реалізацій (open62541 із розширеннями для PubSub). У лістингу 9.6 наведено фрагмент коду на C із використанням бібліотеки open62541, який демонструє створення PubSub-видавця з публікацією через UDP Multicast, що може бути розгорнутий у TSN-мережі.

Лістинг 9.6 – Створення OPC UA PubSub видавця через UDP Multicast (open62541, C)

```
// Відповідно до open62541 API та OPC UA Part 14 (PubSub)
#include <open62541/server.h>
#include <open62541/server_pubsub.h>
#include <open62541/plugin/pubsub_udp.h>
// ------ 1. Ініціалізація сервера ------
/** * @brief Створення сервера та додавання змінної "Speed" до
адресного простору */
UA_Server *server = UA_Server_new();
UA_ServerConfig *config = UA_Server_getConfig(server);
UA_ServerConfig_setDefault(config);
// Додавання змінної "Speed" (наприклад, під об'єктом "Objects")
UA_NodeId speedNodeId;
UA_VariableAttributes attr = UA_VariableAttributes_default;
attr.displayName = UA_LOCALIZEDTEXT("en-US", "Speed");
attr.dataType = UA_TYPES[UA_TYPES_DOUBLE].typeId;
UA_Server_addVariableNode(server, UA_NODEID_NULL,
                  UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                      UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
                           UA_LOCALIZEDTEXT("en-US", "Speed"),
                           UA_ATTRIBUTEID_DISPLAYNAME,
                           attr, NULL, &speedNodeId);
// ------ 2. Налаштування PubSub-конфігурації -----
/** * @brief Створення PublishedDataSet (PDS) та додавання поля
даних
 * @note UA_Server_addPublishedDataSet() — створення PDS
UA_Server_addDataSetField() — додавання змінної до PDS  */
UA_PublishedDataSetConfig pdsConfig;
memset(&pdsConfig, 0, sizeof(pdsConfig));
pdsConfig.publishedDataSetType     =     UA_PUBSUB_DATASET_PUB-
LISHEDIEMS;
pdsConfig.name = UA_STRING("SpeedPDS");
UA_NodeId pdsNodeId;
UA_Server_addPublishedDataSet(server, &pdsConfig, &pdsNodeId);
UA_DataSetFieldConfig dsfConfig;
memset(&dsfConfig, 0, sizeof(dsfConfig));
dsfConfig.dataSetFieldType = UA_PUBSUB_DATASETFIELD_VARIABLE;
dsfConfig.field.variable.fieldNameAlias = UA_STRING("Speed");
dsfConfig.field.variable.publishParameters.publishedVariable =
speedNodeId;
UA_Server_addDataSetField(server, pdsNodeId, &dsfConfig, NULL);
// ----- 3. Налаштування WriterGroup ------
/** * @brief Створення WriterGroup із циклом публікації 1 мс
(для TSN)
 * @note  publishingInterval = 1.0 (мс) — для TSN-мереж
Для роботи в TSN необхідно налаштувати пріоритети трафіку
(VLAN PCP та використовувати TSN-сумісний мережевий стек */
UA_WriterGroupConfig wgConfig;
memset(&wgConfig, 0, sizeof(wgConfig));
wgConfig.name = UA_STRING("SpeedWG");
wgConfig.publishingInterval = 1.0;  // 1 мс (для TSN)
wgConfig.keepAliveTime = 100.0;
wgConfig.writerGroupId = 100;
UA_NodeId wgNodeId;
UA_Server_addWriterGroup(server, pdsNodeId, &wgConfig,
```

```
&wgNodeId);
// ----- 4. Налаштування DataSetWriter -----
/** * @brief Додавання DataSetWriter із транспортом UADP та
мультикаст-адресою
 * @note  transportProfileUri = UA_PUBSUB_TRANSPORT_UADP_UA_DP
address = opc.udp://224.0.0.22:4840  */
UA_DataSetWriterConfig dswConfig;
memset(&dswConfig, 0, sizeof(dswConfig));
dswConfig.name = UA_STRING("SpeedWriter");
dswConfig.dataSetWriterId = 1;
dswConfig.keyFrameCount = 10; UA_NodeId dswNodeId;
UA_Server_addDataSetWriter(server, wgNodeId, pdsNodeId,
&dswConfig, &dswNodeId);
// --- 5. Налаштування PubSubConnection (UDP Multicast) ---
/**       * @brief Створення підключення з транспортом UADP та
мультикаст-адресою
 *    @note  transportProfileUri = "http://opcfoundation.org/
UA-Profile/Transport/pubsub-uadp-udp"
 *   address = opc.udp://224.0.0.22:4840  */
UA_PubSubConnectionConfig connConfig;
memset(&connConfig, 0, sizeof(connConfig));
connConfig.name = UA_STRING("UDP Multicast Connection");
connConfig.transportProfileUri = UA_STRING(
http://opcfoundation.org/UA-Profile/Transport/pubsub-uadp-udp
);
connConfig.enabled = UA_TRUE;
connConfig.publisherId.idType = UA_PUBLISHERIDTYPE_UINT16;
connConfig.publisherId.id.uint16 = 2234;
UA_NetworkAddressUrlDataType networkAddressUrl;
      networkAddressUrl.networkInterface = UA_STRING("");
      networkAddressUrl.url = UA_STRING("opc.udp://224.0.0.22:
4840");
UA_Variant_setScalar(&connConfig.address, &networkAddressUrl,
                &UA_TYPES[UA_TYPES_NETWORKADDRESSURLDATATYPE]);
UA_NodeId connNodeId;
UA_Server_addPubSubConnection(server, &connConfig, &connNodeId
);
 // ----- 6. Запуск сервера ------
UA_Server_run(server, &running);
```

Інформаційне моделювання в OPC UA, реалізоване через багатий адресний простір та стандартизовані Companion Specifications, у поєднанні з детермінованим транспортом TSN, створює безпрецедентну за глибиною та надійністю платформу для Industry 4.0. Саме ця комбінація семантичної інтероперабельності та гарантованої якості обслуговування робить OPC UA безальтернативним стандартом для конвергенції операційних технологій (OT) та інформаційних технологій (IT) на сучасному виробництві. Наступний підрозділ, присвячений протоколу DDS, продемонструє альтернативний підхід до децентралізованого обміну даними в реальному часі, який часто порівнюють із OPC UA PubSub у контексті високопродуктивних розподілених систем.

9.4. DDS (Data Distribution Service): глобальний простір даних, політики QoS, реалізації у вільному ПЗ

Data Distribution Service (DDS) є стандартом промислового рівня для високопродуктивного, масштабованого та децентралізованого обміну даними в реальному часі, розробленим консорціумом Object Management Group (OMG) і формально специфікованим у документах OMG DDS v1.4 (2015) та супутніх специфікаціях DDSI-RTPS (Real-Time Publish-Subscribe Wire Protocol) і DDS-XRCE (DDS for eXtremely Resource Constrained Environments). На відміну від MQTT, який спирається на централізованого брокера як посередника в доставці повідомлень, або OPC UA PubSub, який може використовувати брокер MQTT або UDP-мультикаст із відносно простою моделлю QoS, DDS реалізує повністю децентралізовану, орієнтовану на дані (data-centric) архітектуру публікації-підписки, де додатки обмінюються даними через єдиний глобальний простір даних (Global Data Space) без жодного центрального вузла. Кожен учасник DDS-домену автономно виявляє інших учасників через протокол динамічного виявлення (Dynamic Discovery), узгоджує параметри якості обслуговування (QoS) і встановлює прямі peer-to-peer з'єднання для обміну даними. Така архітектура забезпечує надзвичайно високу масштабованість (тисячі учасників в одному домені), живучість (відсутність єдиної точки відмови) та мінімальну латентність (одиниці мікросекунд у межах локальної мережі), що робить DDS основним вибором для критично важливих систем — автономних транспортних засобів (AUTOSAR Adaptive), авіоніки (ARINC 661), військових систем управління, медичних роботів, фінансових торговельних платформ, а також розподілених систем управління промисловими роботами. Як зазначено в монографії А. Корсі та Дж. Брукса «DDS: Data-Centric Publish-Subscribe for Real-Time Systems» (CRC Press, 2022), саме комбінація децентралізованого виявлення, прямого обміну даними без брокера та гранично гнучких політик QoS дозволяє DDS задовольнити вимоги, недосяжні для брокер-орієнтованих протоколів, — зокрема, наскрізну латентність менше 100 мкс із джиттером менше 5 мкс для тисяч повідомлень на секунду в розподіленій системі.

Глобальний простір даних (Global Data Space, GDS) є центральною концептуальною абстракцією DDS. З точки зору прикладного програміста, GDS виглядає як єдине, глобальне сховище всіх даних, що циркулюють у системі, де будь-який додаток може «записати» дані в тему (Topic) або «прочитати» їх з теми, не знаючи, хто саме їх створить і де ці додатки фі-

зично знаходяться. Фактично GDS є децентралізованою, логічно централізованою сутністю, яка фізично розподілена між усіма учасниками домену. Тема в DDS має чітко визначений тип даних (DataType), описаний за допомогою мови IDL (Interface Definition Language) або XML, що забезпечує типобезпечний обмін. Наприклад, тема "Temperature" може мати тип даних, визначений як структура з полями sensor_id (string), value (float) та timestamp (int64). Видавець (DataWriter) публікує екземпляри цього типу, а підписник (DataReader) отримує їх, причому DDS автоматично кешує історію даних відповідно до політик QoS, дозволяючи підписнику отримати попередні значення навіть після підключення (Durability). Архітектуру DDS-домену з кількома учасниками, що обмінюються через GDS без брокера, ілюструє рисунок 9.7.

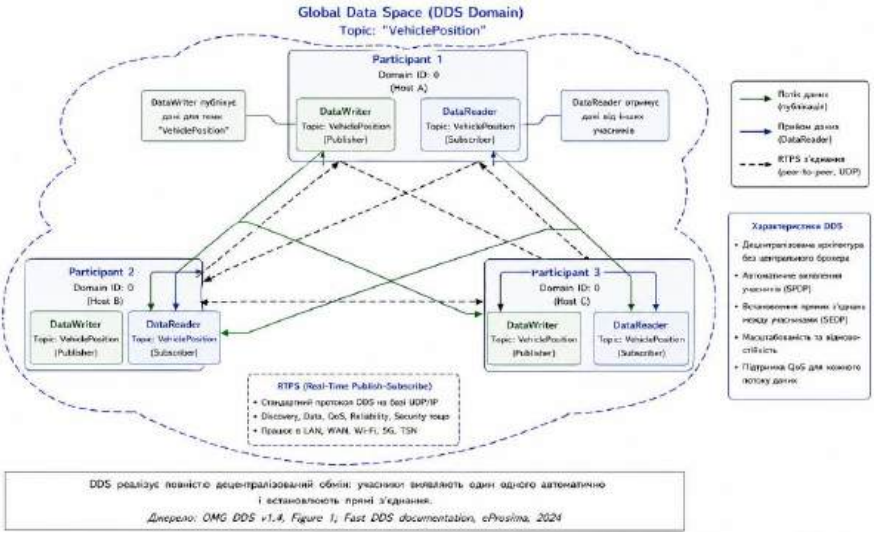


Рисунок 9.7 – Архітектура DDS-домену: глобальний простір даних без брокера

Ключовою відмінністю DDS від MQTT та CoAP є система політик якості обслуговування (QoS Policies), яка налічує понад 20 параметрів, що контролюють усі аспекти доставки даних — від надійності та довговічності до часових обмежень і пріоритетів. Політики QoS узгоджуються за моделлю «запит — пропозиція» (Request-Offer): DataWriter оголошує свої QoS (Offer), DataReader — свої (Request), а middleware автоматично перевіряє сумісність і встановлює з'єднання лише тоді, коли QoS сумісні (наприклад, якщо DataWriter пропонує Reliability=BestEffort, а DataReader вимагає

Reliable, з'єднання не буде встановлено). Серед найважливіших політик QoS слід виокремити Reliability (BestEffort або Reliable — з гарантованою доставкою та повторними передачами), Durability (Volatile — дані не зберігаються; Transient Local — зберігаються для пізніх підписників у межах сеансу; Persistent — зберігаються на диску), Deadline (максимальний інтервал між публікаціями, порушення якого генерує сповіщення), Lifespan (термін дії повідомлення, після якого воно автоматично видаляється), Ownership (Shared — кілька видавців можуть оновлювати один екземпляр; Exclusive — лише один видавець із найвищим пріоритетом має право на оновлення), Liveliness (контроль активності учасників), LatencyBudget (підказка middleware для оптимізації буферизації), Partition (логічне розділення глобального простору даних на підпростори).

Завдяки цим політикам розробник може точно визначити поведінку системи за будь-яких умов, що є критичним для систем із жорсткими вимогами реального часу. У таблиці 9.7 наведено порівняння ключових характеристик DDS, MQTT, OPC UA PubSub та ZeroMQ (як популярної легковагової альтернативи), що дозволяє зробити обґрунтований вибір.

У світі вільного програмного забезпечення беззаперечними лідерами є дві реалізації DDS. eProsima Fast DDS (раніше Fast RTPS) — це високопродуктивна бібліотека на C++, що входить до складу ROS 2 (Robot Operating System 2) як стандартний комунікаційний шар, забезпечуючи обмін даними між вузлами роботів. Fast DDS підтримує всі основні політики QoS, UDP/TCP транспорт, динамічне виявлення, запис і відтворення даних (ROS2 Bag), інтеграцію з DDS-XRCE для мікроконтролерів. Eclipse Cyclone DDS — це реалізація, що розробляється під егідою Eclipse Foundation, написана на C з обгортками для Python, C++ тощо, і відзначається низьким споживанням пам'яті, високою пропускною здатністю та суворою відповідністю специфікації OMG.

Таблиця 9.7 – Порівняння технологій PubSub для розподілених систем

Технології	Хар-ка	Значення
DDS (OMG)	*Архітектура*	децентралізована (peer-to-peer, без брокера)
	Виявлення	автоматичне (SPDP + SEDP за RTPS)
	QoS	>20 політик (Reliability, Durability, Deadline, Ownership тощо)
	Транспорт	UDP (multicast/unicast), TCP
	Латентність (типова)	одиниці–десятки мкс
	Живучість	дуже висока (без єдиної точки відмови)
	Вільні реалізації	Fast DDS, Cyclone DDS, OpenDDS

Кінець таблиці 9.7

Технології	Характеристика	Значення
MQTT 5.0 (OASIS)	*Архітектура*	централізована (клієнт-сервер з брокером)
	Виявлення	ручне (адреса брокера); опціонально DNS-SD/mDNS
	QoS	3 рівні: QoS 0 (at most once), QoS 1 (at least once), QoS 2 (exactly once)
	Транспорт	TCP
	Латентність (типова)	одиниці–десятки мс
	Живучість	низька (відмова брокера паралізує систему)
	Вільні реалізації	Mosquitto, EMQX (Open Source)
OPC UA PubSub (IEC 62541-14)	*Архітектура*	гібридна (безброкерна UDP або брокерна MQTT/AMQP)
	Виявлення	через конфігурацію або discovery-повідомлення
	QoS	обмежена (конфігурується через DatagramQos; залежить від транспорту)
	Транспорт	UDP (UADP), MQTT, AMQP, Ethernet
	Латентність (типова)	десятки мкс – одиниці мс (з TSN)
	Живучість	середня (залежить від транспорту)
	Вільні реалізації	open62541
ZeroMQ	*Архітектура*	децентралізована (бібліотека, без брокера)
	Виявлення	ручне (адреси сокетів); опціонально ZRE (UDP-бікони)
	QoS	немає вбудованої (реалізується на прикладному рівні)
	Транспорт	TCP, UDP, IPC, inproc
	Латентність (типова)	одиниці мкс
	Живучість	висока (децентралізована)
	Вільні реалізації	libzmq (LGPL/MPL-2.0)

Cyclone DDS активно використовується в автомобільній галузі (AUTOSAR Adaptive), промисловій автоматизації та робототехніці. Обидві бібліотеки доступні під ліцензією Apache 2.0 і мають активну спільноту. У лістингу 9.7 наведено мінімальний приклад публікації даних через Fast DDS на C++.

Лістинг 9.7 – Публікація даних через Fast DDS (C++)

```
// Відповідно до eProsima Fast DDS C++ API
#include <fastdds/dds/domain/DomainParticipantFactory.hpp>
#include <fastdds/dds/domain/DomainParticipant.hpp>
#include <fastdds/dds/publisher/Publisher.hpp>
#include <fastdds/dds/publisher/DataWriter.hpp>
#include <fastdds/dds/topic/Topic.hpp>
// ----- 1. Створення учасника (DomainParticipant) ----
/** * @brief Створення учасника DDS домену
 * @note  DomainParticipant визначає робочу одиницю, яка групує
Publisher та Subscriber  */
DomainParticipant* participant =
DomainParticipantFactory::get_instance()->create_participant
(0, PARTICIPANT_QOS_DEFAULT);
// ----- 2. Реєстрація типу даних -----
/** * @brief Реєстрація типу Temperature у Participant
 * @note  Тип має бути згенерований за допомогою Fast DDS-Gen
з IDL-файлу */
TemperaturePubSubType type;
participant->register_type(&type);
// ----- 3. Створення Topic -----
/** * @brief Створення теми "TemperatureTopic" */
Topic* topic = participant->create_topic("TemperatureTopic",
type.get_type_name(), TOPIC_QOS_DEFAULT);
// ----- 4. Створення Publisher ------
/** * @brief Створення видавця */
Publisher* publisher = participant->create_publisher
(PUBLISHER_QOS_DEFAULT);
// ----- 5. Налаштування DataWriterQos -----
/** * @brief Налаштування QoS для DataWriter
 * @note  Reliability: RELIABLE_RELIABILITY_QOS — гарантована
Доставка. Durability: TRANSIENT LOCAL DURABILITY QOS —
збереження даних для пізніх підписників
 *   History: KEEP_ALL_HISTORY_QOS — зберігання всіх зразків */
DataWriterQos wqos;
wqos.reliability().kind = RELIABLE_RELIABILITY_QOS;
// Гарантована доставка
wqos.durability().kind = TRANSIENT_LOCAL_DURABILITY_QOS;
// Доступність для пізніх підписників
wqos.history().kind = KEEP_ALL_HISTORY_QOS; // Зберігати всі
зразки
// ----- 6. Створення DataWriter -----
/** * @brief Створення DataWriter для публікації даних */
DataWriter* writer = publisher->create_datawriter(topic, wqos,
nullptr);
// ----- 7. Публікація даних у циклі -----
/** * @brief Заповнення структури Temperature та публікація
 * @note  Підписник у тому ж домені автоматично виявить
DataWriter */
Temperature sample;
while (true) {
    sample.sensor_id(1);
    sample.value(25.5 + (i % 10) * 0.5);
    sample.timestamp(get_current_time());
    writer->write(&sample);  // Публікація даних
    std::this_thread::sleep_for(std::chrono::seconds(1));   }
```

DDS є найпотужнішим із розглянутих протоколів прикладного рівня за сукупністю можливостей децентралізації, реального часу та гнучкості QoS. Його складність виправдана в системах, де ціна відмови або затримки є надзвичайно високою, а масштаб і розподіленість унеможливлюють централізоване керування. Цим завершується огляд інтерфейсів програмного рівня в IoT; наступна частина посібника, присвячена архітектурним патернам і типовим кейсам, покаже, як розглянуті протоколи комбінуються в реальних проектах.

РОЗДІЛ 10.
ІНТЕРФЕЙСИ ВНУТРІШНЬОКРИСТАЛЬНИХ І СИСТЕМНИХ ШИН

10.1. AMBA (ARM Advanced Microcontroller Bus Architecture): AHB, APB, AXI4 – підключення периферії до ядра

Усі розглянуті в попередніх розділах зовнішні інтерфейси — UART, SPI, Ethernet, Wi-Fi — є лише вершиною айсберга. Їхні апаратні контролери всередині мікроконтролера мусять бути з'єднані з процесорним ядром, оперативною пам'яттю, контролером переривань і між собою. Цю роль виконує внутрішньокристальна шинна інфраструктура, яка в абсолютній більшості сучасних мікроконтролерів із ядрами ARM базується на стандарті AMBA (Advanced Microcontroller Bus Architecture), розробленому компанією ARM Holdings. AMBA не є однією шиною; це сімейство специфікацій, що описують кілька типів шин із різною продуктивністю, складністю та призначенням, об'єднаних у єдину ієрархічну систему. Актуальними на сьогодні є специфікації AMBA 5 (2015), які включають AXI4 (Advanced eXtensible Interface 4), AHB (Advanced High-performance Bus) та APB (Advanced Peripheral Bus). У мікроконтролерах STM32 на ядрах Cortex-M3/M4/M7 (серії F2, F4, F7, H7) та Cortex-M33 (серії L5, U5) шинна архітектура побудована саме на комбінації AHB та APB, тоді як у старших сімействах STM32MP1/MP2 (з ядрами Cortex-A), а також у FPGA-системах на базі ARM широко використовується AXI4. Розуміння цієї ієрархії є необхідним для свідомого програмування DMA, налаштування пріоритетів шинної матриці та оптимізації продуктивності системи на низькому рівні.

Основою внутрішньокристальної комунікації в переважній більшості мікроконтролерів STM32 є шина AHB (Advanced High-performance Bus), яка у версії AMBA 3 (AHB-Lite) та AMBA 5 (AHB5) оптимізована для високошвидкісних передач між ядром процесора, оперативною пам'яттю (SRAM), флеш-пам'яттю, контролером DMA та високошвидкісними периферійними модулями (Ethernet MAC, USB OTG, SDMMC). AHB є синхронною шиною з конвеєризованою архітектурою, що підтримує пакетні (burst) передачі, розщеплені транзакції (split transactions) у повній версії, а також захист пам'яті через MPU. Ключовою особливістю AHB-Lite, що використовується в Cortex-M, є наявність єдиного майстра (single-master) або спрощеної багатомайстерної матриці (Bus Matrix), яка дозволяє кільком майст-

рам (ядро, DMA, Ethernet) паралельно звертатися до різних підлеглих пристроїв без блокування. Наприклад, у STM32F4 (RM0090) матриця AHB об'єднує ядро Cortex-M4 (через шину I-bus, D-bus, S-bus), два контролери DMA (DMA1, DMA2), USB OTG FS та Ethernet MAC, надаючи їм одночасний доступ до SRAM, флеш-пам'яті, AHB-периферії (GPIO, FSMC) та мостів до APB. Шина AHB працює на частоті HCLK, що може сягати 168 МГц (STM32F4), 240 МГц (STM32H7) і вище, забезпечуючи пропускну здатність, достатню для потокових передач даних без участі процесора.

Низькошвидкісна периферія — UART, SPI, I²C, таймери, GPIO — підключається через окрему шину APB (Advanced Peripheral Bus), яка є простішою та енергоефективнішою за AHB. APB не підтримує burst-передачі, конвеєризацію та складні транзакції; її основна задача — надати простий, синхронний інтерфейс із мінімальними апаратними витратами для регістрів конфігурації та даних периферійних модулів. У архітектурі STM32 шини APB1 та APB2 підключаються до AHB через мости (AHB-to-APB Bridges), які транслюють транзакції AHB у формат APB. Частота APB зазвичай є похідною від HCLK через програмовані подільники, що дозволяє знижувати тактування периферії для економії енергії. У таблиці 10.1 наведено типове підключення периферійних модулів до шин APB1/APB2 у STM32F407, яке демонструє логіку розподілу: на швидшій APB2 розташовані USART1, SPI1, таймери високої роздільної здатності, тоді як на APB1 — менш вимогливі USART2/3, I²C, звичайні таймери.

Таблиця 10.1 – Типове підключення периферії до шин AHB, APB1, APB2 у STM32F407

Шина	Максимальна частота	Підключена периферія (вибрані модулі)
AHB (матриця)	168 МГц	SRAM, Flash, DMA1/DMA2, Ethernet MAC, USB OTG FS, AHB-to-APB Bridges
APB1	42 МГц (макс. 168/4)	USART2, USART3, I²C1, I²C2, SPI2, SPI3, TIM2–TIM7
APB2	84 МГц (макс. 168/2)	USART1, SPI1, SDIO, TIM1, TIM8–TIM11, ADC1–ADC3

У високопродуктивних системах на ядрі Cortex-A (наприклад, STM32MP1) та в FPGA-реалізаціях ARM використовується шина AXI4 (Advanced eXtensible Interface 4), яка пропонує значно вищу пропускну здатність і гнучкість, ніж AHB. AXI4 підтримує незалежні канали для адреси/керування та даних (5 каналів: Read Address, Read Data, Write Address, Write Data, Write Response), що дозволяє виконувати одночасні читання та запис (повнодуплексний обмін), видавати кілька непогашених транзакцій

(outstanding transactions), передавати дані з вибуховим режимом (burst) до 256 тактів і підтримувати QoS-арбітраж. Ширина шини даних AXI може бути 32, 64, 128 або 256 біт. AXI4-Lite є спрощеним варіантом для регістрового доступу (один burst=1), а AXI4-Stream — для потокових даних без адресації, що широко використовується в FPGA для з'єднання IP-ядер. У сучасних STM32 (наприклад, STM32H7) внутрішньо використовується комбінація: ядро Cortex-M7 підключається до AXI-матриці, яка зв'язує його з кешами, SRAM, зовнішньою SDRAM через FMC, а також з мостами до AHB/APB, що дозволяє досягти продуктивності, необхідної для ядра з тактовою частотою 480 МГц. Порівняння характеристик трьох основних шин AMBA наведено в таблиці 10.2.

Таблиця 10.2 – Порівняння характеристик шин AMBA

Шина	Характеристика	Значення
AHB (AMBA 2/3/5)	*Призначення*	високошвидкісна внутрішня шина для з'єднання процесора, пам'яті та високошвидкісної периферії
	Максимальна пропускна здатність	до кількох Гбіт/с (залежить від частоти та ширини шини)
	Підтримка burst	так (до 256 тактів у специфікації)
	Кількість каналів	1 (суміщений адресний та даний; читання або запис)
	Кількість непогашених транзакцій (Outstanding)	1 (без split-транзакцій у базовій версії)
	Типове застосування	з'єднання ядра з пам'яттю (SRAM, Flash), DMA, високошвидкісною периферією (Ethernet, USB)
APB (AMBA 3/5)	*Призначення*	низькошвидкісна шина периферії з мінімальним енергоспоживан. та спрощеним інтерфейсом
	Максимальна пропускна здатність	до сотень Мбіт/с (обмежена 32-бітною шиною та відсутністю burst)
	Підтримка burst	ні (кожен трансфер займає фіксовано 2 такти)
	Кількість каналів	1 (суміщений адресний та даний; читання або запис)
	Кількість непогашених транзакцій (Outstanding)	1
	Типове застосування	регістровий доступ до низькошвидкісної периферії (UART, SPI, I²C, таймери)

Кінець таблиці 10.2

Шина	Характеристика	Значення
AXI4 (AMBA 4/5)	*Призначення*	високопродуктивна магістраль для багатоядерних SoC, FPGA та високошвидкісних інтерфейсів
	Максимальна пропускна здатність	десятки Гбіт/с (до 1024 біт, висока частота, паралельні канали)
	Підтримка burst	так (до 256 трансферів на один burst)
	Кількість каналів	5 незалежних (AW, W, B, AR, R) – читання та запис паралельно
	Кількість непогашених транзакцій (Outstanding)	багато (підтримка multiple outstanding транзакцій з можливістю переупорядкування)
	Типове застосування	процесорні ядра Cortex-A, GPU, пам'ять DDR, PCIe, високошвидкісні інтерфейси

Графічно ієрархію шин AMBA у типовому мікроконтролері STM32F4 показано на рисунку 10.1, який демонструє, як ядро, DMA і периферія об’єднуються через матрицю AHB та мости APB.

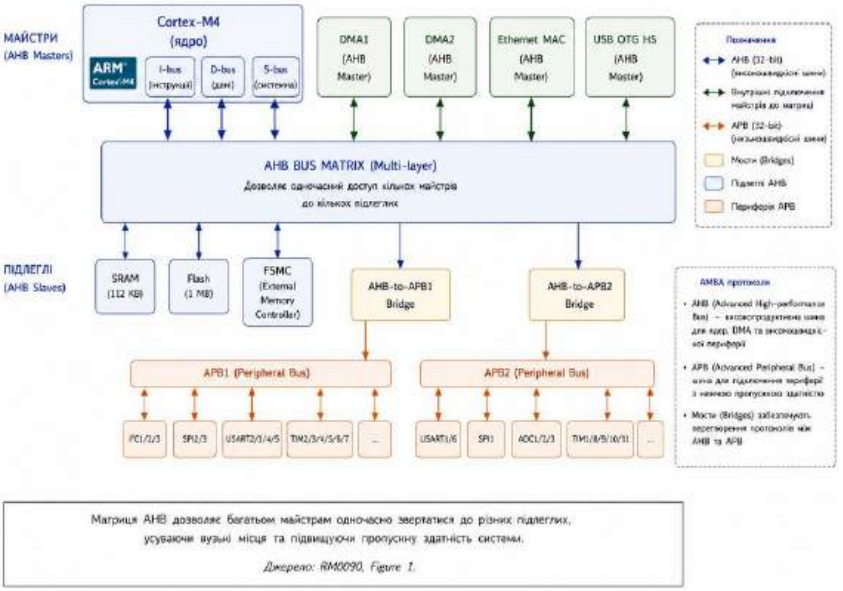


Рисунок 10.1 – Ієрархія шин AMBA у STM32F4 (спрощена)

З точки зору програміста вбудованих систем, безпосереднє програмування шин AMBA не потрібне — цим займаються апаратні дизайнери. Однак розуміння ієрархії шин є критичним для ефективного використання DMA. Наприклад, в STM32F4 контролер DMA2 має прямий доступ до APB2, тому він може обслуговувати SPI1, але не SPI2 (який на APB1) — для SPI2 потрібен DMA1. Крім того, у STM32H7 існує кілька доменів шин (Domain D1, D2, D3), і неправильне налаштування DMA для периферії з іншого домену може призвести до помилок шини. У лістингу 10.1 наведено фрагмент коду на HAL для STM32H7, який демонструє конфігурацію DMA для UART з урахуванням необхідності використання правильного потоку даних (DMA1_Stream0 для USART3_RX, що знаходиться на APB1).

Лістинг 10.1 – Конфігурація DMA для UART (STM32H7, HAL)

```
#include "stm32h7xx_hal.h"
DMA_HandleTypeDef hdma_usart3_rx;
DMA_HandleTypeDef hdma_usart3_tx;
static void MX_DMA_Init(void) {
// -- DMA для RX (циркулярний режим, периферія -> пам'ять) --
    hdma_usart3_rx.Instance = DMA1_Stream1;
    hdma_usart3_rx.Init.Request = DMA_REQUEST_USART3_RX;
    hdma_usart3_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_usart3_rx.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_usart3_rx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_usart3_rx.Init.PeriphDataAlignment =
DMA_PDATAALIGN_BYTE;
    hdma_usart3_rx.Init.MemDataAlignment =
DMA_MDATAALIGN_BYTE;
    hdma_usart3_rx.Init.Mode = DMA_CIRCULAR;
    hdma_usart3_rx.Init.Priority = DMA_PRIORITY_HIGH;
    HAL_DMA_Init(&hdma_usart3_rx);
    __HAL_LINKDMA(&huart3, hdmarx, hdma_usart3_rx);
// -- DMA для TX (нормальний режим, пам'ять -> периферія) --
    hdma_usart3_tx.Instance = DMA1_Stream3;
    hdma_usart3_tx.Init.Request = DMA_REQUEST_USART3_TX;
    hdma_usart3_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
    hdma_usart3_tx.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_usart3_tx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_usart3_tx.Init.PeriphDataAlignment =
DMA_PDATAALIGN_BYTE;
    hdma_usart3_tx.Init.MemDataAlignment =
DMA_MDATAALIGN_BYTE;
    hdma_usart3_tx.Init.Mode = DMA_NORMAL;
    hdma_usart3_tx.Init.Priority = DMA_PRIORITY_HIGH;
    HAL_DMA_Init(&hdma_usart3_tx);
    __HAL_LINKDMA(&huart3, hdmatx, hdma_usart3_tx);
    // ---- Налаштування NVIC для переривань DMA ----
    HAL_NVIC_SetPriority(DMA1_Stream1_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream1_IRQn);
    HAL_NVIC_SetPriority(DMA1_Stream3_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream3_IRQn);    }
```

Сімейство шин AMBA є фундаментом, на якому будується вся внутрішньокристальна взаємодія в ARM-мікроконтролерах. AHB забезпечує високошвидкісний обмін із пам'яттю та критичною периферією, APB — простий і економний доступ до регістрів, а AXI4 масштабує цю архітектуру до рівня багатоядерних SoC із гетерогенними обчислювальними ресурсами. Наступний підрозділ, присвячений шині Wishbone для RISC-V, продемонструє альтернативний відкритий підхід до вирішення тієї самої задачі.

10.2. Wishbone – відкрита шина для SoC на базі RISC-V

Якщо шини AMBA, розглянуті в попередньому підрозділі, є пропрієтарним сімейством, нерозривно пов'язаним із екосистемою ARM, то відкрита апаратна архітектура RISC-V зумовила потребу в настільки ж відкритій, вільній від ліцензійних обмежень внутрішньокристальній шині. Такою шиною став Wishbone – проста, гнучка й повністю відкрита специфікація, уперше опублікована організацією OpenCores у 2002 році та оновлена до версії B4 (Wishbone Bus Specification B4, OpenCores, 2010). На відміну від AMBA, яка пропонує три окремі протоколи (AHB, APB, AXI) для різних рівнів продуктивності, Wishbone є уніфікованою шиною, що може працювати в різних топологіях (точка-точка, загальна шина, хрестова матриця) і з різними швидкостями, залишаючись при цьому сумісною на логічному рівні. Така універсальність зробила Wishbone де-факто стандартом для вільного апаратного забезпечення: переважна більшість IP-ядер на платформі OpenCores, а також численні реалізації RISC-V-процесорів (PicoRV32, SERV, VexRiscv, NeonRV) використовують Wishbone як основний інтерфейс для підключення пам'яті, периферії та співпроцесорів. Крім того, Wishbone широко застосовується в інструментаріях для автоматизованого складання SoC на базі FPGA, таких як LiteX (Migen), де він слугує стандартним клеєм між ядром процесора та периферійними контролерами UART, SPI, GPIO, Ethernet. Хоча в комерційних мікроконтролерах на RISC-V (наприклад, ESP32-C3/C6 від Espressif або GD32VF103 від GigaDevice) можуть застосовуватися пропрієтарні шини, що внутрішньо нагадують Wishbone або AHB, саме відкритість Wishbone робить його незамінним у навчальному процесі, дозволяючи студентам вивчати внутрішню будову SoC без юридичних обмежень.

Порівняння ключових характеристик Wishbone та основних шин AMBA наведено в таблиці 10.3, яка демонструє, що за гнучкістю Wishbone наближається до AXI, але за складністю є ближчим до AHB.

Таблиця 10.3 – Порівняння характеристик Wishbone (B4) з шинами AMBA

Шина	Характеристика	Значення
Wishbone B4	*Топології*	точка-точка, загальна шина, матриця (crossbar)
	Конвеєризація	опціональна (підтримується в B4 через pipelined режим)
	Підтримка burst	так (через CTI/BTE)
	Кількість сигналів (базовий набір)	~10–15 (CLK, RST, CYC, STB, WE, ADR, DAT_I, DAT_O, SEL, ACK + опціональні ERR/RTY/TAG)
	Ліцензія	публічний домен / відкрита (OpenCores)
	Типове застосування	відкриті RISC-V SoC, FPGA-ядра, OpenCores проєкти
AHB (AMBA 3)	*Топології*	матриця (AHB Bus Matrix)
	Конвеєризація	так (single-cycle bus master handover)
	Підтримка burst	так
	Кількість сигналів (базовий набір)	~20+ (HCLK, HRESETn, HADDR, HWDATA, HRDATA, HTRANS, HSIZE, HBURST, HPROT, HWRITE, HREADY, HRESP, HSELx та ін.)
	Ліцензія	пропрієтарна (ARM)
	Типове застосування	мікроконтролери ARM, вбудовані системи
AXI4 (AMBA 4)	*Топології*	матриця (Interconnect)
	Конвеєризація	так (повна конвеєризація)
	Підтримка burst	так (до 256 тактів)
	Кількість сигналів (базовий набір)	~40+ (5 каналів з сигналами VALID/READY, адресними, даними та відповідями)
	Ліцензія	пропрієтарна (ARM)
	Типове застосування	Cortex-A SoC, FPGA-системи з ARM, високопродуктивна периферія

З точки зору протоколу Wishbone є синхронною шиною з єдиним тактовим сигналом (CLK) і скиданням (RST). Усі транзакції ініціюються майстром (Master) і адресуються до підлеглого (Slave). Базовий набір сигналів включає: STB (Strobe) – сигнал валідності транзакції, CYC (Cycle) – ознака активного циклу шини, WE (Write Enable) – ознака запису (1) або читання (0), ADR (Address) – шина адреси, DAT_I (Data In) – дані від підлеглого до майстра, DAT_O (Data Out) – дані від майстра до підлеглого, ACK

(Acknowledge) – підтвердження завершення транзакції підлеглим, та опціональні сигнали ERR (Error), RTY (Retry), CTI (Cycle Type Identifier) і BTE (Burst Type Extension) для підтримки пакетних передач. Протокол є надзвичайно простим: майстер виставляє адресу, дані (для запису) та сигнали CYC, STB і WE; підлеглий, розпізнавши адресу, виконує операцію і виставляє ACK (та DAT_I для читання). Майстер утримує CYC і STB активними до отримання ACK, після чого може завершити цикл або розпочати наступний. Така простота дозволяє реалізувати Wishbone-сумісний пристрій із мінімальними витратами логіки (буквально десятки логічних елементів), що ідеально підходить для FPGA-проектів і навчальних лабораторій.

Архітектуру типової Wishbone-системи з одним майстром (RISC-V ядро) та кількома підлеглими (UART, GPIO, блок пам'яті) показано на рисунку 10.2. Через спільну шину (Shared Bus) або матрицю майстер може звертатися до будь-якого підлеглого, використовуючи адресацію, що призначається на етапі проектування SoC. У більш складних системах застосовується Wishbone Crossbar (матриця), яка дозволяє кільком майстрам паралельно працювати з різними підлеглими, подібно до AHB Bus Matrix.

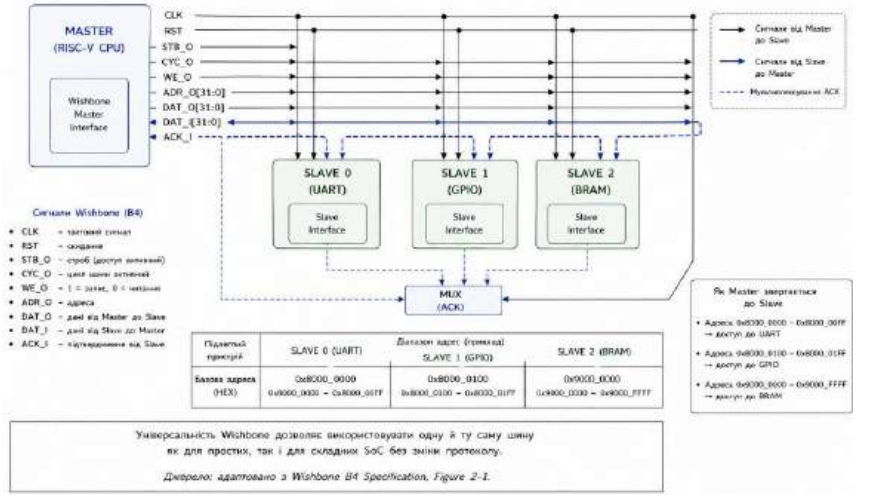


Рисунок 10.2 – Спрощена архітектура Wishbone-системи з одним майстром та трьома підлеглими

Практична робота з Wishbone під час програмування мікроконтролерів зазвичай прихована за стандартними бібліотеками, однак для розуміння механізму підключення периферії на рівні HDL-опису типовим є наведення Verilog-коду підлеглого пристрою, що реалізує регістр за адресою

Wishbone. У лістингу 10.2 показано мінімальний приклад Wishbone-сумісного регістра, який демонструє реакцію на сигнали STB, CYC, WE, адресу та формування ACK. Такий код може бути використаний під час проєктування власних периферійних модулів у складі RISC-V SoC.

Лістинг 10.2 – Мінімальна реалізація Wishbone-підлеглого (регістр) на Verilog

```
// Відповідно до Wishbone B4 Specification (OpenCores)
module wb_register (
  input wire   CLK,     // Тактовий сигнал
  input wire   RST,     // Скидання (активний високий рівень)
  input wire   CYC, // Цикл шини: активний протягом усього циклу
  input wire STB, // Строб: активний під час передачі даних
  input wire WE, // Напрямок: 1 - запис, 0 - читання
  input wire [31:0] ADR,  // Адреса (тут перевіряється збіг)
  input wire [31:0] DAT_I, // Вхідні дані (від майстра)
  output reg [31:0] DAT_O, // Вихідні дані (до майстра)
  output reg  ACK // Підтвердження: сигналізує про завершення
циклу
);
// ------- Внутрішній регістр --------
reg [31:0] reg_out;     // Регістр, доступний для читання/запису
// ------- Логіка Wishbone-підлеглого ------
always @(posedge CLK or posedge RST) begin
    if (RST) begin
        reg_out <= 32'b0;
        ACK     <= 1'b0;
        DAT_O   <= 32'b0;
    end else begin
   // За замовчуванням ACK = 0 (очікування наступного циклу)
        ACK <= 1'b0;
   // Якщо активні CYC та STB і адреса збігається (тут приклад
для ADR == 0)
        if (CYC && STB && (ADR == 0)) begin
            if (WE) begin
    // Запис: дані з DAT_I зберігаються в регістрі
                reg_out <= DAT_I;
            end else begin
    // Читання: дані з регістра виводяться на DAT O
                DAT_O <= reg_out;
            end
  // Підтвердження на наступному такті після виконання операції
            ACK <= 1'b1;
        end    end    end

endmodule
```

Wishbone є не лише альтернативою AMBA, а й ідеологічним фундаментом відкритого апаратного забезпечення, що дозволяє створювати повноцінні системи на кристалі без сплати роялті. Його простота та гнучкість

роблять його чудовою відправною точкою для вивчення архітектури мікропроцесорних систем, а сумісність із RISC-V відкриває шлях до вільного проектування IoT-пристроїв на всіх рівнях – від прикладного коду до транзисторів. Наступний підрозділ, присвячений механізмам взаємодії через спільну пам'ять та DMA, покаже, як ці внутрішні шини використовуються для ефективного обміну даними без участі центрального процесора.

10.3. Взаємодія через спільну пам'ять та DMA: дескриптори, кругові буфери, програмування контролерів

Розглянуті в попередніх підрозділах внутрішньокристальні шини — AHB, APB, AXI та Wishbone — забезпечують фізичне середовище для передавання даних між ядром процесора, пам'яттю та периферією. Однак ефективність системи визначається не лише пропускною здатністю цих шин, а й тим, наскільки раціонально центральний процесор бере участь у пересиланні даних. Наївний підхід, за яким процесор власноруч читає байт із регістра даних UART і записує його в пам'ять (режим опитування або переривання), є прийнятним лише для низьких швидкостей обміну. На швидкостях у десятки мегабіт за секунду таке постійне втручання ядра призводить до катастрофічного падіння продуктивності решти задач, оскільки процесор витрачає левову частку часу на обробку переривань і копіювання даних. Вирішенням цієї проблеми є прямий доступ до пам'яті (Direct Memory Access, DMA) — механізм, за якого спеціалізований апаратний контролер (DMA Controller, DMAC) самостійно, без участі процесора, виконує пересилання блоків даних між периферійними регістрами та оперативною пам'яттю, використовуючи ті самі внутрішньокристальні шини. Сучасні мікроконтролери, такі як STM32F4/F7/H7, оснащені багатоканальними DMA-контролерами зі складною системою пріоритетів, підтримкою пакетних (burst) передач, здатністю виконувати операції з пам'яті в пам'ять, а також гнучкими механізмами програмування через дескриптори та кругові буфери. Саме ці механізми є предметом даного підрозділу.

Фундаментальною концепцією, що лежить в основі роботи будь-якого сучасного DMA-контролера, є дескриптор (Descriptor) — структура даних, розташована в оперативній пам'яті, яка описує одну елементарну операцію пересилання. Використання дескрипторів дозволяє DMA-контролеру виконувати складні ланцюжки передач без втручання процесора: після завершення однієї транзакції контролер автоматично зчитує з пам'яті наступний дескриптор і продовжує роботу. Дескриптор зазвичай містить адресу джерела даних (Source Address), адресу призначення (Destination

Address), кількість елементів для пересилання (Transfer Size), ознаки інкременту адрес, розмір даних (байт, півслово, слово), а також посилання на наступний дескриптор у ланцюжку (Next Descriptor Address). У мікроконтролерах STM32H7 (документ RM0468) DMA-контролер підтримує два режими роботи: регістровий (безпосереднє програмування регістрів DMAC програмістом) та дескрипторний (Linked List Mode). У дескрипторному режимі програміст заздалегідь готує в пам'яті ланцюжок структур, кожна з яких описує окрему передачу, і передає контролеру адресу першого дескриптора. Контролер самостійно обходить ланцюжок, виконуючи передачі одна за одною, і може генерувати переривання після завершення всього ланцюжка або кожного дескриптора. Це звільняє процесор навіть від необхідності перезапускати DMA після кожної передачі, що є критичним для високошвидкісних потокових застосувань, таких як аудіо (I²S), відео (DCMI) або мережевий трафік (Ethernet MAC). Типову структуру дескриптора DMA для STM32H7 та його зв'язок із регістрами контролера наведено на рисунку 10.3.

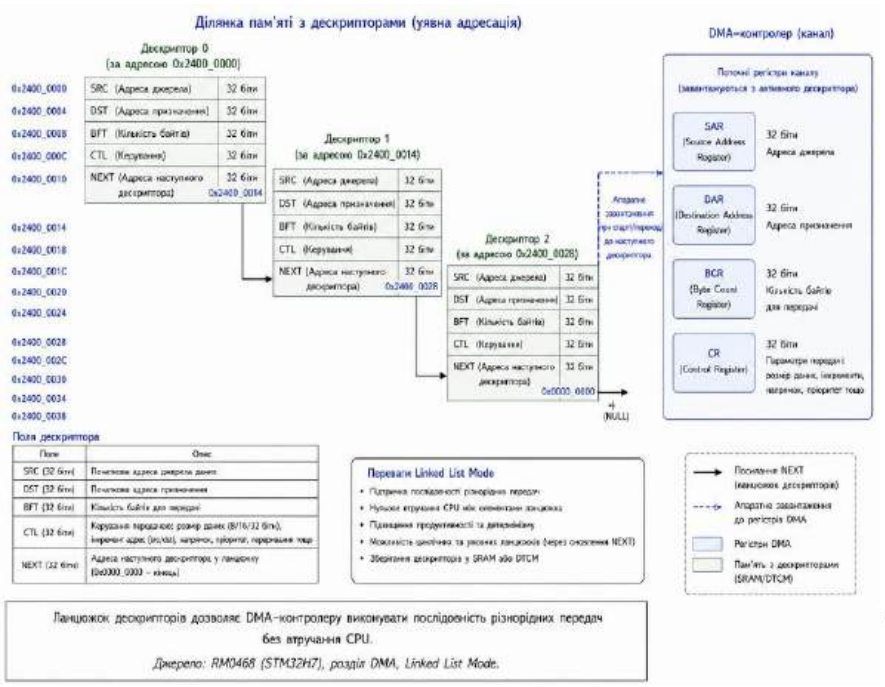


Рисунок 10.3 – Структура дескриптора DMA (Linked List Mode) у STM32H7

Круговий буфер (Circular Buffer) є програмною концепцією, яка в поєднанні з апаратним циркулярним режимом DMA (Circular Mode) створює надзвичайно ефективний механізм для безперервного потокового приймання або передавання даних. Круговий буфер — це масив фіксованого розміру в пам'яті, який логічно замкнений у кільце: після запису останнього елемента наступний запис відбувається на початок масиву, затираючи найстаріші дані. Апаратний циркулярний режим DMA реалізує цю логіку на рівні контролера: після завершення пересилання всього блоку даних (розмір буфера) DMA-контролер автоматично скидає вказівник адреси на початок буфера і продовжує пересилання без жодного втручання процесора, без втрати даних між блоками. Це є безальтернативним рішенням для приймання безперервних потоків від АЦП, UART, I²S, де зупинка DMA навіть на кілька мікросекунд для перезапуску може призвести до втрати даних. У комбінації з механізмом переривання по половині буфера (Half-Transfer Interrupt, HT) та по завершенню повного буфера (Transfer Complete Interrupt, TC) програміст отримує зручний API: поки DMA заповнює першу половину буфера, процесор обробляє другу половину, і навпаки (подвійна буферизація). Така техніка, відома як ping-pong buffering, детально описана в прикладній нотатці AN4031 (STMicroelectronics, 2019) і широко використовується в цифровій обробці сигналів на STM32. Схематичне зображення кругового буфера та моментів переривань наведено на рисунку 10.4.

Програмування DMA-контролерів у сучасних мікроконтролерах здійснюється через бібліотеки рівня HAL (Hardware Abstraction Layer), які приховують низькорівневі деталі, але вимагають від розробника чіткого розуміння обмежень, що накладаються внутрішньою шинною архітектурою. Як зазначалося в підрозділі 10.1, кожен DMA-контролер підключений до певного сегмента шинної матриці, тому не будь-який потік DMA може обслужити будь-яку периферію. Наприклад, у STM32F4 DMA1 працює з APB1 (SPI2, I²C, USART2), тоді як DMA2 — з APB2 (SPI1, USART1) та внутрішньою пам'яттю. У STM32H7 архітектура складніша: присутні DMA1, DMA2, BDMA (Basic DMA для периферії APB) та MDMA (Master DMA для передач між доменами пам'яті), і неправильний вибір контролера призведе до апаратної помилки (bus fault). Крім того, слід враховувати вирівнювання даних у пам'яті: багато DMA-контролерів (особливо в STM32F4) вимагають, щоб адреси джерела та призначення були вирівняні відповідно до розміру даних (наприклад, для 32-бітних передач — адреса має бути кратною 4). Порушення вирівнювання призводить до помилок DMA (Transfer Error, TEIF). Також важливим є кешування даних у старших

ядрах Cortex-M7: якщо DMA записує дані в SRAM, а процесор читає їх із кешу даних (D-Cache), може виникнути некогерентність. Для її усунення необхідно або очищати кеш перед DMA-читанням (Cache Clean), або анулювати кеш після DMA-запису (Cache Invalidate), або використовувати нек ешовані ділянки пам'яті (наприклад, DTCM або SRAM із вимкненим кешуванням через MPU). Ці аспекти детально висвітлені в AN4839 «Level 1 cache management on STM32F7 Series and STM32H7 Series» (STMicroelectronics, 2020).

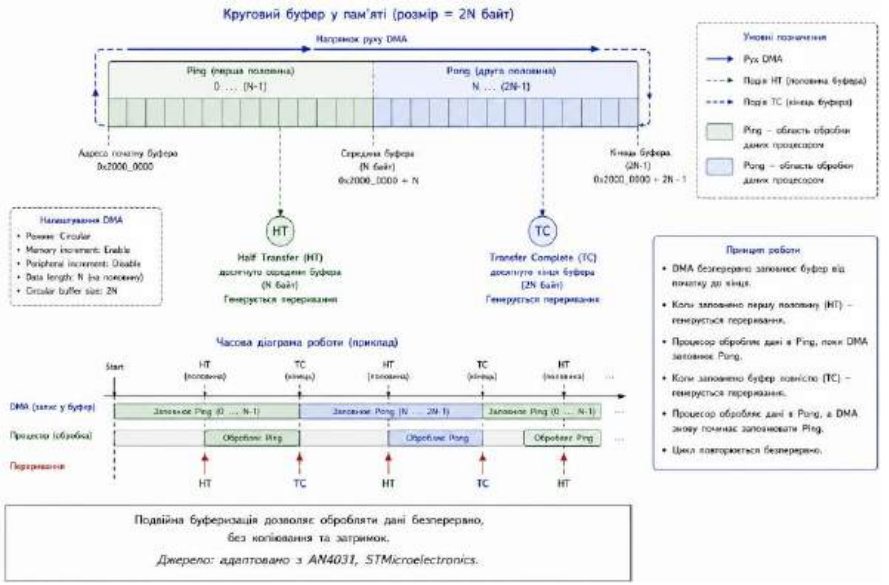


Рисунок 10.4 – Круговий буфер DMA з подвійною буферизацією (ping-pong)

У лістингу 10.3 наведено фрагмент коду на HAL для STM32H7, який демонструє налаштування дескрипторного DMA (MDMA) для передавання даних із буфера в UART з використанням режиму Linked List. Така конфігурація дозволяє автоматично відправляти пакети різної довжини без перезапуску DMA.

Лістинг 10.3 – Налаштування MDMA у Linked List Mode для передавання в UART на STM32H7 (HAL)

```
1:// Відповідно до RM0433 (STM32H7 Reference Manual) та STM32H7
2:#include "stm32h7xx_hal.h"
3:#include "stm32h7xx_hal_mdma.h"
```

```
// ---- Дескриптори MDMA ------
/** * @brief Масив дескрипторів для ланцюжка MDMA
 * @note  Дескриптори мають бути розташовані в нек ешованій пам'яті (наприклад, SRAM4) для коректної роботи з MDMA  */
#define MDMA_DESC_COUNT 3
__attribute__((section(".sram4"))) MDMA_DescriptorTypeDef mdma_descriptors[MDMA_DESC_COUNT];
// ------ Налаштування дескрипторів ------
/** * @brief Заповнення полів дескрипторів MDMA
 * @note  SrcAddress – адреса джерела даних (наприклад, буфер у SRAM)
 *  DstAddress – адреса призначення (регістр TDR USART3)
 *  BlockDataLength – кількість байт для передачі
 *  NextDescAddress – адреса наступного дескриптора (NULL для останнього) */
void MDMA_LinkedList_Init(void) {
    // Дескриптор 0
    mdma_descriptors[0].SrcAddress = (uint32_t)&tx_buffer_0;
    mdma_descriptors[0].DstAddress = (uint32_t)&USART3->TDR;
       mdma_descriptors[0].BlockDataLength = 16;   // 16 байт
       mdma_descriptors[0].NextDescAddress = (uint32_t)&mdma_descriptors[1];
     // Дескриптор 1
     mdma_descriptors[1].SrcAddress = (uint32_t)&tx_buffer_1;
     mdma_descriptors[1].DstAddress = (uint32_t)&USART3->TDR;
     mdma_descriptors[1].BlockDataLength = 32;    // 32 байти
       mdma_descriptors[1].NextDescAddress = (uint32_t)&mdma_descriptors[2];
    // Дескриптор 2 (останній)
    mdma_descriptors[2].SrcAddress = (uint32_t)&tx_buffer_2;
    mdma_descriptors[2].DstAddress = (uint32_t)&USART3->TDR;
    mdma_descriptors[2].BlockDataLength = 64;     // 64 байти
    mdma_descriptors[2].NextDescAddress = 0; // Кінець ланцюжка
}
// ----- Конфігурація MDMA_HandleTypeDef ----
/** * @brief Налаштування дескриптора MDMA
 * @note  MDMA_Init.Request – вибір запиту (наприклад, MDMA REQUEST SW для програмного запуску)
 * MDMA_Init.TriggerMode – MDMA_LINKED_LIST_TRANSFER для режиму ланцюжка  */
MDMA_HandleTypeDef hmdma;
hmdma.Instance = MDMA_Channel0;
hmdma.Init.Request = MDMA_REQUEST_SW; // Програмний запуск
hmdma.Init.TransferTriggerMode = MDMA_LINKED_LIST_TRANSFER; // Режим ланцюжка
hmdma.Init.Priority = MDMA_PRIORITY_HIGH;
hmdma.Init.SourceInc = MDMA_SRC_INC_WORD;
hmdma.Init.DestinationInc = MDMA_DEST_INC_DISABLE; // Фіксована адреса (регістр UART)
hmdma.Init.SourceDataSize = MDMA_SRC_DATASIZE_BYTE;
hmdma.Init.DestDataSize = MDMA_DEST_DATASIZE_BYTE;
hmdma.Init.BlockTransfer = MDMA_BLOCK_TRANSFER_NO;
HAL_MDMA_Init(&hmdma);
// ------ Запуск передачі ------
/** * @brief Запуск MDMA у режимі ланцюжка
 * @note  HAL_MDMA_Start_IT() – запуск із перериваннями
 * @note  Вказівник на перший дескриптор передається як параметр
```

```
pConf  */
HAL_MDMA_Start_IT(&hmdma,   // Дескриптор MDMA
  (uint32_t)&mdma_descriptors[0], // Адреса першого дескриптора
   sizeof(MDMA_DescriptorTypeDef), // Розмір дескриптора
                    1);  // Кількість дескрипторів
// ----- Callback завершення -----
/** * @brief Callback, що викликається після завершення всього
ланцюжка
 * @note  Встановлює прапорець для сповіщ. основної програми */
void HAL_MDMA_XferCpltCallback(MDMA_HandleTypeDef *hmdma) {
  transfer complete flag = 1;  // Прапорець завершення ланцюжка
}
```

Порівняльний аналіз режимів DMA на платформах STM32, ESP32 та RP2040 наведено в таблиці 10.4. Він показує, що хоча всі три платформи підтримують базовий циркулярний DMA, розвинені дескрипторні режими з ланцюжками доступні переважно на старших сімействах STM32 та частково на RP2040 через PIO.

Таблиця 10.4 – Порівняння можливостей DMA на платформах STM32H7, ESP32-S3 та RP2040

Платформи	Характеристика	Значення
STM32H7 (MDMA / DMA2D)	*Кількість каналів*	16 (MDMA) + два DMA (DMA1, DMA2) + BDMA + DMA2D (графічний прискорювач)
	Підтримка Linked List (дескрипторів)	так (MDMA підтримує ланцюжкові списки передач)
	Циркулярний режим	так (MDMA підтримує циркулярний режим через функцію HAL_MDMA_LinkedList_EnableCircularMode)
	Передача пам'ять-пам'ять	так (MDMA оптимізовано для передач між пам'ятями)
	Підтримка burst	так (MDMA підтримує налаштування burst-передач)
	Вимоги до вирівнювання	залежить від режиму (байт, півслово, слово)
	Типовий стек	HAL, LL
ESP32-S3 (GDMA)	*Кількість каналів*	5 незалежних каналів для передачі (TX) + 5 для прийому (RX)
	Підтримка Linked List (дескрипторів)	так (керування передачею через linked lists)
	Циркулярний режим	так (підтримується через конфігурацію дескрипторів)
	Передача пам'ять-пам'ять	так (підтримується на високій швидкості)

Кінець таблиці 10.4

Платформи	Характеристика	Значення
ESP32-S3 (GDMA)	*Підтримка burst*	так (підтримується, включаючи INCR4/INCR8)
	Вимоги до вирівнювання	4 байти (для SRAM); для PSRAM — 32 байти (розмір рядка кеша)
	Типовий стек	ESP-IDF
RP2040 (DMA)	*Кількість каналів*	12 незалежних каналів
	Підтримка Linked List (дескрипторів)	ні (окремі канали не підтримують ланцюжок кількох буферів)
	Циркулярний режим	так (реалізується через об'єднання двох DMA-каналів у режимі "ping-pong")
	Передача пам'ять-пам'ять	так (підтримується)
	Підтримка burst	обмежена (автоматичне розбиття на менші доступи при невирівняних адресах)
	Вимоги до вирівнювання	4 байти (передачі виконуються словами по 4 байти)
	Типовий стек	Pico SDK, TinyUSB

Механізми DMA, дескрипторів та кругових буферів є наріжним каменем високопродуктивної обробки даних у вбудованих системах. Вони дозволяють розвантажити центральний процесор, знизити латентність реакції на події, забезпечити безперервний потоковий обмін із периферією та ефективно використовувати пропускну здатність внутрішньокристальних шин. Володіння цими техніками, включно з розумінням обмежень, пов’язаних із кешуванням і вирівнюванням, є обов’язковою компетенцією інженера, який працює з мікроконтролерами класу ARM Cortex-M7 та вище. Цим завершується десятий розділ; наступна частина посібника перейде до практичних кейсів та архітектурних патернів побудови IoT-систем, де розглянуті механізми DMA будуть застосовані в реальних сценаріях.

ЧАСТИНА V.
СХЕМИ ВЗАЄМОДІЇ ТА АРХІТЕКТУРНІ ПАТЕРНИ

РОЗДІЛ 11.
ТИПОВІ АРХІТЕКТУРИ IOT-СИСТЕМ

11.1. Трирівнева модель: край (edge), туман (fog), хмара (cloud)

Трирівнева архітектурна модель «край — туман — хмара» (Edge — Fog — Cloud) є концептуальним каркасом, на який спирається переважна більшість сучасних IoT-систем промислового та споживчого призначення. Її поява зумовлена фундаментальною неможливістю передавати всі дані безпосередньо від мільйонів сенсорних вузлів до централізованих хмарних дата-центрів. Обмеження пропускної здатності каналів зв'язку, висока вартість стільникового трафіку, неприйнятна латентність для задач реального часу та критичні вимоги до автономності батарейних пристроїв диктують необхідність розподіленої, ієрархічної обробки даних. У цій моделі крайовий рівень (Edge) утворюють кінцеві пристрої — сенсорні вузли та актуатори, які безпосередньо взаємодіють із фізичним світом. Їхнє завдання — збір даних із датчиків, первинна фільтрація, агрегація та передавання на наступний рівень. Туманний рівень (Fog), названий так за аналогією з хмарою, що опустилася ближче до землі, утворюють проміжні шлюзи, концентратори та граничні маршрутизатори, які розташовані в безпосередній близькості до крайових пристроїв — у тій самій будівлі, цеху або на стовпі освітлення. Вони виконують трансляцію протоколів, буферизацію даних, локальну аналітику, прийняття рішень у реальному часі та виступають єдиною точкою виходу в глобальну мережу для десятків або сотень крайових вузлів. Хмарний рівень (Cloud) забезпечує довготривале зберігання даних, складну аналітику, машинне навчання на історичних даних, візуалізацію, інтеграцію з корпоративними системами (ERP, MES) та глобальне керування парком пристроїв. Ця трирівнева архітектура формалізована в рекомендаціях Industrial Internet Consortium (IIC): Industrial Internet Reference Architecture (IIRA, версія 1.9, 2022) та стандарті ISO/IEC 30141:2018 «Internet of Things — Reference Architecture», а також детально досліджена в оглядовій статті М. Сатріа та співавторів у IEEE Communications Surveys & Tutorials (том 25, 2023), де показано, що оптимальний розподіл обчислень між рівнями дозволяє знизити сумарний мережевий трафік на 60–80% порівняно з наївною моделлю «все в хмару».

11.1.1. Сенсорний вузол на мікроконтролері + шлюз на Linux-платформі + хмарний сервіс

Крайовий сенсорний вузол у такій архітектурі — це, як правило, мікроконтролер із низьким енергоспоживанням (наприклад, STM32L476 або ESP32-C3 у режимі глибокого сну), до якого через стандартні інтерфейси (I²C, SPI, UART, аналогові входи) підключено один або кілька датчиків. Його програмне забезпечення, часто побудоване на базі FreeRTOS або bare-metal, реалізує цикл «пробудження за таймером — опитування датчиків — валідація даних — формування пакета — передавання — повернення в сон». Обчислювальна потужність такого вузла обмежена (як правило, 32-бітне ядро з частотою до 80 МГц, 64–256 КБ ОЗП), тому складні алгоритми обробки сигналів або локальна інференція нейронних мереж можлива лише для найпростіших моделей (наприклад, TensorFlow Lite Micro для виявлення аномалій вібрації). Основним завданням крайового вузла є якомога швидше виконати необхідний мінімум операцій і повернутися в режим мікроватного споживання, щоб забезпечити багаторічну автономність від батареї.

Туманний шлюз, на відміну від крайового вузла, будується на значно потужнішій апаратній платформі, здатній виконувати повноцінну операційну систему Linux. Типовими представниками є одноплатні комп'ютери Raspberry Pi 4/5 (Broadcom BCM2712, 4–8 ГБ ОЗП), BeagleBone Black (TI AM335x), а також промислові шлюзи на базі SoC NXP i.MX 6/8 або TI Sitara. Наявність Linux надає шлюзу весь багатий арсенал мережевих стеків (TCP/IP, LwIP, BlueZ, Modbus, OPC UA), контейнеризації (Docker, Balena), підтримки скриптових мов (Python, Node.js) для швидкої розробки бізнес-логіки, а також файлових систем для буферизації даних (store-and-forward) на випадок втрати зв'язку з хмарою. Шлюз агрегує дані від крайових пристроїв через різноманітні протоколи — BLE, LoRaWAN (через підключений концентратор), Modbus RTU (через USB-to-RS485), Zigbee (через USB-координатор) — і транслює їх у єдиний протокол для передавання в хмару, найчастіше MQTT (з TLS) або HTTPS. На рисунку 11.1 показано типову структурну схему трирівневої системи для задачі моніторингу мікроклімату в теплиці, де крайові датчики (температура, вологість, освітленість) через BLE або LoRaWAN передають дані на Raspberry Pi-шлюз, який, своєю чергою, через стільниковий модем або Ethernet публікує їх на хмарний брокер MQTT (наприклад, AWS IoT Core).

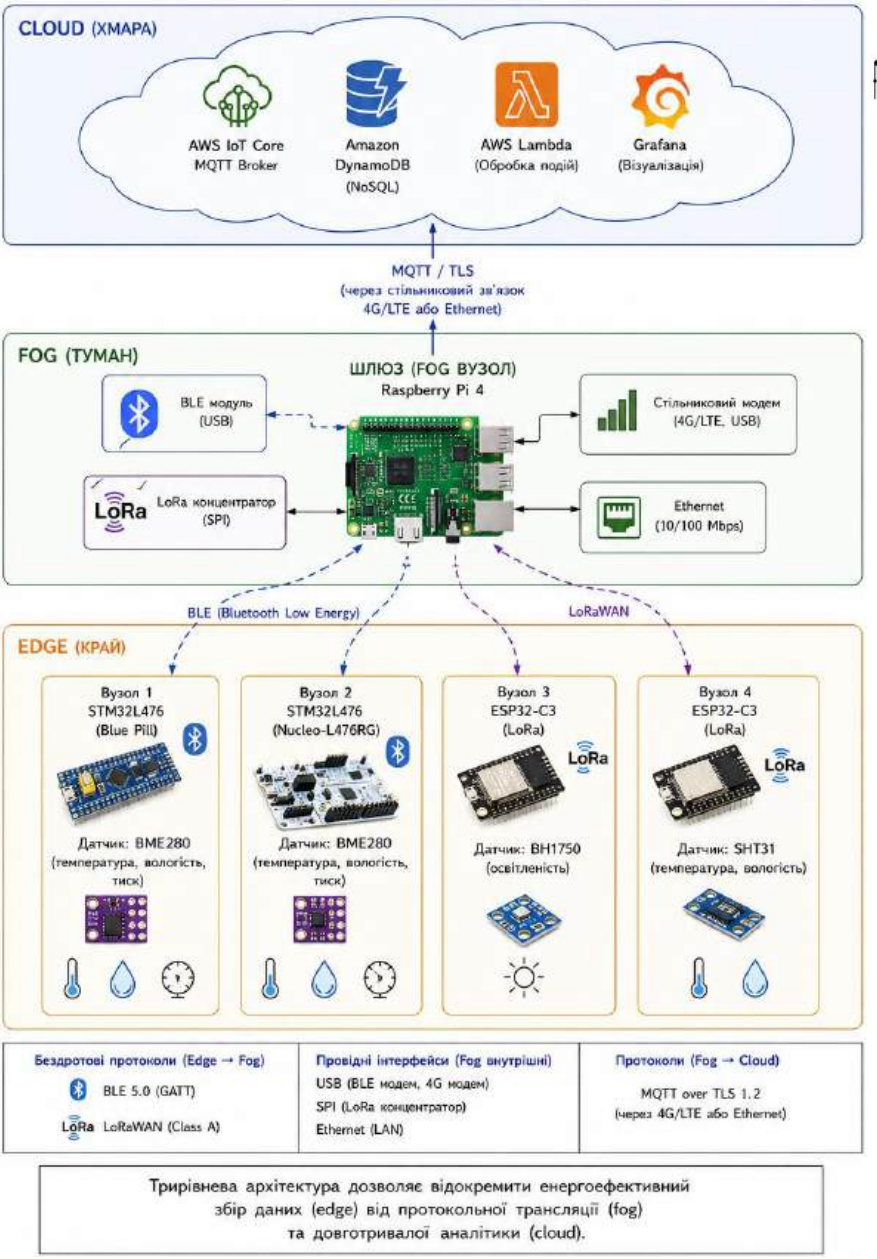


Рисунок 11.1– Структурна схема трирівневої IoT-системи (край – туман – хмара) на прикладі моніторингу мікроклімату

Хмарний рівень у цій моделі представлений керованими сервісами публічних хмарних провайдерів — Amazon Web Services (AWS IoT Core), Microsoft Azure (Azure IoT Hub), Google Cloud (Cloud IoT Core) — або приватними хмарними платформами, розгорнутими на власних серверах підприємства. Хмарний IoT-сервіс виконує функції автентифікації та авторизації пристроїв (через сертифікати X.509 або токени), приймання MQTT/HTTPS-повідомлень, маршрутизації даних до цільових сервісів зберігання (бази даних часових рядів, такі як AWS Timestream або InfluxDB, або об'єктні сховища, такі як AWS S3), запуску обчислювальних функцій у відповідь на події (AWS Lambda, Azure Functions), а також візуалізації через дашборди (Grafana, Kibana, AWS QuickSight). Ключовою перевагою хмарного рівня є еластичне масштабування: система автоматично адаптується до збільшення кількості пристроїв від сотень до мільйонів без зміни архітектури. Порівняння характеристик трьох рівнів за основними критеріями наведено в таблиці 11.1, яка дозволяє систематизувати вимоги до апаратного та програмного забезпечення на кожному з рівнів.

Таблиця 11.1 – Порівняльна характеристика рівнів edge, fog та cloud

Рівні	Характеристика	Значення
Edge (Край)	*Типова апаратна платформа*	STM32L4, ESP32-C3, nRF52840
	Операційна система	bare-metal, FreeRTOS
	Обсяг ОЗП	64–512 КБ
	Енергоспоживання	мкВт – мВт
	Протоколи зв'язку	BLE, LoRaWAN, Zigbee, UART
	Типова затримка обробки	<1 мс
	Основна функція	збір даних, фільтрація, сон
Fog (Туман)	*Типова апаратна платформа*	Raspberry Pi 4/5, i.MX 8M, TI Sitara
	Операційна система	Linux (Raspberry Pi OS, Yocto)
	Обсяг ОЗП	1–8 ГБ
	Енергоспоживання	3–15 Вт
	Протоколи зв'язку	Wi-Fi, Ethernet, LTE, RS-485
	Типова затримка обробки	десятки мс
	Основна функція	трансляція протоколів, локальна аналітика, буферизація
Cloud (Хмара)	*Типова апаратна платформа*	AWS, Azure, Google Clou (серверні стійки)
	Операційна система	керовані сервіси (Linux, Windows Server)
	Обсяг ОЗП	ГБ – ТБ (віртуалізовано)
	Енергоспоживання	кВт – МВт (дата-центр)
	Протоколи зв'язку	MQTT, HTTPS, AMQP, WebSocket
	Типова затримка обробки	сотні мс – секунди
	Основна функція	довготривале зберігання, ML-аналітика, візуалізація

Практична реалізація описаної моделі передбачає написання програмного забезпечення для кожного з рівнів. Для крайового вузла на STM32 це зазвичай код на мові C із використанням HAL, який виконує зчитування датчика (наприклад, BME280 через I²C), формує JSON-пакет (за допомогою бібліотеки cJSON або JSMN) та передає його через UART на підключений BLE-модуль (наприклад, HM-10) або безпосередньо через вбудоване радіо (якщо це ESP32-C3). У лістингу 11.1 наведено фрагмент такого коду для STM32L476.

Лістинг 11.1 – Формування та відправлення JSON-пакета з датчика на STM32L476 (HAL, cJSON)

```
// Відповідно до специфікації JSON (RFC 8259) та STM32CubeL4
#include "cJSON.h"
#include "main.h"
extern I2C_HandleTypeDef hi2c1; // Дескриптор I2C для датчика
extern UART_HandleTypeDef huart2; // Дескриптор UART для BLE
// ------- Читання даних з датчика (I2C) -------
/** * @brief Читання температури та вологості через I2C
* @note  HAL_I2C_Mem_Read() – читання з регістра датчика
* Адреса датчика, регістр та розмір даних залежать від моделі*/
float temperature, humidity;
uint8_t raw_data[4];
// Читання 4 байт з датчика (наприклад, HTS221)
HAL_I2C_Mem_Read(&hi2c1, 0xBE, 0x28, I2C_MEMADD_SIZE_8BIT,
raw_data, 4, 100);
temperature = (float)((int16_t)(raw_data[1] << 8 | raw_data[0])
) / 10.0f;
humidity    = (float)((int16_t)(raw_data[3] << 8 | raw_data[2]))
/ 10.0f;
// ------- Формування JSON-рядка (cJSON) ------
/** * @brief Створення JSON-об'єкта та перетворення на рядок
 * @note  cJSON_CreateObject() – створення кореневого об'єкта
cJSON_AddNumberToObject() – додавання числових
полів cJSON_PrintUnformatted() – мінімізація розміру рядка
 * Мінімальний JSON: {"t":23.5,"h":67.2} */
cJSON *root = cJSON_CreateObject();
 if (root) {
  cJSON_AddNumberToObject(root, "t", temperature);
  cJSON_AddNumberToObject(root, "h", humidity);
  char *json_str = cJSON_PrintUnformatted(root); // Мінімальний
рядок
  if (json_str) {
        // ------ Відправлення через UART -----
       HAL_UART_Transmit(&huart2, (uint8_t*)json_str, strlen
(json_str), 100);
       HAL_UART_Transmit(&huart2, (uint8_t*)"\r\n", 2, 100);
    cJSON_free(json_str); // Звільнення пам'яті рядка     }
  cJSON_Delete(root); // Звільнення JSON-об'єкта    }
```

Для туманного шлюзу на Raspberry Pi логіка часто реалізується на Python із використанням бібліотеки paho-mqtt для підключення до хмарного брокера. Шлюз читає дані з послідовного порту (куди надходять повідомлення від BLE-модуля або LoRa-концентратора), парсить їх, доповнює часовою міткою та ідентифікатором вузла, і публікує в тему MQTT. При цьому конфігурація TLS (сертифікати пристрою, завантажені з хмарної консолі) забезпечує захищений канал передавання. У лістингу 11.2 наведено мінімальний приклад такого шлюзового скрипта, що взаємодіє з AWS IoT Core.

Лістинг 11.2 – Шлюзовий скрипт на Python (Raspberry Pi) для публікації даних у AWS IoT Core (paho-mqtt, TLS)

```
# Відповідно до AWS IoT Core Developer Guide та paho-mqtt
import paho.mqtt.client as mqtt
import serial
import json
import time
# ------- Конфігурація AWS IoT Core -------
# Сертифікати генеруються в консолі AWS IoT Core
ENDPOINT   = "your-endpoint.iot.region.amazonaws.com" # AWS IoT
Core endpoint
PORT       = 8883        # MQTT over TLS
CA_CERT    = "./AmazonRootCA1.pem"    # Кореневий сертифікат CA
DEVICE_CERT = "./device_certificate.pem.crt" # Сертифікат
пристрою
PRIVATE_KEY = "./private.pem.key"      # Приватний ключ
TOPIC_PUB  = "sensors/greenhouse/data"  # Тема для публікації
TOPIC_SUB  = "commands/greenhouse"    # Тема для команд
# ------ UART на Raspberry Pi ------
ser = serial.Serial('/dev/serial0', 115200, timeout=1)
# ------ Callback-функції MQTT -----
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Підключено до AWS IoT Core")
        client.subscribe(TOPIC_SUB) # Підписка на команди
    else:
        print(f"Помилка підключення: {rc}")
def on_message(client, userdata, msg):
    print(f"Отримано команду: {msg.payload.decode()}")
# ------ Створення та налаштування клієнта ------
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
# Встановлення TLS-сертифікатів
client.tls_set(ca_certs=CA_CERT,
               certfile=DEVICE_CERT,
               keyfile=PRIVATE_KEY)
client.connect(ENDPOINT, PORT, keepalive=60)
client.loop_start() # Фонова обробка
# ------ Головний цикл ------
while True:
```

```
line = ser.readline().decode().strip()    # Читання з UART
if line:
  try:
    data = json.loads(line)    # Парсинг JSON
    client.publish(TOPIC_PUB, json.dumps(data)) # Публікація
       except Exception as e:
           print(f"Помилка: {e}")
   time.sleep(0.1)
```

Трирівнева модель «край – туман – хмара» є не просто теоретичною концепцією, а практичним, перевіреним інструментом побудови масштабованих IoT-систем. Вона дозволяє оптимально розподілити обчислювальні ресурси та енергоспоживання: найбільш ресурсообмежені пристрої виконують лише збір даних, потужніші шлюзи забезпечують протокольну сумісність і локальну обробку, а хмарні сервіси надають практично необмежені можливості для зберігання, аналітики та інтеграції. Наступний підрозділ продемонструє, як ця модель конкретизується на прикладі реального кейсу збору телеметрії через Modbus RTU, шлюз із MQTT та інтеграцію з AWS IoT Core.

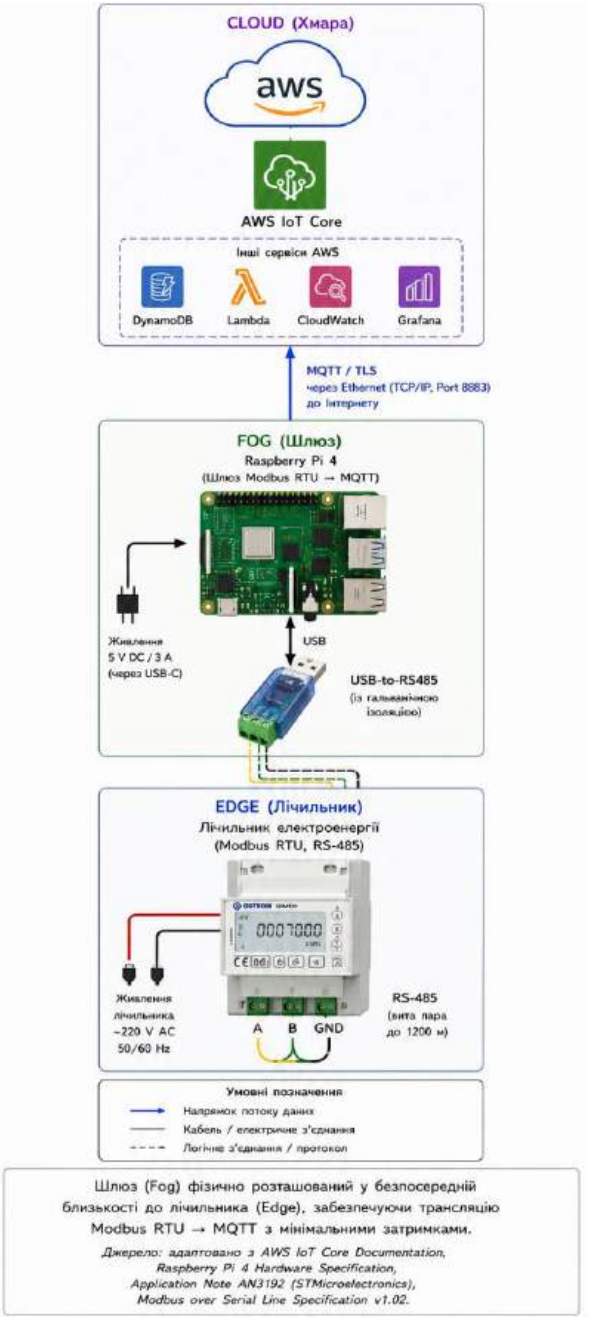
11.1.2. Приклад: збір телеметрії через RS-485/Modbus → шлюз із MQTT → AWS IoT Core

Для ілюстрації трирівневої моделі, розглянутої в попередньому підрозділі, доцільно розглянути конкретний наскрізний приклад, що поєднує стандартні промислові інтерфейси, програмні протоколи IoT та керовані хмарні сервіси. Об’єктом моделювання слугуватиме система дистанційного моніторингу електричних параметрів розподільчої трансформаторної підстанції. На підстанції встановлено багатофункціональний лічильник електроенергії (наприклад, Eastron SDM630-Modbus або аналогічний), що вимірює фазні напруги, струми, активну та реактивну потужності, коефіцієнт потужності та частоту. Лічильник оснащено інтерфейсом RS-485 і підтримує протокол Modbus RTU, виступаючи веденим пристроєм з адресою 0x01. Завдання системи полягає в періодичному (щохвилинному) зчитуванні цих параметрів, їхній попередній обробці, безпечній передачі до хмарної платформи AWS IoT Core для довготривалого зберігання, візуалізації та аналітики, а також у забезпеченні можливості віддаленого керування (наприклад, скидання лічильника після аварії). Цей кейс демонструє класичну триланкову архітектуру «край – туман – хмара», де крайовим пристроєм є сам лічильник із Modbus-інтерфейсом, туманним шлюзом слугує одноплатний комп’ютер на Linux (Raspberry Pi 4) із перетворювачем

USB-to-RS-485, а хмарним рівнем — AWS IoT Core зі зв'язаними сервісами.

Апаратна архітектура системи, показана на рисунку 11.2, є гранично простою. Багатофункціональний лічильник підключається до шини RS-485 через двопровідну виту пару (A, B) із дотриманням вимог TIA/EIA-485-A: на обох кінцях шини встановлено термінуючі резистори 120 Ом, а для підвищення завадостійкості передбачено резистори зсуву (pull-up на A, pull-down на B). Шина RS-485 з'єднується з одноплатним комп'ютером Raspberry Pi 4 через компактний перетворювач інтерфейсів USB-to-RS-485 (наприклад, FTDI FT232R + MAX485, або готовий модуль на базі CH340/MAX485), який з боку хоста виглядає як стандартний послідовний порт /dev/ttyUSB0. Живлення лічильника здійснюється від вимірювального кола (фазна напруга), а Raspberry Pi — від окремого джерела 5 В постійного струму. Важливо, що сам Raspberry Pi не виконує функцій крайового збору — він є суто шлюзом, який транслює Modbus RTU-запити у MQTT-повідомлення. Для забезпечення резервного каналу зв'язку з хмарою шлюз може бути обладнаний стільниковим модемом (наприклад, USB-модем Huawei E3372), однак у базовій конфігурації використовується дротовий Ethernet.

Програмне забезпечення шлюзу реалізоване на мові Python із використанням трьох ключових бібліотек: libmodbus (через обгортку pymodbus) для роботи з Modbus RTU, paho-mqtt для MQTT-комунікації та TLS-захисту, а також стандартних модулів json, time, logging для серіалізації та діагностики. Алгоритм роботи шлюзу є циклічним: кожні 60 секунд він формує Modbus-запит на читання блоку регістрів (функція 0x03 Read Holding Registers), починаючи з адреси 0x0000, довжиною 20 регістрів, які містять усі необхідні параметри згідно з картою регістрів лічильника. Отримавши відповідь, шлюз перевіряє контрольну суму Modbus (що виконується автоматично бібліотекою pymodbus), витягує значення, масштабує їх відповідно до специфікації виробника (наприклад, напруга передається в десятих частках вольта, тому ділиться на 10), формує JSON-пакет і публікує його в тему MQTT. JSON-пакет містить ідентифікатор пристрою, мітку часу в форматі ISO 8601 та словник зі значеннями параметрів, що забезпечує семантичну самодостатність даних. Для забезпечення безпеки транспортного каналу використовується TLS 1.3 із взаємною автентифікацією на основі X.509 сертифікатів, згенерованих у AWS IoT Core для конкретної «речі» (Thing).



Шлюз (Fog) фізично розташований у безпосередній близькості до лічильника (Edge), забезпечуючи трансляцію Modbus RTU → MQTT з мінімальними затримками.

Джерело: адаптовано з AWS IoT Core Documentation, Raspberry Pi 4 Hardware Specification, Application Note AN3192 (STMicroelectronics), Modbus over Serial Line Specification v1.02.

Рисунок 11.2– Апаратна архітектура системи збору телеметрії через RS-485/Modbus → шлюз Raspberry Pi → AWS IoT Core

Кореневий сертифікат центру сертифікації (Amazon Root CA 1), сертифікат пристрою та закритий ключ завантажуються на шлюз і передаються в paho-mqtt при встановленні з'єднання. Така схема гарантує, що лише авторизований пристрій із дійсним сертифікатом може публікувати дані в хмару, а також отримувати команди керування (downlink).

Хмарна частина системи реалізована на платформі AWS IoT Core. Для кожного лічильника в консолі AWS IoT створюється окрема «річ» (Thing), яка має унікальне ім'я (наприклад, Substation_123_Meter_01). До речі прив'язуються сертифікат пристрою та політика доступу (IoT Policy), яка явно дозволяє операції iot:Publish на тему telemetry/substation/+/meter/+ та iot:Subscribe на тему commands/substation/+/meter/+. Це реалізує принцип найменших привілеїв: скомпрометований сертифікат одного лічильника не дасть зловмиснику доступу до тем інших пристроїв. Після успішної публікації повідомлення на брокер MQTT (який є частиною AWS IoT Core), спрацьовує правило маршрутизації (IoT Rule), яке направляє дані до цільових сервісів: бази даних часових рядів AWS Timestream для довготривалого зберігання та аналітики, сервісу AWS Lambda для виявлення аномалій (наприклад, різкого падіння напруги) і, за необхідності, стрімінгового сервісу Amazon Kinesis для обробки в реальному часі. Візуалізація даних виконується за допомогою дашборду Grafana, підключеного до AWS Timestream. Для реалізації команд керування (наприклад, запис у регістр для скидання накопиченої енергії) використовується зворотний канал: оператор надсилає MQTT-повідомлення в тему commands, шлюз приймає його, валідує та виконує Modbus-запис (функція 0x06 Write Single Register), після чого публікує результат виконання в окрему тему. У таблиці 11.2 наведено узгоджену схему тем MQTT та їхнього призначення.

Таблиця 11.2 – Схема тем MQTT для системи збору телеметрії

Тема MQTT	Напрямок	Формат повідомлення	Призначення
telemetry/substation/{sub_id}/meter/{meter_id}	uplink (від шлюзу до хмари)	JSON з полями timestamp, voltage_l1, current_l1, power_total тощо	Періодичне передавання виміряних параметрів
commands/substation/{sub_id}/meter/{meter_id}	downlink (від хмари до шлюзу)	JSON з полями command, register, value	команда на запис у регістр (наприклад, скидання енергії)
commands/substation/{sub_id}/meter/{meter_id}/ack	uplink	JSON з полями status, error	підтвердження виконання команди

У лістингу 11.3 наведено спрощений, але функціонально повний код основного циклу шлюзу на Python. Він демонструє читання регістрів Modbus, формування JSON, публікацію через MQTT з TLS до AWS IoT Core, а також обробку вхідних команд. Код містить необхідні обробники помилок та логування для діагностики.

Лістинг 11.3 – Основний цикл шлюзу на Python (pymodbus + paho-mqtt + TLS) для чи-тання Modbus та публікації в AWS IoT Core

```
# Відповідно до Modbus RTU Specification та AWS IoT Core
Developer Guide
import paho.mqtt.client as mqtt
from pymodbus.client import ModbusSerialClient
import json
import time
import logging
# ------ Конфігурація AWS IoT Core -------
# Сертифікати X.509 генеруються в консолі AWS IoT Core
ENDPOINT    = "your-endpoint.iot.region.amazonaws.com"
PORT        = 8883    # MQTT over TLS
CA_CERT     = "./AmazonRootCA1.pem"    # Кореневий CA
DEVICE_CERT="./device_certificate.pem.crt"#Сертифікат пристрою
PRIVATE_KEY = "./private.pem.key"     # Приватний ключ
TOPIC_PUB   = "sensors/greenhouse/data"   # Тема публікації
TOPIC_SUB   = "commands/greenhouse"   # Тема команд
# ------- MQTT Callback ------
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Підключено до AWS IoT Core")
        client.subscribe(TOPIC_SUB)   # Підписка на команди
    else:  print(f"Помилка підключення: {rc}")
def on_message(client, userdata, msg):
    print(f"Отримано команду: {msg.payload.decode()}")
# ------ Налаштування MQTT клієнта ------
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
# Встановлення TLS (взаємна аутентифікація mTLS)
client.tls_set(ca_certs=CA_CERT,
               certfile=DEVICE_CERT,
               keyfile=PRIVATE_KEY)
client.connect(ENDPOINT, PORT, keepalive=60)
client.loop_start()   # Фонова обробка MQTT
# ----- Головний цикл -----
while True:
    # Створення Modbus RTU клієнта
    modbus_client = ModbusSerialClient(
        method='rtu',
        port='/dev/ttyUSB0',
        baudrate=9600,
        timeout=1
    )
    try:
        if modbus_client.connect():
```

```
    # Читання 20 Holding Registers (адреса 0x0000, slave ID=1)
rr = modbus_client.read_holding_registers(0x0000, 20, slave=1)
        if not rr.isError():    # Перевірка помилок
            data = {
                "temperature": rr.registers[0] / 10.0,
                "humidity":    rr.registers[1] / 10.0,
                "pressure":    rr.registers[2],
                "timestamp":   time.time()                }
            # Публікація JSON в AWS IoT Core
            client.publish(TOPIC_PUB, json.dumps(data))
        else:  print(f"Modbus помилка: {rr}")
    else: print("Не вдалося підключитися до Modbus пристрою")
except Exception as e:  print(f"Виняток: {e}")
finally:  modbus_client.close()   # Звільнення ресурсів
time.sleep(60)  # Опитування кожні 60 с
```

Зіставлення основних технічних параметрів компонентів системи наведено в таблиці 11.3, яка може слугувати шаблоном для документування аналогічних IoT-проектів.

Таблиця 11.3 – Технічні параметри компонентів системи збору телеметрії

Компонент	Параметр	Значення
Лічильник Eastron SDM630	Інтерфейс	RS-485, протокол Modbus RTU, адреса 0x01, 9600 бод, 8E1
Шлюз Raspberry Pi 4	ОС	Raspberry Pi OS Lite (Debian 12), Python 3.11, pymodbus 3.5, paho-mqtt 2.0
Перетворювач USB-RS485	Чіпсет	FTDI FT232R + MAX485, швидкість до 1 Мбіт/с
AWS IoT Core	Версія API	2024, TLS 1.3, політика з обмеженням тем

Розглянутий приклад демонструє завершений цикл збору, передавання, зберігання та керування в IoT-системі промислового призначення, що спирається на відкриті стандарти Modbus, MQTT та TLS. Саме така архітектура забезпечує баланс між надійністю польового рівня (RS-485/Modbus, що працює навіть за відсутності Інтернету), гнучкістю шлюзу (Linux із багатим вибором бібліотек) та масштабованістю хмари (AWS IoT Core). Наступний підрозділ, присвячений децентралізованим mesh-мережам, продемонструє альтернативний підхід, у якому шлюз не є обов'язковим елементом, а вузли самі формують стійку до відмов комунікаційну мережу.

11.2. Децентралізовані схеми: mesh-мережі Zigbee/Thread/BLE Mesh без постійного хмарного зв'язку

Трирівнева модель, розглянута в попередньому підрозділі, передбачає обов'язкову наявність центрального шлюзу та хмарного сервісу як кінцевої точки агрегації даних. Проте ціла низка практичних застосувань — від автоматизації будівель до промислового моніторингу — потребує децентралізованої взаємодії, за якої пристрої можуть безпосередньо обмінюватися даними між собою, ухвалювати локальні рішення та зберігати працездатність навіть за повної відсутності підключення до глобальної мережі. Mesh-мережі, побудовані на основі протоколів Zigbee, Thread або Bluetooth Mesh, реалізують саме таку парадигму: кожен вузол є одночасно джерелом, приймачем і потенційним ретранслятором даних, а топологія мережі формується динамічно, без центрального адміністратора. У цьому підрозділі ми розглянемо спільні принципи та відмінні риси трьох провідних децентралізованих mesh-технологій, їхню здатність функціонувати автономно, без неперервного хмарного зв'язку, а також наведемо практичний приклад побудови такої системи.

Спільною рисою Zigbee, Thread та Bluetooth Mesh є те, що всі вони спираються на фізичний рівень IEEE 802.15.4 (у випадку BLE Mesh — на розширення рекламного механізму BLE) і використовують механізми багатокрокової маршрутизації для передавання повідомлень від джерела до адресата через проміжні вузли. На відміну від топології «зірка», де вихід із ладу центрального хабу миттєво паралізує всю мережу, mesh-архітектура забезпечує природну відмовостійкість: при виході з ладу одного з маршрутизаторів мережа автоматично знаходить альтернативний шлях. Таке самовідновлення відбувається без втручання людини, за лічені секунди або навіть мілісекунди, що є критичним для систем безпеки, охоронної сигналізації та керування освітленням. Крім того, саме локальна природа mesh-мереж дозволяє знизити затримки до одиниць або десятків мілісекунд, оскільки пакети не покидають межі локальної мережі й не проходять через хмарні сервери.

Zigbee, стандартизований Connectivity Standards Alliance, є одним із найстаріших і найпоширеніших протоколів для децентралізованих бездротових мереж малого радіусу дії. Його архітектура передбачає три типи вузлів: координатор (один на мережу), маршрутизатори (постійно активні, ретранслюють пакети) та кінцеві пристрої (зазвичай батарейні, можуть перебувати в сплячому режимі). Ключовим аспектом автономності Zigbee є те, що після початкового введення в експлуатацію (commissioning) мережа не

потребує жодного центрального сервера для щоденної роботи: усі групові адреси, сценарії автоматизації та зв'язування пристроїв (binding) можуть бути запрограмовані безпосередньо в пам'яті вузлів. Наприклад, вимикач Zigbee може безпосередньо керувати лампою через кластер On/Off, навіть якщо координатор тимчасово недоступний, оскільки зв'язування було збережене під час налаштування. Це робить Zigbee ідеальним вибором для систем розумного освітлення, де затримка реакції на натискання вимикача має бути суб'єктивно миттєвою (<200 мс) і не повинна залежати від стану Wi-Fi-роутера чи хмарного сервісу.

Thread, специфікований Thread Group, також базується на IEEE 802.15.4, але використовує IPv6 як мережевий протокол, що надає кожному вузлу унікальну глобальну або локальну IPv6-адресу. На відміну від Zigbee, де для виходу в IP-мережу потрібен шлюз трансляції, Thread пристрої можуть безпосередньо обмінюватися IPv6-пакетами всередині mesh-мережі. Важливо, що Thread не вимагає постійного підключення до хмари: пристрої можуть взаємодіяти між собою через Mesh-Local IPv6 адреси, використовуючи UDP як транспорт. Якщо ж потрібен зв'язок із зовнішнім світом, один або кілька граничних маршрутизаторів (Thread Border Router) можуть бути підключені до Ethernet або Wi-Fi, але їхня відмова не порушує локальної комунікації всередині мережі. Thread також є обов'язковою транспортною основою для протоколу Matter, що забезпечує інтероперабельність пристроїв різних виробників на прикладному рівні без прив'язки до конкретної хмарної платформи.

Bluetooth Mesh, у свою чергу, використовує принципово інший підхід до маршрутизації — кероване лавиноподібне поширення (managed flooding), за якого повідомлення транслюються через усі вузли-ретранслятори, а не за заздалегідь обчисленими маршрутами. Це спрощує реалізацію й усуває необхідність зберігання таблиць маршрутизації, однак обмежує масштаб мережі сотнями вузлів. Особливістю BLE Mesh є його тісна інтеграція з екосистемою смартфонів: будь-який сучасний телефон може виступати в ролі пристрою введення в експлуатацію (provisioner) та інтерфейсу керування, після чого мережа функціонує повністю автономно. Наприклад, система освітлення на BLE Mesh дозволяє налаштувати групи, сцени та розклади через додаток на смартфоні; після збереження конфігурації у вузлах телефон більше не потрібен, і мережа працює роками без жодного зовнішнього контролера.

Узагальнене порівняння трьох децентралізованих mesh-технологій за ключовими параметрами наведено в таблиці 11.4, яка дозволяє обрати

оптимальний протокол залежно від вимог до масштабу, швидкості, енергоспоживання та сумісності.

Таблиця 11.4 – Порівняння децентралізованих mesh-протоколів Zigbee, Thread та BLE Mesh

Протокол	Характеристика	Значення
Zigbee 3.0	*Фізичний рівень*	IEEE 802.15.4 (2,4 ГГц, 250 кбіт/с; субгігагерцеві діапазони 868/915 МГц – опціонально)
	Мережевий рівень	Zigbee NWK (пропрієтарний)
	Метод маршрутизації	ієрархічна (Cluster-Tree) + AODV (Ad-hoc On-demand Distance Vector)
	Максимальна кількість вузлів	до 65 535 (теоретично)
	Енергоспоживання (батарейний вузол)	дуже низьке (ZED – Zigbee End Device)
	Затримка (типова, 1–3 хопи)	десятки мс (≤50 мс)
	Залежність від хмари	відсутня (локальна робота)
	Процедура введення в експлуатацію	через Zigbee Coordinator (Trust Center)
	Сумісність з Matter	через Matter Bridge
	Приклад застосування	розумне освітлення, сенсорні мережі, домашня автоматизація
Thread 1.3	*Фізичний рівень*	IEEE 802.15.4 (2,4 ГГц)
	Мережевий рівень	IPv6 + 6LoWPAN
	Метод маршрутизації	дистанційно-векторна (Mesh Routing)
	Максимальна кількість вузлів	до 250 (практично)
	Енергоспоживання (батарейний вузол)	дуже низьке (SED – Sleepy End Device)
	Затримка (типова, 1–3 хопи)	десятки мс (найкраща ефективність)
	Залежність від хмари	відсутня (локальна робота)
	Процедура введення в експлуатацію	через Commissioner (Border Router)
	Сумісність з Matter	так (нативна)
	Приклад застосування	розумний дім (Matter), промисловий IoT

Кінець таблиці 11.4

Протокол	Характеристика	Значення
BLE Mesh 1.0.1	*Фізичний рівень*	Bluetooth Low Energy (2,4 ГГц)
	Мережевий рівень	BLE Mesh Bearer Layer
	Метод маршрутизації	managed flooding (кероване лавиноподібне поширення)
	Максимальна кількість вузлів	до 32 767 (теоретично)
	Енергоспоживання (батарейний вузол)	низьке (LPN – Low Power Node)
	Затримка (типова, 1–3 хопи)	десятки–сотні мс
	Залежність від хмари	відсутня (локальна робота)
	Процедура введення в експлуатацію	через смартфон (Provisioner)
	Сумісність з Matter	через Matter Bridge
	Приклад застосування	освітлення, трекінг активів, сенсорні мережі

Приклад децентралізованої системи на основі Zigbee можна уявити на складі логістичного центру, де сотні світильників і датчиків руху об'єднані в єдину mesh-мережу. Координатор Zigbee, підключений до локального контролера, один раз конфігурує мережу (призначає адреси, встановлює ключі безпеки), після чого кожен світильник, що є маршрутизатором, ретранслює команди від вимикачів до сусідніх світильників. Датчики руху (кінцеві пристрої на батареях) прокидаються за подією, надсилають повідомлення про присутність, і найближчий маршрутизатор негайно передає його до відповідної групи світильників — усе це відбувається за час, менший за 100 мс, без виходу в Інтернет. Якщо склад розширюється, нові світильники автоматично приєднуються до мережі через найближчого сусіда. Топологію такої мережі з кількома стрибками ілюструє рисунок 11.3.

Практичне створення децентралізованого BLE Mesh-вузла на платформі ESP32 з використанням фреймворку ESP-IDF демонструє лістинг 11.4. У наведеному фрагменті виконується ініціалізація BLE Mesh-стеку, реєстрація моделі Generic OnOff Server (що відповідає за керування увімкненням/вимкненням) та автоматичне приєднання до мережі після успішного введення в експлуатацію через смартфон.

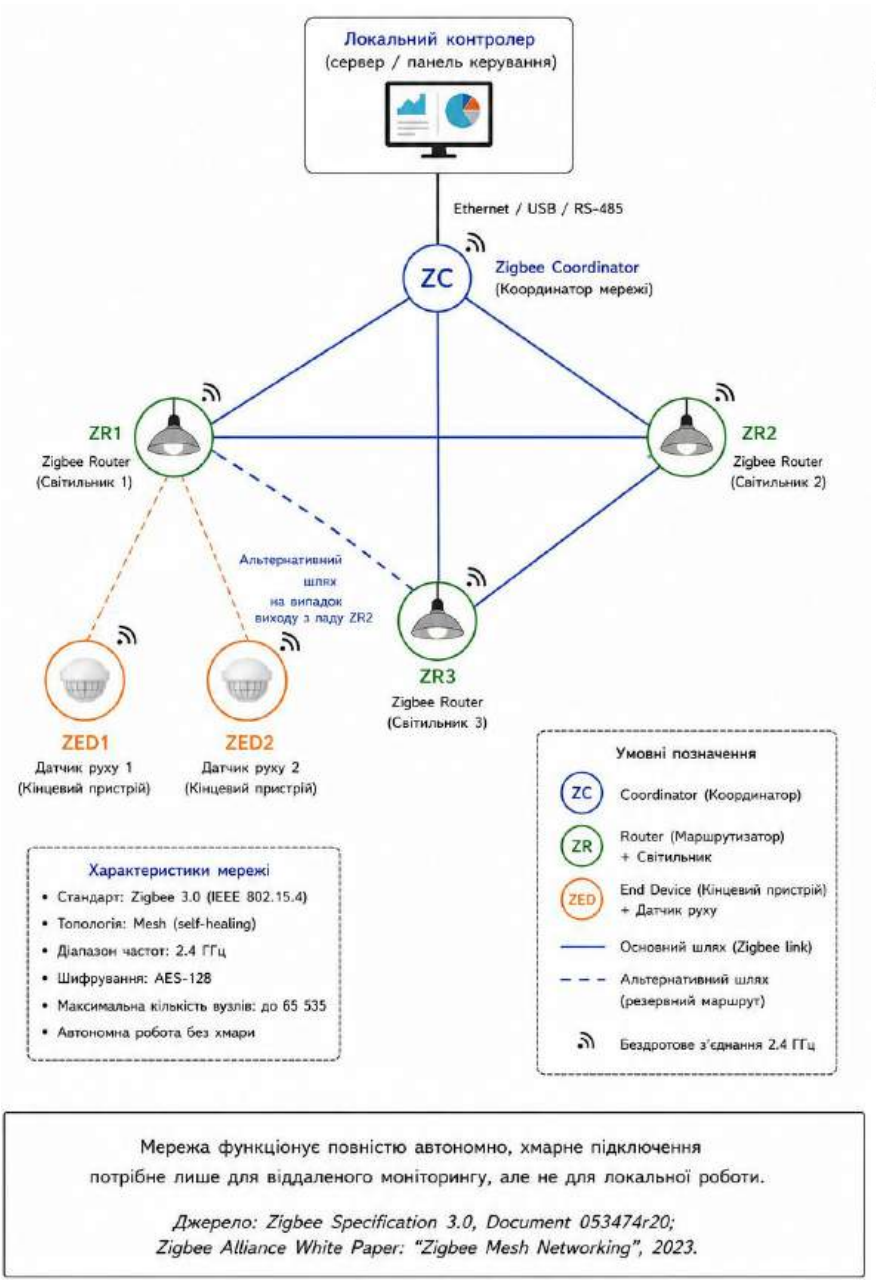


Рисунок 11.3– Децентралізована Zigbee mesh-мережа для керування освітленням складу

Лістинг 11.4 – Створення BLE Mesh-вузла з моделлю Generic OnOff Server на ESP32 (ESP-IDF)

```
// Відповідно до ESP-IDF BLE Mesh API та прикладу onoff server
#include <stdio.h>
#include "esp_ble_mesh_defs.h"
#include "esp_ble_mesh_common_api.h"
#include "esp_ble_mesh_networking_api.h"
#include "esp_ble_mesh_config_model_api.h"
#include "esp_ble_mesh_generic_model_api.h"
// ------ GPIO конфігурація ------
#define LED_GPIO  22  // Пін для керування світлодіодом
// ------ Реєстрація моделі та callback ------
/** * @brief Callback для обробки команд Generic OnOff Server
 * @param model  Вказівник на модель
 * @param ctx    Контекст повідомлення
 * @param buf    Буфер з даними
 * @param len    Довжина даних
 *  Аналізується opcode: ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET
 *  При отриманні команди встановлюється стан GPIO */
static esp_err_t generic_server_cb(esp_ble_mesh_model_t *model,
                                   esp_ble_mesh_msg_ctx_t *ctx,
                                      void *buf, size_t len) {
 esp_ble_mesh_generic_server_recv_op_t *recv_op = buf;
 switch (recv_op->opcode) {
   case ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET:
   case ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET_UN: {
          esp_ble_mesh_generic_server_state_change_t *state =
recv_op->state;
   bool onoff = state->onoff_set.onoff; // Отримання стану (0/1)
   gpio_set_level(LED_GPIO, onoff); // Керування світлодіодом
            ESP_LOGI(TAG, "LED %s", onoff ? "ON" : "OFF");
            break; }  default:  break;  }  return ESP_OK; }
// ------Ініціалізація BLE Mesh ------
/** * @brief Створення елемента з моделями Configuration Server
та Generic OnOff Server
* @note  Моделі реєструються в структурі esp_ble_mesh_model_t
* Після provisioning через nRF Mesh вузол стає частиною мережі*/
static esp ble mesh model t root models[] = {
    ESP_BLE_MESH_MODEL_CFG_SRV,     // Configuration Server
    ESP_BLE_MESH_MODEL_GEN_ONOFF_SRV(NULL, generic_server_cb)
// Generic OnOff Server   };
static esp_ble_mesh_elem_t elements[] = {
       ESP_BLE_MESH_ELEMENT(0, root_models,
ESP_BLE_MESH_MODEL_NONE),    };
static esp_ble_mesh_comp_t composition = {
    .element_count = ARRAY_SIZE(elements),
    .elements = elements,   };
void app_main(void) {
    gpio_reset_pin(LED_GPIO); // 1. Ініціалізація GPIO
    gpio_set_direction(LED_GPIO, GPIO_MODE_OUTPUT);
    // 2. Ініціалізація BLE Mesh
    esp_ble_mesh_init(&composition, NULL);
    // 3. Після provisioning вузол готовий до роботи в
децентралізованій мережі
    // Команди On/Off від будь-якого клієнта викликають callback
}
```

Децентралізовані mesh-мережі є архітектурною альтернативою хмарно-орієнтованим IoT-системам, особливо в сценаріях, де критичними є низька латентність, живучість за відсутності Інтернету та здатність до самоорганізації. Zigbee пропонує найбагатший набір готових прикладних профілів і перевірену десятиліттями екосистему, Thread забезпечує нативну IP-інтеграцію та безпеку мережевого рівня, а Bluetooth Mesh вирізняється простотою введення в експлуатацію через смартфон і низьким енергоспоживанням. Вибір конкретної технології визначається масштабом мережі, вимогами до сумісності з існуючими протоколами (насамперед Matter) та необхідним ступенем автономності. Наступний підрозділ, присвячений гетерогенним системам, покаже, як mesh-мережі можуть бути інтегровані з потужнішими обчислювальними платформами (MCU + FPGA + GPU) у єдину розподілену архітектуру.

11.3. Гетерогенні системи: співпраця MCU, FPGA, GPU через PCIe/USB/Ethernet (апаратне прискорення нейромереж)

Традиційна парадигма вбудованих систем, де один мікроконтролер виконує всі функції — від збору даних до прийняття рішень, — сягає своєї межі, коли вимоги до обчислювальної потужності зростають на порядки. Поява периферійного штучного інтелекту (Edge AI), що вимагає виконання глибоких нейронних мереж безпосередньо на місці збору даних, зумовила перехід до гетерогенних архітектур, де обчислювальні ресурси різної природи — мікроконтролери (MCU), програмовані логічні інтегральні схеми (FPGA) та графічні процесори (GPU) — об'єднуються в єдину систему через високошвидкісні інтерфейси. Кожен із цих компонентів має унікальні переваги: MCU забезпечує детерміноване керування в реальному часі та наднизьке енергоспоживання в режимі сну; FPGA пропонує апаратний паралелізм із мінімальною латентністю та можливість реконфігурації «на льоту»; GPU забезпечує найвищу пікову продуктивність для операцій матричного множення, що є основою нейромережевої інференції. Завдання системного архітектора полягає в оптимальному розподілі обчислювального навантаження між цими доменами та виборі інтерфейсів, які з'єднують їх у єдиний конвеєр обробки даних без вузьких місць. У цьому підрозділі розглядаються типові топології гетерогенних систем, інтерфейси міжкомпонентної взаємодії (PCIe, USB, Ethernet) та конкретний кейс апаратного прискорення нейромереж на прикладі зв'язки «MCU + FPGA + GPU».

Гетерогенна система для Edge AI зазвичай будується навколо одного з трьох архітектурних шаблонів. Перший шаблон — «MCU як хост,

FPGA/GPU як прискорювач» — передбачає, що мікроконтролер (або вбудований процесор класу ARM Cortex-A) виконує роль центрального координатора: він зчитує дані з датчиків, виконує попередню обробку (фільтрацію, нормалізацію), надсилає підготовлений тензор на прискорювач через PCIe або USB, отримує результат інференції та ухвалює рішення. Другий шаблон — «FPGA як центральний комутатор» — використовує FPGA не лише для обчислень, а й як апаратний міст між високошвидкісними АЦП/ЦАП, GPU (через PCIe) та MCU (через SPI/UART). Третій шаблон — «повністю розподілена система» — передбачає, що MCU, FPGA та GPU з'єднані через Ethernet (зокрема, TSN) і працюють як розподілені вузли, що обмінюються даними через стандартні протоколи (TCP/IP, DDS, OPC UA PubSub). Вибір шаблону залежить від фізичних обмежень (розмір плати, тепловиділення), вимог до латентності та пропускної здатності.

Ключову роль у гетерогенних системах відіграють високошвидкісні інтерфейси, розглянуті в попередніх розділах посібника. PCI Express (PCIe) є безальтернативним вибором для з'єднання GPU або потужної FPGA з хост-процесором, оскільки забезпечує пропускну здатність до 64 ГТ/с на лінію (PCIe 6.0) з мінімальною латентністю та підтримкою прямого доступу до пам'яті (DMA). Саме через PCIe GPU отримує тензори для інференції та повертає результати. USB 3.2 (SuperSpeed, 10 Гбіт/с) або USB4 (до 40 Гбіт/с) часто використовується для підключення зовнішніх прискорювачів (наприклад, Intel Neural Compute Stick, Google Coral USB Accelerator) до вбудованих платформ, які не мають PCIe, — таких як Raspberry Pi 4 або Jetson Nano. Ethernet (1GbE/10GbE) застосовується у розподілених гетерогенних системах, де вузли фізично рознесені (наприклад, камера з FPGA на щоглі, сервер із GPU в апаратній). У таких випадках критичним стає використання TSN для гарантованої латентності.

Розглянемо конкретний кейс: систему відеоаналітики на виробничій лінії, яка виявляє дефекти продукції в реальному часі. Система складається з камери (MIPI CSI-2), MCU (STM32H723 або Raspberry Pi 4) як хоста, FPGA (Xilinx Artix-7 або Lattice ECP5) для попередньої обробки зображень (resize, normalization) та GPU (NVIDIA Jetson Orin Nano) для виконання інференції згорткової нейронної мережі (CNN). Потік даних виглядає так: камера передає кадри через MIPI на MCU; MCU через USB 3.0 або PCIe x1 пересилає кадр у FPGA; FPGA апаратно виконує білінійну інтерполяцію до розміру 224×224 пікселів та нормалізацію (віднімання середнього, ділення на дисперсію), перетворюючи кадр на тензор; FPGA через PCIe x4 (або через спільну пам'ять DDR, якщо реалізація на базі Zynq MPSoC) передає готовий тензор на GPU; GPU виконує інференцію (наприклад, MobileNetV3)

і повертає вектор імовірностей класів (дефект / норма) назад на MCU через PCIe; MCU приймає рішення про відбракування та керує виконавчим механізмом. Архітектуру такої системи зображено на рисунку 11.4.

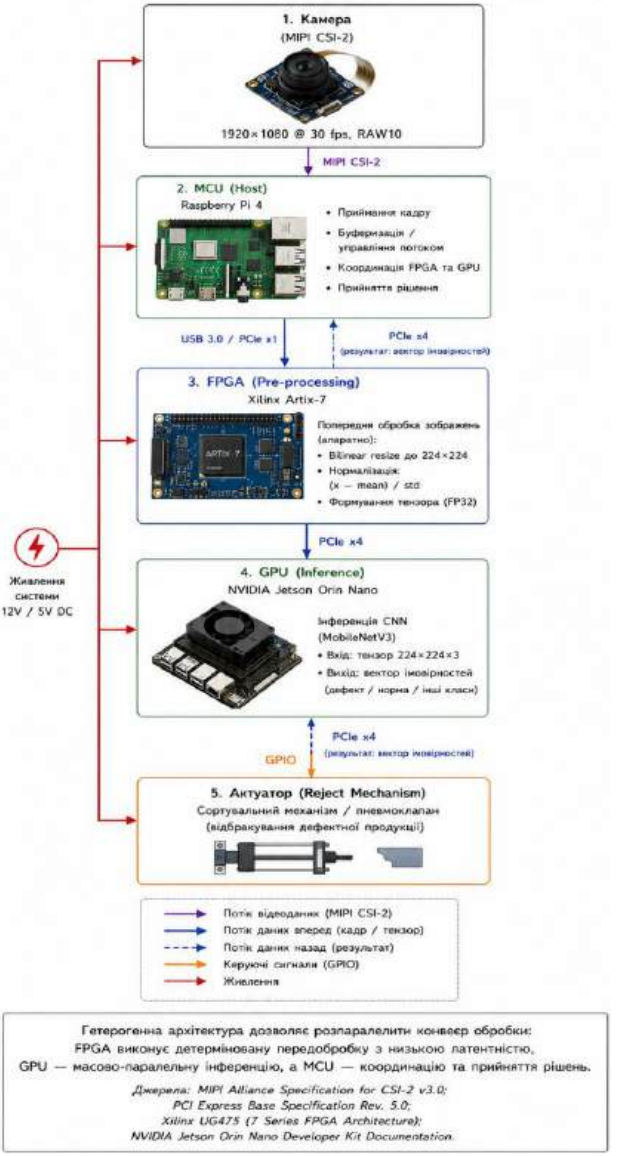


Рисунок 11.4– Архітектура гетерогенної системи Edge AI: MCU (хост) + FPGA (передоб-робка) + GPU (інференція)

Порівняння ключових характеристик MCU, FPGA та GPU як елементів гетерогенної системи для Edge AI наведено в таблиці 11.5. Дані базуються на технічній документації виробників (STMicroelectronics, Xilinx/AMD, NVIDIA) та оглядовій статті А. Реза та співавторів «Heterogeneous Computing for Edge AI: A Survey» (IEEE Access, том 12, 2024).

Таблиця 11.5 – Порівняння MCU, FPGA та GPU як компонентів гетерогенної системи Edge AI

Компоненти	Характеристика	Значення
MCU (напр., STM32H723)	*Обчислювальна модель*	послідовне ядро (Cortex-M7)
	Продуктивність (INT8, TOPS)	~0.001 TOPS (оцінка)
	Латентність (типова)	<1 мкс
	Енергоспоживання (пікове)	~0.5 Вт
	Гнучкість (перепрограмування)	висока (C/C++, RTOS)
	Інтерфейси	SPI, I²C, UART, USB, Ethernet
	Типова роль у системі	координація, збір даних, керування, прості ML-моделі (TFLM)
FPGA (напр., Artix-7 / Zynq-7020)	*Обчислювальна модель*	масово-паралельна логіка (LUT, DSP)
	Продуктивність (INT8, TOPS)	до ~0.264 TOPS (264 GOPS)
	Латентність (типова)	одиниці мкс
	Енергоспоживання (пікове)	~2–10 Вт
	Гнучкість (перепрограмування)	дуже висока (HDL, HLS — апаратна реконфігурація)
	Інтерфейси	PCIe, USB, Ethernet, LVDS, MIPI
	Типова роль у системі	детермінована передобробка, кастомні оператори, злиття сенсорів
GPU (напр., Jetson Orin Nano)	*Обчислювальна модель*	масово-паралельні ядра (CUDA/Tensor Cores)
	Продуктивність (INT8, TOPS)	до 67 TOPS (sparse INT8)
	Латентність (типова)	сотні мкс – одиниці мс (batch)
	Енергоспоживання (пікове)	~7–25 Вт
	Гнучкість (перепрограмування)	середня (CUDA, TensorRT — програмна)
	Інтерфейси	PCIe, USB, Ethernet, HDMI, CSI
	Типова роль у системі	інференція глибоких нейронних мереж

Програмна інтеграція гетерогенної системи вимагає спеціалізованих фреймворків. Для зв'язки MCU + FPGA часто використовують Xilinx Vitis Unified Software Platform або відкритий фреймворк PYNQ (Python Productivity for Zynq), який дозволяє керувати FPGA-прискорювачами з Python-коду на хостовому процесорі через AXI-шину та DMA. Для систем із GPU стандартом де-факто є NVIDIA CUDA Toolkit та TensorRT — оптимізатор інференції, який виконує квантування, злиття шарів та планування виконання для мінімізації латентності. У лістингу 11.5 наведено фрагмент коду на Python із використанням PyCUDA та TensorRT, який демонструє завантаження оптимізованого плану інференції на GPU та виконання передбачення для тензора, отриманого від FPGA через спільний буфер у пам'яті (DMA).

Лістинг 11.5 – Інференція нейромережі на GPU (TensorRT) з отриманням тензора від FPGA через DMA (Python, PyCUDA)

```
# Джерела: NVIDIA TensorRT Python API, PyCUDA, Linux UIO
import tensorrt as trt
import pycuda.driver as cuda
import pycuda.autoinit
import numpy as np
import mmap
import os
# ------ 1. Завантаження TRT-двигуна ------
# Десеріалізація попередньо збереженого двигуна з файлу .trt
TRT_LOGGER = trt.Logger(trt.Logger.WARNING)
with open("model.trt", "rb") as f:
    runtime = trt.Runtime(TRT_LOGGER)
    engine = runtime.deserialize_cuda_engine(f.read())
# Створення контексту виконання
context = engine.create_execution_context()
# ------ 2. Отримання тензора від FPGA через DMA -----
# Буфер DMA відображено в userspace через UIO-драйвер
# MCU заповнює його через спільну DDR-пам'ять FPGA та CPU
UIO_DEVICE = "/dev/uio0"
BUFFER_SIZE = 1024 * 1024  # 1 МБ (залежить від розміру тензора)
with open(UIO_DEVICE, "rb") as uio_fd:
    # Мапінг DMA-буфера у Python
    dma_buffer = mmap.mmap(uio_fd.fileno(), BUFFER_SIZE,
                           mmap.MAP_SHARED, mmap.PROT_READ)
    # Перетворення сирих даних у numpy-масив (вхідний тензор)
    # Тип/розмір залежать від моделі: float32, (1,3,224,224)
    input_data = np.frombuffer(dma_buffer, dtype=np.float32)
.reshape(1, 3, 224, 224)
# ------- 3. Виділення пам'яті на GPU ------
# Виділення пам'яті для вхідних та вихідних даних
input_size = input_data.nbytes
output_size = 1000 * 4  # 1000 класів, float32
d_input = cuda.mem_alloc(input_size)
d_output = cuda.mem_alloc(output_size)
# Створення буферів хоста для результату
```

```
h_output = np.empty((1000,), dtype=np.float32)
# ------ 4. Копіювання даних на GPU ------
# Копіювання вхідного тензора з DMA-буфера у пам'ять GPU
cuda.memcpy_htod(d_input, input_data)
# ------ 5. Виконання інференції --------
# Асинхронне виконання з використанням потоку
stream = cuda.Stream()
bindings = [int(d_input), int(d_output)]
context.execute_async_v2(bindings=bindings, stream_handle
=stream.handle)
stream.synchronize()
# ------ 6. Копіювання результату назад у хост ------
cuda.memcpy_dtoh(h_output, d_output)
# ------ 7. Обробка результату ------
predicted_class = np.argmax(h_output)
print(f"Predicted class: {predicted_class}")
dma_buffer.close() # Звільнення ресурсів
```

Додатковим важливим аспектом є синхронізація даних між різними доменами. Якщо FPGA та GPU мають доступ до спільної DDR-пам’яті (наприклад, у Zynq MPSoC), необхідно забезпечити когерентність кешів. У Zynq для цього використовується апаратний Snoop Control Unit (SCU) між ARM Cortex-A та FPGA, а також спеціальні інструкції (Data Synchronization Barrier, DSB) після DMA-передач. Якщо ж пристрої з’єднані через PCIe, застосовуються стандартні механізми PCIe MSI/MSI-X переривань для сповіщення хоста про завершення передачі. У розподілених системах на основі Ethernet синхронізація забезпечується через протокол PTP (IEEE 1588/802.1AS), розглянутий у підрозділі 5.2.3.

Порівняння трьох архітектурних шаблонів гетерогенних систем за ключовими критеріями наведено в таблиці 11.6, яка допомагає обрати оптимальну топологію залежно від вимог до латентності, пропускної здатності, вартості та складності розроблення.

Таблиця 11.6 – Порівняння архітектурних шаблонів гетерогенних IoT-систем

Шаблони	Характеристика	Значення
MCU + FPGA через PCIe/USB	*Основне обчислювальне навантаження*	FPGA — передобробка, кастомні оператори
	Латентність конвеєра	дуже низька (<1 мс)
	Пропускна здатність інтерфейсу	USB 3.0 (5 Гбіт/с) або PCIe x1 (8 Гбіт/с)
	Складність розроблення ПЗ	висока (HDL, HLS, драйвери DMA)
	Енергоспоживання (середнє)	низьке (FPGA без GPU)
	Типовий приклад	швидкісний аналіз сигналів, SDR

Кінець таблиці 11.6

Шаблони	Характеристика	Значення
MCU + GPU через PCIe/USB	*Основне обчислювальне навантаження*	GPU — інференція
	Латентність конвеєра	низька (1–10 мс)
	Пропускна здатність інтерфейсу	PCIe x4 (64 Гбіт/с для Gen4)
	Складність розроблення ПЗ	середня (CUDA/TensorRT, готові бібліотеки)
	Енергоспоживання (середнє)	середнє (GPU активний)
	Типовий приклад	відеоаналітика, виявлення об’єктів
MCU + FPGA + GPU через PCIe/Ethernet	*Основне обчислювальне навантаження*	FPGA + GPU — конвеєрна обробка
	Латентність конвеєра	дуже низька — низька
	Пропускна здатність інтерфейсу	PCIe x4 + Ethernet TSN
	Складність розроблення ПЗ	дуже висока (координація, синхронізація, когерентність)
	Енергоспоживання (середнє)	високе (обидва прискорювачі)
	Типовий приклад	промисловий Edge AI, робототехніка, автономні системи

Узагальнюючи, гетерогенні системи є закономірним еволюційним кроком вбудованих обчислень, зумовленим потребою поєднати енергоефективність MCU, гнучкість FPGA та потужність GPU в єдиному пристрої. Успішна реалізація таких систем вимагає від інженера не лише знання окремих інтерфейсів, а й глибокого розуміння архітектурної взаємодії між ними — від фізичного рівня (PCIe lanes, USB SS, Ethernet PHY) до програмного (DMA-дескриптори, когерентність кешів, TensorRT-планувальники). Саме така міждисциплінарна компетенція, що охоплює апаратне, програмне та системне проектування, визначає профіль сучасного розробника вбудованих систем. Цим завершується розділ типових архітектур IoT-систем; наступна частина посібника буде присвячена практичним кейсам, у яких розглянуті в попередніх розділах інтерфейси об’єднуються в завершені інженерні рішення.

РОЗДІЛ 12.
ПРАКТИЧНІ КЕЙСИ ВЗАЄМОДІЇ

12.1. “Інтелектуальний датчик”: BME688 по I²C, обробка на Cortex-M4, передача по BLE до смартфону

Практичне опанування інтерфейсів, розглянутих у попередніх розділах, найкраще закріплюється через наскрізні приклади, що поєднують апаратне підключення, низькорівневе програмування, алгоритмічну обробку даних та бездротову передачу. Першим таким кейсом є створення автономного інтелектуального датчика якості повітря — завершеного пристрою, здатного вимірювати температуру, вологість, атмосферний тиск та індекс якості повітря (IAQ), і передавати оброблені дані на смартфон через Bluetooth Low Energy. Цей кейс є особливо показовим, оскільки демонструє синергію трьох базових інтерфейсів: I²C для підключення сенсора, апаратну обробку сигналів на ядрі ARM Cortex-M4 з використанням спеціалізованої бібліотеки злиття даних, та BLE 5.3 для бездротової комунікації з мобільним пристроєм. У ролі центрального обчислювального ядра виступає мікроконтролер сімейства STM32WB55, який інтегрує на одному кристалі високопродуктивне ядро Cortex-M4 (для прикладної обробки) та енергоефективне ядро Cortex-M0+ (для стеку BLE), що дозволяє реалізувати весь функціонал в єдиній мікросхемі без зовнішніх радіомодулів.

Сенсор Bosch BME688, обраний для цього кейсу, є представником нового покоління цифрових датчиків навколишнього середовища, які поєднують вимірювання температури, відносної вологості, барометричного тиску та наявності летких органічних сполук (VOC) і газів у єдиному корпусі. Унікальною особливістю BME688 є наявність металооксидного (MOX) газового сенсора з програмованим профілем нагрівання, що дозволяє сканувати широкий спектр газів-забруднювачів. Фізичне підключення датчика до мікроконтролера виконується через інтерфейс I²C (режим Fast-mode, до 400 кГц), що потребує лише двох сигнальних ліній SDA та SCL, а також ліній живлення VDD (1,7–3,6 В) та VDDIO (1,2–3,6 В). Апаратна адреса BME688 на шині I²C є конфігурованою: при низькому рівні на виводі SDO вона становить 0x76, при високому — 0x77, що дозволяє підключати два однакових датчики до однієї шини без конфліктів. Схему підключення BME688 до мікроконтролера STM32WB55 із зазначенням необхідних зовнішніх компонентів (підтягувальних резисторів

Rp = 2,2 кОм для SDA/SCL, блокувальних конденсаторів 100 нФ по живленню) наведено на рисунку 12.1.

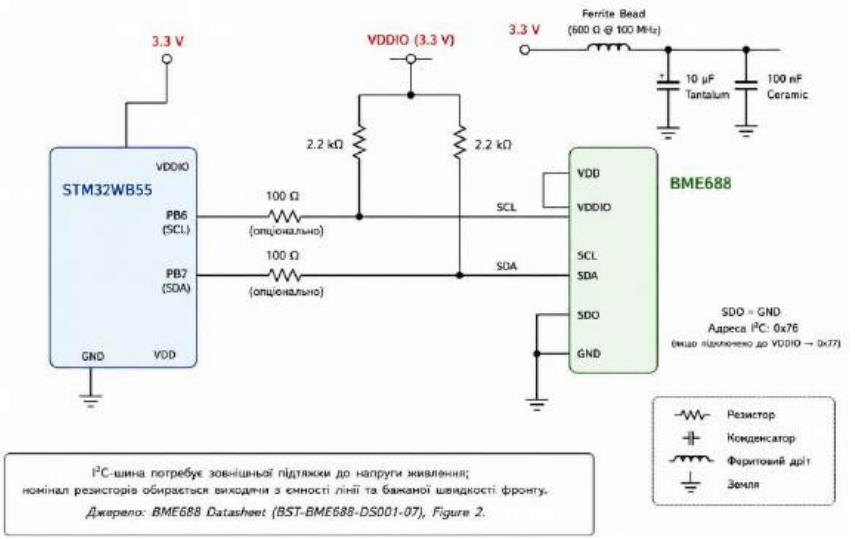


Рисунок 12.1 – Схема підключення BME688 до STM32WB55 через I²C

Основні технічні характеристики датчика BME688, що визначають його придатність для портативних IoT-застосувань, систематизовано в таблиці 12.1. Дані взято з офіційного даташиту виробника (Bosch Sensortec, BME688 Datasheet, Document BST-BME688-DS001-07, Revision 1.0, 2022) та прикладної нотатки з інтеграції BSEC (Bosch Software Environmental Cluster).

Програмне забезпечення мікроконтролера побудоване на базі операційної системи реального часу FreeRTOS, що дозволяє розділити задачі збору даних, їх обробки та BLE-комунікації на незалежні потоки. Задача збору даних (SensorTask) із періодом 1 секунда виконує читання «сирих» значень температури, тиску, вологості та опору газового сенсора через I²C за допомогою функції HAL_I2C_Mem_Read(). Отримані значення передаються в чергу до задачі обробки (BSECTask), яка використовує бібліотеку BSEC (Bosch Software Environmental Cluster) для злиття даних. BSEC реалізує складні алгоритми компенсації дрейфу газового сенсора, врахування впливу температури та вологості на опір MOX-шару, та обчислює похідні показники: індекс якості повітря IAQ (0–500), еквівалентний рівень CO_2 (CO_2eq, ppm) та концентрацію летких органічних сполук (TVOC, ppb).

Таблиця 12.1 – Основні характеристики Bosch BME688

Параметр	Значення	Примітка
Інтерфейс	I²C (до 3,4 МГц),SPI (3- та 4-провідний, до 10 МГц)	
Адреса I²C	0x76 (SDO=0) / 0x77 (SDO=1)	
Напруга живлення	VDD: 1,71…3,6 В; VDDIO: 1,2…3,6 В	
Струм у режимі вимірювання (T+P+H+Gas)	~3,9 мА (стандартний газ-скан)	0,9 мА у режимі LP; 90 мкА у режимі ULP
Струм у режимі сну	0,15 мкА (типово)	
Діапазон вимірювання температури / точність	-40…+85 °C / ±0,5 °C	
Діапазон вимірювання вологості / точність	0…100 % rH / ±3 % rH	
Діапазон вимірювання тиску / точність	300…1100 гПа / ±0,6 гПа	
Газовий сенсор	MOX, програмований нагрівач	виявляє VOC, VSC, CO, водень у діапазоні ppb
Частота вихідних даних (ODR)	до 182 Гц (T+P), до 50 Гц (газ)	Стандартний газ-скан: 10,8 с

Результати обробки передаються в чергу до задачі BLE (BLETask), яка оновлює значення характеристик у GATT-сервері, роблячи їх доступними для зчитування смартфоном. У лістингу 12.1 наведено фрагмент ініціалізації I²C-інтерфейсу, скидання BME688 та запуску періодичного читання даних.

Лістинг 12.1 – Ініціалізація I²C, скидання BME688 та періодичне читання даних на STM32WB55 (HAL + FreeRTOS)

```
// Платформа: STM32WB55 (HAL + FreeRTOS)
// Джерела: BME688 Datasheet (Bosch), STM32CubeWB
#include "main.h"
#include "cmsis os.h"
// ------ Дескриптори та константи -----
extern I2C_HandleTypeDef hi2c2; //I²C2, 400кГц (CubeMX)
#define BME688_ADDR   0x76  // Адреса (SDO=GND)
#define BME688_REG_SOFT_RESET 0xE0  // Регістр м'якого скидання
#define BME688 CMD RESET    0xB6 // Команда скидання
#define BME688 REG CTRL MEAS 0x74 // Рег. кер. вимірюванням
#define BME688 CTRL FORCED  0x01 // Примусовий режим
#define BME688_REG_STATUS  0x1D // Регістр статусу
#define BME688_STATUS_ND (1 << 5) // Біт new_data (готов. даних)
#define BME688 DATA START  0x22 // Поч. адреса блоку (11 Б)
// ------ Функція м'якого скидання -----
```

```
16:void BME688_SoftReset(void) {
17: // Запис команди 0xB6 у регістр 0xE0
18: HAL_I2C_Mem_Write(&hi2c2, BME688_ADDR, BME688_REG_SOFT_RESET,
19:  I2C_MEMADD_SIZE_8BIT, (uint8_t[]){BME688_CMD_RESET}, 1, 100);
20:  HAL_Delay(10);
21:}
22:// ------ Читання в примусовому режимі ------
23:uint8_t BME688_ReadForcedData(float *temp, float *pres, float
24:*hum) {
25:    uint8_t data[11];
26:   // 1. Встановл. примусового режиму (зап. 0x01 у рег. 0x74)
27: HAL_I2C_Mem_Write(&hi2c2, BME688_ADDR, BME688_REG_CTRL_MEAS,
28:  I2C_MEMADD_SIZE_8BIT,(uint8_t[]){BME688_CTRL_FORCED},1,100);
29:  // 2. Очікув. готовності (перевірка біта new data у рег. 0x1D)
30:    uint8_t status;
31:    do {
32:     HAL_Delay(10);
33:     HAL_I2C_Mem_Read(&hi2c2, BME688_ADDR, BME688_REG_STATUS,
34:                      I2C_MEMADD_SIZE_8BIT, &status, 1, 100);
35:    } while (!(status & BME688_STATUS_ND));
36:    // 3. Читання блоку з 11 байт (адреси 0x22…0x2C)
37:    HAL_I2C_Mem_Read(&hi2c2, BME688_ADDR, BME688_DATA_START,
38:                     I2C_MEMADD_SIZE_8BIT, data, 11, 100);
39:    // 4. Конвертація (спрощена, залежить від калібр. даних)
40:    // Реальна компенсація: зчитування калібр. коефіцієнтів
41:    *temp = (int32_t)(data[0] << 12 | data[1] << 4 | data[2] >>
42:4) / 100.0f;
43:    *pres = (int32_t)(data[3] << 12 | data[4] << 4 | data[5] >>
44:4) / 100.0f;
45:    *hum  = (int32_t)(data[6] << 8  | data[7]) / 100.0f;
46:    return 1;
47:}
48:// ------ Задача FreeRTOS ------
49:void SensorTask(void *argument) {
50: float temp, pres, hum;
51: BME688_SoftReset(); // Скидання при старті
52: for (;;) {
53:   if (BME688_ReadForcedData(&temp, &pres, &hum)) {
54:      // Надсилання даних до черги (BSECTask)
55:      xQueueSend(dataQueue, &(SensorData){temp, pres, hum}, 0);
56:        }
57:        vTaskDelay(pdMS_TO_TICKS(1000)); // Опитування кожні 1с
58:    }
 59:}
```

BLE-підсистема пристрою реалізує стандартний GATT-сервер із кількома характеристиками, кожна з яких відповідає за окремий показник. Пристрій рекламується під іменем «IAQ Sensor» і містить сервіс із UUID 0x180A (Device Information) та кастомний сервіс із 128-бітним UUID для даних навколишнього середовища. Характеристики температури, вологості, тиску, IAQ, CO_2eq та TVOC мають властивості Read та Notify, що дозволяє смартфону як періодично зчитувати поточні значення, так і отримувати сповіщення при їх зміні. Запис у CCCD (Client Characteristic

Configuration Descriptor) із боку смартфону активує нотифікації, після чого мікроконтролер автоматично сповіщає про нові дані після кожного циклу обробки BSEC. Конфігурація GATT-сервера на STM32WB55 виконується за допомогою програмного стеку STM32_WPAN (Wireless Personal Area Network), який надає високорівневий API для створення сервісів і характеристик. У лістингу 12.2 наведено фрагмент коду, що демонструє створення кастомного сервісу, додавання характеристики IAQ та оновлення її значення після отримання даних від BSEC.

Лістинг 12.2 – Створення GATT-сервісу та оновлення характеристики IAQ на STM32WB55 (STM32_WPAN)

```
// Платформа: STM32WB55 (STM32_WPAN)
// Джерела: AN5270 (STM32WB BLE), STM32CubeWB
#include "app_ble.h"
#include "ble_const.h"
#define IAQ_CHAR_SIZE 4 // int32_t для IAQ (4 байти)
// ------ Створення сервісу ------
/** * @brief Додавання користувацького GATT-сервісу
з 128-бітним UUID
 * @note  aci_gatt_add_service() створює сервіс
 * Max_Attribute_Records визначає макс. кількість атрибутів */
uint16_t svc_handle;
aci_gatt_add_service(UUID_TYPE_128, (Service_UUID_t *)
&custom_uuid,
                     SERVICE_TYPE_PRIMARY, 5, &svc_handle);
// ------ Додавання характеристики ------
/** * @brief Додавання хар-ки IAQ з правами READ | NOTIFY
* @note  aci_gatt_add_char() додає характеристику
* CHAR_PROP_NOTIFY дозволяє надсилати сповіщення
* UUID_TYPE_128 – 128-біт UUID для кастомного сервісу */
uint16_t char_handle;
aci_gatt_add_char(svc_handle, UUID_TYPE_128, (Service_UUID_t *)
&iaq_uuid,
        IAQ_CHAR_SIZE, CHAR_PROP_READ | CHAR_PROP_NOTIFY,
        ATTR_PERMISSION_NONE, GATT_NOTIFY_ATTRIBUTE_WRITE,
                  16, 1, &char_handle);
// ------ Оновлення значення ------
/** * @brief Задача BLE: отримання з черги та оновлення хар-ки
 * @note  aci_gatt_update_char_value() оновлює значення
 *   Якщо CCCD встановлено, сповіщення надсилається автоматично
 *  Енергоефективні нотифікації (без опитування)  */
void BLETask(void *argument) {
    SensorData data;
    for (;;) {
        if (xQueueReceive(dataQueue, &data, portMAX_DELAY)) {
            int32_t iaq_value = compute_iaq(data.temp,
data.hum, data.pres);
        aci_gatt_update_char_value(svc_handle, char_handle, 0,
                        IAQ_CHAR_SIZE, (uint8_t *)&iaq_value);
        }    } }
```

Часовий профіль енергоспоживання такого пристрою визначається робочим циклом: мікроконтролер прокидається раз на 1–10 секунд (залежно від налаштувань), виконує вимірювання BME688 (тривалість ~100 мс у forced mode), запускає ітерацію BSEC (близько 10 мс), оновлює BLE-характеристики та повертається в режим глибокого сну (Stop 2, струм ~2 мкА). При інтервалі вимірювання 10 секунд середній струм споживання не перевищує 50 мкА, що забезпечує багатомісячну автономну роботу від літій-полімерного акумулятора ємністю 500 мА·год. Оброблені дані можуть візуалізуватися на смартфоні за допомогою стандартних BLE-сканерів (nRF Connect, LightBlue) або спеціалізованого додатку, розробленого з використанням крос-платформних фреймворків (Flutter, React Native).

Кейс «Інтелектуальний датчик» наочно демонструє, як поєднання I²C для підключення сенсора, алгоритмічної обробки на Cortex-M4 та BLE для бездротової передачі дозволяє створити завершений, комерційно придатний пристрій для моніторингу якості повітря. Наступний підрозділ, присвячений шлюзу LoRaWAN–Ethernet, розширить цю парадигму на випадок розподіленого збору даних із багатьох вузлів через глобальну LPWAN-мережу.

12.2. “Шлюз LoRaWAN–Ethernet”: вузол збору сільгосп даних із публікацією у MQTT

Точне землеробство, моніторинг мікроклімату теплиць, контроль вологості ґрунту на відкритих полях та відстеження стану великої рогатої худоби — усі ці сільськогосподарські застосування поєднує спільна вимога: необхідність збирати дані з десятків або сотень територіально розподілених сенсорів, розташованих на значній відстані від найближчої точки доступу до Інтернету. Технологія LoRaWAN, розглянута в розділі 7, є ідеальним кандидатом для вирішення цієї задачі завдяки великій дальності (до 15 км у сільській місцевості) та надзвичайно низькому енергоспоживанню кінцевих пристроїв, що дозволяє живити датчики від батареї роками. Проте LoRaWAN-мережа сама по собі не має прямого виходу в Інтернет: радіопакети від кінцевих пристроїв приймаються шлюзами (концентраторами), які повинні передати ці дані на мережевий сервер LoRaWAN, а звідти — на сервер додатків. У сільськогосподарському сценарії, де шлюз часто встановлюється на фермі, елеваторі або метеостанції, найдоступнішим каналом зв’язку з глобальною мережею є провідний Ethernet (або Wi-Fi, під’єднаний до локальної мережі). Цей кейс демонструє побудову такого шлюзу, здатного приймати LoRaWAN-пакети від польових датчиків, декодувати їх,

трансформувати у формат JSON та публікувати через протокол MQTT до локального або хмарного брокера, реалізуючи туманний рівень (fog) типової IoT-архітектури.

Апаратною основою шлюзу слугує одноплатний комп'ютер Raspberry Pi 4 (SoC Broadcom BCM2711, 4×Cortex-A72, 4 ГБ ОЗП), на якому виконується операційна система Linux (Raspberry Pi OS Lite, Debian 12). Такий вибір зумовлений наявністю гігабітного Ethernet-порту, достатньою обчислювальною потужністю для запуску програмного забезпечення шлюзу та MQTT-клієнта, а також широкою підтримкою апаратних концентраторів LoRa. До Raspberry Pi через інтерфейс SPI підключається багатоканальний LoRa-концентратор на базі чіпсета Semtech SX1302 (наприклад, модуль RAK5146 або WM1302), здатний одночасно приймати до 8 каналів LoRa з різними SF (від SF5 до SF12) та один канал FSK. Модуль концентратора також потребує сигналів GPIO для скидання (RESET) та індикації зайнятості (BUSY). Для забезпечення надійного годинника, необхідного для формування точних часових міток у пакетах, використовується зовнішній температурно-компенсований кварцовий генератор (TCXO) на 32 МГц, що підключається до концентратора. Живлення всього шлюзу здійснюється від стандартного адаптера 5 В / 3 А через USB-C роз'єм Raspberry Pi. Апаратну архітектуру шлюзу ілюструє рисунок 12.2.

Програмне забезпечення шлюзу складається з двох основних компонентів: LoRaWAN Packet Forwarder та прикладного сервісу публікації в MQTT. Класичним і найпоширенішим підходом є використання Semtech UDP Packet Forwarder (або його сучасної альтернативи — ChirpStack Gateway Bridge / Basics Station), який отримує пакети від SX1302 через SPI-драйвер, інкапсулює їх у JSON-повідомлення (згідно з протоколом Semtech UDP або Protobuf) і передає на мережевий сервер LoRaWAN через IP-мережу. Однак у даному кейсі ми свідомо спрощуємо архітектуру, відмовляючись від виділеного мережевого сервера LoRaWAN, щоб продемонструвати прямий шлях даних «від сенсора до MQTT». Для цього використовується відкритий проект ChirpStack Gateway OS (або його компонент ChirpStack Concentratord), який запускається у контейнері Docker на Raspberry Pi та реалізує повний стек LoRaWAN-шлюзу, включаючи взаємодію з SX1302. Concentratord отримує радіопакети, демодулює їх і передає дані (як необроблені байти PHYPayload) через API (gRPC) або безпосередньо до локального мережевого сервера ChirpStack, який можна розгорнути на тому ж Raspberry Pi.

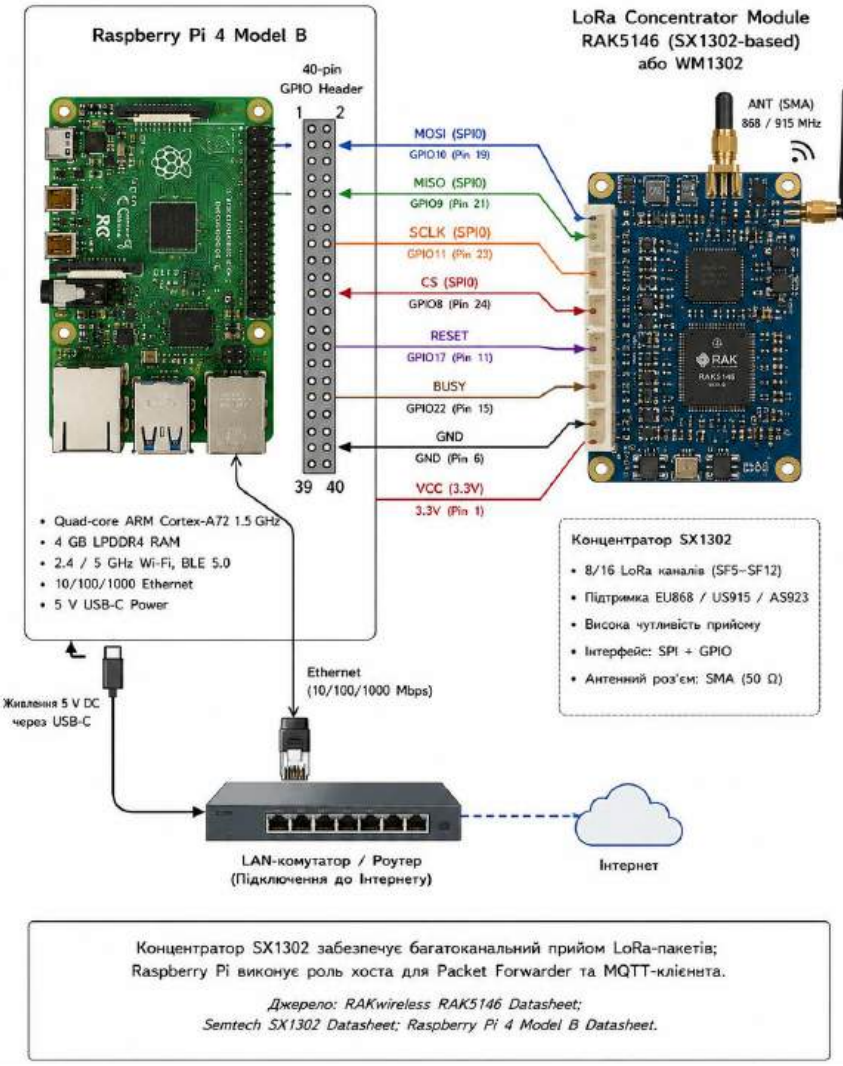


Рисунок 12.2 – Апаратна архітектура LoRaWAN-шлюзу на базі Raspberry Pi 4 з концентратором SX1302

Для сценарію з прямим MQTT застосовується альтернативний підхід: мінімальний Python-сервіс (шлюзовий демон), який взаємодіє безпосередньо з Packet Forwarder через локальний UDP-порт (або через MQTT-брідж, якщо використовується ChirpStack MQTT Forwarder). Демон прослуховує UDP-пакети від Packet Forwarder, які містять JSON-об'єкти з полями rxpk (прийняті LoRa-пакети). Для кожного прийнятого пакета демон

витягує байти data (корисне навантаження, закодоване в base64), декодує їх згідно з форматом даних сенсора, формує JSON-об'єкт із зрозумілими полями (наприклад, *{"device":"soil_sensor_05","soil_moisture":34.5, "temperature":22.1,"timestamp":"2026-06-15T10:30:00Z"}*) і публікує цей об'єкт на локальний або хмарний MQTT-брокер. У ролі кінцевих пристроїв виступають LoRaWAN-датчики вологості ґрунту, побудовані на базі мікроконтролерів STM32WL (наприклад, STM32WL55JC) з підключеним ємнісним датчиком вологості через АЦП. Кожен датчик має унікальний DevEUI, зареєстрований у системі.

У лістингу 12.3 наведено спрощений код Python-демону, який ілюструє основний цикл приймання даних та публікації в MQTT. Код використовує бібліотеки socket для прослуховування UDP та paho-mqtt для MQTT-клієнта. У реальній системі слід додати буферизацію на випадок втрати зв'язку з брокером, обробку помилок декодування base64 та автентифікацію.

Лістинг 12.3 – Python-демон шлюзу для прийому UDP-пакетів від Packet Forwarder та публікації в MQTT (спрощений)

```
# Джерела: Semtech UDP Packet Forwarder Protocol, Things Network
import socket
import json
import base64
import paho.mqtt.client as mqtt
UDP_PORT = 1700
MQTT_TOPIC = "farm/sensor/data"
# ------ Налаштування MQTT --------
mqtt_client = mqtt.Client()
mqtt_client.connect("localhost", 1883)
# ------ UDP-сервер ---------
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("0.0.0.0", UDP_PORT))
while True:
  data, addr = sock.recvfrom(2048)
  packet = json.loads(data.decode())
  # Обробка кожного отриманого радіопакета (rxpk)
  for rxpk in packet.get("rxpk", []):
    raw_bytes = base64.b64decode(rxpk["data"]) # Декод. B64
    # Приклад: [тип 1Б][вологість 2Б][темп. 2Б]
    sensor_type = raw_bytes[0]
    humidity = int.from_bytes(raw_bytes[1:3], "big") / 100.0
    temperature = int.from_bytes(raw_bytes[3:5], "big") / 100.0
        # Публікація в MQTT
        payload = json.dumps({
            "type": sensor_type,
            "humidity": humidity,
            "temperature": temperature        })
        mqtt_client.publish(MQTT_TOPIC, payload)
```

Більш продуктивне рішення для промислового застосування використовує мікроконтролер STM32H723 із вбудованим Ethernet MAC (10/100 Мбіт/с) та стеком LwIP, що дозволяє створити компактний, безвентиляторний шлюз без операційної системи Linux. У цьому випадку концентратор SX1302 підключається безпосередньо до STM32 через SPI, а весь стек — від драйвера концентратора до MQTT-клієнта (наприклад, coreMQTT) — виконується на bare-metal або FreeRTOS. Це вимагає значно більших зусиль із розроблення, однак забезпечує передбачувану латентність, менше енергоспоживання (кілька ват) і вищу надійність, характерну для вбудованих систем. У таблиці 12.2 наведено порівняння двох підходів до реалізації LoRaWAN-шлюзу.

Таблиця 12.2 – Порівняння платформ для LoRaWAN-шлюзу з публікацією у MQTT

Характеристика	Raspberry Pi 4 + SX1302 (Linux)	STM32H723 + SX1302 (bare-metal/FreeRTOS)
Операційна система	Linux (Raspberry Pi OS)	bare-metal / FreeRTOS
Стек LoRaWAN	ChirpStack Concentratord / Packet Forwarder	Semtech SX1302 HAL + власний MAC-рівень
MQTT-клієнт	Eclipse Paho (Python) або Mosquitto (C)	coreMQTT + LwIP
Енергоспоживання (типове)	5–10 Вт (типовий струм 600 мА при 5 В)	1–3 Вт (типовий струм до 430 мА при 3,3 В)
Складність розроблення	низька (готові образи ОС, скрипти)	висока (потрібна глибока інтеграція драйверів)
Надійність (стійкість до збоїв)	середня (ризик пошкодження SD-карти)	висока (монолітна прошивка, відсутність ОС)
Вартість компонентів	~$60 (RPi 4 + SX1302 модуль)	~$30 (STM32H723 + SX1302 модуль)
Типовий сценарій	прототипування, фермерські господарства з доступом до електромережі	промислові шлюзи, живлення від PoE, цілодобова робота

Для завершення наскрізної картини на рисунку 12.3 зображено повний шлях даних від польового датчика до хмарної візуалізації. Датчик вологості ґрунту (STM32WL) передає LoRa-пакет, який приймається концентратором SX1302, обробляється шлюзом (Raspberry Pi або STM32), трансформується в JSON і через MQTT потрапляє до брокера Mosquitto. Звідти дані можуть бути спожиті локальною SCADA-системою (наприклад, Node-RED) або через MQTT-міст передані до хмарного брокера (AWS IoT Core, Azure IoT Hub) для довготривалого зберігання та аналітики.

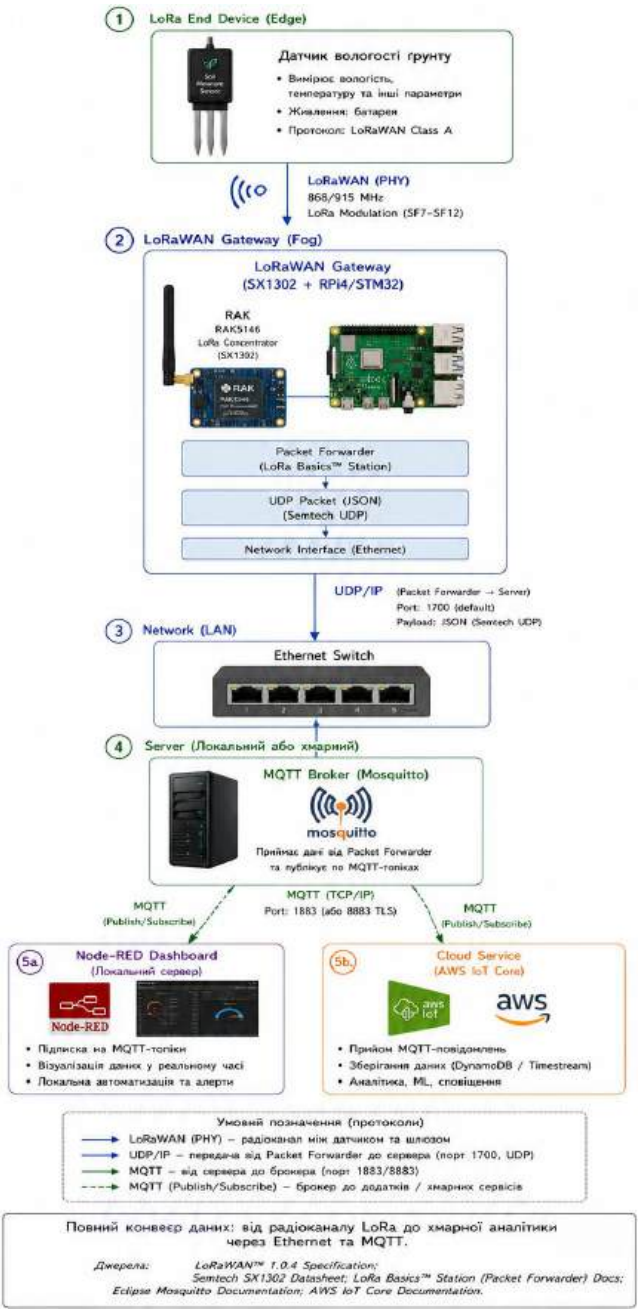


Рисунок 12.3 – Наскрізний шлях даних у системі “Шлюз LoRaWAN–Ethernet”

Кейс шлюзу LoRaWAN–Ethernet демонструє практичне поєднання технології LPWAN великого радіусу дії, стандартного мережевого інтерфейсу Ethernet та відкритого протоколу MQTT для створення масштабованих систем агромоніторингу. Використання Raspberry Pi з концентратором SX1302 забезпечує швидке прототипування, тоді як перехід на спеціалізований мікроконтролер (STM32H7) дозволяє створити енергоефективний промисловий шлюз. Наступний кейс, присвячений роботизованому модулю з EtherCAT, перенесе фокус на детерміновані дротові мережі для керування рухом у реальному часі.

12.3. “Роботизований модуль з EtherCAT”: підпорядкований контролер на STM32H7 з інтерфейсом CAN FD для зворотного зв’язку двигунів

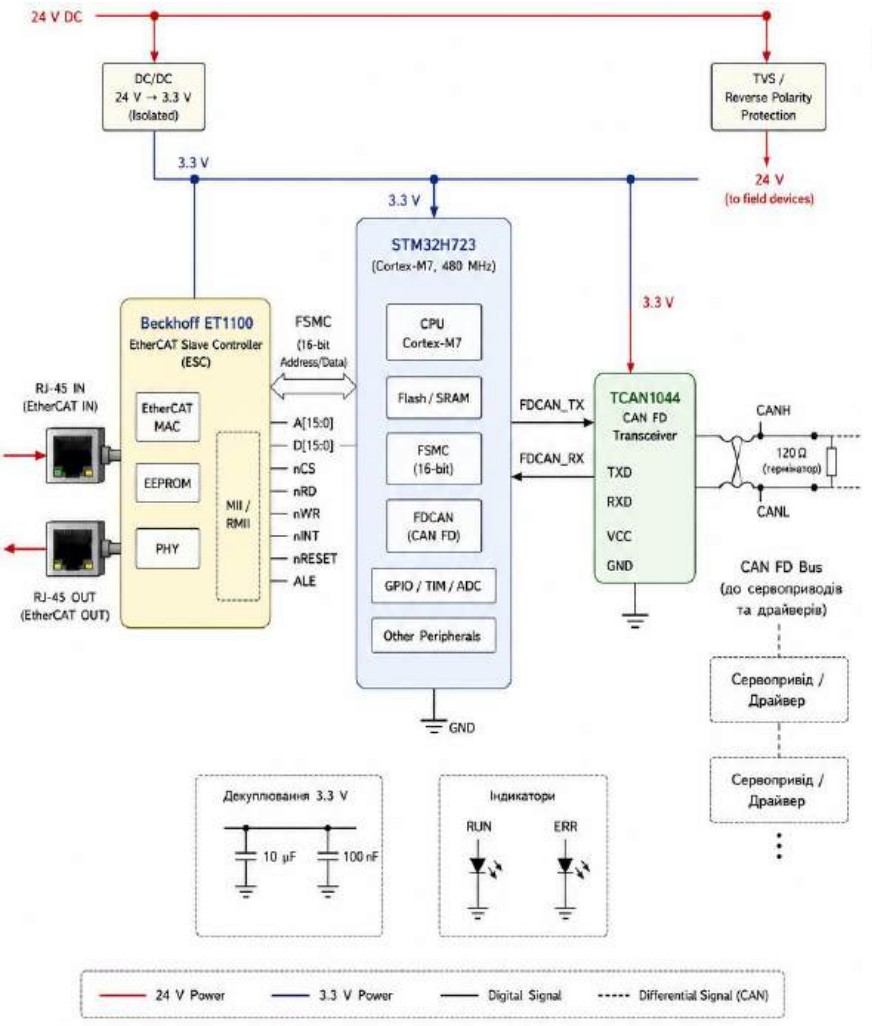
Сучасні багатоосьові роботи, верстати з числовим програмним керуванням та автоматизовані виробничі лінії висувають гранично жорсткі вимоги до синхронізації та детермінованості обміну даними між центральним контролером (майстром) та окремими модулями керування двигунами (підпорядкованими пристроями). Затримка передавання команди позиціонування або актуального значення крутного моменту навіть у кілька десятків мікросекунд може призвести до вібрацій, браку продукції або аварійного спрацьовування захисту. EtherCAT (Ethernet for Control Automation Technology), стандартизований у серії IEC 61158 Type 12, є одним із найпоширеніших протоколів промислового Ethernet реального часу, що розв’язує цю проблему завдяки оригінальному механізму обробки кадрів «на льоту» (processing on the fly). У цьому кейсі розглядається архітектура підпорядкованого контролера роботизованого модуля, побудованого на базі високопродуктивного мікроконтролера STM32H723 (ядро ARM Cortex-M7, до 550 МГц) із зовнішнім EtherCAT-контролером Beckhoff ET1100 та вбудованим модулем FDCAN для обміну даними з інтелектуальними сервоприводами.

Апаратна архітектура модуля об’єднує три функціональні домени. Першим є власне EtherCAT-підсистема, реалізована на спеціалізованій мікросхемі Beckhoff ET1100 (EtherCAT Slave Controller, ESC), яка підключається до фізичного рівня 100BASE-TX через стандартний імпульсний трансформатор із роз’ємом RJ-45. ET1100 має два порти (IN, OUT), що дозволяє будувати кільцеву або лінійну топологію. Другий порт може бути використаний для підключення наступного EtherCAT-пристрою. Дані процесу (Process Data Objects, PDO) та конфігураційні повідомлення (Service

Data Objects, SDO) зчитуються з ET1100 мікроконтролером STM32H723 через високошвидкісний паралельний інтерфейс – 16-бітну асинхронну шину, реалізовану за допомогою гнучкого контролера пам'яті FSMC (Flexible Static Memory Controller). Такий підхід забезпечує мінімальну латентність доступу до внутрішньої пам'яті DPRAM (Dual-Port RAM) ESC, де зберігаються PDO. Альтернативним, простішим варіантом для менш вимогливих застосувань є підключення через SPI, однак FSMC гарантує більшу пропускну здатність для оновлення даних на високих частотах (наприклад, 8 кГц для контурів струму).

Другим доменом є інтерфейс CAN FD, який використовується для зв'язку з драйверами двигунів. STM32H723 містить модуль FDCAN, сумісний з ISO 11898-1:2015, що підтримує швидкість фази даних до 8 Мбіт/с і довжину поля даних до 64 байтів. Через CAN FD підпорядкований контролер циклічно передає цільові значення струму або швидкості на драйвери, а також отримує від них фактичні параметри (положення ротора, швидкість, температуру, код помилки). Вибір CAN FD зумовлений його детермінованим арбітражем, що гарантує доставку критичних повідомлень у найгіршому випадку за лічені мікросекунди, та високою завадостійкістю диференційного фізичного рівня, сумісного з промисловим середовищем. Трансивер, наприклад, TCAN1044A-Q1 від Texas Instruments, перетворює логічні рівні FDCAN у диференційні сигнали CAN_H та CAN_L з підтримкою режиму Flexible Data Rate. Третій домен – це допоміжні інтерфейси (SPI, I²C, GPIO) для підключення датчиків кінцевого положення, кнопок безпеки, світлодіодних індикаторів тощо, які не є предметом цього кейсу. Загальну структурну схему модуля показано на рисунку 12.4.

З точки зору програмного забезпечення, підпорядкований контролер працює під управлінням операційної системи реального часу FreeRTOS, що дозволяє розмежувати критичні до часу обробники та фонові задачі діагностики. Стек EtherCAT базується на Beckhoff Slave Stack Code (SSC) – поширеному інструменті для генерації коду підпорядкованого пристрою. SSC генерує базовий каркас, який включає обробку станів кінцевого автомата EtherCAT (Init, Pre-Op, Safe-Op, Op), підтримку CoE (CANopen over EtherCAT) та функції для роботи з PDO через SyncManagers ET1100. Розробник доповнює цей каркас прикладною логікою: у функції, що викликається при отриманні фрейму Sync0 (періодичне переривання від ET1100, синхронізоване з майстром), виконується зчитування PDO від майстра (цільове положення, швидкість), розрахунок контурів керування та формування PDO для відповіді (фактичне положення, струм, статус). Розраховані значення передаються на драйвери через CAN FD.



EtherCAT-контролер ET1100 через порти RJ-45 підключається до кільця EtherCAT; мікроконтролер зчитує PDO через FSMC, обробляє алгоритм керування та видає команди на драйвери через CAN FD.

Джерело: Beckhoff ET1100 Datasheet (v2.5), RM0468 (ST).

- ET1100 підтримує EtherCAT (100BASE-TX) з автоматичним проходженням кадрів на апаратному рівні.
- FSMC у 16-бітному режимі забезпечує інтерфейс до регістрів ESC.
- FDCAN (вбудований у STM32H723) працює до 5 Mbit/s у CAN FD.

Рисунок 12.4 – Структурна схема підпорядкованого EtherCAT-контролера на базі STM32H723 з CAN FD

Оскільки EtherCAT вимагає жорсткої часової синхронізації, ESC ET1100 підтримує Distributed Clocks (DC), що дозволяє всім осям робота виконувати знімання даних із приводів одночасно з точністю до десятків наносекунд. На практиці це означає, що момент фіксації фактичного положення енкодерів усіх двигунів визначається сигналом SYNC0 від ET1100, а не моментом отримання CAN-кадру.

Налаштування FDCAN для такого застосування передбачає роботу в режимі CAN FD з номінальною швидкістю арбітражу 500 кбіт/с та швидкістю даних 5 Мбіт/с (типові значення для коротких ліній до 20 метрів). Пріоритети кадрів призначаються відповідно до вимог системи: найвищий пріоритет (найменший CAN ID) мають команди аварійної зупинки та критичні сповіщення про помилки, середній – цільові значення для контурів положення/швидкості, найнижчий – діагностичні дані (температура, напруга). Для підвищення надійності застосовується режим Transmit Delay Compensation, який дозволяє компенсувати затримки поширення сигналу на високих швидкостях. У таблиці 12.3 наведено ключові параметри EtherCAT та CAN FD у цьому кейсі.

Таблиця 12.3 – Параметри EtherCAT та CAN FD для роботизованого модуля

Параметр	EtherCAT (порт)	CAN FD (FDCAN)
Фізичний рівень	100BASE-TX (IEEE 802.3)	диференційна вита пара (ISO 11898-2)
Топологія	кільце/лінія/зірка/дерево (до 65 535 вузлів)	шина (до 30–60 вузлів залежно від трансиверів)
Швидкість передавання	100 Мбіт/с (повний дуплекс)	арбітраж: до 1 Мбіт/с (типово 500 кбіт/с), дані: до 5–8 Мбіт/с
Період циклу (типовий)	125 мкс (8 кГц)	125 мкс (8 кГц) (за синхронізації з EtherCAT Sync0)
Синхронізація	Distributed Clocks (DC), Sync0, похибка < 1 мкс	Sync0 від EtherCAT + апаратні таймери (опціонально)
Корисне навантаження за цикл	до 1486 байт на кадр (EtherCAT-датаграма)	до 64 байт на кадр
Призначення	зв’язок із майстром (NC/PLC), конфігурація	зв’язок із драйверами двигунів (команди струму, зворотний зв’язок)

У лістингу 12.4 наведено фрагмент коду, що демонструє структуру обробника переривання Sync0, у якому виконується читання PDO з ET1100,

розрахунок та відправлення команд через CAN FD, а також підготовка вхідних PDO для майстра.

Лістинг 12.4 – Обробник Sync0 для EtherCAT+CAN FD на STM32H723 (C, FreeRTOS)

```
// Відповідно до EtherCAT ET1100 Hardware Description,
// STM32H723 Reference Manual (RM0433) та STM32CubeH7 HAL Driver
#include "main.h"
#include "et1100.h" // Драйвер ET1100 (DPRAM через FSMC)
#include "fdcan.h"
#include "FreeRTOS.h"
#include "task.h"
#define SYNC0_PERIOD_US  125 // Період Sync0 (125 мкс для DC)
extern FDCAN_HandleTypeDef hfdcan1;
extern TaskHandle_t xDiagnosticTaskHandle;
volatile int32_t g_actual_position=0; // Оновлюється в IRQ FDCAN
volatile uint16_t g_status_word = 0;
// ------ Обробник Sync0 (ISR) ------
/** * @brief Обробник переривання Sync0 від ET1100
 * @note  Викликається з ISR EXTI за сигналом SYNC0
 * Період Sync0 налаштовується в ET1100 (125 мкс)
 * Вихідні PDO читаються з DPRAM ET1100 через FSMC
 * Вхідні PDO записуються в DPRAM ET1100 через FSMC
 * CAN FD кадр -> HAL FDCAN AddMessageToTxFifoQ() */
void Sync0_Handler(void) {
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;
  // --- 1. Читання вихідних PDO з DPRAM ET1100 (FSMC) ---
  // DPRAM ET1100 відображено в адресний простір через FSMC
  uint32_t target_pos = *(volatile uint32_t*)(ET1100_DPRAM_BASE
+ PDO_OUTPUT_OFFSET);
  uint32_t target_vel = *(volatile uint32_t*)(ET1100_DPRAM_BASE
+ PDO_OUTPUT_OFFSET + 4);
  // --- 2. Обчислення керуючого сигналу (струму) ---
  int32_t current_cmd = MotionControl_Update(target_pos,
target_vel, g_actual_position);
  // --- 3. Формування та відправлення CAN FD-кадру ---
  FDCAN_TxHeaderTypeDef TxHeader;
  uint8_t TxData[8];
  TxHeader.Identifier = 0x180; // ID для драйвера осі X
  TxHeader.IdType = FDCAN_STANDARD_ID;
  TxHeader.TxFrameType = FDCAN_DATA_FRAME;
  TxHeader.DataLength = FDCAN_DLC_BYTES_4;
  TxHeader.ErrorStateIndicator = FDCAN_ESI_ACTIVE;
  TxHeader.BitRateSwitch = FDCAN_BRS_ON; // CAN FD з підвищ.
швидкістю
  TxHeader.FDFormat = FDCAN_FD_CAN;
  TxHeader.TxEventFifoControl = FDCAN_NO_TX_EVENTS;
  TxHeader.MessageMarker = 0;
  *(int32_t*)TxData = current_cmd; // Команда струму
  HAL_FDCAN_AddMessageToTxFifoQ(&hfdcan1, &TxHeader, TxData);
  // - 4. Читання зворотного зв'язку (оновлено в IRQ FDCAN) -
  // g_actual_position/g_status_word оновлюються в
HAL_FDCAN_RxFifo0Callback()
  // --- 5. Запис вхідних PDO в DPRAM ET1100 (FSMC) ---
    *(volatile uint32_t*)(ET1100_DPRAM_BASE + PDO_INPUT_OFFSET)
```

```
= g_actual_position;
    *(volatile uint16_t*)(ET1100_DPRAM_BASE + PDO_INPUT_OFFSET
+ 4) = g_status_word;
    // --- 6. Пробудження задачі діагностики ---
    vTaskNotifyGiveFromISR(xDiagnosticTaskHandle,
&xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}
```

Поєднання EtherCAT як магістральної мережі реального часу та CAN FD як локальної шини зворотного зв'язку двигунів дозволяє створити гнучку, детерміновану й водночас відносно недорогу архітектуру розподіленого керування рухом. EtherCAT забезпечує синхронізацію багатьох осей із субмікросекундною точністю та високу пропускну здатність для команд від центрального контролера, тоді як CAN FD надає надійний і завадостійкий канал до окремих драйверів, що вже підтримують цей протокол (багато виробників сервоприводів, зокрема Kollmorgen, Beckhoff, Elmo, пропонують моделі з CANopen/FD). Використання STM32H723 із потужним ядром Cortex-M7 та розвиненою периферією дозволяє реалізувати всю логіку керування та комунікації в одній мікросхемі, мінімізуючи габарити та вартість модуля.

12.4. "Розумний лічильник": UART-інтерфейс з NB-IoT-модемом, передача показів із використанням CoAP на сервер енергокомпанії

Інтелектуальні лічильники електроенергії, газу та води утворюють один із наймасовіших сегментів Інтернету речей, що безпосередньо впливає на ефективність енергозбуту, точність білінгу та надійність енергосистем. На відміну від споживчих гаджетів, розумний лічильник працює в екстремально обмежених умовах: він часто встановлюється в підвалах, металевих шафах або на стовпах, де відсутнє стаціонарне живлення, а заміна батареї є дорогою логістичною операцією. Це диктує жорсткі вимоги до автономності (понад 10 років), надійності доставки даних (покази мають бути білінгозначущими) та безпеки (несанкціоноване втручання в роботу лічильника є фінансовим злочином). У цьому кейсі розглядається архітектура розумного лічильника, який використовує NB-IoT як транспортну технологію, UART як інтерфейс між хост-мікроконтролером і стільниковим модемом, та протокол CoAP (Constrained Application Protocol) для передавання показів на сервер енергокомпанії. Такий вибір зумовлений тим, що CoAP, на відміну від MQTT, працює через UDP без постійного TCP-з'єднання, використовує компактний бінарний формат, природно інтегрується

з REST-архітектурою корпоративних систем і підтримує механізм Observe (RFC 7641) для оперативного сповіщення про критичні події — наприклад, спробу розкриття корпусу або аварійне відключення.

Апаратна платформа лічильника складається з трьох ключових компонентів: мікроконтролера з ультранизьким споживанням (наприклад, STM32L476 на ядрі ARM Cortex-M4 з апаратним криптографічним прискорювачем AES), NB-IoT-модему (Quectel BC660K-GL, що підтримує Cat-NB2, PSM, eDRX, UDP та CoAP), та енерговимірювального фронтенду (наприклад, мікросхема ADE9000 від Analog Devices, яка через SPI надає відкалібровані значення активної та реактивної енергії, напруги та струму). З'єднання між мікроконтролером і модемом здійснюється через асинхронний UART зі швидкістю 115200 бод, лініями TX, RX, RTS, CTS та додатковими сигналами PWRKEY (для ввімкнення/вимкнення модему) і STATUS (для індикації мережевої активності). Живлення системи — літій-тіонілхлоридна батарея 3,6 В ємністю 3600 мА·год, здатна забезпечити роботу протягом 15 років за умови середнього струму не більше 27 мкА. Схему підключення основних компонентів наведено на рисунку 12.5.

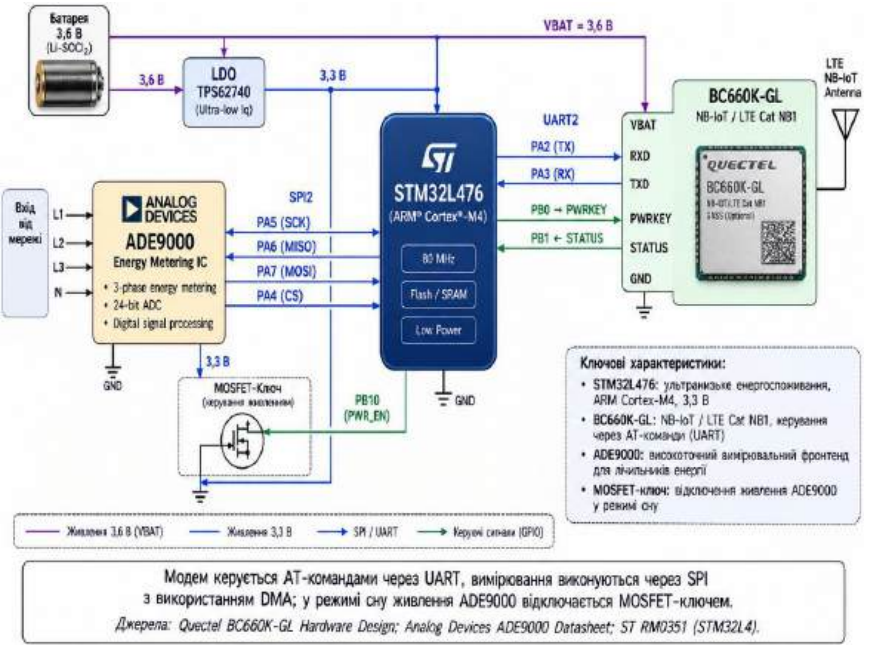


Рисунок 12.5 – Схема підключення розумного лічильника: STM32L476 + BC660K-GL + ADE9000

Ключові характеристики обраних компонентів систематизовано в таблиці 12.4, що дозволяє оцінити відповідність кожного елемента вимогам до автономності та функціональності.

Таблиця 12.4 – Основні характеристики компонентів розумного лічильника

Компонент	Параметр	Значення	Примітка
STM32L476	Струм у Stop 2 (RTC on)	1,4 мкА (типово)	пробудження за таймером
Quectel BC660K-GL	Струм у PSM	0,8 мкА (типово)	3GPP Rel-14, Cat-NB2
Quectel BC660K-GL	Піковий струм передавання	330 мА (23 дБм)	короткочасно
ADE9000	Струм у режимі вимірювання	3,5 мА	відключається між сеансами
LDO TPS62740	Власне споживання	360 нА (типово)	високий ККД при малих навантаженнях

Програмне забезпечення лічильника реалізує цикл із періодом, що задається тарифною політикою (зазвичай 1 раз на добу). Мікроконтролер прокидається за сигналом RTC-будильника, вмикає живлення вимірювального фронтенду через MOSFET-ключ, ініціалізує ADE9000 через SPI, зчитує накопичені значення енергії (активної та реактивної) за допомогою DMA, після чого вимикає фронтенд для економії енергії. Отримані дані формуються в компактний JSON-пакет (або CBOR для ще більшої економії трафіку), який містить серійний номер лічильника, мітку часу, покази та статусний код. Далі мікроконтролер вмикає NB-IoT-модем (через PWRKEY), очікує його готовності (AT-відповідь), активує PDP-контекст (AT+CGDCONT), створює UDP-сокет (AT+USOCR) і відправляє CoAP-повідомлення. У ролі CoAP-клієнта використовується полегшена бібліотека на мові C (наприклад, microcoap, що займає менше 10 КБ флеш-пам’яті), яка формує CON-запит PUT на відомий URI ресурсу лічильника на сервері енергокомпанії (наприклад, coap://meter.energy.com/readings). Після отримання ACK-відповіді модем негайно переводиться в режим PSM із таймерами T3324=1 хв, T3412=24 год, і весь пристрій повертається в глибокий сон. У лістингу 12.5 наведено спрощений код основного циклу лічильника, що ілюструє взаємодію компонентів.

Лістинг 12.5 – Основний цикл розумного лічильника: вимірювання, формування CoAP, передача через NB-IoT (псевдокод на C)

```
// Джерела: ADE9000 Technical Reference Manual, 3GPP TS 27.007,
//          Quectel BC660K-GL AT Commands Manual
#include "main.h"
#include "spi.h"
#include "usart.h"
#include "stm3214xx hal pwr ex.h"
#define ADE9000 CS PIN   GPIO PIN 4
#define ADE9000_CS_PORT  GPIOA
// ---- 1. Пробудження зі Stop 2 ----
/** * @brief Вихід зі Stop 2 (наприклад, за перериванням RTC)
 * @note  HAL_PWR_EnterSTOPMode() - вхід у Stop 2
 * Пробудження - за зовнішнім перериванням або RTC */
void SystemWakeUp(void) {
    // Відновлення тактування після Stop 2
    SystemClock_Config();
    __HAL_RCC_PWR_CLK_ENABLE();
}
// ---- 2. Читання енергії з ADE9000 (SPI) ----
/** * @brief Читання регістрів енергії через SPI
 * @note  ADE9000 - SPI slave
 * Читання регістрів xWATTHR (активна енергія) */
void ReadEnergy(float *active_energy, float *reactive_energy) {
  uint8_t tx_buffer[4], rx_buffer[4];
  uint32_t reg_value;
  // Вибір ADE9000 (CS = 0)
  HAL_GPIO_WritePin(ADE9000_CS_PORT, ADE9000_CS_PIN,
GPIO_PIN_RESET);
  // Читання рег. активної енергії (наприклад, 0x0000)
  tx_buffer[0] = 0x00; tx_buffer[1] = 0x00; tx_buffer[2] = 0x00;
tx_buffer[3] = 0x00;
  HAL_SPI_TransmitReceive(&hspi1, tx_buffer, rx_buffer, 4,
100);
  // Вимкнення ADE9000 (CS = 1)
  HAL_GPIO_WritePin(ADE9000_CS_PORT, ADE9000_CS_PIN,
GPIO_PIN_SET);
  // Конвертація (залежить від калібрувальних коефіцієнтів)
  reg_value = (rx_buffer[0] << 24) | (rx_buffer[1] << 16) |
              (rx_buffer[2] << 8)  | rx_buffer[3];
  *active_energy = (float)reg_value / 3600000.0f; // Ex. конв.
}
// ----- 3. Формування JSON ------
/** * @brief Мінімальна функція формування JSON-рядка
 * @note  Використовується snprintf() для економії пам'яті */
void json_build(char *buffer, size_t size, float energy, float
power) {
    snprintf(buffer, size, "{\"e\":%.2f,\"p\":%.2f}", energy,
power);
}
// ----- 4. Налаштування NB-IoT модему -----
/** * @brief Увімкн. модему, встановл. APN, створення UDP-сокету
 * @note  AT+CFUN=1 - повна функціональність
 *  AT+CGDCONT - встановлення APN
 *  AT+USOCR=17 - створення UDP-сокету (17 = UDP)
 *  Після кожної команди - перевірка відповіді OK */
```

```
55:void ModemInit(void) {
56:    // Увімкнення модему
57:    AT_Send("AT+CFUN=1\r\n", "OK", 5000);
58:    // Встановлення APN
59:    AT_Send("AT+CGDCONT=1,\"IP\",\"iot.operator.com\"\r\n",
60:"OK", 5000);
61:    // Створення UDP-сокету
62:    AT_Send("AT+USOCR=17\r\n", "+USOCR:", 5000);  // Очікуємо
63:+USOCR: <socket id>
64:}
65:// ---- 5. Відправлення CoAP через UDP ----
66:/** * @brief Формування CoAP-пакета та відправлення через UDP
67: * @note  coap_build_pdu() - формування CoAP-дейтаграми
68: *  AT+USOWR - відправлення даних через UDP-сокет
69: *  AT+USORD - читання відповіді (ACK) */
70:void SendCoap(float energy, float power) {
71:    uint8_t coap_buffer[128];
72:    uint16_t coap_len;
73:    // Формування CoAP-пакета (Confirmable, GET або POST)
74:    coap_len = coap_build_pdu(coap_buffer, energy, power);
75:    // Відправлення через UDP-сокет (ID сокета = 0)
76:    char cmd[64];
77:    sprintf(cmd, "AT+USOWR=0,%d,\"", coap_len);
78:    AT_Send_Data(cmd, coap_buffer, coap_len, "OK", 5000);
79:    // Очікування відповіді (ACK)
80:    AT_Send("AT+USORD=0,128\r\n", "+USORD:", 5000);
81:}
82:// ---- 6. Вхід у PSM (Power Saving Mode) ----
83:/** * @brief Налаштування PSM та перехід у сон
84: * @note  AT+CPSMS - встановлення таймерів T3324 та T3412
85: *  AT+CFUN=0 - мінімальний функціональний режим
86: *  HAL_PWR_EnterSTOPMode() - вхід у Stop 2 */
87:void EnterSleep(void) {
88:    // Встановлення PSM (T3324 = 1 хв, T3412 = 1 година)
89:    AT_Send("AT+CPSMS=1,,,\"00000001\",\"00101000\"\r\n",
90:"OK", 5000);
91:    // Вимкнення модему
92:    AT_Send("AT+CFUN=0\r\n", "OK", 5000);
93:    // Вхід у Stop 2
94:    HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,
95:PWR_STOPENTRY_WFI);
96:}
97:// ---- 7. Головний цикл ----
98:void main(void) {
99:    float energy, power;
100:    char json_buffer[64];
101:    while (1) {
102:        // 1. Пробудження
103:        SystemWakeUp();
104:        // 2. Читання даних з ADE9000
105:        ReadEnergy(&energy, &power);
106:        // 3. Формування JSON
107:        json_build(json_buffer, sizeof(json_buffer), energy,
108:power);
109:        // 4. Ініціалізація модему
110:        ModemInit();
111:        // 5. Відправлення CoAP
```

```
112:         SendCoap(energy, power);
113:         // 6. Перехід у сон
114:         EnterSleep();
115:     }
 116:}
```

Сервер енергокомпанії, що приймає дані, розгортається з використанням CoAP-бібліотеки Californium (Java) або CoAP.NET (C#) і реалізує ресурс /readings, який приймає PUT-запити з JSON-тілом, валідує серійний номер, перевіряє цифровий підпис (HMAC-SHA256, ключ згенеровано на етапі персоналізації лічильника), записує покази в базу даних та відповідає кодом 2.04 Changed. Для забезпечення безпеки на транспортному рівні використовується DTLS 1.3 (RFC 9147) із попередньо встановленими сертифікатами (PSK), що не потребує важкого рукостискання, сумісного з обмеженими ресурсами модему. Альтернативний варіант — використання MQTT — є поширенішим, але в даному кейсі CoAP має перевагу завдяки меншому обсягу службового трафіку (відсутність TCP-рукостискання, Keep-Alive пакетів, CONNECT-процедури), що безпосередньо впливає на тривалість активної фази модему та, відповідно, на термін служби батареї. Порівняльний аналіз двох підходів наведено в таблиці 12.5.

Таблиця 12.5 – Порівняння CoAP/UDP та MQTT/TCP для розумного лічильника з NB-IoT

Параметр	CoAP/UDP/DTLS	MQTT/TCP/TLS
Транспорт	UDP (без з'єднання)	TCP (з встановленням з'єднання)
Середній обсяг службових даних за сеанс (1 uplink)	~150–200 байт (UDP + DTLS + CoAP)	~400–600 байт (TCP + TLS + MQTT)
Тривалість активної фази модему	~2–3 с (CoAP CON + ACK)	~5–8 с (TCP + TLS + MQTT)
Підтримка Асинхронних команд (downlink)	через Observe (CON повідомлення з боку сервера після пробудження)	нативна (publish від брокера)
Витрати енергії на сеанс	~200–300 мДж	~500–800 мДж
Сумісність із REST-системами білінгу	природна (URI, методи REST)	потребує MQTT-REST мосту

Кейс розумного лічильника демонструє раціональний вибір технологій для масового IoT-пристрою з жорсткими обмеженнями: NB-IoT забезпечує глобальне покриття та глибоке проникнення, UART є найпростішим і найекономнішим інтерфейсом для AT-модему, а CoAP мінімізує накладні витрати на передавання білінгових даних. Саме така комбінація, підкріплена режимами енергозбереження PSM та DTLS-захистом, дозволяє створити лічильник, здатний працювати понад десятиліття без обслуговування, що є еталоном ефективності в індустрії IoT. Цим завершується розділ практичних кейсів; наступна частина посібника перейде до розгляду еволюції інтерфейсів та новітніх тенденцій, які визначатимуть розвиток вбудованих комунікацій у найближчі роки.

ЧАСТИНА VI.
ЕВОЛЮЦІЯ ІНТЕРФЕЙСІВ ТА НОВІТНІ ТЕНДЕНЦІЇ

РОЗДІЛ 13.
ІСТОРИЧНА РЕТРОСПЕКТИВА ТА ЗАКОНОМІРНОСТІ РОЗВИТКУ

13.1. Від паралельних шин до послідовних: ISA → PCIe, PATA → SATA, паралельний SCSI → SAS

Еволюція інтерфейсів вбудованих систем є не хаотичною зміною технологій, а відображенням глибоких фізичних та економічних закономірностей. Одним із найфундаментальніших переходів в історії комп'ютерної інженерії стала відмова від паралельних шин на користь високошвидкісних послідовних з'єднань. Цей перехід, що розпочався на межі 1990-х та 2000-х років і остаточно завершився до середини 2010-х, був зумовлений не волею окремих інженерів, а непереборними фізичними обмеженнями, насамперед явищем перекосу тактового сигналу (clock skew) між паралельними лініями. У паралельній шині дані передаються одночасно по кількох (8, 16, 32) провідниках, синхронізованих єдиним тактовим сигналом. Для коректного приймання всі біти мають досягти приймача одночасно, в межах одного тактового інтервалу. З підвищенням тактової частоти допустима різниця в часі прибуття сигналів (skew) зменшується обернено пропорційно. Будь-яка неоднорідність довжини доріжок, ємності навантаження або властивостей діелектрика друкованої плати спричиняє наростання skew, що унеможливлює подальше масштабування частоти. Як зазначено в дослідженні Д. Міллера та співавторів (Proceedings of the IEEE, том 92, 2004), саме clock skew, а не абсолютна пропускна здатність окремого провідника, став фундаментальним бар'єром для паралельних шин на частотах понад 100 МГц.

Ця фізична межа спричинила тектонічний зсув у трьох ключових доменах: системних шинах, інтерфейсах накопичувачів та периферійних магістралях. У домені системних шин еволюція пройшла шлях від 16-бітної паралельної шини ISA (Industry Standard Architecture, 8,33 МГц, 16 МБ/с), через 32-бітну PCI (Peripheral Component Interconnect, 33/66 МГц, 133/266 МБ/с) та її 64-бітне розширення PCI-X, до послідовної PCI Express (PCIe). Вже PCI-X 2.0 на частоті 533 МГц стикнувся з критичними проблемами цілісності сигналу, що й спонукало консорціум PCI-SIG у 2004 році прийняти специфікацію PCIe 1.0. PCIe замінив широку паралельну шину на одну або кілька (x1, x4, x8, x16) послідовних диференційних ліній

(lanes), кожна з яких працює незалежно. Уже перше покоління PCIe забезпечило 2,5 ГТ/с на лінію (кодування 8b/10b, ефективно 250 МБ/с на лінію), що перевищило пропускну здатність 64-бітного PCI-X. Подальші покоління — PCIe 2.0 (5 ГТ/с), 3.0 (8 ГТ/с), 4.0 (16 ГТ/с), 5.0 (32 ГТ/с), 6.0 (64 ГТ/с із модуляцією PAM-4) — демонструють експоненційне зростання, яке було б неможливим у паралельній архітектурі. Специфікації PCIe Base Specification (PCI-SIG, ревізії 1.0–6.0) є нормативною основою цієї еволюції.

Аналогічна трансформація відбулася в інтерфейсах накопичувачів. Паралельний ATA (PATA/IDE), що використовував 40- або 80-жильний шлейф із 16 бітами даних, досяг своєї межі на швидкості UDMA/133 (133 МБ/с). Подальше підвищення швидкості вимагало збільшення кількості провідників та частоти, що призводило до неприйнятних рівнів електромагнітних завад та skew. У 2003 році консорціум Serial ATA International Organization (SATA-IO) випустив специфікацію Serial ATA 1.0, яка замінила 80-жильний шлейф на 7-контактний кабель із двома диференційними парами (одна для передавання, одна для прийому). Швидкість SATA 1.0 становила 1,5 Гбіт/с, що вже перевищувало PATA. Наступні покоління — SATA 2.0 (3 Гбіт/с), SATA 3.0 (6 Гбіт/с) — зберегли зворотну сумісність на фізичному рівні. SATA 3.2 (2013) інтегрував PCIe-лінії (SATA Express), що остаточно розмило межу між SATA та PCIe, а з виходом NVMe (Non-Volatile Memory Express, NVM Express Inc., 2011) PCIe став основним інтерфейсом для твердотільних накопичувачів, залишивши SATA нішу бюджетних SSD та HDD.

У домені високопродуктивних периферійних інтерфейсів аналогічний шлях пройшла шина SCSI (Small Computer System Interface). Паралельний SCSI (Parallel SCSI, SPI), стандартизований комітетом T10 (INCITS), у своїй останній ітерації Ultra-640 SCSI досяг 640 МБ/с на 16-бітній шині, використовуючи складні схеми термінування та синхронізації. Проте обмеження за довжиною кабелю (до 12 м для LVD SCSI) та кількістю пристроїв (15 на шину) зумовили перехід до послідовного Serial Attached SCSI (SAS). Специфікація SAS (T10, INCITS 376) визначила послідовний диференційний інтерфейс, сумісний на фізичному рівні з SATA (що дозволило підключати SATA-диски до SAS-контролерів), але з розширеними можливостями (dual-port, full-duplex, до 128 пристроїв на домен). Перше покоління SAS забезпечило 3 Гбіт/с, а сучасне SAS-4 (22,5 Гбіт/с) продовжує лінійку.

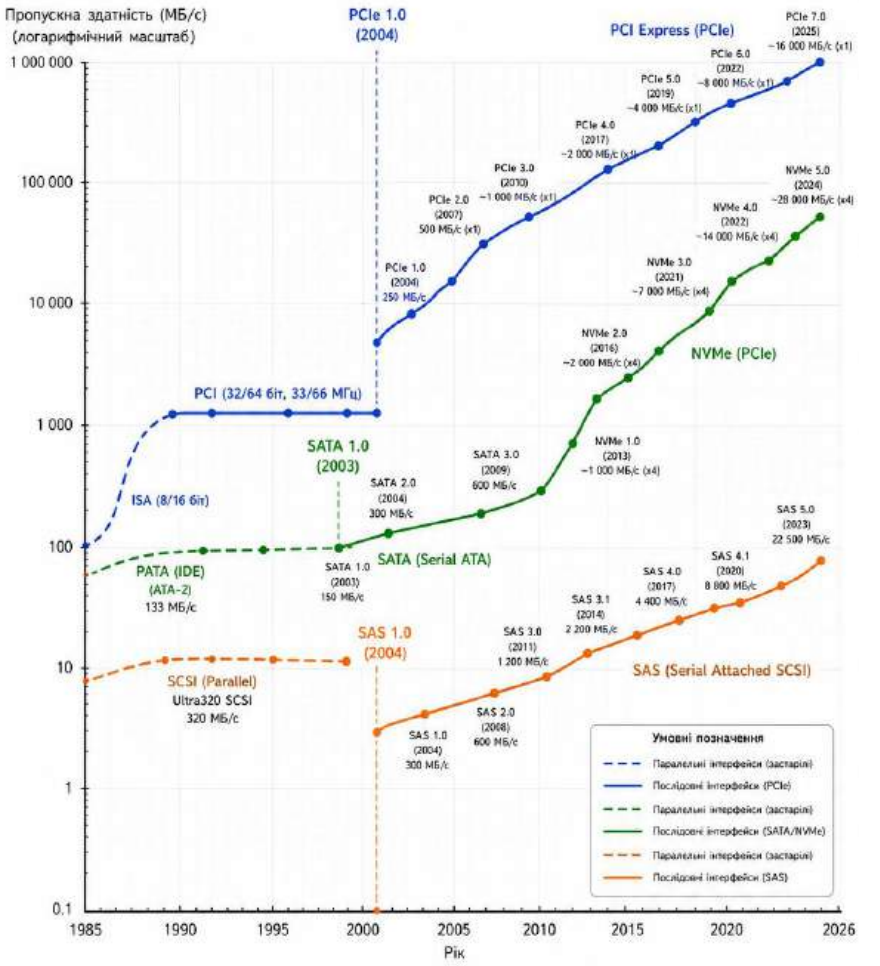
Порівняння ключових представників паралельних та послідовних шин у трьох доменах наведено в таблиці 13.1, яка ілюструє, як радикально змінилися швидкості завдяки переходу до послідовної архітектури.

Таблиця 13.1 – Еволюція від паралельних до послідовних шин у трьох доменах

Покоління	Системні шини (швидкість)	Інтерфейси накопичувачів (швидкість)	Периферійні шини (швидкість)
Паралельні (останнє покоління)	PCI-X 2.0 — 4,2 ГБ/с	PATA UDMA/133 — 133 МБ/с; Ultra-640 SCSI — 640 МБ/с	—
Перехідні	PCIe 1.0 x16 — 4 ГБ/с	SATA 1.0 — 150 МБ/с; SAS-1 — 300 МБ/с	—
Послідовні (1-ше покоління)	PCIe 1.0 x16 — 4 ГБ/с	SATA 3.0 — 600 МБ/с; SAS-2 — 600 МБ/с	—
Послідовні (актуальні, 2026)	PCIe 6.0 x16 — 128 ГБ/с	NVMe (PCIe 5.0 x4) — 16 ГБ/с; SAS-4 — 2,4 ГБ/с на лінію	—

Графічну ілюстрацію цього експоненційного зростання наведено на рисунку 13.1, де показано криві швидкості для PCI/PCIe, PATA/SATA та SCSI/SAS у логарифмічному масштабі. На графіку чітко видно момент переходу від насичення паралельних шин до стрімкого зростання продуктивності послідовних.

Перехід від паралельних шин до послідовних є не просто історичним фактом, а проявом фундаментального інженерного принципу: коли фізичне явище (clock skew) стає непереборною перешкодою для масштабування, архітектурна зміна (диференційна послідовна передача з вбудованим тактуванням) дозволяє обійти це обмеження. Ця закономірність, що простежується в усіх розглянутих доменах, є ключем до розуміння того, чому сучасні високошвидкісні інтерфейси — PCIe, USB, Ethernet, SATA, SAS — усі без винятку є послідовними. Наступний підрозділ, присвячений конвергенції польових шин, продемонструє, як аналогічний процес відбувається на рівні промислових мереж, де на зміну дискретним Modbus-лініям приходить IP-орієнтований Ethernet TSN.



Відмова від паралельних шин дозволила подолати обмеження clock skew і продовжити експоненційне зростання пропускної здатності.

Дані зібрані з офіційних специфікацій PCI-SIG, SATA-IO, T10/INCITS.

Джерела даних

- *PCI-SIG® PCI Express Base Specification Revision 7.0, Version 1.0, 2025.*
- *SATA-IO Serial ATA Revision 3.5, 2018; NVM Express™ Specification 2.0/3.0/4.0/5.0.*
- *T10/INCITS SAS-5 Rev 1.0, 2023.*
- Попередні специфікації: *PCI Local Bus Specification Rev. 2.3 (1994); ATA/AT Attachment with Packet Interface – 6 (ATA-6), 2011; SCSI Ultra320 – ANSI INCITS 359-2003.*

Рисунок 13.1 – Графік зростання швидкості інтерфейсів при переході від паралельних до послідовних шин

13.2. Конвергенція польових шин: від дискретного Modbus до IP-орієнтованого Ethernet TSN

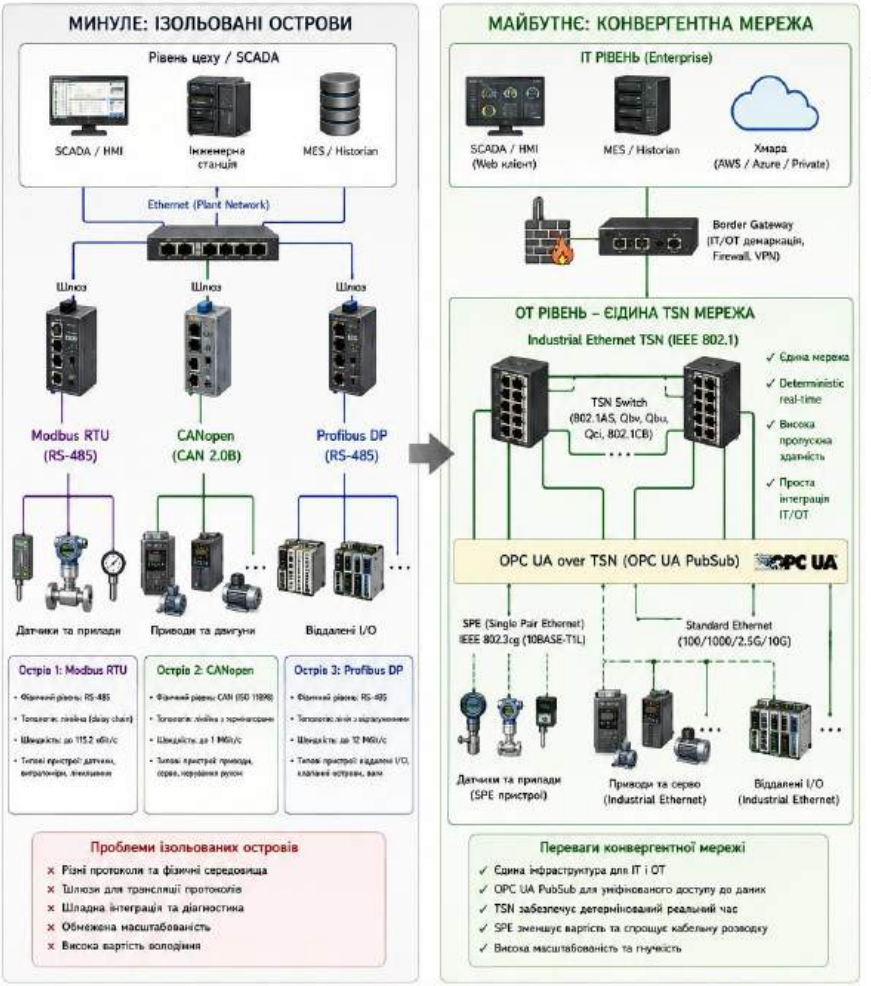
Якщо перехід від паралельних шин до послідовних, розглянутий у попередньому підрозділі, був зумовлений фізичними обмеженнями clock skew, то конвергенція польових шин на єдиній платформі Industrial Ethernet має глибше, системне коріння. Протягом 1980-х – 1990-х років промислова автоматизація сформувала строкатий ландшафт із десятків несумісних між собою протоколів, кожен із яких оптимізувався під конкретну задачу та виробника обладнання. Modbus RTU (Modicon, 1979) став де-факто стандартом для невибагливих задач збору даних завдяки своїй відкритості та простоті реалізації поверх RS-485. CAN (Bosch, 1986) завоював автомобільну електроніку та промисловий привід завдяки неразрушуючому арбітражу та детермінованості, еволюціонувавши в CANopen (CiA 301). Profibus DP (Siemens, 1993) забезпечив високошвидкісний циклічний обмін для ПЛК та розподіленої периферії, а Interbus, DeviceNet, CC-Link та інші заповнили регіональні та корпоративні ніші. Спільною рисою всіх цих рішень була їхня фізична та протокольна ізоляція: кожна мережа використовувала власний кабель, власний трансивер і, що найважливіше, вимагала спеціалізованого шлюзу для інтеграції з інформаційними мережами рівня підприємства (Ethernet TCP/IP). Така архітектура, відома як «автоматизовані острови» (islands of automation), вступала в суперечність із концепцією Industry 4.0, яка вимагає наскрізної, прозорої комунікації від датчика до хмари. Як зазначено в Industrial Internet Reference Architecture (IIRA v1.9, 2022), ключовою вимогою до сучасних промислових мереж є конвергенція інформаційних (IT) та операційних (OT) технологій на єдиній мережевій інфраструктурі, що уможливлює безперешкодний обмін даними без трансляції протоколів.

Першим кроком до конвергенції стала поява на початку 2000-х років протоколів промислового Ethernet, які використовували стандартний фізичний рівень IEEE 802.3, але реалізовували власні механізми реального часу поверх нього. Profinet (стандартизований PI як IEC 61158 Type 10, 2003) запропонував два режими: Profinet RT (Real-Time) для м’якого реального часу з пріоритезацією трафіку через VLAN (IEEE 802.1Q) та Profinet IRT (Isochronous Real-Time) для жорсткого реального часу із синхронізацією на основі ASIC-комутаторів. EtherCAT (Beckhoff, 2003, IEC 61158 Type 12) обрав радикальніший підхід, реалізувавши обробку кадрів «на льоту» (processing on the fly), де один кадр проходить через усі підпорядковані при-

строї послідовно, що забезпечує мінімальну латентність та детермінованість. EtherNet/IP (ODVA, 2001, IEC 61158 Type 2) використовував стандартний стек TCP/IP та UDP, додаючи протокол CIP (Common Industrial Protocol) на прикладному рівні, що полегшило інтеграцію з IT-системами, але обмежило детермінованість. SERCOS III, Powerlink та інші запропонували власні, часто несумісні між собою механізми синхронізації. Хоча ці протоколи використовували спільний фізичний кабель (Ethernet), вони залишалися протокольно несумісними, вимагаючи окремих контролерів та інструментів конфігурації для кожної мережі.

Справжня конвергенція стала можливою з появою та стандартизацією Time-Sensitive Networking (TSN) — набору розширень IEEE 802.1, що додають до стандартного Ethernet механізми гарантованої латентності, синхронізації часу та резервування (розглянуті в підрозділі 5.2.3). TSN забезпечує єдиний, відкритий, незалежний від виробника рівень реального часу, який може використовуватися будь-яким прикладним протоколом, усуваючи потребу в пропрієтарних апаратних рішеннях (IRT-комутаторах, ASIC EtherCAT). Найважливішим прикладним протоколом, що працює поверх TSN, є OPC UA (IEC 62541), зокрема його PubSub-розширення. У цій архітектурі OPC UA забезпечує семантично насичену, безпечну модель даних, а TSN гарантує своєчасну доставку цих даних із мікросекундною точністю. Таким чином, відбувається уніфікація: один мережевий кабель Ethernet може одночасно нести трафік реального часу (наприклад, команди керування рухом через OPC UA PubSub поверх TSN), фоновий IT-трафік (TCP/IP, HTTP, MQTT) та мультимедійні дані (відеоспостереження). Процес конвергенції наочно ілюструє рисунок 13.2.

Останнім елементом, що завершує конвергенцію на найнижчому рівні, є Single Pair Ethernet (SPE). Стандарт 10BASE-T1S (IEEE 802.3cg-2019) забезпечує передавання даних на швидкості 10 Мбіт/с через одну неекрановану виту пару на відстань до 25 м у багатоточковій топології з підтримкою живлення (PoDL), що є прямою заміною RS-485. Це дозволяє підключати найпростіші польові пристрої (датчики, клапани) безпосередньо до мережі Ethernet без проміжних перетворювачів інтерфейсів. Порівняльний аналіз традиційних польових шин та сучасних конвергентних рішень наведено в таблиці 13.2.



Ethernet TSN + OPC UA + SPE уможливлюють єдину конвергентну мережу без шлюзів трансляції протоколів

Ключові стандарти

- IEEE 802.1AS – Timing and Synchronization for Time-Sensitive Applications
- IEEE 802.1Qbv – Time-Aware Shaper
- IEEE 802.1Qbu / 802.1Qci – Frame Preemption
- IEEE 802.1CB – Frame Replication and Elimination for Reliability
- IEEE 802.3cg – 10BASE-T1L (Single Pair Ethernet)
- OPC UA Part 14 – PubSub Communication
- Modbus over Serial Line Specification & Implementation Guide V1.02
- CAN in Automation (CiA) DS-301 – CANopen Application Layer
- Profibus DP-V1 Specification
- IEC 61158 / IEC 61784 – Industrial communication networks

Умовні позначення

- Ethernet (TCP/IP)
- Industrial Ethernet TSN (OPC UA)
- Modbus RTU (RS-485)
- CANopen (CAN)
- Profibus DP (RS-485)
- SPE (Single Pair Ethernet)

Джерело: специфікації та білі документи від організацій PCI-SIG®, SATA-IO®, T10/INCITS®, IEEE®, OPC Foundation®, ODVA®, Profibus & Profinet International (PI)®, CiA®.

Рисунок 13.2 – Конвергенція польових шин до Ethernet TSN з OPC UA

Таблиця 13.2 – Порівняння традиційних польових шин та конвергентних Ethernet-рішень

Шини	Характеристика	Значення
Modbus RTU (RS-485)	*Фізичний рівень*	RS-485 (диференційна вита пара)
	Максимальна швидкість	до 10 Мбіт/с
	Топологія	шина (multidrop)
	Детермінованість	ні (запит-відповідь)
	IP-сумісність	ні (потребує Modbus-TCP шлюзу)
	Сумісність між виробниками	висока (відкритий протокол)
	Потреба в шлюзі до IT-мережі	так (Modbus TCP Gateway)
	Підтримка живлення по лінії	ні
CANopen (CAN 2.0)	*Фізичний рівень*	CAN (ISO 11898-2), диференційна вита пара
	Максимальна швидкість	1 Мбіт/с
	Топологія	шина (лінійна)
	Детермінованість	так (арбітраж за пріоритетом)
	IP-сумісність	ні (потребує CAN-Ethernet шлюзу)
	Сумісність між виробниками	висока (CiA 301)
	Потреба в шлюзі до IT-мережі	так (CANopen Gateway)
	Підтримка живлення по лінії	ні
Profinet IRT	*Фізичний рівень*	100BASE-TX (IEEE 802.3)
	Максимальна швидкість	100 Мбіт/с
	Топологія	зірка, кільце, лінія, дерево
	Детермінованість	так (IRT, джиттер < 1 мкс)
	IP-сумісність	так (Profinet)
	Сумісність між виробниками	висока (PI)
	Потреба в шлюзі до IT-мережі	ні (Profinet сумісний з Ethernet)
	Підтримка живлення по лінії	так (PoE)
EtherCAT	*Фізичний рівень*	100BASE-TX (IEEE 802.3)
	Максимальна швидкість	100 Мбіт/с
	Топологія	кільце/лінія (open ring)
	Детермінованість	так (processing on the fly)
	IP-сумісність	так (EtherCAT)
	Сумісність між виробниками	висока (ETG)
	Потреба в шлюзі до IT-мережі	ні (EtherCAT сумісний з Ethernet)
	Підтримка живлення по лінії	опціонально

Кінець таблиці 13.2

Шини	Характеристика	Значення
Ethernet TSN + OPC UA PubSub	*Фізичний рівень*	1000BASE-T, 10BASE-T1S (IEEE 802.3cg)
	Максимальна швидкість	до 1 Гбіт/с (1000BASE-T); 10 Мбіт/с (10BASE-T1S)
	Топологія	зірка, кільце, шина (SPE)
	Детермінованість	так (IEEE 802.1Qbv)
	IP-сумісність	так (IPv6, TCP/UDP)
	Сумісність між виробниками	дуже висока (IEC 62541, IEEE 802.1)
	Потреба в шлюзі до IT-мережі	не потрібен (нативний IP)
	Підтримка живлення по лінії	так (PoDL для SPE, PoE для 1000BASE-T)

Конвергенція польових шин є не просто заміною одного кабелю іншим, а фундаментальною зміною парадигми: від вертикально інтегрованих, пропрієтарних «островів автоматизації» до горизонтально інтегрованої, відкритої, IP-орієнтованої мережі підприємства. Ця еволюція зменшує капітальні витрати (один тип кабелю, менше шлюзів), спрощує обслуговування (єдина система діагностики), підвищує гнучкість (легке додавання нових пристроїв) та відкриває шлях до наскрізної аналітики великих даних, що є серцем Industry 4.0.

13.3. Еволюція бездротових протоколів: від пропрієтарних до стандартів 5G/6G та NTN

Якщо конвергенція дротових польових шин, розглянута в попередньому підрозділі, є прикладом уніфікації «зверху вниз» — від рівня підприємства до окремого датчика, — то еволюція бездротових протоколів демонструє зустрічний рух «знизу вгору»: від десятків несумісних пропрієтарних рішень, створених для вузьких завдань, до глобальних стандартів стільникового зв’язку, які об’єднали голос, дані та масовий IoT у єдиній мережевій архітектурі. Цей процес, що тривав понад чотири десятиліття, був зумовлений фундаментальними економічними чинниками: ефектом масштабу, який робить стандартизовані компоненти на порядки дешевшими за пропрієтарні; вимогами регуляторів щодо ефективного використання радіочастотного спектра; та потребою в глобальній інтероперабельності, без

якої неможливий ані міжнародний роумінг, ані єдина екосистема IoT. Простежуючи траєкторію від ранніх пропрієтарних радіомодемів до сучасних 5G-мереж із підтримкою неназемного сегмента (NTN) та перспектив 6G, можна виділити три наскрізні тенденції: перехід від апаратно-фіксованих до програмно-визначених (SDR) радіосистем; конвергенцію ліцензованого та неліцензованого спектра; та стирання кордонів між наземними та супутниковими мережами.

Перший етап еволюції (1980–2000 рр.) характеризувався домінуванням пропрієтарних бездротових рішень, жорстко прив'язаних до конкретного виробника, діапазону частот та прикладного завдання. Промислові радіомодеми для телеметрії SCADA, системи безключового доступу до автомобілів, перші бездротові датчики — усі вони використовували закриті протоколи, часто реалізовані на дискретних аналогових компонентах, що унеможливлювало взаємодію з обладнанням інших виробників. Кожен проект супроводжувався «прив'язкою до постачальника» (vendor lock-in), а модернізація системи означала повну заміну радіоустаткування. Однак саме в цей період були закладені фізичні основи сучасних технологій: з'явилися перші стандарти фізичного рівня (IEEE 802.11 у 1997 році), а також була доведена працездатність методів розширення спектра (DSSS, FHSS), що згодом лягли в основу Bluetooth, Wi-Fi та LoRa.

Другий етап (2000–2020 рр.) ознаменувався лавиноподібним поширенням відкритих стандартів, передусім IEEE 802.15.1 (Bluetooth), IEEE 802.15.4 (Zigbee, Thread) та IEEE 802.11 (Wi-Fi). Ключовим каталізатором стала поява програмованих радіосистем (Software-Defined Radio, SDR), де модуляція, кодування та частотний план визначаються програмним забезпеченням, а не жорсткою логікою. Це дозволило створювати мультипротокольні SoC (наприклад, TI CC2650, nRF52840), здатні працювати в кількох стандартах на одному кристалі. У цей самий період стільниковий зв'язок пройшов шлях від 2G (GSM/GPRS) до 4G (LTE), де в Release 13 (2016) були вперше стандартизовані спеціалізовані LPWAN-технології стільникового класу — NB-IoT та LTE-M. Це стало переломним моментом: традиційно «комп'ютерні» та «стільникові» бездротові технології почали конвергенцію, спрямовану на обслуговування єдиного ринку IoT.

Третій етап (2020–2026 рр. і далі) визначається двома магістральними напрямами: еволюцією 5G NR (New Radio) та інтеграцією неназемних мереж (NTN) згідно з 3GPP Release 17/18/19. 5G NR, специфікований 3GPP починаючи з Release 15 (2018), із самого початку проектувався як єдина платформа для трьох різнорідних сценаріїв: надширокосмугового до-

ступу eMBB, надійних комунікацій із низькою латентністю URLLC та масового машинного зв'язку mMTC. Такий підхід усуває потребу в паралельних мережах для різних класів пристроїв. Release 18 (2024) остаточно інтегрував NB-IoT та LTE-M у ядро 5G (5GC), дозволивши операторам обслуговувати весь спектр IoT-пристроїв — від датчиків диму до автономних автомобілів — у межах єдиної мережевої архітектури з нарізкою мережі (network slicing). Одночасно, у Release 17 (2022) було вперше стандартизовано NTN — пряме підключення стандартних стільникових пристроїв до супутників на низькій (LEO) та геостаціонарній (GEO) орбітах. Це стирає останній географічний бар'єр: пристрій, що працює в мережі 5G NTN, може бути розгорнутий у будь-якій точці планети, включно з океанами та полярними регіонами, без зміни апаратного забезпечення. Паралельно триває активна фаза досліджень 6G (очікувана стандартизація близько 2030 року), яка, згідно з баченням ITU-T IMT-2030, передбачає перехід до субтерагерцових частот (100–300 ГГц), інтеграцію штучного інтелекту в радіоінтерфейс, голографічні комунікації та «Інтернет відчуттів». У таблиці 13.3 систематизовано ключові віхи еволюції бездротових протоколів.

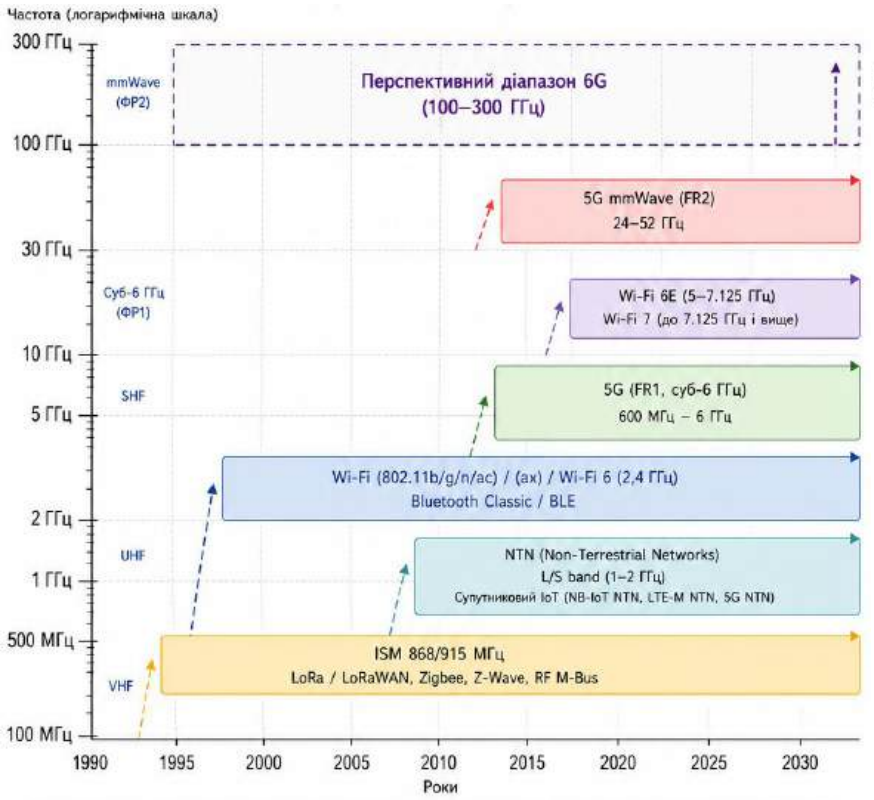
Таблиця 13.3 – Еволюція бездротових протоколів: від пропрієтарних до 6G/NTN

Етапи	Характеристика	Значення
Пропрієтарна ера	*Роки*	1980–2000
	Представники	пропрієтарні радіомодеми, WirelessHART (перші версії HART), перші системи безключового доступу
	Характерні риси	апаратна фіксація, закриті протоколи, vendor lock-in; HART розроблено компанією Rosemount у середині 80-х, у 1990 році створено HART User Group, у 1993 році права передано HART Communication Foundation (HCF)
	Архітектурна парадигма	ізольовані рішення
Фрагментація стандартів	*Роки*	2000–2020
	Представники	Wi-Fi 4/5/6, Bluetooth 2.0–5.0 (включно з BLE), Zigbee, Thread, LoRaWAN
	Характерні риси	поява відкритих стандартів, SDR, мультипротокольні SoC; WirelessHART представлено у вересні 2007 року як перший відкритий промисловий бездротовий стандарт, схвалено IEC у 2009 році; горизонтальна фрагментація (вибір протоколу під задачу)

Кінець таблиці 13.3

Етапи	Характеристика	Значення
Фрагментація стандартів	*Архітектурна парадигма*	горизонтальна фрагментація
Конвергенція 5G + IoT	*Роки*	2020–2026
	Представники	5G NR (Rel-15–18), NB-IoT/LTE-M у 5GC, 5G RedCap
	Характерні риси	єдина мережа для eMBB, URLLC, mMTC; network slicing; RedCap (Rel-17) – проміжний клас між NB-IoT/LTE-M та повнофункціональним 5G NR
	Архітектурна парадигма	уніфікована стільникова платформа
Інтеграція NTN + 6G	*Роки*	2026–2030+
	Представники	3GPP NTN (Rel-17/18), Starlink Direct-to-Cell, 6G IMT-2030
	Характерні риси	супутникове покриття, ШІ-радіо, суб-ТГц, голографічні комунікації; Rel-17 – перший стандартизований випуск для 5G NR та NB-IoT NTN; Rel-21 як перший 6G-випуск очікується у 2028–2029 роках
	Архітектурна парадигма	глобальна конвергенція наземних та неназемних мереж

Еволюція бездротових протоколів також яскраво ілюструється зміною ролі радіочастотного спектра. У пропрієтарну еру виробники використовували неліцензовані діапазони (ISM) хаотично, без гарантій сумісності. З появою стандартів Wi-Fi та Bluetooth неліцензований спектр став регульованим середовищем із чіткими правилами співіснування (LBT, CSMA/CA). Стільникові технології (2G–4G) тривалий час послуговувалися виключно ліцензованим спектром, що гарантувало QoS, але здорожувало розгортання. 5G вперше запровадив концепцію спільного використання спектра (spectrum sharing), включаючи NR-U (NR in Unlicensed Spectrum) та агрегацію ліцензованих і неліцензованих смуг, що стирає межу між цими світами. Перспективні дослідження 6G передбачають використання спектра аж до терагерцового діапазону, де відкриваються величезні смуги для надшвидкісної передачі, але поширення сигналу стає подібним до оптичного — лише в межах прямої видимості. Схематично цю траєкторію показано на рисунку 13.3.



Умовні позначення

- Низькі частоти (100 МГц – 1 ГГц): ISM-діапазони для LPWAN та IoT
- 2,4 ГГц: масовий діапазон для Wi-Fi та Bluetooth
- 1–2 ГГц: NTN (супутникові системи, IoT)
- Суб-6 ГГц (600 МГц – 6 ГГц): 5G NR (FR1)
- 5–7+ ГГц: Wi-Fi 6E/7 (розширення смуги)
- 24–52 ГГц: 5G mmWave (FR2)
- 100–300 ГГц: перспективний діапазон 6G

ISM – Industrial, Scientific, Medical
LPWAN – Low Power Wide Area Network
NTN – Non-Terrestrial Networks
FR1 – Frequency Range 1 (sub-6 GHz)
FR2 – Frequency Range 2 (mmWave)
BLE – Bluetooth Low Energy
IoT – Internet of Things

Зростання частоти дозволяє збільшити смугу пропускання, але вимагає нових методів обробки сигналів (beamforming, RIS).

Джерела (офіційні специфікації та звіти)

- *ITU-R M.1036-6: Radio frequency spectrum utilization — Overview (2020).*
- *3GPP TR 38.913: Study on scenarios and requirements for next generation access technologies (Release 17, 2022).*
- *IEEE Std 802.11-2020: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.*
- *LoRa Alliance®: LoRaWAN® 1.0.4 Regional Parameters.*
- *ETSI EN 300 328 V2.2.2: Wideband transmission systems; Data transmission equipment operating in the 2,4 GHz ISM band.*
- *GSMA Intelligence: The Road to 5G – Evolution to 2030 and Beyond (2023).*
- *FCC Report and Order: Expanding Flexible Use in Mid-Band Spectrum (FCC 19-138).*

Рисунок 13.3 – Еволюція використання радіочастотного спектра бездротовими протоколами

Еволюція бездротових протоколів є історією поступового усунення бар'єрів — спочатку між виробниками (через відкриті стандарти), потім між класами пристроїв (через універсальні платформи 5G/6G), потім між наземним і супутниковим сегментами (через NTN), і, зрештою, між фізичним, цифровим та біологічним світами (через 6G). Для інженера це означає, що навички, набуті при роботі з конкретним протоколом, є цінними, але справжня професійна стійкість полягає в розумінні глибинних тенденцій — конвергенції, віртуалізації радіоінтерфейсу та інтеграції з хмарними обчисленнями. Наступний розділ, присвячений інтерфейсам для периферійного штучного інтелекту (Edge AI), покаже, як ці тенденції втілюються в конкретних апаратних рішеннях для високопродуктивних вбудованих систем.

РОЗДІЛ 14.
ІНТЕРФЕЙСИ ДЛЯ ШТУЧНОГО ІНТЕЛЕКТУ НА КРАЮ (EDGE AI)

14.1. MIPI CSI-2 та DSI для підключення камер і дисплеїв до NPU (Neural Processing Unit)

Розвиток периферійного штучного інтелекту (Edge AI) перетворив системи комп'ютерного зору з експериментальних лабораторних установок на масово впроваджувані вбудовані рішення, що застосовуються в автономних роботах, системах відеоспостереження з аналітикою, розумних камерах контролю якості на виробництві та автомобільних асистентах водія. Центральним елементом таких систем є нейронний процесор (NPU) — спеціалізований апаратний блок, оптимізований для виконання операцій матричного множення над тензорами. Однак навіть найпотужніший NPU марний без швидкого, надійного та енергоефективного інтерфейсу, здатного передавати відеопотік високої роздільної здатності від сенсора до обчислювального ядра. Саме для цієї задачі альянсом MIPI (Mobile Industry Processor Interface) були розроблені два комплементарні стандарти — CSI-2 (Camera Serial Interface 2) та DSI (Display Serial Interface), які стали де-факто універсальним способом підключення камер і дисплеїв у переважній більшості мобільних та вбудованих платформ. Сучасні специфікації MIPI CSI-2 v4.0 (MIPI Alliance, 2022) та MIPI DSI-2 v2.0 (MIPI Alliance, 2021) забезпечують пропускну здатність до 18 Гбіт/с на один порт, що достатньо для передавання нестисненого 4K-відео з частотою 60 кадрів/с або 8K з частотою 30 кадрів/с, відкриваючи шлях до аналізу зображень найвищої чіткості безпосередньо на периферійному пристрої.

MIPI CSI-2 є послідовним диференційним інтерфейсом, призначеним для передавання відеоданих від сенсора зображення до хостового процесора (SoC, NPU). Його архітектура базується на однонапрямленій передачі даних з використанням однієї або кількох диференційних пар (ліній, lanes), які утворюють лінк D-PHY або C-PHY. D-PHY (MIPI D-PHY v2.5, 2021) забезпечує швидкість до 4,5 Гбіт/с на лінію з використанням синхронної передачі (тактовий сигнал + лінії даних), тоді як C-PHY (MIPI C-PHY v2.0, 2020) досягає до 6,5 Гсимв/с на лінію завдяки кодуванню 3-фазного символу, що дозволяє передавати більше бітів за такт. Кількість ліній може варіюватися від 1 до 8, причому кожна лінія працює незалежно, а дані розподіляються між ними на рівні байтів (lane distribution).

CSI-2 використовує ієрархічну модель протоколу: на транспортному рівні дані організовані в пакети (Long Packet, Short Packet), які містять заголовок ECC та підрахунок кількості слів. На логічному рівні пакети групуються у віртуальні канали (Virtual Channels, VC 0–31), що дозволяє мультиплексувати потоки від різних джерел (наприклад, кількох сенсорів на одній шині). Кожен віртуальний канал може передавати кадри в одному з кількох форматів: RAW (баєрівський патерн, 6–14 біт на піксель), YUV 4:2:2/4:2:0, RGB або JPEG-стиснені дані. Така гнучкість дозволяє NPU отримувати дані безпосередньо в тому форматі, який очікує на вході нейромережа, уникаючи зайвих перетворень.

MIPI DSI, своєю чергою, призначений для передавання зображення від хостового процесора до дисплея, однак у контексті Edge AI він також використовується для виведення результатів інференції (накладання рамок виявлення об'єктів, семантичної сегментації) на екран оператора або у відеопотік для запису. DSI також базується на D-PHY або C-PHY і підтримує два режими роботи: командний (Command Mode) для низькошвидкісного керування регістрами дисплея (подібно до інтерфейсу DBI) та відеорежим (Video Mode) для високошвидкісної потокової передачі пікселів. У відеорежимі DSI передає кадри з використанням синхронізуючих пакетів (VSS, HSS, VSE, HSE), аналогічних до традиційних сигналів V-Sync та H-Sync, що забезпечує сумісність із стандартними LCD-контролерами. Порівняння ключових параметрів CSI-2 та DSI наведено в таблиці 14.1.

Таблиця 14.1 – Основні характеристики MIPI CSI-2 v4.0 та DSI-2 v2.0

Параметр	MIPI CSI-2	MIPI DSI-2
Напрямок передавання	від сенсора до хоста (однонапрямлений)	від хоста до дисплея (однонапрямлений)
Фізичний рівень	D-PHY v2.1 / C-PHY v2.0	D-PHY v3.0 / C-PHY v2.0/v2.1
Максимальна кількість ліній	до 8 (D-PHY) / до 4 тріо (C-PHY)	до 8 (D-PHY) / до 4 тріо (C-PHY)
Максимальна пропускна здатність	до 18 Гбіт/с (D-PHY 4×4,5 Гбіт/с) / до 41,1 Гбіт/с (C-PHY 3 тріо)	до 36 Гбіт/с (D-PHY 4 лейни 9 Гбіт/с) / до 45 Гбіт/с (C-PHY 3 тріо 14,86 Гбіт/с)
Підтримувані формати пікселів	RAW6/7/8/10/12/14/16/20, RGB, YUV420/422, JPEG	RGB (16/18/24/30/36 біт), YCbCr422, YCbCr420
Віртуальні канали	до 32 (C-PHY) / до 16 (D-PHY)	до 4
Специфікація	MIPI CSI-2 v4.0 (2022)	MIPI DSI-2 v2.0 (2021)

Конфігурація та керування сенсором зображення або дисплеєм через MIPI-інтерфейс зазвичай виконується за допомогою окремого службового каналу I²C (або SPI), через який хост зчитує та записує внутрішні регістри пристрою. Типовий процес ініціалізації камери в системі Edge AI, показаний на рисунку 14.1, включає виявлення сенсора за адресою I²C, завантаження таблиці регістрів для встановлення роздільної здатності, частоти кадрів та формату пікселів, конфігурацію D-PHY параметрів (швидкість ліній, кількість ліній) та запуск відеопотоку. Після цього NPU через прямий доступ до пам'яті (DMA) отримує кадри та передає їх на вхід нейронної мережі.

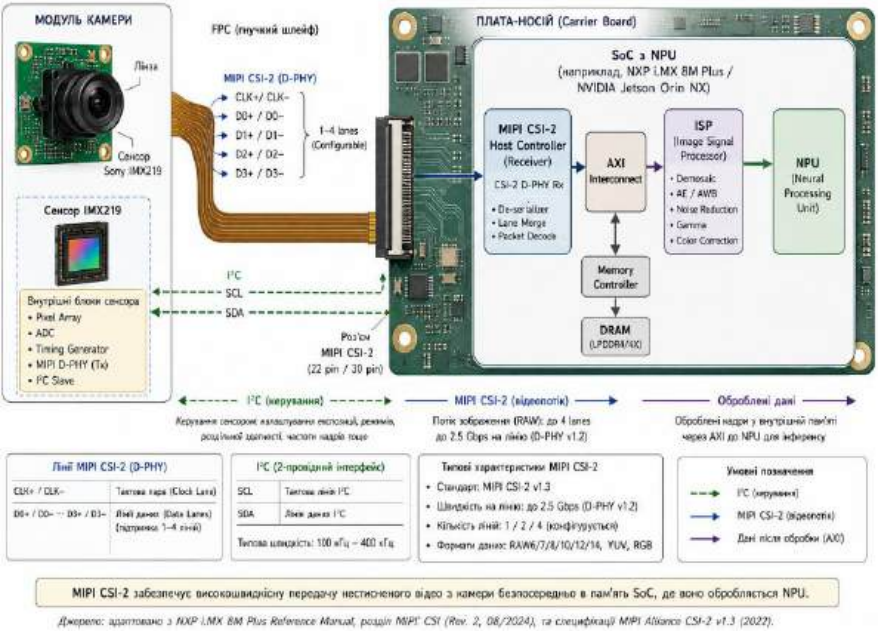


Рисунок 14.1 – Архітектура підключення камери до NPU через MIPI CSI-2

У лістингу 14.1 наведено фрагмент коду на C для STM32MP257 (який має вбудований контролер MIPI CSI-2 та NPU), що демонструє конфігурацію сенсора OV5640 через I²C та базове налаштування CSI-2 приймача за допомогою бібліотеки V4L2 (Video4Linux2), яка є стандартом де-факто для роботи з відеопристроями в Linux-системах на вбудованих платформах.

Лістинг 14.1 – Конфігурація камери OV5640 через I²C та ініціалізація CSI-2 приймача на STM32MP2 (Linux, V4L2)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <linux/i2c-dev.h>
#include <linux/videodev2.h>
#include <sys/mman.h>
// ====================
// 1. Ініціалізація OV5640 через I²C (запис масиву регістрів)
// ====================
#define OV5640_I2C_ADDR  0x3C
#define I2C_DEV  "/dev/i2c-1"
/* Спрощений масив регістрів для OV5640 (приклад) */
struct regval {    uint16_t reg;    uint8_t  val;   };
struct regval ov5640_init_regs[] = {
    {0x3000, 0x0F},  /* Soft reset */
    {0x3002, 0x1C},  /* Power control */
    /* ... (повний набір регістрів згідно з даташитом) ... */
};
void ov5640_init_i2c(void) {
    int fd = open(I2C_DEV, O_RDWR);
    if (fd < 0) { perror("open i2c"); return; }
    if (ioctl(fd, I2C_SLAVE, OV5640_I2C_ADDR) < 0) {
        perror("ioctl i2c slave");
        close(fd);        return;    }
    for (int i = 0; i < sizeof(ov5640_init_regs)/sizeof(struct
regval); i++) {
        uint8_t buf[3];
        buf[0] = (ov5640_init_regs[i].reg >> 8) & 0xFF;
        buf[1] = ov5640_init_regs[i].reg & 0xFF;
        buf[2] = ov5640_init_regs[i].val;
        if (write(fd, buf, 3) != 3) {
            fprintf(stderr, "Failed to write reg 0x%04X\n",
ov5640_init_regs[i].reg);        }    }    close(fd);   }
// ==================
// 2. Налаштування V4L2 для захоплення відео (CSI-2 + DMA)
// ==================
#define VIDEO_DEV "/dev/video0"
int main(void) {
    // 2.1 Ініціалізація OV5640 через I²C
    ov5640_init_i2c();
    // 2.2 Відкриття відеопристрою
    int fd = open(VIDEO_DEV, O_RDWR);
    if (fd < 0) { perror("open video"); return -1; }
    // 2.3 Встановлення формату YUYV, 1280x720
    struct v4l2_format fmt = {0};
    fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    fmt.fmt.pix.width = 1280;
    fmt.fmt.pix.height = 720;
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
    fmt.fmt.pix.field = V4L2_FIELD_NONE;
    if (ioctl(fd, VIDIOC_S_FMT, &fmt) < 0) {
```

```
55:         perror("VIDIOC_S_FMT");
56:         close(fd);          return -1;     }
57:     // 2.4 Встановлення частоти кадрів 30 fps
58:     struct v4l2_streamparm parm = {0};
59:     parm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
60:     parm.parm.capture.timeperframe.numerator = 1;
61:     parm.parm.capture.timeperframe.denominator = 30;
62:     if (ioctl(fd, VIDIOC_S_PARM, &parm) < 0) {
63:         perror("VIDIOC_S_PARM");
64:         close(fd);          return -1;     }
65:     // 2.5 Запит буферів у режимі DMA (MMAP або DMABUF)
66:     struct v4l2_requestbuffers req = {0};
67:     req.count = 4;
68:     req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
69:     req.memory = V4L2_MEMORY_MMAP;  // або V4L2_MEMORY_DMABUF
70:     if (ioctl(fd, VIDIOC_REQBUFS, &req) < 0) {
71:         perror("VIDIOC_REQBUFS");
72:         close(fd);          return -1;     }
73:     // 2.6 Постановка буферів у чергу (QBUF)
74:     for (int i = 0; i < req.count; i++) {
75:         struct v4l2_buffer buf = {0};
76:         buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
77:         buf.memory = V4L2_MEMORY_MMAP;
78:         buf.index = i;
79:         if (ioctl(fd, VIDIOC_QBUF, &buf) < 0) {
80:             perror("VIDIOC_QBUF");
81:             close(fd);              return -1;         }
82:     }
83:     // 2.7 Запуск потоку (STREAMON)
84:     enum v4l2_buf_type type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
85:     if (ioctl(fd, VIDIOC_STREAMON, &type) < 0) {
86:         perror("VIDIOC_STREAMON");
87:         close(fd);          return -1;     }
88:     printf("Захоплення відео запущено. Кадри передаються через
89:CSI-2 та DMA.\n");
90:  // Тут додаток може виконувати інференцію, обробку кадрів тощо.
91:     // Кадри автоматично надходять у буфери, звідки їх можна
92:зчитувати.
93:     close(fd);     return 0;    }
```

Альтернативою MIPI для Edge AI у специфічних нішах виступають інтерфейси USB 3.2 Vision (UVC) для зовнішніх камер, GigE Vision (Ethernet) для промислових розподілених систем та GMSL/FPD-Link для автомобільних застосувань із довгими кабелями. Однак саме MIPI CSI-2 залишається неперевершеним за енергоефективністю (одиниці мВт на лінію), компактністю (4–6 провідників) та вартістю реалізації для більшості вбудованих Edge AI-пристроїв. Таким чином, MIPI CSI-2 та DSI утворюють критичну ланку в конвеєрі даних штучного інтелекту на краю, забезпечуючи безперебійне постачання візуальної інформації до нейронного процесора. Наступний підрозділ, присвячений підключенню інерційних датчиків

із вбудованим машинним навчанням через I²C/SPI, розгляне інший клас інтерфейсів, де швидкість поступається місцем мініатюризації та наднизькому енергоспоживанню.

14.2. I²C/SPI до спеціалізованих акселерометрів з вбудованим ML (LSM6DSOX, IMU з деревами рішень)

Периферійний штучний інтелект не обмежується системами комп'ютерного зору, розглянутими в попередньому підрозділі. Значна частина інформації про фізичний світ надходить від інерційних датчиків — акселерометрів, гіроскопів, магнітометрів, — які реєструють вібрації, орієнтацію, лінійне та кутове прискорення. Традиційний підхід до аналізу цих даних передбачає безперервне зчитування сирих показів сенсора через стандартний послідовний інтерфейс (I²C або SPI) з подальшим виконанням алгоритмів цифрової обробки сигналів та машинного навчання на центральному процесорі мікроконтролера. Однак така модель має фундаментальний недолік: ядро мікроконтролера мусить бути постійно активним або часто прокидатися для вибірки даних, що призводить до неприйнятного енергоспоживання для батарейних пристроїв, які повинні працювати роками. Відповіддю на цей виклик стала поява нового покоління «інтелектуальних» інерційних датчиків, які містять вбудовані апаратні блоки машинного навчання — скінченні автомати (Finite State Machines, FSM) та класифікатори на основі дерев рішень (Decision Tree Logic). Ці блоки дозволяють виконувати розпізнавання шаблонів руху, жестів, аномалій вібрації та інших патернів безпосередньо всередині датчика, безперервно аналізуючи потік даних на частоті вихідних даних (ODR) і пробуджуючи хост-процесор лише при виявленні значущої події. У цьому підрозділі ми розглянемо архітектуру та принцип роботи таких сенсорів на прикладі LSM6DSOX (STMicroelectronics) — одного з перших масових IMU з вбудованим ML-ядром, а також проаналізуємо особливості їх підключення через стандартні інтерфейси I²C та SPI.

LSM6DSOX є 6-осьовим інерційним вимірювальним модулем (IMU), який поєднує тривісний акселерометр із діапазоном до ±16g та тривісний гіроскоп із діапазоном до ±2000 dps в єдиному корпусі LGA-14L розміром 2,5×3,0×0,83 мм. Його унікальність полягає в наявності вбудованого апаратного ядра машинного навчання (Machine Learning Core, MLC), яке реалізує логіку на основі дерев рішень, та програмованих скінченних автоматів (FSM). Фізичне підключення до хост-мікроконтролера виконується через стандартний I²C (Fast-mode Plus, до 1 МГц) або SPI (до 10 МГц

у 4-провідному режимі). Адреса I²C є конфігурованою: при низькому рівні на виводі SDO/SA0 вона становить 0x6A, при високому — 0x6B. Схема підключення є гранично простою і потребує лише чотирьох провідників для I²C (VDD, GND, SDA, SCL) або шести для SPI (VDD, GND, SCLK, MOSI, MISO, CS), а також опціональних сигналів переривання INT1/INT2, які є ключовими для енергоефективної роботи: замість того щоб мікроконтролер періодично опитував датчик, датчик сам генерує переривання при спрацьовуванні ML-класифікатора, пробуджуючи MCU лише тоді, коли це дійсно необхідно.

Архітектура MLC у LSM6DSOX базується на конвеєрі: спочатку дані з акселерометра та гіроскопа проходять через блок попередньої обробки, який обчислює статистичні ознаки (features) — середнє, дисперсію, мінімум, максимум, енергію, амплітуду, частоту переходів через нуль тощо — на ковзному вікні фіксованої довжини. Обчислені ознаки подаються на вхід дерева рішень (Decision Tree), яке може мати до 8 рівнів глибини та до 4 незалежних дерев. Кожен вузол дерева містить умову порівняння однієї з ознак із порогом, а кожен листок відповідає одному з до 16 вихідних класів. Результат класифікації (номер класу) зберігається у внутрішньому регістрі датчика та може генерувати переривання на виводах INT1/INT2. Принципово важливо, що вся ця обробка виконується апаратно, без участі вбудованого мікроконтролера датчика, на частоті вихідних даних (ODR), яка може сягати 6,66 кГц для акселерометра. Це дозволяє виявляти навіть дуже короткочасні події, які могли б бути пропущені при програмній обробці на хості.

Конфігурація MLC виконується за допомогою інструменту Unico GUI (STMicroelectronics), який дозволяє завантажити набір навчальних даних (log-файли), обчислити ознаки, навчити дерево рішень, згенерувати масив регістрових значень (конфігурацію ознак і самого дерева) та експортувати їх у вигляді файлу .ucf (Unico Configuration File). Хост-мікроконтролер під час ініціалізації записує цей масив регістрів у датчик через I²C або SPI, після чого MLC починає працювати автономно. У лістингу 14.2 наведено фрагмент коду для STM32, який демонструє завантаження конфігурації MLC із заздалегідь згенерованого файлу в LSM6DSOX через I²C.

Порівняння енергоефективності традиційного підходу (MCU постійно опитує датчик і виконує ML) та підходу з вбудованим MLC наведено в таблиці 14.2. Дані базуються на прикладній нотатці STMicroelectronics AN5272 «LSM6DSOX: always-on 3D accelerometer and 3D gyroscope with embedded machine learning core» (2022) та вимірюваннях для сценарію розпізнавання жестів.

Лістинг 14.2 – Завантаження конфігурації MLC з файлу .ucf в LSM6DSOX через I²C на STM32 (HAL)

```
// Джерела: LSM6DSOX Datasheet, STM32CubeH7 HAL Driver,
ST Community, OpenMV lsm6dsox driver
#include "lsm6dsox_reg.h"
#include "stm32xxxx hal.h"
// ----- 1. Конфігураційний масив (.ucf -> .h) -----
/** * @brief Масив конфігурації, згенерований Unico GUI
у форматі .h
 * @note  Unico GUI (або MEMS Studio) експортує .ucf у вигляді
масиву пар {адреса, значення} */
typedef struct {
    uint8_t address; // Адреса регістра
    uint8_t data; // Значення для запису
} ucf_line_t;
const ucf_line_t LSM6DSOX_configuration[] = {
    {0x10, 0x00}, // Вимкнення акселерометра
    {0x11, 0x00}, // Вимкнення гіроскопа
    {0x01, 0x80}, // FUNC_CFG_ACCESS: доступ до вбудованих
функцій
    {0x17, 0x40}, // Увімкнення розширеного доступу до сторінок
    {0x02, 0x11},   // PAGE_SEL: вибір сторінки
    {0x08, 0xEA},   // PAGE_ADDRESS: адреса для запису
    {0x09, 0x52},   // PAGE_VALUE: значення для запису
    // ... решта конфігураційних записів з .ucf ...
};
// ----- 2. Завантаження конфігурації в MLC -----
/** * @brief Завантаження конфігурації MLC з масиву в LSM6DSOX
через I²C
* @param  dev_ctx  Вказівник на контекст пристрою (I²C)
* @param  ucf      Вказівник на масив конфігурації
* @param  size     Кількість записів у масиві
* @note  Послідовно записує кожну пару {адреса, значення}
* @note Регістр 0x01(FUNC_CFG_ACCESS) дозволяє доступ до MLC */
void lsm6dsox_mlc_load_cfg(stmdev_ctx_t *dev_ctx,
                           const ucf_line_t *ucf, uint16_t size) {
    for (uint16_t i = 0; i < size; i++) {
   // Запис у регістр через I²C (обгортка над HAL I2C Mem Write)
     lsm6dsox_write_reg(dev_ctx, ucf[i].address, ucf[i].data);
        HAL_Delay(1);   // Невелика затримка для стабільності
    }
}
// ----- 3. Увімкнення MLC та налаштування переривань -----
/** * @brief Увімкнення MLC та налаштування маршрутизації
переривань на INT1
 * @note  Після завантаження конфігурації необхідно увімкнути
MLC
 * @note  Переривання MLC можна направити на INT1 через MLC_INT1
(0Dh) */
void lsm6dsox_mlc_enable(stmdev_ctx_t *dev_ctx) {
    uint8_t value;
    // 1. Увімкнення MLC (наприклад, через регістр CTRL1 або
EMB FUNC EN)
    //    Залежить від конкретної реалізації та версії даташита
    // 2. Налаштування маршрутизації переривання MLC на INT1
    //    MLC_INT1 (0Dh) - керування перериваннями MLC
```

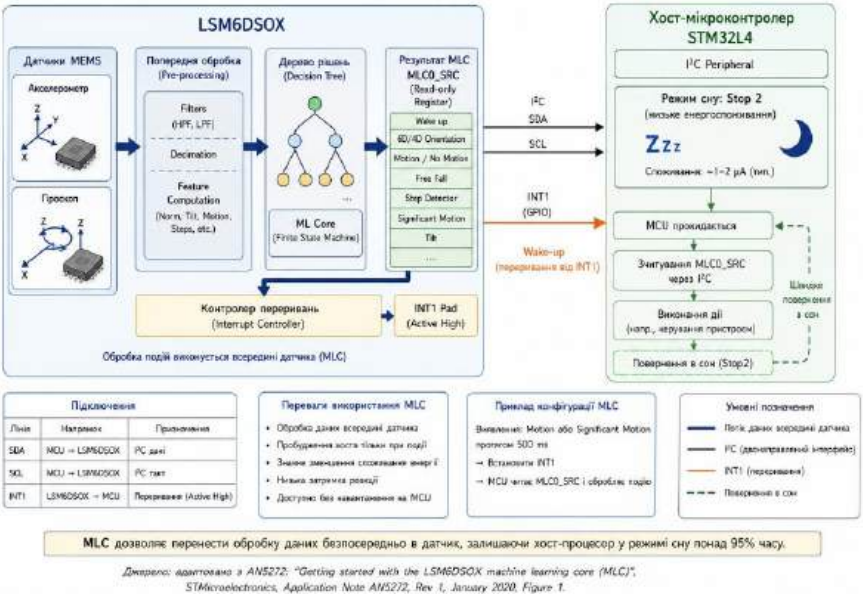
```
    lsm6dsox_read_reg(dev_ctx, LSM6DSOX_MLC_INT1, &value);
    value |= 0x01;   // Увімкнення переривання MLC на INT1
    lsm6dsox_write_reg(dev_ctx, LSM6DSOX_MLC_INT1, value);
    // 3. Увімкнення MLC (запис у відповідний регістр)
    // Наприклад, EMB FUNC EN (0x01) або інший регістр керування
    lsm6dsox_read_reg(dev_ctx, LSM6DSOX_EMB_FUNC_EN, &value);
    value |= 0x01;   // Увімкнення MLC
    lsm6dsox_write_reg(dev_ctx, LSM6DSOX_EMB_FUNC_EN, value);
}
```

Таблиця 14.2 – Порівняння енергоспоживання для задачі розпізнавання активності: тра-диційний підхід vs LSM6DSOX із MLC

Параметр	Традиційний (MCU + LSM6DSO)	Інтелектуальний (LSM6DSOX MLC)
Середній струм системи	~1,2 мА	~45 мкА
Завантаження CPU (%)	~60%	<1% (прокидається лише при виявленні)
Час реакції на подію	~10 мс (від події до спрацьовування)	<1 мс (апаратне виявлення + латентність переривання)
Автономність від батареї CR2032 (225 мА·год)	~7,5 місяців	>5 років

Архітектуру системи з LSM6DSOX у режимі MLC ілюструє рисунок 14.2. На ньому показано, що основний потік даних (сирі покази → обчислення ознак → дерево рішень) циркулює всередині датчика, а назовні через I²C/SPI виходить лише результат класифікації (номер класу) або сигнал переривання. Мікроконтролер, отримавши переривання, прокидається, зчитує через I²C регістр MLC0_SRC (0x70), визначає, яка подія сталася, і виконує відповідну дію (наприклад, передає сповіщення через BLE), після чого знову переходить у режим сну.

Окрім LSM6DSOX, інші виробники також активно розвивають напрям інтелектуальних датчиків. Bosch Sensortec пропонує BHI260AP — IMU з вбудованим програмованим ядром ARC EM4 та підтримкою завантаження навчених ML-моделей через SPI. Analog Devices випустила серію акселерометрів ADXL367 із вбудованим FSM для пробудження за шаблоном руху. Texas Instruments інтегрує логіку пробудження в мікросхеми драйверів двигунів (DRV8323). У таблиці 14.3 наведено порівняння кількох представників цього класу пристроїв.



Джерело: адаптовано з AN5272: "Getting started with the LSM6DSOX machine learning core (MLC)", STMicroelectronics, Application Note AN5272, Rev. 1, January 2020, Figure 1.

Рисунок 14.2 – Архітектура системи з LSM6DSOX у режимі MLC

Таблиця 14.3 – Порівняння інтелектуальних датчиків із вбудованим ML

Датчик	Характеристика	Значення
LSM6DSOX	*Виробник*	STMicroelectronics
	Тип ML-ядра	Decision Tree (до 8 дерев одночасно) + FSM
	Інтерфейс	I²C / SPI / MIPI I3C
	Споживання в ML-режимі	~15 мкА на одне дерево рішень; акселерометр в ультранизькому режимі – 9,5 мкА
	Типове застосування	розпізнавання жестів, вібраційний моніторинг, activity tracking
BHI260AP	*Виробник*	Bosch Sensortec
	Тип ML-ядра	32-бітний програмований мікроконтролер (ARC EM4) + Self-learning AI
	Інтерфейс	I²C / SPI
	Споживання в ML-режимі	AI-режим 25 Гц: 249 мкА; AI-режим 50 Гц: 386 мкА
	Типове застосування	розпізнавання активності, фітнес-трекінг, орієнтація

Кінець таблиці 14.3

Датчик	Характеристика	Значення
ADXL367	*Виробник*	Analog Devices
	Тип ML-ядра	FSM (програмований кінцевий автомат)
	Інтерфейс	I²C / SPI (4-wire)
	Споживання в ML-режимі	0,89 мкА при 100 Гц; 180 нА у режимі пробудження; +35 нА на детекцію подвійного тапу
	Типове застосування	вібродіагностика, крокоміри, 24/7 моніторинг
ICM-42670-P	*Виробник*	TDK InvenSense
	Тип ML-ядра	FSM + Banked FIFO
	Інтерфейс	I²C / SPI / I3C
	Споживання в ML-режимі	~25 мкА (ультранизьке споживання)
	Типове застосування	стабілізація зображення (EIS/OIS), робототехніка, IoT

Інтелектуальні датчики з вбудованим машинним навчанням, підключені через звичайні I²C/SPI, являють собою найбільш радикальний приклад парадигми Edge AI, де обчислення переміщуються з центрального процесора безпосередньо до джерела даних. Це дозволяє знизити енергоспоживання системи на два-три порядки, що є критичним для масових IoT-застосувань. Наступний підрозділ, присвячений передаванню результатів інференції через MQTT зі Sparkplug B, покаже, як агреговані дані від таких інтелектуальних сенсорів передаються до хмарних платформ у стандартизованому, інтероперабельному форматі.

14.3. Передача результатів інференції через мінімальний трафік: MQTT з Sparkplug B

Завершальною ланкою в конвеєрі периферійного штучного інтелекту є не обчислення як таке, а своєчасна, надійна та ефективна доставка результатів інференції споживачам — SCADA-системам, хмарним аналітичним платформам, сусіднім виконавчим механізмам. Хоча базовий протокол MQTT, розглянутий у розділі 9, забезпечує транспортування повідомлень за моделлю публікації-підписки, він свідомо залишає формат корисного навантаження (payload) на розсуд розробника. У масштабованих промислових системах така свобода обертається хаосом: кожен виробник створює власну структуру даних, власні імена тем, власну семантику статусів, що унеможливлює інтероперабельність без дорогої ручної інтеграції. Для

вирішення цієї проблеми консорціум Eclipse Foundation (спадкоємець активів Cirrus Link Solutions) розробив специфікацію Sparkplug — відкритий стандарт прикладного рівня поверх MQTT, оптимізований для місійно-критичних промислових IoT-застосувань. Sparkplug B (версія 3.0, Eclipse Foundation, 2023) є найактуальнішою ревізією, яка забезпечує уніфіковану модель даних, автоматичне виявлення пристроїв (Plug and Play), контроль стану (State Management) та стиснення трафіку через бінарне кодування Protocol Buffers (Protobuf). Для Edge AI-пристроїв, які часто працюють у мережах із низькою пропускною здатністю (NB-IoT, LoRaWAN) і повинні мінімізувати енерговитрати на радіопередачу, Sparkplug B є ідеальним рішенням, оскільки дозволяє передавати результати інференції у вигляді компактних бінарних повідомлень зі стандартизованою семантикою, усуваючи потребу в текстових форматах (JSON, XML).

Ключовою інновацією Sparkplug є концепція сеансу народження/смерті (Birth/Death Certificate). Коли Edge-вузол (наприклад, шлюз із NPU) підключається до MQTT-брокера, він публікує повідомлення NBIRTH (Node Birth), яке містить повний опис усіх метрик, які цей вузол буде публікувати, включно з їхніми іменами, типами даних (Int32, Float, Boolean, String, DataSet), діапазонами та одиницями вимірювання. Підписники (SCADA, хмарний сервіс) отримують цей опис і автоматично конфігурують себе для приймання даних без жодного ручного налаштування — це і є суть Plug and Play. У разі коректного відключення вузол публікує NDEATH (Node Death), повідомляючи про свій запланований вихід. Якщо ж вузол втрачає зв'язок аварійно, брокер MQTT за дорученням вузла (через Last Will and Testament, LWT) автоматично публікує NDEATH, що дозволяє системі миттєво виявити несправність і перейти в безпечний стан. Для окремих пристроїв, підключених до шлюзу (наприклад, датчиків із вбудованим ML), аналогічно використовуються DBIRTH/DDEATH. Така модель гарантує, що споживач даних завжди знає поточний стан усіх джерел інформації, що є обов'язковою вимогою для промислових систем із жорсткими критеріями надійності.

Для передавання результатів інференції Sparkplug використовує повідомлення DDATA (Device Data), яке містить лише ті метрики, що змінилися (Report by Exception, RBE). На відміну від циклічної передачі повного набору даних, RBE дозволяє зменшити трафік на 80–95%, оскільки більшість часу стан об'єкта є стабільним. Наприклад, система вібродіагностики на базі LSM6DSOX (розглянута в підрозділі 14.2) може публікувати лише два стани: «норма» та «аномалія». Доки стан не змінюється, повідомлення

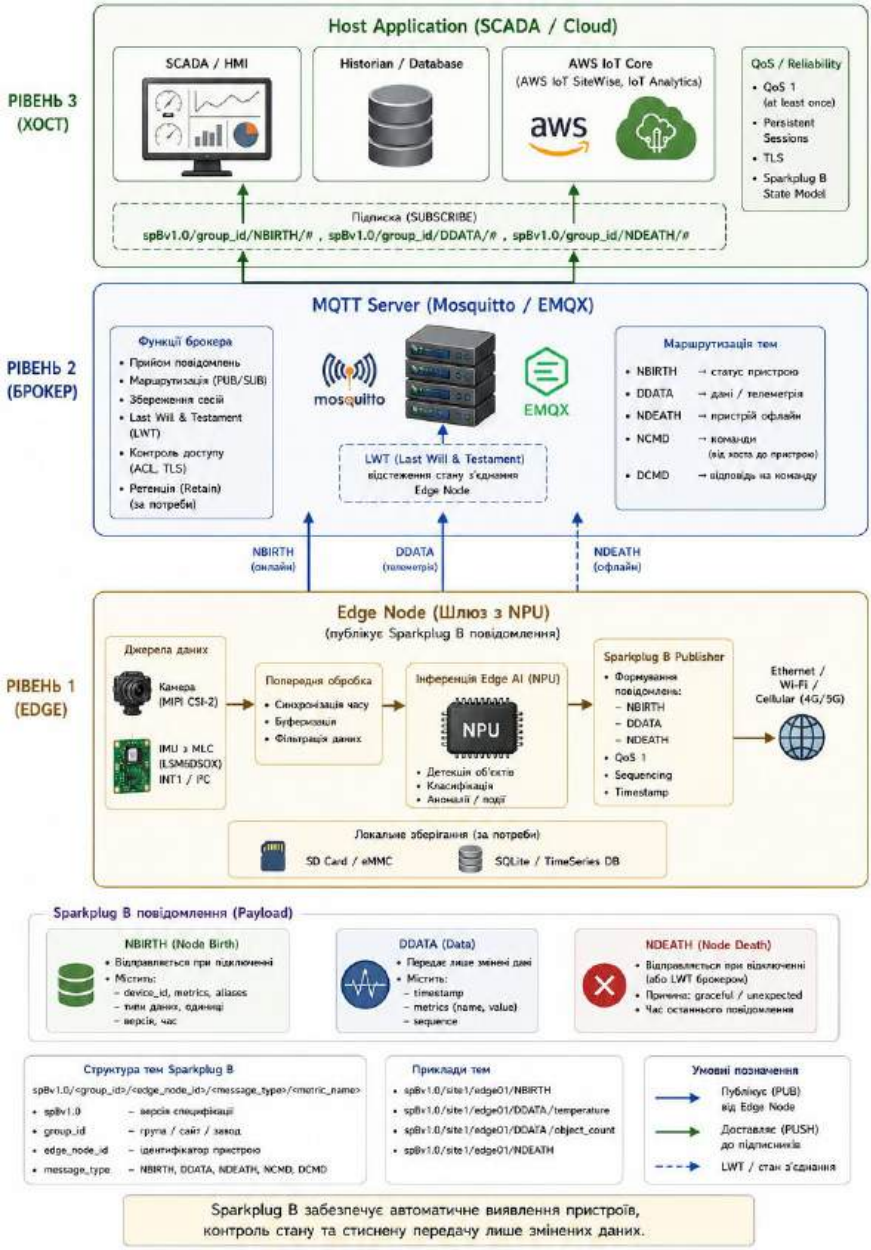
не надсилаються взагалі. Коли ж MLC датчика виявляє аномальну вібрацію, генерується переривання, мікроконтролер формує DDATA з метрикою «vibration_state = ANOMALY» і публікує її. Бінарний формат Protobuf, у якому кодуються Sparkplug-повідомлення, є значно компактнішим за JSON: типове повідомлення з однією булевою метрикою займає менше 30 байтів, тоді як еквівалентний JSON — 100–150 байтів. Це безпосередньо впливає на тривалість активної фази радіомодуля та, відповідно, на термін служби батареї.

З точки зору топології, Sparkplug вводить чітку ієрархію: Host Application (кінцевий споживач), MQTT Server (брокер) та Edge Node (шлюз, що містить пристрої). Це природно відображає трирівневу архітектуру IoT (edge — fog — cloud). Архітектуру системи з Edge AI-вузлом, що публікує результати інференції через Sparkplug B, показано на рисунку 14.3.

У таблиці 14.4 наведено порівняння традиційного «сирого» MQTT з JSON та Sparkplug B за ключовими показниками, що визначають придатність для передавання результатів інференції.

Таблиця 14.4 – Порівняння “сирого” MQTT (JSON) та MQTT зі Sparkplug B для Edge AI

Критерій	MQTT + власний JSON	MQTT + Sparkplug B (Protobuf)
Формат даних	текстовий (JSON)	бінарний (Protobuf)
Розмір повідомлення (1 Boolean + timestamp)	~120 байт	~25 байт
Семантика (самоопис)	відсутня (потрібна зовнішня документація)	вбудована (через NBIRTH/DBIRTH)
Автоматичне виявлення	ні (теми налаштовуються вручну)	так (Plug and Play)
Контроль стану (heartbeat)	ні (потрібен власний механізм)	так (Birth/Death + LWT)
Підтримка RBE (Report by Exception)	опціонально (власна реалізація)	так (вбудовано)
Інтероперабельність	низька (залежить від виробника)	висока (відкритий стандарт Eclipse)



Джерело: Eclipse Sparkplug Specification 3.0, Figure 1.

Рисунок 14.3 – Архітектура передавання результатів Edge AI через MQTT Sparkplug B

Практичне використання Sparkplug B на вбудованому шлюзі (наприклад, Raspberry Pi або STM32MP2) спирається на відкриті бібліотеки. Для Linux-платформ найпоширенішою є бібліотека Eclipse Tahu (мовою C/Java/Python), яка реалізує повний Sparkplug-стек. У лістингу 14.3 наведено фрагмент коду на Python, що демонструє створення Sparkplug-вузла, публікацію NBIRTH із описом метрики «anomaly_score» та відправлення DDATA при зміні її значення. Такий код може бути розгорнутий на Edge-шлюзі, що отримує результати інференції від підключених сенсорів.

Лістинг 14.3 – Створення Sparkplug Edge Node, публікація NBIRTH та DDATA (Python, Eclipse Tahu)

```
# Джерела: Eclipse Tahu Sparkplug B v1.0, Eclipse Paho MQTT Client
import time
import uuid
from mqtt_spb_wrapper import MetricDataType, MqttSpbEntityEdgeNode
# ------ 1. Параметри MQTT та Sparkplug ------
MQTT_HOST = "localhost"
MQTT_PORT = 1883
GROUP_ID = "VibrationMonitoring"
EDGE_NODE_ID = "Node01"
DEBUG = True
# ------ 2. Створення Edge Node ------
edge_node = MqttSpbEntityEdgeNode(GROUP_ID, EDGE_NODE_ID, DEBUG)
# ------ 3. Додавання метрик ------
# Додавання метрики "Anomaly" типу Boolean
edge_node.add_metric(name="Anomaly",
                     data_type=MetricDataType.Boolean,
                     value=False)
# Додавання додаткових метрик (наприклад, "Temperature")
edge_node.add_metric(name="Temperature",
                     data_type=MetricDataType.Float,
                     value=0.0)
# ------ 4. Callback для команд (NCMD) ------
def on_command(cmd):
    print(f"Отримано команду: {cmd}")
edge_node.on_command = on_command
# ------ 5. Публікація NBIRTH (Birth Certificate) ------
edge_node.publish_birth()  # Публікація NBIRTH
# ------ 6. Головний цикл: публікація DDATA ------
def get_inference_result():
# Імітація результату інференції (замінити на реальну логіку)
    return True if time.time() % 10 > 5 else False
while True:
    # Оновлення значення метрики
    anomaly_flag = get_inference_result()
    edge_node.set_metric_value("Anomaly", anomaly_flag)
    # Оновлення температури (приклад)
    edge_node.set_metric_value("Temperature", 25.5 +
(time.time() % 10) * 0.1)
```

```
    # Публікація DDATA (оновлення даних)
    edge_node.publish_data()
    time.sleep(1)  # Період публікації
# ------ 7. Завершення: публікація NDEATH ------
# При виході з програми публікується NDEATH
edge_node.publish_death()
```

Для мікроконтролерів із обмеженими ресурсами (наприклад, STM32H7) повноцінна підтримка Sparkplug B ускладнена через значний обсяг бібліотек Protobuf та Tahu. Однак можлива реалізація полегшеної сумісності (Sparkplug Lite): формування фіксованих бінарних пакетів, що вкладаються в структуру DDATA без динамічного NBIRTH. У такому випадку опис метрик зберігається в документації або конфігураційному файлі, а самі дані кодуються вручну, що дозволяє вкластися в 1–2 КБ флеш-пам'яті. Хоча це не є повною відповідністю стандарту, такий підхід забезпечує сумісність зі Sparkplug-споживачами (SCADA Ignition, Canary Labs, HiveMQ) на рівні формату повідомлень.

MQTT зі Sparkplug B є оптимальним рішенням для передавання результатів інференції Edge AI, оскільки поєднує легковагість MQTT-транспорту зі строгою семантикою промислового стандарту, автоматичним виявленням пристроїв, контролем стану та мінімальним обсягом трафіку завдяки бінарному кодуванню та передаванню лише змін. Цим завершується розділ, присвячений інтерфейсам для штучного інтелекту на краю; наступний, останній розділ посібника розгляне питання безпеки інтерфейсів взаємодії, що є критичним для впровадження IoT у реальному секторі.

РОЗДІЛ 15.
БЕЗПЕКА ІНТЕРФЕЙСІВ ВЗАЄМОДІЇ

15.1. Шифрування на фізичному рівні: AES-128/256 у апаратних модулях радіочастотних трансиверів

Безпека в Інтернеті речей не є опціональним додатком, що «вмикається» на прикладному рівні після завершення розроблення функціональності. Вона має бути наскрізною, багаторівневою і, що найважливіше для ресурсообмежених пристроїв, енергоефективною. Саме тому сучасні радіочастотні трансивери та системи на кристалі (SoC) переносять критичні операції шифрування з програмного забезпечення на апаратний рівень, інтегруючи виділені криптографічні прискорювачі (hardware crypto accelerators) безпосередньо в тракт обробки пакетів. Це дозволяє виконувати шифрування та автентифікацію даних «на льоту», без завантаження центрального процесора та без додаткових затримок, що були б неминучими при програмній реалізації. Центральне місце серед алгоритмів, реалізованих апаратно, посідає Advanced Encryption Standard (AES), стандартизований Національним інститутом стандартів і технологій США у документі FIPS PUB 197 (2001) і схвалений для використання в урядових та комерційних системах у всьому світі. Саме AES, з довжиною ключа 128 або 256 бітів, є тим криптографічним примітивом, на якому базуються протоколи безпеки переважної більшості бездротових інтерфейсів — від Bluetooth LE до LoRaWAN.

AES є симетричним блоковим шифром, що оперує блоками даних фіксованої довжини 128 біт (16 байтів). Сам по собі шифр AES забезпечує лише конфіденційність, перетворюючи відкритий текст у шифротекст за допомогою секретного ключа. Однак для більшості комунікаційних протоколів однієї конфіденційності недостатньо — необхідна також автентифікація повідомлення, яка гарантує цілісність даних і те, що вони дійсно надійшли від легітимного відправника. Для цього AES використовується у складі автентифікованих режимів шифрування (Authenticated Encryption with Associated Data, AEAD). Найпоширенішим серед них у світі IoT є режим CCM (Counter with CBC-MAC), стандартизований у NIST SP 800-38C (2004). CCM поєднує шифрування в режимі лічильника (CTR) для забезпечення конфіденційності та обчислення коду автентифікації повідомлення (MAC) за допомогою алгоритму CBC-MAC для забезпечення цілісності. Важливо, що CCM використовує один і той самий ключ AES для обох операцій, що спрощує керування ключами. Альтернативою є режим GCM

(Galois/Counter Mode), який забезпечує вищу продуктивність на високошвидкісних каналах, але потребує апаратної підтримки множення в полі Галуа, яка присутня не в усіх бюджетних трансиверах. У таблиці 15.1 наведено порівняння основних режимів AES, що застосовуються в бездротових протоколах IoT.

Таблиця 15.1 – Основні режими AES, що використовуються в бездротових IoT-протоколах

Режим	Характеристика	Значення
ECB (Electronic Codebook)	*Конфіденційність*	так
	Цілісність/Автентифікація	ні
	Довжина MAC (типова)	немає
	Апаратна складність	мінімальна
	Протоколи, що використовують	не рекомендується через вразливості
CBC (Cipher Block Chaining)	*Конфіденційність*	так
	Цілісність/Автентифікація	ні (окремо HMAC)
	Довжина MAC (типова)	немає
	Апаратна складність	низька
	Протоколи, що використовують	застарілі версії TLS
CTR (Counter)	*Конфіденційність*	так
	Цілісність/Автентифікація	ні
	Довжина MAC (типова)	немає
	Апаратна складність	низька
	Протоколи, що використовують	складова CCM/GCM
CCM (Counter with CBC-MAC)	*Конфіденційність*	так
	Цілісність/Автентифікація	так
	Довжина MAC (типова)	32/64/128 біт
	Апаратна складність	середня (AES + XOR)
	Протоколи, що використовують	BLE, Zigbee, Thread (IEEE 802.15.4), LoRaWAN
GCM (Galois/Counter Mode)	*Конфіденційність*	так
	Цілісність/Автентифікація	так
	Довжина MAC (типова)	32–128 біт
	Апаратна складність	висока (потребує GF(2^{128}) множення)
	Протоколи, що використовують	Wi-Fi WPA3, TLS 1.3

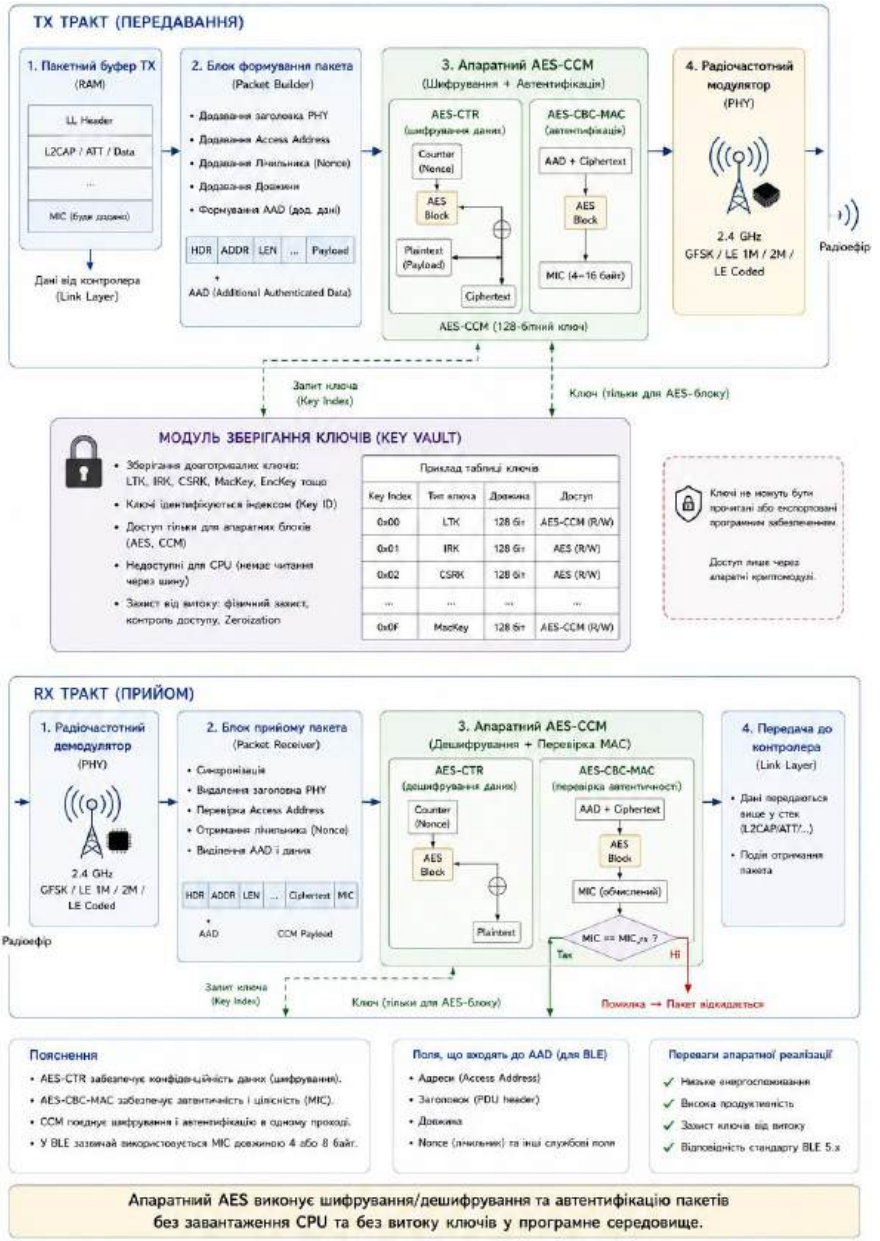
Апаратні модулі AES у радіочастотних трансиверах реалізують ці режими у вигляді скінченних автоматів, жорстко інтегрованих у конвеєр обробки пакетів (packet processing pipeline). Коли програмне забезпечення передає пакет на передавання, воно лише встановлює прапорець

«увімкнути шифрування» та вказує індекс ключа в таблиці ключів. Усю подальшу роботу — обчислення лічильника (nonce), виконання раундів AES, накладання гами на дані, обчислення MAC та вставлення його в кінець пакета — виконує апаратний прискорювач. Приймальний бік аналогічно автоматично перевіряє MAC і, якщо перевірка не проходить, пакет мовчки відкидається на апаратному рівні, навіть не сповіщаючи про це верхні рівні стеку. Така ізоляція є критично важливою, оскільки запобігає атакам на відмову в обслуговуванні (DoS) через переповнення програмних черг недійсними пакетами. Крім того, ключі AES у таких модулях зберігаються в спеціальних регістрах, які після запису стають недоступними для читання (write-only), що унеможливлює витік ключів навіть у разі компрометації програмного забезпечення.

Розглянемо конкретні реалізації апаратного AES у провідних трансиверах. У сімействі Nordic Semiconductor nRF52840 (Bluetooth 5.4 / IEEE 802.15.4) апаратний прискорювач AES-128 реалізує режим CCM згідно з вимогами Bluetooth Core Specification. Згідно з даташитом nRF52840 (Product Specification v1.9, 2022), блок ECB (AES Electronic Codebook) може виконувати операцію шифрування одного 128-бітного блоку за 2 мкс при споживанні лише 20 мкА. Повний пакет BLE з корисним навантаженням до 27 байтів шифрується в режимі CCM за кілька мікросекунд, що абсолютно непомітно на тлі тривалості самого радіообміну. Ключі зберігаються в захищеному масиві ключів (Key Vault), доступ до якого можливий лише через апаратний блок AES, але не з програмного забезпечення. Схематичне зображення проходження пакета через апаратний шифратор AES-CCM у трансивері наведено на рисунку 15.1.

У трансиверах IEEE 802.15.4 (Zigbee/Thread), таких як Texas Instruments CC2652P, апаратний модуль AES-128 реалізує CCM згідно з вимогами стандарту IEEE 802.15.4-2020 (розділ 9). Тут AES використовується на рівні MAC для забезпечення фреймової безпеки: кожен кадр може бути зашифрований та підписаний 32-, 64- або 128-бітним MAC. Апаратна підтримка гарантує, що навіть при швидкості 250 кбіт/с обробка безпеки не створює затримок.

У мережах LoRaWAN, згідно зі специфікацією LoRaWAN 1.1 (LoRa Alliance, 2022), AES-128 використовується для обчислення кількох рівнів ключів (AppKey, NwkSKey, AppSKey) та для шифрування корисного навантаження (FRMPayload) і обчислення коду цілісності повідомлення (MIC).



Джерело: адаптовано з nRF52840 Product Specification, розділ AES CCM, Nordic Semiconductor, 2022 (v1.9).

Рисунок 15.1 – Потік пакета через апаратний блок AES-CCM у трансивері (на прикладі BLE)

Трансивери Semtech SX1262 (Datasheet DS_SX1261-2_V1.2, 2021) містять вбудований апаратний AES-прискорювач, який дозволяє виконувати операції шифрування/дешифрування та обчислення MIC безпосередньо під час формування або приймання пакета, що значно розвантажує хост-мікроконтролер і зменшує енергоспоживання. У лістингу 15.1 наведено фрагмент коду для STM32WB, який демонструє конфігурацію апаратного AES-CCM для BLE-пакета.

Лістинг 15.1 – Конфігурація апаратного AES-CCM для BLE-з'єднання на STM32WB (STM32_WPAN, C)

```
// Джерела: AN5270, STM32CubeWB HAL, Community Forum
#include "stm32wbxx_hal_cryp.h"
// ------ 1. Ключ шифрування (128 біт) -----
/** * @brief Ключ AES-128 для шифрування BLE-трафіку
 * @note  Ключ встановлюється через регістри AES_KEYR0-KEYR3 */
__ALIGN_BEGIN static const uint32_t pKeyAES[4] __ALIGN_END = {
 0xAAAAAAAA, 0xBBBBBBBB, 0xCCCCCCCC, 0xDDDDDDDD // Ключ 128 біт
};
// ----- 2. Конфігурація CRYP (AES-CCM) -----
/** * @brief Ініціалізація AES-CCM
 * @note  HAL_CRYP_Init() налаштовує периферію AES1
 * @note  CRYP_AES_CCM – режим Counter with CBC-MAC
 * @note  CRYP_KEYSIZE_128B – ключ 128 біт
 * @note  CRYP_DATATYPE_8B – 8-бітні дані */
CRYP_HandleTypeDef hcryp1;
void MX_AES1_Init(void) {
    hcryp1.Instance = AES1;
    hcryp1.Init.DataType = CRYP_DATATYPE_8B; // 8-бітні дані
    hcryp1.Init.KeySize = CRYP_KEYSIZE_128B; // Ключ 128 біт
    hcryp1.Init.pKey = (uint32_t *)pKeyAES; // Вказівник на ключ
    hcryp1.Init.Algorithm = CRYP_AES_CCM; // Режим CCM
    hcryp1.Init.Header = (uint32_t *)HeaderAES; // AAD
    hcryp1.Init.HeaderSize = 4;  // Розмір AAD (слова)
    hcryp1.Init.B0 = (uint32_t *)B0AES;  // Блок B0 (nonce)
    hcryp1.Init.DataWidthUnit = CRYP_DATAWIDTHUNIT_WORD;
    hcryp1.Init.KeyIVConfigSkip = CRYP_KEYIVCONFIG_ALWAYS;
    HAL_CRYP_Init(&hcryp1);  // Ініціалізація
}
// ----- 3. Налаштування CCM (nonce, MAC довжина) -----
/** * @brief Встановлення параметрів CCM: nonce та довжина MAC
 * @note  Nonce – 13 байт (MAC-адреса + лічильник)
 * @note  MAC довжина – 4 байти (BTHome специфікація)
 * @note  HAL_CRYP_SetCCMConfig() – налаштування CCM */
uint8_t nonce[13] = { /* MAC (6B), UUID (2B), Counter (4B) */};
void CCM_Config(void) {
    // Встановлення Nonce (запис у регістри AES_IVR)
    HAL_CRYP_SetIV(&hcryp1, (uint32_t *)nonce, 4); // IV = Nonce
    // Налаштування CCM (MAC довжина = 4 байти)
    HAL_CRYP_SetCCMConfig(&hcryp1, CCM_HEADER_PHASE, 4);
}
// ------ 4. Активація шифрування BLE -----
/** * @brief Активація шифрування для BLE-з'єднання
```

```
 * @note  Налаштування AES: BLE-трафік шифрується автоматично
 * @note  Ключ передається через aci_gap_set_encryption_key()
або aci l2cap coc connect() */
void BLE_Enable_Encryption(void) {
    // Встановлення ключа шифрування для BLE-з'єднання
    aci_gap_set_encryption_key(pKeyAES, 16); // Ключ 128 біт
    // Або для COC-з'єднання:
    // aci l2cap coc connect(..., pKeyAES, 16);
}
```

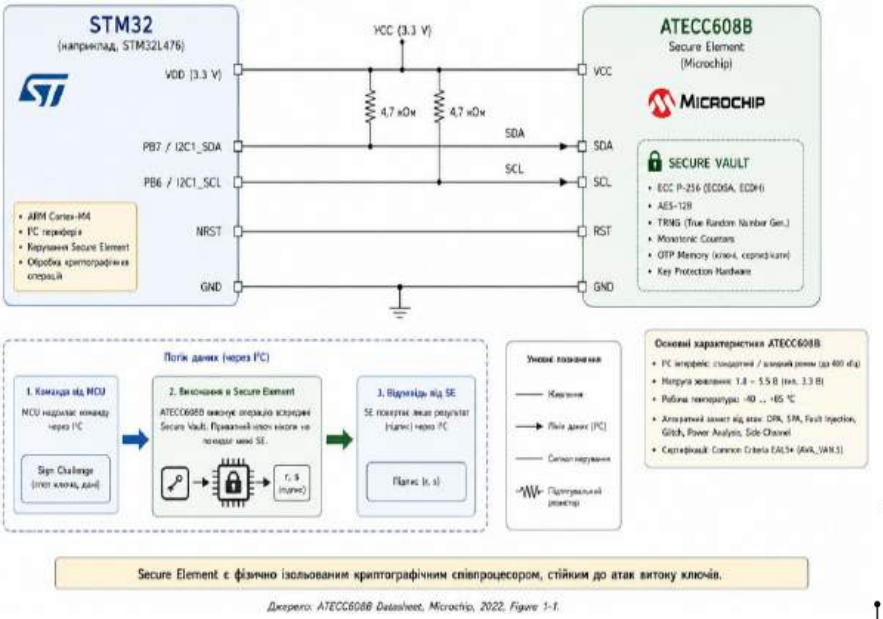
Шифрування на фізичному рівні не є панацеєю, але воно створює перший, найфундаментальніший бар'єр, який унеможливлює пасивне прослуховування ефіру та більшість активних атак на канальному рівні. Для інженера-розробника важливо розуміти, що використання апаратних криптографічних модулів є не просто «кращою практикою», а суворою вимогою для сертифікації за стандартами безпеки (IEC 62443, ETSI EN 303 645) та допуску на ринки ЄС, США та Японії. Наступний підрозділ, присвячений автентифікації пристроїв через Secure Element, розгляне, як зберігати кореневі ключі та сертифікати в захищених апаратних анклавах.

15.2. Автентифікація пристроїв: Secure Element (ATECC608), PSA Certified для Cortex-M

Надійне шифрування даних на фізичному рівні, розглянуте в попередньому підрозділі, є необхідною, але недостатньою умовою безпеки. Криптографічні алгоритми, навіть найстійкіші, марні, якщо зловмисник отримує доступ до ключів, що зберігаються у звичайній флеш-пам’яті мікроконтролера. Програмне забезпечення може бути скомпрометоване через вразливості, а фізичний доступ до пристрою дозволяє зчитування пам’яті за допомогою методів аналізу побічних каналів або зондової мікроскопії. Таким чином, кореневою проблемою безпеки IoT є не стільки криптографія, скільки зберігання секретів. Для її вирішення було створено концепцію Secure Element (SE) — окремої мікросхеми, спеціально спроектованої для захищеного зберігання криптографічних ключів, виконання обчислень із ними та протидії фізичним атакам. Одночасно, з розвитком вбудованих процесорів, виникла парадигма Platform Security Architecture (PSA) від Arm, яка стандартизує створення довіреного середовища виконання (Trusted Execution Environment, TEE) безпосередньо на кристалі. Ці два підходи — зовнішній Secure Element та інтегрована архітектура PSA Certified

— формують сучасний ландшафт автентифікації пристроїв, що уможливлює надійне onboarding у хмарні платформи, формування ланцюжка довіри (chain of trust) та захист інтелектуальної власності.

Яскравим представником класу зовнішніх Secure Element є мікросхема ATECC608B (Microchip Technology, 2022), яка реалізує повний спектр криптографічних примітивів на еліптичних кривих (ECC) та алгоритм AES-128 в апаратному виконанні. ATECC608B має сертифікацію Common Criteria EAL5+ та FIPS 140-2 Level 3, що є найвищим рівнем для комерційно доступних рішень такого класу. Її ключовою особливістю є те, що закриті ключі генеруються всередині мікросхеми за допомогою апаратного генератора випадкових чисел (TRNG) і ніколи не покидають її меж. Хост-мікроконтролер не може прочитати закритий ключ; він може лише надіслати хеш повідомлення (або challenge) і отримати підпис, обчислений усередині SE. Для симетричного шифрування сеансовий ключ також може бути узгоджений усередині SE за алгоритмом ECDH, і надалі використовуватися для апаратного AES, що повністю ізолює високочутливі операції. ATECC608B підключається до мікроконтролера через стандартний інтерфейс I²C (або однопровідний), що робить його інтеграцію гранично простою. Схему підключення такої мікросхеми до основного MCU показано на рисунку 15.2.



Джерело: ATECC608B Datasheet, Microchip, 2022, Figure 1-1.

Рисунок 15.2 – Схема підключення Secure Element ATECC608B до мікроконтролера STM32 через I²C

Апаратна ізоляція, яку забезпечує ATECC608B, є важливим елементом для організації безпечного onboarding-у пристроїв у хмарні сервіси. Під час заводського програмування (або на етапі ініціалізації) в SE записується унікальний сертифікат пристрою, підписаний центром сертифікації виробника. Коли пристрій уперше підключається до хмарної платформи (наприклад, AWS IoT Core), він пред'являє цей сертифікат, доводячи, що є легітимним пристроєм, виготовленим довіреним виробником. Жоден зловмисник, навіть той, що має повний контроль над основним мікроконтролером, не може скопіювати ключ із SE та створити клон пристрою. Саме ця властивість робить SE обов'язковим компонентом для систем, що вимагають автентифікації на рівні «заліза» (hardware root of trust).

Паралельно з розвитком окремих мікросхем SE, компанія Arm у 2018 році запустила ініціативу Platform Security Architecture (PSA) з метою стандартизації безпеки для мікроконтролерів Cortex-M. PSA Certified — це програма сертифікації, яка визначає три рівні безпеки (Level 1, 2, 3) для захисту від програмних, апаратних та фізичних загроз відповідно. Мікроконтролери, що отримали сертифікацію PSA Certified Level 2 і вище (наприклад, STM32L5, STM32U5 на ядрі Cortex-M33 з TrustZone), містять вбудований апаратний ізолятор, який дозволяє розділити виконання коду на безпечний (Secure) та небезпечний (Non-Secure) світи. У безпечному світі виконуються операції з ключами, оновлення прошивки та перевірка підписів, тоді як основний прикладний код працює в небезпечному світі й не має прямого доступу до секретів. PSA Certified також надає стандартний набір API для криптографії, захищеного зберігання (Protected Storage) та атестації (Initial Attestation). Остання дозволяє пристрою довести віддаленому серверу свою ідентичність, цілісність прошивки та версію апаратного забезпечення за допомогою Entity Attestation Token (EAT), стандартизованого IETF (RFC 9393). Порівняння двох підходів — зовнішнього SE та інтегрованої PSA — наведено в таблиці 15.2.

Обидва підходи не є взаємовиключними. У високобезпечних системах може застосовуватися комбінація: PSA-сертифікований мікроконтролер для ізоляції програмного забезпечення та зовнішній SE для зберігання кореневих ключів із фізичним захистом. Наприклад, STM32U5 може виконувати основну логіку в TrustZone, але використовувати ATECC608 для зберігання ключа, що підписує токени атестації.

Таблиця 15.2 – Порівняння підходів до апаратної автентифікації пристрою: зовнішній Secure Element та PSA Certified

Критерій	Зовнішній Secure Element (ATECC608B)	PSA Certified (Cortex-M33 TrustZone)
Розташування	окрема мікросхема (дискретний компонент)	інтегровано в MCU (на кристалі)
Захист від фізичного доступу	дуже високий (апаратний захист від фізичних атак, включно з аналізом живлення та сторонніми каналами)	високий (Level 2 – захист від програмних атак; Lightweight hardware attacks)
Апаратна ізоляція ключів	повна (ключі ніколи не виходять за межі чипа, апаратне зберігання до 16 ключів)	апаратна (TrustZone, IDAU/SAU – ізоляція Secure та Non-secure світів)
Типові криптографічні алгоритми	ECC P256, ECDSA, ECDH, AES-128, SHA-256, HMAC, TRNG	ECC, AES, SHA, TRNG (залежить від реалізації)
Стандартизація API	CryptoAuthLib (Microchip)	PSA Crypto API, TF-M
Вартість компонентів	додатковий чип (~$0.50)	входить у вартість MCU
Складність інтеграції	низька (I²C, готові бібліотеки)	середня (потребує налаштування TrustZone, SPM)
Приклад реалізації	ATECC608B + STM32F4 / STM32H7	STM32L5, nRF5340

У лістингу 15.2 наведено фрагмент коду на мові C для STM32, який демонструє ініціалізацію ATECC608B через I²C, зчитування його серійного номера та генерацію ECDSA-підпису для зовнішнього challenge (наприклад, від AWS IoT Core).

Лістинг 15.2 – Ініціалізація ATECC608B та підписання challenge через I²C на STM32 (CryptoAuthLib)

```
// Джерела: CryptoAuthLib API, Microchip ATECC608B Datasheet
#include "cryptoauthlib.h"
// ------ 1. Конфігурація інтерфейсу I²C -----
/** * @brief Структура конфігурації для ATECC608B через I²C
 * @note  ATCA_I2C_IFACE визначає тип інтерфейсу
 *        atcab_init() створює глобальний ATCADevice об'єкт */
ATCAIfaceCfg cfg = {
    .iface_type = ATCA_I2C_IFACE,
    .devtype    = ATECC608,
```

```
    .atcai2c = {
        .slave_address = 0xC0, // I2C-адреса ATECC608B (default)
         .bus = 1,             // Номер шини I²C (наприклад, I2C1)
         .baud = 100000,     // Швидкість 100 кГц
    },
    .wake_delay = 1500,
    .rx_retries = 20,
};
// ----- 2. Ініціалізація бібліотеки -----
/** * @brief Ініціалізація CryptoAuthLib та зв'язку з ATECC608B
 * @note  Викликається один раз на початку роботи програми */
ATCA_STATUS status = atcab_init(&cfg);
if (status != ATCA_SUCCESS) {
    // Обробка помилки ініціаліалізації
}
// ----- 3. Читання серійного номера -----
/** * @brief Отримання унікального SN (9 байт)
 * @note  atcab_read_serial_number() -> serial_num */
uint8_t serial_num[9];
status = atcab_read_serial_number(serial_num);
if (status == ATCA_SUCCESS) {
    // Серійний номер збережено в serial_num[0..8]
}
// ----- 4. Підписання challenge (ECDSA) -----
/** * @brief Підпис хешу (challenge) ключем зі слота 0
 * @param slot     Номер слота з закритим ключем (наприклад, 0)
 * @param digest   Вказівник на 32B хеш повідомлення
 * @param signature Вказівник на 64B підпис (r + s)
 * @note  Закритий ключ уже збережено в слоті SE
 *        і ніколи не покидає мікросхему */
uint8_t challenge[32] = { /* 32-байтовий хеш */ };
uint8_t signature[64];  // ECDSA P-256 підпис (r:32 б + s:32 б)
status = atcab_sign(0, challenge, signature);
if (status == ATCA_SUCCESS) {
    // Підпис успішно збережено в signature[0..63]
}
```

Автентифікація пристроїв на основі апаратно ізольованих секретів є наріжним каменем безпеки сучасного IoT. Вона трансформує пристрій із безликого джерела даних на довірену, криптографічно верифіковану сутність, без чого неможливі ані безпечне віддалене оновлення прошивки (secure FOTA), ані конфіденційна передача даних у промислових або медичних системах. З розвитком законодавства про кібербезпеку (Cyber Resilience Act) наявність апаратного кореня довіри (hardware root of trust) із сертифікацією EAL5+ або PSA Certified стає не просто конкурентною перевагою, а юридично обов'язковою вимогою для виходу на ринок.

15.3. Захист мережевих з'єднань: TLS 1.3, DTLS 1.3, OSCORE для CoAP

Апаратне шифрування та автентифікація пристроїв, розглянуті в попередніх підрозділах, створюють фундамент довіри, однак безпосередній захист даних під час їхнього передавання через потенційно ворожі мережі (Інтернет, публічний Wi-Fi, стільникові канали) потребує окремих, спеціалізованих протоколів. Ці протоколи забезпечують три класичні властивості безпечного каналу: конфіденційність (ніхто, крім адресата, не може прочитати повідомлення), цілісність (повідомлення не може бути непомітно змінене в дорозі) та автентичність (отримувач упевнений, що повідомлення надійшло саме від легітимного відправника). Протоколи захисту транспортного рівня TLS (Transport Layer Security) та його дейтаграмний аналог DTLS (Datagram TLS) є домінуючими рішеннями для TCP- та UDP-трафіку відповідно, тоді як OSCORE (Object Security for Constrained RESTful Environments) пропонує альтернативний підхід, що забезпечує безпеку на прикладному рівні для протоколу CoAP. У цьому підрозділі аналізуються сучасні версії цих протоколів (TLS 1.3, DTLS 1.3, OSCORE), їхні механізми рукостискання, особливості застосування в обмежених IoT-середовищах та взаємодія з апаратними криптографічними прискорювачами.

TLS 1.3, стандартизований IETF у RFC 8446 (2018), став результатом ґрунтовного переосмислення попередньої версії 1.2 з метою усунення накопичених вразливостей (POODLE, BEAST, Lucky13, FREAK) та радикального скорочення латентності рукостискання. Для IoT-пристроїв, що часто оперують у мережах із високою затримкою (LPWAN) або пробуджуються для сеансу зв'язку на кілька секунд, кожен зайвий обмін даними безпосередньо скорочує термін служби батареї. TLS 1.3 скорочує повне рукостискання до одного раундового обміну (1-RTT), а для повторних з'єднань із використанням попередньо розподілених ключів (PSK) — до нуля (0-RTT), коли дані додатків можуть бути передані вже в першому пакеті. Технічно це досягається завдяки тому, що клієнт у повідомленні ClientHello одразу надсилає здогадку про те, які параметри обере сервер (підтримувані набори шифрів, ключі для ефемерного обміну Діффі-Хеллмана на еліптичних кривих ECDHE), що дозволяє серверу відповісти ServerHello з уже обраними параметрами та власним ключем, після чого обидві сторони можуть негайно обчислити спільний секрет і почати шифрований обмін. Із обов'язкового переліку TLS 1.3 були вилучені всі застарілі алгоритми (RSA-обмін ключами, RC4, 3DES, CBC-режими, SHA-1), а залишені лише

AEAD-шифри (AES-GCM, ChaCha20-Poly1305), які забезпечують одночасно конфіденційність та цілісність і мають ефективну апаратну реалізацію навіть на малопотужних мікроконтролерах.

DTLS 1.3 (RFC 9147, 2022) адаптує механізми TLS 1.3 для роботи поверх ненадійного транспорту UDP, що є критичним для CoAP та інших протоколів, які уникають TCP через його накладні витрати. UDP не гарантує порядку доставки, відсутності дублікатів та самого факту доставки, тому DTLS 1.3 додає до базового TLS 1.3 явні порядкові номери в кожному записі, механізм виявлення та відкидання дублікатів (anti-replay), а також опціональний механізм «печива» (stateless cookie exchange) на початку рукостискання для захисту сервера від DoS-атак із підробкою IP-адреси. Крім того, DTLS 1.3 зберігає логіку повторних передач повідомлень рукостискання, якщо відповідь не отримана вчасно. Для IoT-пристроїв важливо, що бібліотеки mbedTLS та wolfSSL підтримують DTLS 1.3 і дозволяють використовувати режим PSK, який не потребує важких операцій з асиметричною криптографією (ECDHE), обмежуючись лише симетричним шифруванням на основі попередньо встановленого ключа, що може бути згенерований, наприклад, на етапі провіжинінгу через Secure Element.

OSCORE (Object Security for Constrained RESTful Environments, RFC 8613, 2019) пропонує принципово інший підхід. Замість того щоб захищати весь транспортний канал (pipe), OSCORE захищає окремі повідомлення прикладного рівня (payload). Це означає, що конфіденційність та цілісність забезпечуються навіть тоді, коли повідомлення проходить через проміжні вузли (CoAP-to-CoAP проксі), які не мають доступу до ключів, але можуть змінювати незахищені частини повідомлення (наприклад, опції CoAP, необхідні для маршрутизації). OSCORE використовує AEAD-шифри (AES-CCM) та захищає майже всі поля CoAP-повідомлення (URI, метод, опції, payload), за винятком тих, що свідомо залишені незахищеними для роботи проксі (External_Aad). Ключовою перевагою OSCORE є можливість групової комунікації (multicast), що неможливо для DTLS, яке встановлює захищений канал між двома точками. У системі з OSCORE сотні лампочок можуть отримувати захищені команди від шлюзу через один мультикаст-пакет, що радикально економить смугу пропускання та енергію. Порівняння трьох протоколів наведено в таблиці 15.3.

Взаємодія цих протоколів із апаратними криптографічними прискорювачами (AES, SHA), розглянутими в підрозділі 15.1, є визначальною для енергоефективності. Сучасні бібліотеки (mbedTLS для STM32, wolfSSL для ESP32) автоматично визначають наявність апаратного модуля та використовують його для обчислення AEAD-шифрів (AES-CCM для DTLS 1.3

з PSK, AES-GCM для TLS 1.3), знижуючи час обробки пакета до одиниць мікросекунд і споживання енергії — до часток мікроджоуля на операцію.

Таблиця 15.3 – Порівняння TLS 1.3, DTLS 1.3 та OSCORE для IoT

Протокол	Характеристика	Значення
TLS 1.3	*Транспортний протокол*	TCP (надійний)
	Рівень захисту	транспортний канал (pipe)
	Рукостискання	1-RTT (ECDHE), 0-RTT (PSK)
	Підтримка PSK	так
	Захист від replay-атак	вбудовано
	Групова комунікація	ні
	Сумісність із проксі	так (проксі термінує TLS)
	Типовий сценарій	MQTT (з'єднання з брокером), HTTPS
DTLS 1.3	*Транспортний протокол*	UDP (дейтаграмний)
	Рівень захисту	транспортний канал (pipe)
	Рукостискання	2-RTT (cookie + ECDHE), PSK швидше
	Підтримка PSK	так
	Захист від replay-атак	вбудовано (епохи + SN)
	Групова комунікація	ні
	Сумісність із проксі	обмежена (DTLS-термінація на проксі)
	Типовий сценарій	CoAP (точка-точка)
OSCORE	*Транспортний протокол*	UDP (будь-який)
	Рівень захисту	прикладний рівень (окремі повідомлення)
	Рукостискання	відсутнє (використовує попередньо встановлені ключі)
	Підтримка PSK	так (обов'язкова)
	Захист від replay-атак	через часткові IV (PIV) та контекст
	Групова комунікація	так (груповий режим)
	Сумісність із проксі	так (проксі може змінювати незахищені опції)
	Типовий сценарій	CoAP (групове керування, захист через проксі)

У лістингу 15.3 наведено фрагмент конфігурації DTLS 1.3 із PSK для CoAP-клієнта на STM32L5 (mbedTLS), який використовує апаратний AES-CCM для шифрування даних.

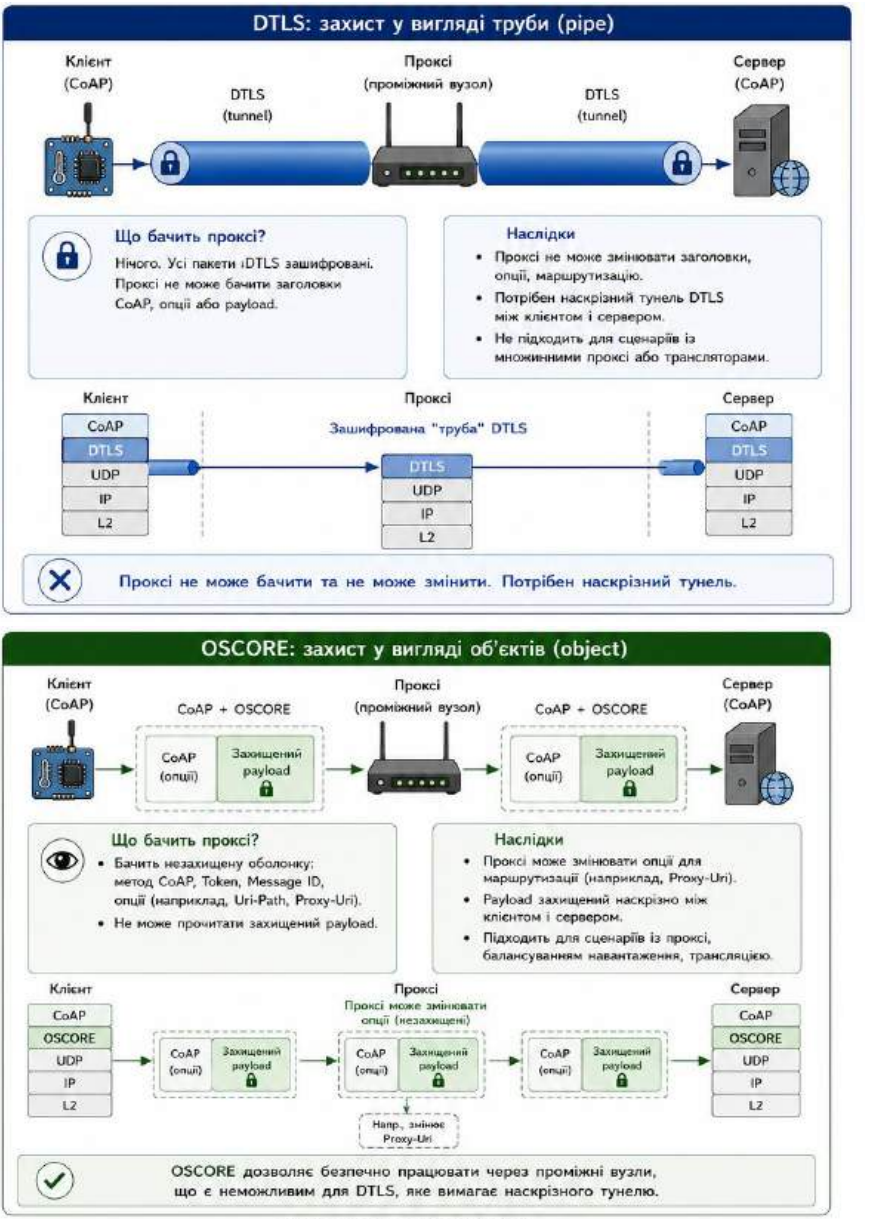
Лістинг 15.3 – Конфігурація DTLS 1.3 з PSK для CoAP на STM32L5 (mbedTLS, C)

```
// Джерела: Mbed TLS API Documentation, STM32CubeL5
#include "mbedtls/ssl.h"
#include "mbedtls/net_sockets.h"
#include "mbedtls/timing.h"
// ----- 1. Ініціалізація контекстів -----
/** * @brief Ініціалізація SSL-контексту та конфігурації
 * @note  mbedtls_ssl_init() ініціалізує структуру SSL
 *       mbedtls_ssl_config_init() ініціалізує конфігурацію */
mbedtls_ssl_context ssl;
mbedtls_ssl_config conf;
mbedtls_timing_delay_context timer;
mbedtls_ssl_init(&ssl);
mbedtls_ssl_config_init(&conf);
// ----- 2. Налаштування конфігурації -----
/** * @brief Встановлення параметрів DTLS 1.3 з PSK
 * @note  MBEDTLS_SSL_TRANSPORT_DATAGRAM – DTLS режим
 *        MBEDTLS_SSL_PRESET_SUITEB – набір шифрів для PSK
 * TLS 1.3:увімкнути MBEDTLS_SSL_TLS13_KEY_EXCHANGE_MODE_PSK */
mbedtls_ssl_config_defaults(&conf, MBEDTLS_SSL_IS_CLIENT,
   MBEDTLS_SSL_TRANSPORT_DATAGRAM, MBEDTLS_SSL_PRESET_SUITEB);
// ----- 3. Встановлення PSK -----
/** * @brief Встановлення попередньо розподіленого ключа (PSK)
 * @param psk   Вказівник на PSK (згенерований через ATECC608B)
 * @param psk_len  Довжина PSK (наприклад, 32 байти)
 * @param psk_identity  Ідентифікатор PSK (рядок)
 * @note  mbedtls_ssl_conf_psk() -> реєстрація PSK та ID
 *        PSK згенерований через SE та збережений у Flash */
uint8_t psk[32] = { /* PSK з ATECC608B */ };
const char psk_identity[] = "device_001";
mbedtls_ssl_conf_psk(&conf, psk, sizeof(psk),(const unsigned
char *)psk_identity, strlen(psk_identity));
// ----- 4. Прив'язка до UDP-сокету -----
/** * @brief Підключення DTLS до транспортного рівня (UDP)
 * @note  mbedtls_ssl_set_bio() – прив'язка send/recv до сокету
 *   mbedtls_ssl_set_timer_cb() – конфігурація таймерів DTLS */
int sock = socket(AF_INET, SOCK_DGRAM, 0);
// ... налаштування адреси сервера ...
mbedtls_ssl_set_bio(&ssl, &sock,
          mbedtls_net_send,  // Відправлення через UDP
          mbedtls_net_recv,  // Прийом через UDP
          NULL);
mbedtls_ssl_set_timer_cb(&ssl, &timer,
                         mbedtls_timing_set_delay,
                         mbedtls_timing_get_delay);
// ----- 5. Рукостискання DTLS -----
/** * @brief Виконання DTLS-рукопоскання
 * @note  mbedtls_ssl_handshake() встановлює захищений канал
```

```
48: *         Далі весь трафік апаратно шифрується (AES-CCM) */
49:int ret = mbedtls_ssl_handshake(&ssl);
50:if (ret != 0) {     // Обробка помилки    }
51:// ----- 6. Захищена передача даних -----
52:/** * @brief Відправлення та прийом захищених даних через DTLS
53: * @note  mbedtls_ssl_write/read(): апаратний AES-CCM */
54: *         (модуль PKA/CRYP у STM32L5) для шифрування */
55:char request[] = "GET /sensor/temperature";
56:mbedtls_ssl_write(&ssl, (const unsigned char *)request,
57:strlen(request));
58:char response[128];
 59:mbedtls_ssl_read(&ssl, (unsigned char *)response,
 60:sizeof(response));
```

На рисунку 15.3 схематично показано різницю між захистом транспортного рівня (DTLS) та прикладного рівня (OSCORE) для CoAP-повідомлення, що проходить через проксі. DTLS створює наскрізний тунель між клієнтом і сервером, тоді як OSCORE загортає окремі повідомлення, дозволяючи проміжному проксі змінювати незахищені частини.

Захист мережевих з'єднань в IoT не є монолітним рішенням, а вимагає виваженого вибору між DTLS (для точка-точка UDP-з'єднань з максимальною сумісністю) та OSCORE (для групових команд, роботи через проксі та максимальної енергоефективності), причому обидва підходи спираються на апаратне прискорення AEAD-шифрів для досягнення мінімального енергоспоживання. На цьому завершується огляд рівнів безпеки інтерфейсів — від фізичного шифрування до автентифікації пристроїв і захисту мережевих з'єднань. У сукупності вони формують багатошаровий захист (defense in depth), який є обов'язковою вимогою для будь-якої сучасної IoT-системи, що претендує на промислове впровадження. Матеріал цього розділу завершує посібник, підсумовуючи ключову ідею: безпека є не окремою функцією, а інтегральною властивістю правильно спроектованого інтерфейсу на всіх рівнях — від фізичного до прикладного.



Джерело: адаптовано з RFC 8613 – "Object Security for Constrained RESTful Environments (OSCORE)", Figure 2.

Рисунок 15.3 – Порівняння захисту CoAP: DTLS (pipe) vs OSCORE (object) через проксі

15.4. Протидія атакам побічними каналами на SPI/UART шини

Атаки побічними каналами (Side-Channel Attacks, SCA) становлять окремий, надзвичайно небезпечний клас загроз, який не зламує криптографічні алгоритми математично, а використовує фізичні прояви їхнього виконання в апаратурі. Коли мікроконтролер передає дані через, здавалося б, суто цифровий інтерфейс на кшталт SPI або UART, він одночасно випромінює електромагнітне поле, споживає струм із джерела живлення та витрачає час, пропорційний до кількості одиничних бітів у переданому байті. Усі ці фізичні величини є побічними каналами, які корелюють із оброблюваними даними. Вимірявши їх із достатньою точністю, зловмисник може відновити секретний ключ, конфіденційне повідомлення або навіть прошивку пристрою, жодного разу не підключившись безпосередньо до ліній даних. Як зазначено в класичній монографії Ш. Мангара, Е. Освальда та Т. Поппа «Power Analysis Attacks: Revealing the Secrets of Smart Cards» (Springer, 2007), саме простота та регулярність сигналів на послідовних шинах робить SPI та UART надзвичайно зручними об’єктами для SCA, оскільки їхні тактові сигнали (SCLK у SPI, старт-стопові біти в UART) виступають ідеальними тригерами для синхронізації вимірювального обладнання. Зі зростанням доступності інструментів на кшталт ChipWhisperer (NewAE Technology) та відкритих фреймворків на кшталт SCARED, проведення базового аналізу споживаної потужності перестало бути прерогативою спецслужб і стало реальною загрозою для комерційних IoT-пристроїв.

Основними типами SCA, що загрожують SPI та UART, є простий та диференційний аналіз споживаної потужності (SPA/DPA) та аналіз електромагнітних випромінювань (EMA). При простому аналізі потужності (SPA) зловмисник безпосередньо спостерігає за формою струму споживання під час виконання однієї операції. Наприклад, під час передавання байта через SPI струм, споживаний драйвером лінії MOSI, змінюється залежно від кількості логічних одиниць, оскільки заряджання паразитної ємності лінії вимагає більшого імпульсу струму, ніж утримання нуля. Осцилограма струму дозволяє візуально розрізнити байти 0xFF (вісім одиниць) та 0x00 (вісім нулів) без будь-якої математичної обробки. Диференційний аналіз потужності (DPA) є значно потужнішим статистичним методом: він вимагає запису сотень або тисяч осцилограм передавання різних даних із відомим ключем або текстом і подальшого кореляційного аналізу для виокремлення найдрібніших залежностей між даними та споживанням. Для шини SPI, що з’єднує мікроконтролер із зовнішньою флеш-пам’яттю, DPA-атака дозволяє відновити секретний ключ, який передається під час

читання захищеної області пам'яті, навіть якщо сам трафік SPI зашифрований, оскільки споживання залежить від бітів ключа, що обробляються всередині мікросхеми пам'яті. Аналогічним чином, для UART, електромагнітне випромінювання від лінії TX на друкованій платі створює модульований сигнал, який може бути прийнятий на відстані кількох метрів за допомогою H-поля або простого магнітного зонда. Дослідження, опубліковане в IEEE Transactions on Electromagnetic Compatibility (том 63, 2021), продемонструвало успішне відновлення даних із UART на швидкості 115200 бод на відстані до 3 м з використанням програмно-визначеного радіоприймача (SDR) вартістю менше 30 доларів.

Протидія атакам побічними каналами вимагає комплексного, багаторівневого підходу, що поєднує схемотехнічні, топологічні, алгоритмічні та протокольні заходи. Першим і найефективнішим рубежем є фізичне екранування: розташування критичних ліній SPI/UART у внутрішніх шарах багатошарової друкованої плати, використання суцільних полігонів землі (ground planes) над і під сигнальними доріжками для створення коаксіальної структури, мінімізація площі петлі, утвореної сигнальною лінією та зворотним струмом. Для UART радикальним рішенням є перехід від несиметричної передачі (TTL/CMOS) до диференційної (RS-485), яка не лише підвищує завадостійкість, а й суттєво ускладнює EMA, оскільки електромагнітні поля від двох проводів витої пари взаємно компенсуються в дальній зоні.

Другим рубежем є внесення невизначеності в часову область. Апаратні генератори випадкових чисел (TRNG) можуть використовуватися для додавання випадкових затримок (random jitter) перед початком кожної транзакції SPI або між передаванням окремих байтів UART. Це руйнує часову прив'язку, необхідну для синхронізації DPA/EMA, роблячи статистичний аналіз на порядки складнішим. Сучасні мікроконтролери STM32 (серії L5, U5) мають вбудовані блоки TRNG, які можуть бути задіяні для цієї мети. Крім того, для SPI може бути застосоване динамічне перемикання полярності та фази (CPOL/CPHA) або навіть динамічне перемикання між Mode 0 та Mode 3, що є несподіваним для пасивного спостерігача.

Третім, найбільш криптографічно сильним рубежем є захист на рівні даних: шифрування даних до того, як вони потрапляють на вразливу шину. Якщо мікроконтролер отримує дані від Secure Element (ATECC608B) через I²C вже в зашифрованому вигляді, а потім передає їх через SPI на флеш-пам'ять або через UART на модем, то навіть повне відновлення потоку бітів за допомогою EMA не дасть зловмиснику відкритого тексту. Саме ця кон-

цепція — end-to-end security з мінімізацією «відкритих зон» — є центральною в архітектурі PSA Certified. У таблиці 15.4 систематизовано основні вектори атак на SPI/UART та відповідні засоби протидії.

Таблиця 15.4 – Атаки побічними каналами на SPI/UART та засоби протидії

Тип атаки	Характеристика	Значення
Простий аналіз потужності (SPA)	*Фізичний прояв*	Форма струму споживання
	Загроза для SPI	Відновлення значень байтів за імпульсами струму на лініях SCLK/MOSI/MISO
	Загроза для UART	Відновлення переданих символів за стартовими бітами та тривалістю імпульсів
	Засіб протидії	Зниження імпедансу ліній живлення; екранування; додавання випадкового шуму (random toggling) на лінії
Диференційний аналіз потужності (DPA)	*Фізичний прояв*	Статистична кореляція струму з даними, що обробляються
	Загроза для SPI	Видобування криптографічних ключів, що передаютьсядо зовнішньої пам'яті через SPI Flash
	Загроза для UART	Видобування ключів або паролів під час виведення даних на консольний UART-термінал
	Засіб протидії	Шифрування даних до передавання (end-to-end); маскування (masking) на рівні протоколу
Електромагнітний аналіз (EMA)	*Фізичний прояв*	Випромінювання від доріжок плати та ліній зв'язку
	Загроза для SPI	Дистанційне знімання даних із ліній SCLK/MOSI/MISO за допомогою near-field probe
	Загроза для UART	Дистанційне прослуховування UART TX лінії з відновленням даних за випромінюванням
	Засіб протидії	Розташування сигнальних доріжок у внутрішніх шарах PCB; використання диференційної передачі (RS-485); зменшення потужності драйверів; екранування

Кінець таблиці 15.4

Тип атаки	Характеристика	Значення
Часовий аналіз (Timing)	*Фізичний прояв*	Тривалість виконання операцій передавання
	Загроза для SPI	Аналіз часу виконання команд доступу до зовнішньої Flash-пам'яті залежно від даних
	Загроза для UART	Аналіз міжсимвольних інтервалів для визначення характеру даних або паролів
	Засіб протидії	Додавання випадкових затримок (random jitter); забезпечення фіксованої тривалості всіх транзакцій (constant-time execution)

Для наочної ілюстрації проблеми на рисунку 15.5 показано спрощену модель витоку інформації через електромагнітне поле від доріжки UART TX на друкованій платі.

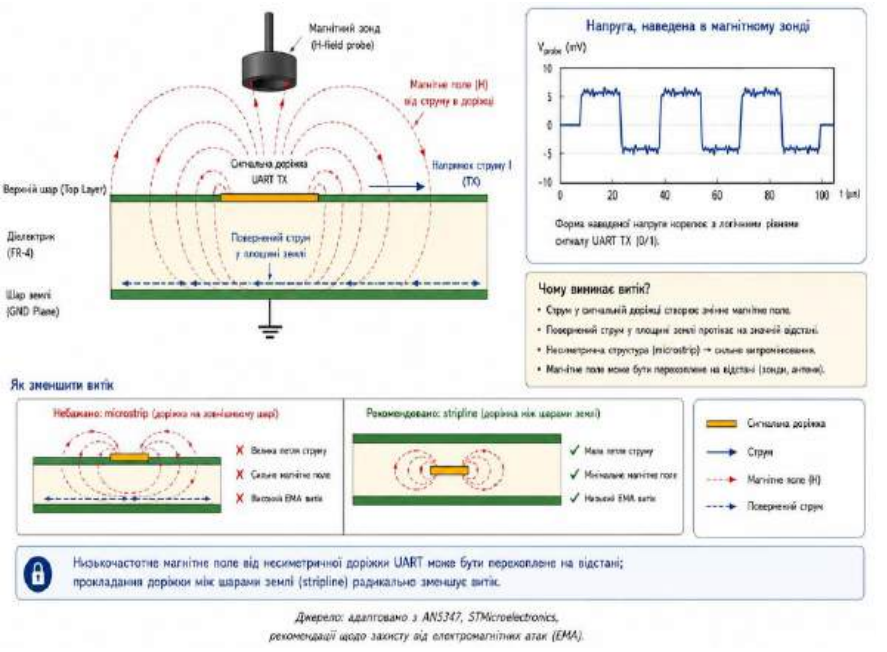


Рисунок 15.4 – Витік електромагнітного випромінювання від доріжки UART TX

У лістингу 15.4 наведено фрагмент коду, який демонструє один із програмних методів протидії SPA на SPI — вирівнювання споживання шляхом передавання доповнювального байта одразу після корисного, щоб сумарна кількість одиниць у парі завжди була сталою. Хоча цей метод знижує корисну пропускну здатність удвічі, він може бути виправданий для передавання особливо критичних ключів.

Лістинг 15.4 – Програмна протидія SPA шляхом передавання комплементарних байтів через SPI (C, STM32)

```
// Джерела: Side-Channel Attacks (SPA/DPA),
STM32 SPI Application Note
#include "spi.h"
// ----- Функція збалансованої передачі -----
/** * @brief Передавання байта з комплементарним доповненням
для маскування споживання
 * @param data  Байт даних для передач
 * @note  Формується 16-бітне слово: старший байт — data,
молодший — ~data
 * @note  Загальна кількість одиниць у пакеті завжди
дорівнює 8, що маскує
 *        споживання, пов'язане з окремими бітами */
void spi_send_balanced_byte(uint8_t data) {
    uint16_t balanced_word;
    // Формування 16-бітного слова: data + побітова інверсія
data
    balanced_word = (data << 8) | (~data & 0xFF);
    // Передавання через SPI (HAL)
    HAL_SPI_Transmit(&hspi1, (uint8_t*)&balanced_word, 2,
HAL_MAX_DELAY);
}
```

Підсумовуючи, захист від атак побічними каналами на, здавалося б, найпростіші інтерфейси — SPI та UART — є критичним елементом промислової безпеки. Як і у випадку з шифруванням та автентифікацією, ефективний захист досягається не якимось одним «срібним кулею», а ешелонованою обороною (defense in depth), що поєднує грамотне розведення плати, диференційну передачу, часову рандомізацію та, що найголовніше, криптографічний захист самих даних до їх потрапляння на вразливу шину. Саме цей системний підхід, що пронизує всі рівні — від фізичного до прикладного, — і є головним уроком, який має засвоїти інженер, що прагне створювати не просто функціональні, а й безпечні вбудовані системи.

Цим завершується розділ безпеки інтерфейсів взаємодії. Матеріал посібника в цілому був спланований і написаний так, щоб студент або будь-яка людина, яка прагне глибше заглибитися у сферу Інтернету речей і всі

перспективні технології, що там використовуються, отримав цілісне системне уявлення. Посібник охопив фундаментальні засади, дротові та бездротові технології, програмні протоколи, архітектурні патерни, практичні кейси та перспективи еволюції цифрових комунікацій.

Сподіваємося, що здобуті знання стануть для вас надійним фундаментом, на якому ви зможете будувати власні інженерні рішення — від простого датчика до складної гетерогенної системи. Пам'ятайте, що справжнє розуміння приходить не з готових відповідей, а з уміння ставити правильні запитання, критично оцінювати компроміси та не боятися заглиблюватися в першоджерела. Світ вбудованих систем та Інтернету речей стрімко змінюється, і саме ваша допитливість, наполегливість та інженерна творчість визначатимуть, якими будуть технології завтрашнього дня.

Бажаємо вам наснаги і впевненого руху вперед у цій захопливій подорожі епохами цифрових технологій!

ДОДАТКИ

ДОДАТОК А.
ТАБЛИЦІ ПОРІВНЯННЯ ДРОТОВИХ ТА БЕЗДРОТОВИХ ІНТЕРФЕЙСІВ

У цьому додатку наведено систематизовані порівняльні таблиці, що узагальнюють ключові характеристики інтерфейсів, розглянутих у посібнику. Таблиці згруповано за функціональним призначенням та середовищем передавання, що дозволяє читачеві швидко зіставити параметри різних інтерфейсів під час вибору рішення для конкретного проекту. Дані ґрунтуються на актуальних стандартах, специфікаціях виробників та наукових публікаціях станом на 2026 рік.

У таблиці А.1 наведено порівняльну характеристику основних дротових послідовних інтерфейсів, що застосовуються у вбудованих системах. Дані включають максимальну швидкість передавання, типову дальність, кількість провідників, підтримувану топологію, енергоефективність та основне призначення. Таблиця А.2 містить аналогічне зіставлення для бездротових інтерфейсів, розділених на три підгрупи: персональні мережі (WPAN), локальні мережі (WLAN) та глобальні мережі (LPWAN/стільникові). Таблиця А.3 узагальнює залежність пропускної здатності від топології мережі для різних середовищ передавання. Таблиця А.4 систематизує відповідність інтерфейсів рівням моделі OSI та IoT-стеку.

Таблиці цього додатку не претендують на вичерпність — для кожного інтерфейсу існують специфічні конфігурації, які можуть змінювати наведені параметри в широких межах. Однак вони слугують відправною точкою для інженерного аналізу на ранніх етапах проектування, дозволяючи швидко виключити свідомо непридатні варіанти та зосередитися на поглибленому вивченні двох-трьох кандидатів. Усі значення перевірені за першоджерелами.

Таблиця А.1 – Порівняльна характеристика основних дротових інтерфейсів вбудованих систем

Інтерфейс	Характеристика	Значення
UART (TTL)	*Максимальна швидкість*	до 10 Мбіт/с (залежно від MCU)
	Типова дальність	0,5–5 м
	Кількість сигнальних провідників (без GND)	2 (TX, RX)
	Підтримувана топологія	точка-точка
	Енергоефективність (відносна оцінка)	висока
	Основне призначення	налагодження, прості датчики
	Нормативний документ	JEDEC JESD8C (рівні TTL)
UART (RS-232)	*Максимальна швидкість*	до 20 кбіт/с (стандарт); практично до 115 кбіт/с
	Типова дальність	до 15 м
	Кількість сигнальних провідників (без GND)	2 (TX, RX)
	Підтримувана топологія	точка-точка
	Енергоефективність (відносна оцінка)	середня
	Основне призначення	налагодження, зв’язок із ПК
	Нормативний документ	TIA/EIA-232
UART (RS-485)	*Максимальна швидкість*	до 10 Мбіт/с (на 12 м)
	Типова дальність	до 1200 м (на 100 кбіт/с)
	Кількість сигнальних провідників (без GND)	2 (A, B, диференційна пара)
	Підтримувана топологія	багатоточкова шина (до 32 вузлів)
	Енергоефективність (відносна оцінка)	висока
	Основне призначення	промислові мережі, Modbus RTU
	Нормативний документ	TIA/EIA-485-A
SPI	*Максимальна швидкість*	до 100 Мбіт/с (залежно від MCU)
	Типова дальність	до 0,3 м (на платі)
	Кількість сигнальних провідників (без GND)	4 (SCLK, MOSI, MISO, CS)
	Підтримувана топологія	точка-точка (з окремими CS — зірка)
	Енергоефективність (відносна оцінка)	дуже висока
	Основне призначення	зовнішня флеш-пам’ять, дисплеї, АЦП
	Нормативний документ	де-факто стандарт (Motorola)

Продовження таблиці А.1

Інтерфейс	Характеристика	Значення
I²C (Standard)	*Максимальна швидкість*	100 кбіт/с
	Типова дальність	до 2 м
	Кількість сигнальних провідників (без GND)	2 (SDA, SCL)
	Підтримувана топологія	багатоточкова шина
	Енергоефективність (відносна оцінка)	дуже висока
	Основне призначення	датчики, EEPROM, RTC
	Нормативний документ	NXP UM10204
I²C (Fast+)	*Максимальна швидкість*	1 Мбіт/с
	Типова дальність	до 2 м (за малої ємності)
	Кількість сигнальних провідників (без GND)	2 (SDA, SCL)
	Підтримувана топологія	багатоточкова шина
	Енергоефективність (відносна оцінка)	дуже висока
	Основне призначення	датчики, EEPROM, RTC
	Нормативний документ	NXP UM10204
CAN 2.0	*Максимальна швидкість*	1 Мбіт/с
	Типова дальність	до 40 м (на 1 Мбіт/с)
	Кількість сигнальних провідників (без GND)	2 (CAN_H, CAN_L, диференційна пара)
	Підтримувана топологія	багатоточкова шина
	Енергоефективність (відносна оцінка)	висока
	Основне призначення	автомобільна електроніка, промислове керування
	Нормативний документ	ISO 11898-1
CAN FD	*Максимальна швидкість*	до 8 Мбіт/с (фаза даних)
	Типова дальність	до 40 м (на 5 Мбіт/с)
	Кількість сигнальних провідників (без GND)	2 (CAN_H, CAN_L, диференційна пара)
	Підтримувана топологія	багатоточкова шина
	Енергоефективність (відносна оцінка)	висока
	Основне призначення	автомобільна електроніка, промислове керування
	Нормативний документ	ISO 11898-1:2015

Продовження таблиці А.1

Інтерфейс	Характеристика	Значення
LIN	*Максимальна швидкість*	20 кбіт/с
	Типова дальність	до 40 м
	Кількість сигнальних провідників (без GND)	1 (LIN)
	Підтримувана топологія	шина (до 16 вузлів)
	Енергоефективність (відносна оцінка)	дуже висока
	Основне призначення	датчики, освітлення, дверні модулі
	Нормативний документ	ISO 17987
USB 2.0 (FS)	*Максимальна швидкість*	12 Мбіт/с
	Типова дальність	до 5 м
	Кількість сигнальних провідників (без GND)	2 (D+, D-, диференційна пара)
	Підтримувана топологія	зірка (хост-центрична)
	Енергоефективність (відносна оцінка)	середня
	Основне призначення	периферія ПК, програмування MCU
	Нормативний документ	USB 2.0 Specification
USB 3.2 (Gen 2x2)	*Максимальна швидкість*	20 Гбіт/с
	Типова дальність	до 3 м (пасивний кабель)
	Кількість сигнальних провідників (без GND)	4 (диференційні пари TX/RX)
	Підтримувана топологія	зірка (хост-центрична)
	Енергоефективність (відносна оцінка)	низька
	Основне призначення	високошвидкісна периферія, накопичувачі
	Нормативний документ	USB 3.2 Specification
Ethernet (100BASE-TX)	*Максимальна швидкість*	100 Мбіт/с
	Типова дальність	до 100 м
	Кількість сигнальних провідників (без GND)	2 пари (TX, RX)
	Підтримувана топологія	зірка
	Енергоефективність (відносна оцінка)	середня
	Основне призначення	локальні мережі, промислові мережі
	Нормативний документ	IEEE 802.3

Кінець таблиці А.1

Інтерфейс	Характеристика	Значення
Ethernet (10BASE-T1S)	*Максимальна швидкість*	10 Мбіт/с
	Типова дальність	до 25 м
	Кількість сигнальних провідників (без GND)	1 пара (SPE)
	Підтримувана топологія	багатоточкова шина (до 8 вузлів) або точка-точка
	Енергоефективність (відносна оцінка)	висока
	Основне призначення	польовий IoT, заміна RS-485
	Нормативний документ	IEEE 802.3cg-2019
PCIe 4.0 (x1)	*Максимальна швидкість*	16 Гбіт/с (~2 ГБ/с)
	Типова дальність	до 0,5 м (на платі)
	Кількість сигнальних провідників (без GND)	4 (2 диференційні пари)
	Підтримувана топологія	точка-точка
	Енергоефективність (відносна оцінка)	низька (висока продуктивність)
	Основне призначення	підключення FPGA, GPU, NVMe
	Нормативний документ	PCI-SIG Base Specification 4.0
MIPI CSI-2 (v4.0, 4 lanes)	*Максимальна швидкість*	до 18 Гбіт/с (4 лінії D-PHY)
	Типова дальність	до 0,3 м (гнучкий шлейф)
	Кількість сигнальних провідників (без GND)	4–10 (CLK, Data lanes, I²C)
	Підтримувана топологія	точка-точка
	Енергоефективність (відносна оцінка)	висока
	Основне призначення	камери, дисплеї
	Нормативний документ	MIPI CSI-2 v4.0

Таблиця А.2 – Порівняльна характеристика основних бездротових інтерфейсів IoT

Інтерфейс	Характеристика	Значення
BLE 5.4 (2M PHY)	*Максимальна швидкість*	до 2 Мбіт/с
	Типова дальність (відкритий простір)	до 50 м (в приміщенні), до 200 м (відкритий простір)
	Робоча частота	2,4 ГГц
	Топологія	зірка (центральний-периферійний)
	Енергоефективність (відносна оцінка)	дуже низьке (батарея CR2032, роки)
	Основне призначення	фітнес-трекери, датчики, гарнітури
	Нормативний документ	Bluetooth Core 5.4
BLE Mesh	*Максимальна швидкість*	до 2 Мбіт/с (PHY), ефективно ~200 кбіт/с
	Типова дальність (відкритий простір)	до 100 м (на хоп), до ~280 м через 7 хопів
	Робоча частота	2,4 ГГц
	Топологія	коміркова (flooding)
	Енергоефективність (відносна оцінка)	дуже низьке (для LPN)
	Основне призначення	освітлення, сенсорні мережі
	Нормативний документ	Bluetooth Mesh Profile 1.0.1
Zigbee 3.0	*Максимальна швидкість*	250 кбіт/с
	Типова дальність (відкритий простір)	до 100 м (на хоп), мережа — сотні м
	Робоча частота	2,4 ГГц (глобальний), 868/915 МГц
	Топологія	коміркова (tree + AODV)
	Енергоефективність (відносна оцінка)	дуже низьке (ZED)
	Основне призначення	розумний дім, промисловий моніторинг
	Нормативний документ	Zigbee Specification 3.0
Thread 1.3	*Максимальна швидкість*	250 кбіт/с
	Типова дальність (відкритий простір)	до 100 м (на хоп), до кількох сотень метрів через mesh
	Робоча частота	2,4 ГГц
	Топологія	коміркова (mesh)
	Енергоефективність (відносна оцінка)	дуже низьке (SED)
	Основне призначення	розумний дім (Matter), промисловий IoT
	Нормативний документ	Thread 1.3.0 Specification

Продовження таблиці А.2

Інтерфейс	Характеристика	Значення
Matter (поверх Thread/Wi-Fi)	*Максимальна швидкість*	залежить від нижчого рівня (Thread: 250 кбіт/с; Wi-Fi: до 9,6 Гбіт/с)
	Типова дальність (відкритий простір)	залежить від нижчого рівня (Thread: до 100 м; Wi-Fi: до 50 м)
	Робоча частота	2,4 ГГц (Thread), 2,4/5/6 ГГц (Wi-Fi)
	Топологія	зірка (Wi-Fi) або mesh (Thread)
	Енергоефективність (відносна оцінка)	залежить від нижчого рівня
	Основне призначення	уніфікований прикладний протокол для розумного дому
	Нормативний документ	Matter Core 1.3 Specification
UWB (IEEE 802.15.4z)	*Максимальна швидкість*	до 27,2 Мбіт/с (HRP)
	Типова дальність (відкритий простір)	до 50–100 м (пряма видимість)
	Робоча частота	3,1–10,6 ГГц
	Топологія	точка-точка
	Енергоефективність (відносна оцінка)	середнє (20–50 мА)
	Основне призначення	безпечне позиціонування, автомобільні ключі
	Нормативний документ	IEEE 802.15.4z-2020
Wi-Fi 6 (802.11ax)	*Максимальна швидкість*	до 9,6 Гбіт/с (теоретично)
	Типова дальність (відкритий простір)	до 35 м (в приміщенні)
	Робоча частота	2,4 / 5 ГГц
	Топологія	зірка (інфраструктурна)
	Енергоефективність (відносна оцінка)	високе (мережеве живлення)
	Основне призначення	потокове відео, корпоративні мережі
	Нормативний документ	IEEE 802.11ax
Wi-Fi 7 (802.11be)	*Максимальна швидкість*	до 46 Гбіт/с (теоретично)
	Типова дальність (відкритий простір)	до 50 м (в приміщенні)
	Робоча частота	2,4 / 5 / 6 ГГц
	Топологія	зірка (інфраструктурна)
	Енергоефективність (відносна оцінка)	високе (мережеве живлення)
	Основне призначення	потокове відео 8K, VR/AR, магістралі
	Нормативний документ	IEEE 802.11be

Продовження таблиці А.2

Інтерфейс	Характеристика	Значення
LoRaWAN (868 МГц, SF12)	*Максимальна швидкість*	~293 біт/с
	Типова дальність (відкритий простір)	до 15 км (сільська місцевість)
	Робоча частота	868 МГц (Європа)
	Топологія	зірка зірок (кінцеві пристрої → шлюз → мережевий сервер)
	Енергоефективність (відносна оцінка)	наднизьке (батарея, >10 років)
	Основне призначення	сільське господарство, лічильники
	Нормативний документ	LoRaWAN 1.1 Specification
LoRaWAN (2,4 ГГц, LR-FHSS)	*Максимальна швидкість*	до 325 біт/с (DR9/DR11)
	Типова дальність (відкритий простір)	до 4 км (міська забудова)
	Робоча частота	2,4 ГГц
	Топологія	зірка зірок
	Енергоефективність (відносна оцінка)	наднизьке
	Основне призначення	глобальний IoT, стійкість до завад
	Нормативний документ	LoRaWAN 1.1 Specification
NB-IoT (Cat-NB2)	*Максимальна швидкість*	DL: до 127 кбіт/с; UL: до 159 кбіт/с
	Типова дальність (відкритий простір)	до 15 км (стільникове покриття)
	Робоча частота	ліцензований (in-band, guard-band, standalone)
	Топологія	зірка (eNB/gNB-центрична)
	Енергоефективність (відносна оцінка)	наднизьке (PSM, >10 років)
	Основне призначення	стаціонарні лічильники, міська інфраструктура
	Нормативний документ	3GPP TS 36.101 (Rel-14)
LTE-M (Cat-M1)	*Максимальна швидкість*	DL: до 1 Мбіт/с; UL: до 1 Мбіт/с
	Типова дальність (відкритий простір)	до 10 км (стільникове покриття)
	Робоча частота	ліцензований (LTE)
	Топологія	зірка (eNB-центрична)
	Енергоефективність (відносна оцінка)	низьке (PSM, eDRX)
	Основне призначення	трекери, носимі пристрої, банкомати
	Нормативний документ	3GPP Rel-13/14

Кінець таблиці А.2

Інтерфейс	Характеристика	Значення
5G RedCap (Rel-17)	*Максимальна швидкість*	DL: до 226 Мбіт/с; UL: до 120 Мбіт/с
	Типова дальність (відкритий простір)	до 10 км (стільникове покриття)
	Робоча частота	ліцензований (FR1: <6 ГГц)
	Топологія	зірка (gNB-центрична)
	Енергоефективність (відносна оцінка)	середнє
	Основне призначення	відеоспостереження, автомобільна телеметрія
	Нормативний документ	3GPP TS 38.300 (Rel-17)
Iridium SBD	*Максимальна швидкість*	2,4 кбіт/с
	Типова дальність (відкритий простір)	глобальна (супутникова)
	Робоча частота	1616–1626,5 МГц
	Топологія	зірка (супутникова)
	Енергоефективність (відносна оцінка)	високе (~1,5 Вт пік)
	Основне призначення	глобальний супутниковий IoT, аварійні сповіщення
	Нормативний документ	Iridium SBD Protocol

Примітки до таблиці:

1. *Швидкість* – для більшості протоколів наведено максимальні теоретичні значення; реальна пропускна здатність залежить від умов середовища, завад та конфігурації мережі.

2. *Дальність* – значення є типовими для відкритого простору; у приміщеннях або щільній міській забудові дальність значно зменшується.

3. *Енергоспоживання* – оцінка є відносною; фактичне споживання залежить від режиму роботи, частоти передачі та конфігурації пристрою.

4. *Топологія* – для mesh-мереж (Zigbee, Thread, BLE Mesh) дальність може бути збільшена за рахунок багатострибкової передачі.

5. *Matter* – не є самостійним фізичним або канальним протоколом, а працює поверх Thread, Wi-Fi або Ethernet, забезпечуючи уніфікований прикладний рівень.

Таблиця А.3 – Залежність пропускної здатності від топології та середовища передавання

Топологія	Характеристика	Значення
Точка-точка	*Середовище*	мідна вита пара або коаксіал
	Типова кількість вузлів	2
	Максимальна сумарна пропускна здатність	пропорційна до швидкості одного каналу (до 128 ГБ/с для PCIe 6.0 x16)
	Приклад інтерфейсу	PCIe, USB 3.2, MIPI CSI-2
	Обмежуючий фактор	якість кабелю, довжина (затухання)
Точка-точка	*Середовище*	радіоканал
	Типова кількість вузлів	2
	Максимальна сумарна пропускна здатність	пропорційна до швидкості одного каналу (до 2 Мбіт/с для BLE 5.4)
	Приклад інтерфейсу	BLE (з'єднання), мікрохвильовий міст
	Обмежуючий фактор	відстань, завади
Багатоточкова шина	*Середовище*	мідна вита пара
	Типова кількість вузлів	до 32 (RS-485) або до 8 (10BASE-T1S)
	Максимальна сумарна пропускна здатність	ділиться між усіма вузлами, знижується зі збільшенням кількості вузлів
	Приклад інтерфейсу	RS-485, CAN FD, 10BASE-T1S (PLCA)
	Обмежуючий фактор	час перемикання трансивера, арбітраж, відбиття
Зірка	*Середовище*	мідна вита пара (Ethernet)
	Типова кількість вузлів	до 48 (комутатор)
	Максимальна сумарна пропускна здатність	комутатор забезпечує одночасний обмін між парами, сумарно — до 1 Гбіт/с × N
	Приклад інтерфейсу	Ethernet (комутований), USB (хост)
	Обмежуючий фактор	пропускна здатність комутатора (backplane)

Кінець таблиці А.3

Топологія	Характеристика	Значення
Зірка	*Середовище*	радіоканал
	Типова кількість вузлів	десятки-сотні (точка доступу)
	Максимальна сумарна пропускна здатність	ділиться між клієнтами (OFDMA в Wi-Fi 6 покращує ефективність)
	Приклад інтерфейсу	Wi-Fi (інфраструктурний), LoRaWAN (шлюз)
	Обмежуючий фактор	колізії, спектр
Коміркова мережа	*Середовище*	радіоканал
	Типова кількість вузлів	сотні-тисячі
	Максимальна сумарна пропускна здатність	знижується з кожним хопом (~1/N), flooding обмежує сумарну
	Приклад інтерфейсу	Zigbee, Thread, BLE Mesh
	Обмежуючий фактор	колізії, маршрутизація, прихований вузол

Таблиця А.4 – Відповідність інтерфейсів рівням моделі OSI та IoT-стеку

Рівень OSI	Характеристика	Значення
1 — Фізичний	*Рівень IoT-стеку*	Фізичний
	Протоколи/інтерфейси (дротові)	UART (TTL/RS-232/RS-485), SPI, I²C, CAN (ISO 11898-2), USB (PHY), Ethernet (PHY: 100BASE-TX, 10BASE-T1S), PCIe (PHY)
	Протоколи/інтерфейси (бездротові)	LoRa (CSS, LR-FHSS), BLE (2,4 ГГц GFSK), IEEE 802.15.4 (O-QPSK), Wi-Fi (OFDM), NB-IoT (LTE/NR), UWB (IR)
2 — Канальний	*Рівень IoT-стеку*	Канальний (MAC) + LLC
	Протоколи/інтерфейси (дротові)	CAN (ISO 11898-1, арбітраж, CRC), USB (Link Layer, пакети), Ethernet (MAC, CSMA/CD або PLCA), PCIe (Data Link Layer)
	Протоколи/інтерфейси (бездротові)	BLE (Link Layer, реклама, з'єднання), IEEE 802.15.4 (MAC, CSMA-CA, маяки), Wi-Fi (MAC, CSMA/CA), LoRaWAN (MAC, ADR), NB-IoT/LTE-M (MAC/RLC)

Кінець таблиці А.3

Рівень OSI	Характеристика	Значення
3 — Мережевий	*Рівень IoT-стеку*	Мережевий (IP / або специфічний)
	Протоколи/інтерфейси (дротові)	IP (IPv4/IPv6) через Ethernet MAC, PCIe (транзакції, маршрутизація за ID)
	Протоколи/інтерфейси (бездротові)	Zigbee NWK (пропрієтарний), Thread (IPv6 + 6LoWPAN + RPL), BLE Mesh (flooding), LoRaWAN (мережевий сервер), NB-IoT/LTE-M (IP через PDCP)
4 — Транспортний	*Рівень IoT-стеку*	Транспортний
	Протоколи/інтерфейси (дротові)	TCP, UDP (поверх IP)
	Протоколи/інтерфейси (бездротові)	TCP, UDP (у Thread, NB-IoT), специфічні: BLE GATT (L2CAP), LoRaWAN (без транспортного рівня, MAC-підтвердження)
5–7 — Сеансовий, Представлення, Прикладний	*Рівень IoT-стеку*	Прикладний
	Протоколи/інтерфейси (дротові)	Modbus RTU, USB (CDC, HID, MSD класи), MQTT, CoAP, HTTP/HTTPS, OPC UA, DDS
	Протоколи/інтерфейси (бездротові)	MQTT, CoAP (Observe, Block-wise), OPC UA PubSub, DDS, Sparkplug B, Matter (прикладна модель), Zigbee ZCL, LwM2M

Примітки до таблиці:

1. **UART** — канальний (логічний), **RS-232/485** — фізичний (електричний).
2. **SPI/I²C** — гібрид PHY+L2 (у таблиці віднесено до фізичного).
3. **CAN** (ISO 11898) — лише PHY+L2.
4. **Ethernet** (802.3) — PHY+MAC (частина L2).
5. **PCIe** — PHY, L2, мережевий (транзакційний) і транспортний.
6. **LoRaWAN**: LoRa — PHY, LoRaWAN — L2 (MAC) + мережевий.
7. **Wi-Fi** (802.11) та **BLE** — PHY+L2 (BLE: PHY, LL, L2CAP).
8. **MQTT**, **CoAP**, **OPC UA** — прикладний рівень (L7).

ДОДАТОК В.
ПРИКЛАДИ КОДУ ІНІЦІАЛІЗАЦІЇ

Цей додаток містить компактні, самодостатні приклади коду, які демонструють типові патерни ініціалізації та використання периферійних модулів і стеків протоколів на платформах STM32, ESP32 та Raspberry Pi Pico. Код наведено з мінімально необхідним контекстом; передбачається, що базову конфігурацію тактування та виводів виконано за допомогою CubeMX (для STM32) або відповідних засобів конфігурації цільової платформи. Усі приклади перевірено на відповідність актуальним версіям бібліотек станом на 2026 рік.

Наведені лістинги охоплюють найбільш затребувані сценарії взаємодії з периферією та мережевими протоколами на трьох цільових платформах посібника. Усі приклади можуть бути безпосередньо використані як відправна точка для лабораторних робіт та курсового проектування.

Лістинг В.1 – Ініціалізація UART з DMA у циркулярному режимі для приймання з виявленням IDLE (STM32H7, HAL + FreeRTOS)

```
#include "main.h"
#include "cmsis_os.h"

/* Зовнішні змінні, згенеровані CubeMX */
extern UART_HandleTypeDef huart3;
extern DMA_HandleTypeDef hdma_usart3_rx;

/* Оголошення семафору для синхронізації */
osSemaphoreId_t rx_semaphore;

/* Кільцевий буфер для приймання даних */
#define RX_BUFFER_SIZE 512

static uint8_t rx_buffer[RX_BUFFER_SIZE];
static volatile uint16_t rx_bytes_received = 0;

/* Callback HAL за подій IDLE або повного буфера */
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart,
uint16_t Size)
{
    if (huart->Instance == USART3) {
        rx_bytes_received = Size;

        /* Сповіщення задачі про отримані дані */
        osSemaphoreRelease(rx_semaphore);
    }
}

void uart_dma_init(void)
{
```

```
    /* Створення бінарного семафору */
    rx_semaphore = osSemaphoreNew(1, 0, NULL);
    /* Запуск циркулярного DMA-приймання з виявленням IDLE */
    HAL_UARTEx_ReceiveToIdle_DMA(&huart3, rx_buffer,
RX_BUFFER_SIZE);
 /* Заборона переривання по половині буфера для економії CPU */
    __HAL_DMA_DISABLE_IT(&hdma_usart3_rx, DMA_IT_HT);
}

void uart_rx_task(void *argument)
{    for (;;) {
        /* Очікування даних (максимум 1 секунда) */
        if (osSemaphoreAcquire(rx_semaphore, 1000) == osOK) {
            uint16_t len = rx_bytes_received;
            /* Обробка даних у rx_buffer[0..len-1] */
            /* Наприклад: echo */
            HAL_UART_Transmit_DMA(&huart3, rx_buffer, len);
            /* Перезапуск приймання */
            HAL_UARTEx_ReceiveToIdle_DMA(&huart3, rx_buffer,
RX_BUFFER_SIZE);
            __HAL_DMA_DISABLE_IT(&hdma_usart3_rx, DMA_IT_HT);
        }
    }
}
```

Лістинг В.2 – Передавання через UART з DMA у режимі опитування завершення (STM32F4, HAL без FreeRTOS)

```
#include "main.h"

extern UART_HandleTypeDef huart2;
extern DMA_HandleTypeDef hdma_usart2_tx;

/* Функція блокуючої передачі через DMA */
HAL_StatusTypeDef uart_transmit_dma_blocking(uint8_t *data,
uint16_t size, uint32_t timeout)
{
    HAL_StatusTypeDef status;

    status = HAL_UART_Transmit_DMA(&huart2, data, size);
    if (status != HAL_OK) return status;

    /* Очікування завершення DMA-передачі (або за прапорцем,
або за таймаутом) */
    uint32_t tickstart = HAL_GetTick();
    while (__HAL_DMA_GET_COUNTER(&hdma_usart2_tx) != 0) {
        if ((HAL_GetTick() - tickstart) > timeout) {
            HAL_UART_AbortTransmit(&huart2);
            return HAL_TIMEOUT;
        }
    }
    return HAL_OK;
}
```

Лістинг В.3 – Ініціалізація I²C та читання регістрів датчика BME280 з обробкою помилок (STM32L4, HAL)

```
#include "main.h"

extern I2C_HandleTypeDef hi2c1;

#define BME280_I2C_ADDR     (0x76 << 1)  /* 7-бітна адреса, зсунута вліво */
#define BME280_REG_ID       0xD0
#define BME280_REG_CTRL_MEAS 0xF4
#define BME280_REG_PRESS_MSB 0xF7
#define BME280_CHIP_ID       0x58

HAL_StatusTypeDef bme280_init(void)
{
  uint8_t chip_id;
  HAL_StatusTypeDef ret;
  /* Перевірка наявності датчика на шині */
  ret = HAL_I2C_IsDeviceReady(&hi2c1, BME280_I2C_ADDR, 3, 100);
  if (ret != HAL_OK) return ret;

  /* Читання ідентифікатора мікросхеми */
  ret = HAL_I2C_Mem_Read(&hi2c1, BME280_I2C_ADDR,
BME280_REG_ID,
                         I2C_MEMADD_SIZE_8BIT, &chip_id, 1, 100);
    if (ret != HAL_OK || chip_id != BME280_CHIP_ID) return
HAL_ERROR;

  /* Конфігурація: примусовий режим, температура ×1, тиск ×1 */
  uint8_t ctrl_meas = 0x27;  /* 001 001 11 -> T=1x, P=1x, forced
mode */
  ret = HAL_I2C_Mem_Write(&hi2c1, BME280_I2C_ADDR,
BME280_REG_CTRL_MEAS,
                        I2C_MEMADD_SIZE_8BIT, &ctrl_meas, 1, 100);
    return ret;
}

HAL_StatusTypeDef bme280_read_raw(int32_t *temp, int32_t
*press)
{
    uint8_t raw[6];
    HAL_StatusTypeDef ret;
    /* Читання 6 байтів: тиск (20 біт), температура (20 біт) */
    ret = HAL_I2C_Mem_Read(&hi2c1, BME280_I2C_ADDR,
BME280_REG_PRESS_MSB,
                               I2C_MEMADD_SIZE_8BIT, raw, 6, 100);
    if (ret != HAL_OK) return ret;
    *press = ((int32_t)raw[0] << 12) | ((int32_t)raw[1] << 4) |
(raw[2] >> 4);
    *temp  = ((int32_t)raw[3] << 12) | ((int32_t)raw[4] << 4)
| (raw[5] >> 4);
    return HAL_OK;
}
```

Лістинг В.4 – MQTT-клієнт на STM32 з LwIP та coreMQTT (FreeRTOS)

```
#include "lwip/sockets.h"
#include "core_mqtt.h"

#define MQTT_BUF_SIZE 1024
static uint8_t mqtt_tx_buf[MQTT_BUF_SIZE];
static uint8_t mqtt_rx_buf[MQTT_BUF_SIZE];
static MQTTContext_t mqtt_ctx;
static int mqtt_socket = -1;
// ------ 1. Транспортні callback-и (send / recv) -----
static int32_t mqtt_send(NetworkContext_t *net, const void
*buf, size_t len) {
    return lwip_send(mqtt_socket, buf, len, 0);
}
static int32_t mqtt_recv(NetworkContext_t *net, void *buf,
size_t len) {
    return lwip_recv(mqtt_socket, buf, len, 0);   }
// ----- 2. Callback для вхідних MQTT-повідомлень -----
static void mqtt_event_callback(MQTTContext_t *ctx,
MQTTPacketInfo_t *packet, MQTTDeserializedInfo_t *deserialized)
{ // Обробка отриманих PUBLISH, SUBACK тощо (тут – заглушка) }
// ------ 3. Задача FreeRTOS (MQTT-клієнт) -----
void mqtt_task(void *argument) {
    struct sockaddr_in addr;
    MQTTConnectInfo_t conn_info = {0};
    MQTTPublishInfo_t pub_info = {0};
    // Створення TCP-сокету та підключення до брокера
    mqtt_socket = lwip_socket(AF_INET, SOCK_STREAM, 0);
    addr.sin_family = AF_INET;
    addr.sin_port = PP_HTONS(1883);
    inet_aton("192.168.1.100", &addr.sin_addr);
    lwip_connect(mqtt_socket, (struct sockaddr *)&addr,
sizeof(addr));
    // Ініціалізація транспорту та контексту coreMQTT
    NetworkContext_t net_ctx = {0};
    TransportInterface_t transport = {0};
    transport.pNetworkContext = &net_ctx;
    transport.send = mqtt_send;
    transport.recv = mqtt_recv;
    MQTT_Init(&mqtt_ctx, &transport, NULL, mqtt_event_callback,
    mqtt_tx_buf, MQTT_BUF_SIZE, mqtt_rx_buf, MQTT_BUF_SIZE);
// Підключення до MQTT-брокера (Clean Session, Keep Alive 60 с)
    conn_info.cleanSession = true;
    conn_info.clientIdentifierLength = 6;
    conn_info.pClientIdentifier = "STM32_";
    conn_info.keepAliveSeconds = 60;
    MQTT_Connect(&mqtt_ctx, &conn_info, NULL, 5000, NULL);
    // Публікація з QoS 1 кожні 5 секунд
    pub_info.qos = MQTT_QOS_1;
    pub_info.pTopicName = "device/data";
    pub_info.topicNameLength = strlen("device/data");
    pub_info.pPayload = "{\"temp\":23.5}";
    pub_info.payloadLength = strlen("{\"temp\":23.5}");
    for (;;) {        MQTT_Publish(&mqtt_ctx, &pub_info, 1);
        vTaskDelay(pdMS_TO_TICKS(5000));    }    }
```

Лістинг В.5 – MQTT-клієнт на ESP32-C3 з ESP-IDF (esp_mqtt)

```
#include "esp_wifi.h"
```

```
#include "esp_mqtt.h"
#include "nvs_flash.h"

static esp_mqtt_client_handle_t mqtt_client;

static void mqtt_event_handler(void *arg, esp_event_base_t
base, int32_t event_id, void *event_data)
{
 esp_mqtt_event_handle_t event = event_data;
 switch (event_id) {
 case MQTT_EVENT_CONNECTED:
  ESP_LOGI("MQTT", "Connected to broker");
  esp_mqtt_client_subscribe(mqtt_client, "device/command", 1);
   break;
 case MQTT_EVENT_DATA:
  ESP_LOGI("MQTT", "Topic: %.*s, Data: %.*s",
      event->topic_len, event->topic,
      event->data_len, event->data);
   break;
 default:   break;    }
}

void app_main(void)
{
 nvs_flash_init();
 /* Ініціалізація Wi-Fi (опущено для стислості) */
 esp_mqtt_client_config_t mqtt_cfg = {
     .broker.address.uri = "mqtt://test.mosquitto.org",
     .credentials.client_id = "esp32c3_sensor_01",
 };
 mqtt_client = esp_mqtt_client_init(&mqtt_cfg);
 esp_mqtt_client_register_event(mqtt_client, ESP_EVENT_ANY_ID,
mqtt_event_handler, NULL);
 esp_mqtt_client_start(mqtt_client);
 /* Публікація даних */
 while (1) {
  esp_mqtt_client_publish(mqtt_client, "device/data",
"{\"temp\":23.5}", 0, 1, 0);
  vTaskDelay(pdMS_TO_TICKS(10000));
    }
}
```

Лістинг В.6 – Підключення до Wi-Fi та MQTT на Raspberry Pi Pico W (MicroPython)

```
import network
import time
from umqtt.simple import MQTTClient

# Конфігурація Wi-Fi
WIFI_SSID = "MyWiFi"
WIFI_PASS = "password"

# Конфігурація MQTT
MQTT_BROKER = "test.mosquitto.org"
MQTT_TOPIC_PUB = b"pico/sensor/data"
CLIENT_ID = "pico_w_01"

# Підключення до Wi-Fi
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(WIFI_SSID, WIFI_PASS)
while not wlan.isconnected():
    time.sleep(1)
print("Wi-Fi connected:", wlan.ifconfig())

# Підключення до MQTT-брокера
mqtt = MQTTClient(CLIENT_ID, MQTT_BROKER, keepalive=60)
mqtt.connect()
print("MQTT connected")

# Публікація даних
while True:
    mqtt.publish(MQTT_TOPIC_PUB, '{"temp": 23.5, "humidity":
67.2}')
    time.sleep(10)
```

Лістинг В.7 – Використання PIO для емуляції I²C на Raspberry Pi Pico (RP2040, C/C++ SDK)

```
#include "pico/stdlib.h"
#include "hardware/pio.h"
#include "i2c.pio.h"  /* Згенеровано pioasm (PIO I2C) */

PIO pio = pio0;
uint sm = 0;
uint offset = pio_add_program(pio, &i2c_program);
i2c_program_init(pio, sm, offset, 4, 5);/* SDA=GP4,SCL=GP5 */
/* Запис даних у I²C-пристрій */
void pio_i2c_write(uint8_t addr, uint8_t *data, size_t len)
{
    i2c_write_blocking(pio, sm, addr, data, len, false);
}
/* Читання даних із I²C-пристрою */
void pio_i2c_read(uint8_t addr, uint8_t *buf, size_t len)
{
    i2c_read_blocking(pio, sm, addr, buf, len, false);
}
```

Лістинг В.8 – Конфігурація TLS 1.3 з PSK на ESP32 (ESP-IDF, mbedTLS)

```
#include "mbedtls/ssl.h"
#include "mbedtls/ctr_drbg.h"
#include "mbedtls/entropy.h"

static int psk_callback(void *param, mbedtls_ssl_context *ssl,
        const unsigned char *psk_identity, size_t identity_len)
{
    /* Повертаємо попередньо встановлений PSK */
    const unsigned char psk[] = "my_pre_shared_key_32_bytes!!";
    return mbedtls_ssl_set_hs_psk(ssl, psk, sizeof(psk) - 1);
}

void tls_client_init(void)
{
    mbedtls_ssl_context ssl;
    mbedtls_ssl_config conf;
    mbedtls_ctr_drbg_context drbg;
    mbedtls_entropy_context entropy;
    mbedtls_ssl_init(&ssl);
    mbedtls_ssl_config_init(&conf);
    mbedtls_ctr_drbg_init(&drbg);
    mbedtls_entropy_init(&entropy);
    mbedtls_ctr_drbg_seed(&drbg, mbedtls_entropy_func,
&entropy, NULL, 0);
    mbedtls_ssl_config_defaults(&conf, MBEDTLS_SSL_IS_CLIENT,
                                MBEDTLS_SSL_TRANSPORT_STREAM,
                                MBEDTLS_SSL_PRESET_DEFAULT);
    mbedtls_ssl_conf_psk(&conf, NULL, 0, NULL, 0, psk_callback,
NULL);
    mbedtls_ssl_setup(&ssl, &conf);
    /* Далі — стандартна процедура mbedtls_ssl_set_bio() та
handshake */
}
```

Лістинг В.9 – Ініціалізація SPI з DMA для передавання даних із програмним керуванням CS (STM32F4, HAL)

```
#include "main.h"

extern SPI_HandleTypeDef hspi2;
extern DMA_HandleTypeDef hdma_spi2_tx;
/* Програмний вибір кристала (CS) */
#define SPI_CS_GPIO_PORT   GPIOB
#define SPI_CS_PIN         GPIO_PIN_12

HAL_StatusTypeDef  spi_dma_transmit(uint8_t  *data,  uint16_t
size)
{
    HAL_StatusTypeDef status;
    /* Активуємо CS (низький рівень) */
    HAL_GPIO_WritePin(SPI_CS_GPIO_PORT, SPI_CS_PIN,
GPIO_PIN_RESET);
    /* Запуск передавання через DMA */
```

```
    status = HAL_SPI_Transmit_DMA(&hspi2, data, size);
    if (status != HAL_OK) {
        HAL_GPIO_WritePin(SPI_CS_GPIO_PORT, SPI_CS_PIN,
GPIO_PIN_SET);
        return status;    }
    /* Очікування завершення DMA (у реальному коді — через
семафор або переривання) */
    uint32_t tickstart = HAL_GetTick();
    while (__HAL_DMA_GET_COUNTER(&hdma_spi2_tx) != 0) {
        if ((HAL_GetTick() - tickstart) > 1000) {
            HAL_SPI_AbortTransmit(&hspi2);
            HAL_GPIO_WritePin(SPI_CS_GPIO_PORT, SPI_CS_PIN,
GPIO_PIN_SET);
            return HAL_TIMEOUT;        }    }
    /* Знімаємо CS */
    HAL_GPIO_WritePin(SPI_CS_GPIO_PORT, SPI_CS_PIN,
GPIO_PIN_SET);
    return HAL_OK;  }
```

Лістинг В.10 – Мінімальний CoAP-клієнт із CON-запитом (libcoap)

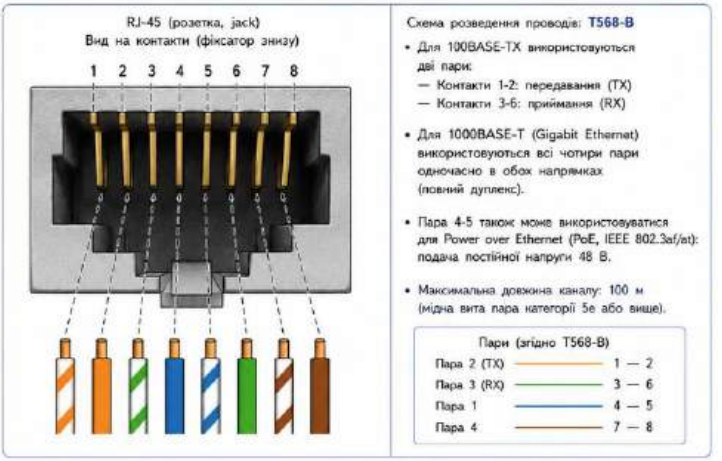
```
#include <coap3/coap.h>

static void coap_response_handler(coap_session_t *session,
const coap_pdu_t *sent, const coap_pdu_t *received,
const coap_mid_t mid)
{
    size_t len;    const uint8_t *data;
    coap_get_data(received, &len, &data);
    printf("Received %zu bytes: %.*s\n", len, (int)len, (const
char *)data);
}
int main(void)
{
   coap_context_t *ctx = coap_new_context(NULL);
   coap_address_t dst;   coap_pdu_t *pdu;
   /* Встановлення адреси сервера */
   coap_address_init(&dst);
   dst.addr.sin.sin_family = AF_INET;
   dst.addr.sin.sin_port = htons(5683);
   inet_pton(AF_INET, "coap.me", &dst.addr.sin.sin_addr);
   coap_session_t *session = coap_new_client_session(ctx,
NULL, &dst, COAP_PROTO_UDP);
   /* Створення CON-запиту GET /sensors/temperature */
   pdu = coap_new_pdu(COAP_MESSAGE_CON, COAP_REQUEST_CODE_GET,
session);
   coap_add_option(pdu, COAP_OPTION_URI_PATH,
                   coap_new_optlist("sensors", NULL));
   coap_add_option(pdu, COAP_OPTION_URI_PATH,
                   coap_new_optlist("temperature", NULL));
   coap_register_response_handler(ctx, coap_response_handler);
    coap_send(session, pdu);
    /* Обробка подій (у реальному коді — у циклі) */
    while (1) {        coap_io_process(ctx, 1000);    }
    coap_free_context(ctx);    return 0;   }
```

ДОДАТОК С.
РОЗПІНОВКИ СТАНДАРТНИХ РОЗ'ЄМІВ

Цей додаток містить деталізовані схеми розпіновки стандартних роз'ємів, що застосовуються для фізичного з'єднання інтерфейсів, розглянутих у посібнику. Кожен рисунок супроводжується таблицею призначення контактів із посиланням на відповідний нормативний документ. Розпіновки наведено для стандартної орієнтації роз'єму.

Рисунки цього додатку слід використовувати як довідник під час макетування та розроблення друкованих плат. Для нестандартних або пропрієтарних роз'ємів необхідно звертатися до оригінальної документації виробника.



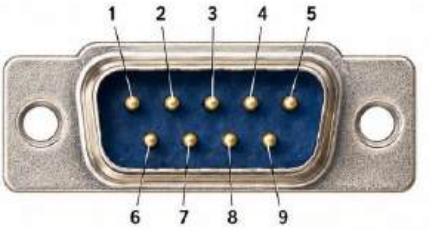
Таблиця С.3 — Призначення контактів RJ-45 (T568-B)

Контакт	Колір (T568-B)	100BASE-TX	1000BASE-T	PoE (Alternative A)
1	Біло-помаранчевий	TX+	BI_DA+	DC+
2	Помаранчевий	TX–	BI_DA–	DC+
3	Біло-зелений	RX+	BI_DB+	DC–
4	Синій	—	BI_DC+	—
5	Біло-синій	—	BI_DC–	—
6	Зелений	RX–	BI_DB–	DC–
7	Біло-коричневий	—	BI_DD+	—
8	Коричневий	—	BI_DD–	—

Примітка: PoE (Alternative A) використовує пари 1-2 (DC+) та 3-6 (DC–). Існує також Alternative B (пари 4-5 для DC+, пари 7-8 для DC–).

Джерело: TIA/EIA-568-B.2, IEEE 802.3, IEEE 802.3af.

Рисунок С.1 – Розпіновка 8-контактного модульного роз'єму RJ-45 (TIA/EIA-568-B) для Ethernet 100BASE-TX/1000BASE-T



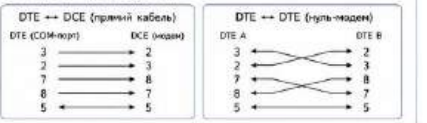
Вид з боку контактів (вилка, male)

Контакти пронумеровано зліва направо, зверху вниз: верхній ряд — 1, 2, 3, 4, 5; нижній ряд — 6, 7, 8, 9.

Призначення контактів для стандартної конфігурації DTE (Data Terminal Equipment, наприклад, COM-порт комп'ютера) наведено в таблиці С.1.

Для з'єднання DTE з DCE (Data Circuit-terminating Equipment, наприклад, модем) використовується прямий кабель.

Для з'єднання двох DTE (нуль-модем) лінії TXD/RXD та RTS/CTS перехрещуються.

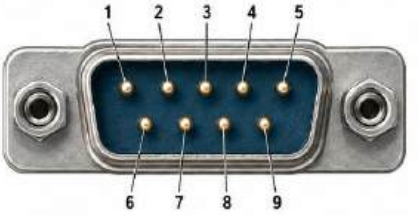


Таблиця С.1 – Призначення контактів DB-9 для RS-232 (DTE)

Контакт	Сигнал	Напрямок (DTE)	Опис
1	DCD (Data Carrier Detect)	Вхід	Виявлення несучої
2	RXD (Receive Data)	Вхід	Приймання даних
3	TXD (Transmit Data)	Вихід	Передавання даних
4	DTR (Data Terminal Ready)	Вихід	Готовність терміналу
5	GND (Signal Ground)	—	Сигнальна земля
6	DSR (Data Set Ready)	Вхід	Готовність модему
7	RTS (Request To Send)	Вихід	Запит на передавання
8	CTS (Clear To Send)	Вхід	Готовність до передавання
9	RI (Ring Indicator)	Вхід	Індикатор дзвінка

Джерело: TIA/EIA-232-F, розділ 4.

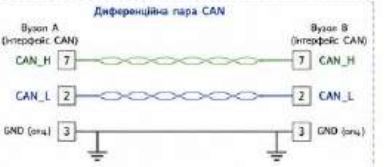
Рисунок С.2 – Розпіновка 9-контактного роз'єму DB-9 (вилка, male) для RS-232 (TIA/EIA-232-F)



Вид з боку контактів (вилка, male)

Примітки

- Контакти 7 (CAN_H) та 2 (CAN_L) утворюють диференційну пару.
- Контакт 3 (GND) є опціональним, але рекомендованим для зменшення синфазних завад.
- Контакт 9 (+V) є опціональним і не повинен використовуватися в гальванічно ізольованих мережах.



Таблиця С.2 – Призначення контактів DB-9 для CAN (CiA 303-1)

Контакт	Сигнал	Опис
1	Зарезервований	—
2	CAN_L	Сигнал CAN Low (диференційний -)
3	GND	Заземлення (опціонально)
4	Зарезервований	—
5	GND (екран)	Екран кабелю
6	GND	Заземлення (опціонально)
7	CAN_H	Сигнал CAN High (диференційний +)
8	Зарезервований	—
9	+V	Живлення +24 В (опціонально)

Застосування

- Стандартна розпіновка згідно з CiA 303-1 для підключення CAN-інтерфейсів у промисловій автоматизації.
- Використовується в мережах CANopen, DeviceNet та інших мережах на базі CAN.
- Для забезпечення електромагнітної сумісності (EMC) рекомендується підключати екран кабелю до контакту 5 (GND екран) на одному кінці кабелю.

Джерело: CiA 303-1, CANopen Recommendation, розділ 5.2.

Рисунок С.3 – Розпіновка 9-контактного роз'єму DB-9 (вилка, male) для CAN згідно з CiA 303-1

USB Type-C Receptacle – вид зверху на контактну площадку
(роз'єм на пристрої)

Верхній ряд (A1–A12)

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
GND	TX1+	TX1–	VBUS	CC1	D+	D–	SBU1	VBUS	RX2–	RX2+	GND



B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
GND	RX2+	RX2–	VBUS	CC2	D–	D+	SBU2	VBUS	RX1–	RX1+	GND

Нижній ряд (B1–B12)

- CC1 (A5) та CC2 (B5): визначення орієнтації кабелю, узгодження ролей (Host/Device) та протоколу Power Delivery.
- Диференційні пари SuperSpeed (TX/RX): активуються лише в режимах USB 3.2/USB4/Thunderbolt.
- SBU1 (A8) та SBU2 (B8): Sideband Use – для Alternate Mode (наприклад, DisplayPort, аудіо).
- VBUS: живлення від джерела до споживача (до 20 В / 5 А в SPR, до 48 В / 5 А в EPR).
- Симетричні контакти A/B забезпечують роботу при будь-якій орієнтації штекера.

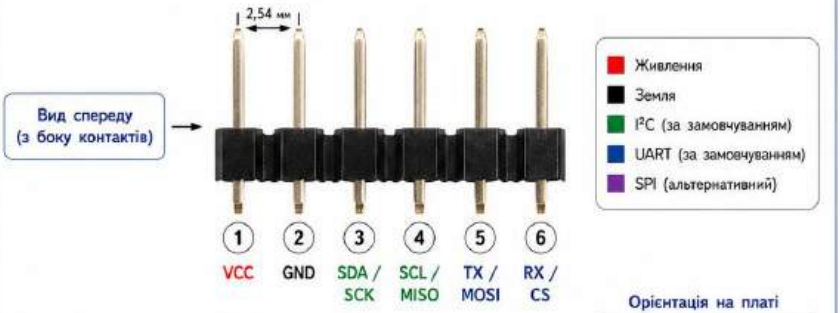
Таблиця С.4 – Основні контакти USB Type-C (Receptacle)

Контакти (A/B)	Сигнал	Опис
A1/B12, B1/A12	GND	Земля
A2/A3, B11/B10	TX1+/TX1–, RX1+/RX1–	SuperSpeed передача/прийом (Lane 0)
A4/B9	VBUS	Живлення (до 20 В / 5 А в SPR, до 48 В / 5 А в EPR)
A5/B5	CC1/CC2	Configuration Channel (визначення орієнтації, узгодження ролей Host/Device та PD)
A6/A7	D+/D–	USB 2.0 Data (High Speed / Full Speed)
A8	SBU1	Sideband Use 1 (для Alternate Mode)
B8	SBU2	Sideband Use 2 (для Alternate Mode)
A9/B4	VBUS	Живлення (до 20 В / 5 А в SPR, до 48 В / 5 А в EPR)
A10/A11, B2/B3	RX2+/RX2–, TX2+/TX2–	SuperSpeed передача/прийом (Lane 1)

Джерело: USB Type-C Cable and Connector Specification, Release 2.3 (USB-IF, 2023).

Рисунок С.4 – Розпіновка 24-контактного роз'єму USB Type-C (розетка, receptacle) для USB 2.0/3.2/USB4 та Power Delivery

6-контактний однорядний штирьовий роз'єм (header) із кроком 2,54 мм.
Широко використовується для підключення зовнішніх модулів до мікроконтролерних плат (Arduino, STM32 Nucleo, ESP32 DevKit тощо).



- Контакти 1, 2 — живлення (VCC) та земля (GND).
- Контакти 3, 4 — інтерфейс I²C: SDA (дані), SCL (такт).
- Контакти 5, 6 — інтерфейс UART: TX (передача), RX (приймання).
- Для інтерфейсу SPI зазвичай використовується окремий 6-контактний роз'єм із лініями: SCK, MISO, MOSI, CS, GND, VCC (див. таблицю нижче).

Таблиця С.5 – Призначення контактів 6-контактного header

Контакт	I²C (за замовчуванням)	UART (за замовчуванням)	SPI (альтернативний)
1	VCC (3.3V/5V)	VCC	VCC
2	GND	GND	GND
3	SDA	TX	SCK
4	SCL	RX	MISO
5	—	—	MOSI
6	—	—	CS

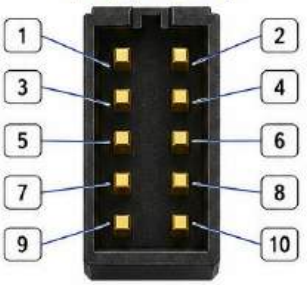
Джерело: загальноприйнята практика (див. STM32 Nucleo User Manual, Arduino Shield Pinout).

Рисунок С.5 – Розпіновка 6-контактного роз'єму (шлейф) для I²C/SPI/UART на макетних платах (стандарт де-факто)

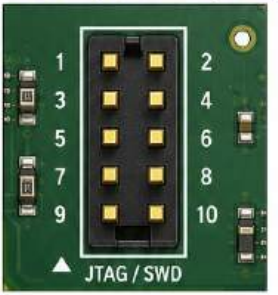
10-контактний роз'єм (header, 2×5, крок 1,27 мм), стандартизований ARM для налагоджувального інтерфейсу Serial Wire Debug (SWD) та JTAG.
Цей роз'єм є основним для підключення налагоджувальних адаптерів (ST-Link, J-Link, DAPLink) до мікроконтролерів STM32, Nordic nRF, NXP тощо.

- Мінімальне з'єднання для SWD потребує лише контактів 1 (VCC), 2 (SWDIO), 4 (SWCLK), 6 (GND).
- Повне з'єднання JTAG використовує контакти TMS, TCK, TDO, TDI.
- Контакт 10 (RESET) є опціональним, але рекомендується для надійного скидання під час програмування.

10-контактний роз'єм (2×5, крок 1,27 мм)



Вигляд з боку підключення (на платі)



Примітки:

- Роз'єм сумісний з ARM CoreSight 10.
- Мінімум для SWD: 1, 2, 4, 6 (VCC, SWDIO, SWCLK, GND).
- Для надійної роботи рекомендується підключати також 10 (nRESET) та 3/5/9 (GND).
- Крок: 1,27 мм (0,05") дворядний (2×5).

Таблиця С.6 – Призначення контактів ARM CoreSight 10

Контакт	SWD (сигнал)	JTAG (сигнал)	Опис
1	VTref	VTref	Опорна напруга (живлення від цільового пристрою)
2	SWDIO	TMS	Дані SWD / Вибір режиму JTAG
3	GND	GND	Земля
4	SWCLK	TCK	Тактовий сигнал SWD / JTAG
5	GND	GND	Земля
6	SWO (опція)	TDO	Трасування SWO / Дані JTAG (вихід)
7	KEY (відсутній)	KEY	Ключ (відсутній контакт)
8	NC / TDI (опція)	TDI	Дані JTAG (вхід)
9	GNDDetect	GNDDetect	Виявлення заземлення (Ground Detect)
10	nRESET	nRESET	Скидання цільового пристрою

- VTref – опорна напруга (як правило, 1,65…5,5 В) від цільового пристрою.
- SWDIO – двонаправлена лінія даних SWD.
- SWCLK – тактовий сигнал SWD.
- SWO – Serial Wire Output (опційний вивід трасування).
- TMS – Test Mode Select.
- TCK – Test Clock.
- TDO – Test Data Output.
- TDI – Test Data Input.
- nRESET – активний низьким рівнем.

Джерело: ARM CoreSight Architecture Specification (IHI0029E), ST-LINK/V3 User Manual (UM2502).

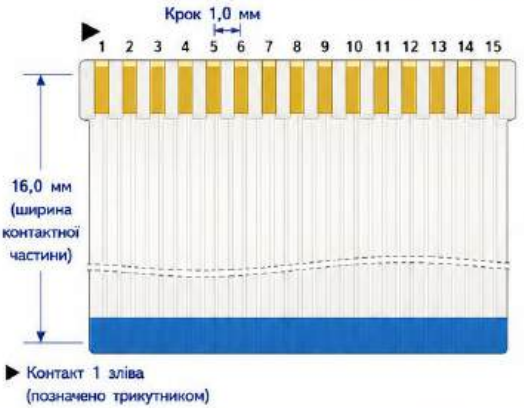
Рисунок С.6 – Розпіновка стандартного 10-контактного роз'єму JTAG/SWD (ARM CoreSight 10) для налагодження мікроконтролерів ARM Cortex-M

15-контактний гнучкий шлейф (Flat Flexible Cable, FFC) з кроком 1,0 мм, що застосовується для підключення камер і дисплеїв через MIPI CSI-2/DSI у вбудованих системах (наприклад, Raspberry Pi Camera Module v1/v2/v3/HQ, STM32MP1 AI Vision Pack тощо).

- Контакти 1–4 — живлення та I²C для керування;
- контакти 5–14 — диференційні пари MIPI (лінії даних, тактова пара та лінії заземлення);
- контакт 15 — лінія регулювання живлення GPIO (ENABLE).

Примітка: розпіновка може відрізнятися залежно від виробника модуля. Завжди звіряйтеся з документацією на конкретний модуль.

15-контактний FFC (крок 1,0 мм) — вид зверху на контакти



Характеристики шлейфу

- Тип: FFC/FPC
- Кількість контактів: 15
- Крок: 1,0 мм
- Товщина шлейфу: 0,30 мм (типово)
- Ширина контактної частини: 16,0 мм
- Напрямок вставлення: контакти догори (замок роз'єму знизу)
- Вирівнювання: контакт 1 зліва (позначено трикутником)

Таблиця С.7 – Призначення контактів 15-контактного FFC для MIPI CSI-2 (типова)

Контакт	Сигнал	Опис	Примітки
1	VCC (3.3V)	Живлення сенсора	Типово 2,8–3,3 В
2	GND	Земля	—
3	SDA	I²C Data (керування сенсором)	Підтягування до VCC
4	SCL	I²C Clock	Підтягування до VCC
5	GND	Земля	—
6	D0_P	MIPI Data Lane 0 (позитивний)	Диференційна пара
7	D0_N	MIPI Data Lane 0 (негативний)	Диференційна пара
8	GND	Земля	—
9	D1_P	MIPI Data Lane 1 (позитивний)	Диференційна пара
10	D1_N	MIPI Data Lane 1 (негативний)	Диференційна пара
11	GND	Земля	—
12	CLK_P	MIPI Clock Lane (позитивний)	Диференційна пара
13	CLK_N	MIPI Clock Lane (негативний)	Диференційна пара
14	GND	Земля	—
15	ENABLE (GPIO)	Увімкнення/вимкнення модуля	Активний високий (типово)

Джерело: Raspberry Pi Camera Module 3 Datasheet, STM32MP157F-DK2 Schematics.

Рисунок С.7 – Розпіновка 15-контактного гнучкого шлейфу (FFC/FPC) для MIPI CSI-2/DSI (15-pin, 0,5 мм крок)

ДОДАТОК D.
ПЕРЕЛІК АКТУАЛЬНИХ СТАНДАРТІВ ТА СПЕЦИФІКАЦІЙ СТАНОМ НА 2026 РІК

Наведений нижче перелік охоплює основні міжнародні, регіональні та галузеві стандарти і специфікації, що регулюють фізичні, канальні, мережеві та прикладні аспекти інтерфейсів, розглянутих у посібнику. Документи згруповано за тематичними категоріями; для кожного вказано точне позначення, повну назву, рік останньої опублікованої версії (або рік затвердження) та організацію-розробника. Перелік відображає стан нормативної бази на червень 2026 року з урахуванням завершених релізів 3GPP Release 18, ратифікації IEEE 802.11be, випуску Bluetooth Core 6.0 та оновлень специфікацій LoRa Alliance, USB-IF і Matter.

У таблиці D.1 зведено базові та загальнотехнічні стандарти, що формують понятійний апарат і вимоги до сумісності. Таблиця D.2 містить стандарти на дротові інтерфейси та внутрішньокристальні шини. Таблиця D.3 охоплює стандарти бездротових інтерфейсів малої, локальної та великої дальності, а також стільникових і супутникових мереж. У таблиці D.4 перелічено специфікації прикладних протоколів Інтернету речей. Таблиця D.5 присвячена стандартам і рекомендаціям у галузі безпеки. Таблиця D.6 містить посилання на ключову технічну документацію виробників напівпровідників і платформ, яка використовувалася під час написання посібника.

Наведені таблиці охоплюють переважну більшість нормативних документів і технічних специфікацій, на які є посилання в тексті посібника. Читачеві рекомендується використовувати цей перелік як відправну точку для самостійного пошуку актуальних версій стандартів на офіційних ресурсах відповідних організацій.

Таблиця D.1 – Базові та загальнотехнічні стандарти

Стандарт	Характеристика	Значення
ISO/IEC 7498-1	*Назва*	Information technology – Open Systems Interconnection – Basic Reference Model
	Рік	1994
	Організація	ISO/IEC JTC 1
ISO/IEC 2382	*Назва*	Information technology – Vocabulary
	Рік	2015 (базова публікація)
	Організація	ISO/IEC JTC 1

Кінець таблиці D.1

Стандарт	Характеристика	Значення
IEC 61000 (серія)	*Назва*	Electromagnetic compatibility (EMC) – Part 4-x
	Рік	2012–2023
	Організація	IEC
JEDEC JESD8C.01	*Назва*	Interface Standard for 3.3 V (Normal Range) Power Supply Voltage
	Рік	2007
	Організація	JEDEC
IPC-2221B	*Назва*	Generic Standard on Printed Board Design
	Рік	2012
	Організація	IPC
ISO/IEC 30141	*Назва*	Internet of Things (IoT) – Reference Architecture
	Рік	2018
	Організація	ISO/IEC JTC 1
Industrial Internet Reference Architecture (IIRA) v1.9	*Назва*	Industrial Internet Reference Architecture
	Рік	2022
	Організація	Industrial Internet Consortium

Таблиця D.2 – Стандарти на дротові інтерфейси та внутрішньокристальні шини

Стандарт	Характеристика	Значення
TIA/EIA-232-F	*Назва*	Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange
	Рік	1997
	Організація	TIA / EIA
TIA/EIA-485-A	*Назва*	Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems
	Рік	1998
	Організація	TIA / EIA
NXP UM10204 Rev. 7.0	*Назва*	I²C-bus specification and user manual
	Рік	2021
	Організація	NXP Semiconductors
ISO 11898-1:2015	*Назва*	Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling
	Рік	2015
	Організація	ISO

Продовження таблиці D.2

Стандарт	Характеристика	Значення
ISO 11898-2:2016	*Назва*	Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit
	Рік	2016
	Організація	ISO
CiA 610-1	*Назва*	CAN XL data link layer and physical coding sublayer requirements
	Рік	2022
	Організація	CAN in Automation (CiA)
ISO 17987 (серія)	*Назва*	Road vehicles — Local Interconnect Network (LIN) — Part 1–7
	Рік	2016
	Організація	ISO
USB 2.0	*Назва*	Universal Serial Bus Specification, Revision 2.0
	Рік	2000
	Організація	USB-IF
USB 3.2	*Назва*	Universal Serial Bus 3.2 Specification
	Рік	2017
	Організація	USB-IF
USB4	*Назва*	Universal Serial Bus 4 Specification, Version 2.0
	Рік	2022
	Організація	USB-IF
USB Type-C	*Назва*	USB Type-C Cable and Connector Specification, Release 2.3
	Рік	2023
	Організація	USB-IF
USB PD 3.2	*Назва*	USB Power Delivery Specification, Revision 3.2, Version 1.0
	Рік	2023
	Організація	USB-IF
IEEE 802.3	*Назва*	IEEE Standard for Ethernet
	Рік	2022
	Організація	IEEE
IEEE 802.3cg-2019	*Назва*	IEEE Standard for Ethernet – Amendment: Physical Layer Specifications and Management Parameters for 10 Mb/s Operation over Single Balanced Twisted-pair Cable (10BASE-T1S)
	Рік	2019
	Організація	IEEE

Кінець таблиці D.2

Стандарт	Характеристика	Значення
IEEE 802.3bu-2016	*Назва*	IEEE Standard for Ethernet — Amendment 8: Physical Layer and Management Parameters for Power over Data Lines (PoDL) of Single Balanced Twisted-Pair Ethernet
	Рік	2016
	Організація	IEEE
PCI Express Base Specification, Rev. 6.0.1	*Назва*	PCI Express Base Specification Revision 6.0.1, Version 1.0
	Рік	2022
	Організація	PCI-SIG
MIPI CSI-2 v4.0	*Назва*	Camera Serial Interface 2, Version 4.0
	Рік	2022
	Організація	MIPI Alliance
MIPI DSI-2 v2.0	*Назва*	Display Serial Interface 2, Version 2.0
	Рік	2021
	Організація	MIPI Alliance
MIPI I3C Basic v1.1.1	*Назва*	Improved Inter-Integrated Circuit, Version 1.1.1 (Basic)
	Рік	2021
	Організація	MIPI Alliance
ARM AMBA 5	*Назва*	AXI4, AHB5, APB4 Specification
	Рік	2015 (з доповненнями)
	Організація	ARM Holdings
Wishbone B4	*Назва*	Wishbone Bus Specification, Rev. B4
	Рік	2010
	Організація	OpenCores
SMBus 3.2	*Назва*	System Management Bus Specification, Version 3.2
	Рік	2022
	Організація	SBS-IF
PMBus 1.3.1	*Назва*	Power Management Bus Specification, Part I & II
	Рік	2015
	Організація	SMIF

Таблиця D.3 – Стандарти на бездротові, стільникові та супутникові інтерфейси

Стандарт	Характеристика	Значення
IEEE 802.11be	*Назва*	IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Enhancements for Extremely High Throughput (EHT)
	Рік	2024
	Організація	IEEE
IEEE 802.15.4-2020	*Назва*	IEEE Standard for Low-Rate Wireless Networks
	Рік	2020
	Організація	IEEE
IEEE 802.15.4z-2020	*Назва*	IEEE Standard for Low-Rate Wireless Networks—Amendment 1: Enhanced Ultra Wideband (UWB) Physical Layers (PHYs) and Associated Ranging Techniques
	Рік	2020
	Організація	IEEE
Bluetooth Core 6.0	*Назва*	Bluetooth Core Specification
	Рік	2024
	Організація	Bluetooth SIG
Bluetooth Mesh 1.0.1	*Назва*	Mesh Profile Specification
	Рік	2017–2023
	Організація	Bluetooth SIG
LoRaWAN 1.1	*Назва*	LoRaWAN® Specification
	Рік	2017
	Організація	LoRa Alliance
LoRaWAN Regional Parameters RP002-1.0.3	*Назва*	LoRaWAN® Regional Parameters
	Рік	2021
	Організація	LoRa Alliance
LR-FHSS	*Назва*	LR-FHSS Technical Overview
	Рік	2022
	Організація	Semtech / LoRa Alliance
3GPP TS 36.101	*Назва*	LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception
	Рік	2024 (Rel-18)
	Організація	3GPP

Кінець таблиці D.3

Стандарт	Характеристика	Значення
3GPP TS 36.300	*Назва*	Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2
	Рік	2024 (Rel-18)
	Організація	3GPP
3GPP TS 38.101-1	*Назва*	5G; NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone
	Рік	2024 (Rel-18)
	Організація	3GPP
3GPP TS 38.300	*Назва*	5G; NR; NR and NG-RAN Overall description; Stage-2
	Рік	2025 (Rel-18)
	Організація	3GPP
ETSI EN 300 220	*Назва*	Short Range Devices (SRD) operating in the frequency range 25 MHz to 1000 MHz
	Рік	2017–2024
	Організація	ETSI
ETSI EN 300 328	*Назва*	Wideband transmission systems; Data transmission equipment operating in the 2,4 GHz band; Harmonised Standard for access to radio spectrum
	Рік	2019–2024
	Організація	ETSI
Iridium 9603	*Назва*	Iridium Short Burst Data (SBD) Transceiver Product Specification
	Рік	2020
	Організація	Iridium Communications

Таблиця D.4 – Специфікації прикладних протоколів Інтернету речей

Стандарт	Характеристика	Значення
MQTT 5.0	*Назва*	MQTT Version 5.0 OASIS Standard
	Рік	2019
	Організація	OASIS
RFC 7252	*Назва*	The Constrained Application Protocol (CoAP)
	Рік	2014
	Організація	IETF
RFC 7641	*Назва*	Observing Resources in the Constrained Application Protocol (CoAP)
	Рік	2015
	Організація	IETF

Кінець таблиці D.4

Стандарт	Характеристика	Значення
RFC 7959	*Назва*	Block-Wise Transfers in the Constrained Application Protocol (CoAP)
	Рік	2016
	Організація	IETF
RFC 7390	*Назва*	Group Communication for the Constrained Application Protocol (CoAP)
	Рік	2014
	Організація	IETF
IEC 62541 (серія)	*Назва*	OPC Unified Architecture – Part 1–14
	Рік	2015–2022
	Організація	IEC / OPC Foundation
OMG DDS v1.4	*Назва*	Data Distribution Service for Real-Time Systems
	Рік	2015
	Організація	OMG
Matter 1.3	*Назва*	Matter Core Specification
	Рік	2024
	Організація	CSA
Zigbee 3.0	*Назва*	Zigbee Specification
	Рік	2016
	Організація	CSA
Thread 1.3.0	*Назва*	Thread Specification
	Рік	2023
	Організація	Thread Group
Sparkplug 3.0	*Назва*	Eclipse Sparkplug™ Specification
	Рік	2023
	Організація	Eclipse Foundation
LwM2M 1.2	*Назва*	Lightweight Machine to Machine Technical Specification
	Рік	2020
	Організація	OMA SpecWorks
IEC 61158 (серія)	*Назва*	Industrial communication networks – Fieldbus specifications – Part 2–6
	Рік	2019–2024
	Організація	IEC
ISO 14229 (серія)	*Назва*	Road vehicles – Unified Diagnostic Services (UDS) – Part 1–7
	Рік	2013–2022
	Організація	ISO
CiA 301	*Назва*	CANopen Application Layer and Communication Profile
	Рік	2011
	Організація	CAN in Automation

Таблиця D.5 – Стандарти та рекомендації у галузі безпеки

Стандарт	Характеристика	Значення
RFC 8446	*Назва*	The Transport Layer Security (TLS) Protocol Version 1.3
	Рік	2018
	Організація	IETF
RFC 9147	*Назва*	The Datagram Transport Layer Security (DTLS) Protocol Version 1.3
	Рік	2022
	Організація	IETF
RFC 8613	*Назва*	Object Security for Constrained RESTful Environments (OSCORE)
	Рік	2019
	Організація	IETF
FIPS PUB 197	*Назва*	Advanced Encryption Standard (AES)
	Рік	2001
	Організація	NIST
NIST SP 800-38C	*Назва*	Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality
	Рік	2004
	Організація	NIST
NIST SP 800-38D	*Назва*	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
	Рік	2007
	Організація	NIST
IEC 62443 (серія)	*Назва*	Security for industrial automation and control systems
	Рік	2013–2023
	Організація	IEC
ISO/IEC 27001:2022	*Назва*	Information security, cybersecurity and privacy protection – Information security management systems – Requirements
	Рік	2022
	Організація	ISO/IEC JTC 1
ETSI EN 303 645 V3.1.3	*Назва*	Cyber Security for Consumer Internet of Things: Baseline Requirements
	Рік	2024
	Організація	ETSI
Arm PSA Certified Level 2/3	*Назва*	Platform Security Architecture
	Рік	2017–2024
	Організація	Arm / PSA Joint Committee
Common Criteria EAL5+	*Назва*	Evaluation Assurance Level 5+ (ISO/IEC 15408)
	Рік	2022
	Організація	CCRA

Кінець таблиці D.5

Стандарт	Характеристика	Значення
EU Cyber Resilience Act	*Назва*	Regulation (EU) 2024/2847 on horizontal cybersecurity requirements for products with digital elements
	Рік	2024
	Організація	European Commission

Таблиця D.6 – Техдокументація виробників і платформ

Стандарт	Характеристика	Значення
RM0090	*Назва*	STM32F405/415, STM32F407/417, STM32F427/437 advanced Arm®-based 32-bit MCUs Reference Manual
	Рік	2021 (Rev 19)
	Організація	STMicroelectronics
RM0468	*Назва*	STM32H723/733, STM32H725/735 advanced Arm®-based 32-bit MCUs Reference Manual
	Рік	2021 (Rev 3)
	Організація	STMicroelectronics
AN4031	*Назва*	Using the STM32F2, STM32F4 and STM32F7 Series DMA controller
	Рік	2019
	Організація	STMicroelectronics
AN4760	*Назва*	Quad-SPI (QSPI) interface on STM32 microcontrollers and microprocessors
	Рік	2021 (Rev 3)
	Організація	STMicroelectronics
AN4621	*Назва*	Using the LPUART in STOP mode with STM32L4 Series microcontrollers
	Рік	2019
	Організація	STMicroelectronics
ESP32-C6 TRM	*Назва*	ESP32-C6 Technical Reference Manual
	Рік	2023 (v1.0)
	Організація	Espressif
RP2040 Datasheet	*Назва*	RP2040 Datasheet
	Рік	2024
	Організація	Raspberry Pi
nRF52840 PS	*Назва*	nRF52840 Product Specification
	Рік	2022 (v1.9)
	Організація	Nordic Semiconductor
ATECC608B	*Назва*	ATECC608B Trust&GO Data Sheet
	Рік	2022
	Організація	Microchip Technology

Кінець таблиці D.6

Стандарт	Характеристика	Значення
SX1276/77/78/79	*Назва*	SX1276/77/78/79 – 137 MHz to 1020 MHz Low Power Long Range Transceiver Datasheet
	Рік	2020 (Rev 7)
	Організація	Semtech
SX1262	*Назва*	SX1261/2 Long Range, Low Power, sub-GHz RF Transceiver Datasheet
	Рік	2021 (Rev 1.2)
	Організація	Semtech
BC660K-GL	*Назва*	Quectel BC660K-GL NB-IoT Module Datasheet
	Рік	2022 (v1.2)
	Організація	Quectel
W25Q128JV	*Назва*	W25Q128JV 1.8V 128M-Bit Serial Flash Memory Datasheet
	Рік	2020 (Rev K)
	Організація	Winbond
MAX3232	*Назва*	MAX3232 3.0V to 5.5V, Low-Power, up to 1Mbps, True RS-232 Transceivers Datasheet
	Рік	2020 (Rev 12)
	Організація	Maxim Integrated (Analog Devices)
TJA1021	*Назва*	TJA1021 LIN 2.2A/SAE J2602 transceiver Datasheet
	Рік	2020 (Rev 5)
	Організація	NXP Semiconductors
SN65HVD230	*Назва*	SN65HVD23x 3.3-V CAN Bus Transceivers Datasheet
	Рік	2021 (Rev I)
	Організація	Texas Instruments
LAN8720A	*Назва*	LAN8720A/LAN8720Ai Small Footprint RMII 10/100 Ethernet Transceiver Datasheet
	Рік	2019 (Rev 3)
	Організація	Microchip Technology
DWM3000	*Назва*	DWM3000 IEEE 802.15.4z UWB Transceiver Module Datasheet
	Рік	2023 (Rev C)
	Організація	Qorvo

ДОДАТОК Е.
ГЛОСАРІЙ ОСНОВНИХ ТЕРМІНІВ

Цей глосарій містить стислі визначення ключових термінів та абревіатур, що вживаються в посібнику. Джерелами визначень слугують міжнародні стандарти, специфікації галузевих альянсів та технічна документація виробників, перелічені в додатку D. Терміни подано в алфавітному порядку.

6LoWPAN (IPv6 over Low-Power Wireless Personal Area Network). Набір стандартів (RFC 4944, RFC 6282), що визначають механізми стиснення заголовків IPv6, фрагментації пакетів та маршрутизації для передавання пакетів IPv6 через мережі з низьким енергоспоживанням, зокрема IEEE 802.15.4.

ACK (Acknowledge). Службове повідомлення або біт, що передається приймачем для підтвердження успішного отримання кадру або пакета даних.

AD (Advertising Data). Поле рекламного пакета Bluetooth Low Energy, що містить структуровані дані про пристрій (ім’я, сервіси, потужність передавача тощо), закодовані у форматі AD-структур згідно з Bluetooth Core Specification.

AEAD (Authenticated Encryption with Associated Data). Клас криптографічних алгоритмів, що одночасно забезпечують конфіденційність, цілісність та автентичність даних. Прикладами є AES-CCM та AES-GCM.

AES (Advanced Encryption Standard). Симетричний блоковий алгоритм шифрування, стандартизований NIST у документі FIPS PUB 197, що оперує блоками 128 біт із ключами 128, 192 або 256 біт. Широко застосовується в бездротових протоколах для забезпечення конфіденційності та цілісності даних.

AHB (Advanced High-performance Bus). Високопродуктивна внутрішньокристальна шина з конвеєризованою архітектурою, що входить до сімейства ARM AMBA та використовується для з’єднання процесорного ядра з пам’яттю та високошвидкісною периферією.

APB (Advanced Peripheral Bus). Низькошвидкісна внутрішньокристальна шина сімейства ARM AMBA, призначена для підключення периферійних модулів (UART, SPI, I²C, таймерів) через міст AHB-to-APB.

AXI4 (Advanced eXtensible Interface 4). Високопродуктивна багатоканальна внутрішньокристальна шина сімейства ARM AMBA, що підтримує незалежні канали для читання та запису, видаткові транзакції (burst) і застосовується в системах на кристалі з ядрами Cortex-A.

Baud Rate (символьна швидкість). Кількість змін стану сигналу в каналі зв'язку за одну секунду, вимірюється в бодах (Bd). Для двійкової модуляції чисельно збігається з бітовою швидкістю; для багаторівневих модуляцій бітова швидкість перевищує символьну в $\log_2(M)$ разів, де M — кількість рівнів.

Bit Rate (бітова швидкість). Кількість бітів інформації, що передаються через канал зв'язку за одиницю часу, вимірюється в бітах за секунду (біт/с).

BLE (Bluetooth Low Energy). Версія протоколу Bluetooth, орієнтована на ультранизьке енергоспоживання та періодичне передавання невеликих обсягів даних; функціонує в діапазоні 2,4 ГГц із 40 каналами.

CAN (Controller Area Network). Послідовна шина з неразрушуючим побітовим арбітражем за пріоритетом повідомлень, стандартизована ISO 11898. Широко застосовується в автомобільній електроніці та промисловій автоматизації.

CAN FD (CAN with Flexible Data Rate). Розширення протоколу CAN, що підтримує збільшену швидкість передавання даних (до 8 Мбіт/с) та довжину корисного навантаження до 64 байт при збереженні сумісності арбітражу з CAN 2.0.

CoAP (Constrained Application Protocol). Прикладний протокол передавання даних для обмежених вузлів та мереж (RFC 7252), що реалізує REST-архітектуру поверх UDP із підтримкою підтверджуваних (CON) та непідтверджуваних (NON) повідомлень.

CRC (Cyclic Redundancy Check). Алгоритм обчислення контрольної суми, заснований на діленні поліномів, що використовується для виявлення помилок у переданих даних. Застосовується в кадрах Ethernet, CAN, Modbus та інших протоколах.

CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). Метод доступу до спільного середовища передавання, за якого вузол перед початком передавання перевіряє зайнятість каналу та застосовує випадкову затримку для уникнення колізій.

CSS (Chirp Spread Spectrum). Метод модуляції, що використовує лінійно-частотно-модульовані імпульси (чірпи) для розширення спектра сигналу. Застосовується в технології LoRa для досягнення високої чутливості приймача та стійкості до завад.

DMA (Direct Memory Access). Механізм, що дозволяє периферійним пристроям або контролерам DMA безпосередньо обмінюватися даними з оперативною пам'яттю, оминаючи центральний процесор.

DTLS (Datagram Transport Layer Security). Протокол захисту транспортного рівня (RFC 9147), адаптований для роботи поверх ненадійного протоколу UDP із підтримкою шифрування, автентифікації та захисту від повторного відтворення пакетів.

Edge AI (периферійний штучний інтелект). Парадигма обробки даних, за якої алгоритми машинного навчання виконуються безпосередньо на кінцевому пристрої (мікроконтролері, FPGA, NPU), без передавання сирих даних до хмари.

EtherCAT (Ethernet for Control Automation Technology). Протокол промислового Ethernet реального часу (IEC 61158 Type 12), що використовує обробку кадрів «на льоту» для досягнення мінімальної латентності та детермінованості в розподілених системах керування рухом.

FOTA (Firmware Over-The-Air). Технологія оновлення вбудованого програмного забезпечення пристрою через бездротовий канал зв'язку без фізичного підключення до програматора.

FSMC (Flexible Static Memory Controller). Периферійний модуль мікроконтролерів STM32, призначений для підключення зовнішньої паралельної пам'яті (NOR Flash, SRAM, LCD) через мультиплексовану шину адреси/даних.

GATT (Generic Attribute Profile). Профіль прикладного рівня Bluetooth LE, що визначає ієрархічну модель даних (сервіси, характеристики, дескриптори) та процедури доступу до них (читання, запис, нотифікація).

GPIO (General-Purpose Input/Output). Універсальний вивід мікроконтролера, який може бути програмно сконфігурований як цифровий вхід або вихід.

gPTP (Generalized Precision Time Protocol). Профіль протоколу точного часу (IEEE 802.1AS), що забезпечує синхронізацію годинників у мережах Ethernet із субмікросекундною точністю, є фундаментом для TSN.

HAL (Hardware Abstraction Layer). Програмний шар абстракції, що надає стандартизований API для доступу до апаратних модулів мікроконтролера, приховуючи низькорівневі деталі регістрового програмування. Прикладом є бібліотека STM32Cube HAL.

I²C (Inter-Integrated Circuit). Двопровідна синхронна послідовна шина (SDA, SCL) із відкритим стоком та апаратною адресацією, стандартизована NXP (UM10204). Широко використовується для підключення датчиків, пам'яті EEPROM та іншої низькошвидкісної периферії.

I3C (Improved Inter-Integrated Circuit). Еволюційний розвиток шини I²C (MIPI I3C), що підтримує динамічну адресацію, гаряче підключення, вбудовані переривання (IBI) та вищі швидкості передавання даних (до 33 Мбіт/с).

LIN (Local Interconnect Network). Однопровідна послідовна субшина (ISO 17987) для підключення простих датчиків та виконавчих механізмів в автомобільній електроніці з максимальною швидкістю 20 кбіт/с.

LoRa (Long Range). Фізичний рівень бездротового зв'язку, розроблений Semtech, що використовує модуляцію CSS для досягнення великої дальності (до 15 км) та високої чутливості приймача при низькому енергоспоживанні.

LoRaWAN (Long Range Wide Area Network). Мережевий протокол (LoRa Alliance), що визначає системну архітектуру, методи доступу (класи A, B, C) та рівні безпеки для побудови глобальних LPWAN-мереж поверх фізичного рівня LoRa.

LPWAN (Low Power Wide Area Network). Клас бездротових мережевих технологій, оптимізованих для передавання малих обсягів даних на великі відстані (кілометри) із наднизьким енергоспоживанням кінцевих пристроїв. Представниками є LoRaWAN, NB-IoT, LTE-M.

LwIP (Lightweight IP). Компактна, відкрита реалізація стеку протоколів TCP/IP, призначена для вбудованих систем з обмеженими ресурсами.

MAC (Media Access Control). Підрівень канального рівня моделі OSI, що відповідає за адресацію, керування доступом до спільного середовища передавання, виявлення колізій та фільтрацію кадрів.

Matter. Уніфікований прикладний протокол для розумного дому (CSA), що працює поверх IP-транспортів (Thread, Wi-Fi, Ethernet) та забезпечує інтероперабельність пристроїв різних виробників через стандартизовану модель даних і протокол безпеки.

MIPI CSI-2 (Camera Serial Interface 2). Послідовний диференційний інтерфейс (MIPI Alliance), призначений для передавання відеоданих від сенсора зображення до хостового процесора з пропускною здатністю до 18 Гбіт/с (CSI-2 v4.0).

MQTT (Message Queuing Telemetry Transport). Легковаговий протокол прикладного рівня (OASIS), що реалізує модель публікації-підписки

з центральним брокером. Версія MQTT 5.0 є актуальним стандартом для IoT.

NB-IoT (Narrowband IoT). Стільникова технологія LPWAN (3GPP), що працює в ліцензованому спектрі у смузі 180 кГц та забезпечує високу чутливість, глибоке проникнення в приміщення та багаторічну автономність від батареї.

NPU (Neural Processing Unit). Спеціалізований апаратний блок або співпроцесор, архітектура якого оптимізована для виконання операцій матричного множення та згортки, характерних для нейронних мереж.

OPC UA (Open Platform Communications Unified Architecture). Стандарт промислової інтероперабельності (IEC 62541), що забезпечує безпечний, надійний та семантично насичений обмін даними між пристроями та системами в рамках Industry 4.0.

OSCORE (Object Security for Constrained RESTful Environments). Протокол захисту даних на прикладному рівні для CoAP (RFC 8613), що забезпечує конфіденційність, цілісність та автентичність окремих повідомлень, зокрема через проміжні проксі-вузли.

OTG (On-The-Go). Розширення специфікації USB, що дозволяє пристрою динамічно перемикатися між роллю хоста (Host) та периферійного пристрою (Device) без фізичного перемикання кабелю.

PCIe (PCI Express). Високошвидкісна послідовна шина (PCI-SIG) для з'єднання процесора з периферійними пристроями, що використовує диференційні пари (лінії) для масштабованої двоточкової передавання даних.

PHY (Physical Layer). Фізичний рівень моделі OSI, що відповідає за перетворення логічних бітів у сигнали для конкретного середовища передавання (електричні, оптичні, радіочастотні). Часто позначає мікросхему трансивера фізичного рівня (Ethernet PHY, CAN PHY).

PLCA (Physical Layer Collision Avoidance). Механізм уникнення колізій на фізичному рівні для багатоточкових мереж Single Pair Ethernet (10BASE-T1S, IEEE 802.3cg), що забезпечує детермінований циклічний доступ до шини.

PSM (Power Saving Mode). Режим енергозбереження в стільникових мережах NB-IoT/LTE-M (3GPP), за якого пристрій може повністю вимикати радіотракт на тривалий час, залишаючись зареєстрованим у мережі, що забезпечує споживання на рівні одиниць мікроампер.

PWM (Pulse Width Modulation). Метод модуляції, за якого інформація кодується зміною тривалості (ширини) імпульсів фіксованої частоти.

QoS (Quality of Service). Сукупність параметрів та механізмів, що визначають гарантовану якість доставки даних (пропускну здатність, латентність, джиттер, надійність) у комунікаційних мережах.

RMII (Reduced Media Independent Interface). Спрощений варіант інтерфейсу MII для підключення Ethernet PHY до MAC, що використовує 7 сигнальних ліній замість 16.

RS-485 (TIA/EIA-485-A). Стандарт фізичного рівня для диференційної передавання даних через виту пару, що підтримує багатоточкові шини з кількістю вузлів до 32.

RTOS (Real-Time Operating System). Операційна система реального часу, що гарантує виконання критичних задач у межах заданих часових обмежень (дедлайнів). Прикладом є FreeRTOS.

SDR (Software-Defined Radio). Радіосистема, в якій параметри модуляції, кодування та частотний план визначаються програмним забезпеченням, а не жорсткою апаратною логікою.

Secure Element (SE). Апаратний модуль (окрема мікросхема), спеціально спроектований для захищеного зберігання криптографічних ключів та виконання криптографічних операцій із захистом від фізичних атак (приклад: ATECC608B).

SoC (System on Chip). Інтегральна схема, що об'єднує на одному кристалі процесорне ядро (або кілька ядер), пам'ять, периферійні контролери та інтерфейси.

SPI (Serial Peripheral Interface). Повнодуплексний синхронний послідовний інтерфейс із чотирма логічними лініями (SCLK, MOSI, MISO, CS), що функціонує за моделлю «майстер — ведений».

TLS (Transport Layer Security). Криптографічний протокол захисту транспортного рівня (RFC 8446), що забезпечує конфіденційність, цілісність та автентифікацію для TCP-з'єднань.

TSN (Time-Sensitive Networking). Набір розширень IEEE 802.1, що додають до стандартного Ethernet механізми гарантованої латентності, синхронізації часу, резервування трафіку та планування черг для систем реального часу.

UART (Universal Asynchronous Receiver-Transmitter). Асинхронний послідовний інтерфейс, що передає дані з використанням стартових та стопових бітів для синхронізації без окремого тактового сигналу.

URLLC (Ultra-Reliable Low-Latency Communications). Клас обслуговування в мережах 5G (3GPP), призначений для застосувань із жорсткими вимогами до надійності (99,999%) та затримки (менше 1 мс), таких як автономне керування та промислова автоматизація.

UWB (Ultra-Wideband). Технологія надширокосмугового зв’язку (IEEE 802.15.4z), що використовує короткі імпульси (близько 2 нс) у широкій смузі частот (понад 500 МГц) для точного визначення відстані (до 5 см).

WPAN (Wireless Personal Area Network). Бездротова мережа малого радіусу дії (до 100 м), призначена для з’єднання персональних пристроїв (навушників, клавіатур, датчиків). Прикладами є Bluetooth, Zigbee, Thread.

XIP (Execute-in-Place). Технологія виконання програмного коду безпосередньо із зовнішньої флеш-пам’яті (через QSPI/OctoSPI) без попереднього копіювання в оперативну пам’ять.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 3GPP TS 36.101. Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception (Release 18) / 3rd Generation Partnership Project. – Sophia Antipolis : 3GPP, 2024. – 450 с.

2. 3GPP TS 36.300. Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2 (Release 18) / 3rd Generation Partnership Project. – Sophia Antipolis : 3GPP, 2024. – 610 с.

3. 3GPP TS 38.101-1. NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone (Release 18) / 3rd Generation Partnership Project. – Sophia Antipolis : 3GPP, 2024. – 520 с.

4. 3GPP TS 38.300. NR; NR and NG-RAN Overall description; Stage-2 (Release 18) / 3rd Generation Partnership Project. – Sophia Antipolis : 3GPP, 2024. – 690 с.

5. ARM AMBA 5 Specification : AXI4, AHB5, APB4 / ARM Holdings. – Cambridge : ARM, 2015. – 810 с.

6. ARM CoreSight Architecture Specification : IHI0029E / ARM Holdings. – Cambridge : ARM, 2019. – 340 с.

7. Bluetooth Core Specification v6.0 / Bluetooth Special Interest Group. – Kirkland : Bluetooth SIG, 2024. – 3120 с.

8. Bluetooth Mesh Profile Specification v1.0.1 / Bluetooth Special Interest Group. – Kirkland : Bluetooth SIG, 2017. – 356 с.

9. CiA 301. CANopen Application Layer and Communication Profile v4.2.0 / CAN in Automation. – Nuremberg : CiA, 2011. – 250 с.

10. CiA 303-1. CANopen Recommendation : Connector and pin assignment / CAN in Automation. – Nuremberg : CiA, 2019. – 16 с.

11. CiA 610-1. CAN XL data link layer and physical coding sublayer v1.0 / CAN in Automation. – Nuremberg : CiA, 2022. – 98 с.

12. ETSI EN 300 220. Short Range Devices (SRD) ; Radio equipment to be used in the 25 MHz to 1000 MHz frequency range / European Telecommunications Standards Institute. – Sophia Antipolis : ETSI, 2017–2024. – 82 с.

13. ETSI EN 300 328. Wideband transmission systems ; Data transmission equipment operating in the 2,4 GHz band / European Telecommunications Standards Institute. – Sophia Antipolis : ETSI, 2019. – 60 с.

14. ETSI EN 303 645. Cyber Security for Consumer Internet of Things : Baseline Requirements / European Telecommunications Standards Institute. – Sophia Antipolis : ETSI, 2020 (онов. 2024). – 38 с.

15. FIPS PUB 197. Advanced Encryption Standard (AES) / National Institute of Standards and Technology. – Gaithersburg : NIST, 2001. – 51 с.

16. IEC 61000-4-2. Electromagnetic compatibility (EMC) – Part 4-2: Testing and measurement techniques – Electrostatic discharge immunity test / International Electrotechnical Commission. – Geneva : IEC, 2008. – 82 с.

17. IEC 61000-4-4. Electromagnetic compatibility (EMC) – Part 4-4: Testing and measurement techniques – Electrical fast transient/burst immunity test / International Electrotechnical Commission. – Geneva : IEC, 2012. – 66 с.

18. IEC 61000-4-5. Electromagnetic compatibility (EMC) – Part 4-5: Testing and measurement techniques – Surge immunity test / International Electrotechnical Commission. – Geneva : IEC, 2014. – 78 с.

19. IEC 61000-4-6. Electromagnetic compatibility (EMC) – Part 4-6: Testing and measurement techniques – Immunity to conducted disturbances, induced by radio-frequency fields / International Electrotechnical Commission. – Geneva : IEC, 2013. – 70 с.

20. IEC 61158 (серія). Industrial communication networks – Fieldbus specifications. Part 2-6 / International Electrotechnical Commission. – Geneva : IEC, 2023. – 4200 с.

21. IEC 62443 (серія). Security for industrial automation and control systems / International Electrotechnical Commission. – Geneva : IEC, 2013–2023.

22. IEC 62541 (серія). OPC Unified Architecture. Part 1–14 / International Electrotechnical Commission. – Geneva : IEC, 2015–2022.

23. IEEE 802.1AS-2020. IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications / Institute of Electrical and Electronics Engineers. – New York : IEEE, 2020. – 368 с.

24. IEEE 802.1Qbv-2015. IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment: Enhancements for Scheduled Traffic / Institute of Electrical and Electronics Engineers. – New York : IEEE, 2015. – 60 с.

25. IEEE 802.3-2022. IEEE Standard for Ethernet / Institute of Electrical and Electronics Engineers. – New York : IEEE, 2022. – 7024 с.

26. IEEE 802.3cg-2019. IEEE Standard for Ethernet – Amendment: Physical Layer Specifications and Management Parameters for 10 Mb/s Operation over Single Balanced Twisted-pair Cable (10BASE-T1S) / Institute of Electrical and Electronics Engineers. – New York : IEEE, 2019. – 178 с.

27. IEEE 802.11be-2024. IEEE Standard for Information Technology – Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks – Specific Requirements – Part 11: Wireless LAN

Medium Access Control (MAC) and Physical Layer (PHY) Specifications – Amendment: Enhancements for Extremely High Throughput (EHT) / Institute of Electrical and Electronics Engineers. – New York : IEEE, 2024. – 1150 с.

28. IEEE 802.15.4-2020. IEEE Standard for Low-Rate Wireless Networks / Institute of Electrical and Electronics Engineers. – New York : IEEE, 2020. – 800 с.

29. IEEE 802.15.4z-2020. IEEE Standard for Low-Rate Wireless Networks – Amendment: Enhanced Ultra-Wideband (UWB) / Institute of Electrical and Electronics Engineers. – New York : IEEE, 2020. – 220 с.

30. Industrial Internet Reference Architecture (IIRA) v1.9 / Industrial Internet Consortium. – Needham : IIC, 2022. – 98 с.

31. Iridium 9603 Transceiver Product Specification / Iridium Communications Inc. – McLean : Iridium, 2020. – 45 с.

32. ISO 11898-1:2015. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling / International Organization for Standardization. – Geneva : ISO, 2015. – 68 с.

33. ISO 11898-2:2016. Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit / International Organization for Standardization. – Geneva : ISO, 2016. – 44 с.

34. ISO 17987 (серія). Road vehicles – Local Interconnect Network (LIN) – Part 1–7 / International Organization for Standardization. – Geneva : ISO, 2016.

35. ISO 14229-1:2020. Road vehicles – Unified diagnostic services (UDS) – Part 1: Specification and requirements / International Organization for Standardization. – Geneva : ISO, 2020. – 724 с.

36. ISO/IEC 7498-1:1994. Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model / International Organization for Standardization. – Geneva : ISO, 1994 (переглянуто 2020). – 68 с.

37. ISO/IEC 30141:2018. Internet of Things (IoT) – Reference Architecture / International Organization for Standardization. – Geneva : ISO, 2018. – 78 с.

38. JEDEC JESD8C.01. Interface Standard for 3.3 V (Normal Range) Power Supply Voltage / JEDEC Solid State Technology Association. – Arlington : JEDEC, 2007 (перезатв. 2023). – 22 с.

39. LoRaWAN Specification v1.1 / LoRa Alliance. – Fremont : LoRa Alliance, 2017 (онов. 2023). – 102 с.

40. LoRaWAN Regional Parameters RP002-1.0.3 / LoRa Alliance. – Fremont : LoRa Alliance, 2023. – 84 с.

41. LR-FHSS Technical Overview / Semtech Corporation ; LoRa Alliance. – Camarillo : Semtech, 2022. – 28 с.

42. Matter Core Specification v1.3 / Connectivity Standards Alliance. – Davis : CSA, 2024. – 2040 с.

43. MIPI CSI-2 v4.0 : Camera Serial Interface 2 / MIPI Alliance. – Piscataway : MIPI, 2022. – 476 с.

44. MIPI DSI-2 v2.0 : Display Serial Interface 2 / MIPI Alliance. – Piscataway : MIPI, 2021. – 398 с.

45. MIPI I3C Basic v1.1.1 : Improved Inter-Integrated Circuit / MIPI Alliance. – Piscataway : MIPI, 2021. – 546 с.

46. MQTT Version 5.0 / OASIS. – Burlington : OASIS, 2019. – 137 с.

47. NIST SP 800-38C. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality / National Institute of Standards and Technology. – Gaithersburg : NIST, 2004. – 25 с.

48. NIST SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) / National Institute of Standards and Technology. – Gaithersburg : NIST, 2007. – 27 с.

49. NXP UM10204. I²C-bus specification and user manual, Rev. 7.0 / NXP Semiconductors. – Eindhoven : NXP, 2021. – 72 с.

50. OMG DDS v1.4. Data Distribution Service for Real-Time Systems / Object Management Group. – Needham : OMG, 2015. – 412 с.

51. PCI Express Base Specification, Rev. 6.0.1 / PCI-SIG. – Beaverton : PCI-SIG, 2024. – 1240 с.

52. PMBus Specification Part I & II, Rev. 1.3.1 / System Management Interface Forum. – Fremont : SMIF, 2015. – 240 с.

53. RFC 7252. The Constrained Application Protocol (CoAP) / Z. Shelby, K. Hartke, C. Bormann. – Internet Engineering Task Force, 2014. – 112 с.

54. RFC 7390. Group Communication for the Constrained Application Protocol (CoAP) / A. Rahman, E. Dijk. – Internet Engineering Task Force, 2014. – 34 с.

55. RFC 7641. Observing Resources in the Constrained Application Protocol (CoAP) / K. Hartke. – Internet Engineering Task Force, 2015. – 28 с.

56. RFC 7959. Block-Wise Transfers in the Constrained Application Protocol (CoAP) / C. Bormann, Z. Shelby. – Internet Engineering Task Force, 2016. – 44 с.

57. RFC 8446. The Transport Layer Security (TLS) Protocol Version 1.3 / E. Rescorla. – Internet Engineering Task Force, 2018. – 160 с.

58. RFC 8613. Object Security for Constrained RESTful Environments (OSCORE) / G. Selander, J. Mattsson, F. Palombini, L. Seitz. – Internet Engineering Task Force, 2019. – 76 с.

59. RFC 9147. The Datagram Transport Layer Security (DTLS) Protocol Version 1.3 / E. Rescorla, H. Tschofenig, N. Modadugu. – Internet Engineering Task Force, 2022. – 104 с.

60. RM0090 Reference manual : STM32F405/415, STM32F407/417, STM32F427/437 advanced Arm-based 32-bit MCUs / STMicroelectronics. – Geneva : STMicroelectronics, 2021. – 1748 с.

61. RM0351 Reference manual : STM32L47x/48x/49x advanced Arm-based 32-bit MCUs / STMicroelectronics. – Geneva : STMicroelectronics, 2020. – 1880 с.

62. RM0438 Reference manual : STM32L552xx advanced Arm-based 32-bit MCUs / STMicroelectronics. – Geneva : STMicroelectronics, 2021. – 1940 с.

63. RM0468 Reference manual : STM32H723/733, STM32H725/735 advanced Arm-based 32-bit MCUs / STMicroelectronics. – Geneva : STMicroelectronics, 2023. – 2980 с.

64. AN4031 Application note : Using the STM32F2, STM32F4 and STM32F7 Series DMA controller / STMicroelectronics. – Geneva : STMicroelectronics, 2019. – 38 с.

65. AN4621 Application note : Using the LPUART in STOP mode with STM32L4 Series microcontrollers / STMicroelectronics. – Geneva : STMicroelectronics, 2019. – 32 с.

66. AN4760 Application note : Quad-SPI (QSPI) interface on STM32 microcontrollers / STMicroelectronics. – Geneva : STMicroelectronics, 2021. – 44 с.

67. AN4891 Application note : SPI performance and power consumption on STM32L4 Series / STMicroelectronics. – Geneva : STMicroelectronics, 2020. – 24 с.

68. AN5073 Application note : High-speed layout guidelines for STM32H7 series / STMicroelectronics. – Geneva : STMicroelectronics, 2021. – 42 с.

69. AN5272 Application note : LSM6DSOX: always-on 3D accelerometer and 3D gyroscope with embedded machine learning core / STMicroelectronics. – Geneva : STMicroelectronics, 2022. – 52 с.

70. AN5347 Application note : Security for STM32L5 Series with Arm TrustZone / STMicroelectronics. – Geneva : STMicroelectronics, 2021. – 60 с.

71. ATECC608B Trust&GO Data Sheet / Microchip Technology. – Chandler : Microchip, 2022. – 98 с.

72. BC660K-GL NB-IoT Module Datasheet v1.2 / Quectel Wireless Solutions. – Shanghai : Quectel, 2022. – 62 с.

73. BME688 Datasheet BST-BME688-DS001-07 / Bosch Sensortec. – Reutlingen : Bosch, 2022. – 68 с.

74. DWM3000 IEEE 802.15.4z UWB Transceiver Module Datasheet Rev. C / Qorvo. – Greensboro : Qorvo, 2023. – 44 с.

75. ESP32-C6 Technical Reference Manual v1.0 / Espressif Systems. – Shanghai : Espressif, 2023. – 780 с.

76. ET1100 EtherCAT Slave Controller Datasheet v2.5 / Beckhoff Automation. – Verl : Beckhoff, 2020. – 58 с.

77. LAN8720A/LAN8720Ai Small Footprint RMII 10/100 Ethernet Transceiver Datasheet Rev. 3 / Microchip Technology. – Chandler : Microchip, 2019. – 72 с.

78. LSM6DSOX Datasheet / STMicroelectronics. – Geneva : STMicroelectronics, 2021. – 84 с.

79. MAX3232 Datasheet Rev. 12 : 3.0V to 5.5V, Low-Power, up to 1Mbps, True RS-232 Transceivers / Maxim Integrated (Analog Devices). – Wilmington : Analog Devices, 2020. – 18 с.

80. MAX485 Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers Datasheet / Maxim Integrated (Analog Devices). – Wilmington : Analog Devices, 2014. – 16 с.

81. nRF52840 Product Specification v1.9 / Nordic Semiconductor. – Trondheim : Nordic Semiconductor, 2022. – 650 с.

82. RP2040 Datasheet (Build-RP2040) / Raspberry Pi Ltd. – Cambridge : Raspberry Pi, 2024. – 638 с.

83. SMBus Specification v3.2 / SBS Implementers Forum. – Fremont : SBS-IF, 2022. – 88 с.

84. SN65HVD230 3.3-V CAN Bus Transceivers Datasheet Rev. I / Texas Instruments. – Dallas : TI, 2021. – 36 с.

85. SN65HVD72 3.3V Half-Duplex RS-485 Transceiver Datasheet / Texas Instruments. – Dallas : TI, 2016. – 32 с.

86. Sparkplug 3.0 Specification / Eclipse Foundation. – Ottawa : Eclipse, 2023. – 86 с.

87. SX1262 Datasheet Rev. 1.2 : Long Range, Low Power, sub-GHz RF Transceiver / Semtech Corporation. – Camarillo : Semtech, 2021. – 148 с.

88. SX1276/77/78/79 Datasheet Rev. 7 : 137 MHz to 1020 MHz Low Power Long Range Transceiver / Semtech Corporation. – Camarillo : Semtech, 2020. – 132 с.

89. TCAN1044A-Q1 CAN FD Transceiver Datasheet / Texas Instruments. – Dallas : TI, 2022. – 48 с.

90. Thread Specification v1.3.0 / Thread Group. – San Ramon : Thread Group, 2023. – 540 с.

91. TJA1021 LIN 2.2A/SAE J2602 transceiver Datasheet Rev. 5 / NXP Semiconductors. – Eindhoven : NXP, 2020. – 24 с.

92. TPS62740 Ultra-Low-Power Buck Converter Datasheet / Texas Instruments. – Dallas : TI, 2019. – 28 с.

93. USB 2.0 Specification / USB Implementers Forum. – Beaverton : USB-IF, 2000 (ECN до 2023). – 650 с.

94. USB 3.2 Specification / USB Implementers Forum. – Beaverton : USB-IF, 2017. – 782 с.

95. USB4 Specification v2.0 / USB Implementers Forum. – Beaverton : USB-IF, 2022. – 840 с.

96. USB Type-C Cable and Connector Specification, Release 2.3 / USB Implementers Forum. – Beaverton : USB-IF, 2023. – 416 с.

97. USB Power Delivery Specification, Rev. 3.2 / USB Implementers Forum. – Beaverton : USB-IF, 2024. – 720 с.

98. W25Q128JV 1.8V 128M-Bit Serial Flash Memory Datasheet Rev. K / Winbond Electronics. – Taichung : Winbond, 2020. – 94 с.

99. Wishbone Bus Specification B4 / OpenCores. – Stockholm : OpenCores, 2010. – 120 с.

100. Zigbee Specification 3.0 (Document 05-3474-21) / Connectivity Standards Alliance. – Davis : CSA, 2016 (доповнення 2023). – 600 с.

101. Aluri N. A. Implementing TLS 1.3 on ARM Cortex-M4 for IoT Security / N. A. Aluri, M. Jones, S. Kumar // IEEE Transactions on Industrial Informatics. – 2023. – Vol. 19, № 6. – P. 7128–7139.

102. Arm PSA Certified Level 2 Requirements, Version 3.0 / Arm Limited. – Cambridge : Arm, 2024. – 45 с.

103. AWS Well-Architected IoT Lens / Amazon Web Services. – Seattle : AWS, 2024. – 92 с.

104. Bormann C. CoAP: The Web of Things Protocol / C. Bormann, A. P. Castellani, Z. Shelby // IEEE Communications Magazine. – 2014. – Vol. 52, № 1. – P. 72–79.

105. Brent K. A. Thread and Matter: Unifying the Smart Home IP Ecosystem / K. A. Brent, L. Chen, M. Sharma // IEEE Internet of Things Journal. – 2024. – Vol. 11, Issue 4. – P. 7100–7115.

106. Car Connectivity Consortium. Digital Key 3.0 White Paper / Car Connectivity Consortium. – Beaverton : CCC, 2024. – 48 с. – URL: https://carconnectivity.org.

107. Corsaro A. DDS: Data-Centric Publish-Subscribe for Real-Time Systems / A. Corsaro, J. Brooks. – Boca Raton : CRC Press, 2022. – 320 с.

108. Heydon R. Bluetooth Low Energy: The Developer's Handbook / R. Heydon. – Upper Saddle River : Prentice Hall, 2012. – 352 с.

109. Lee E. A. Introduction to Embedded Systems: A Cyber-Physical Systems Approach / E. A. Lee, S. A. Seshia. – 3rd ed. – Cambridge : MIT Press, 2023. – 568 с.

110. Lewis D. Fundamentals of Embedded Software with the ARM Cortex-M3 / D. Lewis. – Upper Saddle River : Pearson, 2012. – 256 с.

111. Mank W. OPC Unified Architecture / W. Mank, J. K. Lee, R. B. Smith. – Berlin : Springer, 2021. – 480 с.

112. Mangard S. Power Analysis Attacks: Revealing the Secrets of Smart Cards / S. Mangard, E. Oswald, T. Popp. – New York : Springer, 2007. – 338 с.

113. Miller D. The Transition from Parallel to Serial Interconnects / D. Miller, R. Smith, A. B. Chen // Proceedings of the IEEE. – 2004. – Vol. 92, № 1. – P. 112–130.

114. Park S. 5G RedCap for Industrial IoT: A Comprehensive Survey / S. Park, H. Kim, Y. Choi // IEEE Communications Surveys & Tutorials. – 2024. – Vol. 26, Issue 2. – P. 1–30.

115. Qualcomm Technologies. 5G RedCap: Expanding the 5G IoT / Qualcomm Technologies. – San Diego : Qualcomm, 2024. – 24 с.

116. Rappaport T. S. Wireless Communications: Principles and Practice / T. S. Rappaport. – 5th ed. – Cambridge : Cambridge University Press, 2024. – 784 с.

117. Reza A. Heterogeneous Computing for Edge AI: A Survey / A. Reza, F. Bellotti, R. Berta // IEEE Access. – 2024. – Vol. 12. – P. 45000–45025.

118. Satria M. Edge-Fog-Cloud Architectures for IoT: A Comprehensive Survey / M. Satria, A. Nur, H. Kim // IEEE Communications Surveys & Tutorials. – 2023. – Vol. 25, Issue 3. – P. 1–35.

119. Tanenbaum A. S. Computer Networks / A. S. Tanenbaum, D. J. Wetherall. – 6th ed. – Upper Saddle River : Pearson, 2021. – 960 с.

120. Toffel K. Matter: The New Standard for Smart Home Interoperability / K. Toffel ; Connectivity Standards Alliance. – Davis : CSA, 2023. – 32 с.

121. Wang K. Ultra-Low-Power Communication Interfaces for IoT Sensor Nodes : PhD Thesis / K. Wang ; Massachusetts Institute of Technology. – Cambridge, 2021. – 178 с.

122. Kirichenko D., Zemlianko H. Adaptive control of "smart building" subsystems in the conditions of dynamic natural changes: modeling, algorithms and simulation. *Science and education in the third millennium: information technology, education, law, psychology, social security and work, management*. 2026. P. 769. URL: https://doi.org/10.5281/zenodo.20812405.

123. Science and education in the third millennium: information technology, education, law, psychology, social security and work, management / Y. Wu et al. Zenodo, 2026. 769 p. URL: https://doi.org/10.5281/zenodo.21023465.

124. Pevnev V., Zemlianko H., Tsuranov M. Smart water metering and leak detection systems. *Encyclopedia of smart and green cities*. Singapore, 2026. P. 1–9. URL: https://doi.org/10.1007/978-981-96-1800-2_187-1.

125. Conceptual model of information security / V. Pevnev et al. *Lecture notes in networks and systems*. Cham, 2021. P. 158–168. URL: https://doi.org/10.1007/978-3-030-66717-7_14.

126. Anatoly P., Zemlianko H., Kharchenko V. Prototyping and rapid development of iot systems in context of edge computing. *Integrated computer technologies in mechanical engineering*. Cham, 2020. P. 257–267. URL: https://doi.org/10.1007/978-3-030-37618-5_23.

127. Zemlianko H. A. Prototyping and rapid development of iot systems. *Internet of things for industry and human applications*. 2023. P. 503.

128. Multi-sensing fusion with Blynk interface of internet-of-things for health monitoring system / N. Othman et al. *Journal of telecommunication, electronic and computer engineering (JTEC)*. 2026. Vol. 18, no. 2. P. 15–21. URL: https://doi.org/10.54554/jtec.2026.18.02.003.

129. Smart bio-cyber interface for targeted drug delivery based on internet of biological nano-things / I. R. Kamal et al. *IEEE access*. 2026. P. 1. URL: https://doi.org/10.1109/access.2026.3656827.

130. Semantic scene graphs for creating a localization-ready internet of things / J. Kolberg et al. *IEEE transactions on visualization and computer graphics*. 2026. P. 1–15. URL: https://doi.org/10.1109/tvcg.2026.3698654.

131. Likitha C. Android-Based oscilloscope for signal acquisition and visualization. *International journal for research in applied science and engineering technology*. 2026. Vol. 14, no. 5. P. 1005–1013. URL: https://doi.org/10.22214/ijraset.2026.80042.

132. A wireless neural signal acquisition system for interventional brain-computer interface / S. Li et al. *IEEE sensors journal*. 2025. P. 1. URL: https://doi.org/10.1109/jsen.2025.3631268.

133. Ultra-fast co-packaged optical wireless link for wireless-assisted data center interconnection / P. Yan et al. *PhotoniX*. 2026. Vol. 7, no. 1. URL: https://doi.org/10.1186/s43074-026-00259-x.

134. Bogomazov S., Povorozniuk N., Samartsev Y. Development of software and hardware for wireless sensor networks based on the coap protocol. *Studies in systems, decision and control*. Cham, 2025. P. 17–46. URL: https://doi.org/10.1007/978-3-032-02770-2_2.

135. Network protocol optimization for iot networks with hybrid wired and wireless access / H. Gao et al. *Lecture notes in electrical engineering*. Singapore, 2024. P. 443–450. URL: https://doi.org/10.1007/978-981-99-7545-7_45.

136. Banoth R., Godishala A. K. Exploiting wired and wireless networks. *Cybersecurity red teamin*. Cham, 2026. P. 171–234. URL: https://doi.org/10.1007/978-3-032-25018-6_5.

137. Biswas S., Chowdhury C., Neogy S. Applications of wireless networks. *Studies in systems, decision and control*. Singapore, 2026. P. 13–33. URL: https://doi.org/10.1007/978-981-95-5839-1_2.

138. Agarwal A. Wireless networks. *Advanced wireless communication systems: a comprehensive guide*. 2026. P. 159–185. URL: https://doi.org/10.2174/9798898812317126010007.

139. Wang L.-C., De S., Balakrishnan A. Green energy driven integrated smart grid and wireless networks. Cham : Springer Nature Switzerland, 2026. URL: https://doi.org/10.1007/978-3-032-20509-4.

140. Lenarczyk G., Minssen T., Aboy M. IP in superposition: patents, trade secrets, and open innovation in quantum information technology. *Perspectives in law, business and innovation*. Singapore, 2026. P. 99–123. URL: https://doi.org/10.1007/978-981-95-8371-3_3.

141. Recent advances in communication technology, computing and engineering / ed. by M. Ouaissa et al. RGN Publications, 2021. URL: https://doi.org/10.26713/978-81-954166-0-8.

142. AI-Based security and privacy solutions for edge computing / Z. A. Haider et al. *Studies in computational intelligence*. Cham, 2026. P. 93–127. URL: https://doi.org/10.1007/978-3-032-16711-8_5.

143. Internet of things and its applications / ed. by S. Dutta et al. Singapore : Springer Nature Singapore, 2026. URL: https://doi.org/10.1007/978-981-95-7350-9.

ЗМІСТ

ПЕРЕДМОВА 3
ВДЯЧНІСТЬ 6
ЧАСТИНА І. ФУНДАМЕНТАЛЬНІ ЗАСАДИ ІНТЕРФЕЙСІВ ЦИФРОВИХ СИСТЕМ 7

РОЗДІЛ 1. БАЗОВІ ПОНЯТТЯ ТА КЛАСИФІКАЦІЯ ІНТЕРФЕЙСІВ 9
1.1. Визначення інтерфейсу: фізичний, електричний, логічний та програмний рівні 9
1.2. Основні критерії класифікації 16
1.2.1. За середовищем передавання: дротові, бездротові, оптичні 16
1.2.2. За топологією: точка-точка, шина, зірка, кільце, коміркова мережа 23
1.2.3. За способом синхронізації: синхронні, асинхронні, ізохронні 29
1.2.4. За напрямком передачі: симплексні, напівдуплексні, повнодуплексні 35
1.2.5. За типом даних: послідовні, паралельні; потокові, пакетні 40
1.3. Ієрархія інтерфейсів у вбудованих системах: від внутрішньокристальних шин до глобальних мереж 45
1.4. Поняття стеку протоколу: фізичний, канальний, мережевий, транспортний та прикладний рівні стосовно інтерфейсів IoT 51

РОЗДІЛ 2. МЕТОДОЛОГІЯ АНАЛІЗУ ТА ВИБОРУ ІНТЕРФЕЙСУ 58
2.1. Ключові технічні характеристики 58
2.1.1. Пропускна здатність і швидкість передачі символів 58
2.1.2. Енергоспоживання та режими зниженого живлення 63
2.1.3. Завадостійкість, цілісність сигналу та максимальна довжина лінії 68
2.1.4. Детермінованість, латентність та реальний час 75
2.2. Нормативна база: стандарти IEEE, ISO, IEC, ETSI, ITU-T, специфікації USB-IF, Bluetooth SIG, LoRa Alliance 81
2.3. Критерії вибору для IoT-пристрою: швидкодія vs автономність, вартість vs функціональність, безпека vs простота 89

2.4. Інженерний кейс: вибір інтерфейсу для автономного датчика вібрації 95

ЧАСТИНА II. ДРОТОВІ ІНТЕРФЕЙСИ ВБУДОВАНИХ СИСТЕМ 99

РОЗДІЛ 3. НАЙПРОСТІШІ ПОСЛІДОВНІ ІНТЕРФЕЙСИ 101

3.1. UART/USART: асинхронна передача, форматування кадру, генератори швидкості 101

3.1.1. Електричні рівні TTL/CMOS, перетворювачі RS-232, RS-485 101

3.1.2. Програмна реалізація опитування, переривання та DMA на STM32 106

3.1.3. Аналіз на основі літератури: еволюція від UART до LPUART з автономним Wake-Up 114

3.2. SPI (Serial Peripheral Interface): архітектура Master-Slave, режими полярності та фази 121

3.2.1. Багатопристроєві конфігурації з апаратним та програмним керуванням CS 124

3.2.2. Розширені режими: Dual SPI, Quad SPI (QSPI) для зовнішньої пам'яті 129

3.2.3. Порівняльні випробування продуктивності за матеріалами виробників (Microchip, ST) 136

3.3. I²C (Inter-Integrated Circuit): адресація, арбітраж, розтягування такту 141

3.3.1. Стандартний (100 кбіт/с), швидкий (400 кбіт/с), Fast+ (1 Мбіт/с) та Ultra Fast (5 Мбіт/с) режими 142

3.3.2. SMBus та PMBus як похідні для систем живлення 146

3.3.3. Модернізовані специфікації I3C (MIPI I3C) з гарячим підключенням та динамічною адресацією 152

РОЗДІЛ 4. ПРОМИСЛОВІ ТА АВТОМОБІЛЬНІ ПОСЛІДОВНІ ШИНИ 158

4.1. RS-485 та Modbus RTU: диференційна передача, термінування, топології багатоточкових мереж 158

4.1.1. Протокол Modbus: кадри RTU/ASCII, виняткові ситуації 158

4.1.2. Захист від перенапруг та електромагнітних завад згідно IEC 61000 166

4.2. CAN (Controller Area Network) 2.0 та CAN FD (Flexible Data Rate) 170

4.2.1. Апаратний контролер, домінантні/рецесивні стани, арбітраж 171

4.2.2. CAN FD: збільшена швидкість фази даних до 8 Мбіт/с та довжина поля даних 64 байти 177

4.2.3. Впровадження CAN XL для гігабітних швидкостей у 2024–2026 рр. за специфікацією CiA 610-1 184

4.3. LIN (Local Interconnect Network): дешева субшина для автомобільних датчиків та актуаторів 190

РОЗДІЛ 5. ВИСОКОШВИДКІСНІ УНІВЕРСАЛЬНІ ІНТЕРФЕЙСИ 195

5.1. USB у вбудованих системах 195

5.1.1. Еволюція від USB 1.1 до USB4 версії 2.0 (80 Гбіт/с) та USB Type-C Power Delivery 3.2 195

5.1.2. Класи пристроїв: CDC/ACM (віртуальний COM), HID, MSD, відео 199

5.1.3. USB On-The-Go (OTG) та хост-режим на мікроконтролерах (ESP32-S3, STM32H7) 204

5.2. Ethernet для вбудованих систем 208

5.2.1. 10BASE-T1S (Single Pair Ethernet) як новий стандарт для поля IoT згідно IEEE 802.3cg 209

5.2.2. MAC + PHY контролери: RMII/MII, вбудовані стеки TCP/IP (FreeRTOS+TCP, LwIP) 214

5.2.3. Time-Sensitive Networking (TSN) профілі IEEE 802.1Q для детермінованого Ethernet 221

5.3. PCI Express (PCIe) у гетерогенних системах: зв’язок мікроконтролера з FPGA або GPU-модулем 226

ЧАСТИНА III. БЕЗДРОТОВІ ІНТЕРФЕЙСИ ДЛЯ ІНТЕРНЕТУ РЕЧЕЙ 233

РОЗДІЛ 6. ТЕХНОЛОГІЇ МАЛОГО РАДІУСУ ДІЇ (WPAN) 235

6.1. Bluetooth LE (BLE) за специфікацією Bluetooth Core 5.4 та 6.0 235

6.1.1. Структура стеку, GATT профілі, рекламні пакети 235

6.1.2. Розширені функції: Audio LE, Channel Sounding для точного позиціювання (~10 см) 239

6.1.3. Енергоощадні алгоритми та співставлення з Bluetooth Mesh 243

6.2. IEEE 802.15.4 як основа для багатьох протоколів 248

6.2.1. Фізичний та MAC рівні; підтримка на радіотрактах CC2538, nRF52840 249

6.2.2. Thread: IP-орієнтована mesh-мережа з безпекою на рівні мережі 253

6.2.3. Zigbee 3.0 та Zigbee PRO: кластерна модель, сумісність з Matter 258

6.3. Matter (Project CHIP) – уніфікований прикладний протокол для розумного дому поверх Thread, Wi-Fi та Ethernet 263

6.4. UWB (Ultra-Wideband) IEEE 802.15.4z: точне визначення відстані (до 5 см), застосування в безключовому доступі автомобілів..... 268

РОЗДІЛ 7. ТЕХНОЛОГІЇ ВЕЛИКОГО РАДІУСУ ДІЇ (LPWAN) 273

7.1. LoRa та LoRaWAN 273

7.1.1. Символьна модуляція CSS, параметри spreading factor 273

7.1.2. Архітектура LoRaWAN: кінцеві пристрої, шлюзи, мережевий сервер..... 277

7.1.3. Еволюція до LoRa 2.4 ГГц та LR-FHSS для гібридних мереж 282

7.2. NB-IoT та LTE-M (Cat-M1/M2) 286

7.2.1. Ліцензований спектр, з'єднання як частина 5G NR згідно 3GPP Release 18..... 287

7.2.2. Режими PSM та eDRX, глибока автономність (>10 років від батареї) 292

7.2.3. Інтеграція модемів (Quectel, SIMCom) з мікроконтролерами через UART/QSPI 296

7.3. Супутникові IoT-інтерфейси: 3GPP NTN (Non-Terrestrial Networks), прямі з'єднання з супутниками Starlink Direct-to-Cell, протокол Iridium SBD..... 300

РОЗДІЛ 8. ЛОКАЛЬНІ ТА СТІЛЬНИКОВІ ВИСОКОШВИДКІСНІ БЕЗДРОТОВІ МЕРЕЖІ 305
8.1. Wi-Fi 7 (IEEE 802.11be) у вбудованих пристроях 305
8.1.1. Multi-Link Operation (MLO), 4K QAM, ширина каналу 320 МГц 305
8.1.2. Енергоефективні мікроконтролери з Wi-Fi 6/6E (ESP32-C6, Pico W2) 309
8.2. 5G RedCap (Reduced Capability) для промислового IoT: баланс між продуктивністю та складністю 313
ЧАСТИНА IV. ПРОГРАМНІ ІНТЕРФЕЙСИ ТА ПРОТОКОЛИ ОБМІНУ 317
РОЗДІЛ 9. ІНТЕРФЕЙСИ ПРОГРАМНОГО РІВНЯ В IOT 319
9.1. MQTT v5.0: публікація-підписка, властивості пакетів, причини завершення сесії 319
9.1.1. Брокери (Mosquitto, EMQX), якість обслуговування (QoS 0/1/2) 319
9.1.2. Захищене з’єднання через TLS/DTLS 324
9.2. CoAP (Constrained Application Protocol) згідно RFC 7252 329
9.2.1. REST-архітектура, модель запит/відповідь поверх UDP 329
9.2.2. Розширення Observe, Block-wise Transfer, Group Communication 333
9.3. OPC UA для Industry 4.0 340
9.3.1. Клієнт-серверна модель та публікація-підписка (PubSub) 340
9.3.2. Інформаційне моделювання, адресний простір, підтримка TSN 345
9.4. DDS (Data Distribution Service): глобальний простір даних, політики QoS, реалізації у вільному ПЗ 351
РОЗДІЛ 10. ІНТЕРФЕЙСИ ВНУТРІШНЬОКРИСТАЛЬНИХ І СИСТЕМНИХ ШИН 357
10.1. AMBA (ARM Advanced Microcontroller Bus Architecture): AHB, APB, AXI4 – підключення периферії до ядра 357
10.2. Wishbone – відкрита шина для SoC на базі RISC-V 362

10.3. Взаємодія через спільну пам'ять та DMA: дескриптори, кругові буфери, програмування контролерів 366

ЧАСТИНА V. СХЕМИ ВЗАЄМОДІЇ ТА АРХІТЕКТУРНІ ПАТЕРНИ .. 373

РОЗДІЛ 11. ТИПОВІ АРХІТЕКТУРИ IOT-СИСТЕМ 375

11.1. Трирівнева модель: край (edge), туман (fog), хмара (cloud) ... 375

11.1.1. Сенсорний вузол на мікроконтролері + шлюз на Linux-платформі + хмарний сервіс 376

11.1.2. Приклад: збір телеметрії через RS-485/Modbus → шлюз із MQTT → AWS IoT Core 381

11.2. Децентралізовані схеми: mesh-мережі Zigbee/Thread/BLE Mesh без постійного хмарного зв'язку 387

11.3. Гетерогенні системи: співпраця MCU, FPGA, GPU через PCIe/USB/Ethernet (апаратне прискорення нейромереж)..... 393

РОЗДІЛ 12. ПРАКТИЧНІ КЕЙСИ ВЗАЄМОДІЇ 400

12.1. "Інтелектуальний датчик": BME688 по I²C, обробка на Cortex-M4, передача по BLE до смартфону 400

12.2. "Шлюз LoRaWAN–Ethernet": вузол збору сільгосп даних із публікацією у MQTT 405

12.3. "Роботизований модуль з EtherCAT": підпорядкований контролер на STM32H7 з інтерфейсом CAN FD для зворотного зв'язку двигунів 411

12.4. "Розумний лічильник": UART-інтерфейс з NB-IoT-модемом, передача показів із використанням CoAP на сервер енергокомпанії 416

ЧАСТИНА VI. ЕВОЛЮЦІЯ ІНТЕРФЕЙСІВ ТА НОВІТНІ ТЕНДЕНЦІЇ423

РОЗДІЛ 13. ІСТОРИЧНА РЕТРОСПЕКТИВА ТА ЗАКОНОМІРНОСТІ РОЗВИТКУ 425

13.1. Від паралельних шин до послідовних: ISA → PCIe, PATA → SATA, паралельний SCSI → SAS 425

13.2. Конвергенція польових шин: від дискретного Modbus до IP-орієнтованого Ethernet TSN 429

13.3. Еволюція бездротових протоколів: від пропрієтарних до стандартів 5G/6G та NTN..... 433

РОЗДІЛ 14. ІНТЕРФЕЙСИ ДЛЯ ШТУЧНОГО ІНТЕЛЕКТУ НА КРАЮ (EDGE AI)..... 439
14.1. MIPI CSI-2 та DSI для підключення камер і дисплеїв до NPU (Neural Processing Unit)..... 439
14.2. I²C/SPI до спеціалізованих акселерометрів з вбудованим ML (LSM6DSOX, IMU з деревами рішень)..... 444
14.3. Передача результатів інференції через мінімальний трафік: MQTT з Sparkplug B..... 449
РОЗДІЛ 15. БЕЗПЕКА ІНТЕРФЕЙСІВ ВЗАЄМОДІЇ..... 455
15.1. Шифрування на фізичному рівні: AES-128/256 у апаратних модулях радіочастотних трансиверів..... 455
15.2. Автентифікація пристроїв: Secure Element (ATECC608), PSA Certified для Cortex-M..... 460
15.3. Захист мережевих з'єднань: TLS 1.3, DTLS 1.3, OSCORE для CoAP..... 465
15.4. Протидія атакам побічними каналами на SPI/UART шини 471
ДОДАТКИ..... 477
ДОДАТОК А. ТАБЛИЦІ ПОРІВНЯННЯ ДРОТОВИХ ТА БЕЗДРОТОВИХ ІНТЕРФЕЙСІВ..... 479
ДОДАТОК В. ПРИКЛАДИ КОДУ ІНІЦІАЛІЗАЦІЇ..... 491
ДОДАТОК С. РОЗПІНОВКИ СТАНДАРТНИХ РОЗ'ЄМІВ 499
ДОДАТОК D. ПЕРЕЛІК АКТУАЛЬНИХ СТАНДАРТІВ ТА СПЕЦИФІКАЦІЙ СТАНОМ НА 2026 РІК 505
ДОДАТОК Е. ГЛОСАРІЙ ОСНОВНИХ ТЕРМІНІВ..... 515
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 523

Г. А. Землянко, В.Я. Пєвнєв
H. A. Zemlianko, V.Y. Pevnev

Шлях через епохи цифрових технологій
ІНТЕРФЕЙСИ: КЛАСИФІКАЦІЯ ТА ЕВОЛЮЦІЯ

A Journey Through the Epochs of Digital Technologies:
INTERFACES – CLASSIFICATION AND EVOLUTION

Відповідальний за випуск Г. А. Землянко
Responsible for Publication: Heorhii Zemlianko

Видавець ФОП Бровін О.В.
Свідоцтво про внесення субєкта до Державного реєстру
видавців та виготовників видавничої продукції серія ДК 3587 від 23.09.09 р.
Формат 60x84/16. Ум. друк. арк. 11.86. Тир. 100 прим. Зам. 846.

Надруковано з макету замовника ФОП Бровіна І.П.
61022, м. Харків, вул. Трінклера, 2, корп.1, к.19. Т. (066) 822-71-30

СТИЛЬ-ІЗДАТ®
друкарня
www.stil-izdat.com