

Міністерство освіти і науки України
Вінницький національний технічний університет

Кваліфікаційна наукова праця
на правах рукопису

СКОЩУК ВАЛЕНТИН КОСТЯНТИНОВИЧ

УДК 621.396.6:621.373.5:681.586.7:004.7

ДИСЕРТАЦІЯ

**РАДІОТЕХНІЧНІ ПРИЛАДИ НА FPGA З ЧАСТОТНИМИ
СЕНСОРАМИ ТА ГЕНЕРАТОРАМИ ДЕТЕРМІНОВАНОГО
ХАОСУ ДЛЯ ІОТ-СИСТЕМ ЗБОРУ ДАНИХ**

Спеціальність – 172 «Телекомунікації та радіотехніка»

Галузь знань – 17 «Електроніка та телекомунікації»

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить матеріали власних досліджень. Використання ідей,
результатів і текстів інших авторів мають посилання на відповідне джерело

_____ В. К. Скощук

Науковий керівник: Осадчук Олександр Володимирович
доктор технічних наук, професор

Вінниця - 2026

АНОТАЦІЯ

Скоцук В.К. Радіотехнічні прилади на FPGA з частотними сенсорами та генераторами детермінованого хаосу для IoT-систем збору даних.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 172 «Телекомунікації та радіотехніка» галузі знань 17 «Електроніка та телекомунікації». - Вінницький національний технічний університет, Міністерство освіти і науки України, Вінниця, 2026.

Дисертаційна робота присвячена вирішенню науково-прикладної задачі забезпечення одночасного багатоканального вимірювання частотних сенсорів та відтворюваного цифрового генерування хаотичних сигналів у радіотехнічних приладах на FPGA для IoT-систем збору даних.

В дисертаційній роботі представлені результати наукових досліджень і практичних розробок радіотехнічних приладів на FPGA для збору, обробки та передавання даних від частотних і цифрових сенсорів в IoT-системах, а також FPGA-реалізації генераторів детермінованого хаосу. Досягнуто мету дослідження, яка полягає у забезпеченні одночасного багатоканального вимірювання частотних сенсорів та відтворюваного цифрового генерування хаотичних сигналів у радіотехнічних приладах на FPGA для IoT-систем збору даних за рахунок розробки нових та удосконалення існуючих методів апаратної реалізації на програмованій логіці із застосуванням кусково-лінійної апроксимації нелінійних функцій.

У вступі наведено обґрунтування вибору теми дослідження, зумовлене необхідністю створення IoT-вузлів на FPGA для паралельного збору даних від частотних та цифрових сенсорів, наскрізної передачі телеметрії через бездротові мережі та цифрового генерування хаотичних сигналів. Зазначена актуальність напрямку досліджень у контексті розвитку периферійних обчислень та зростання вимог до детермінованої обробки даних у реальному часі. Сформульовано мету та

сім задач дослідження, зазначено зв'язок роботи з науковими програмами та держбюджетними науково-дослідними роботами кафедри інформаційних радіоелектронних технологій і систем ВНТУ, визначено наукову новизну (чотири пункти) та практичне значення отриманих результатів. Відзначено особистий внесок здобувача у дванадцяти наукових публікаціях та наведено дані про апробацію результатів дисертаційної роботи у дев'яти доповідях на семи науково-практичних конференціях.

У першому розділі виконано аналіз сучасних підходів до побудови IoT-систем збору даних на FPGA та FPGA-реалізацій хаотичних систем. Систематизовано архітектурні підходи до побудови IoT-сенсорних мереж та обґрунтовано доцільність застосування FPGA як обчислювальної платформи для IoT-вузлів сенсорного рівня завдяки апаратному паралелізму, детермінованим затримкам обробки та можливості реконфігурації апаратної логіки без заміни обладнання. Наведено порівняльні приклади практичного застосування FPGA в IoT, зокрема у вузлі FlexiS, архітектурі Мурмес та для апаратного прискорення криптографії (SHA-256 з пропускнуою здатністю 1.4 Гбіт/с).

Проаналізовано класичні та перспективні інтерфейси підключення сенсорів до FPGA (I²C, SPI, UART, I3C), а також виконано порівняльний аналіз бездротових технологій LoRa та Wi-Fi для IoT-мереж з оцінкою дальності, пропускнуої здатності, затримок та енергоспоживання. Систематизовано існуючі FPGA-реалізації хаотичних генераторів (Лоренца, Росслера, Чуа, Чена та інших) та виконано аналіз методів чисельного інтегрування (Ейлера, Рунге-Кутти) і форматів представлення чисел (фіксована та плаваюча кома) для апаратної реалізації хаотичних систем. Встановлено, що модифікована система Аніщенка-Астахова раніше не була реалізована в цифровій формі. Проведено порівняльний аналіз методів апаратної апроксимації функції гіперболічного тангенса (CORDIC, табличний метод, кусково-лінійна апроксимація) та обґрунтовано вибір методу

PWL-апроксимації для задачі високошвидкісної генерації хаотичних сигналів на FPGA.

У другому розділі розроблено архітектуру FPGA-вузла збору даних від частотних та цифрових сенсорів з апаратно-програмним розподілом функцій. Обґрунтовано вибір гібридного архітектурного підходу шляхом порівняння суто апаратного (FSM), суто програмного (MCU) та гібридного (FPGA + soft-core) варіантів реалізації. Реалізовано систему на кристалі (SoC) на базі мікросхеми EP4CE15F23C8 сімейства Cyclone IV E з інтеграцією програмного ядра NIOS II у конфігурації Classic low-performance (20 КБ вбудованої RAM, тактова частота 50 МГц) через утиліту Qsys. Розроблено 12-канальний блок вимірювання частоти із синхронною часовою базою та детермінованим вікном підрахунку 100 мс, що забезпечує роздільність 10 Гц та відносну похибку порядку 10^{-4} – 10^{-5} для діапазону сотень кілогерц. Модуль керування block_form синтезує синхронні сигнали qRst, qCnt, qLd із детермінованим таймінгом.

Вперше розроблено багатоканальну систему збору даних від частотних сенсорів на FPGA, яка, на відміну від існуючих мікроконтролерних рішень з обмеженою кількістю апаратних таймерів, забезпечує одночасне паралельне вимірювання частоти 12 незалежних каналів. Розроблено контролер шини I²C мовою VHDL у вигляді синхронного цифрового автомата з дев'ятьма станами, що забезпечує роботу в режимі ведучого з частотою шини 400 кГц та підтримує механізм clock stretching, транзакції із декількох байтів та Repeated Start. Розроблено багаторівневу архітектуру програмного забезпечення у режимі bare-metal з драйверами частотоміра, I²C та MPU6050. Основний цикл програми працює з фіксованою періодичністю 10 Гц, формуючи вихідні дані у форматі TSV через UART зі швидкістю 115200 бод, обраною як достатня для запланованого обсягу телеметричного трафіку.

У третьому розділі реалізовано наскрізну передачу телеметрії в IoT-мережі від FPGA-вузла до хмарної платформи через LoRa. Розроблено архітектуру IoT-

системи з топологією «зірка», де FPGA-вузли виконують роль периферійних одиниць багатоканального збору та первинної обробки даних з підключених сенсорів, а передача здійснюється через бездротові модулі LilyGO LoRa32 (ESP32 + SX1276, 915 МГц, SF6, BW 500 кГц) на центральний концентратор. Реалізовано протокол фрагментації пакетів для передачі даних обсягом понад 100 байт із серіалізацією у компактному бінарному форматі CBOR, що зменшує обсяг трафіку у радіоканалі LoRa порівняно з текстовими форматами.

Розроблено програмну реалізацію FPGA-вузла та центрального хабу на базі модулів LilyGO LoRa32 у середовищі FreeRTOS із багатозадачною архітектурою, потокобезпечною чергою для буферизації вимірювань та ізоляцією критичних до часу операцій прийому на окремому ядрі процесора. Хаб забезпечує прийом даних по LoRa від віддалених FPGA-вузлів, буферизацію та передачу на хмарну платформу AWS через Wi-Fi з використанням захищеного HTTPS-з'єднання. Розроблено серверну частину IoT-системи на Python (FastAPI, Uvicorn) з REST API для прийому телеметричних даних та їх зберігання у базі даних SQLite, з підтримкою ендпоінтів для отримання списку пристроїв та вимірювань з можливістю фільтрації за ідентифікаторами вузлів. Розроблено мобільний Android-застосунок мовою Kotlin для моніторингу сенсорних даних із асинхронним оновленням табличного відображення вимірювань та захищеним TLS-з'єднанням з хмарним сервісом.

Набув подальшого розвитку метод побудови периферійних вузлів IoT-сенсорних мереж шляхом застосування FPGA як вузла збору даних з наскрізною передачею телеметрії до хмарної платформи через LoRa.

У четвертому розділі виконано математичне моделювання та FPGA-реалізацію генераторів детермінованого хаосу на основі п'яти хаотичних систем третього порядку: Лоренца, Росслера, Нозе-Гувера, Рікітакі та модифікованої системи Аніщенка-Астахова. Розроблено уніфіковану конвеєрну архітектуру для FPGA-реалізації генераторів хаотичних сигналів на основі методу Ейлера першого

порядку з арифметикою фіксованої коми Q16.16 (32-бітний формат зі знаком, діапазон ± 32767 , точність $2^{-16} \approx 1.5 \times 10^{-5}$). Архітектура забезпечує паралельне обчислення трьох змінних стану з синхронним оновленням на основі міток готовності та працює на тактовій частоті 50 МГц на мікросхемі EP4CE15F23C8. Виконано математичне моделювання динаміки всіх п'яти систем методом Ейлера у середовищі Python з арифметикою float64. Для кожної системи підібрано параметри, що забезпечують стійкі хаотичні режими при кроках інтегрування $dt = 0.02$ та $dt = 0.005$, та побудовано еталонні фазові портрети для верифікації.

Реалізовано на FPGA чотири поліноміальні хаотичні системи (Лоренца, Росслера, Нозе-Гувера, Рікітакі) мовою Verilog з апаратними витратами від 476 логічних елементів (система Росслера, 3% EP4CE15) до 610 логічних елементів (система Рікітакі, 4% EP4CE15) та від 24 до 44 вбудованих множників 9×9 .

Вперше реалізовано на FPGA модифіковану систему Аніщенко-Астахова, математичну модель якої запропоновано у працях Семенова та Осадчука. На відміну від поліноміальних систем, модифікована система містить трансцендентну нелінійність $\tanh$ у функції $G(x_1)$, що описує вольт-амперну характеристику транзисторної структури з від'ємним опором. Для апаратного обчислення $\tanh$ запропоновано кусково-лінійну апроксимацію з 31 сегментом на діапазоні $x \in [-10, +10]$ із максимальною похибкою 2.575×10^{-2} . З урахуванням коефіцієнта $d = 0.01$ у рівнянні системи ефективний вплив PWL-похибки на динаміку не перевищує 2.6×10^{-4} , що менше за похибку методу Ейлера. Реалізація модифікованої системи Аніщенко-Астахова (А-А) потребує 4990 логічних елементів (32% EP4CE15) та 90 вбудованих множників 9×9 (80%), що у 8-10 разів перевищує поліноміальні системи. Цифрова FPGA-реалізація, на відміну від аналогових схем, забезпечує детерміновану відтворюваність хаотичних послідовностей за заданих початкових умов та відсутність дрейфу параметрів. Верифікація підтвердила коректність усіх п'яти реалізацій: базові статистичні характеристики (середнє значення, стандартне відхилення, мінімум, максимум)

збігаються з Python-еталоном з точністю до 5%, а форми атракторів візуально ідентичні. Зменшення кроку інтегрування з $dt = 0.02$ до $dt = 0.005$ покращило RMSE від 4.1 разу (система Росслера) до 20.1 разу (система Рікітакі), що підтверджує домінуючий вплив методу Ейлера першого порядку як джерела похибки.

Випробування на платформі EP4CE15F23C8 з виведенням сигналів через 14-бітний ЦАП на цифровий осцилограф підтвердили генерацію стійких аперіодичних обмежених коливань для всіх п'яти систем. Ресурсний аналіз продемонстрував можливість інтеграції генератора хаотичних сигналів з компонентами IoT-вузла на одному кристалі EP4CE15.

У висновках зазначено основні результати дисертаційної роботи. Вирішено актуальне науково-прикладне завдання забезпечення одночасного багатоканального вимірювання частотних сенсорів та відтворюваного цифрового генерування хаотичних сигналів у радіотехнічних приладах на FPGA для IoT-систем збору даних.

Додатки містять копії актів впровадження результатів дисертації, а також список опублікованих праць за темою дисертації.

Ключові слова: FPGA, радіотехнічний прилад, частотний сенсор, багатоканальний частотомір, генератор детермінованого хаосу, IoT-система збору даних, NIOS II, модифікована система Аніщенко-Астахова, транзисторна структура з від'ємним опором, хаотичний атрактор, кусково-лінійна апроксимація, метод Ейлера, апаратно-програмний розподіл функцій, LoRa, бездротова сенсорна мережа.

ABSTRACT

Skoshchuk V.K. FPGA-Based Radio Engineering Devices with Frequency Sensors and Deterministic Chaos Generators for IoT Data Acquisition Systems.

Dissertation for the degree of Doctor of Philosophy in specialty 172 "Telecommunications and Radio Engineering", field of knowledge 17 "Electronics and Telecommunications". – Vinnytsia National Technical University, Ministry of Education and Science of Ukraine, Vinnytsia, 2026.

The dissertation is devoted to solving the scientific and applied problem of ensuring simultaneous multichannel measurement of frequency sensors and reproducible digital generation of chaotic signals in FPGA-based radio engineering devices for IoT data acquisition systems.

The dissertation presents the results of scientific research and practical development of FPGA-based radio engineering devices for collecting, processing, and transmitting data from frequency and digital sensors in IoT systems, as well as FPGA implementation of deterministic chaos generators. The research goal is achieved, which consists in ensuring simultaneous multichannel measurement of frequency sensors and reproducible digital generation of chaotic signals in FPGA-based radio engineering devices for IoT data acquisition systems through the development of new and improvement of existing methods of hardware implementation on programmable logic with the application of piecewise-linear approximation of nonlinear functions.

The introduction provides justification for the research topic, driven by the need to create FPGA-based IoT nodes for parallel data acquisition from frequency and digital sensors, end-to-end telemetry transmission through wireless networks, and digital generation of chaotic signals. The relevance of the research direction in the context of edge computing development and growing requirements for deterministic real-time data processing is stated. The goal and seven research tasks are formulated, the connection of the work with scientific programs and state-funded research projects of the Department

of Information Radio-Electronic Technologies and Systems at VNTU is indicated, scientific novelty (four points) and practical significance of the obtained results are defined. The author's personal contribution to twelve scientific publications is noted, and data on approbation of dissertation results at nine scientific and practical conferences are provided.

The first chapter analyzes modern approaches to building FPGA-based IoT data acquisition systems and FPGA implementations of chaotic systems. Architectural approaches to building IoT sensor networks are systematized and the feasibility of using FPGA as a computational platform for sensor-level IoT nodes is substantiated due to hardware parallelism, deterministic processing delays, and the ability to reconfigure hardware logic without equipment replacement. Comparative examples of practical FPGA applications in IoT are provided, including the FlexiS node, Myrmec architecture, and hardware cryptography acceleration (SHA-256 with throughput of 1.4 Gbps).

Classical and promising sensor connection interfaces to FPGA (I²C, SPI, UART, I3C) are analyzed, and a comparative analysis of LoRa and Wi-Fi wireless technologies for IoT networks is performed with evaluation of range, throughput, latency, and power consumption. Existing FPGA implementations of chaotic generators (Lorenz, Rössler, Chua, Chen, and others) are systematized, and numerical integration methods (Euler, Runge-Kutta) and number representation formats (fixed-point and floating-point) for hardware implementation of chaotic systems are analyzed. It is established that the modified Anishchenko-Astakhov system has not been previously implemented in digital form. A comparative analysis of hardware approximation methods for the hyperbolic tangent function (CORDIC, look-up table, piecewise-linear approximation) is performed, and the PWL approximation method is substantiated for the task of high-speed chaotic signal generation on FPGA.

The second chapter develops the architecture of an FPGA-based data acquisition node for frequency and digital sensors with hardware-software function distribution. The choice of a hybrid architectural approach is substantiated by comparing purely hardware

(FSM), purely software (MCU), and hybrid (FPGA + soft-core) implementation options. A System-on-Chip (SoC) based on the EP4CE15F23C8 chip of the Cyclone IV E family is implemented with integration of the NIOS II soft-core processor in Classic low-performance configuration (20 KB built-in RAM, 50 MHz clock frequency) through the Qsys utility. A 12-channel frequency measurement unit with synchronous time base and deterministic 100 ms counting window is developed, providing 10 Hz resolution and relative error of the order of 10^{-4} – 10^{-5} for the range of hundreds of kilohertz. The block_form control module synthesizes synchronous signals qRst, qCnt, qLd with deterministic timing.

For the first time, a multichannel frequency sensor data acquisition system on FPGA is developed that, unlike existing microcontroller solutions with a limited number of hardware timers, provides simultaneous parallel frequency measurement of 12 independent channels. An I²C bus controller is developed in VHDL as a synchronous digital state machine with nine states, providing master mode operation at 400 kHz bus frequency with support for clock stretching, multi-byte transactions, and Repeated Start. A multi-level bare-metal software architecture with frequency meter, I²C, and MPU6050 drivers is developed. The main program cycle operates at a fixed 10 Hz periodicity, producing output data in TSV format via UART at 115200 baud, selected as sufficient for the planned telemetry traffic volume.

The third chapter implements end-to-end telemetry transmission in an IoT network from an FPGA node to a cloud platform via LoRa. An IoT system architecture with a "star" topology is developed, where FPGA nodes serve as peripheral units for multichannel data acquisition and primary processing from connected sensors, with transmission through LilyGO LoRa32 wireless modules (ESP32 + SX1276, 915 MHz, SF6, BW 500 kHz) to a central hub. A packet fragmentation protocol for transmitting data exceeding 100 bytes with serialization in compact binary CBOR format is implemented, reducing traffic volume in the LoRa radio channel compared to text formats.

Software implementation of the FPGA node and central hub based on LilyGO LoRa32 modules in FreeRTOS environment with multitask architecture, thread-safe queue for measurement buffering, and isolation of time-critical reception operations on a separate processor core is developed. The hub receives data via LoRa from remote FPGA nodes, buffers it, and transmits to the AWS cloud platform via Wi-Fi using a secure HTTPS connection. The server part of the IoT system is developed in Python (FastAPI, Uvicorn) with REST API for receiving telemetry data and storing it in an SQLite database, with support for endpoints for obtaining device lists and measurements with filtering by node identifiers. A mobile Android application in Kotlin for sensor data monitoring with asynchronous update of tabular measurement display and secure TLS connection to the cloud service is developed.

The method of building peripheral nodes of IoT sensor networks through the use of FPGA as a data acquisition node with end-to-end telemetry transmission to a cloud platform via LoRa is further developed.

The fourth chapter performs mathematical modeling and FPGA implementation of deterministic chaos generators based on five third-order chaotic systems: Lorenz, Rössler, Nosé-Hoover, Rikitake, and the modified Anishchenko-Astakhov system. A unified pipeline architecture for FPGA implementation of chaotic signal generators based on the first-order Euler method with Q16.16 fixed-point arithmetic (32-bit signed format, range ± 32767 , precision $2^{-16} \approx 1.5 \times 10^{-5}$) is developed. The architecture provides parallel computation of three state variables with synchronous update based on readiness flags and operates at 50 MHz clock frequency on the EP4CE15F23C8 chip. Mathematical modeling of the dynamics of all five systems using the Euler method in the Python environment with float64 arithmetic is performed. For each system, parameters ensuring stable chaotic regimes at integration steps $dt = 0.02$ and $dt = 0.005$ are selected, and reference phase portraits for verification are constructed.

Four polynomial chaotic systems (Lorenz, Rössler, Nosé-Hoover, Rikitake) are implemented on FPGA in Verilog with hardware costs from 476 logic elements (Rössler

system, 3% of EP4CE15) to 610 logic elements (Rikitake system, 4% of EP4CE15) and from 24 to 44 embedded 9×9 multipliers.

For the first time, the modified Anishchenko-Astakhov system is implemented on FPGA, the mathematical model of which was proposed in the works of Semenov and Osadchuk. Unlike polynomial systems, the modified system contains the transcendental nonlinearity $\tanh$ in the function $G(x_1)$, which describes the volt-ampere characteristic of a transistor structure with negative resistance. For hardware computation of $\tanh$, piecewise-linear approximation with 31 segments over the range $x \in [-10, +10]$ with maximum error of 2.575×10^{-2} is proposed. Considering the coefficient $d = 0.01$ in the system equation, the effective impact of PWL error on the system dynamics does not exceed 2.6×10^{-4} , which is less than the Euler method error. The implementation of the modified Anishchenko-Astakhov (A-A) system requires 4990 logic elements (32% of EP4CE15) and 90 embedded 9×9 multipliers (80%), which is 8–10 times more than polynomial systems. The digital FPGA implementation, unlike analog circuits, provides deterministic reproducibility of chaotic sequences for given initial conditions and absence of parameter drift. Verification confirmed the correctness of all five implementations: basic statistical characteristics (mean, standard deviation, minimum, maximum) match the Python reference within 5%, and attractor shapes are visually identical. Reducing the integration step from $dt = 0.02$ to $dt = 0.005$ improved RMSE by 4.1 times (Rössler system) to 20.1 times (Rikitake system), confirming the dominant influence of the first-order Euler method as the error source.

Field tests on the EP4CE15F23C8 platform with signal output through a 14-bit DAC to a digital oscilloscope confirmed generation of stable aperiodic bounded oscillations for all five systems. Resource analysis demonstrated the possibility of integrating a chaotic signal generator with IoT node components on a single EP4CE15 chip.

The conclusions state the main results of the dissertation. The relevant scientific and applied problem of ensuring simultaneous multichannel measurement of frequency

sensors and reproducible digital generation of chaotic signals in FPGA-based radio engineering devices for IoT data acquisition systems is solved.

The appendices contain copies of implementation acts and a list of published works on the dissertation topic.

Keywords: FPGA, radio engineering device, frequency sensor, multichannel frequency meter, deterministic chaos generator, IoT data acquisition system, NIOS II, modified Anishchenko-Astakhov system, transistor structure with negative differential resistance, chaotic attractor, piecewise-linear approximation, Euler method, hardware-software partitioning, LoRa, wireless sensor network.

СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові та практичні результати дисертації:

- матеріали дисертаційної роботи, викладені у періодичних виданнях, що індексовані в наукометричних базах Scopus та Web of Science:

1. Oleksandr V. Osadchuk, Iaroslav O. Osadchuk, Valentyn K. Skoshchuk. IoT system with frequency converters of physical quantities on FPGA. *Informatyka, Automatyka, Pomiarы w Gospodarce i Ochronie Środowiska*, Volume 15, Issue 3, Pages 59–66, 2025. DOI: <https://doi.org/10.35784/iapgos.7076> (Scopus) ISSN 2083-0157 (print); eISSN 2391-6761 (online)

2. Oleksandr Osadchuk, Iaroslav Osadchuk, Valentyn Skoshchuk. Using FPGA for modelling and generating chaotic processes. *Informatyka, Automatyka, Pomiarы w Gospodarce i Ochronie Środowiska*, Volume 16, Issue 1, Pages 72–77, 2026. DOI: <https://doi.org/10.35784/iapgos.7583> (Scopus) ISSN 2083-0157 (print); eISSN 2391-6761 (online)

3. Oleksandr Osadchuk, Cezary Kaczmarek, Iaroslav Osadchuk, Olena Nanaka, Oleksandr Melnyk, Valentyn Skoshchuk, Vitaliy Petrenko, Nurzhigit Smailov. Optical-electronic generator of determinative chaos based on bipolar field transistor structure with negative differential resistance. *Proceedings of SPIE - The International Society for Optical Engineering*, Volume 14009, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2025, Article 140091A, 2025. DOI: <https://doi.org/10.1117/12.3098870> (Scopus, Web of Science). ISSN: 0277-786X; eISSN: 1996-756X. ISBN: 9781510699748; eISBN: 9781510699755

- матеріали дисертаційної роботи, викладені у виданнях, що внесені до переліку наукових фахових видань України категорії Б для захисту дисертацій з технічних наук:

4. Осадчук О.В., Осадчук Я.О., Скощук В.К. Багатоканальний частотомір на програмованій логічній інтегральній схемі для радіовимірювальної системи з частотними сенсорами фізичних величин. *Вісник Хмельницького національного університету*. Серія: Технічні науки. № 6, 2021 (303). С. 186–194. DOI: <https://www.doi.org/10.31891/2307-5732-2021-303-6-186-194> ISSN 2307-5732

5. Осадчук О., Осадчук Я., Скощук В. Використання ядра NIOS II у багатоканальному частотомірі на FPGA для радіотехнічної системи з частотними сенсорами фізичних величин. *Вимірювальна та обчислювальна техніка в технологічних процесах*, (1), 2023, 137–148. DOI: <https://doi.org/10.31891/2219-9365-2023-73-1-19> ISSN 2219-9365

6. Осадчук Олександр, Осадчук Ярослав, Скощук Валентин. Удосконалення багатоканальної радіотехнічної системи на FPGA для частотних перетворювачів фізичних величин підтримкою цифрових сенсорів. *Вимірювальна та обчислювальна техніка в технологічних процесах*, (2), 2023, С. 72–82. DOI: <https://doi.org/10.31891/2219-9365-2023-74-10> ISSN 2219-9365

7. Осадчук О., Осадчук Я., Скощук В. Інтеграція LILYGO LORA32 у багатоканальну радіотехнічну систему на FPGA для частотних перетворювачів фізичних величин. *Вимірювальна та обчислювальна техніка в технологічних процесах*, № 4, 2024, С. 74–83. DOI: <https://doi.org/10.31891/2219-9365-2024-80-10> ISSN 2219-9365

8. Осадчук О., Скощук В. Бездротова система збору даних на FPGA для частотних перетворювачів фізичних величин із використанням LoRa. *Herald of Khmelnytskyi National University. Technical Sciences (Вісник Хмельницького національного університету. Серія: Технічні науки)*. Том 347 № 1, 2025. С. 375–386. URL: <https://heraldts.khmnua.edu.ua/index.php/heraldts/article/view/1047> ISSN 2307-5732

9. Осадчук О.В., Осадчук Я.О., Петренко В.І., Скощук В.К., Шикун К.В. Експериментальне дослідження генератора детермінованого хаосу на основі

транзисторної структури. *Опτικο-електронні інформаційно-енергетичні технології*. Том 49 № 1, 2025, С. 235–246. DOI: <https://doi.org/10.31649/1681-7893-2025-49-1-235-246> ISSN 2311-2662 (Online), ISSN 1681-7893 (Print)

10. Осадчук О.В., Осадчук Я.О., Петренко В.І., Скощук В.К. Розробка генератора детермінованого хаосу на основі транзисторної структури з від'ємним опором. *Опτικο-електронні інформаційно-енергетичні технології*. Том 49 № 1, 2025. С. 247–256. DOI: <https://doi.org/10.31649/1681-7893-2025-49-1-247-256> ISSN 2311-2662 (Online), ISSN 1681-7893 (Print)

11. Осадчук О., Осадчук Я., Скощук В., Петренко В. Генератор детермінованого хаосу на основі біполярно-польової транзисторної структури з від'ємним диференціальним опором. *Вимірювальна та обчислювальна техніка в технологічних процесах*, 82(2), 2025, С. 240–249. URL: <https://vottp.khmnu.edu.ua/index.php/vottp/article/view/511> ISSN 2219-9365

12. Осадчук О.В., Осадчук Я.О., Скощук В.К., Петренко В.І., Шикун К.В. FPGA-реалізація хаотичного атрактора на основі структури з від'ємним опором за допомогою модифікованої моделі Аніщенка-Астахова. *Опτικο-електронні інформаційно-енергетичні технології*. Том 50, № 2 (2025). С. 311–321. DOI: <https://doi.org/10.31649/1681-7893-2025-50-2-311-321> ISSN 2311-2662 (Online), ISSN 1681-7893 (Print)

- наукові праці, які засвідчують апробацію матеріалів дисертації:

13. Скощук В.К., Осадчук О.В. Передачі накопичених даних радіотехнічною системою на FPGA для частотних перетворювачів фізичних величин використовуючи технологію WiFi. *Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)»*, 15–16 червня 2025 р., Вінниця, ВНТУ, 2025. С. 1–4. [Електронний ресурс]. URL:

<https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/download/23195/19197>

14. Скощук В.К., Осадчук О.В. Збір вимірювань із багатоканальних радіотехнічних систем на FPGA для частотних перетворювачів фізичних величин використовуючи LoRa концентратор на базі LilyGO LoRa32. *Тези доповідей LIV Всеукраїнської науково-технічної конференції факультету інформаційних електронних систем (ВНТКП ВНТУ-2025)*, 24–27 березня 2025 р., Вінниця, ВНТУ, 2025. С. 1–4. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/all-frtzip/all-frtzip-2025/paper/download/23194/19193>

15. Скощук В. К., Осадчук О. В. Проектування структури системи передачі даних для багатоканальної радіотехнічної системи на FPGA з використанням LoRa. *Матеріали IX Міжнародної науково-практичної конференції «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем (СПІРН-2023)»*, 15–17 листопада 2023 р., Вінниця, ВНТУ, 2023. С. 1–3. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/spirn/spirn2023/paper/view/19141>

16. Скощук В.К., Осадчук О.В., Осадчук Я.О. Використання ядра NIOS II як шлях до удосконалення багатоканальної радіотехнічної система на FPGA для частотних перетворювачів фізичних величин. *Матеріали LII Науково-технічної конференції факультету інформаційних електронних систем (2023)*, 21–23 червня 2023 р., Вінниця, ВНТУ, 2023. С. 1–4. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/all-frtzip/all-frtzip-2023/paper/download/17111/14402>

17. Скощук В.К., Осадчук О.В., Осадчук Я.О. Використання цифрових сенсорів у багатоканальній радіотехнічній системі на FPGA для частотних перетворювачів фізичних величин із ядром NIOS II. *Матеріали Міжнародної науково-практичної інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)»*, 22–23 червня 2023 р., Вінниця, ВНТУ, 2023. С. 1–4. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/mn/mn2023/paper/view/17110>

18. Осадчук О.В., Скощук В.К. Багатоканальний частотомір на FPGA для радіовимірювальної системи з частотними сенсорами. *8-а Міжнародна науково-практична конференція «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем» (СПРН-2021)*, 03–05 листопада 2021 р., Вінниця, ВНТУ, 2021. С. 1–3. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/spirn/spirn2021/paper/download/13742/11583>

19. Осадчук О.В., Осадчук Я.О., Скощук В.К. Багатоканальна радіовимірювальна система на ПЛІС для частотних перетворювачів фізичних величин. *Матеріали I Науково-технічної конференції підрозділів Вінницького національного технічного університету (HTKП ВНТУ-2021)*, м. Вінниця, 10–12 березня 2021 р., Вінниця: ВНТУ, 2021. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/all-frtzip/all-frtzip-2021/paper/view/11435>

20. Oleksandr Osadchuk, Iaroslav Osadchuk, Paweł Komada, Aliya Kalizhanova, Valentyn Skoshchuk, Vitaliy Petrenko, Yaroslav Yaroslavskyy, Mikhailo Vatso. Self-oscillating parametric optical power transducer with photosensitive bipolar transistor. *Proceedings of SPIE - The International Society for Optical Engineering*, 30 December 2025. DOI: <https://doi.org/10.1117/12.3100537> (Scopus, Web of Science). ISSN: 0277-786X; eISSN: 1996-756X. ISBN: 9781510699748; eISBN: 9781510699755

21. Скощук В. К., Осадчук О. В. Реалізація системи передачі даних для багатоканальної радіотехнічної системи на FPGA з використанням LoRa. *IX Міжнародна науково-технічна конференція «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем» (СПРН-2023)*, 15–17 листопада 2023 р., Україна, м. Вінниця, ВНТУ, 2023. С. 1–4. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/spirn/spirn2023/paper/view/19142>

ЗМІСТ

ВСТУП	21
РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ПОБУДОВИ ІОТ-СИСТЕМ ЗБОРУ ДАНИХ НА FPGA ТА FPGA-РЕАЛІЗАЦІЙ ХАОТИЧНИХ СИСТЕМ.....	
	32
1.1 Місце FPGA в архітектурі IoT-систем збору даних	32
1.2 Частотні сенсори та методи багатоканального вимірювання частоти	38
1.3 Сенсорні інтерфейси та технології передачі даних в IoT-мережах	47
1.4 Хаотичні системи та їх цифрова реалізація на FPGA	54
1.5 Методи реалізації нелінійних функцій на FPGA	68
1.6 Висновки до розділу 1	72
РОЗДІЛ 2. РОЗРОБКА FPGA-ВУЗЛА ЗБОРУ ДАНИХ ВІД ЧАСТОТНИХ ТА ЦИФРОВИХ СЕНСОРІВ З АПАРАТНО-ПРОГРАМНИМ РОЗПОДІЛОМ ФУНКЦІЙ	
	76
2.1 Загальна архітектура FPGA-системи збору даних.....	76
2.2 Інтеграція процесорного ядра NIOS II.....	80
2.3 Модуль зчитування цифрових сенсорів по інтерфейсу I2C	90
2.4 Модуль зчитування частотних сенсорів	97
2.5 Апаратні та програмні аспекти підключення сенсорів	103
2.6 Висновки до розділу 2	117
РОЗДІЛ 3. НАСКРІЗНА ПЕРЕДАЧА ТЕЛЕМЕТРІЇ В ІОТ-МЕРЕЖІ ВІД FPGA-ВУЗЛА ДО ХМАРНОЇ ПЛАТФОРМИ ЧЕРЕЗ LORA	
	119
3.1 Архітектура IoT-системи та вибір топології	119
3.2 Розробка FPGA-вузла на базі LilyGO LoRa32	123
3.3 Розробка центрального хабу на базі LilyGO LoRa32	134
3.4 Хмарний сервіс AWS та мобільний додаток для моніторингу сенсорних даних	144
3.5 Висновки до розділу 3	152

РОЗДІЛ 4. МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ТА

FPGA-РЕАЛІЗАЦІЯ ГЕНЕРАТОРІВ ДЕТЕРМІНОВАНОГО ХАОСУ	154
4.1 Уніфікована конвеєрна архітектура генераторів хаотичних сигналів на FPGA	154
4.2 Математичне моделювання хаотичної динаміки.....	158
4.3 FPGA-реалізація поліноміальних хаотичних систем	165
4.4 FPGA-реалізація модифікованої системи Аніщенка-Астахова	170
4.5 Верифікація та експериментальні результати.....	176
4.6 Висновки до розділу 4	191
ВИСНОВКИ.....	194
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	199
ДОДАТКИ.....	212
ДОДАТОК 1 Акти впровадження результатів дисертаційної роботи.....	213
ДОДАТОК 2 Список публікацій здобувача. Відомості про апробацію.....	215

ВСТУП

Обґрунтування вибору теми дослідження.

Подальший розвиток Інтернету речей (Internet of Things, IoT) та перехід до концепції периферійних обчислень (Edge Computing) задають нові вимоги до сенсорних мереж: зростають потреби у детермінованій обробці даних у реальному часі, масштабованості, енергоефективності, а також у захисті телеметричної інформації під час передавання у бездротових мережах і хмарних платформах. Для прикладних сценаріїв (промисловий моніторинг, «розумне місто», роботизовані системи) критичними стають не лише середні значення, а й гарантовані часові характеристики на периферійному рівні та стійкість до атак/помилки на всьому шляху даних. У таких умовах традиційні мікроконтролерні рішення не завжди забезпечують одночасно необхідну швидкодію, паралелізм і гнучкість апаратної реалізації алгоритмів обробки й захисту, особливо коли сенсорні потоки потрібно не лише зчитувати, а й попередньо агрегувати, фільтрувати та формувати пакети для передавання.

Програмовані логічні інтегральні схеми (FPGA) є перспективною апаратною платформою для побудови IoT-вузлів завдяки можливості апаратного паралелізму, детермінованому часовому виконанню, реалізації інтерфейсів та обчислювальних модулів, а також інтеграції програмного керування на базі soft-core процесора (зокрема NIOS II) у межах концепції «система на кристалі» (SoC). Така платформа дозволяє винести критичні до часу операції (зчитування сенсорів, формування протоколів, первинну обробку) в апаратну частину, а задачі керування та інтеграції - у програмну частину, що створює основу для побудови продуктивних і енергоефективних периферійних вузлів IoT. Одночасно актуальним є підвищення захищеності IoT-вузлів, оскільки телеметрія, що передається через бездротові протоколи та хмарні сервіси, є потенційною ціллю перехоплення, підміни або несанкціонованої модифікації.

Окремий інтерес становить використання хаотичних систем як джерела псевдовипадковості та основи для побудови апаратних криптографічних примітивів. Проте практична реалізація хаотичних генераторів у вбудованих системах потребує вирішення задачі високошвидкісного чисельного інтегрування, ефективної фіксованої арифметики та апаратної реалізації нелінійних функцій.

У зв'язку з цим актуальним науково-прикладним завданням є побудова IoT-сенсорної мережі на базі FPGA з наскрізною передачею даних до хмари та розробка генераторів детермінованого хаосу на FPGA як основи для апаратного захисту інформації.

Актуальність теми.

Актуальність теми зумовлена необхідністю створення IoT-вузлів, здатних забезпечувати:

- паралельне зчитування та попередню обробку даних від різномірних сенсорів (цифрових і частотних) з мінімальними затримками;
- надійну дальню бездротову передачу даних у топології «зірка» (зокрема з використанням LoRa) із подальшим передаванням у хмарні сервіси (AWS) та візуалізацією даних користувачеві;
- підвищення захищеності телеметрії шляхом апаратної реалізації високошвидкісних джерел псевдовипадковості та модулів шифрування, що можуть бути інтегровані безпосередньо в структуру FPGA.

Практична значущість цієї проблематики посилюється тим, що для прикладних сценаріїв (промисловий моніторинг, «розумне місто», системи дистанційного збору даних) важливі одночасно часові характеристики, автономність та наскрізна захищеність на всьому шляху даних, від сенсора до хмарного сервісу та клієнтського застосунку.

Науково-прикладним завданням, вирішенню якому присвячена дисертаційна робота, є забезпечення одночасного багатоканального вимірювання частотних сенсорів та відтворюваного цифрового генерування хаотичних сигналів у

радіотехнічних приладах на FPGA для IoT-систем збору даних за рахунок розробки нових та удосконалення існуючих методів апаратної реалізації на програмованій логіці.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертаційні дослідження виконувались у відповідності до наукового напряму кафедри інформаційних радіoeлектронних технологій і систем Вінницького національного технічного університету, в межах низки держбюджетних науково-дослідних робіт: «Методи та пристрої формування й оброблення хаотичних сигналів, контролю доступу та позиціювання у робототехнічних та інфокомунікаційних системах» (2021-2022 р., номер державної реєстрації 0121U109722); «Радіовимірювальні частотні сенсори з автономним прийняттям рішення для робототехнічних пристроїв, систем телеуправління та спеціалізованих хмарних платформ» (2022-2023 р., номер держреєстрації 0122U001272).

Мета і завдання дослідження.

Мета дисертаційної роботи полягає в забезпеченні одночасного багатоканального вимірювання частотних сенсорів та відтворюваного цифрового генерування хаотичних сигналів у радіотехнічних приладах на FPGA для IoT-систем збору даних за рахунок розробки нових та удосконалення існуючих методів апаратної реалізації на програмованій логіці із застосуванням кусково-лінійної апроксимації нелінійних функцій.

Завданнями дисертаційного дослідження є:

1. Проаналізувати сучасні архітектури IoT-сенсорних мереж, визначити вимоги до периферійних вузлів та обґрунтувати доцільність застосування FPGA як обчислювальної платформи.
2. Розробити гібридну архітектуру IoT-вузла на базі FPGA Cyclone IV з інтеграцією soft-core процесора NIOS II та реалізувати апаратно-програмний розподіл функцій (Hardware/Software Co-design).

3. Реалізувати багатоканальну систему збору даних від частотних сенсорів на FPGA, зокрема апаратний 12-канальний частотомір, контролер ІС для цифрових сенсорів та UART-інтерфейс обміну даними.

4. Побудувати наскрізний канал передавання та моніторингу телеметрії за схемою «сенсор - FPGA - LoRa - шлюз (ESP32) - хмара (AWS)» та реалізувати протоколи взаємодії між компонентами.

5. Розробити програмні компоненти хмарної частини та клієнтський мобільний додаток для візуалізації сенсорних даних у реальному часі та в історичному розрізі.

6. Розробити та верифікувати бібліотеку IP-ядер генераторів детермінованого хаосу (Лоренца, Росслера, Нозе-Гувера, Рікітакі, Аніщенко-Астахова) на FPGA з можливістю подальшого використання у задачах захисту інформації, включаючи розробку методу кусково-лінійної апроксимації нелінійної функції гіперболічного тангенса для модифікованої системи Аніщенко-Астахова.

7. Провести програмне моделювання (Python) та експериментальну верифікацію апаратної реалізації (RTL-симуляція, випробування на FPGA з виведенням сигналів через ЦАП), оцінити точність (RMSE, відносні похибки) і швидкодію.

Об'єктом дослідження є процеси багатоканального вимірювання частотних сенсорів, наскрізної передачі телеметрії в IoT-мережах та цифрового генерування хаотичних сигналів для задач захисту інформації з використанням радіотехнічних приладів на базі FPGA.

Предметом дослідження є схемотехнічні рішення, методи реалізації та параметри радіотехнічних приладів на FPGA для IoT-систем збору даних та генерування хаотичних сигналів, зокрема:

- структурна організація та параметри багатоканального частотоміра на FPGA з апаратно-програмним розподілом функцій (NIOS II);

- протокольні рішення наскрізної передачі телеметрії в IoT-мережі (UART, LoRa, REST/HTTPS) та організація каналу «сенсор - FPGA - шлюз - хмара - клієнт»;

- методи та показники точності і швидкодії (RMSE, максимальна похибка) цифрової FPGA-реалізації генератора хаотичних коливань на основі транзисторної структури з від'ємним диференціальним опором за модифікованим методом Аніщенка-Астахова із кусково-лінійною апроксимацією нелінійної функції гіперболічного тангенса.

Методи дослідження. Під час проведення дослідження були використані такі методи: аналіз і порівняння сучасних архітектур IoT та протоколів зв'язку; методи апаратно-програмного проектування (co-design) та моделювання цифрових систем; чисельні методи інтегрування систем диференціальних рівнянь (зокрема явний метод Ейлера); методи фіксованої арифметики та кусково-лінійної апроксимації нелінійних функцій; RTL-симуляція та верифікація (включно з підходом «HDL - Python» через формування файлів із відліками у форматі Q16.16); експериментальні вимірювання з використанням ЦАП та осцилографа; статистичний аналіз похибок (RMSE, максимальна абсолютна та відносна похибка) і порівняння з еталонними програмними моделями (Python).

Наукова новизна роботи полягає у:

1. *Вперше розроблено багатоканальну систему збору даних від частотних сенсорів на FPGA, яка, на відміну від існуючих мікроконтролерних рішень, що використовують послідовне опитування каналів з обмеженою кількістю апаратних таймерів, забезпечує одночасне паралельне вимірювання частоти 12 незалежних каналів із синхронною часовою базою та детермінованим вікном вимірювання 100 мс.*

2. *Удосконалено структурну організацію IoT-вузла на FPGA шляхом апаратно-програмного розподілу функцій, яка, на відміну від суто апаратних або суто програмних підходів, забезпечує одночасну роботу апаратного*

багатоканального частотоміра та контролера ІС для цифрових сенсорів під керуванням програмного ядра NIOS II на одному кристалі з детермінованим циклом опитування ~ 10 Гц.

3. *Вперше розроблено* цифрову FPGA-реалізацію модифікованої хаотичної системи Аніщенко-Астахова, яка, на відміну від існуючих досліджень цієї системи, що обмежуються теоретичним аналізом та аналоговими схемами на дискретних компонентах, забезпечує детерміновану відтворюваність хаотичних послідовностей за заданих початкових умов, відсутність дрейфу параметрів та можливість інтеграції генератора в систему на кристалі (SoC) для криптографічних застосувань.

4. *Набув подальшого розвитку* метод побудови периферійних вузлів IoT-сенсорних мереж шляхом застосування FPGA як вузла збору даних з наскрізною передачею телеметрії до хмарної платформи через LoRa, який, на відміну від типових MCU-базованих IoT-архітектур, забезпечує наскрізну доставку телеметрії від FPGA-вузла з частотними сенсорами до хмарного сервісу з можливістю масштабування кількості вузлів у топології «зірка» та моніторингу даних у режимі, близькому до реального часу.

Практичне значення одержаних результатів.

Практичне значення одержаних результатів полягає у створенні технологічної та методичної основи для проєктування IoT-пристроїв із підвищеною швидкістю обробки даних на периферії та можливістю інтеграції апаратних засобів захисту телеметрії. Основні практичні результати та їх прикладне значення полягають у наступному:

- розроблено архітектуру периферійного вузла збору даних на базі FPGA Cyclone IV із апаратно-програмним розподілом функцій (SoC, FPGA + NIOS II), що забезпечує детерміноване зчитування сенсорів, формування та підготовку телеметрії до передавання;

- реалізовано сенсорні інтерфейси та режим роботи вузла збору даних: ІС для цифрових сенсорів у режимі 400 кГц; формування та передавання телеметрії через UART зі швидкістю 115200 бод, яка не є фізичним обмеженням системи та може бути збільшена відповідно до вимог конкретного застосування; детермінований цикл опитування з фіксованою періодичністю порядку 10 Гц;
- запропоновано та практично реалізовано наскрізний канал передавання телеметрії в гетерогенній системі «сенсор - FPGA - LoRa - шлюз (ESP32) - хмара (AWS)» із можливістю масштабування у топології «зірка», що забезпечує віддалений моніторинг даних у прикладних сценаріях;
- розроблено серверну частину для приймання та збереження телеметрії у хмарному середовищі, включно з прийманням CBOR-повідомлень через REST API, декодуванням та збереженням вимірювань у базі даних із подальшим наданням доступу до вибірок через API;
- розроблено клієнтський мобільний застосунок для візуалізації телеметрії, із підтримкою фільтрації даних за ідентифікаторами вузлів та відображенням поточних і історичних вимірювань;
- спроектовано та верифіковано бібліотеку ІР-ядер генераторів детермінованого хаосу для FPGA (Лоренца, Росслера, Нозе-Гувера, Рікітакі, Аніщенка-Астахова) на основі різницевих схем методу Ейлера першого порядку у форматі фіксованої коми Q16.16; для системи Аніщенка-Астахова застосовано кусково-лінійну апроксимацію (PWL) з 31 сегментом для реалізації трансцендентної нелінійності;
- працездатність і точність апаратної реалізації хаотичних генераторів підтверджено верифікацією та експериментально для всіх п'яти систем: базові статистичні характеристики (середнє значення, стандартне відхилення, мінімум, максимум) збігаються з Python-еталоном з точністю до 5%, а форми атракторів візуально ідентичні. Зменшення кроку інтегрування з $dt = 0.02$ до $dt = 0.005$ покращило RMSE від 4.1 разу (система Росслера) до 20.1 разу (система Рікітакі),

що підтверджує домінуючий вплив методу Ейлера першого порядку як джерела похибки. Верифікацію виконано підходом «HDL - Python» із формуванням файлів відліків у шістнадцятковому форматі Q16.16 та порівнянням з еталонними програмними моделями. На платі з FPGA Cyclone IV E (EP4CE15F23C8) продемонстровано роботу генераторів із тактуванням 50 МГц та виведенням сигналів через 14-бітну шину на ЦАП із подальшим спостереженням на осцилографі;

Реалізація та впровадження результатів роботи.

Результати дисертаційної роботи мають такі впровадження: архітектуру периферійного IoT-вузла на базі FPGA з апаратно-програмним розподілом функцій, наскрізну гетерогенну архітектуру збору та передачі телеметрії в системі «сенсор - FPGA - LoRa - шлюз - хмара», а також методи апаратної реалізації генераторів детермінованого хаосу на FPGA впроваджені у проєктно-конструкторській діяльності Sinai RnD LLC м. Вінниця при розробці програмного забезпечення для вбудованих систем та IoT-рішень (Довідка від «10» грудня 2025 р.).

Лабораторні макети периферійного IoT-вузла на базі FPGA Cyclone IV EP4CE15F23C8 з апаратно-програмним розподілом функцій, що включають 12-канальний апаратний частотомір, апаратний I²C-контролер та процесор NIOS II, лабораторні макети генераторів детермінованого хаосу на FPGA (системи Лоренца, Росслера, Нозе-Гувера, Рікітакі та модифікована система Аніщенка-Астахова) з виведенням сигналів через ЦАП, архітектуру наскрізного каналу передачі телеметрії в гетерогенній IoT-системі «сенсор - FPGA - LoRa - хмара», методологію проєктування апаратних модулів на FPGA мовами Verilog HDL та VHDL, а також відповідне методичне забезпечення впроваджені у навчальному процесі Вінницького національного технічного університету під час викладання навчальних дисциплін «Сенсори фізичних величин», «Радіовимірювальні перетворювачі інформації», «Вимірювання в телекомунікаціях та радіотехніці»,

«Конструювання та технологія телекомунікаційних та радіотехнічних систем», «Радіотехнічні прилади з від'ємним опором», «Радіоелектронні системи» першого (бакалаврського) рівня підготовки за спеціальністю 172 – Телекомунікації та радіотехніка, а також при виконанні бакалаврських і магістерських кваліфікаційних робіт студентами спеціальності 172 – Телекомунікації та радіотехніка (Акт впровадження від «12» січня 2026 р.).

Особистий внесок здобувача.

Наукові результати, що наведені в дисертаційній роботі, одержані автором самостійно. У публікаціях, які були підготовлені та опубліковані в співавторстві [1–12], авторові належать (особистий внесок): розробка повного IoT-стеку включаючи серверну частину на FastAPI з SQLite та HTTPS, мобільний додаток на Android/Kotlin, інтеграцію з хмарною інфраструктурою AWS та реалізацію FreeRTOS черги повідомлень для асинхронної обробки із захистом SSL/TLS [1]; участь у Python-моделюванні методом Ейлера та FPGA-реалізації хаотичних систем Нозе-Гувера та Пікітакі у Verilog HDL з арифметикою Q16.16, верифікація у ModelSim, розробка UART-модулів для передачі обчислень на ПК у реальному часі [2]; участь у дослідженні оптоелектронного генератора хаосу з біполярним транзистором та двозатворним MOSFET із часом встановлення 15.36-19.82 нс та гальванічною ізоляцією виходу [3]; запропоновано архітектуру 12-канального частотоміра на FPGA Altera Cyclone IV EP4CE15F23C8 з діапазоном вимірювання 0-1 МГц та роздільною здатністю 1 Гц, реалізовано модулі формування керуючих сигналів, лічильників імпульсів, генератора тексту та UART-передавача на Verilog HDL [4]; результати інтеграції soft-core процесора NIOS II Classic у багатоканальну радіотехнічну систему для забезпечення програмної конфігурації сенсорів без зміни апаратної частини та розробка програмного забезпечення обробки даних на C/C++ [5]; реалізація апаратного I2C-master блоку та програмного драйвера для підтримки цифрових сенсорів, результати інтеграції акселерометра, гіроскопа MPU6050 для 6-осьового вимірювання руху [6];

результати інтеграції модуля LilyGO LoRa32 (ESP32 + SX1276) з FPGA-системою через інтерфейс UART та розробка структури даних JSON для передачі телеметрії [7]; запропоновано архітектуру бездротової мережі збору даних з LoRa концентратором, розроблено бібліотеку LoraTransport з підтримкою багаточастинних пакетів та CBOR-кодуванням, результати оптимізації LoRa параметрів (915 МГц, SF6, 500 кГц BW, Implicit Header) [8]; участь у проведенні експериментального дослідження характеристик генератора хаосу в діапазоні 15 кГц - 3 ГГц включаючи вимірювання S-параметрів, імпедансу та добротності [9]; участь у розробці математичної моделі генератора детермінованого хаосу на двох біполярних транзисторах з від'ємним диференціальним опором методом змінних стану та проведення комп'ютерного моделювання в MATLAB Simulink [10]; участь у дослідженні гібридного генератора хаосу на основі біполярно-польової транзисторної структури з надшвидким часом встановлення стаціонарних коливань 17.25-21.28 нс та розробці математичної моделі з рівняннями польового транзистора [11]; участь у розробці комбінаторної FPGA-реалізації хаотичного атрактора на основі структури з від'ємним опором за допомогою модифікованої моделі Аніщенко-Астахова у фіксованій точці Q16.16 з чисельним інтегруванням методом Ейлера та підтвердженням стійкого хаотичного режиму шляхом Python-моделювання [12]. Апробація основних положень дисертації здійснена за матеріалами доповідей і тез конференцій [13–21].

Апробація результатів дисертації. Основні положення та результати дисертаційної роботи доповідалися та обговорювалися на: 1) Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)», 15–16 червня 2025 р., Вінниця, ВНТУ; 2) LIV Всеукраїнській науково-технічній конференції факультету інформаційних електронних систем (ВНТКП ВНТУ-2025), 24–27 березня 2025 р., Вінниця, ВНТУ; 3) IX Міжнародній науково-практичній конференції «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем (СПРН-2023)», 15–17

листопада 2023 р., Вінниця, ВНТУ; 4) ЛП Науково-технічній конференції факультету інформаційних електронних систем (2023), 21–23 червня 2023 р., Вінниця, ВНТУ; 5) Міжнародній науково-практичній інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)», 22–23 червня 2023 р., Вінниця, ВНТУ; 6) 8-й Міжнародній науково-практичній конференції «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем (СПРН-2021)», 03–05 листопада 2021 р., Вінниця, ВНТУ; 7) Науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ-2021), 10–12 березня 2021 р., Вінниця, ВНТУ; 8) Міжнародній конференції SPIE «Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2025», грудень 2025 р.; 9) IX Міжнародній науково-технічній конференції «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем (СПРН-2023)», 15–17 листопада 2023 р., Україна, м. Вінниця, ВНТУ.

Публікації на тему дисертації. За матеріалами дисертації опубліковано 21 наукову працю, серед яких: 3 публікації у періодичних виданнях, що індексуються в наукометричних базах Scopus та Web of Science; 9 статей у наукових фахових виданнях України категорії Б; 9 тез доповідей, що відображають апробацію результатів дисертаційної роботи на науково-практичних конференціях.

Структура, зміст і обсяг дисертації. Дисертація складається з реферату, вступу, чотирьох розділів і висновків до кожного з них, висновків, списку використаних джерел і додатків. Загальний обсяг дисертації складає 219 сторінок, з яких основний зміст - 178 сторінок. Дисертація містить 14 таблиць та 156 рисунків. Список використаних джерел налічує 99 найменувань на 13 сторінках. Дисертація містить 2 додатки на 7 сторінках.

РОЗДІЛ 1

АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ПОБУДОВИ ІОТ-СИСТЕМ ЗБОРУ ДАНИХ НА FPGA ТА FPGA-РЕАЛІЗАЦІЙ ХАОТИЧНИХ СИСТЕМ

1.1 Місце FPGA в архітектурі IoT-систем збору даних

Інтернет речей (Internet of Things, IoT) - це концепція, за якою численні сенсори та пристрої, підключені до мережі, збирають та обмінюються даними без участі людини. Типова архітектура IoT передбачає декілька рівнів: сенсорний рівень, на якому розподілені пристрої взаємодіють із фізичним середовищем і збирають дані; проміжний або мережевий рівень, де шлюзи та вузли збору даних агрегують і пересилають інформацію; та верхній рівень - централізовані сервери або хмарні сервіси для глобальної обробки та зберігання (рис. 1.1).

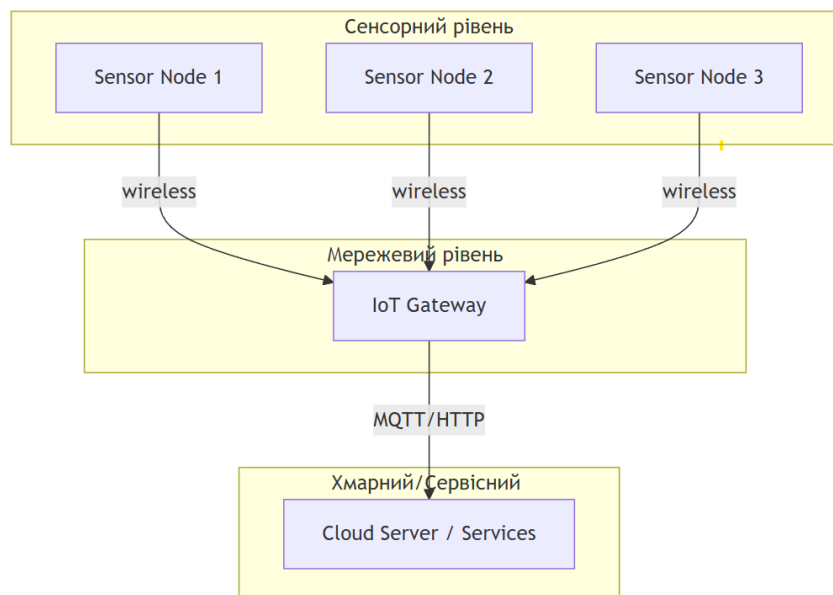


Рисунок 1.1 - Схема типової архітектури IoT: сенсорний рівень, мережевий рівень, хмарний рівень

Сенсорні мережі складаються з великої кількості малопотужних вузлів пристроїв, оснащених датчиками, засобами бездротового зв'язку та

мікроконтролерами для локальної обробки (рис. 1.2). Такі вузли живляться від батарей і повинні функціонувати автономно протягом тривалого часу. Мінімізація енергоспоживання є критичною вимогою.

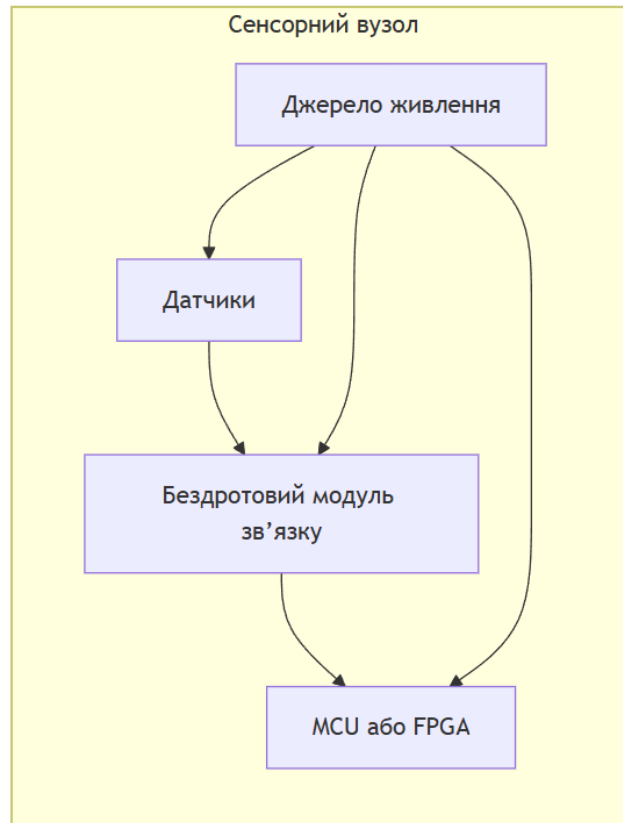


Рисунок 1.2 - Блок-діаграма сенсорного вузла: датчики, бездротовий модуль, MCU, джерело живлення

Традиційно обчислювальним ядром сенсорного вузла слугує мікроконтролер (MCU). Проте зі зростанням вимог до продуктивності периферійних пристроїв IoT постає питання про використання більш гнучких апаратних платформ, зокрема FPGA [22] (Field-Programmable Gate Array) - програмованої логічної інтегральної схеми (рис. 1.3) конфігурацію якої можна багаторазово змінювати після виробництва, реалізуючи паралельні апаратні обчислення.

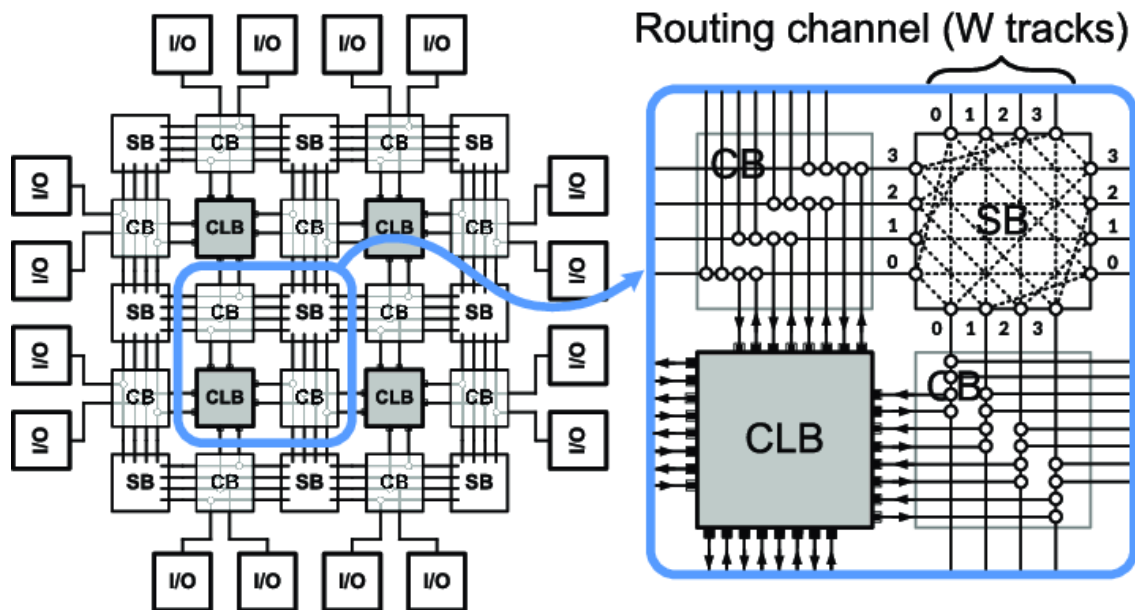


Рисунок 1.3 - Схематична внутрішня архітектура FPGA: програмовані логічні блоки (CLB), блоки підключення (CB), комутаційні блоки (SB), I/O-модулі та маршрутні канали (W-траси) [22].

FPGA заповнює нішу між вузькоспеціалізованими ASIC та універсальними MCU в IoT-архітектурах [23]. З появою енергоефективних серій (наприклад, Intel MAX 10, Lattice iCE40) та вбудованих механізмів енергоменеджменту: clock gating, динамічне масштабування напруги, стало можливим впроваджувати FPGA там, де раніше застосовувалися лише MCU [31]. В огляді 2017 року показано, що FPGA у вузлах IoT забезпечує зниження затримок та більш детермінований час відгуку порівняно з MCU-рішеннями [25]. Ще у 2012 році проведено ґрунтовне порівняння FPGA-орієнтованих та MCU-вузлів у сенсорних мережах, де зафіксовано перевагу FPGA за пропускнуою здатністю при порівнянному енергоспоживанні [29]. FPGA можна задіяти на різних рівнях IoT-систем - від носимих медичних пристроїв [26] до мережевої інфраструктури - для апаратного прискорення обробки та посилення безпеки, причому вони залишаються реконфігурованими після розгортання [23].

Ключові переваги FPGA для IoT-вузлів зводяться до трьох аспектів: апаратний паралелізм, реконфігурованість та детермінований час відгуку. У роботі [24] представлено прототип бездротового сенсорного вузла FlexiS на базі FPGA (рис. 1.4), який напряду порівнювався з аналогічним вузлом на MCU.

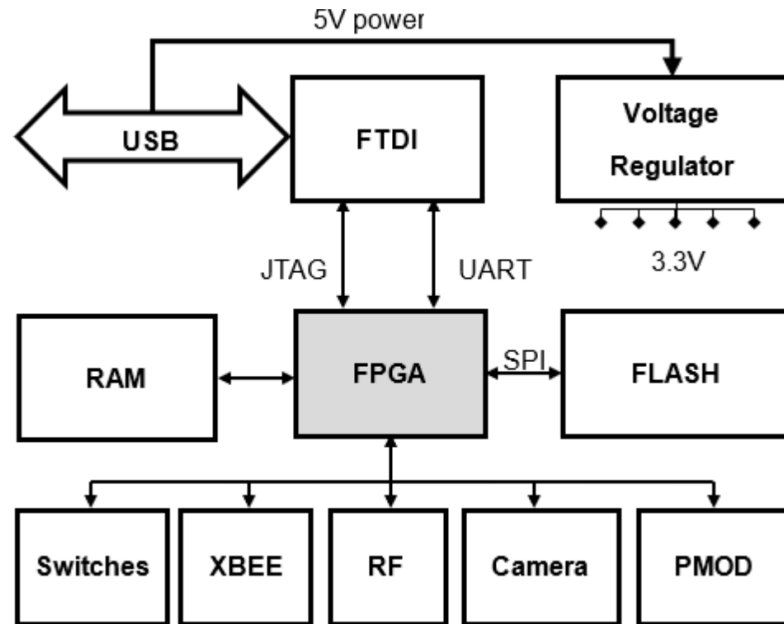


Рисунок 1.4 - Блок-діаграма вузла FlexiS: FPGA-ядро, модуль радіо, сенсорна периферія [24].

FPGA-вузол FlexiS продемонстрував істотно вищу обчислювальну продуктивність за рахунок паралельної обробки даних від кількох різномірних датчиків одночасно. Конфігурацію логіки можна змінювати під вимоги конкретного застосунку без заміни апаратури. Енергоефективність залишалася на конкурентному рівні: вузол швидко виконував задачі і переходив у сплячий режим, скорочуючи сумарне споживання [24].

Подібний підхід реалізовано в архітектурі Мурмес (2023), де малопотужний FPGA використано як інтелектуальну мережеву карту (SmartNIC) поряд з основним MCU (рис. 1.5) [27].

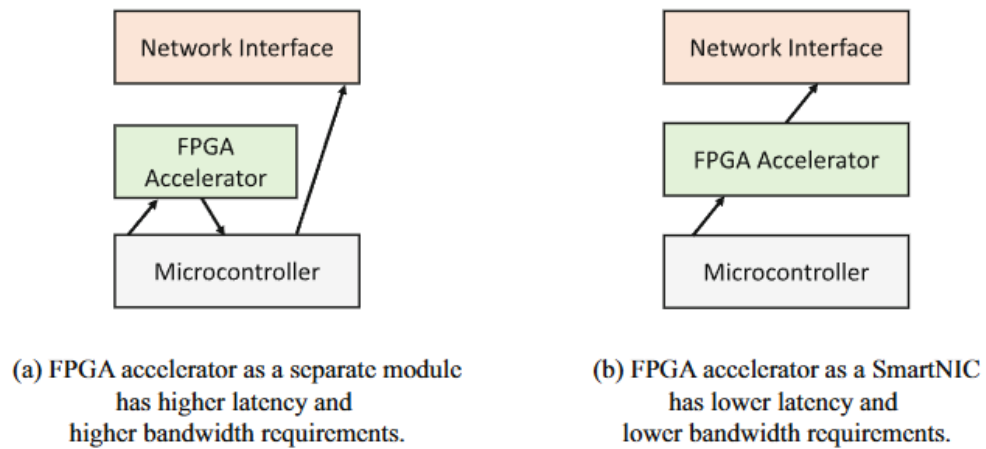


Рисунок 1.5 - Схема SmartNIC: MCU, FPGA (як мережевий прискорювач), трансивер для передачі даних [27].

FPGA-співпроцесор розташовано між мікроконтролером і бездротовим трансивером. Він бере на себе стиснення і фільтрацію даних безпосередньо на рівні вузла, розвантажуючи MCU. За результатами експериментів такий вузол досяг утричі більшої ефективної пропускної здатності порівняно зі звичайним MCU-вузлом, а обсяг бездротової передачі зменшився в десять разів - що, попри додатковий апаратний модуль, знизило загальне енергоспоживання [27].

Щодо енергоефективності FPGA-рішень: апаратний паралелізм дозволяє виконувати обчислення швидше і збільшувати час протягом якого система знаходиться у сплячому режимі. Сучасні FPGA підтримують селективне вимкнення тактування невикористовуваної логіки та динамічне масштабування частоти [31]. Для ресурсоемних криптографічних операцій різниця ще суттєвіша: апаратна реалізація SHA-256 на FPGA досягла пропускної здатності $\sim 1,4$ Гбіт/с при споживанні у 1000 разів нижчому, ніж у програмних реалізаціях на CPU [28]. Це робить можливим виконання криптографічних обчислень безпосередньо на периферійних IoT-вузлах без істотного впливу на автономність.

Реконфігурованість FPGA - можливість змінювати апаратну структуру після розгортання пристрою подовжує життєвий цикл IoT-вузла. У разі появи нових

алгоритмів обробки, зміни протоколів зв'язку чи додавання нестандартних датчиків конфігурацію FPGA можна оновити без фізичної заміни обладнання [23, 25]. Не менш суттєвою є детермінованість: на відміну від MCU, де час виконання варіюється через переривання та планувальники задач, у FPGA кожна операція реалізована як цифрова схема з фіксованим часом затримки. Це критично для задач, де сенсорний модуль повинен гарантовано відреагувати за визначений інтервал часу [25, 30].

Конфігурація апаратної частини IoT-вузла на базі FPGA може бути різною. В одних проєктах FPGA повністю замінює мікроконтролер, як у платформі FlexiS [24]. В інших - застосовується гібридна архітектура, де FPGA виступає співпроцесором поруч із MCU. Наприклад, в архітектурі на базі SoC Xilinx Zynq ARM-ядро виконує задачі загального керування та мережевої взаємодії, тоді як програмована логіка FPGA бере на себе ресурсоемні операції обробки сигналів, що запускаються паралельно [30] (рис. 1.6). Така розподілена архітектура дозволяє отримати переваги обох підходів: FPGA прискорює критичні задачі із мінімальними затримками, а MCU полегшує реалізацію високорівневих функцій і підтримує сумісність із стандартними протоколами [30].

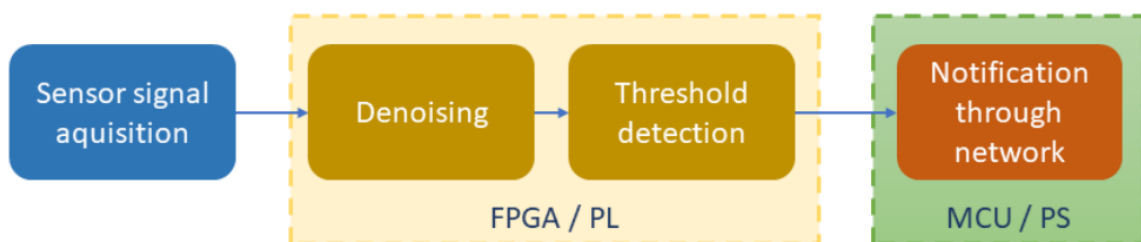


Рисунок 1.6 - Приклад побудови сенсорного вузла IoT із використанням FPGA як співпроцесора [30].

Використання FPGA у сенсорних мережах доцільне у випадках, коли вимоги до паралельної обробки, швидкодії чи детермінованості не можуть бути задоволені традиційними мікроконтролерами. Зокрема, можливість побудови

гібридних SoC-архітектур FPGA з вбудованим soft-core або hard-core процесором відкриває шлях до створення спеціалізованих IoT-вузлів збору даних, здатних одночасно виконувати паралельну обробку сигналів від множини сенсорів і керувати комунікаційними інтерфейсами.

1.2 Частотні сенсори та методи багатоканального вимірювання частоти

Серед різноманітних принципів побудови первинних перетворювачів фізичних величин окреме місце займають сенсори з частотним виходом, у яких інформативним параметром є частота електричного сигналу. На відміну від амплітудних (потенціальних) сенсорів, де інформація кодується рівнем напруги або струму, частотний сигнал за своєю природою є квазіцифровим і не потребує аналого-цифрового перетворення для подальшої обробки мікропроцесорними засобами [32]. Ще у 1961 році П.В. Новицький прогнозував, що клас частотних сенсорів за кількістю відомих реалізацій перевищить клас амплітудних і сучасний стан техніки значною мірою підтверджує цей прогноз [32].

Частотні перетворювачі мають низку принципових переваг перед амплітудними [32, 33]. Завадостійкість частотного сигналу суттєво вища: він може передаватися по двопровідній лінії на значні відстані без втрати інформації, оскільки частота не залежить від затухання амплітуди в каналі зв'язку. Динамічний діапазон перевищує 100 дБ, бо не обмежений напругою живлення чи рівнем шуму. Комутація частотних каналів не вносить суттєвих похибок на відміну від аналогових мультіплексорів, де перехідні опори та паразитні ЕРС спотворюють сигнал [32]. Нарешті, підрахунок імпульсів є ідеальною операцією інтегрування, що працює необмежений час без накопичення похибки дрейфу [32].

До частотних перетворювачів належать кварцові мікробаланси (QCM) з типовими частотами 5-30 МГц, сенсори на поверхневих акустичних хвилях (SAW) з робочими частотами 100 МГц -2,5 ГГц, мікрокантиліверні структури з частотами

в діапазоні кГц-МГц, а також автогенераторні перетворювачі на основі транзисторних структур з від'ємним диференціальним опором [33].

У роботах О.В. Осадчука та співавторів розроблено сімейство частотних перетворювачів фізичних величин на основі транзисторних структур з від'ємним диференціальним опором. Принцип дії цих перетворювачів полягає у формуванні автогенератора, в якому від'ємний опір на колекторно-стоковому переході транзисторної структури компенсує втрати в коливальному контурі, а зміна вимірюваної фізичної величини модулює реактивну складову імпедансу структури, що призводить до зміни частоти генерації. Такий підхід дозволяє уникнути застосування підсилювальних пристроїв та аналого-цифрових перетворювачів, що здешевлює систему вимірювання та підвищує її надійність [34, 35]. Типову електричну схему такого перетворювача наведено на рисунку 1.7.

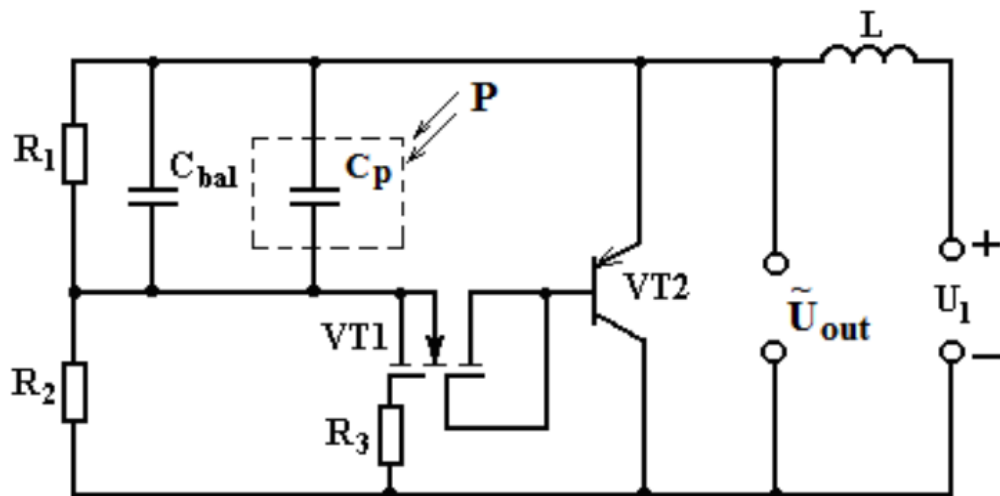


Рисунок 1.7 - Електрична схема частотного перетворювача тиску на основі транзисторної структури з від'ємним опором та МЕМ-конденсатором [35].

На цьому принципі реалізовано частотні перетворювачі тиску з чутливістю 0,95-1,65 кГц/кПа у діапазоні 50-150 кПа та робочими частотами 1,5-1,7 МГц [34, 35]; температури безконтактний ІЧ-термометр з чутливістю 10 кГц/°С у діапазоні 32-43 °С [36] та піроелектричний перетворювач з чутливістю 2500-6500 Гц/°С у

діапазоні 10-1000 °С на частотах 1,5-4,0 МГц [37]; оптичного випромінювання з чутливістю 25-150 кГц/(мкВт/см²) на частотах 550-960 МГц [38]; концентрації газу NH₃ з чутливістю 2,1-3,4 кГц/ppm на частотах 0,53-2,12 МГц [39]; витрати газу з чутливістю 146 кГц/(л/год) на частоті ~850 МГц [40]; вологості з чутливістю 130-333 кГц/% у НВЧ-діапазоні ~810-820 МГц [41]; а також магнітного поля з чутливістю 1,4-2,8 кГц/мТл [42]. Для обробки сигналів таких частотних перетворювачів розроблено відповідні чисельні методи [43].

Характерний вигляд функції перетворення частотного перетворювача тиску наведено на рисунку 1.8 - залежність є близькою до лінійної при відносній похибці моделі не більше $\pm 1,5\%$ [35].

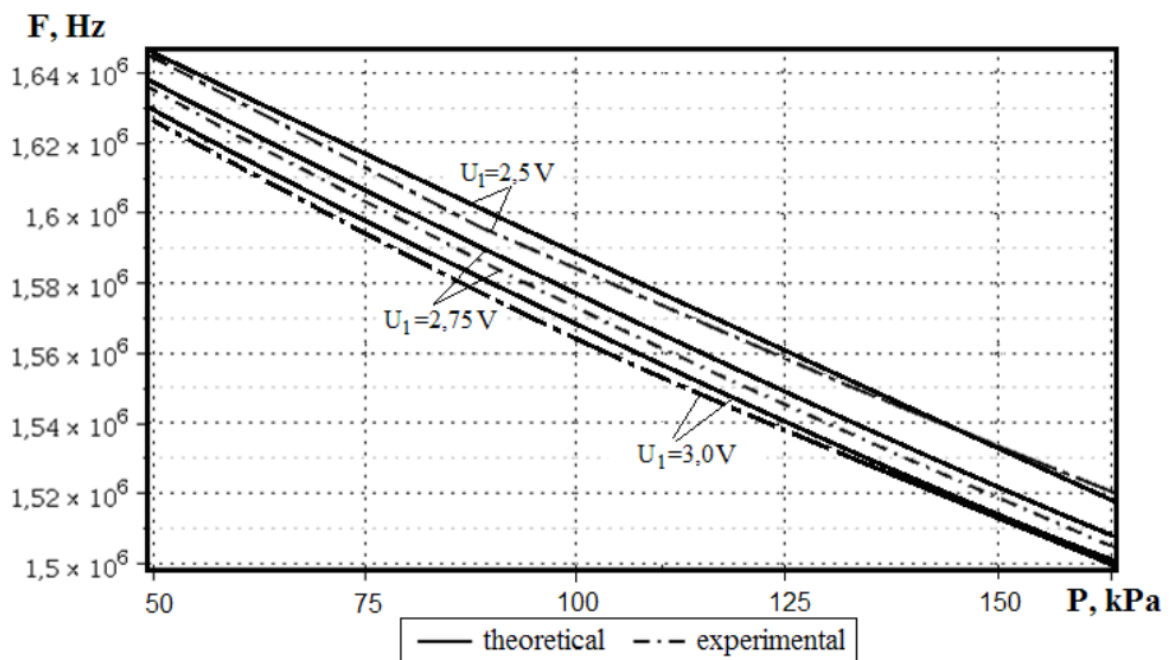


Рисунок 1.8 - Функція перетворення частотного перетворювача тиску: залежність частоти генерації від тиску (теоретична та експериментальна криві) [35].

Зведені характеристики частотних перетворювачів на основі транзисторних структур з від'ємним диференціальним опором наведено у таблиці 1.1.

Таблиця 1.1 - Частотні перетворювачі фізичних величин на основі транзисторних структур з від'ємним опором

Фізична величина	Діапазон вимірювання	Робоча частота	Чутливість	Джерело
<i>Тиск</i>	50–150 кПа	1,5–1,7 МГц	0,95–1,65 кГц/кПа	[35]
<i>Температура (ІЧ)</i>	32–43 °С	1,08–1,16 МГц	10 кГц/°С	[36]
<i>Температура (піро)</i>	10–1000 °С	1,5–4,0 МГц	2,5–6,5 кГц/°С	[37]
<i>Оптичне випромін</i>	0–120 мкВт/см ²	550–960 МГц	25–150 кГц/(мкВт/см ²)	[38]
<i>Конц. газу NH₃</i>	0–1000 ppm	0,53–2,12 МГц	2,1–3,4 кГц/ppm	[39]
<i>Витрата газу</i>	0–4 л/год	~850 МГц	146 кГц/(л/год)	[40]
<i>Вологість</i>	30–99 %	807–822 МГц	130–333 кГц/%	[41]
<i>Магнітне поле</i>	0–80 мТл	275–575 кГц	1,4–2,8 кГц/мТл	[42]

Робочі частоти наведених перетворювачів охоплюють діапазон від сотень кілогерц до одиниць гігагерц. Коли в IoT-системі збору даних потрібно одночасно опитувати масив таких сенсорів, виникає задача паралельного багатоканального вимірювання частоти - тобто одночасного підрахунку імпульсів від кількох незалежних джерел з єдиною часовою базою.

Існують два базових методи цифрового вимірювання частоти [33, 44, 47]. Перший - метод прямого рахунку (conventional counting): за фіксований інтервал часу T_{gate} підраховується кількість імпульсів N_x вимірюваного сигналу, після чого $f_x = N_x / T_{gate}$. Роздільна здатність визначається тривалістю вікна: $\delta f = 1/T_{gate}$, тобто для роздільності 1 Гц потрібне вікно 1 с, для 0,1 Гц - 10 с.

Метод добре працює для високих частот, де за час вікна набирається велике N_x , але для низьких частот точність різко падає. Структуру звичайного лічильника частоти наведено на рисунку 1.9.

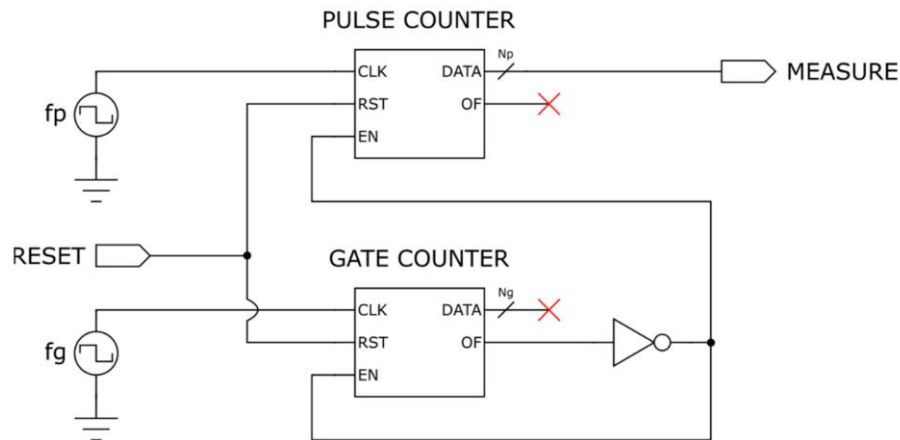


Рисунок 1.9 – Структура звичайного цифрового лічильника частоти [47].

Другий - метод зворотного рахунку (reciprocal counting), або метод рівної точності: вимірюється період сигналу шляхом підрахунку імпульсів опорного генератора N_s за ціле число M періодів вимірюваного сигналу, після чого $f_x = (M / N_s) \cdot f_{ref}$ [44]. Роздільна здатність визначається частотою опорного генератора і не залежить від вимірюваної частоти, що дозволяє досягти рівної точності в усьому діапазоні вимірювання. Структуру лічильника рівної точності наведено на рисунку 1.10 - єдина відмінність від звичайного лічильника полягає у додаванні D-тригера, що синхронізує початок рахунку з фронтом вимірюваного сигналу.

У роботі [44] реалізовано частотомір рівної точності на FPGA Altera Cyclone II з опорним генератором 50 МГц; відносна похибка не перевищила $2 \cdot 10^{-8}$ у діапазоні 100 кГц – 24 МГц. У роботі [47] показано, що максимальна похибка лічильника рівної точності визначається виразом $\varepsilon = (1/n_g) \cdot (f_x^2 / f_{clk})$, де n_g - кількість підрахованих періодів вимірюваного сигналу, f_{clk} - частота опорного генератора. Звідси випливає, що для зменшення похибки ефективні два

шляхи: збільшення f_{clk} та попереднє зниження f_x шляхом перетворення частоти вниз (змішування з опорним сигналом).

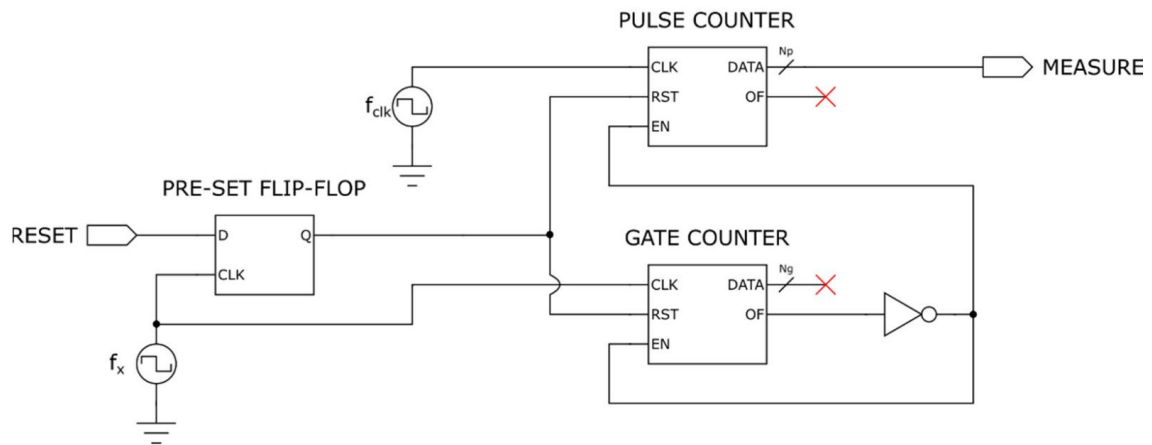


Рисунок 1.10 - Структура лічильника частоти рівної точності з синхронізацією вхідним сигналом [47].

Для задачі одночасного вимірювання частоти від масиву сенсорів критичним є апаратне забезпечення паралельних незалежних лічильників. Кожен канал потребує окремого апаратного лічильника імпульсів, а всі канали мають працювати з єдиною часовою базою для забезпечення порівнянності результатів. Послідовне опитування каналів через мультиплексор порушує синхронність вимірювань - значення, виміряні в різні моменти часу, не підходять для задач кореляційного аналізу.

Мікроконтролери мають обмежену кількість апаратних таймерів і каналів захоплення, які до того ж поділяють спільні ресурси. ESP32 містить 8 модулів PCNT (Pulse Counter) з 16-бітним лічильником, але вони мультиплексовані через спільну шину APB (80 МГц), а максимальна вхідна частота обмежена ~ 40 МГц при вимкненому фільтрі завад і ~ 10 - 20 МГц на практиці. STM32F407 має 14 таймерів, кожен з до 4 каналами Input Capture, але канали одного таймера розділяють спільний лічильник - тобто незалежних одночасних вимірювань лише стільки, скільки таймерів. Максимальна вхідна частота через ETR - ~ 42 МГц для

таймерів на шині APB2. При обробці через переривання (ISR) виникає часове зсув фронтів (jitter), що знижує точність вимірювання.

FPGA принципово вирішує цю проблему. Кожен канал частотоміра - це незалежний апаратний лічильник, синтезований із логічних елементів FPGA. На кристалі Intel Cyclone IV EP4CE15 з 15408 логічними елементами один 32-бітний лічильник з керуванням вікном займає ~16-32 LE, тобто 12 каналів потребують лише ~192-384 LE (~2-4 % ресурсів). Усі канали працюють паралельно, синхронно запускаються та зупиняються від єдиного генератора часового вікна, що гарантує повну синхронність вимірювань. Максимальна вхідна частота обмежена лише швидкодією вхідних буферів: ~200 МГц для LVTTTL, ~400 МГц для LVDS. Порівняння наведено у таблиці 1.2.

Таблиця 1.2 - Порівняння можливостей вимірювання частоти: MCU та FPGA

Параметр	ESP32	STM32F407	Cyclone IV EP4CE15
<i>Апаратних лічильників</i>	8 PCNT (16-біт)	14 таймерів (до 4 IC)	Необмежено (з логіки)
<i>Одночасних незалежних каналів</i>	8 (мультиплексовані)	до 14 (ISR overhead)	Необмежено (з логіки)
<i>Макс. вхідна частота</i>	~40 МГц	~84 МГц (IC), ~42 МГц (ETR)	~200 МГц (LVTTTL)
<i>Синхронність каналів</i>	Ні (послідовний APB)	Частково (спільний лічильник)	Так (спільна часова база)
<i>Детермінованість</i>	Ні (ISR jitter)	Частково (DMA)	Так (апаратна)
<i>Ресурс на 12 каналів</i>	-	-	~2-4 % LE

Аналіз існуючих FPGA-реалізацій частотомірів показує, що більшість опублікованих рішень є одноканальними. У роботі [47] описано одноканальний частотомір зворотного рахунку на Xilinx Artix-7 з попереднім перетворенням частоти вниз для вимірювання частоти QCM-сенсорів; досягнуто роздільність $\sim 0,33$ Гц при тактовій частоті 462,5 МГц. У роботі [45] реалізовано двоканальний частотомір зворотного рахунку на Xilinx Virtex-6 з GPS-синхронізацією часового вікна (1PPS); точність становила 0,2 ppm, роздільність - краща за 0,1 Гц при використанні лише 188 слайсів (1% ресурсів FPGA). Максимальна кількість одночасних каналів, знайдена у відкритих публікаціях, - чотири: у роботі [45] описано чотиріканальний частотомір зворотного рахунку на Xilinx Spartan-6 XC6SLX9 для масиву QCM-сенсорів (рис. 1.11), із роздільністю 0,033 Гц при 10 МГц, що потребував лише 5 % регістрів та 5 % LUT.

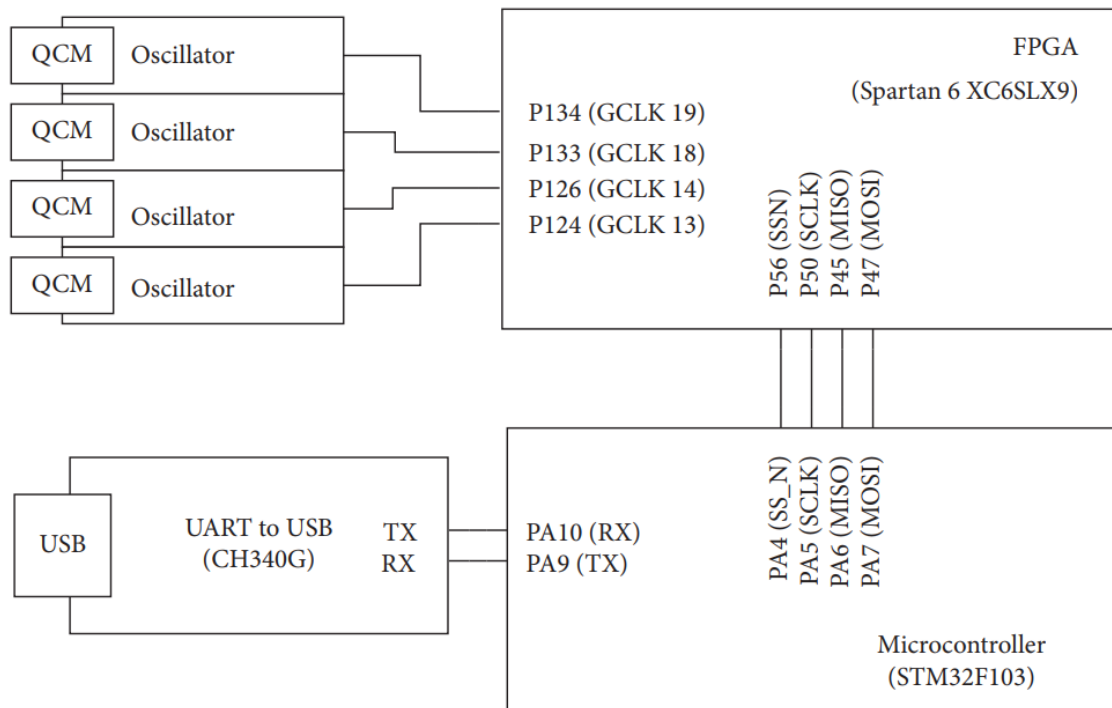


Рисунок 1.11 - Блок-схема чотиріканального FPGA-частотоміра для масиву QCM-сенсорів [45].

Зведені характеристики FPGA-частотомірів наведено у таблиці 1.3.

Таблиця 1.3 - Існуючі FPGA-реалізації частотомірів

Джерело	Рік	Каналів	FPGA	Метод	Макс. частота	Роздільність	Ресурси
[44]	2012	1	Cyclone II	Рівної точності	~100 МГц	$2 \cdot 10^{-8}$ (відн.)	н/д
[47]	2022	1	Artix-7	Зворотн. + змішувач	~50 МГц	0,33 Гц	мінімальні
[46]	2023	2	Virtex-6	Зворотн. + GPS	324 МГц	<0,1 Гц	188 слайсів (1%)
[45]	2023	4	Spartan-6	Зворотного рахунку	~10 МГц	0,033 Гц	5% рег., 5% LUT

Аналіз таблиці 1.3 показує, що існуючі FPGA-частотоміри переважно орієнтовані на прецизійне вимірювання частоти QCM-сенсорів і містять від 1 до 4 каналів. Дванадцятиканальний частотомір із синхронною часовою базою для одночасного паралельного вимірювання частоти від масиву сенсорів у відомій літературі відсутній. Водночас, як показано вище, ресурси навіть бюджетного FPGA Cyclone IV EP4CE15 дозволяють реалізувати 12 і більше незалежних каналів при використанні менше 4% логічних елементів. Водночас самі апаратні лічильники не є завершеною системою збору даних, для конфігурування каналів, математичної обробки результатів та організації передачі даних необхідний контролер. Використання зовнішнього мікроконтролера ускладнює систему та збільшує кількість компонентів, тоді як вбудований soft-core процесор (наприклад, NIOS II) дозволяє об'єднати апаратні лічильники та програмне керування на одному кристалі FPGA, формуючи гібридну архітектуру.

1.3 Сенсорні інтерфейси та технології передачі даних в IoT-мережах

Побудова IoT-системи збору даних на базі FPGA передбачає вирішення двох взаємопов'язаних задач: підключення різнорідних сенсорів до обчислювального вузла через стандартизовані провідні інтерфейси та організацію бездротової передачі зібраних даних до серверної інфраструктури. У попередньому підрозділі показано, що FPGA забезпечує принципову перевагу при паралельному вимірюванні частоти від масиву частотних перетворювачів. Проте частотні перетворювачі лише один із класів вимірювальних пристроїв: більшість сучасних цифрових сенсорів (температури, вологості, інерціальні модулі) видають дані через послідовні інтерфейси I²C, SPI або UART [48, 49]. Типова точність інтегрованих цифрових термометрів становить $\pm 0,5$ °C, датчиків вологості ± 3 %RH, а інерціальні сенсори забезпечують частоту дискретизації від сотень герц і вище [50].

Серед послідовних інтерфейсів підключення сенсорів особливе місце займає шина I²C (Inter-Integrated Circuit) двопровідний синхронний протокол із лініями SDA (дані) та SCL (синхронізація), що підтримує шинну топологію з адресацією підлеглих пристроїв (рис. 1.12) [51]. Стандартний режим I²C працює на частоті 100 кГц, швидкий 400 кГц, а розширені режими Fast-Mode Plus та High-Speed досягають 1 МГц та 3,4 МГц відповідно. Завдяки лише двом сигнальним лініям I²C мінімізує кількість з'єднань, суттєва перевага для компактних сенсорних модулів [53]. Обмеження інтерфейсу полягає у відносно невеликій пропускній здатності, та використанні відкритого колектора з підтягуючими резисторами, що накладає обмеження на ємність шини й довжину ліній.

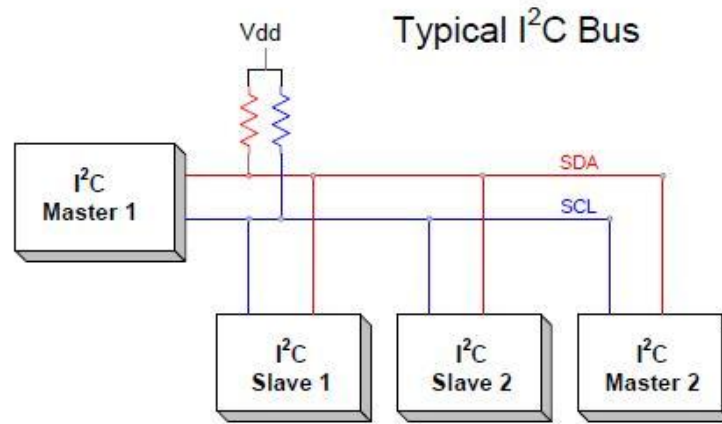


Рисунок 1.12 - Топологія шини I²C із ведучим та підлеглими пристроями [53].

SPI (Serial Peripheral Interface) - синхронний чотирипровідний інтерфейс (MOSI, MISO, SCLK та окремий CS для кожного підлеглого), що забезпечує значно вищу швидкість обміну десятки мегагерц (рис. 1.13) [49]. На відміну від I²C, SPI формує зіркоподібну топологію і не потребує підтягуючих резисторів, проте кількість провідників зростає пропорційно числу підлеглих пристроїв.

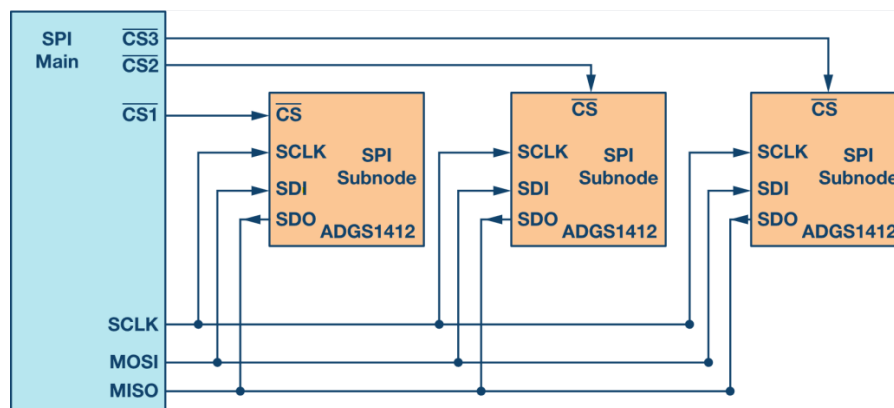


Рисунок 1.13 - Структура 4-провідної шини SPI та розведення сигналів [49].

UART (Universal Asynchronous Receiver-Transmitter) потребує лише двох ліній (TX і RX) для двостороннього асинхронного зв'язку і застосовується переважно для з'єднання «точка-точка» та підключення GPS-приймачів, бездротових модулів або діагностичних портів [49].

I²C підходить для підключення масиву сенсорів невеликої та середньої швидкодії на спільну шину, SPI - для високошвидкісних з'єднань із окремими лініями вибору, а UART - для зв'язку з одним зовнішнім модулем. Більшість сучасних інерціальних сенсорів підтримують обидва інтерфейси - I²C та SPI [49, 50]. Для IoT-вузла із 3-5 цифровими сенсорами помірної швидкодії (температура, вологість, інерціальний модуль) оптимальним є I²C.

На відміну від мікроконтролерів, де апаратні модулі I²C та SPI вбудовано безпосередньо у кристал, в FPGA ці інтерфейси створюються у вигляді внутрішньої логіки. Зазвичай контролер шини будується як скінченний автомат станів (FSM), що послідовно генерує умови старту і стопу, надсилає адресу пристрою, приймає або передає байти даних та обробляє підтвердження (ACK/NACK) [52, 53]. Наприклад, у роботі [53] реалізовано FPGA-контролер I²C на платі Basys 3 (Artix-7), який опитує три цифрових сенсори: TCN75A (температура), HDC1080 (вологість) та TCS3472, по черзі у циклічному режимі. Апаратні витрати мінімальні. У роботі [52] контролер I²C, що одночасно керує трьома різними пристроями (камера OV7620, годинник DS1307, EEPROM 24C04), зайняв лише близько 2% логічних елементів FPGA Spartan-6. Це підтверджує можливість інтеграції інтерфейсних контролерів разом із багатоканальним частотоміром та іншими модулями на одному кристалі.

Розвитком стандарту I²C є шина I³C, запропонована консорціумом MIPI, яка зберігає сумісність на фізичному рівні, але вводить динамічне призначення адрес та переривання від підлеглих пристроїв безпосередньо по лінії даних (In-Band Interrupt) [51]. На цей час I³C обмежено підтримується у промисловій практиці.

Окрім провідних інтерфейсів, IoT-система потребує бездротового каналу для передачі зібраних даних від віддалених вузлів до центрального шлюзу. Для цього у системах з вимогами до великої дальності та низького споживання широко застосовується технологія LoRa (Long Range), що базується на модуляції розширенням спектра лінійною частотною модуляцією (Chirp Spread Spectrum,

CSS) і працює у субгігагерцових ISM-діапазонах: 868 МГц в Європі та 915 МГц у Північній Америці [59, 63]. Фізичний рівень LoRa визначається трьома основними параметрами: коефіцієнтом розширення спектра (Spreading Factor, SF), шириною смуги (Bandwidth, BW) та коефіцієнтом кодування (Coding Rate, CR) [63]. Збільшення SF підвищує дальність та завадостійкість, але знижує швидкість передачі (рис. 1.14), типові значення для LoRa знаходяться в межах від сотень біт/с до десятків кбіт/с. Для передачі телеметрії від сенсорного вузла (~100 байт на цикл) такої пропускної здатності цілком достатньо.

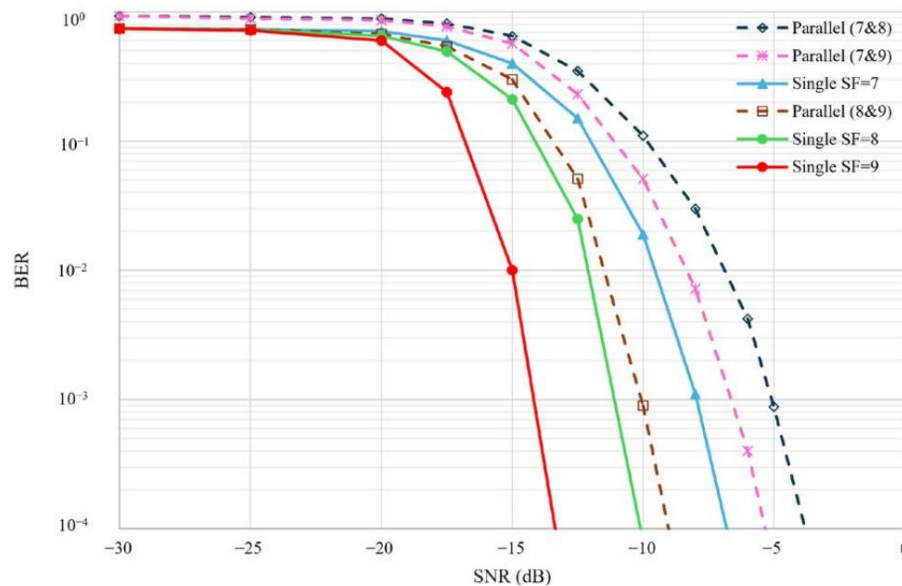


Рисунок 1.14 - Залежність BER від SNR для різних значень коефіцієнта розширення спектра SF [63].

Полюві випробування Kane та ін. продемонстрували високу надійність LoRa-каналу: рівень втрати пакетів не перевищував 1,6% на максимальній тестовій відстані 1000 м (рис. 1.15) [59]. Wi-Fi HaLow (IEEE 802.11ah), що також працює у субгігагерцовому діапазоні, забезпечує значно вищу пропускну здатність (~4985 кбіт/с), проте ефективність цього стандарту суттєво знижується на відстанях понад 200 м [59, 61].

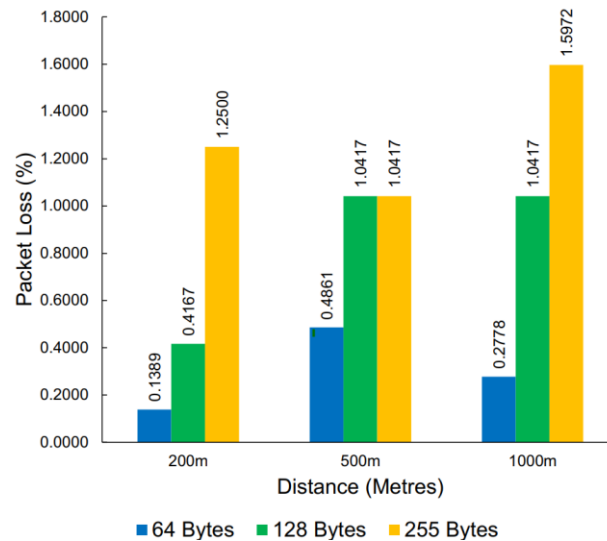


Рисунок 1.15 - Рівень втрат пакетів LoRa при різних відстанях та розмірах пакетів [59].

Енергоспоживання є однією з визначальних переваг технології LoRa для автономних сенсорних вузлів. Емпіричні дослідження комерційних дослідних плат (рис. 1.16) показали середній струм у режимі глибокого сну на рівні 2,50 мкА (плата Grasshopper), а розроблені поліноміальні моделі дозволяють прогнозувати споживання залежно від швидкості передачі, потужності передавача та розміру повідомлення [64].

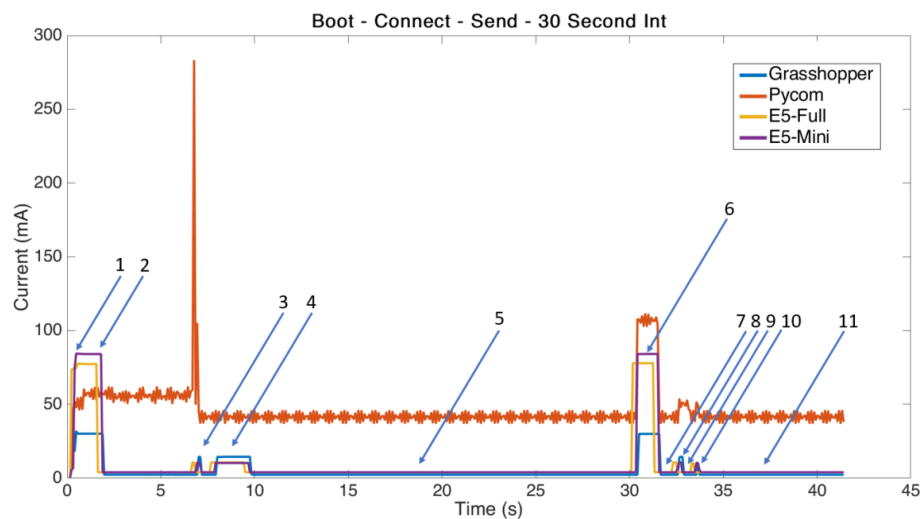


Рисунок 1.16 - Порівняння енергоспоживання дослідних плат LoRa [64].

Подальший розвиток у цьому напрямку пов'язаний із адаптивним керуванням енергетичним бюджетом на основі багатоджерельного збору енергії сонячних панелей та термоелектричних генераторів [65]. Також досліджуються підходи для збільшення пропускної здатності LoRa: використання паралельних приймачів-передавачів із різними SF дозволило досягти приросту бітової швидкості понад 70% [63], а гібридні протоколи забезпечують конвеєрну передачу мультимедійних даних зі швидкістю $\sim 0,3$ с на зображення розміром 24 КБ [62].

Для захисту LoRa-каналу запропоновано легковагові схеми автентифікованого шифрування (AEAD) на основі ChaCha20-Poly1305, що забезпечують конфіденційність та цілісність без помітного впливу на продуктивність зв'язку [58]. FPGA-платформи також застосовуються для обробки LoRa-сигналів зокрема, на платформі Xilinx RFSoc досягнуто точність класифікації понад 99,9% навіть за SNR -15 дБ [60].

Зібрані сенсорним вузлом дані через бездротовий канал надходять до шлюзу, що з'єднує сенсорну мережу з хмарною інфраструктурою (рис. 1.17) [54, 56]. Для організації сенсорної мережі обрано топологію «зірка» із централізованим хабом (шлюзом) та віддаленими вузлами, кожен з яких обмінюється даними безпосередньо з хабом. На відміну від сітчастої (mesh) топології, «зірка» мінімізує складність маршрутизації та енергоспоживання кінцевих вузлів, оскільки не потребує ретрансляції пакетів між вузлами [59]. За своєю структурою типова IoT-архітектура складається з трьох рівнів: сенсорного (периферійні вузли із FPGA та сенсорами), мережевого (шлюз, що виконує перетворення протоколів та маршрутизацію) та хмарного (сервер обробки й зберігання даних) [56]. Шлюз забезпечує агрегацію даних, захист передачі через TLS/DTLS та локальну буферизацію критичної телеметрії на випадок тимчасових розривів зв'язку [54, 58]. Архітектури периферійних обчислень (edge computing) дозволяють перенести частину обробки ближче до сенсорів, зменшуючи навантаження на мережу [55].

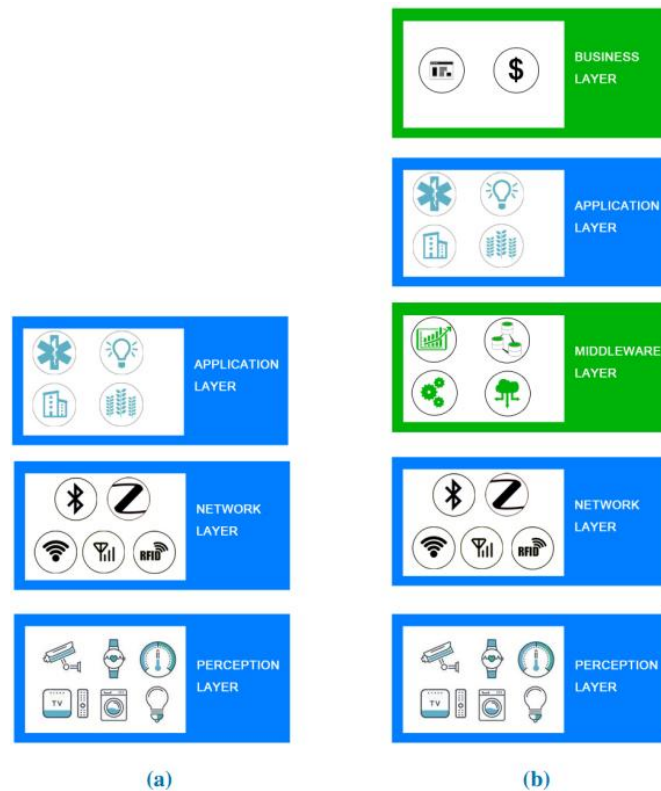


Рисунок 1.17 - Трирівнева архітектура IoT: сенсорний рівень - шлюз - хмара [56].

На хмарному рівні дані зберігаються у базах даних та надаються споживачам через REST API - стандартний інтерфейс взаємодії на основі HTTP/HTTPS. Порівняння протоколів обміну показало, що MQTT забезпечує нижчу затримку (~122-127 мс) та менше навантаження на CPU (1,6-3,3%) порівняно з HTTP (~460 мс, 6-16% CPU), проте REST API залишається найбільш сумісним інтерфейсом для клієнтських застосунків мобільних додатків та веб-панелей моніторингу [57]. Провідні хмарні платформи (AWS IoT Core, Google Cloud IoT, Azure IoT Hub) пропонують готові SDK для інтеграції IoT-шлюзів, включно зі спеціалізованими сервісами периферійних обчислень для запуску логіки безпосередньо на шлюзі [56].

Таким чином, для побудови IoT-системи збору даних на базі FPGA обрано інтерфейс I²C для підключення цифрових сенсорів завдяки шинній топології та

мінімальній кількості провідників, технологію LoRa для бездротової передачі телеметрії завдяки надійності (втрати $<1,6\%$ на дистанції 1 км), наднизькому споживанню ($\sim 2,5$ мкА у режимі сну) та достатній для телеметрії пропускній здатності, топологію «зірка» для мережі сенсорних вузлів, а також трирівневу архітектуру «сенсор - шлюз - хмара» із REST API для доступу до даних. Реалізація ІС-контролера на FPGA потребує лише $\sim 2\%$ логічних елементів, що дозволяє інтегрувати його на одному кристалі разом із багатоканальним частотоміром та іншими модулями обробки.

1.4 Хаотичні системи та їх цифрова реалізація на FPGA

У попередніх підрозділах обґрунтовано вибір FPGA як обчислювальної платформи сенсорного вузла, визначено інтерфейси підключення сенсорів та технологію бездротової передачі даних. Окрім задач збору та передачі телеметрії, FPGA-платформа відкриває можливість апаратної реалізації генераторів детермінованого хаосу, джерел псевдовипадкових сигналів для криптографічного захисту IoT-каналів. Детермінований хаос поєднує дві на перший погляд суперечливі властивості: динаміка системи повністю визначається початковими умовами та параметрами, проте згенеровані часові ряди за статистичними характеристиками наближаються до шумоподібних і задовольняють вимоги стандартних тестів випадковості. Ця подвійна природа робить хаотичні системи привабливими для побудови генераторів псевдовипадкових чисел (ГПВЧ) на IoT-пристроях.

Тривимірні безперервні хаотичні системи мають спільний набір характерних ознак. Дисипативність проявляється у від'ємній дивергенції векторного поля, що означає експоненційне стиснення фазового об'єму: усі траєкторії з часом потрапляють до обмеженої підмножини фазового простору, яка має нульовий об'єм, але ненульову фрактальну розмірність. Фрактальна структура атратора відображає складну багатомасштабну геометрію, породжену механізмом

«розтягнути й скласти»: траєкторії розтягуються вздовж нестійких напрямків і водночас стискаються вздовж стійких. Часові ряди координат у хаотичному режимі мають широкосмуговий спектр без домінування вузьких резонансних піків. Кількісною мірою хаотичності є показники Ляпунова: додатний максимальний показник $\lambda_{\max} > 0$ означає експоненційне розходження близьких траєкторій, тобто мінімальні похибки початкових умов швидко призводять до макроскопічно різних реалізацій. Саме ця чутливість у поєднанні з детермінованістю забезпечує придатність хаотичних систем для генерації псевдовипадкових послідовностей.

Серед численних хаотичних моделей для подальшого аналізу та апаратної реалізації обрано п'ять класичних тривимірних систем, що ілюструють різні фізичні механізми виникнення хаосу: системи Лоренца, Росслера, Нозе-Гувера, Рікітакі та Аніщенко-Астахова. Кожна з цих моделей описується системою трьох звичайних диференціальних рівнянь із квадратичними або кусково-лінійними нелінійностями; їхні математичні моделі та параметри хаотичних режимів детально наведено у підрозділі 4.1.

Система Лоренца, запропонована у 1963 році як спрощений опис атмосферної конвекції, залишається найбільш дослідженою хаотичною моделлю. За класичних параметрів $\sigma = 10, \rho = 28, \beta = 8/3$ фазові траєкторії формують характерний дволопатевий атрактор «метеликоподібної» форми із фрактальною розмірністю $D_{KY} \approx 2.06$ та показниками Ляпунова (0.906, 0, -14.572). Широкосмуговий спектр і чутливість до початкових умов зробили систему Лоренца еталонною моделлю для побудови ГПВЧ та схем хаотичного шифрування (рис. 1.18).

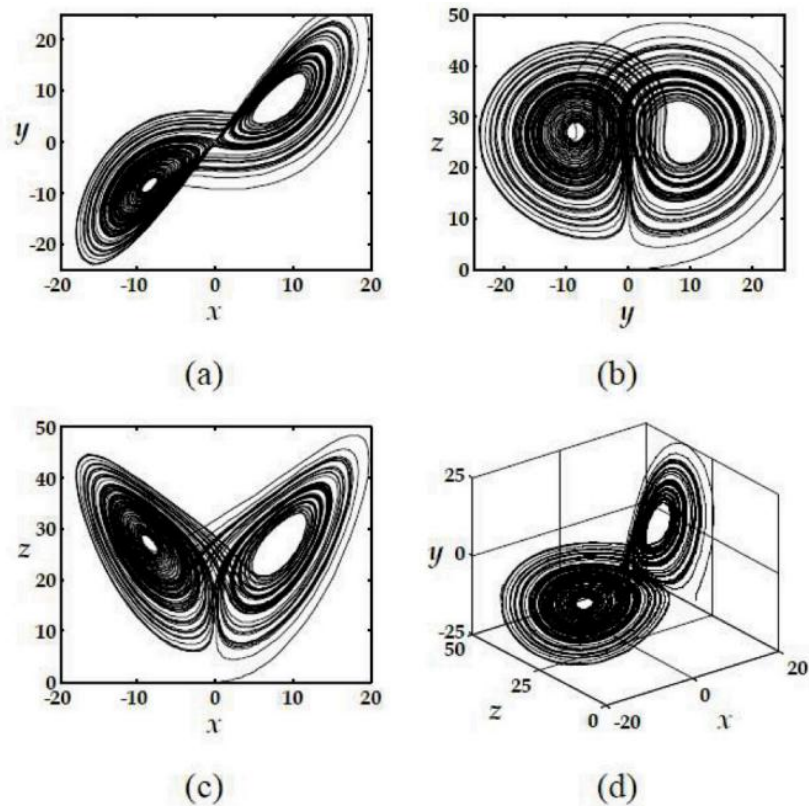


Рисунок 1.18 - Фазові портрети атрактора Лоренца у двовимірних проекціях та тривимірному просторі [70].

Система Росслера, запропонована Отто Росслером у 1970-х роках, є мінімальною тривимірною системою з єдиним квадратичним нелінійним членом. За стандартних параметрів $a = 0.2, b = 0.2, c = 5.7$ вона демонструє спіральний атрактор з показниками Ляпунова $(0.075, 0, -5.372)$. Каскад подвоєнь періоду за параметром c є класичним прикладом переходу до хаосу. Простота рівнянь робить систему Росслера зручною для перших реалізацій на програмованій логіці.

Система Нозе-Гувера описує термостатований гармонічний осцилятор, де змінна z виконує роль динамічного коефіцієнта тертя, що підтримує сталу кінетичну енергію. Параметр α задає інерційність термостата: при його зростанні система переходить від регулярних коливань через квазіперіодичні режими до хаотичних вікон. На відміну від решти розглянутих моделей, система Нозе-Гувера

є часово-оберненою, що робить її корисною для дослідження зв'язку між мікроскопічною оборотністю та макроскопічним хаосом.

Система Рікітакі моделює дво-дискове динамо з перехресним зв'язком контурів і застосовується для пояснення нерегулярних реверсій геомагнітного поля. Параметр a описує електромеханічне демпфування, p - різницю кутових швидкостей дисків; хаотичний режим виникає при $p \approx 3.46$. Система має симетрію щодо зміни знака струмів, що породжує двоцентровий аттрактор з хаотичними переключеннями між двома рівноважними конфігураціями «динамо».

Модифікована система Аніщенка-Астахова, на відміну від перелічених абстрактних моделей, описує реальний радіотехнічний пристрій - автоколивальний генератор з інерційною нелінійністю, побудований на транзисторній структурі з від'ємним диференціальним опором. Детальний аналіз цієї системи, включно з фізичною інтерпретацією, спектральними характеристиками та оглядом існуючих досліджень, наведено далі у цьому підрозділі.

Вибір саме цих п'яти систем обґрунтовано їхньою роллю в ілюстрації різних механізмів хаотичної динаміки: конвекційний хаос (Лоренц), мінімальний хаос із єдиною нелінійністю (Росслер), термостатований хаос (Нозе-Гувер), геомагнітний хаос (Рікітакі) та радіотехнічний хаос з інерційним зворотним зв'язком (Аніщенка-Астахова). Разом ці моделі утворюють репрезентативний набір тривимірних хаотичних систем, що охоплює як добре досліджені в апаратному плані системи (Лоренц, Чуа), так і ті, для яких FPGA-реалізації відсутні.

Практична мотивація до апаратної реалізації хаотичних генераторів на FPGA підтверджується результатами досліджень у галузі захисту IoT-каналів. У роботі [66] запропоновано високопродуктивне хаотичне шифрування на FPGA Intel Stratix 10 із використанням фізично неклоновної функції (PUF) для генерації ключів: п'ятимірна гіперхаотична система Лоренца шифрує дані операцією виключного АБО (XOR) зі швидкістю до 60 МБ/с при споживанні менше 0,4 Вт,

забезпечуючи простір ключів понад 2^{500} . Інший підхід - інтеграція хаотичних генераторів у існуючі криптосистеми: у [67] дискретне відображення вбудовано у полегшену версію поточного шифру ZUC (алгоритм LZUC), що забезпечило економію логічних ресурсів FPGA на 30-38% при збереженні швидкодії (такт $\sim 16,8$ нс) та проходженні всіх 15 тестів NIST із ентропією 0,9999. Комплексний підхід SCBLC, описаний у [68], поєднує генератор псевдохаотичних чисел на основі слабо зв'язаних логістичного та 3D-чебишевського відображень із блочним шифром Speck64/128; апаратна реалізація на FPGA Xilinx PYNQZ2 продемонструвала $NPCR = 99,609\%$, $UACI = 33,464\%$ та споживання ~ 12 мВт, підтверджуючи придатність хаотичних генераторів для енергообмежених IoT-вузлів.

Наведені приклади засвідчують, що FPGA-реалізація хаотичних систем є активним напрямком досліджень із безпосереднім прикладним значенням для захисту IoT-комунікацій. Для систематичного аналізу поточного стану цих досліджень далі виконано огляд існуючих FPGA-реалізацій хаотичних осциляторів.

Для визначення поточного стану досліджень у галузі цифрової реалізації хаотичних систем на програмованій логіці проведено систематичний пошук у наукометричних базах Google Scholar, IEEE Xplore та Semantic Scholar за період 2000–2026 рр. за ключовими словами «FPGA chaotic system implementation», «FPGA chaotic oscillator», «digital chaos generator FPGA». Загальну картину досліджень відображено в оглядовій роботі Tasdemir M.F., Tuna M. та Koyuncu I. [69], де проаналізовано 23 FPGA-реалізації хаотичних осциляторів за 2016–2025 рр. з порівнянням за максимальною робочою частотою, типом чисельного методу та областю застосування. На основі цього огляду та власного пошуку сформовано зведену таблицю реалізацій для хаотичних систем, обраних у цій роботі (табл. 1.4).

Таблиця 1.4 - FPGA-реалізації хаотичних систем (за результатами систематичного пошуку)

Хаотична система	Автори	Рік	FPGA	Метод	Арифметика	Fmax, МГц
<i>Лоренца</i>	Damaj et al. [70]	2024	Stratix IV	Euler/RK4/RK6	плав. 8–64 біт	800
<i>Лоренца (4D гіпер)</i>	Sadoudi et al. [71]	2013	Virtex-II Pro	RK4	фікс. 16Q16	25
<i>Росслера</i>	Schmitz, Zhang [72]	2017	Zynq-7000	Euler	фікс. Q10.9	71
<i>Чена (PCO)</i>	Kemdoum et al. [73]	2024	Artix-7	RK4	фікс. 32Q20	51
<i>Чуа</i>	Renteria et al. [74]	2024	Spartan 3E	Euler	фікс. Q6.25	50
<i>Чуа (багатозвивна)</i>	Al-Musawi et al. [75]	2021	Artix-7	Euler	фікс. 24 біт	25
<i>Jerk (3D)</i>	Vaidyanathan et al. [76]	2023	Zybo Z7-20	Euler	фікс. 24 біт	111
<i>Jerk (з перемиканням)</i>	Bonny, Elwakil [77]	2018	Zynq-7000	-	-	173
<i>Лю (Lü-Chen)</i>	Tuna et al. [78]	2019	Virtex-6	Heun	фікс. IQ-Math 32	465
<i>Нозе-Гувера</i>	Garg et al. [79]	2021	Kintex-7	Euler/Heun/RK4	-	-
<i>Рікітакі</i>	-	-	-	-	-	-
<i>Аніщенко-Астахова</i>	-	-	-	-	-	-

Типовий приклад апаратної архітектури FPGA-хаотичного осцилятора наведено на рисунку 1.19, конвеєрна структура 3D-системи Jerk, де кількість множників зменшено з 9 до 4 завдяки мультиплексуванню та скінченному автомату станів (FSM) [76].

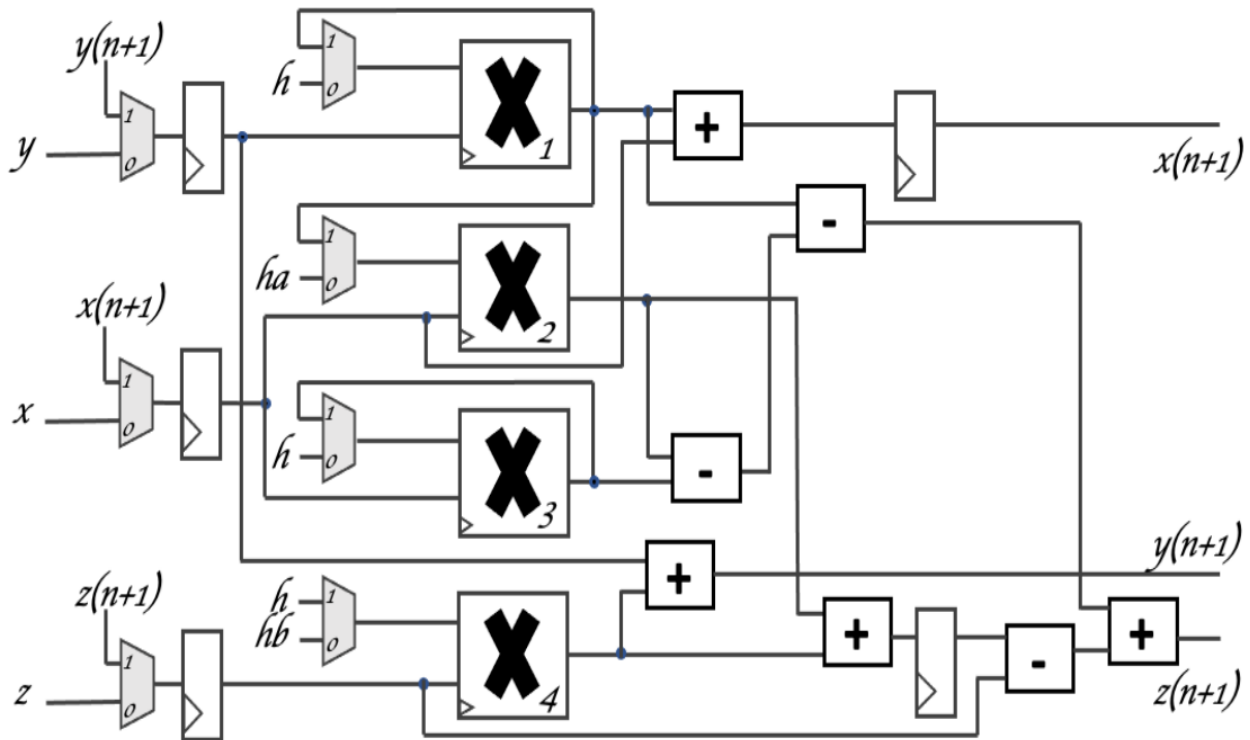


Рисунок 1.19 - Структурна схема конвеєрної FPGA-реалізації 3D-системи Jerk [76].

Аналіз таблиці 1.4 дозволяє виділити декілька характерних тенденцій. По-перше, переважна більшість реалізацій використовує арифметику з фіксованою точкою (fixed-point) із розрядністю 24-32 біти - це забезпечує баланс між точністю відтворення хаотичної динаміки та ефективністю використання ресурсів FPGA. Єдиним винятком є робота Датаї та ін. [70], де реалізовано систему Лоренца у форматі IEEE 754 з плаваючою точкою (8-64 біти) на потужній FPGA Intel Stratix IV; при цьому досягнуто максимальну робочу частоту 800 МГц та пропускну

здатність до 21,17 Гбіт/с, проте ціною значних апаратних витрат - до 43 749 LUT і 810 DSP-блоків.

По-друге, серед чисельних методів інтегрування домінує метод Ейлера (Forward Euler), який використано у шести з десяти розглянутих реалізацій. Метод Ейлера є найпростішим для апаратної реалізації він потребує лише одного обчислення правої частини диференціального рівняння на кожному кроці, що мінімізує кількість множників та суматорів. Наприклад, у роботі Renteria та ін. [74] реалізація контуру Чуа методом Ейлера на Spartan 3E зайняла лише 16 LUT та 29 тригерів при робочій частоті 50 МГц - менше 1% ресурсів FPGA. Для криптографічних застосувань, де точність відтворення фазової траєкторії є критичною, застосовується метод Рунге-Кутта 4-го порядку (RK4): у роботі Kemdoun та ін. [75] RK4-реалізація збуреного осцилятора Чена на Artix-7 (1527 LUT, 8 DSP, 112 мВт) пройшла всі 15 тестів NIST SP 800-22 та обидва набори TestU01 (Rabbit 40/40, Alphabit 17/17). Порівняння результатів чисельного моделювання з осцилограмами, отриманими безпосередньо з FPGA (рис. 1.20), підтверджує коректність апаратної реалізації для різних значень параметра d .

По-третє, системи Лоренца та Чуа є найбільш дослідженими для кожної з них у таблиці представлено по дві реалізації, а в оглядовій роботі [69] кількість публікацій значно більша. Системи Росслера, Чена, Jerk, Лю-Чена та Нозе-Гувера мають по одній-дві FPGA-реалізації. Водночас для систем Рікітакі та Аніщенко-Астахова FPGA-реалізацій у відомій літературі не виявлено. Для системи Рікітакі існують лише програмні моделі (MATLAB), а для системи Аніщенко-Астахова теоретичні роботи та аналогові схеми на дискретних компонентах, що детально розглянуто далі.

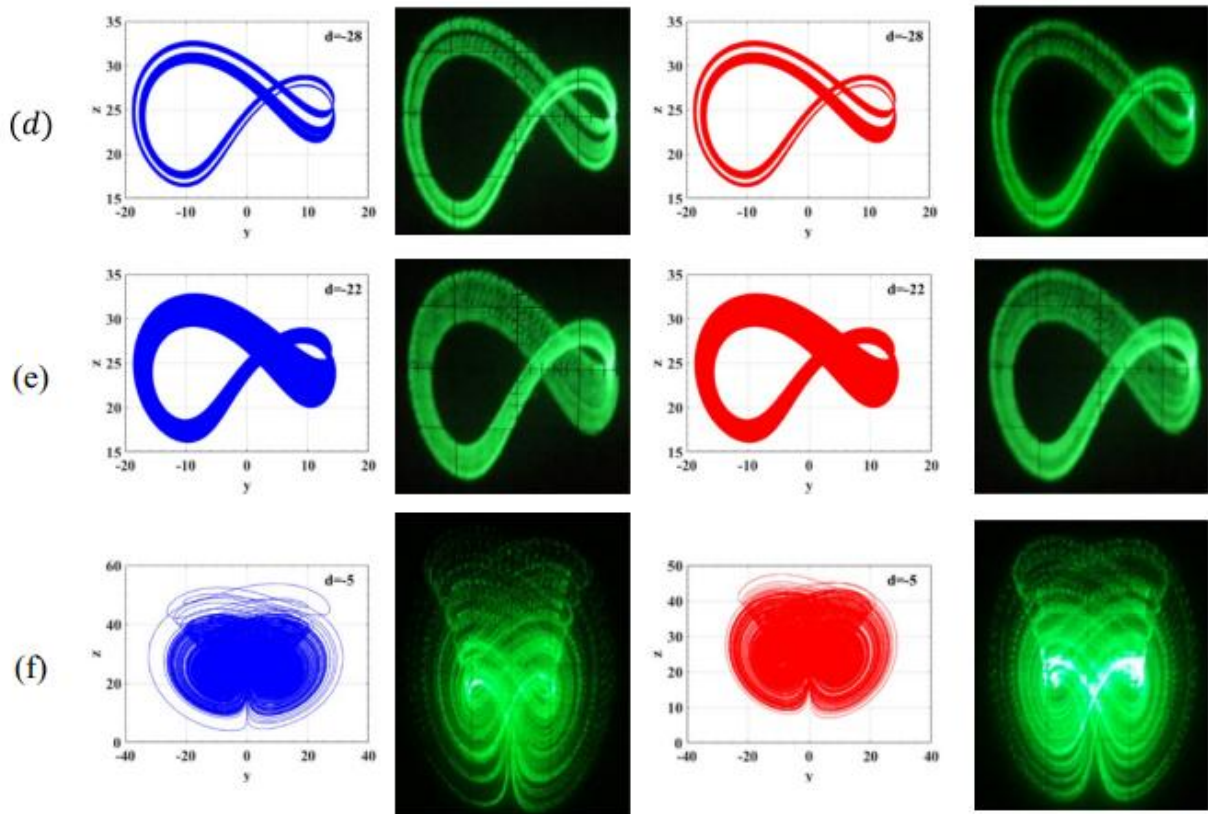


Рисунок 1.20 - Порівняння фазових портретів осцилятора Чена: чисельне моделювання (MATLAB) та осцилограми з FPGA Artix-7 для різних параметрів [73]

Окремо зазначимо застосування FPGA для обробки та класифікації хаотичних сигналів. Наприклад, у роботі Schmitz та Zhang [72] реалізовано систему захищеного зв'язку на основі синхронізації двох генераторів Росслера на Zynq-7000 із пропускнуою здатністю 2850 Мбіт/с (19-бітна арифметика з фіксованою точкою). Sadoudi та ін. [71] продемонстрували бездротову передачу шифрованого відео (25 кадрів/с, 128×128 пікселів) через XВее-модулі із використанням 4D-гіперхаотичної системи Лоренца на Virtex-II Pro, де NIST-тестування підтвердило якість шифрування (NPCR = 99,586%, усі 18 тестів пройдено). Реалізація багатозвивного генератора Чуа [75] на Artix-7 показала, що ускладнення атрактора (від 2 до 4 звитків) збільшує використання LUT з 761 до 1042 при збереженні частоти ~22-25 МГц (рис. 1.21).

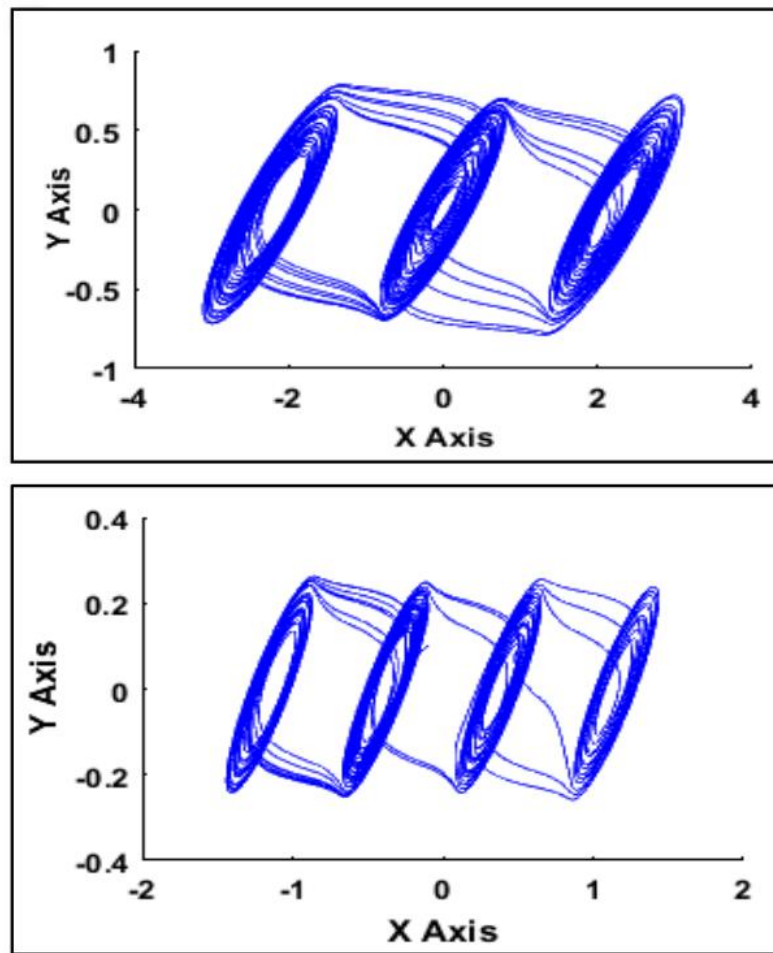


Рисунок 1.21 - Багатозвивні атрактори Чуа (3, 4 звитки), отримані апаратним косимулюванням на FPGA Artix-7 [75].

У роботі Vaidyanathan та ін. [76] нову 3D-систему Jerk з мультистабільністю (показники Ляпунова $L1 = +0,1176, L2 = 0, L3 = -1,1176; D_{KY} = 2,1052$) реалізовано на Zybo Z7-20 із конвеєризацією: 4 множники замість 9 при прямій реалізації, 2 такти на ітерацію, робоча частота 111 МГц при витратах лише 134 LUT.

Відсутність FPGA-реалізацій для системи Аніщенка-Астахова є ключовим висновком проведеного огляду і потребує детальнішого аналізу стану досліджень цієї системи.

Система Аніщенко-Астахова належить до класу автоколивальних генераторів із інерційною нелінійністю та є однією з базових тривимірних моделей детермінованого хаосу в радіофізичних системах [80]. На відміну від абстрактних математичних конструкцій (конвекційна модель Лоренца, мінімальний осцилятор Росслера), ця система описує конкретний радіотехнічний пристрій генератор на транзисторній структурі з від'ємним диференціальним опором. Фазова змінна $x(t)$ відповідає напрузі в коливальному контурі, $y(t)$ - індуктивній складовій (швидкості), а $z(t)$ - стану інерційного перетворювача у колі зворотного зв'язку. Параметр m характеризує ефективне підсилення активного елемента та визначає ступінь від'ємного опору, тоді як параметр g задає швидкість релаксації інерційної змінної z .

Математична модель системи А-А задається системою трьох звичайних диференціальних рівнянь (детально наведених у підрозділі 4.1), що містять три нелінійні функції: $G(x)$ - нормована апроксимація вольт-амперної характеристики (ВАХ) приладу з від'ємним опором, $F(x)$ - кубічний поліном апроксимації ВАХ напівпровідникового діода та $I(x)$ - індикаторна функція одnobічної провідності. Принципово важливою є функція $G(x)$, яка у практичних реалізаціях апроксимується гладкою гіперболічною функцією типу $\tanh(\cdot)$ - трансцендентною нелінійністю, коефіцієнти якої підбираються за експериментально виміряними ВАХ транзисторної структури з від'ємним опором. Наявність цієї трансцендентної функції відрізняє систему А-А від усіх інших розглянутих хаотичних моделей, де нелінійності мають поліноміальний або кусково-лінійний характер, і створює додаткову задачу при апаратній реалізації необхідність ефективного обчислення $\tanh$ на FPGA, методи якого розглянуто у підрозділі 1.5.

Хаотичні коливання у системі А-А спостерігаються при певних поєднаннях параметрів m , g та d . Для модифікованої моделі, орієнтованої на генератор з від'ємним опором λ -типу, хаотичний режим отримано при $m = 1,117, d = 0,01, g = 2,0$. Спектральні характеристики сигналів у цьому режимі відповідають

широкопasmовому детермінованому хаосу: амплітудно-частотні спектри мають вигляд безперервного «гребеня» з розмитими лініями, де енергія розподілена у широкому діапазоні частот. Додатний максимальний показник Ляпунова підтверджує експоненційну чутливість до початкових умов. Поєднання шумоподібності з детермінованою природою джерела робить систему А-А перспективною для генерації псевдовипадкових послідовностей.

Пряма радіотехнічна інтерпретація системи А-А тісно пов'язана з дослідженнями Осадчука О.В. та співавторів у галузі генераторів на транзисторних структурах з від'ємним диференціальним опором [9]-[12]. У цих роботах досліджено автоколивальні процеси в біполярно-польових транзисторних структурах, виконано аналіз ВАХ та побудовано аналогові моделі генераторів, які демонструють хаотичну поведінку. Експериментальне підтвердження хаотичних режимів отримано у лабораторних умовах на дискретних елементах та в середовищі MATLAB. Зв'язок між транзисторною структурою з від'ємним опором та математичною моделлю Аніщенка-Астахова полягає у тому, що модифікована система А-А адекватно описує динаміку саме таких генераторів, де функція $G(x) \approx \tanh(\cdot)$ апроксимує реальну ВАХ активного елемента. на рисунку 1.22 наведено сімейство експериментальних ВАХ транзисторної структури на основі біполярних транзисторів BFT93 та BFR92, отриманих у роботі [9]: характерна N-подібна форма кривих із ділянкою від'ємного диференціального опору є фізичним підґрунтям нелінійності $G(x)$ у моделі А-А.

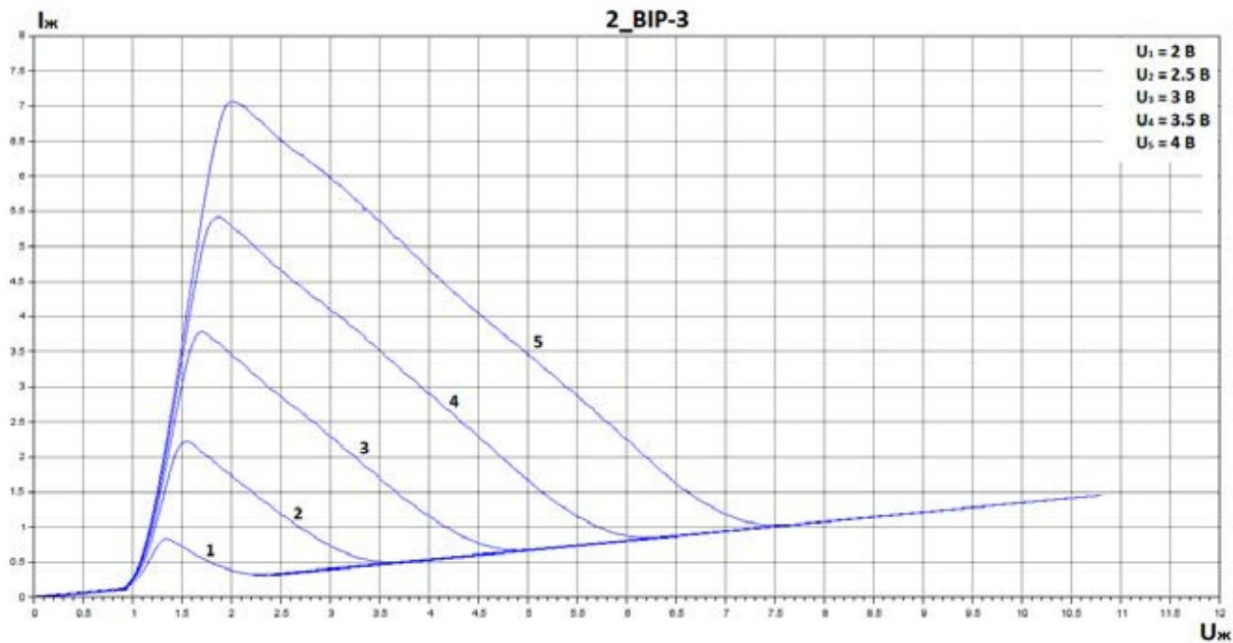


Рисунок 1.22 - Сімейство експериментальних ВАХ транзисторної структури з від'ємним диференціальним опором при різних керуючих напругах [9].

Експериментальний фазовий портрет хаотичного атрактора, отриманий на осцилоскопі у ХУ-режимі з реального генератора на транзисторній структурі (рис. 1.23), підтверджує наявність детерміновано-хаотичних коливань спірального типу, що відповідають теоретичним передбаченням моделі А-А [10].

Аналіз існуючих досліджень системи А-А підтверджує відсутність цифрових реалізацій. Монографія Anishchenko V.S. та ін. [80] є фундаментальною теоретичною роботою, де виконано повний аналіз біфуркацій, показників Ляпунова та ергодичних властивостей системи. У 2022 році Hu C. та ін. [81] запропонували модифікацію системи А-А із заміною кусково-лінійної функції на $|x|$ та реалізували її виключно на аналогових компонентах операційних підсилювачах AD711JN, діодах 1N4007 та аналогових множниках AD633JN.

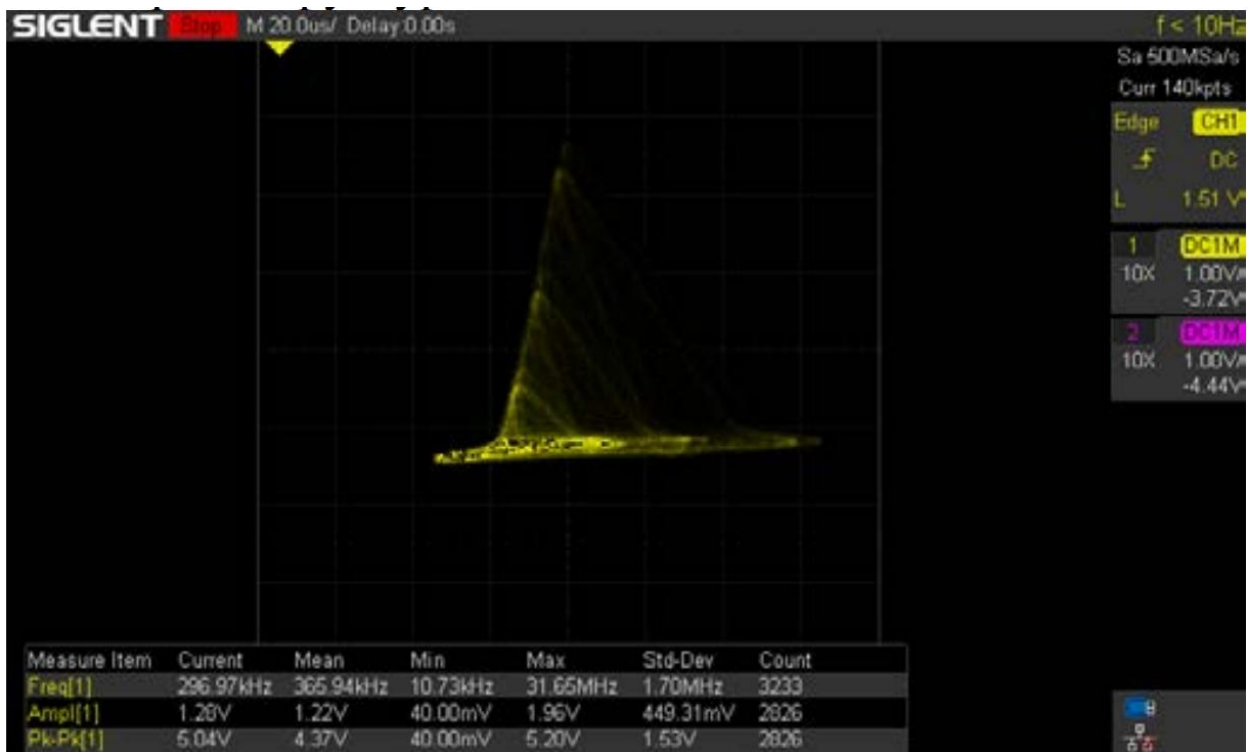


Рисунок 1.23 - Експериментальний фазовий портрет хаотичного атратора генератора на транзисторній структурі з від'ємним опором (осцилоскоп, XY-режим) [10].

Kilina A. та ін. [82] у 2024 році дослідили неавтономну модифікацію генератора А-А під дією зовнішніх коротких імпульсів та виявили сценарії виникнення гіперхаосу через ієрархію атракторів Шильнікова, однак ця робота обмежується теоретичним аналізом без будь-якої апаратної реалізації. Жодна з виявлених публікацій не містить цифрової FPGA-реалізації системи Аніщенко-Астахова.

Таким чином, аналіз показав, що, незважаючи на значну кількість робіт із FPGA-реалізації хаотичних систем (Лоренца, Росслера, Чена, Чуа, Jerk, Лю-Чена, Нозе-Гувера), цифрова реалізація системи Аніщенко-Астахова на FPGA у відомій літературі відсутня. Наявні дослідження цієї системи обмежуються теоретичним аналізом [80, 82] та аналоговими схемами на дискретних компонентах [81], а експериментальне підтвердження хаотичних режимів виконано лише в

лабораторних умовах [9]-[12]. Водночас система А-А має пряму радіотехнічну інтерпретацію та відрізняється від інших хаотичних систем наявністю трансцендентної нелінійності $G(x) \approx \tanh(\cdot)$, що потребує спеціального підходу до апаратної реалізації. Це визначає актуальність розробки цифрової FPGA-реалізації системи Аніщенка-Астахова, яку представлено у подальших розділах роботи.

1.5 Методи реалізації нелінійних функцій на FPGA

У підрозділі 1.4 встановлено, що система Аніщенка-Астахова містить трансцендентну нелінійність $G(x) \approx \tanh(\cdot)$, яка відрізняє її від інших розглянутих хаотичних моделей із поліноміальними або кусково-лінійними нелінійностями. Для апаратної реалізації системи А-А на FPGA необхідно обрати метод обчислення гіперболічного тангенса, який задовольняє вимоги щодо швидкодії, точності та ресурсоемності. У літературі представлено чотири основні підходи до апаратної реалізації трансцендентних функцій на програмованій логіці [83, 84, 85].

Алгоритм CORDIC (Coordinate Rotation Digital Computer) обчислює $\tanh(x)$ через відношення $\sinh(x)/\cosh(x)$, де гіперболічні синус та косинус визначаються ітераційними зсувами й додаваннями у гіперболічному координатному просторі з подальшим діленням у лінійному режимі CORDIC [83]. Перевагою алгоритму є відсутність апаратних множників кожна ітерація потребує лише зсуву та додавання, тобто обходиться без DSP-блоків. За даними Sorayaasa [83], реалізація CORDIC для $\tanh$ на FPGA Artix-7 при 14-бітній арифметиці з фіксованою точкою досягає максимальної абсолютної похибки 0,0012 та середньоквадратичну похибку (RMSE) 0,0015 за 13 ітерацій. Однак ціною високої точності є значна затримка - 36,3 нс (максимальна робоча частота лише 27,5 МГц), а використання ресурсів становить 2,20% LUT та 0,42% тригерів найбільше серед

розглянутих методів. Hajduk [85] наводить ще більшу затримку для CORDIC: до 2700 нс при 64-бітній точності.

Найшвидший доступ за 1 такт забезпечує метод таблиці (LUT), що полягає у зберіганні попередньо обчислених значень $\tanh(x)$ у пам'яті ROM. Точність визначається розміром таблиці: за даними [83], таблиця з 50 точками та лінійною інтерполяцією забезпечує максимальну похибку 0,003, а з 10 точками 0,065. Ресурси мінімальні (0,04-0,07% LUT), проте для високої точності потрібен великий обсяг пам'яті: повна таблиця для формату Q16.16 потребувала б 2^{32} записів, що не реалізовано на практиці. Компромісом є таблиця з діапазонною адресацією та лінійною інтерполяцією (Range Addressable LUT, RALUT), яка при 50 точках дає похибку 0,001 при витратах 0,04% LUT [83].

Поліноміальна апроксимація використовує розклад $\tanh(x)$ у степеневий ряд (Тейлора, Маклорена або Чебишева) з обчисленням за схемою Горнера. Hajduk та Дес [84] реалізували пряму та чебишевську апроксимацію $\tanh$ на Artix-7 з поділом діапазону на 15-244 інтервали. Для арифметики з фіксованою точкою (32-бітна дробова частина) поліном 3-го степеня на 59 інтервалах досягає похибки $3,7 \times 10^{-8}$ із затримкою 59,8 нс та витратах 514 LUT і 4 блоки цифрової обробки сигналів (DSP). Поліноми вищих степенів (5-8-й) забезпечують точність до 10^{-10} - 10^{-11} , проте потребують 12-18 тактів та 4-8 DSP-блоків. Цей метод є оптимальним для задач, де критична точність (наприклад, навчання нейронних мереж зі зворотним поширенням), однак потребує апаратних множників, відсутніх у найпростіших серіях FPGA.

Кусково-лінійна апроксимація (PWL) розбиває діапазон вхідних значень на N сегментів, у кожному з яких функція апроксимується лінійною залежністю $y = a \cdot x + b$. Для обчислення потрібно одне множення та одне додавання, що забезпечує затримку 1-2 такти. За даними [83], PWL з 9 сегментами на Artix-7 дає максимальну похибку 0,068 та RMSE = 0,025 при витратах 0,45–1,5% LUT, нульовому використанні DSP та RAM і максимальній частоті ~200 МГц найвищій

серед усіх розглянутих методів. Лі та ін. [86] запропонували вдосконалений варіант PWL із розбиттям за аналізом кривизни функції (PWLC): при 9 сегментах для sigmoid досягнуто максимальну похибку 0,0078 та затримку 9,6 нс на Virtex-7 без використання DSP-блоків. Зі збільшенням кількості сегментів точність покращується, а ресурси зростають лінійно (від 18 суматорів для 3 сегментів до 43 для 9 при 16-бітній арифметиці) [83]. Як показано на рисунку 1.24, залежність похибки апроксимації від кількості точок розбиття має характерне «коліно»: після 8-10 сегментів подальше збільшення їх кількості дає незначне покращення точності при пропорційному зростанні апаратних витрат [86].

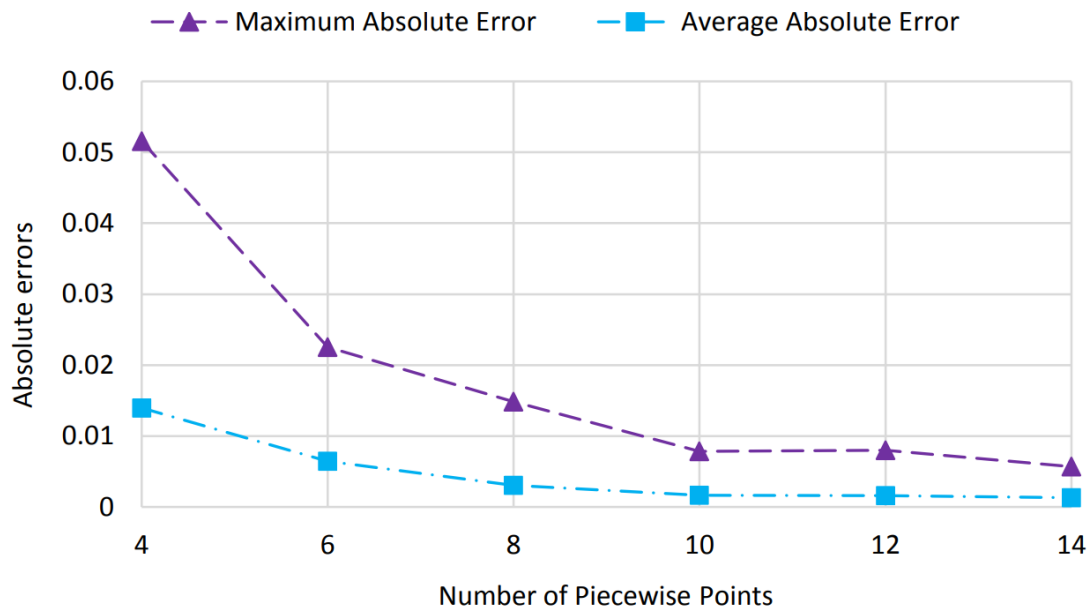


Рисунок 1.24 - Залежність максимальної та середньої абсолютної похибки PWL-апроксимації від кількості точок розбиття [86]

За своєю структурою PWL-модуль (рис. 1.25) складається з паралельних компараторів для визначення сегмента, мультиплексора для вибору відповідних коефіцієнтів a та b , одного множника та одного суматора для обчислення $y = a \cdot x + b$ [86]. Така структура забезпечує обчислення за 1-2 такти без використання DSP-блоків.

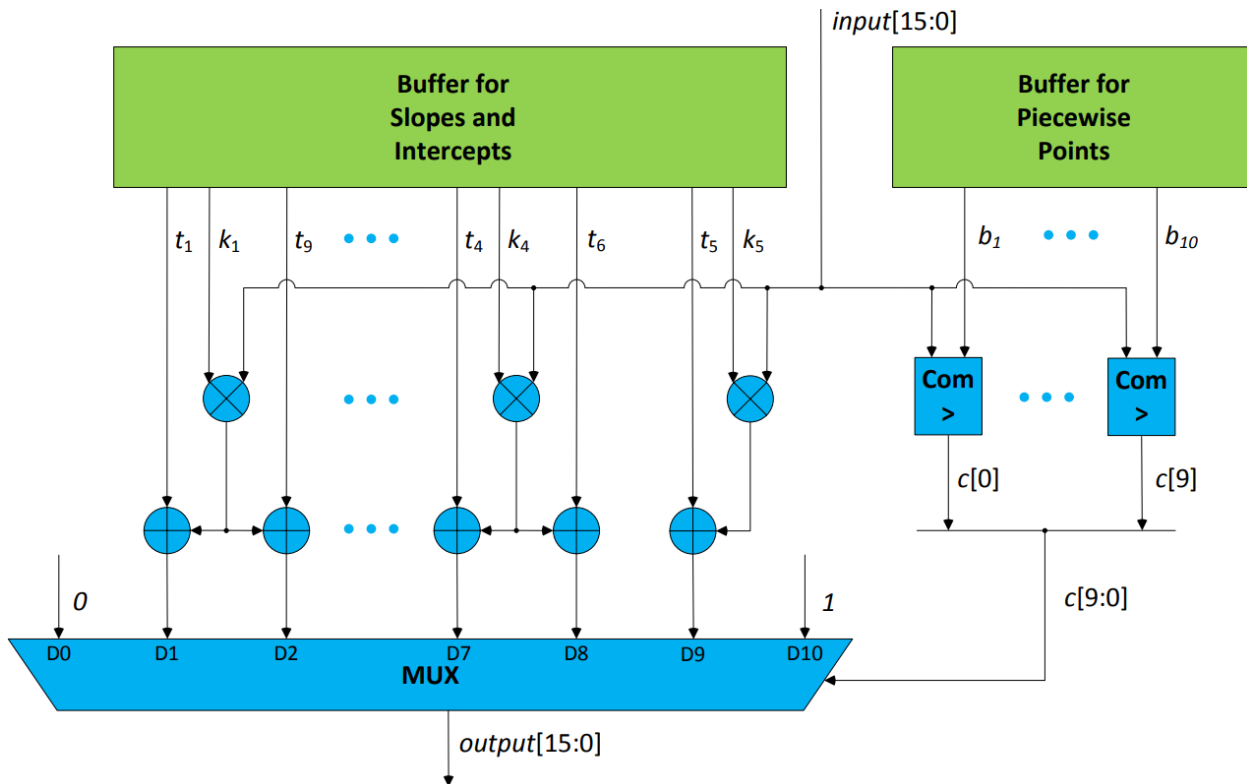


Рисунок 1.25 - Апаратна архітектура модуля кусково-лінійної апроксимації (PWL) [86].

Для гіперболічного тангенса, що є непарною функцією ($\tanh(-x) = -\tanh(x)$) та насичується при $|x| > 3$ ($\tanh(x) \rightarrow \pm 1$), апроксимація потрібна лише на обмеженому діапазоні, що додатково зменшує кількість сегментів.

Таким чином, порівняння розглянутих методів (табл. 1.5) демонструє принциповий компроміс між точністю та швидкодією. CORDIC і поліноміальна апроксимація забезпечують високу точність (10^{-3} - 10^{-11}), проте потребують 8-87 тактів та значних апаратних ресурсів. Метод LUT дає 1-тактовий доступ із помірною точністю, проте обмежений обсягом пам'яті. PWL поєднує мінімальну затримку (1-2 такти) з мінімальними ресурсами (без DSP, без RAM) за рахунок помірної точності.

Таблиця 1.5 - Порівняння методів апаратної реалізації $\tanh(x)$ на FPGA

Метод	Затримка	Ресурси	Точність	DSP
<i>CORDIC [83]</i>	13+ ітерацій (~36 нс)	2,20% LUT	0,0012	0
<i>LUT + інтерполяція [83]</i>	1 такт	0,04–0,07% LUT + ROM	0,001–0,003	0
<i>Поліноміальна [84]</i>	8–18 тактів (~60–176 нс)	478–690 LUT	10^{-5} – 10^{-11}	4–8
<i>PWL [83, 86]</i>	1–2 такти (~5– 10 нс)	0,45–1,5% LUT	0,007–0,068	0

Для задачі реалізації системи Аніщенка-Астахова на FPGA метод PWL є найбільш доцільним з таких міркувань: мінімальна затримка обчислення (1 такт); відсутність потреби у DSP-блоках та блоків пам'яті; достатня для хаотичних систем точність. Затримка в 1 такт є критичною для конвеєрного обчислення системи ОДР методом Ейлера, де кожен крок інтегрування має виконуватися за мінімальну кількість тактів. Відсутність апаратних множників дозволяє інтегрувати модуль $\tanh$ на одному кристалі разом із частотоміром, І²С-контролером та іншими компонентами IoT-вузла. Точність PWL-апроксимації порядку 10^{-2} є достатньою для генерації хаотичних послідовностей: хаотична динаміка за своєю природою стійка до малих збурень параметрів, на відміну від навчання нейронних мереж, де потрібна точність 10^{-6} і вища [84]. Детальне обґрунтування вибору кількості сегментів та кількісне порівняння точності апроксимації наведено у підрозділі 4.3.

1.6 Висновки до розділу 1

У першому розділі виконано аналітичний огляд сучасного стану досліджень у галузі побудови IoT-систем збору даних на базі FPGA та цифрової реалізації

хаотичних генераторів на програмованій логіці. На основі аналізу літературних джерел сформульовано такі висновки.

1) Обґрунтовано доцільність застосування FPGA як обчислювальної платформи сенсорного вузла IoT-системи. Показано, що FPGA займає нішу між ASIC та MCU, забезпечуючи апаратний паралелізм, детерміновані затримки та можливість реконфігурації після розгортання. Практичні приклади (FlexiS, Myrmec SmartNIC) підтвердили ефективність FPGA як вузла локальної обробки сенсорних даних із суттєвим зменшенням обсягу бездротової передачі.

2) Встановлено, що для паралельного вимірювання частоти від масиву частотних сенсорів на основі транзисторних структур з від'ємним опором мікроконтролери не забезпечують необхідної кількості синхронних каналів: ESP32 має лише 8 модулів PCNT з обмеженням ~ 40 МГц, STM32F407 до 14 таймерів без синхронної часової бази. Аналіз літератури показав, що максимальна кількість каналів в існуючих FPGA-частотомірах не перевищує 4. Водночас ресурсний аналіз підтвердив, що 12-канальний частотомір на FPGA Cyclone IV EP4CE15 потребує лише ~ 2 - 4% логічних елементів, а вбудоване процесорне ядро NIOS II дозволяє реалізувати гібридну архітектуру з апаратно-програмним розподілом функцій на одному кристалі.

3) Обґрунтовано вибір технологій для побудови каналу передачі даних IoT-вузла: інтерфейс I²C для підключення цифрових сенсорів завдяки шинній топології та мінімальній кількості провідників (реалізація I²C-контролера на FPGA потребує $\sim 2\%$ логічних елементів); технологію LoRa для бездротової передачі телеметрії завдяки надійності (втрати пакетів $< 1,6\%$ на дистанції 1 км) та низькому споживанню ($\sim 2,5$ мкА у режимі глибокого сну); топологію «зірка» для організації мережі сенсорних вузлів; трирівневу архітектуру «сенсор - шлюз - хмара» із REST API для доступу до даних.

4) Показано, що FPGA-реалізація хаотичних систем є активним напрямком досліджень із безпосереднім прикладним значенням для криптографічного захисту

IoT-каналів. Наведено приклади апаратних реалізацій: гіперхаотична система Лоренца на Intel Stratix 10 із PUF-генерацією ключів забезпечує шифрування зі швидкістю до 60 МБ/с при споживанні менше 0,4 Вт; алгоритм LZUC із хаотичним логістичним відображенням скорочує використання LUT на 30-38% при проходженні всіх 15 тестів NIST; криптосистема SCBLC на FPGA Xilinx PYNQZ2 демонструє споживання лише ~12 мВт.

5) Виконано систематичний огляд FPGA-реалізацій хаотичних осциляторів за період 2000-2026 рр. у базах Google Scholar, IEEE Xplore та Semantic Scholar. Встановлено, що FPGA-реалізації існують для систем Лоренца, Росслера, Чена, Чуа, Jerk, Лю-Чена та Нозе-Гувера із використанням переважно методу Ейлера та арифметики з фіксованою точкою 24-32 біти. Для систем Рікітакі та Аніщенко-Астахова FPGA-реалізацій у відомій літературі не виявлено.

6) Детальний аналіз стану досліджень системи Аніщенко-Астахова підтвердив, що наявні роботи обмежуються теоретичним аналізом та аналоговими схемами на дискретних компонентах (операційні підсилювачі AD711JN, аналогові множники AD633JN). Система А-А описує реальний радіотехнічний пристрій - генератор на транзисторній структурі з від'ємним диференціальним опором і відрізняється від інших хаотичних моделей наявністю трансцендентної нелінійності $G(x) \approx \tanh(\cdot)$, що потребує спеціального підходу до апаратної реалізації.

7) Проаналізовано чотири методи апаратної реалізації гіперболічного тангенса на FPGA: CORDIC (13+ ітерацій, затримка ~36 нс, точність 0,0012), таблиця LUT (1 такт, обмежена обсягом пам'яті), поліноміальна апроксимація (8-18 тактів, точність до 10^{-11} , потребує DSP-блоків) та кусково-лінійна апроксимація PWL (1-2 такти, без DSP, точність $\sim 10^{-2}$). Для задачі реалізації системи А-А метод PWL є найбільш доцільним, оскільки забезпечує мінімальну затримку при відсутності потреби у DSP-блоках та блоковій пам'яті, а точність порядку 10^{-2} є достатньою для генерації хаотичних послідовностей.

На основі проведеного аналізу визначено такі задачі дослідження: розробити багатоканальний частотомір на FPGA для паралельного вимірювання частоти від масиву сенсорів із синхронною часовою базою; удосконалити структурну організацію IoT-вузла на FPGA шляхом апаратно-програмного розподілу функцій із використанням процесорного ядра NIOS II; вперше реалізувати цифрову FPGA-реалізацію хаотичної системи Аніщенка-Астахова із кусково-лінійною апроксимацією нелінійності $\tanh$; побудувати наскрізний канал передачі телеметрії від FPGA-вузла через LoRa до хмарної платформи з можливістю моніторингу даних у режимі, близькому до реального часу.

РОЗДІЛ 2

РОЗРОБКА FPGA-ВУЗЛА ЗБОРУ ДАНИХ ВІД ЧАСТОТНИХ ТА ЦИФРОВИХ СЕНСОРІВ З АПАРАТНО-ПРОГРАМНИМ РОЗПОДІЛОМ ФУНКЦІЙ

2.1 Загальна архітектура FPGA-системи збору даних

На основі аналізу, виконаного у першому розділі, визначено вимоги до апаратної платформи вузла збору даних. Платформа має забезпечувати: одночасне паралельне вимірювання частоти від 12 незалежних сенсорів із синхронною часовою базою (підрозділ 1.2); підключення цифрових сенсорів (температура, інерціальний модуль) через інтерфейс I²C (підрозділ 1.3); передачу зібраних даних через UART до зовнішнього бездротового модуля для подальшої доставки у хмарну інфраструктуру (підрозділ 1.3); програмну гнучкість для конфігурації режимів опитування без зміни апаратної частини. У підрозділі 1.2 показано, що мікроконтролери не задовольняють першу вимогу: ESP32 має лише 8 модулів PCNT з обмеженням ~40 МГц, а STM32F407 має до 14 таймерів без синхронної часової бази для 12 каналів (табл. 1.2). Для побудови системи, що відповідає всім переліченим вимогам, розглянуто три архітектурні підходи (табл. 2.1).

Суто апаратний підхід забезпечує максимальну швидкодію та детермінованість, проте будь-яка зміна логіки опитування потребує модифікації HDL-коду та рекомпіляції проєкту. Однак суто програмний підхід на MCU, попри гнучкість, не здатний реалізувати 12 паралельних каналів вимірювання частоти із синхронною часовою базою (підрозділ 1.2). Гібридний підхід поєднує переваги обох: апаратні лічильники забезпечують паралельне детерміноване вимірювання, а вбудоване програмне ядро, тобто soft-core процесор, керує логікою збору, форматуванням даних та комунікаційними інтерфейсами. Це дозволяє змінювати режими опитування програмно без зміни апаратної частини. Саме цей підхід обрано для реалізації.

Таблиця 2.1 - Порівняння архітектурних підходів до побудови вузла збору даних

Параметр	Суто апаратний (FSM)	Суто програмний (MCU)	Гібридний (FPGA + soft-core)
<i>Паралелізм вимірювання</i>	12 каналів одночасно	Обмежений кількістю таймерів	12 каналів одночасно
<i>Детермінованість</i>	Так (апаратна)	Ні (ISR jitter)	Так (апаратна для лічильників)
<i>Гнучкість конфігурації</i>	Ні (зміна потребує рекомпіляції HDL)	Так (програмне оновлення)	Так (програмне ядро)
<i>Підтримка I²C-сенсорів</i>	Потребує окремого FSM	Вбудований контролер	FSM + програмне керування
<i>Складність модифікації</i>	Висока (перепроєктування FSM)	Низька (зміна коду)	Середня (зміна коду + HDL)
<i>Ресурси FPGA</i>	Мінімальні	Не використовує FPGA	Помірні (~30% LE на soft-core)

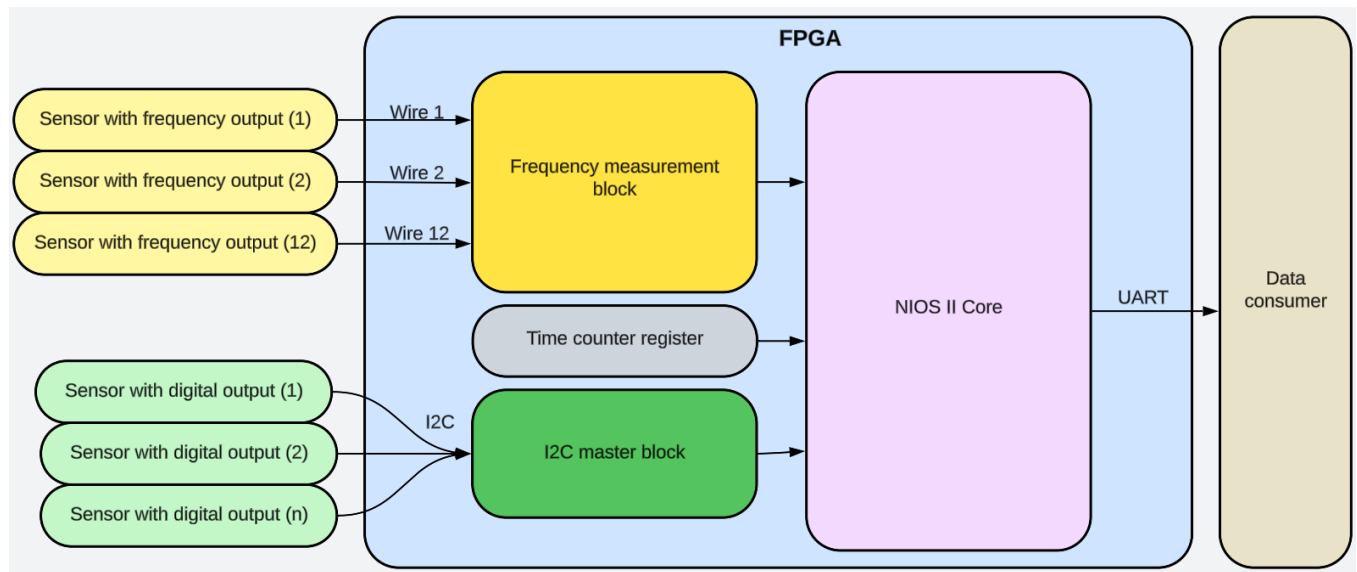


Рисунок 2.1 - Загальна структурна схема вузла збору даних на FPGA

Систему реалізовано на FPGA Intel (Altera) Cyclone IV EP4CE15F17C8, мікросхемі у корпусі BGA з 484 виводами та 15408 логічними елементами [50]. Затримка проходження сигналу від входу до внутрішнього регістра становить

близько 3 нс, що відповідає теоретичній максимальній вимірюваній частоті ~333 МГц. Для розробки використано середовище Quartus 15.1 [51]. Обґрунтування вибору саме цієї FPGA наведено у підрозділі 1.2. Структурну схему системи подано на рисунку 2.1, а фінальну реалізацію у середовищі Quartus (рис. 2.2).

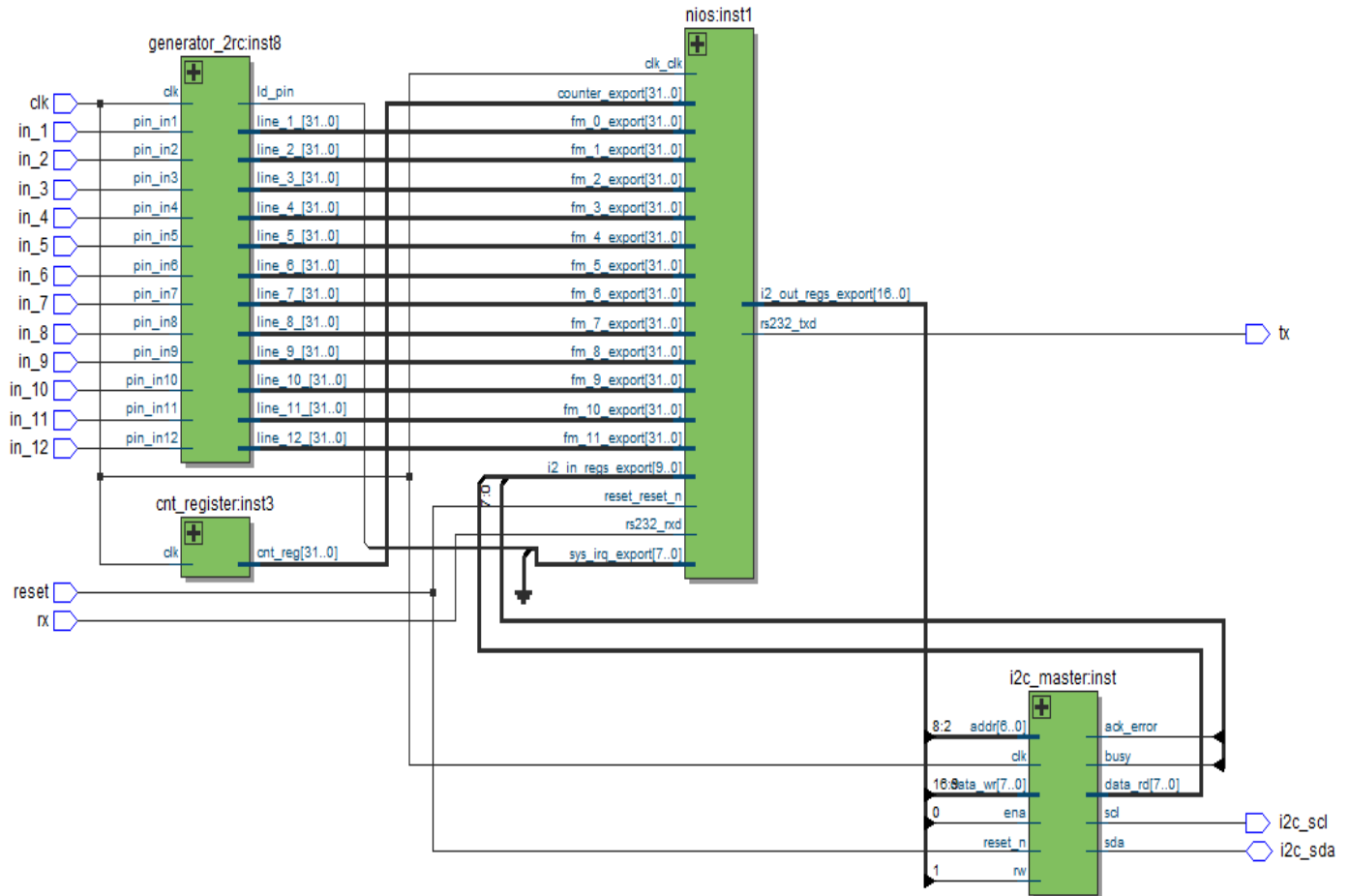


Рисунок 2.2 - Фінальна реалізація системи у Quartus

Одним із основних джерел даних у системі є сенсори з частотним виходом. Таких сенсорів передбачено 12, причому кожен підключений до FPGA на окрему вхідну лінію (*Wire 1...Wire 12* на рисунку 2.1). Кожен сенсор генерує імпульсний сигнал, частота якого відповідає вимірюваній фізичній величині. Завдяки тому, що сигнали з усіх 12 сенсорів паралельно надходять до FPGA, система може одночасно відстежувати частоту кожного з них. На відміну від MCU-рішень, де

послідовне опитування порушує синхронність вимірювань (підрозділ 1.2), тут усі канали працюють паралельно з єдиною часовою базою.

Для обробки імпульсних сигналів від сенсорів у FPGA реалізовано блок вимірювання частоти (Frequency measurement block). Цей блок здійснює підрахунок імпульсів за фіксований інтервал часу для кожного каналу. Результатом є цифрові значення частоти $fm_0_export...fm_11_export$ (рис. 2.2). Блок використовує внутрішній лічильник часу, синхронізований із системним тактуванням FPGA. Після завершення кожного циклу вимірювання блок формує сигнал переривання по лінії ld_pin , підключеній до підсистеми переривань NIOS II (рис. 2.2), що сповіщає процесорне ядро про готовність нових даних у всіх 12 вихідних регістрах.

Окремо використано Time counter register, тобто блок опорного відліку часу (шина $cnt_reg[31:0]$ на рисунку 2.2), призначений для внутрішнього обліку часу роботи ядра та планування циклів опитування. Він не залучається до алгоритму вимірювання частоти, який має власну часову базу.

Окрім частотних сенсорів, система працює з цифровими сенсорами, що передають дані по послідовній шині I²C (двопровідний інтерфейс з лініями SDA та SCL, рис. 2.1). На відміну від частотних сенсорів, цифрові видають калібровані значення безпосередньо у цифровому форматі. Управління обміном даними здійснює контролер шини I²C, реалізований як окремий апаратний блок у FPGA ($i2c_master.inst$ на рисунку 2.2). Вибір інтерфейсу I²C обґрунтовано у підрозділі 1.3: шинна топологія із двома провідниками є оптимальною для підключення 3-5 цифрових сенсорів помірної швидкодії.

Центральним елементом системи є програмоване мікропроцесорне ядро NIOS II, реалізоване всередині FPGA Cyclone IV. Процесор координує роботу всіх модулів: отримує результати вимірювання частоти від частотомірного блоку, зчитує показники цифрових сенсорів через I²C, виконує попередню обробку та форматування даних, а також керує послідовністю роботи системи: ініціює нові

цикли збору, обробляє переривання та формує масив даних для передачі. Використання програмного ядра підвищує гнучкість: логіка збору та обробки реалізується програмно і може бути модифікована без зміни апаратної схеми.

Для передачі результатів за межі FPGA передбачено інтерфейс UART зі швидкістю 115200 бод, тобто $\sim 11,5$ кБ/с при форматі 8N1. NIOS II формує пакет вимірюваних значень та надсилає його через UART до зовнішнього пристрою (вихід *tx* на рисунку 2.2). Для типового пакета з 12 частотних каналів, температури та 6 осей інерціального модуля (~ 200 байт) час передачі не перевищує 20 мс, що повністю вкладається у 10-герцовий цикл опитування. Зазначена швидкість не є фізичним обмеженням системи та може бути збільшена відповідно до вимог конкретного застосування. У ролі споживача даних виступає бездротовий модуль LilyGO LoRa32 (ESP32 + SX1276), що забезпечує подальшу передачу телеметрії через LoRa до хмарної платформи (Розділ 3).

Таким чином, загальна архітектура FPGA-вузла збору даних побудована за гібридним принципом із апаратно-програмним розподілом функцій: апаратна частина (12-канальний частотомір, I²C-контролер) забезпечує детерміноване паралельне зчитування сенсорів, а програмна частина (NIOS II) - гнучке керування, форматування даних та комунікацію через UART.

2.2 Інтеграція процесорного ядра NIOS II

Гібридний підхід до побудови вузла збору даних, обґрунтований у підрозділі 2.1, передбачає поєднання апаратних модулів (частотомір, I²C-контролер) із програмним ядром, яке координує збір, обробку та передачу даних. Для реалізації програмної частини необхідно обрати процесорне ядро, інтегроване безпосередньо у структуру FPGA.

Розглянуто три варіанти: зовнішній мікроконтролер, soft-core ядро RISC-V та soft-core ядро NIOS II. Застосування зовнішнього MCU (наприклад, STM32 або ESP32) потребує додаткового чіпа на друкованій платі, ускладнює розводку та

позбавляє програмну частину прямого доступу до внутрішньої логіки FPGA. Soft-core ядра RISC-V (VexRiscv, PicoRV32) на момент розробки мали обмежену підтримку в середовищі Quartus 15.1 та не забезпечували стандартного BSP з драйверами периферії. NIOS II, навпаки, є нативним soft-core процесором для FPGA сімейства Intel (Altera), інтегрується через утиліту Qsys із автоматичною генерацією шини Avalon та повним набором інструментів розробки [52, 53]. Саме тому для даної системи обрано NIOS II.

NIOS II є конфігурованим 32-бітним процесорним ядром з архітектурою RISC [52]. Ядро підтримує три варіанти продуктивності (Economy, Standard, Fast), кожен з яких відрізняється наявністю кешу, апаратного множника та конвеєра. Периферійні модулі підключаються через шину Avalon Memory-Mapped, а для обробки зовнішніх подій передбачено контролер переривань [53, 54]. Генерація та налаштування мікропроцесорної системи виконується в утиліті Qsys.

Першим елементом, який визначає роботу всієї системи, є Clock Source. На платі разом з FPGA знаходиться кварцовий резонатор на 50 МГц, який використано як вхідний генератор для мікропроцесорної системи. В елементі Clock Source встановлено робочу частоту 50 МГц, (рис. 2.3).

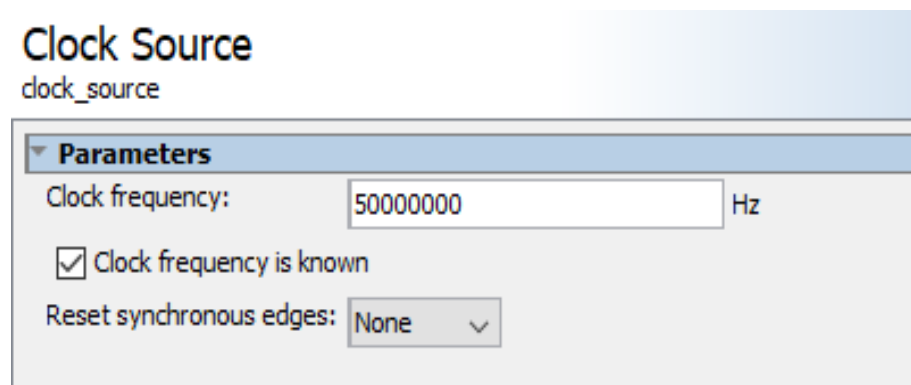


Рисунок 2.3 - Параметри налаштування елемента «Clock Source»

On-Chip Memory використовується як блок оперативної пам'яті RAM для мікропроцесорної системи. Розмір слова відповідає 32 бітам, загальний обсяг

RAM дорівнює 20 КБ (рис. 2.4). Цього обсягу достатньо для bare-metal застосунку з 10-герцовим циклом опитування, де відсутня потреба в динамічному виділенні пам'яті.

On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2

Memory type

Type: RAM (Writable) ▾

☐ Dual-port access

☐ Single clock operation

Read During Write Mode: DONT_CARE ▾

Block type: AUTO ▾

Size

Data width: 32 ▾

Total memory size: 20480 bytes

☐ Minimize memory block usage (may impact fmax)

Read latency

Slave s1 Latency: 1 ▾

Slave s2 Latency: 1 ▾

ROM/RAM Memory Protection

Reset Request: Enabled ▾

ECC Parameter

Extend the data width to support ECC bits: Disabled ▾

Рисунок 2.4 - Параметри налаштування елемента «On-Chip Memory»

UART використовується для послідовної передачі даних до зовнішнього пристрою. Основні параметри: швидкість 115 200 бод, розмір даних 8 біт, без біта парності, (рис. 2.5).

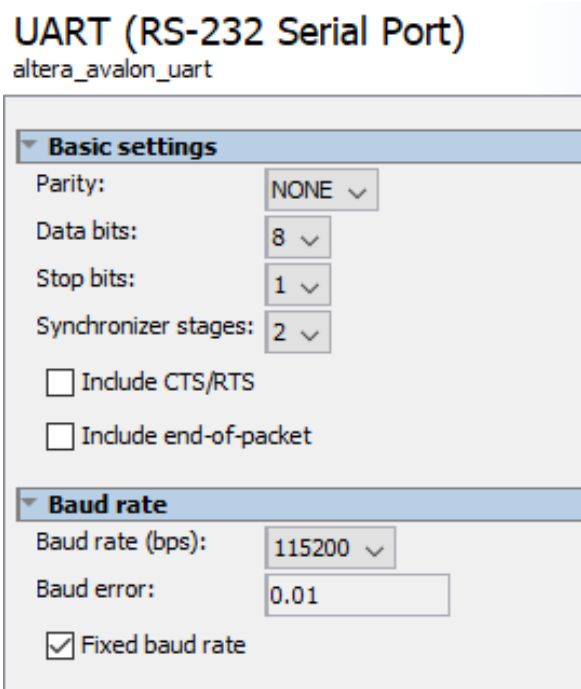


Рисунок 2.5 - Параметри налаштування інтерфейсу «UART»

I²C реалізовано через два блоки паралельного вводу-виводу (PIO) (рис. 2.6): 10-розрядна вихідна шина з підтримкою переривань (на стороні процесора) та 17-розрядна вхідна шина для зчитування стану контролера I²C (рис. 2.7). Такий підхід дозволяє програмно керувати апаратним I²C-контролером (підрозділ 2.3) через стандартні регістри PIO без створення спеціалізованого Avalon-інтерфейсу.

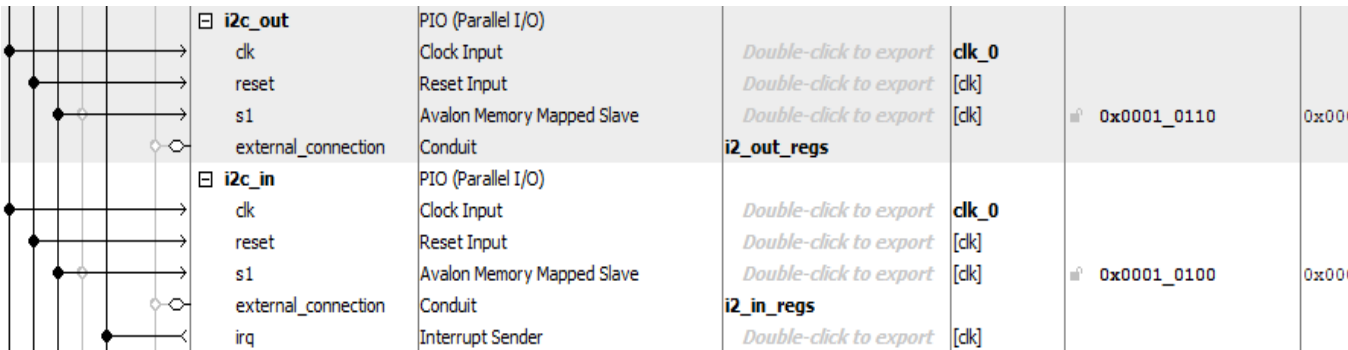


Рисунок 2.6 - Графічне відображення інтерфейсу для блоку «I2C»

System: nios Path: i2c_out	System: nios Path: i2c_in
PIO (Parallel I/O) altera_avalon_pio	
Basic Settings Width (1-32 bits): 17 Direction: ☐ Bidir ☐ Input ☐ InOut ☒ Output Output Port Reset Value: 0x0000000000000000	Basic Settings Width (1-32 bits): 10 Direction: ☐ Bidir ☒ Input ☐ InOut ☐ Output Output Port Reset Value: 0x0000000000000000
Output Register ☐ Enable individual bit setting/clearing	Output Register ☐ Enable individual bit setting/clearing
Edge capture register ☐ Synchronously capture Edge Type: RISING ☐ Enable bit-clearing for edge capture register	Edge capture register ☒ Synchronously capture Edge Type: ANY ☐ Enable bit-clearing for edge capture register
Interrupt ☐ Generate IRQ IRQ Type: LEVEL Level: Interrupt CPU when any unmasked I/O pin is logic true Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled	Interrupt ☒ Generate IRQ IRQ Type: EDGE Level: Interrupt CPU when any unmasked I/O pin is logic true Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

Рисунок 2.7 - Параметри налаштування інтерфейсу для блоку «I2C»

Блок *Counter* призначений для підрахунку часу роботи системи та синхронізації операцій, пов'язаних з вимірюваннями. Реалізація цього блоку відсутня у стандартній бібліотеці Qsys, тому його розроблено самостійно. В якості інтерфейсу використовується вхідна 32-розрядна шина (рис. 2.8). Counter побудований як звичайний регістр, значення якого інкрементується з кожним імпульсом від генератора. Частота генератора пропускається через подільник, що визначає часовий інтервал одного значення регістра (рис. 2.9). Для вимірювання 1 мс подільник встановлено у значення 1000, оскільки при частоті основного резонатора 50 МГц одне значення регістра відповідає 50 000 тактам.

System: nios Path: counter

PIO (Parallel I/O)

altera_avalon_pio

Basic Settings

Width (1-32 bits):

Direction:

☐ Bidir

☒ Input

☐ InOut

☐ Output

Output Port Reset Value:

Output Register

☐ Enable individual bit setting/clearing

Edge capture register

☐ Synchronously capture

Edge Type:

☐ Enable bit-clearing for edge capture register

Interrupt

☐ Generate IRQ

IRQ Type:

Level: Interrupt CPU when any unmasked I/O pin is logic true
Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

Рисунок 2.8 - Параметри налаштування інтерфейсу для блока «Counter»

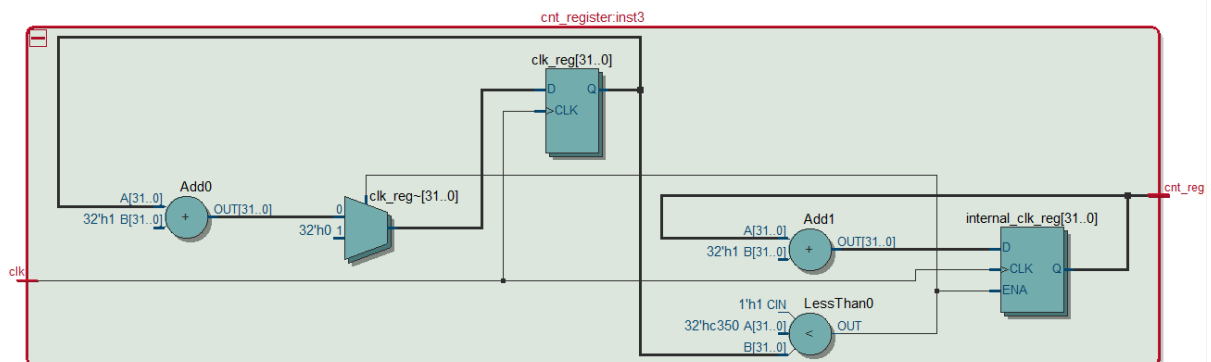


Рисунок 2.9 - Схематичне зображення лічильника часу

Sys IRQ забезпечує підключення зовнішніх джерел переривань. Підтримується до 8 джерел з детекцією фронту (рис. 2.10).

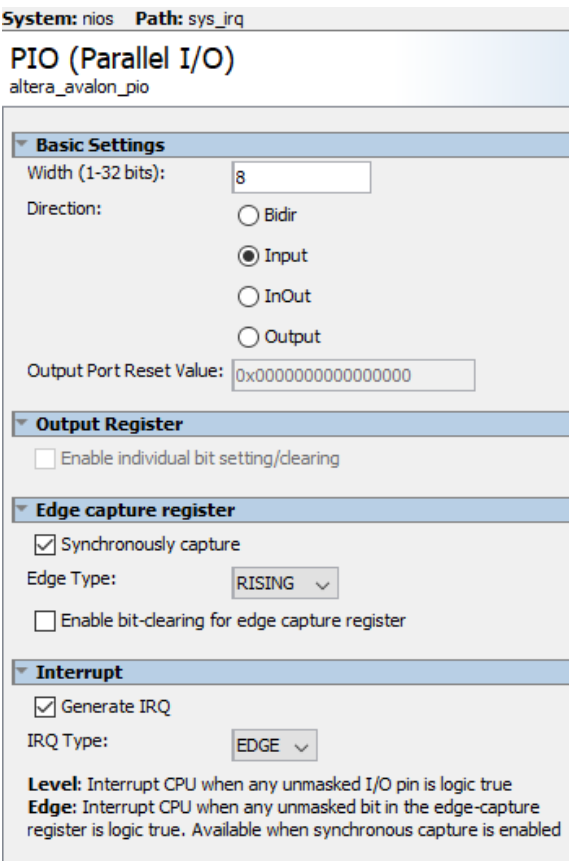


Рисунок 2.10 - Параметри налаштування інтерфейсу «Sys IRQ»

ФМ (Frequency Meter) являє собою інтерфейс для підключення блоків вимірювання частоти. Загалом підтримується 12 частотомірів *fm_0...fm_11*, кожен з яких підключено через окрему 32-розрядну вхідну шину (рис. 2.11-2.12).

fm_0	PIO (Parallel I/O)	clk_0	0x0001_00d0	0x0001_00d0
fm_1	PIO (Parallel I/O)	clk_0	0x0001_00c0	0x0001_00c0
fm_2	PIO (Parallel I/O)	clk_0	0x0001_00b0	0x0001_00b0
fm_3	PIO (Parallel I/O)	clk_0	0x0001_00a0	0x0001_00a0
fm_4	PIO (Parallel I/O)	clk_0	0x0001_0020	0x0001_0020
fm_5	PIO (Parallel I/O)	clk_0	0x0001_0050	0x0001_0050
fm_6	PIO (Parallel I/O)	clk_0	0x0001_0060	0x0001_0060
fm_7	PIO (Parallel I/O)	clk_0	0x0001_0040	0x0001_0040
fm_8	PIO (Parallel I/O)	clk_0	0x0001_0030	0x0001_0030
fm_9	PIO (Parallel I/O)	clk_0	0x0001_0070	0x0001_0070
fm_10	PIO (Parallel I/O)	clk_0	0x0001_0080	0x0001_0080
fm_11	PIO (Parallel I/O)	clk_0	0x0001_0090	0x0001_0090

Рисунок 2.11 - Графічне відображення інтерфейсу для блоків «ФМ»

System: nios Path: fm_0

PIO (Parallel I/O)

altera_avalon_pio

Basic Settings

Width (1-32 bits):

Direction:

☐ Bidir

☒ Input

☐ InOut

☐ Output

Output Port Reset Value:

Output Register

☐ Enable individual bit setting/clearing

Edge capture register

☐ Synchronously capture

Edge Type:

☐ Enable bit-clearing for edge capture register

Interrupt

☐ Generate IRQ

IRQ Type:

Level: Interrupt CPU when any unmasked I/O pin is logic true
Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

Рисунок 2.12 - Параметри налаштування інтерфейсу для блоків «FM»

Ядро Nios2 є центральним компонентом системи. Обрано варіант Classic з малою продуктивністю (Economy) (рис. 2.13), що мінімізує використання логічних елементів FPGA та залишає ресурси для апаратних модулів частотоміра й I²C-контролера. Компоненти JTAG-налагодження відключено (рис. 2.14), оскільки система призначена для автономної роботи без зовнішнього відлагоджувача. Решту параметрів згенеровано автоматично.

Nios II (Classic) Processor

altera_nios2_qsys

Nios II Core:

☒ Nios II/e

☐ Nios II/s

☐ Nios II/f

	Nios II/e	Nios II/s	Nios II/f
Nios II Selector Guide	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction
Memory Usage (e.g Stratix IV)	Two M9Ks (or equiv.)	Two M9Ks + cache	Three M9Ks + cache

Hardware Arithmetic Operation

Hardware multiplication type: Embedded Multipliers

☐ Hardware divide

Reset Vector

Reset vector memory: onchip_memory2_0.s1

Reset vector offset: 0x00000000

Reset vector: 0x00008000

Рисунок 2.13 - Вибір продуктивності ядра мікропроцесорної системи

Nios II (Classic) Processor

altera_nios2_qsys

Debug level:

☒ No Debugger

☐ Level 1

☐ Level 2

☐ Level 3

☐ Level 4

No Debugger	Level 1	Level 2	Level 3	Level 4
	JTAG Target Connection Download Software Software Breakpoints	JTAG Target Connection Download Software Software Breakpoints 2 Hardware Breakpoints 2 Data Triggers	JTAG Target Connection Download Software Software Breakpoints 2 Hardware Breakpoints 2 Data Triggers Instruction Trace On-Chip Trace	JTAG Target Connection Download Software Software Breakpoints 4 Hardware Breakpoints 4 Data Triggers Instruction Trace On-Chip Trace Data Trace Off-chip Trace
No M9Ks	One M9K	One M9K	One M9K + trace	One M9K + trace

☐ Include debugreq and debugack Signals

These signals appear on the top-level Qsys system.
You must manually connect these signals to logic external to the Qsys system.

Break Vector

Break vector memory: onchip_memory2_0.s1

Break vector offset: 0x00000020

Рисунок 2.14 - Налаштування налагоджувального компонента

На (рис. 2.15) зображено блок, що містить усі компоненти згенерованої мікропроцесорної системи.

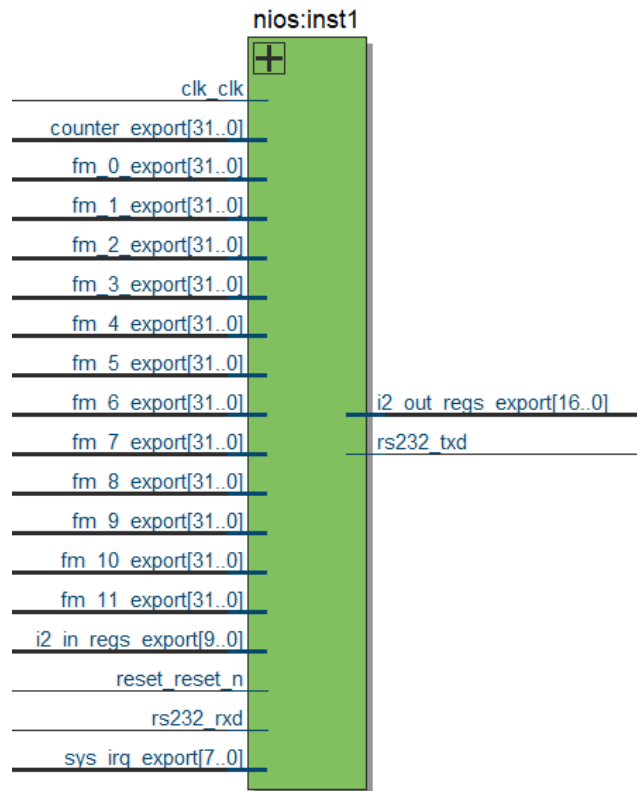


Рисунок 2.15 - Блок згенерованої мікропроцесорної системи

Інтеграція ядра NIOS II у структуру SoC на базі FPGA Cyclone IV сформувала компакту керуючу платформу вузла збору даних. Обмін між процесорним ядром і периферійними модулями (UART, I²C, 12-канальний блок частотомірів, системний лічильник часу, контролер переривань) реалізовано через внутрішню шину Avalon із низькими затримками. Обрана конфігурація з малоресурсним варіантом ядра, 20 КБ вбудованої RAM та тактовою частотою 50 МГц мінімізує використання логічних елементів, залишаючи резерв для апаратних модулів.

2.3 Модуль зчитування цифрових сенсорів по інтерфейсу I2C

Вибір інтерфейсу I²C для підключення цифрових сенсорів обґрунтовано у підрозділі 1.3: шинна топологія із двома провідниками (SDA, SCL) є оптимальною для підключення 3-5 пристроїв помірної швидкодії на спільну шину. Контролер I²C реалізовано як окремий VHDL-модуль (*i2c_master*, рис. 2.16) замість використання готового IP-блоку Altera. Таке рішення дає повний контроль над таймінгом обміну, мінімізує використання логічних елементів та усуває залежності від стороннього IP. Модуль працює в режимі ведучого (Master), генерує сигнали SCL та SDA, забезпечує формування умов старту і стопу, передачу адреси та даних, обробку підтверджень ACK/NACK від підлеглого пристрою, а також підтримує механізм розтягування такту (clock stretching) згідно зі специфікацією I²C [87].

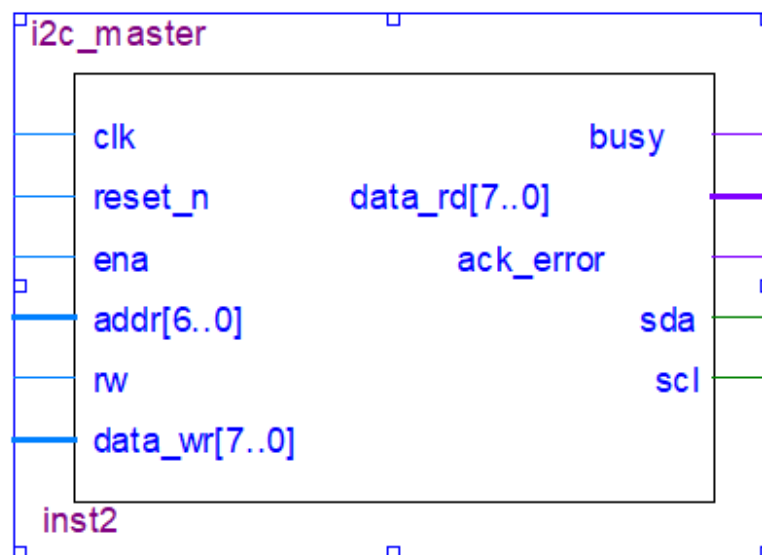


Рисунок 2.16 - Блок реалізації I²C приймача/передавача

Модуль описано мовою VHDL як синхронний цифровий автомат з двома процесами: перший відповідає за генерацію тактового сигналу SCL (поділ частоти), другий реалізує автомат станів для послідовної передачі та прийому

бітів. Інтерфейс модуля складається з генериків для налаштування частоти та портів вводу-виводу (рис. 2.17).

```

ENTITY i2c_master IS
  GENERIC(
    input_clk : INTEGER := 50_000_000; -- input clock speed from user logic in Hz
    bus_clk   : INTEGER := 400_000); -- speed the i2c bus (scl) will run at in Hz
  PORT(
    clk      : IN STD_LOGIC;           -- system clock
    reset_n  : IN STD_LOGIC;           -- active low reset
    ena      : IN STD_LOGIC;           -- latch in command
    addr     : IN STD_LOGIC_VECTOR(6 DOWNTO 0); -- address of target slave
    rw       : IN STD_LOGIC;           -- '0' is write, '1' is read
    data_wr  : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- data to write to slave
    busy     : OUT STD_LOGIC;           -- indicates transaction in progress
    data_rd  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- data read from slave
    ack_error : BUFFER STD_LOGIC;       -- flag if improper acknowledge from slave
    sda      : INOUT STD_LOGIC;         -- serial data line (I2C bus)
    scl      : INOUT STD_LOGIC;         -- serial clock line (I2C bus)
  );
END i2c_master;

```

Рисунок 2.17 – Фрагмент програмного коду

Призначення портів наведено у табл. 2.2.

Таблиця 2.2 - Порти модуля i2c_master

Порт	Напрямок	Розрядність	Призначення
<i>clk</i>	IN	1	Системний тактовий сигнал (50 МГц)
<i>reset_n</i>	IN	1	Асинхронне скидання (активний 0)
<i>ena</i>	IN	1	Запуск операції: подача '1' ініціює транзакцію
<i>addr</i>	IN	7	7-бітна адреса підлеглого пристрою
<i>rw</i>	IN	1	Режим: '0' запис, '1' читання
<i>data_wr</i>	IN	8	Байт даних для запису
<i>busy</i>	OUT	1	'1' під час виконання транзакції
<i>data_rd</i>	OUT	8	Байт даних, прочитаний від підлеглого
<i>ack_error</i>	BUFFER	1	'1' якщо підлеглий не підтвердив АСК
<i>sda, scl</i>	INOUT	1+1	Двонаправлені лінії шини I ² C (відкритий сток)

Генерик *input_clk* задає частоту системного тактового сигналу (за замовчуванням 50 МГц), *bus_clk* визначає бажану частоту шини SCL (400 кГц).

Для формування SCL модуль використовує лічильник поділу частоти. Константа *divider* обчислюється за формулою:

$$divider = \frac{input_clk}{4 \times bus_clk} \quad (2.1)$$

де множник 4 відповідає поділу одного періоду SCL на чотири рівні часові інтервали. Для значень за замовчуванням $\frac{50 \text{ МГц}}{4 \times 400 \text{ кГц}} = 31.25 \text{ МГц}$, тобто ціле значення 31, що дає фактичну частоту SCL близько 403 кГц. Генерація SCL відбувається у першому процесі архітектури, де лічильник *count* змінюється від 0 до $4 \times divider - 1$ (рис. 2.18).

```

IF(count = divider*4-1) THEN      -- досягнуто кінця циклу
    count := 0;                  -- скинути лічильник
ELSIF(stretch = '0') THEN        -- немає розтягування такту підлеглим
    count := count + 1;          -- збільшити лічильник
END IF;
CASE count IS
    WHEN 0 TO divider-1 =>      -- 1-ша чверть періоду (SCL низький)
        scl_clk <= '0';
        data_clk <= '0';
    WHEN divider TO divider*2-1 => -- 2-га чверть періоду (SCL низький)
        scl_clk <= '0';
        data_clk <= '1';
    WHEN divider*2 TO divider*3-1 => -- 3-тя чверть періоду (SCL високий)
        scl_clk <= '1';          -- відпустити SCL (високий рівень)
        IF(scl = '0') THEN      -- перевірка розтягування SCL підлеглим
            stretch <= '1';    -- поставити прапор паузи
        ELSE
            stretch <= '0';
        END IF;
        data_clk <= '1';
    WHEN OTHERS =>              -- 4-та чверть періоду (SCL високий)
        scl_clk <= '1';
        data_clk <= '0';
END CASE;

```

Рисунок 2.18 – Фрагмент програмного коду

Лічильник *count* інкрементується на кожному такті системного *clk* і скидається після досягнення значення $4 \times divider - 1$. Один повний цикл SCL поділяється на чотири фази. У першій та другій чвертях (*count* від 0 до $2 \times$

divider – 1) внутрішній сигнал *scl_clk* = '0', що утримує лінію SCL у низькому стані. У третій та четвертій чвертях *scl_clk* = '1', тобто SCL відпущена у високий рівень через зовнішні підтягувальні резистори. Сигнал *data_clk* формує імпульси для синхронізації подачі та захоплення даних: він дорівнює '1' у другій та третій чвертях, забезпечуючи зміну SDA лише при *SCL* = 0, як того вимагає специфікація I²C [87]. Якщо під час третьої чверті (коли ведучий очікує *SCL* = 1) фактична лінія *scl* залишається у '0', це означає, що підлеглий пристрій застосовує розтягування такту (clock stretching). У такому випадку прапор *stretch* встановлюється в '1', що зупиняє інкрементування лічильника *count* до моменту відпускання лінії підлеглим.

Основна логіка обміну реалізована як автомат з дев'ятьма станами: *ready* (очікування команди), *start* (формування стартової умови), *command* (передача адреси та біта R/W), *slv_ack1* (очікування ACK після адреси), *wr* (передача байта даних), *rd* (зчитування байта від підлеглого), *slv_ack2* (очікування ACK після запису), *mstr_ack* (генерація ACK/NACK від ведучого після читання), *stop* (формування стопової умови). При скиданні (*reset_n* = '0') автомат повертається у стан *ready*, вихідні сигнали встановлюються у безпечні значення: *busy* = '1', *scl_ena* = '0' (SCL відпущена), *sda_int* = '1' (SDA відпущена), *ack_error* = '0'. Після скидання *busy* переводиться в '0', сигналізуючи готовність до нової транзакції.

Автомат реагує на фронти *data_clk*: на наростаючому фронті (коли SCL низький) відбувається зміна стану та виставлення наступного біту на SDA, а на спадному фронті (коли SCL високий) виконується захоплення вхідних бітів та перевірка ACK від підлеглого. Внутрішній лічильник *bit_cnt* відраховує біти від 7 до 0 при передачі або прийомі кожного байту. Коли всі 8 біт оброблено (*bit_cnt* = 0), автомат переходить до наступного стану.

Умови Start та Stop формуються за рахунок керування лінією SDA відносно SCL. Для Start SDA переходить з високого рівня в низький при *SCL* = 1, для Stop

SDA переходить з низького в високий при $SCL = 1$. Керування лініями SDA та SCL реалізовано за принципом відкритого стоку (рис. 2.19).

```
WITH state SELECT
  sda_ena_n <= data_clk_prev WHEN start, -- drive SDA low for START
    NOT data_clk_prev WHEN stop, -- drive SDA high for STOP
  sda_int      WHEN OTHERS;
-- Open-drain outputs for SCL and SDA lines
scl <= '0' WHEN (scl_ena = '1' AND scl_clk = '0') ELSE 'Z';
sda <= '0' WHEN sda_ena_n = '0' ELSE 'Z';
```

Рисунок 2.19 – Фрагмент програмного коду

Вихід *scl* активується на низький рівень, коли дозвіл *scl_ena* = '1' і внутрішній такт *scl_clk* = '0'. В інших випадках на *scl* подається 'Z' (високий імпеданс), а зовнішні підтягувальні резистори формують високий рівень. Аналогічно для *sda*: при *sda_ena_n* = '0' лінія притягується до нуля, при *sda_ena_n* = '1' переходить у трирівневий стан.

У режимі запису послідовність роботи така: зовнішня логіка подає *ena* = '1' разом з адресою (*addr*), бітом режиму (*rw* = '0') та байтом даних (*data_wr*). Автомат формує Start, після чого у стані *command* передає 8 біт адресного байту (7 біт адреси + 1 біт R/W, об'єднаних у внутрішній регістр *addr_rw*). По завершенні передачі модуль відпускає SDA та переходить у стан *slv_ack1*, де на 9-му такті перевіряє рівень SDA: якщо підлеглий притягнув лінію до '0', це ACK, тобто пристрій з такою адресою присутній на шині. Якщо *SDA* = '1' (NACK), встановлюється прапор *ack_error* = '1'. Далі автомат переходить у стан *wr* і аналогічно передає 8 біт даних, після чого очікує ACK у стані *slv_ack2*. Завершується транзакція формуванням *Stop*.

У режимі читання (*rw* = '1') після адресного ACK автомат переходить у стан *rd*, де відпускає SDA, щоб підлеглий міг керувати лінією. На кожному спадному фронті *data_clk* (коли SCL високий) модуль захоплює поточний біт з SDA та записує його у відповідний розряд внутрішнього регістру *data_rx(bit_cnt)*. Після

прийому 8 біт ведучий вирішує, чи продовжувати читання. Якщо зовнішня логіка утримує $ena = '1'$ і нова команда адресує той самий пристрій у режимі читання, модуль генерує ACK (притягує SDA до '0'), сигналізуючи підлеглому про готовність прийняти ще один байт. Якщо продовження не потрібне, генерується NACK (SDA залишається у '1'). Прийнятий байт переміщується з $data_rx$ до вихідного порту $data_rd$.

Модуль підтримує серії операцій без звільнення шини через механізм Repeated Start. Якщо після завершення поточної операції сигнал $ena = '1'$, автомат не переходить у стан *stop*, а продовжує нову операцію. Логіка стану slv_ack2 ілюструє цей механізм (рис. 2.20).

```

WHEN slv_ack2 =>          -- slave acknowledge bit (after write)
  IF(ena = '1') THEN      -- continue transaction
    busy <= '0';          -- signal ready for next command
    addr_rw <= addr & rw;  -- load next address and command
    data_tx <= data_wr;    -- load next data byte to write
    IF(addr_rw = addr & rw) THEN -- same slave & operation
      sda_int <= data_wr(bit_cnt);
      state <= wr;         -- continue with another write byte
    ELSE                  -- new read or different slave
      state <= start;      -- repeated start for new transaction
    END IF;
  ELSE                    -- no further commands
    state <= stop;        -- end transaction with STOP
  END IF;

```

Рисунок 2.20 – Фрагмент програмного коду

Якщо нова команда адресує той самий пристрій у тому самому режимі (умова $addr_rw = addr \& rw$), автомат переходить у стан *wr* для передачі наступного байту без повторного старту. Якщо адреса або режим змінюються, генерується Repeated Start, тобто автомат переходить у стан *start* без попереднього *Stop*. Це дозволяє виконувати складені транзакції, типові для роботи з цифровими сенсорами. Зокрема, зчитування регістра MPU6050 вимагає саме такої послідовності: спочатку запис адреси регістра (наприклад, $ACCEL_XOUT_H = 0x3B$), потім Repeated Start з тим самим пристроєм у режимі читання для

отримання значення. Без підтримки Repeated Start довелося б виконувати два окремих звертання до шини із ризиком втрати арбітражу між ними. Аналогічна логіка реалізована у стані *mstr_ack* для режиму читання: якщо нова команда продовжує зчитування з того самого пристрою, автомат повертається в стан *rd* без повторного старту, забезпечуючи безперервний прийом послідовних байтів.

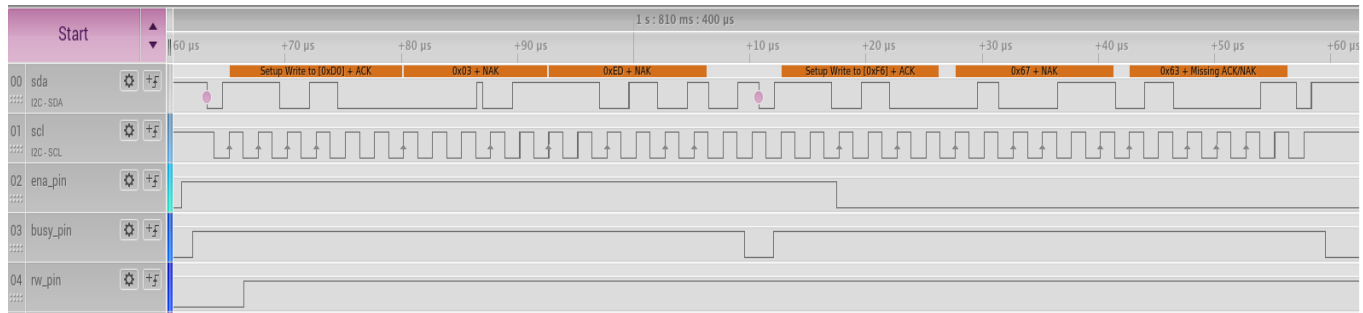


Рисунок 2.21 - Відображення сигналів на лініях інтерфейсу I²C при передачі даних

Часова діаграма (рис. 2.21) підтверджує коректність формування сигналів: Start, адресний байт з ACK, байт даних з ACK та Stop. Сигнал *busy* активний протягом усієї транзакції і переходить у '0' лише після формування *Stop* або на короткий момент між байтами при безперервній серії операцій. Прапор *ack_error* встановлюється у '1', якщо підлеглий не підтвердив прийом (*SDA* $\neq$ 0 на 9-му такті), і зберігається до початку наступної транзакції. Виставлення *ack_error* не зупиняє автомат: транзакція завершується за стандартним сценарієм, а зовнішня логіка може перевірити стан прапора після переходу *busy* у '0' і вжити відповідних заходів (наприклад, повторити спробу або зафіксувати помилку).

Через описаний контролер до FPGA-вузла підключено інерціальний модуль MPU6050, що забезпечує вимірювання прискорення за трьома осями, кутової швидкості за трьома осями та температури. Конфігурація MPU6050 та програмне зчитування його показників описані у підрозділі 2.5.

2.4 Модуль зчитування частотних сенсорів

Принцип роботи частотних сенсорів та метод прямого рахунку детально розглянуто у підрозділі 1.2. Частота обчислюється за формулою

$$f = \frac{N}{T}, \quad (2.2)$$

де N - кількість зафіксованих фронтів за інтервал T . Квантова роздільна здатність за частотою визначається як $\Delta f = 1/T$, а відносна похибка ідеального лічильника становить $\varepsilon = 1/N$. Вибір T є компромісом між роздільністю та швидкістю оновлення результату.

Для розроблюваної системи обрано вікно вимірювання $T = 100$ мс. Роздільність за частотою при цьому дорівнює $\Delta f = 1/0,1 = 10$ Гц. Для типових частотних перетворювачів на основі транзисторних структур з від'ємним опором (підрозділ 1.2), робочі частоти яких лежать у діапазоні від сотень кілогерц до одиниць мегагерц, таке значення Δf забезпечує відносну похибку порядку 10^{-4} – 10^{-5} , що є достатнім для задач моніторингу фізичних величин. Зменшення T до 10 мс погіршило б роздільність у 10 разів ($\Delta f = 100$ Гц), а збільшення до 1 с уповільнило б оновлення до неприйняттого для інтерактивного моніторингу рівня. Повний цикл вимірювання з урахуванням службових інтервалів (скидання, завантаження) становить ~ 106 мс, що забезпечує частоту оновлення $\sim 9,4$ Гц.

З погляду інтеграції з FPGA враховано три аспекти. Вхідний сигнал сенсора доводиться до логічних рівнів формувачем на тригері Шмітта з ESD-захистом. Оскільки імпульси сенсора асинхронні відносно системного такту clk (50 МГц), перед лічбою застосовується двоступенева синхронізація для мінімізації метастабільності з подальшим детектуванням фронту. Практичне обмеження полягає у тому, що ширина імпульсу сенсора має бути не меншою за один період clk , а частота має бути нижчою за $clk/2$.

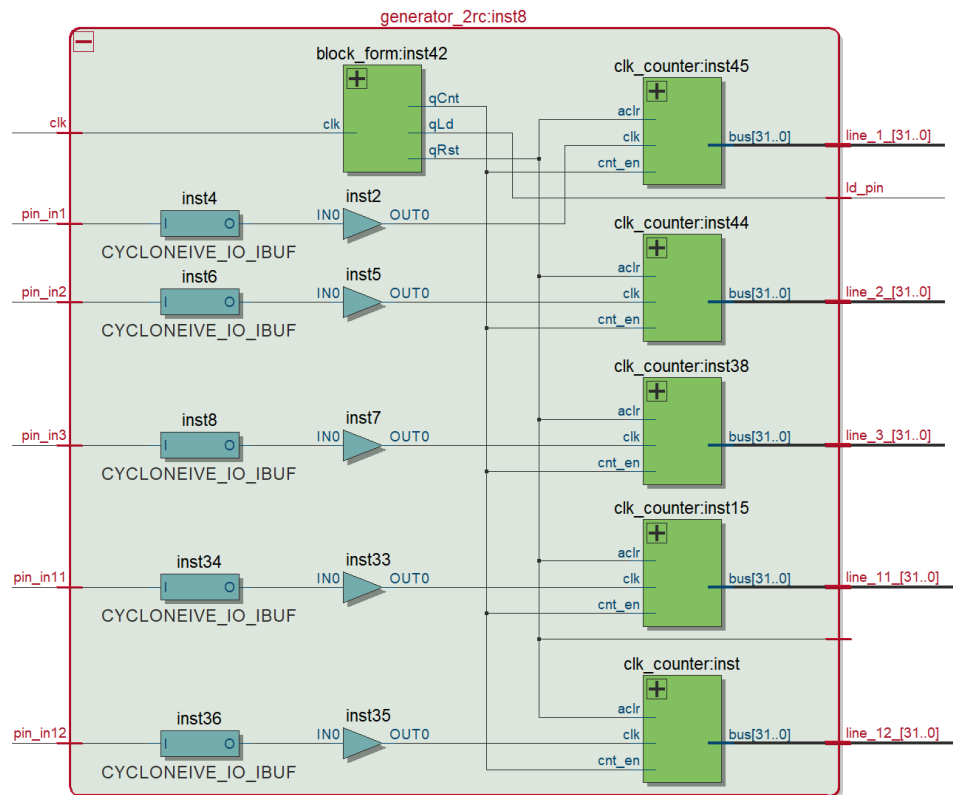


Рисунок 2.22 - Структурна схема блоку частотоміра

Сигнал із сенсора подається на виділений вхід FPGA і проходить внутрішній буфер вводу кристала (рис. 2.22, примітиви *CYCLONEIVE_IO_IBUF*). За потреби сигнал інвертується для уніфікації полярності: надалі подія визначається по фронту «0-1» на всіх каналах. Таке формування виконується на кожному каналі окремо.

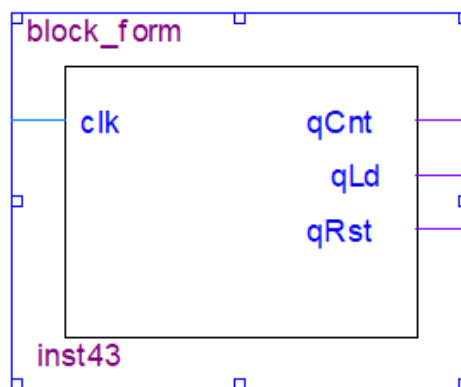


Рисунок 2.23 - Відображення блоку block_form у середовищі Quartus

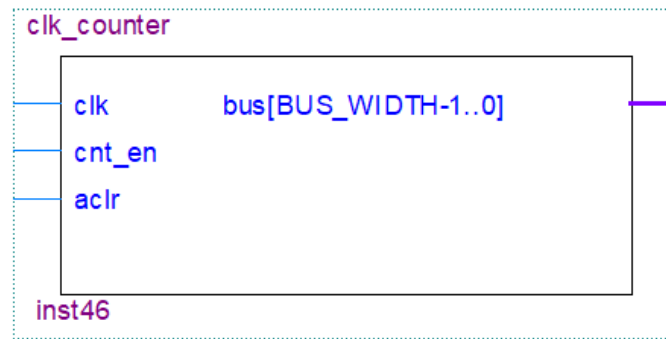


Рисунок 2.24 - Відображення блоку `clk_counter` у середовищі Quartus

Вікном підрахунку керує модуль *block_form* (рис. 2.23), що подає на всі канали спільні керувальні лінії: *qRst* (скидання перед початком вимірювання), *qCnt* (дозвіл рахунку протягом інтервалу 100 мс) та *qLd* (захоплення результату в регістр по завершенні інтервалу). На відміну від MCU, де кількість паралельних каналів обмежена апаратними таймерами (підрозділ 1.2, табл. 1.2), в FPGA-реалізації ці сигнали розгалужуються від єдиного *block_form* до кожного лічильника *clk_counter* (рис. 2.24), тож усі 12 входів сенсорів зчитуються одночасно у єдиній часовій системі відліку. Максимальна кількість каналів в існуючих FPGA-частотомірах, знайдена у літературі, не перевищує 4 [45]. Результат кожного каналу виводиться на 32-розрядну шину *line_#[31..0]*, звідки він доступний процесору NIOS II.

Модуль *block_form* синтезує три синхронні керувальні сигнали за допомогою одного 24-бітного таймера counter, що працює від системного такту *clk* (50 МГц, рис. 2.25).

```
module block_form (
  input wire clk,
  output reg qCnt,
  output reg qLd,
  output reg qRst
);
reg [23:0] counter;
localparam F = 50; // МГц
```

Рисунок 2.25 – Фрагмент програмного коду

Інтервали задаються у мікросекундах: $t1 = 500$ (ширина імпульсу скидання), $t2 = 200$ (пауза перед стартом підрахунку), $t3 = 99\,800$ (вікно підрахунку $\approx 99,8$ мс), $t4 = 200$ (пауза після зупинки підрахунку), $t5 = 5000$ (ширина імпульсу завантаження ≈ 5 мс), $t6 = 200$ (пауза перед новим циклом). Ці інтервали конвертуються у «тіки» тактової частоти за формулою $ni = \sum t_j \cdot F$ (при 50 МГц один мікросекунді відповідає 50 тактів) (рис. 2.26).

```
// (мкс)
localparam t1 = 500, t2 = 200, t3 = 99800,
            t4 = 200, t5 = 5000, t6 = 200;

// мітки часу у «тіках» такту
localparam n1 = 0,
            n2 = n1 + t1 * F,
            n3 = n2 + t2 * F,
            n4 = n3 + t3 * F,
            n5 = n4 + t4 * F,
            n6 = n5 + t5 * F,
            n7 = n6 + t6 * F;
```

Рисунок 2.26 – Фрагмент програмного коду

Для наведених параметрів конкретні мітки становлять: $n2 = 25\,000$, $n3 = 35\,000$, $n4 = 5\,025\,000$, $n5 = 5\,035\,000$, $n6 = 5\,285\,000$, $n7 = 5\,295\,000$. Ширини імпульсів: $qRst = 500$ мкс, $qCnt = 99,8$ мс, $qLd = 5$ мс; повний цикл $\approx 105,9$ мс. У позиціях $n1$, $n3$, $n5$ відповідний сигнал виставляється в '1', а у $n2$, $n4$, $n6$ повертається в '0'; між цими позначками присвоєнь немає, тому регістри утримують попередній стан, формуючи імпульси потрібної тривалості без глітчів (рис. 2.27).

Сигнал $qRst$ під'єднується до входу скидання лічильників, $qCnt$ до входу дозволу рахунку (cnt_en), а qLd до стробу захоплення результату (лінія ld_pin та внутрішні регістри виходу). Всі три сигнали синтезуються в одному тактовому домені clk , тому фронти узгоджені та детерміновані. Короткі паузи $t2$ та $t4$ навмисно розділяють події «скидання - початок рахунку» та «кінець рахунку - завантаження», щоб уникнути накладання та забезпечити стабільний стан даних у

момент захоплення. Вибір розрядності таймера обґрунтовується максимальною міткою: $W \geq \lceil \log_2(n_7 + 1) \rceil$. Для $n_7 = 5\,295\,000$ мінімальна розрядність $W = 23$, тому обрано 24 біти із запасом.

```
always @(posedge clk) begin
  if (counter == n7) counter <= 0;
  else counter <= counter + 1;

  case (counter)
    n1: begin qRst <= 1; qCnt <= 0; qLd <= 0; end
    n2: begin qRst <= 0; qCnt <= 0; qLd <= 0; end

    n3: begin qRst <= 0; qCnt <= 1; qLd <= 0; end
    n4: begin qRst <= 0; qCnt <= 0; qLd <= 0; end

    n5: begin qRst <= 0; qCnt <= 0; qLd <= 1; end
    n6: begin qRst <= 0; qCnt <= 0; qLd <= 0; end

    n7: begin qRst <= 0; qCnt <= 0; qLd <= 0; end
    default: /* утримання попередніх значень */;
  endcase
end
endmodule
```

Рисунок 2.27 – Фрагмент програмного коду

Модуль *cnt_register* є універсальним параметризованим лічильником, який рахує такти вхідного *clk* і при досягненні межі *CLK_CNT_MAX* інкрементує вихідний регістр (рис. 2.28).

```
 #(parameter CLK_CNT_MAX=1, parameter REG_WIDTH=32)
 (
   input clk,
   output [REG_WIDTH-1:0]cnt_reg
 );
 `define CLEAR_REG 0
 reg [31:0]clk_reg = `CLEAR_REG;
 reg [REG_WIDTH-1:0]internal_clk_reg = `CLEAR_REG;
 assign cnt_reg[REG_WIDTH-1:0] = internal_clk_reg[REG_WIDTH-1:0];
 always @(posedge clk) begin
   clk_reg <= clk_reg + 1;

   if (clk_reg >= CLK_CNT_MAX) begin
     // Clear register
     internal_clk_reg <= internal_clk_reg + 1;
     clk_reg <= `CLEAR_REG;
   end
 end
 endmodule
```

Рисунок 2.28 – Фрагмент програмного коду

Поведінково цей модуль є подільником частоти: *clk_reg* відраховує такти системного домену, а *internal_clk_reg* збільшується щоразу, коли набрано *CLK_CNT_MAX* тактів. Еквівалентний інтервал між інкрементами виходу:

$$T_{tick} \approx \frac{\max(1, CLK_CNT_MAX)}{f_{clk}} \quad (2.3)$$

$$f_{tick} \approx \frac{f_{clk}}{\max(1, CLK_CNT_MAX)} \quad (2.4)$$

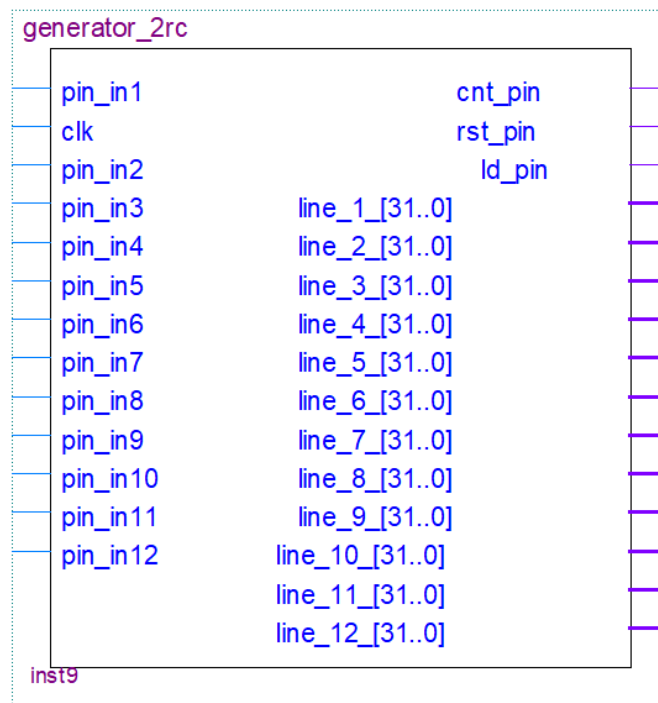


Рисунок 2.29 - Відображення блоку *generator_2rc* у середовищі Quartus

Після кожного циклу вимірювання отримане число імпульсів зберігається у 32-розрядній шині (наприклад, *line_1[31..0]* для першого каналу, рис. 2.29) і стає доступним для зчитування процесором NIOS II через шину Avalon. Похибка вимірювання визначається квантованістю підрахунку і становить ± 1 імпульс на інтервал T .

Реалізований модуль частотоміра має ряд переваг порівняно з альтернативними підходами до вимірювання. Оскільки інформативний параметр

передається у вигляді частоти, відпадає потреба в аналого-цифровому перетворенні: цифрова логіка безпосередньо рахує імпульси, що підвищує завадостійкість та спрощує апаратну частину. Усі 12 каналів працюють паралельно від спільної часової бази *block_form*, тому результати вимірювань є синхронними та порівнянними між собою. Додавання нових каналів потребує лише інстанціювання додаткових лічильників *clk_counter* без зміни логіки керування, що забезпечує масштабованість системи. При цьому апаратні витрати на один канал мінімальні: 32-розрядний лічильник, синхронізатор та детектор фронту займають порядку 30–40 логічних елементів (менше 0,4% ресурсів Cyclone IV EP4CE15).

2.5 Апаратні та програмні аспекти підключення сенсорів

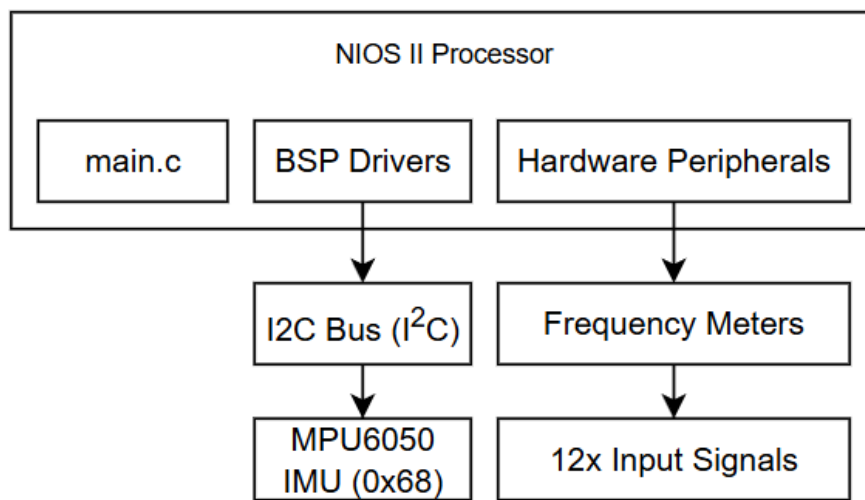


Рисунок 2.30 - Структура програмної реалізації поверх апаратної платформи

Програмне забезпечення системи реалізовано поверх апаратної платформи на базі NIOS II (рис. 2.30). Функціональна мета програми полягає в отриманні узгоджених у часі вимірювань з 12-канального частотоміра та інерціального модуля MPU6050 [88] через I²C із фіксованою періодичністю 100 мс (10 Гц) та передачі їх у вигляді табличного потоку через UART.

Програмне забезпечення реалізовано без операційної системи реального часу, тобто у режимі bare-metal. Для системи з одним потоком виконання та фіксованим 10-герцовим циклом опитування застосування RTOS (зокрема FreeRTOS [89] або Zephyr [90]) є надлишковим: додатковий рівень абстракції збільшує обсяг коду, споживання RAM та вносить нондетерміновані затримки через планувальник задач. Bare-metal підхід мінімізує розмір прошивки (менше 20 КБ RAM), забезпечує повний контроль над таймінгом та спрощує верифікацію системи.

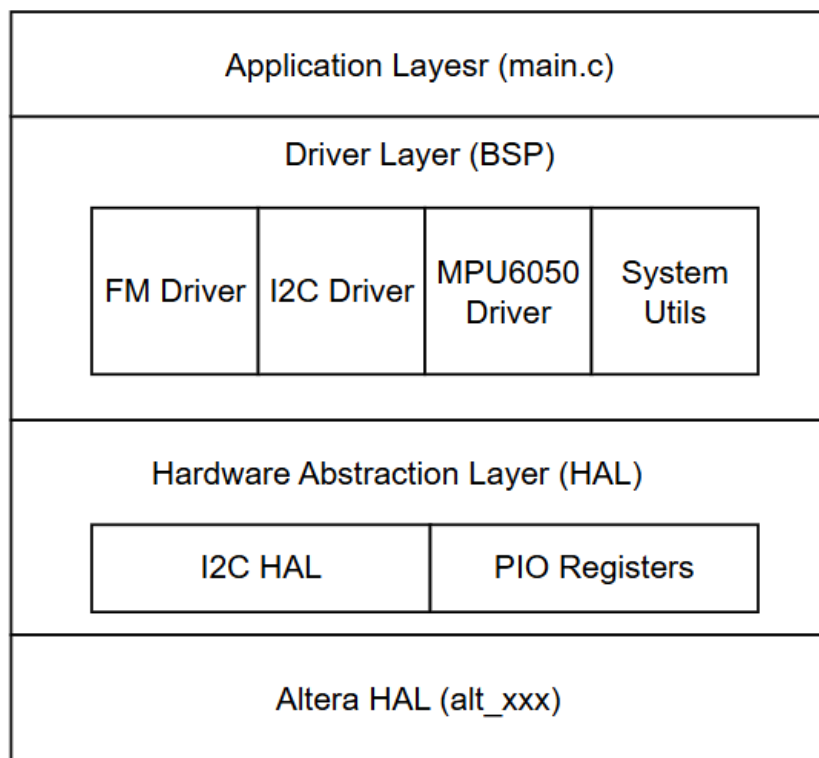


Рисунок 2.31 - Архітектура програмного забезпечення системи

Архітектура програмного забезпечення має три рівні (рис. 2.31): прикладна логіка (main.c), шар драйверів BSP (Board Support Package) та шар апаратної абстракції HAL (Hardware Abstraction Layer). Такий поділ ізолює апаратно-залежний код у нижніх рівнях, а основна програма оперує з ними через узгоджений інтерфейс функцій BSP.

Модуль *frequency_meter* реалізує підсистему вимірювання частоти по 12 незалежних каналах. Драйвер працює в режимі переривань: апаратний лічильник частоти генерує переривання по завершенні вікна вимірювання, обробник якого (ISR) послідовно зчитує значення всіх каналів та зберігає їх у глобальному масиві. Прикладна програма отримує дані через неблокуючу функцію *fm_get_freq(channel)*, яка повертає останнє оновлене значення без звертання до апаратури.

Модуль *i2c* відповідає за обмін даними по шині I²C та забезпечує зв'язок процесора з цифровими сенсорами. Архітектурно драйвер розділений на дві частини: файл *i2c.c* містить логіку високого рівня (послідовність операцій, інтерфейс для прикладного коду), а файл *i2c_hal.c* зосереджує всю взаємодію з апаратними регістрами РІО-контролера. Для гарантії коректності I²C-транзакцій драйвер реалізує механізми атомарності через критичні секції та поетапну перевірку підтверджень АСК.

Драйвер *mpu6050* забезпечує ініціалізацію інерціального модуля (перевірка *WHO_AM_I*, налаштування діапазонів вимірювання, вихід з режиму сну) та подальше зчитування показників акселерометра, гіроскопа і температури. Обмін даними з MPU6050 виконується виключно через виклики I²C-драйвера.

Модуль *system* об'єднує допоміжні компоненти: *sys_irq.c* (керування перериваннями та критичні секції), *time.c* (системний таймер на базі апаратного лічильника Counter) та *bsp_printf.c* (полегшена реалізація форматowanego виводу через UART).

У фазі ініціалізації програма послідовно активує всі підсистеми (рис. 2.32)

```
sys_printf("Frequency meter initializing\r\n");
if (fm_init()) sys_printf("Error during frequency meter initializing\r\n");
sys_printf("I2C initializing\r\n");
if (i2c_init()) sys_printf("Error during I2C initializing\r\n");
sys_printf("MPU6050 initializing\r\n");
if (mpu6050_init(&mpu_dev)) sys_printf("Error during MPU6050 initializing\r\n");
```

Рисунок 2.32 – Фрагмент програмного коду

Спочатку *fm_init()* реєструє обробник переривань частотоміра та активує відповідні апаратні переривання. Далі *i2c_init()* конфігурує контролер I²C. Після цього *mpu6050_init()* виконує перевірку наявності сенсора (*WHO_AM_I*), встановлює джерело тактового сигналу, діапазони вимірювання та виводить пристрій із режиму сну.

Основний цикл виконується нескінченно з періодом ~100 мс (рис. 2.33).

```

/* Event loop never exits. */
while (1) {
    alt_u16 x = 0x00, y = 0x00, z = 0x00;
    alt_u16 temp = 0x00;
    alt_u8 i = 0x00;

    for (; i < CHANNEL_MAX; i++) data_printf(i == 0? "%d" : "\t%d", fm_get_freq(i));
    mpu6050_temp(&mpu_dev, &temp);
    data_printf("\t%d", temp);

    mpu6050_accel(&mpu_dev, &x, &y, &z);
    data_printf("\t%d\t%d\t%d", x, y, z);

    x = y = z = 0x00;
    mpu6050_gyro(&mpu_dev, &x, &y, &z);
    data_printf("\t%d\t%d\t%d\r\n", x, y, z);

    data_printf("\r\n");
    sys_delay_ms(100);
}

```

Рисунок 2.33 – Фрагмент програмного коду

На кожній ітерації послідовно зчитуються значення частоти з 12 каналів через *fm_get_freq()*, температура, прискорення та кутова швидкість від MPU6050. Зчитування частот є неблокуючим, оскільки драйвер частотоміра оновлює значення асинхронно через ISR.

виконується затримка `sys_delay_ms(100)` для забезпечення стабільного інтервалу між вибірками.

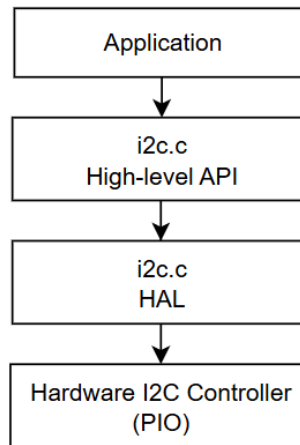


Рисунок 2.36 - Архітектура I²C драйвера

Драйвер I²C побудовано за дворівневою архітектурою (рис. 2.36). Верхній рівень (*i2c.c*) містить логіку роботи шини та інтерфейс для прикладного коду: функції *i2c_write_bytes*, *i2c_read_bytes* для роботи з масивами, а також *i2c_write_bits*, *i2c_read_bits* для операцій з окремими бітами регістрів. Нижній рівень (*i2c_hal.c*) реалізує HAL для безпосереднього доступу до регістрів PIO контролера I²C.

Ініціалізація HAL встановлює маску переривань, очищує регістр захоплення подій та реєструє обробник переривань (рис. 2.37).

```

alt_8 i2c_init(void) {
    i2c_hal_init(_i2c_irq_handler, NULL);
    return RES_OK;
}

alt_8 i2c_hal_init(i2c_hal_irq_func_t user_func, void *user_context) {
    _i2c_data.func = user_func;
    _i2c_data.context = user_context;
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(I2C_IN_BASE, IRQ_MASK); // enable interrupt
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(I2C_IN_BASE, IRQ_CLEAR); // clear interrupt edge capture register
    alt_ic_isr_register(I2C_IN_IRQ_INTERRUPT_CONTROLLER_ID, I2C_IN_IRQ, i2c_hal_irq_handler, (void *)&_i2c_data,
0x00);
    return RES_OK;
}
  
```

Рисунок 2.37 – Фрагмент програмного коду

Функція запису масиву байтів *i2c_write_bytes* ілюструє типову I²C-транзакцію. Алгоритм складається з таких кроків: перевірка зайнятості шини, входження у критичну секцію (блокування всіх переривань для атомарності), встановлення адреси підлеглого та режиму запису через HAL, передача адреси цільового регістра, послідовна передача байтів даних у циклі з перевіркою АСК після кожного, формування *Stop* після останнього байта та відновлення переривань. Якщо на будь-якому етапі підлеглий не підтверджує прийом (*ACK_ERR*), функція негайно завершує транзакцію, звільняє шину та повертає код помилки *RES_ERR* (рис. 2.38).

```
alt_8 i2c_write_bytes(alt_u8 dev_addr, alt_u8 reg_addr, alt_u8 *w_data_arr, alt_u8 len) {
    alt_u8 i = 0x00;
    alt_u32 irq_state = 0x00;
    if (i2c_is_busy() || !w_data_arr || !len) return RES_ERR;
    // Disable all interrupts
    irq_state = sys_irq_critical_section_begin();
    i2c_hal_set_slave_addr(dev_addr);
    i2c_hal_set_rw(I2C_WRITE);
    i2c_hal_set_write_data(reg_addr);
    i2c_hal_set_ena(I2C_ENABLE_TRANSMISSION);
    // Wait stage of set register value
    _i2c_wait_for_busy(TRUE);
    for (i = 0x00; i < len; i++) {
        i2c_hal_set_write_data(w_data_arr[i]);
        if (i2c_is_err()) {
            i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
            sys_irq_critical_section_end(irq_state);
            sys_printf("\r\n%s() -> Err at set register value stage, dev: %d, reg: %d\r\n", __func__, dev_addr, reg_addr + i);
            return RES_ERR;
        }
        // Wait for write register value stage
        _i2c_wait_for_busy(FALSE);
        _i2c_wait_for_busy(TRUE);
    }
    i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
    // End of transmission
    _i2c_wait_for_busy(FALSE);
    if (i2c_is_err()) {
        i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
        sys_irq_critical_section_end(irq_state);
        sys_printf("\r\n%s() -> Err at end of read process, dev: %d, reg: %d\r\n", __func__, dev_addr, reg_addr + i);
        return RES_ERR;
    }
    // Enable interrupts
    sys_irq_critical_section_end(irq_state);
    return RES_OK;
}
```

Рисунок 2.38 – Фрагмент програмного коду

Функція зчитування масиву байтів *i2c_read_bytes* використовує Repeated Start: спочатку в режимі запису передається адреса регістра, потім без звільнення шини виконується перемикання в режим читання та послідовний прийом заданої кількості байтів (рис. 2.39).

```
alt_8 i2c_read_bytes(alt_u8 dev_addr, alt_u8 reg_addr, alt_u8 *r_data_arr, alt_u8 len) {
    alt_u8 i = 0x00;
    alt_u32 irq_state = 0x00;
    if (i2c_is_busy() || !r_data_arr || !len) return RES_ERR;
    // Disable all interrupts
    irq_state = sys_irq_critical_section_begin();
    // Set up I2C registers for read
    i2c_hal_set_slave_addr(dev_addr);
    i2c_hal_set_rw(I2C_WRITE);
    i2c_hal_set_write_data(reg_addr);
    i2c_hal_set_ena(I2C_ENABLE_TRANSMISSION);
    // Wait for start transmission
    _i2c_wait_for_busy(TRUE);
    i2c_hal_set_rw(I2C_READ);
    if (i2c_is_err()) {
        i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
        sys_irq_critical_section_end(irq_state);
        sys_printf("\r\n%s() -> Err at start transmission, dev: %d, reg: %d\r\n", __func__, dev_addr, reg_addr);
        return RES_ERR;
    }
    // Wait read stage
    _i2c_wait_for_busy(FALSE);
    for (i = 0x00; i < len; i++) {
        _i2c_wait_for_busy(TRUE);
        // Disable transmission before end of current transaction for avoid starting new transaction
        if (i == len - 1) i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
        _i2c_wait_for_busy(FALSE);
        i2c_hal_get_read_data(r_data_arr + i);
        if (i2c_is_err()) {
            i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
            sys_irq_critical_section_end(irq_state);
            sys_printf("\r\n%s() -> Err during read register, dev: %d, reg: %d\r\n", __func__, dev_addr, reg_addr + i);
            return RES_ERR;
        }
    }
    // Enable interrupts
    sys_irq_critical_section_end(irq_state);
    return RES_OK;
}
```

Рисунок 2.39 – Фрагмент програмного коду

Після передачі адреси регістра в режимі запису драйвер перемикає RW на *I2C_READ*, що спричиняє Repeated Start на шині. У циклі зчитування кожен прийнятий байт забирається з регістра *R_DATA* через *i2c_hal_get_read_data()*.

Перед зчитуванням останнього байту драйвер скидає *ENA*, що змушує контролер сформувати NACK та Stop після прийому цього байту. Синхронізація з апаратним контролером здійснюється методом опитування (busy-wait) через функцію *_i2c_wait_for_busy*. Такий підхід спрощує синхронізацію порівняно з використанням переривань, а в контексті системи із 10-герцовим циклом опитування не створює помітного навантаження на процесор.

HAL реалізує доступ до двох 32-бітних PIO-регістрів FPGA. Регістр *I2C_OUT* призначений для керування передавачем і має таку структуру полів: *ENA* (біт 0, маска 0x01) активує передачу, *RW* (біт 1, маска 0x01) визначає напрямок (0 для запису, 1 для читання), *SLAVE_ADDR* (біти 2–8, маска 0x7F) містить 7-бітну адресу підлеглого, *W_DATA* (біти 9-16, маска 0xFF) зберігає байт даних для передачі. Регістр *I2C_IN* призначений для зчитування стану: *BUSY* (біт 0) вказує на зайнятість шини, *ACK_ERR* (біт 1) сигналізує про відсутність підтвердження, *R_DATA* (біти 2-9) містить прийнятий байт. Запис і читання окремих полів виконуються через узагальнену функцію read-modify-write (рис. 2.40).

```
alt_8 i2c_hal_reg_write(alt_u8 reg_mask, alt_u8 reg_offset, alt_u8 value) {
    // Read last reg value and clear bits for set new value
    alt_u32 reg_value = IORD_ALTERA_AVALON_PIO_DATA(I2C_OUT_BASE) & ~(alt_u32)reg_mask << reg_offset;
    // Update value for reg
    reg_value |= ((alt_u32)value & reg_mask) << reg_offset;
    // Write new value into register
    IOWR_ALTERA_AVALON_PIO_DATA(I2C_OUT_BASE, reg_value);
    return RES_OK;
}
```

Рисунок 2.40 – Фрагмент програмного коду

На базі цієї функції побудовано набір inline-обгортки для керування окремими полями регістра. Наприклад, встановлення адреси підлеглого та зчитування отриманого байту (рис. 2.41).

```
inline alt_8 i2c_hal_set_slave_addr(alt_u8 addr){
```

```

    return i2c_hal_reg_write(SLAVE_ADDR_MASK, SLAVE_ADDR_OFFSET, addr);
}

inline alt_8 i2c_hal_get_read_data(alt_u8 *r_data){
    return i2c_hal_reg_read(R_DATA_MASK, R_DATA_OFFSET, r_data);
}

```

Рисунок 2.41 – Фрагмент програмного коду

Решта обгортки (*set_ena*, *set_rw*, *set_write_data*, *get_busy*, *get_ack_error*) побудовані ідентично з відповідними масками та зсувами. Драйвер частотоміра (*frequency_meter*) працює в режимі переривань. Функція *fm_init()* реєструє обробник та активує апаратне переривання по лінії *ld_pin* (рис. 2.42).

```

alt_8 fm_init() {
    sys_irq_enable(IRQ_MASK);
    // Register ISR
    alt_ic_isr_register(SYS_IRQ_IRQ_INTERRUPT_CONTROLLER_ID, SYS_IRQ_IRQ, fm_irq_handler, NULL, 0x00);
    return RES_OK;
}

```

Рисунок 2.42 – Фрагмент програмного коду

Обробник переривання зчитує всі 12 каналів та зберігає значення у глобальному масиві (рис. 2.43).

```

void fm_irq_handler(void* context) {
    sys_irq_clear_interrupt_register(IRQ_MASK);
    alt_u8 i = 0x00;
    for (i = 0x00; i < CHANNEL_MAX; i++) {
        channels_data[i] = IORD_ALTERA_AVALON_PIO_DATA(channel_id_to_addr(i));
    }
}

```

Рисунок 2.43 – Фрагмент програмного коду

ISR спочатку очищує прапорець переривання через *sys_irq_clear_interrupt_register*, після чого послідовно зчитує значення з усіх 12 регістрів PIO (*FM_0_BASE...FM_11_BASE*) за допомогою допоміжної функції *channel_id_to_addr*, яка відображає номер каналу 0–11 на відповідну базову адресу.

Код ISR максимально компактний: зчитування 12 регістрів займає кілька десятків тактів процесора, що мінімізує час перебування у контексті переривання. Використання єдиного переривання для всіх каналів замість 12 окремих ISR суттєво знижує накладні витрати на переключення контексту.

Функція *fm_get_freq(channel)* повертає значення з масиву *channels_data* за відповідним індексом. Ніяких звернень до апаратури у цей момент не відбувається, оскільки вимірювання вже виконано ISR. Виклик є неблокуючим і завжди повертає останнє оновлене значення частоти. Відображення номера каналу (0–11) на відповідну базову адресу PIO (*FM_0_BASE*–*FM_11_BASE*) реалізовано через допоміжну функцію *channel_id_to_addr*, побудовану як switch-case з 12 гілками.

Системний модуль (system) об'єднує утиліти загального призначення. Підмодуль *sys_irq.c* забезпечує централізоване керування перериваннями: функції *sys_irq_enable* та *sys_irq_disable* встановлюють або знімають біти у регістрі масок переривань PIO через макроси IORD/IOWR. Для атомарного виконання критичних ділянок коду реалізовано пару функцій *sys_irq_critical_section_begin* (зберігає поточну маску та блокує всі переривання) та *sys_irq_critical_section_end* (відновлює збережену маску). Цей механізм використовується в усіх I²C-транзакціях для запобігання гонитвам.

Підмодуль *time.c* реалізує таймер на базі апаратного лічильника Counter (підрозділ 2.2). Функція *sys_time_ms()* зчитує поточне значення регістра *COUNTER_BASE*, яке інтерпретується як кількість мілісекунд з моменту запуску системи. На основі цього лічильника побудовано функцію блокуючої затримки (рис. 2.44).

```
/**
 * Blocking wait for required time
 */
void sys_delay_ms(alt_u32 time_ms) {
```

```

volatile alt_u32 stop_time = sys_time_ms() + time_ms;
while (sys_time_ms() < stop_time);
}

/**
 * Return system time in ms
 */
alt_u32 sys_time_ms() {
    return IORD_ALTERA_AVALON_PIO_DATA(COUNTER_BASE);
}

```

Рисунок 2.44 – Фрагмент програмного коду

Використання ключового слова `volatile` для змінної `stop_time` запобігає оптимізації компілятором циклу очікування. У `bsp_printf.c` реалізовано полегшену версію `printf` з підтримкою лише базових форматних специфікаторів (`%d`, `%x`, `%s`, `%c`), що суттєво зменшує обсяг коду та потребу у пам'яті порівняно зі стандартною бібліотекою. Для виводу телеметрії використовується окрема функція `data_printf`, яка направляє дані через UART.

Драйвер MPU6050 забезпечує роботу з 6-вісним інерціальним модулем, що містить тривісний акселерометр, тривісний гіроскоп та датчик температури [87]. Стан кожного підключеного пристрою зберігається у структурі (рис. 2.45).

```

typedef struct mpu6050_t mpu6050_t;
struct mpu6050_t {
    mpu6050_t *next;
    mpu6050_t *prev;
    alt_u8 addr;
    alt_u8 clk;
    alt_u8 gyro_fs;
    alt_u8 accel_fs;
    float acc_coeff;
    float gyro_coeff;
};

```

Рисунок 2.45 – Фрагмент програмного коду

Поля `gyro_fs` та `accel_fs` визначають обрані діапазони вимірювання ($\pm 250/500/1000/2000^\circ/\text{с}$ для гіроскопа та $\pm 2/4/8/16g$ для акселерометра). На етапі ініціалізації за цими значеннями розраховуються масштабні коефіцієнти `gyro_coeff` та `accel_coeff`, які перетворюють сирі 16-бітні відліки сенсора у фізичні

одиниці. Наприклад, при діапазоні $\pm 2g$ коефіцієнт акселерометра становить $2.0/32768.0 = 6.1 \times 10^{-5} g$ на одиницю відліку [88]. Ініціалізація MPU6050 виконує перевірку наявності пристрою (регістр *WHO_AM_I* має повернути 0x68), конфігурацію внутрішніх регістрів та розрахунок коефіцієнтів (рис. 2.46).

Зчитування показників акселерометра виконується за один виклик І²С-читання 6 байтів (по 2 байти на кожну вісь) з подальшим масштабуванням (рис. 2.47).

```
alt_8 mpu6050_init(mpu6050_t *mpu_dev) {
    alt_u8 data = 0x00;
    if (_mpu6050_add_new_dev(mpu_dev)) {
        sys_printf("\r\n%s() -> Error adding new device\r\n", __func__);
        return RES_ERR;
    }
    if (i2c_read_byte(mpu_dev->addr, MPU6050_RA_WHO_AM_I, &data) || mpu_dev->addr != data) {
        sys_printf("\r\n%s() -> Error device with address %d is absent on the bus\r\n", __func__, mpu_dev->addr);
        return RES_ERR;
    }
    if (i2c_write_bits(mpu_dev->addr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_CLKSEL_MASK <<
MPU6050_PWR1_CLKSEL_OFFSET, mpu_dev->clk << MPU6050_PWR1_CLKSEL_OFFSET)) {
        sys_printf("\r\n%s() -> Error during setup clk, dev: %d, reg: %d\r\n", __func__, mpu_dev->addr,
MPU6050_RA_PWR_MGMT_1);
        return RES_ERR;
    }
    if (i2c_write_bits(mpu_dev->addr, MPU6050_RA_GYRO_CONFIG, MMPU6050_RA_GYRO_MASK <<
MPU6050_RA_GYRO_OFFSET, mpu_dev->gyro_fs << MPU6050_RA_GYRO_OFFSET)) {
        sys_printf("\r\n%s() -> Error during setup gyro, dev: %d, reg: %d\r\n", __func__, mpu_dev->addr,
MPU6050_RA_GYRO_CONFIG);
        return RES_ERR;
    }
    if (i2c_write_bits(mpu_dev->addr, MPU6050_RA_ACCEL_CONFIG, MPU6050_RA_ACCEL_CONFIG_MASK <<
MPU6050_RA_ACCEL_CONFIG_OFFSET, mpu_dev->accel_fs << MPU6050_RA_ACCEL_CONFIG_OFFSET)) {
        sys_printf("\r\n%s() -> Error during setup accel, dev: %d, reg: %d\r\n", __func__, mpu_dev->addr,
MPU6050_RA_ACCEL_CONFIG);
        return RES_ERR;
    }
    if (i2c_write_bits(mpu_dev->addr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_SLEEP_MASK <<
MPU6050_PWR1_SLEEP_OFFSET, 0x00 << MPU6050_PWR1_SLEEP_OFFSET)) {
        sys_printf("\r\n%s() -> Error get up device from sleep, dev: %d, reg: %d\r\n", __func__, mpu_dev->addr,
MPU6050_RA_PWR_MGMT_1);
        return RES_ERR;
    }
    return RES_OK;
}
```

Рисунок 2.46 – Фрагмент програмного коду

```
alt_8 mpu6050_accel(mpu6050_t *mpu_dev, alt_16 *x, alt_16 *y, alt_16 *z) {
    alt_u8 accel_data[MPU6050_ACCEL_REG_NUM] = {0x00};
    if (!mpu_dev || !_mpu6050_get_dev_by(mpu_dev->addr) || !x || !y || !z) {
        sys_printf("\r\n%s() -> Error incorrect parameters\r\n", __func__);
    }
```

```

        return RES_ERR;
    }
    if (i2c_read_bytes(mpu_dev->addr, MPU6050_RA_ACCEL_XOUT_H, accel_data, MPU6050_ACCEL_REG_NUM)) {
        sys_printf("\r\n%s() -> Error read from dev: %d, reg: %d\r\n", __func__, mpu_dev->addr, MPU6050_RA_ACCEL_XOUT_H);
        return RES_ERR;
    }
    // Converting received data into accel x, y, z
    *x = (((alt_16)accel_data[0]) << 8) | accel_data[1];
    *y = (((alt_16)accel_data[2]) << 8) | accel_data[3];
    *z = (((alt_16)accel_data[4]) << 8) | accel_data[5];
    *x = *x < 0? (alt_16)(mpu_dev->acc_coeff * (*x * -1)) * -1 : (alt_16)(mpu_dev->acc_coeff * *x);
    *y = *y < 0? (alt_16)(mpu_dev->acc_coeff * (*y * -1)) * -1 : (alt_16)(mpu_dev->acc_coeff * *y);
    *z = *z < 0? (alt_16)(mpu_dev->acc_coeff * (*z * -1)) * -1 : (alt_16)(mpu_dev->acc_coeff * *z);
    return RES_OK;
}

```

Рисунок 2.47 – Фрагмент програмного коду

Функція *mpu6050_gyro()* побудована аналогічно із використанням регістру *GYRO_XOUT_H* та коефіцієнта *gyro_coeff*. Зчитування температури реалізовано з механізмом повторних спроб (до 15 ітерацій) для обробки ситуацій, коли сенсор повертає нульове значення (рис. 2.48).

```

alt_8 mpu6050_temp(mpu6050_t *mpu_dev, alt_u16 *temp) {
    alt_u8 temp_data[MPU6050_TEMP_REG_NUM] = {0x00}, guard_cnt = MPU6050_TRYS_TO_READ;
    if (!mpu_dev || !mpu6050_get_dev_by(mpu_dev->addr) || !temp) {
        sys_printf("\r\n%s() -> Error incorrect parameters\r\n", __func__);
        return RES_ERR;
    }
    read_temp:
    if (i2c_read_bytes(mpu_dev->addr, MPU6050_RA_TEMP_OUT_H, temp_data, MPU6050_TEMP_REG_NUM)) {
        sys_printf("\r\n%s() -> Error read from dev: %d, reg: %d\r\n", __func__, mpu_dev->addr, MPU6050_RA_TEMP_OUT_H);
        return RES_ERR;
    }
    if (temp_data[0] == 0 && temp_data[1] == 0) {
        // Read allowed only guard_cnt numbers
        if (guard_cnt-- > 0) {
            goto read_temp;
        } else {
            //sys_printf("\r\n%s() -> Error max count of read temperature\r\n", __func__);
            return RES_ERR;
        }
    }
    // Converting received data into correct temperature value
    *temp = (((alt_u16)temp_data[0]) << 8) | temp_data[1];
    *temp = ((float)*temp / 340) + 36.53;
    return RES_OK;
}

```

Рисунок 2.48 – Фрагмент програмного коду

Формула перетворення сирого значення у градуси Цельсія ($temp/340 + 37$) відповідає специфікації MPU6050 [88]. Механізм *retry* обробляє типову ситуацію,

коли датчик температури ще не виконав перше вимірювання після виходу з режиму сну

Завдяки розділеній архітектурі BSP код, що працює з конкретними регістрами, ізольовано у рівнях HAL та драйверів, а файл main.c оперує лише функціями API. У разі заміни сенсора (наприклад, MPU6050 на ICM-20948) достатньо модифікувати відповідний модуль BSP без зміни основного алгоритму програми.

2.6 Висновки до розділу 2

У другому розділі виконано розробку апаратної архітектури та програмного забезпечення FPGA-вузла збору даних від частотних та цифрових сенсорів. На основі результатів сформульовано такі висновки.

1) Обґрунтовано вибір гібридного архітектурного підходу з апаратно-програмним розподілом функцій. Порівняння суто апаратного (FSM), суто програмного (MCU) та гібридного (FPGA + soft-core) підходів показало, що гібридний забезпечує одночасно детермінованість апаратного вимірювання та гнучкість програмного керування. На відміну від MCU-рішень, де кількість паралельних каналів обмежена апаратними таймерами (ESP32: 8 PCNT, STM32F407: до 14 таймерів без синхронної часової бази), в розробленій FPGA-системі всі 12 каналів частотоміра працюють паралельно від спільної часової бази із детермінованим вікном вимірювання 100 мс.

2) Інтегровано мікропроцесорне ядро NIOS II в архітектуру SoC через утиліту Qsys. Обрано конфігурацію Classic low-performance без JTAG-налагодження, 20 КБ вбудованої RAM, тактова частота 50 МГц. Вибір NIOS II обґрунтовано нативною інтеграцією з FPGA Cyclone IV через шину Avalon та повним набором інструментів розробки у Quartus 15.1. Детермінований обмін між ядром та периферійними модулями (UART, I²C, 12-канальний частотомір, системний

лічильник, контролер переривань) забезпечує низькі затримки та можливість масштабування.

3) Розроблено контролер шини I²C мовою VHDL у вигляді синхронного цифрового автомата з дев'ятьма станами. Модуль забезпечує роботу в режимі ведучого з частотою шини ~ 403 кГц ($divider = 31$ при системному такті 50 МГц), підтримує механізм clock stretching, транзакції із декількох байтів та Repeated Start згідно зі специфікацією I²C [87]. Керування лініями SDA та SCL реалізовано за принципом відкритого стоку.

4) Розроблено 12-канальний блок вимірювання частоти з вікном підрахунку 100 мс, що забезпечує роздільність 10 Гц та відносну похибку порядку 10^{-4} – 10^{-5} для діапазону сотень кілогерц. Усі канали використовують спільну часову базу від модуля block_form (сигнали $qRst$, $qCnt$, qLd), що гарантує синхронність вимірювань. Максимальна кількість каналів в існуючих FPGA-частотомірах, знайдена у літературі, не перевищує 4 [45]; розроблений 12-канальний модуль є першою відомою реалізацією з такою кількістю синхронних каналів.

5) Розроблено багаторівневу архітектуру програмного забезпечення (BSP) у режимі bare-metal з чітким розмежуванням HAL, драйверів та прикладного рівня. Драйвер частотоміра працює в режимі переривань (ISR зчитує всі 12 каналів за один прохід), драйвер I²C реалізує атомарні транзакції через критичні секції, драйвер MPU6050 [88] забезпечує ініціалізацію та зчитування показників акселерометра, гіроскопа і температури. Основний цикл програми працює з фіксованою періодичністю 10 Гц, формуючи вихідні дані у форматі TSV через UART зі швидкістю 115 200 бод, обраною як достатня для запланованого обсягу трафіку.

6) Основні наукові та практичні результати розділу опубліковані в роботах [4, 5, 6, 18, 19].

РОЗДІЛ 3

НАСКРІЗНА ПЕРЕДАЧА ТЕЛЕМЕТРІЇ В ІОТ-МЕРЕЖІ ВІД FPGA-ВУЗЛА ДО ХМАРНОЇ ПЛАТФОРМИ ЧЕРЕЗ LORA

3.1 Архітектура IoT-системи та вибір топології

У Розділі 2 розроблено FPGA-вузол збору даних, що забезпечує паралельне вимірювання частоти від 12 сенсорів та зчитування показників цифрового інерціального модуля MPU6050 через I²C. Сформовані вимірювання передаються через UART до зовнішнього бездротового модуля. У цьому розділі розглянуто побудову наскрізного каналу передачі телеметрії від FPGA-вузла до хмарної платформи з можливістю моніторингу даних у мобільному застосунку.

Вибір технології LoRa для бездротової передачі та топології «зірка» для організації мережі сенсорних вузлів обґрунтовано у підрозділі 1.3 на основі аналізу компромісів між дальністю, пропускнуою здатністю та енергоспоживанням. Трирівнева архітектура «сенсор (FPGA-вузол) - шлюз (LoRa-концентратор) - хмара (REST API)» відповідає типовій організації IoT-систем збору даних (підрозділ 1.3, рис. 1.17). Загальну структуру розробленої системи подано на рисунку 3.1.

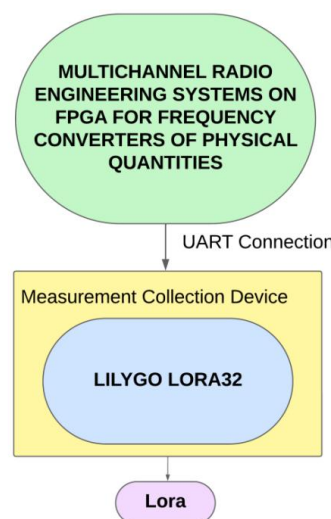


Рисунок 3.1- Блок-схема FPGA вузла

Передача зібраних даних здійснюється через бездротовий модуль LilyGO LoRa32. Цей модуль включає в себе радіотрансивер LoRa та вбудований мікроконтролер (ESP32), що забезпечує зв'язок на великі відстані при низькому енергоспоживанні. Модуль LilyGO LoRa32 виступає бездротовим інтерфейсом, що з'єднує FPGA з IoT-мережею.

Для зменшення трафіку у радіоканалі результати вимірювань серіалізуються у формат CBOR (Concise Binary Object Representation) [92] замість текстового JSON. При типовому пакеті з 12 частотних каналів, температури та 6 осей інерціального модуля CBOR-кодування формує бінарне представлення обсягом ~80 байт, тоді як JSON з іменами полів потребував би ~250 байт, що перевищує обмеження корисного навантаження одного LoRa-кадру (100 байт) [91]. До складу CBOR-пакета входять ідентифікатори вузла та концентратора для коректної агрегації на серверній стороні.

Наступним рівнем ієрархії у створеній системі є концентратор, який виступає центральним вузлом. Його основна задача полягає у зборі даних від підключених пристроїв вимірювань на базі модуля LilyGO LoRa32 (рис. 3.2). У результаті формується мережа із топологією «зірка». Центральний хаб побудований на модулі LilyGO LoRa32 і виконує функції приймання та акумуляції повідомлень від вузлів. Для доступу до серверної частини та інтеграційних сценаріїв він використовує бездротові інтерфейси Wi-Fi і Bluetooth, завдяки чому радіосегмент LoRa (від вузлів до хаба) та IP-канал (від хаба до сервера) розмежовані. Така організація спрощує розміщення хаба у зоні покриття вузлів і гнучко під'єднує його до мережевої інфраструктури.

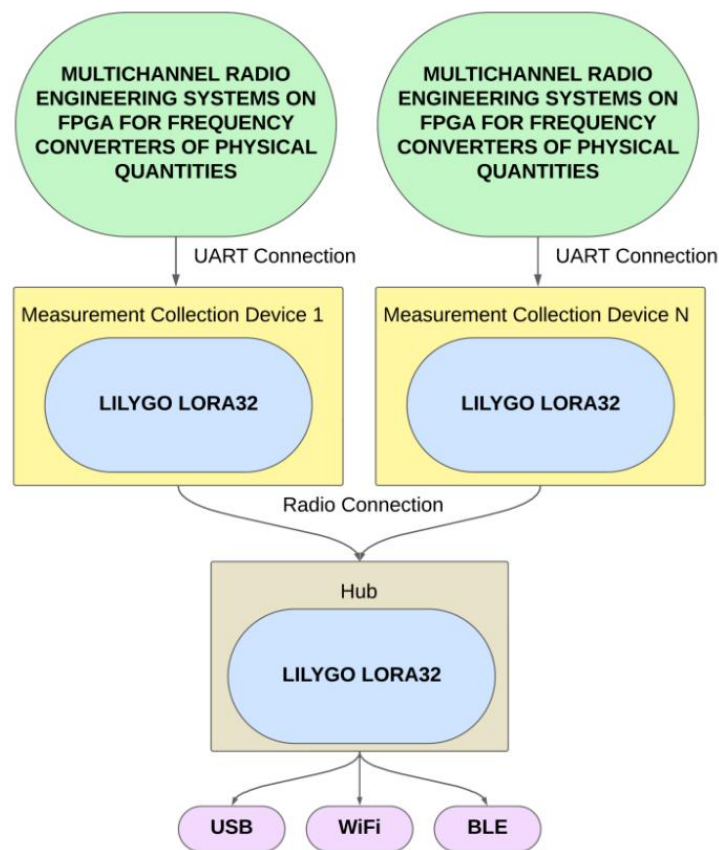


Рисунок 3.2 - Блок-схема системи збору та передачі вимірювань за допомогою протоколу LoRa

Причини вибору «зірки»: проста структура мережі полегшує управління і відлагодження: відмови окремих вузлів не призводять до розриву всієї мережі, адже кожен працює автономно. Мережу легко масштабувати: додавання нових пристроїв не потребує переналаштування всієї системи - достатньо вивести їх у радіозону хаба. До того ж у топології «зірка» зберігається висока енергоефективність: вузли передають дані без проміжних ретрансляторів, що дозволяє зменшити витрати.

Архітектура «зірка» забезпечує масштабування за рахунок здатності концентратора приймати дані від численних пристроїв збору в єдиній точці. Централізоване збирання ідентифікованих пакетів спрощує збільшення кількості сенсорів і зменшує витрати на підтримку мережі.

На серверній стороні використано REST API для отримання CBOR-повідомлень, їх декодування та збереження у базі SQLite [96]. Вибір SQLite обумовлений мінімальними вимогами до інфраструктури: база даних зберігається у єдиному файлі без потреби у окремому сервері, що спрощує розгортання на хмарній платформі AWS EC2. Передавання даних між хабом і сервером здійснюється захищеним каналом (HTTPS), що забезпечує конфіденційність, автентичність і цілісність вимірювань при роботі через публічні або ненадійні мережі. Така конфігурація мінімізує бар'єри інтеграції й спрощує під'єднання клієнтських застосунків.

Група віддалених вузлів на базі FPGA та LilyGO LoRa32 передає CBOR-пакети безпосередньо на центральний хаб (LoRa-концентратор), який акумулює трафік у єдиній точці. Далі хаб з'єднується із серверною частиною через IP-канал, який реалізовано через Wi-Fi; окремо передбачена підтримка Bluetooth для інтеграційних сценаріїв (рис. 3.3). Згідно з прикладеною схемою мережі (рис. 3.2), хаб може мати зовнішні інтерфейси підключення (Wi-Fi, BLE, а також USB для сервісного доступу), тоді як радіосегмент між вузлами та хабом повністю побудований на LoRa.

Клієнтський рівень представлено мобільним застосунком, який отримує дані з сервера, відображає доступні хаби та вузли і підтримує оновлення вимірювань у реальному часі. Це покращує гнучкість і доступність системи та зменшує потребу у локальному доступі до вузлів, що відповідає вимогам до розгортання у важкодоступних місцях.

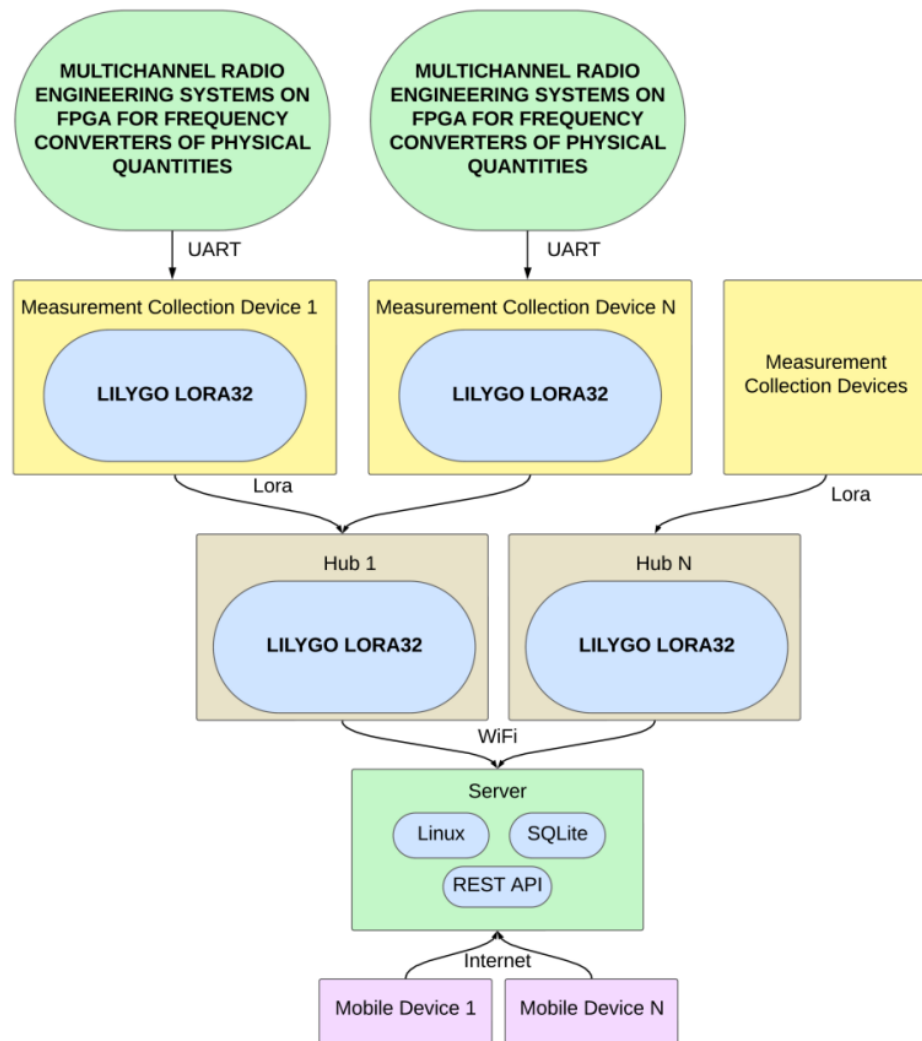


Рисунок 3.3 - Блок-схема IoT-системи збору, передачі та аналізу вимірювань

3.2 Розробка FPGA-вузла на базі LilyGO LoRa32

У цьому розділі розглянуто програмну реалізацію FPGA-вузла на базі модуля LilyGO LoRa32 (ESP32), який виконує роль проміжної ланки між FPGA-системою збору даних та центральним хабом бездротової мережі. Основне завдання вузла полягає у прийомі сенсорних даних через UART-інтерфейс, їх попередній аналіз та структурування, формуванні CBOR пакета та передачі цього пакета по бездротовому каналу LoRa (рис. 3.4). Опис охоплює архітектуру програмних компонентів, механізми обробки та передачі даних, формати та протоколи, а також засоби забезпечення надійності й цілісності інформації. Особливу увагу

приділено взаємодії між процесами в середовищі FreeRTOS, роботі з потокобезпечною чергою та механізмам фрагментації великих повідомлень для безпечної LoRa-передачі.

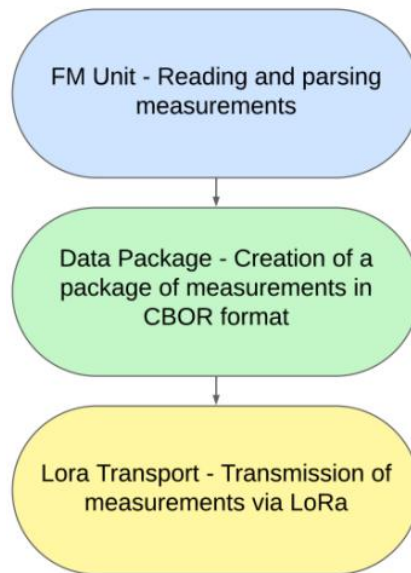


Рисунок 3.4 - Послідовність збору і передачі вимірювань

Для прийому даних з FPGA на стороні ESP32 (плата LilyGO LoRa32) використовується UART-порт Serial1 зі швидкістю 115200 бод і форматом кадру 8N1; лінії приймання й передавання прив'язуються до виділених GPIO (RX - 13, TX - 15) (рис. 3.5).

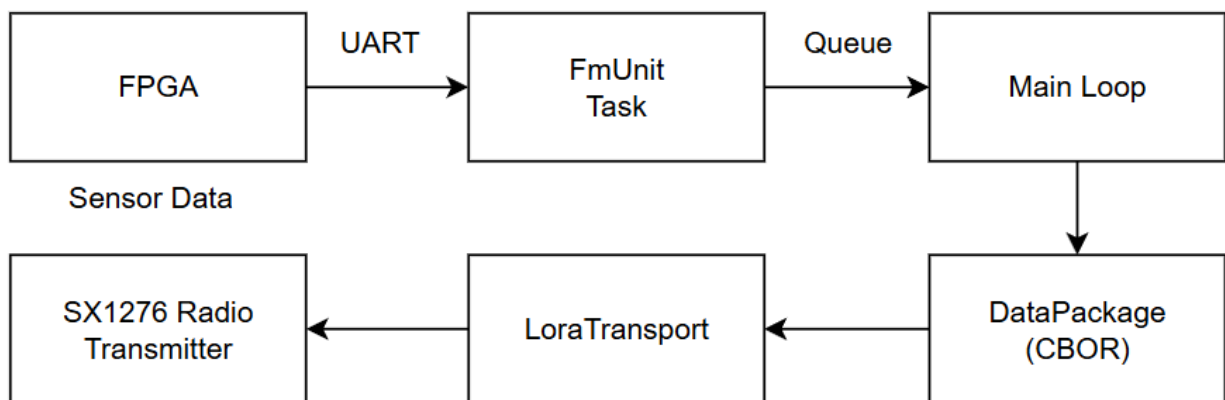


Рисунок 3.5 - Компоненти вузла задіяні у обробці і передачі вимірювань

Програмна ініціалізація зосереджена у компоненті, відповідальному за отримання вимірювань, який, окрім конфігурації UART, створює окрему FreeRTOS-задачу [89] для безперервного читання та попередньої обробки байтів, отриманих із FPGA. Вибір FreeRTOS обумовлений тим, що ця RTOS вбудована у платформу ESP-IDF [93] і нативно підтримує двоядерну архітектуру ESP32, що дозволяє ізолювати чутливий до часу прийом даних на окремому ядрі. Створена задача прив'язується до ядра 0 з пріоритетом, достатнім для гарантованого відпрацювання під час пікових навантажень, та зі збільшеним стеком (≈ 7 КБ), що враховує витрати на буферизацію та парсинг рядків. Така ізоляція дає змогу відокремити чутливий до часових проміжків прийом даних від решти логіки застосунку, зменшуючи затримку і ризик втрати байтів під час радіопередачі (рис. 3.6).

```
bool fm_begin() {
    FM_SERIAL.begin(FM_SERIAL_BAUDRATE, FM_SERIAL_CFG, FM_SERIAL_RX, FM_SERIAL_TX);
    // Function|Task name|Stack sz|Task input|Priority|Task handle|Core
    xTaskCreatePinnedToCore(fm_task, "ddLibLoraDetector", 7000, NULL, 1, &_fm_task, 0);
    return true;
}
```

Рисунок 3.6 – Фрагмент програмного коду

В основі задічі прийому даних лежить нескінченний цикл, який зчитує байти з UART і накопичує їх у динамічному буфері до виявлення ознаки завершення повідомлення. Маркер кінця повідомлення - символ нового рядка '\n'; безпосередньо перед обробкою, з буфера видаляється останій байт (рис. 3.7).

Після отримання повного повідомлення, буфер перетворюється на ASCII-рядок і передається на синтаксичний розбір (рис. 3.8).

Формат вхідних даних - це табуляцією відокремлені десяткові цілі значення в зафіксованому порядку: 12 частотних вимірів, температура, три компоненти акселерометра та три компоненти гіроскопа; очікується рівно 19 полів.

```

static void fm_task(void* unused) {
    while(1) {
        static std::vector<uint8_t> buffer;
        if (FM_SERIAL.available() > 0) {
            uint8_t byte = FM_SERIAL.read();
            if (byte == '\n') {
                FmData tmpData;
                buffer.pop_back();
                if (fm_parse_data(buffer, tmpData) {
                    if (_fm_data_queue.is_full()) _fm_data_queue.pop();
                    _fm_data_queue.push(&tmpData);
                } else {
                    LOGE(TAG, "Received damaged data");
                }
                buffer.clear();
            } else {
                buffer.push_back(byte);
            }
        }
        vTaskDelay(10);
    }
}

```

Рисунок 3.7 – Фрагмент программного коду

```

static bool fm_parse_data(const std::vector<uint8_t> &buffer, FmData &data) {
    std::string input(buffer.begin(), buffer.end());
    std::istringstream stream(input);
    std::string token;
    int count = 0;
    while (std::getline(stream, token, '\t')) {
        if (token.find_first_not_of("-0123456789") != std::string::npos) break;
        int value;
        std::istringstream value_stream(token);
        value_stream >> value;
        value *= 10;
        if (count <= FM_FREQ_MAX_ID) {
            data.freqs[count] = value;
        } else if (count == FM_TEMP_ID) {
            data.temp = value * 10;
        } else if (count == FM_ACCEL_X_ID) {
            data.accel_x = value;
        } else if (count == FM_ACCEL_Y_ID) {
            data.accel_y = value;
        } else if (count == FM_ACCEL_Z_ID) {
            data.accel_z = value;
        } else if (count == FM_GYRO_X_ID) {
            data.gyro_x = value;
        } else if (count == FM_GYRO_Y_ID) {
            data.gyro_y = value;
        } else if (count == FM_GYRO_Z_ID) {
            data.gyro_z = value;
        } else {
            break; // Too many fields
        }
        ++count;
    }
    return count == FM_PARAM_CNT;
}

```

Рисунок 3.8 – Фрагмент программного коду

Порушення формату будь-якого токена або відхилення кількості полів від очікуваної норми призводить до відбракування всього повідомлення з реєстрацією помилки у журналі, що усуває ризик потрапляння зіпсованих даних на подальші етапи обробки.

Числові значення нормалізуються під цільові внутрішні одиниці вимірювання. Для збереження роздільної здатності без використання чисел з плаваючою комою застосовано фіксовані масштабувальні коефіцієнти: усі цілі значення попередньо домножуються на 10; додатково температура масштабується ще на порядок (у сумі $\times 100$), що відповідає збереженню в сотих частках градуса. Частотні вимірювання зберігаються у десятих частках одиниці, а компоненти інерційних датчиків, у відповідно нормованих квантах, це уніфікує подальші операції серіалізації та забезпечує сталу розрядність без втрат точності у межах робочих діапазонів. Заповнення цільової структури даних виконується суворо за індексами: перші дванадцять полів потрапляють у масив частот, далі послідовно записуються температура, три акселерометричні й три гіроскопічні компоненти. Після успішної побудови структури підсумкова кількість оброблених полів додатково фіксується у логах як контрольний індикатор коректного синтаксичного розбору.

Для взаємодії декількох задач застосовано безпечний кільцевий буфер фіксованого розміру, що виключає динамічні виділення пам'яті та гарантує детермінований час виконання операцій *push/pop*. У разі заповнення черги реалізовано політику видалення найстаршого елемента перед додаванням нового, що мінімізує латентність і запобігає блокуванню UART-задачі при тимчасових затримках передавача. Семантика доступу забезпечує атомарне копіювання елементів і відсутність “гонок” між завданням приймання та головним циклом, який надалі здійснює запаковування, серіалізацію й передачу. Додатково в частині приймання використовується коротка затримка, що знижує привілейованість задачі без втрати пропускну здатності каналу.

У головному циклі програми на ESP32 формується структура *FmData* (рис. 3.9).

```
enum DataId {
    MAC_ADDR = 0, // 6 bytes array, example: {0x12, 0x34, 0x56, 0x78, 0x90, 0xAB} == 12:34:56:78:90:AB
    PKG_ID = 1, // uint32_t
    TIME_STAMP_MS = 2, // uint64_t
    FM_DATA = 3, // FmData, optional
    HUB_INFO = 4, // HubData, optional
    MAX_DATA_ID
};

#define FM_FREQS_CNT 12
struct FmData {
    uint32_t freqs[FM_FREQS_CNT];
    int16_t temp;
    int16_t accel_x;
    int16_t accel_y;
    int16_t accel_z;
    int16_t gyro_x;
    int16_t gyro_y;
    int16_t gyro_z;
};
```

Рисунок 3.9 – Фрагмент програмного коду

На цьому етапі за допомогою механізму черг FreeRTOS приймається наступний доступний об'єкт для обробки. Наприклад, функція *xQueueReceive* дозволяє зчитати елемент з черги, скопіювавши його у виділений буфер. Після цього з отриманого об'єкта зчитуються необхідні поля й присвоюються відповідним полям структури *FmData*, готуючи її до включення у пакет даних (рис. 3.10).

```
bool fm_is_data_exist() {
    return !_fm_data_queue.is_empty();
}

size_t fm_get_data_size() {
    return _fm_data_queue.size();
}

bool fm_get_data(FmData &data) {
    if (fm_is_data_exist()) {
        _fm_data_queue.back(&data);
        _fm_data_queue.pop();
        return true;
    }
    return false;
}
```

Рисунок 3.10 – Фрагмент програмного коду

Потім формується об'єкт *DataPackage*: йому присвоюється MAC-адреса пристрою, генерується унікальний ідентифікатор пакету, додається поточна мітка часу, і до пакету включається підготовлена структура *FmData*. MAC-адреса отримується викликом *esp_read_mac*- функції ESP-IDF, що повертає базову MAC-адресу, запрограмовану у чипі ESP32. Сформований пакет міститиме ідентифікатор пристрою, часову мітку та необхідні дані *FmData* для передачі (рис. 3.11).

```
if (fm_is_data_exist()) {
    FmData data;
    // Get data from the queue
    if (fm_get_data(data)) {
        DataPackage pkg;
        uint32_t pkgId = generate_pkg_id();
        pkg.setMacAddr(macAddr);
        pkg.setPkgId(pkgId);
        pkg.setTimeStampMs(millis());
        pkg.setFmData(data);
        loraTransport->sendBlocking(pkgId, pkg.getRawData());
        LOGI(TAG, "Package sent: %d", pkgId);
    }
}
```

Рисунок 3.11 – Фрагмент програмного коду

Пакет *DataPackage* далі серіалізується у формат CBOR із застосуванням бібліотеки *TinyCBOR*. Спочатку ініціалізується зовнішній енкодер викликом *cbor_encoder_init* з буфером для запису закодованих даних. Потім створюється контейнер у вигляді CBOR-карти (map) за допомогою функції *cbor_encoder_create_map*, вказуючи кількість пар «ключ–значення». Далі поступово кодуються поля пакету: наприклад, для кодування ключа «*DataId*» використовується *cbor_encode_text_string*, а для кодування значення цього поля, *cbor_encode_int* чи інша відповідна функція. Після додавання всіх пар ключ-значення контейнер закривається викликом *cbor_encoder_close_container*, завершуючи формування CBOR-структури (рис. 3.12). Бібліотека *TinyCBOR* при цьому безпосередньо записує результат у вказаний буфер без динамічного виділення пам'яті. Отриманий масив байтів готується до передачі по LoRa.

```

std::vector<uint8_t> DataPackage::getRawData() {
    std::vector<uint8_t> data(1024);
    CborEncoder encoder, mapEncoder;
    cbor_encoder_init(&encoder, data.data(), data.size(), 0);
    cbor_encoder_create_map(&encoder, &mapEncoder, CborIndefiniteLength);
    if (isMacAddrExist()) encodeMacAddr(&mapEncoder, _macAddr);
    if (isPkgIdExist()) encodePkgId(&mapEncoder, *_pkgId);
    if (isTimeStampExist()) encodeTimeStamp(&mapEncoder, *_timeStamp);
    if (isFmDataExist()) encodeFmData(&mapEncoder, *_fmData);
    if (isHubDataExist()) encodeHubData(&mapEncoder, *_hubData);
    cbor_encoder_close_container(&encoder, &mapEncoder);
    data.resize(cbor_encoder_get_buffer_size(&encoder, data.data()));
    return data;
}

```

Рисунок 3.12 – Фрагмент програмного коду

Процес передачі даних ініціюється викликом методу *loraTransport->sendBlocking(pkgId, pkg.getRawData())*. Ця функція реалізує синхронну (блокуючу) відправку даних, що означає призупинку виконання програми до повного завершення операції передачі. На вхід функція приймає унікальний ідентифікатор пакета (*pkgId*) та безпосередньо самі дані для відправки у вигляді масиву байтів (*pkg.getRawData()*) (рис. 3.13).

```

bool LoraTransport::sendBlocking(uint32_t dataPkgId, const std::vector<uint8_t> &data) {
    if (_mode != Mode::IDLE) {
        LOGE(TAG, "Already sending/receiving a package");
        return false;
    }

    if (!data.size()) {
        LOGE(TAG, "Data is empty");
        return false;
    }
}

```

Рисунок 3.13 – Фрагмент програмного коду

Перед початком передачі виконуються ключові перевірки. По-перше, функція перевіряє, чи транспортний рівень LoRa знаходиться у стані спокою (*Mode::IDLE*). Це запобігає конфліктам, якщо модуль вже зайнятий відправкою або прийомом іншого пакета. По-друге, перевіряється, що вхідний вектор даних не порожній, оскільки відправка нульової кількості байт є безглуздою операцією. У разі невдачі будь-якої з цих перевірок, функція логує помилку та повертає *false*.

Протокол LoRa має обмеження на максимальний розмір корисного навантаження одного пакету. У поточній реалізації, що визначається константою *LORA_PKG_PAYLOAD_SIZE_BYTES*, цей розмір складає 100 байт. Якщо вихідні дані перевищують цей ліміт, вони повинні бути розділені на кілька менших частин (фрагментів). Код обчислює необхідну кількість частин (*partsCnt*) перед початком передачі (рис. 3.14).

```
uint16_t partsCnt = (data.size() / LORA_PKG_PAYLOAD_SIZE_BYTES) + (data.size() % LORA_PKG_PAYLOAD_SIZE_BYTES ? 1 : 0);
```

Рисунок 3.14 – Фрагмент програмного коду

Кожен фрагмент даних упаковується у спеціальну структуру *LoraPkg*. Ця структура виконує роль транспортного контейнера, додаючи важливі метадані для коректної збірки пакета на стороні приймача (рис. 3.15).

```
#pragma pack(push, 1)
struct LoraPkg {
    uint32_t magic; // Magic number to identify the package
    uint8_t srcMacAddr[LORA_PKG_SRC_MAC_ADDR_SIZE]; // Source MAC address
    uint32_t dataPkgId; // Payload package ID
    uint16_t partsCnt; // Total parts count, package can be splitted into multiple parts if payload is more than
LORA_PKG_PAYLOAD_SIZE_BYTES bytes
    uint16_t partId; // Current part ID
    uint8_t payloadSize; // Count of bytes in the payload
    uint8_t payload[LORA_PKG_PAYLOAD_SIZE_BYTES]; // Payload data, always fixed size, even if payload is less than
LORA_PKG_PAYLOAD_SIZE_BYTES, unused bytes are filled with 0
};
#pragma pack(pop)
```

Рисунок 3.15 – Фрагмент програмного коду

Поля цієї структури включають "магічне число" для ідентифікації пакета, MAC-адресу відправника, ідентифікатор усього блоку даних, загальну кількість частин, порядковий номер поточної частини та фактичний розмір корисного навантаження в ній.

Сама передача організована в циклі. У кожній ітерації створюється екземпляр *LoraPkg*, заповнюється відповідними метаданими та черговою порцією даних. Після цього підготовлений пакет передається радіомодулю для фізичної відправки через метод *_radio.transmit()*. Оскільки виклик є блокуючим, цикл перейде до наступної частини лише після того, як поточна буде успішно надіслана (рис. 3.16).

```

for (uint16_t i = 0; i < partsCnt; i++) {
    LoraPkg pkg;
    memset(&pkg, 0, sizeof(pkg));
    pkg.magic = LORA_PKG_MAGIC;
    memcpy(pkg.srcMacAddr, _macAddr.data(), LORA_PKG_SRC_MAC_ADDR_SIZE);
    pkg.dataPkgId = dataPkgId;
    pkg.partsCnt = partsCnt;
    pkg.partId = i + 1;
    pkg.payloadSize = (i == partsCnt - 1) ? data.size() - (partsCnt - 1) * LORA_PKG_PAYLOAD_SIZE_BYTES :
LORA_PKG_PAYLOAD_SIZE_BYTES;
    memcpy(pkg.payload, data.data() + i * LORA_PKG_PAYLOAD_SIZE_BYTES, pkg.payloadSize);
    LOGI(TAG, "Transmitting package %d of %d", i+1, partsCnt);
    uint16_t state = _radio.transmit((uint8_t*)&pkg, sizeof(pkg));
    if (state != RADIOLIB_ERR_NONE) {
        LOGI(TAG, "Transmit failed: %d", state);
        return false;
    }
}
return true;
}

```

Рисунок 3.16 – Фрагмент програмного коду

У випадку збою на будь-якому етапі передачі, виконання функції негайно припиняється з результатом *false*. Якщо всі частини успішно надіслані, функція повертає *true*, сигналізуючи про повну передачу блоку даних.

Представлена реалізація FPGA-вузла (рис. 3.17) забезпечує стабільну роботу в автономному режимі для збору сенсорних даних та їх подальшої бездротової передачі за допомогою технології LoRa. В основі архітектури лежить багатозадачність операційної системи FreeRTOS, що забезпечує чіткий розподіл функцій та стабільну роботу пристрою.

Ключовим архітектурним рішенням є виділення процесу збору даних у окрему, високопріоритетну задачу (FmUnit Task). Ця задача працює на виділеному ядрі процесора (Core 0) і займається виключно читанням та парсингом даних, що

надходять із FPGA через UART. Такий підхід гарантує, що процес збору даних не буде перервано іншими операціями, що мінімізує ризик втрати вимірювань.

Для безпечної передачі даних між задачею збору та основним циклом програми використовується потокобезпечна черга, тобто *SafeQueue*. Це дозволяє ефективно роз'єднати процеси збору та обробки/передачі даних, усуваючи необхідність у складних механізмах синхронізації (як-от м'ютекси) та підвищуючи загальну продуктивність системи.

На етапі підготовки до передачі система використовує модуль *DataPackage* для серіалізації структурованих даних у формат CBOR, тобто компактне бінарне представлення об'єктів (Concise Binary Object Representation). Вибір цього формату є важливою оптимізацією, оскільки він створює значно компактніше бінарне представлення даних у порівнянні з текстовими форматами (наприклад, JSON). Це напряду впливає на ефективність передачі: зменшується час перебування в ефірі, знижується енергоспоживання та навантаження на радіоканал.

Рівень транспорту реалізований у модулі *LoraTransport*, який абстрагує всю складність роботи з LoRa-радіомодулем. Його найважливішою функцією є автоматична фрагментація великих пакетів даних. Якщо розмір серіалізованого CBOR-пакета перевищує максимальне корисне навантаження одного LoRa-кадру (100 байт), *LoraTransport* самостійно розбиває його на частини. Кожна частина загортається у власний транспортний "конверт" (*LoraPkg*) з метаданими (магічне число, MAC-адреса, ID пакета, номер та загальна кількість частин), завдяки чому приймач надійно відновлює вихідне повідомлення.

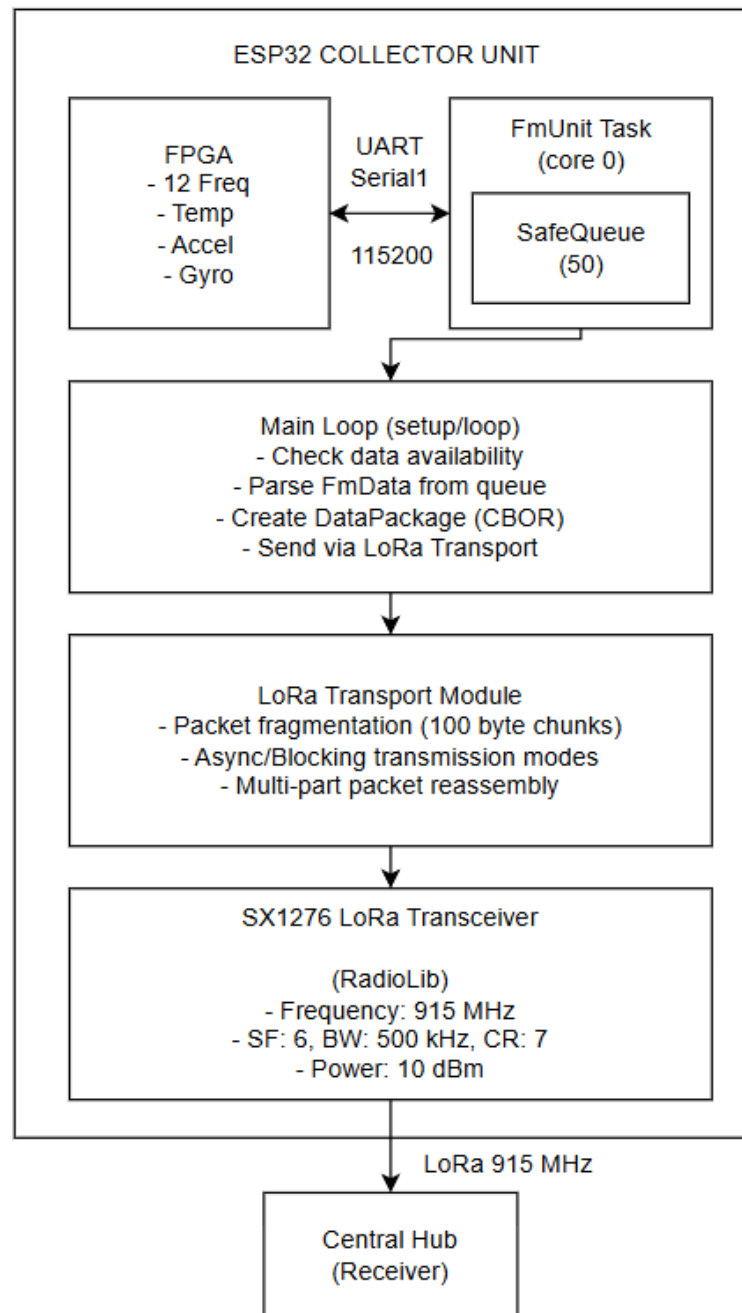


Рисунок 3.17 – Реалізація FPGA-вузла

3.3 Розробка центрального хабу на базі *LilyGO LoRa32*

Центральний хаб є ключовим вузлом в розробленій системі збору даних і виконує роль інтелектуального IoT-шлюзу між віддаленими FPGA-вузлами та центральним сервером. Його основне призначення - агрегація, попередня обробка та надійна передача телеметричної інформації. В якості апаратної платформи для

хабу було обрано пристрій LilyGO LoRa32. Цей вибір обґрунтований наявністю двоядерного мікроконтролера Espressif ESP32, який забезпечує достатню обчислювальну потужність для паралельної обробки даних, та інтегрованого радіомодуля Semtech SX1276 для дальнього зв'язку за технологією LoRa. Програмне забезпечення хабу спроектовано за модульним принципом для забезпечення гнучкості та простоти подальшої підтримки. Архітектура складається з трьох логічних рівнів: *LoraTransport* для низькорівневої роботи з радіоканалом, *DataPackage* для серіалізації та десеріалізації даних, та *ServerConn*, що відповідає за взаємодію з хмарним сервером через Wi-Fi та HTTPS (рис. 3.18).

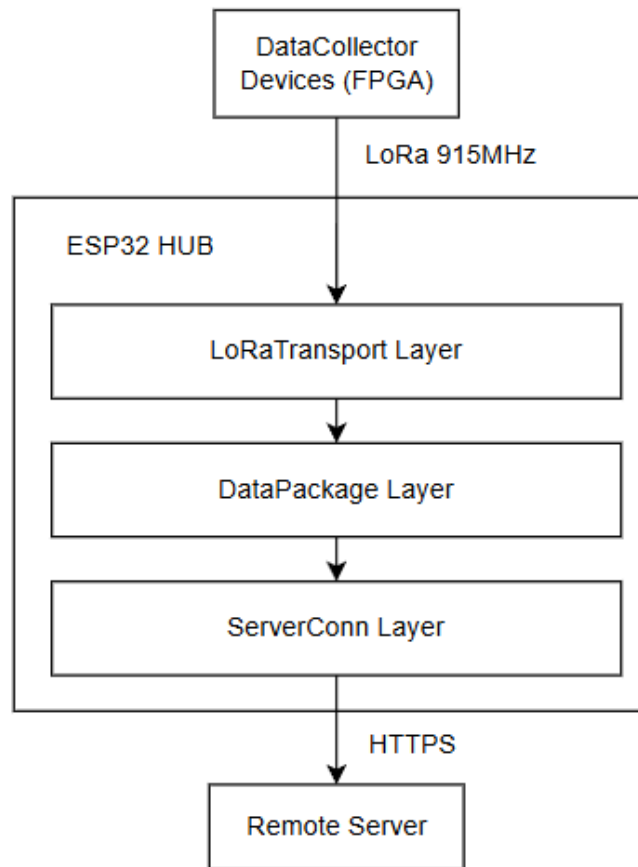


Рисунок 3.18 – Архітектура концентратора

Життєвий цикл даних в межах хабу є чітко визначеним асинхронним конвеєром (рис. 3.19). Процес починається, коли радіомодуль SX1276 приймає

LoRa-сигнал. Модуль *LoraTransport*, що працює на низькому рівні, обробляє вхідний пакет, перевіряє його цілісність та, у випадку фрагментованої передачі, накопичує частини до отримання повного повідомлення. Після успішного отримання повного пакету даних, він через callback-функцію передається на логічний рівень програми. Тут в дію вступає модуль *DataPackage*, який виконує десеріалізацію бінарних даних з формату CBOR у структурований вигляд, доступний для аналізу. Наступним кроком є збагачення даних: хаб додає до отриманої телеметрії службову інформацію, зокрема, свою унікальну MAC-адресу, що дозволяє ідентифікувати шлюз на стороні сервера. Доповнений пакет даних знову серіалізується у формат CBOR і передається модулю *ServerConn*. Щоб уникнути блокування основного потоку виконання, дані не відправляються на сервер негайно, а ставляться у спеціальну чергу, реалізовану засобами операційної системи FreeRTOS. Окрема фонові задача (task) постійно моніторить цю чергу і, як тільки з'являються нові дані, встановлює безпечне HTTPS-з'єднання та відправляє їх на сервер, що забезпечує повністю неблокуючу роботу пристрою.

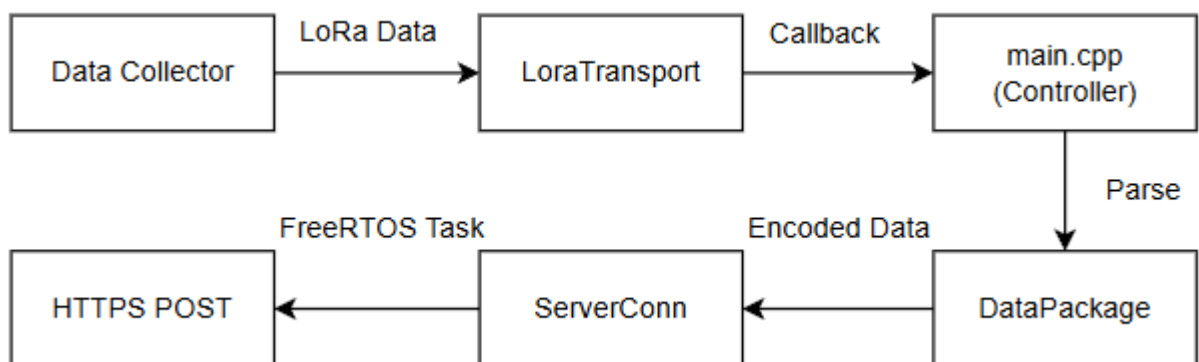


Рисунок 3.19 - Життєвий цикл даних у концентраторі: від прийому LoRa-пакета до відправки на сервер

Програмна реалізація хабу складається з кількох ключових модулів, кожен з яких виконує свою специфічну функцію. Ініціалізація включає налаштування SPI-

шини та конфігурацію радіо для роботи на частоті 915 МГц (ISM-діапазон Північної Америки) з шириною смуги 500 кГц та коефіцієнтом розширення спектру SF6 [91]. Такі параметри обрано для максимізації пропускної здатності (підрозділ 1.3, рис. 1.15): при SF6 та BW 500 кГц досягається найвища серед конфігурацій LoRa бітова швидкість, що є достатнім для передачі телеметричних пакетів з періодичністю 10 Гц на відстанях до кількох сотень метрів (рис. 3.20).

```
LoraTransport::LoraTransport(const std::vector<uint8_t> &macAddr, ErrorCallback errorCallback) : _macAddr(macAddr),
    _module(LORA_CS_PIN, LORA_IO0_PIN, LORA_RST_PIN, LORA_IO1_PIN, _spi, RADIOLIB_DEFAULT_SPI_SETTINGS),
    _radio(&_module), _mode(Mode::IDLE), _errorCallback(errorCallback) {
    if (macAddr.size() != LORA_PKG_SRC_MAC_ADDR_SIZE) {
        LOGE(TAG, "Invalid MAC address size");
        return;
    }
    _spi.begin(LORA_SCK_PIN, LORA_MISO_PIN, LORA_MOSI_PIN, LORA_CS_PIN);
    pinMode(LORA_IO1_PIN, INPUT);
    pinMode(LORA_IO2_PIN, INPUT);
    uint8_t status = _radio.begin(LORA_FREQ_MHZ, LORA_BW_KHZ, LORA_SF, LORA_CR, LORA_SYNC_WORD,
        LORA_POWER_DBM, LORA_PREAMBLE_LEN, LORA_GAIN);
    if (status != RADIOLIB_ERR_NONE) {
        LOGE(TAG, "Radio initialization failed");
    }
    _radio.implicitHeader(0);
}
```

Рисунок 3.20 – Фрагмент програмного коду

Для надійності кожен LoRa-пакет має фіксовану структуру *LoraPkg* з магічним числом *0xDEADBEEF* для валідації та полями для підтримки фрагментації, що дозволяє передавати об'ємні масиви даних. Прийом пакетів організовано асинхронно: метод *startReceiving* переводить радіо в режим безперервного прийому та реєструє функцію *pkgIsReceived* як обробник апаратного переривання. Це дозволяє системі миттєво реагувати на вхідні дані, не витрачаючи ресурси процесора на постійне опитування.

Модуль *LoraTransport* є низькорівневим драйвером, що абстрагує роботу з радіомодулем *SX1276* через бібліотеку *RadioLib* [94]. Для надійності кожен LoRa-пакет має фіксовану структуру *LoraPkg* (рис. 3.21).

```
#pragma pack(push, 1)
struct LoraPkg {
    uint32_t magic; // Магічне число 0xDEADBEEF для ідентифікації
    uint8_t srcMacAddr[6]; // MAC-адреса відправника
    uint32_t dataPkgId; // ID пакету даних
    uint16_t partsCnt; // Загальна кількість частин
    uint16_t partId; // Порядковий номер поточної частини
    uint8_t payloadSize; // Розмір корисного навантаження
    uint8_t payload[100]; // Корисне навантаження
};
#pragma pack(pop)
```

Рисунок 3.21 – Фрагмент програмного коду

```
static bool pkgIsReceivedFlag = false;
ICACHE_RAM_ATTR void pkgIsReceived(void) {
    pkgIsReceivedFlag = true;
}
void LoraTransport::loop() {
    switch (_mode) {
        case Mode::RECEIVING: {
            if (!pkgIsReceivedFlag) break;
            pkgIsReceivedFlag = false;
            LoraPkg pkg;
            memset(&pkg, 0, sizeof(pkg));
            if (_radio.getPacketLength() != sizeof(pkg)) {
                LOGI(TAG, "Received package with invalid length: %d", _radio.getPacketLength());
                break;
            }
            _radio.readData((uint8_t*)&pkg, sizeof(pkg));
            processReceivedPackage(pkg);
            break;
        }
        default:
            _radio.standby();
            break;
    }
}
```

Рисунок 3.22 – Фрагмент програмного коду

Прийом пакетів організовано асинхронно. Метод *startReceiving* реєструє функцію *pkgIsReceived* як обробник апаратного переривання від радіомодуля. Ця функція, позначена як *ICACHE_RAM_ATTR*, завантажується в оперативну пам'ять для миттєвого виконання та лише встановлює прапорець *pkgIsReceivedFlag*. Основна обробка відбувається вже в методі *loop()*, що дозволяє мінімізувати час перебування у перериванні (рис. 3.22). Модуль *DataPackage* відповідає за роботу з даними на логічному рівні. Для ефективності дані в пакеті організовані у вигляді карти з числовими ключами, які визначені в *DataId* (рис. 3.23).

```
enum DataId {
    MAC_ADDR = 0,
    PKG_ID = 1,
    TIME_STAMP_MS = 2,
    FM_DATA = 3,
    HUB_INFO = 4,
};
```

Рисунок 3.23 – Фрагмент програмного коду

Декодування даних з бінарного формату CBOR є зворотнім процесом. Метод *setRawData* послідовно викликає функції декодування для кожного типу даних. Наприклад, *decodeFmData* шукає в карті ключ *FM_DATA* і, якщо знаходить, парсить масив значень, що відповідає структурі *FmData* (рис. 3.24).

```
bool DataPackage::decodeFmData(const std::vector<uint8_t> &data, FmData &fmData) {
    CborParser parser;
    CborValue it, map, array;
    CborError err = cbor_parser_init(data.data(), data.size(), 0, &parser, &it);
    // ...
    while (!cbor_value_at_end(&map)) {
        int dataId;
        cbor_value_get_int(&map, &dataId);
        cbor_value_advance(&map);
        if (dataId == static_cast<int>(DataId::FM_DATA)) {
            cbor_value_enter_container(&map, &array);
            // Послідовно читаємо частоти
            for (size_t i = 0; i < FM_FREQS_CNT; i++) {
                cbor_value_get_int(&array, reinterpret_cast<int32_t*>(&fmData.freqs[i]));
                cbor_value_advance(&array);
            }
            int16_t* fields[] = {&fmData.temp, &fmData.accel_x, ...};
            for (auto field : fields) {
                int tmp;
                cbor_value_get_int(&array, &tmp);
                *field = static_cast<int16_t>(tmp);
                cbor_value_advance(&array);
            }
            // ...
            return true;
        }
        cbor_value_advance(&map);
    }
    return false;
}
```

Рисунок 3.24 – Фрагмент програмного коду

Модуль *ServerConn* забезпечує зв'язок з сервером. Його ініціалізація у функції *begin()* є ключовою для запуску всієї асинхронної логіки (рис. 3.25).

```

bool begin() {
    LOGI(TAG, "Initializing WiFi connection");
    WiFi.begin(WIFI_SSID.c_str(), WIFI_PASSWORD.c_str());
    WiFi.setAutoReconnect(true);

    LOGI(TAG, "Creating payload queue");
    _payloadQueue = xQueueCreate(PAYLOAD_QUEUE_SIZE, sizeof(Payload));
    if (_payloadQueue == NULL) {
        LOGE(TAG, "Failed to create payload queue");
        return false;
    }

    LOGI(TAG, "Starting server task");
    xTaskCreatePinnedToCore(serverTask, "server", 10000, NULL, 5, &_serverTaskHandle, 0);
    LOGI(TAG, "Server task created");
    if (_serverTaskHandle == NULL) {
        LOGE(TAG, "Failed to create server task");
        vQueueDelete(_payloadQueue);
        return false;
    }

    return true;
}

```

Рисунок 3.25 – Фрагмент програмного коду

Тут створюється черга FreeRTOS для зберігання пакетів даних та запускається окрема задача *serverTask*, яка буде виконуватися на ядрі 0. Розмір стеку (10000 байт) виділяється з розрахунком на потреби бібліотек Wi-Fi та TLS (рис. 3.26).

Мережева взаємодія між хабом та FPGA-вузлами побудована за топологією "зірка", де хаб є центральним координатором. Ідентифікація кожного вузла відбувається за його унікальною MAC-адресою. Ця адреса використовується як ключ для відновлення фрагментованих пакетів, що дозволяє одночасно обробляти дані від кількох пристроїв. Логіка цього процесу зосереджена в методі *LoraTransport::processReceivedPackage*. Коли надходить перший фрагмент пакету (*partId == 1*), для даного MAC створюється новий запис у *std::map*. Наступні фрагменти доповнюють існуючий запис.

```

static void serverTask(void *pvParameters) {
    while (true) {
        Payload payload;
        if (!isWifiConnected() || xQueueReceive(_payloadQueue, &payload, 0) != pdPASS) {
            vTaskDelay(1);
            continue;
        }
        WiFiClientSecure client;
        client.setCACert(CA_CERT);
        HTTPClient https;
        if (https.begin(SERVER_DATA_PATH.c_str())) {
            // Add header
            https.addHeader("Content-Type", "application/octet-stream");
            int httpResponseCode = https.POST(payload.data, payload.size);
            LOGI(TAG, "HTTP Response: %d", httpResponseCode);
            if (httpResponseCode == HTTP_CODE_OK) LOGI(TAG, "POST Success");
            else LOGE(TAG, "POST Failed, Error: %d (%s)", httpResponseCode, https.errorToString(httpResponseCode).c_str());
            https.end();
        } else {
            LOGE(TAG, "Failed to connect to server.");
        }
        delete[] payload.data;
        vTaskDelay(1);
    }
}

```

Рисунок 3.26 – Фрагмент програмного коду

```

void LoraTransport::processReceivedPackage(LoraPkg &pkg) {
    if (pkg.magic != LORA_PKG_MAGIC) return;
    uint64_t identifier = 0;
    memcpy(&identifier, pkg.srcMacAddr, LORA_PKG_SRC_MAC_ADDR_SIZE);
    // Якщо це новий пакет (частина 1)
    if (pkg.partId == 1) {
        // Якщо пакет складається з однієї частини, одразу відправляємо
        if (pkg.partsCnt == 1) {
            _receivedPkgCallback(/*...*/);
        } else { // Інакше, зберігаємо першу частину
            ReceivedPkg receivedPkg;
            _receivedPkgs[identifier] = receivedPkg;
        }
    } else if (_receivedPkgs.count(identifier)) { // Якщо це продовження
        // ... перевірки коректності ...
        _receivedPkgs[identifier].data.insert(/*...*/); // Додаємо дані
        // Якщо це остання частина
        if (_receivedPkgs[identifier].partId == pkg.partsCnt) {
            _receivedPkgCallback(
                _receivedPkgs[identifier].srcMacAddr,
                _receivedPkgs[identifier].dataPkgId,
                _receivedPkgs[identifier].data
            );
            _receivedPkgs.erase(identifier); // Видаляємо з буфера
        }
    }
}

```

Рисунок 3.27 – Фрагмент програмного коду

Коли надходить останній фрагмент, повний пакет даних передається через callback *_receivedPkgCallback* на верхній рівень для подальшої обробки, а запис з *std::map* видаляється (рис. 3.27).

На верхньому рівні, у файлі *main.cpp*, знаходиться реалізація *receivedPkgCallback*. Ця функція є точкою, де з'єднуються всі модулі: вона приймає зібрані дані від *LoraTransport*, використовує *DataPackage* для їх парсингу та збагачення, і нарешті, передає фінальний пакет модулю *ServerConn* для відправки на сервер (рис. 3.28).

```
void receivedPkgCallback(const std::vector<uint8_t> &srcMacAddr, uint32_t pkgId, const std::vector<uint8_t> &data) {
    DataPackage dataPackage;
    uint8_t hubMac[DATA_PACKAGE_MAC_ADDR_SIZE];
    WiFi.macAddress(hubMac);
    dataPackage.setRawData(data);
    // Add hub data to the package
    HubData hubData;
    memcpy(hubData.macAddr, hubMac, DATA_PACKAGE_MAC_ADDR_SIZE);
    dataPackage.setHubData(hubData);
    ServerConn::sendData(dataPackage.getRawData());
}
```

Рисунок 3.28 – Фрагмент програмного коду

Така архітектура забезпечує чіткий та логічний потік даних, де кожен модуль виконує свою відокремлену функцію, що спрощує розуміння, тестування та подальшу модифікацію системи.

Точкою входу в програму є файл *main.cpp*. Вся початкова конфігурація системи відбувається у функції *setup()*, яка викликається один раз при старті мікроконтролера. На цьому етапі відбувається ініціалізація послідовного порту для логування, запуск модуля *ServerConn*, який, у свою чергу, підключається до Wi-Fi та запускає фонову задачу для відправки даних. Далі визначається MAC-адреса пристрою, яка буде використовуватися для ідентифікації хабу, і створюється екземпляр класу *LoraTransport*. В залежності від режиму компіляції (TEST), хаб або переходить у режим прийому даних з LoRa-ефіру, або готується до роботи з тестовими даними (рис. 3.29).

```

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  LOGI(TAG, "RF hub");

  if (!ServerConn::begin()) {
    LOGE(TAG, "Failed to start server");
    esp_restart();
  }

  uint8_t mac[DATA_PACKAGE_MAC_ADDR_SIZE];
  WiFi.macAddress(mac);
  macAddr = std::vector<uint8_t>(mac, mac + 6);
  loraTransport = new LoraTransport(macAddr, transportErrorCallback);
  LOGI(TAG, "LoraTransport initialized");
  #ifndef TEST
  loraTransport->startReceiving(receivedPkgCallback);
  #endif

  LOGI(TAG, "Device MAC Address: %02X:%02X:%02X:%02X:%02X:%02X", mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
}

```

Рисунок 3.29 – Фрагмент програмного коду

Основний цикл програми, функція *loop()*, є відносно простим завдяки багатозадачній архітектурі. Його основна відповідальність - викликати метод *loraTransport->loop()* у робочому режимі, який перевіряє стан радіомодуля та обробляє внутрішні події. Для відладки та тестування системи передбачено режим TEST. У цьому режимі, замість очікування реальних даних, *loop()* з певною періодичністю генерує тестові пакети даних та викликає *receivedPkgCallback* напряду, що дозволяє імітувати отримання даних від FPGA-вузлів та перевіряти весь ланцюжок обробки та відправки на сервер без наявності реального LoRa-сигналу. Виклик *vTaskDelay(1)* в кінці циклу передає керування планувальнику FreeRTOS, дозволяючи виконуватися іншим задачам, таким як *serverTask* (рис. 3.30).

Розроблений центральний хаб забезпечує безперервну агрегацію телеметрії від множини вузлів для збору даних у розподілених системах моніторингу. Застосована модульна, асинхронна архітектура забезпечує високу продуктивність та стабільність роботи.

```

void loop() {
    static unsigned long lastTimeStampMs = 0;
    if (millis() - lastTimeStampMs > 5000) {
        LOGI(TAG, "Time elapsed: %d ms, server connected: %d", millis(), ServerConn::isConnected());
        lastTimeStampMs = millis();
    }

    #ifndef TEST
    // В робочому режимі обробляємо події LoRa
    loraTransport->loop();
    #else
    // В тестовому режимі періодично генеруємо тестові дані
    static unsigned long lastStubDataTimeMs = 0;
    if (millis() - lastStubDataTimeMs > 3000) {
        LOGI(TAG, "Sending stub data");
        receiveStubData(receivedPkgCallback);
        lastStubDataTimeMs = millis();
    }
    #endif

    vTaskDelay(1); // Передача керування планувальнику FreeRTOS
}

```

Рисунок 3.30 – Фрагмент програмного коду

Використання мікроконтролера ESP32 надало достатні обчислювальні ресурси для одночасної обробки даних з LoRa-каналу та управління захищеним Wi-Fi з'єднанням. Технологія LoRa підтвердила свою ефективність для передачі даних на великі відстані, а формат серіалізації CBOR дозволив значно зменшити обсяг трафіку. Застосування операційної системи реального часу FreeRTOS стало ключовим для реалізації неблокуючої, багатозадачної логіки, що є критично важливим для пристроїв такого типу.

3.4 Хмарний сервіс AWS та мобільний додаток для моніторингу сенсорних даних

Для обробки та зберігання даних, що надходять від IoT-пристроїв, було розроблено серверну частину, призначену для розгортання на хмарній платформі, такій як AWS EC2. Роль сервера полягає у прийомі телеметричних даних, їх декодуванні, збереженні у базі даних та наданні доступу до них через API. Початкове налаштування серверного середовища вимагає виконання кількох

ключових кроків. Передусім, необхідно налаштувати мережевий екран для дозволу вхідних з'єднань на порт 8080. Для забезпечення безпечної передачі даних генерується підписаний SSL/TLS сертифікат за допомогою бібліотеки OpenSSL. Це створює приватний ключ (*selfsigned.key*) та публічний сертифікат (*selfsigned.crt*), які використовуються для встановлення HTTPS-з'єднання (рис. 3.31).

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout selfsigned.key -out selfsigned.crt
```

Рисунок 3.31 – Фрагмент програмного коду

Програмне забезпечення сервера розроблено на мові Python, тому середовище вимагає наявності Python 3, менеджера пакетів *pip* та інструменту *venv* для створення ізолюваних віртуальних середовищ. Усі необхідні бібліотеки Python перелічені у файлі *requirements.txt* і встановлюються однією командою. Для збереження даних використовується легка система керування базами даних SQLite (рис. 3.32).

```
pip install -r requirements.txt
```

Рисунок 3.32 – Фрагмент програмного коду

В основі сервера лежить веб-застосунок, створений на базі фреймворку FastAPI [95] - асинхронного Python-фреймворку з автоматичною генерацією OpenAPI-документації та вбудованою валідацією даних, що є сучасним високопродуктивним інструментом для побудови API на Python. Застосунок працює під керуванням ASGI-сервера Uvicorn, який ефективно обробляє асинхронні запити. Сервер надає REST API (*/device/data*) для прийому телеметричних даних від IoT-пристроїв методом *HTTP POST*. Ключовою

особливістю реалізації є використання формату CBOR для серіалізації даних. CBOR було обрано замість текстових форматів, як-от JSON, через менший розмір повідомлень та швидше декодування, що є критичною перевагою для IoT-пристроїв з обмеженими ресурсами. При отриманні запиту сервер декодує бінарне CBOR-повідомлення (рис. 3.33).

```
@app.post("/device/data")
async def receive_data(request: Request):
    payload = await request.body()
    decoded_payload = cbor2.loads(payload)
```

Рисунок 3.33 – Фрагмент програмного коду

Після успішного декодування дані обробляються та зберігаються в локальній базі даних SQLite з назвою *data.db*. База даних містить таблицю *measurements*, яка створюється автоматично при першому запуску сервера (рис. 3.34).

```
@app.on_event("startup")
async def startup():
    # ...
    global db_conn
    db_conn = sqlite3.connect("data.db")
    db_conn.execute("CREATE TABLE IF NOT EXISTS measurements (id INTEGER PRIMARY KEY AUTOINCREMENT,
db_timestamp_ms INTEGER, hub_mac_addr TEXT, unit_mac_addr TEXT, pkg_id INTEGER, pkg_timestamp_ms INTEGER, fm_data
TEXT)")
    db_conn.commit()
```

Рисунок 3.34 – Фрагмент програмного коду

Структура таблиці призначена для зберігання ключової інформації з кожного пакету даних: мітка часу бази даних, MAC-адреси центрального хабу та вузла, ідентифікатор пакета, власна мітка часу вузла та самі виміряні дані (*fm_data*). MAC-адреси, що надходять у вигляді масивів байтів, перетворюються у рядковий формат для зручного читання перед збереженням. Взаємодія з базою даних реалізована за допомогою стандартної бібліотеки *sqlite3* мови Python, що забезпечує надійне та постійне зберігання всієї вхідної телеметрії (рис. 3.35).

```

unit_mac_addr_str = ":".join(f"{byte:02X}" for byte in decoded_payload[DP_UNIT_MAC_ADDR_ID])
hub_mac_addr_str = ":".join(f"{byte:02X}" for byte in decoded_payload[DP_HUB_MAC_ADDR])
db_conn.execute("INSERT INTO measurements (db_timestamp_ms, hub_mac_addr, unit_mac_addr, pkg_id, pkg_timestamp_ms,
fm_data) VALUES (?, ?, ?, ?, ?, ?)",
(int(time.time() * 1000), hub_mac_addr_str, unit_mac_addr_str, int(decoded_payload[DP_PKG_ID]),
int(decoded_payload[DP_TIME_STAMP_MS]), str(decoded_payload[DP_FM_DATA])))
db_conn.commit()

```

Рисунок 3.35 – Фрагмент програмного коду

Для надання доступу до зібраних даних сервер реалізує REST API через HTTP GET, для запитів до бази даних. */info/hub_devices* повертає список усіх унікальних центральних хабів, що надсилали дані. Аналогічно, */info/unit_devices* надає перелік усіх кінцевих пристроїв, з можливістю фільтрації за MAC-адресою конкретного хабу. Найбільш функціональним є */info/measurements*, який дозволяє отримувати вимірювання. Він підтримує фільтрацію як за MAC-адресою хабу, так і кінцевого пристрою, а також може обмежувати вибірку останньою N-кількістю записів. Це є корисним для додатків, моніторингу або візуалізації даних (рис. 3.36).

```

@app.get("/info/measurements")
async def get_measurements(hub_mac_addr: str = Query(None, description="Hub MAC address, if not provided, all measurements will be returned"),
unit_mac_addr: str = Query(None, description="Unit MAC address, if not provided, all measurements will be returned"),
last_n_measurements: int = Query(None, description="Number of last measurements to return, if not provided, all measurements will be returned")):
    global db_conn
    if hub_mac_addr and unit_mac_addr:
        measurements = db_conn.execute("SELECT pkg_timestamp_ms, fm_data FROM measurements WHERE hub_mac_addr = ? AND unit_mac_addr = ? ORDER BY id DESC LIMIT ?", (hub_mac_addr, unit_mac_addr, last_n_measurements)).fetchall()
    elif hub_mac_addr:
        measurements = db_conn.execute("SELECT pkg_timestamp_ms, fm_data FROM measurements WHERE hub_mac_addr = ? ORDER BY id DESC LIMIT ?", (hub_mac_addr, last_n_measurements)).fetchall()
    elif unit_mac_addr:
        measurements = db_conn.execute("SELECT pkg_timestamp_ms, fm_data FROM measurements WHERE unit_mac_addr = ? ORDER BY id DESC LIMIT ?", (unit_mac_addr, last_n_measurements)).fetchall()
    else:
        measurements = db_conn.execute("SELECT pkg_timestamp_ms, fm_data FROM measurements ORDER BY id DESC LIMIT ?", (last_n_measurements,)).fetchall()
    logging.info(f"Measurements: {measurements}")
    return measurements

```

Рисунок 3.36 – Фрагмент програмного коду

Запуск сервера здійснюється через головний скрипт, який ініціалізує та запускає Uvicorn сервер для прослуховування вхідних з'єднань на порту 8080 з використанням раніше згенерованих SSL/TLS ключів (рис. 3.37).

```
if __name__ == "__main__":  
    uvicorn.run(  
        "server:app",  
        host="0.0.0.0",  
        port=8080,  
        ssl_certfile="ssl_keys/selfsigned.crt",  
        ssl_keyfile="ssl_keys/selfsigned.key"  
    )
```

Рисунок 3.37 – Фрагмент програмного коду

Таким чином, розроблена серверна інфраструктура забезпечує надійний та масштабований механізм для збору, зберігання та надання доступу до телеметричних даних. Наданий API дозволяє зовнішнім клієнтам легко інтегруватися з системою для отримання необхідної інформації. Наступним логічним кроком є розробка клієнтського програмного забезпечення, а саме мобільного застосунку, який буде використовувати цей API для візуалізації отриманих вимірювань та моніторингу стану IoT-пристроїв у реальному часі.

Розроблено мобільний додаток для моніторингу сенсорних даних, що функціонує на платформі Android. Застосунок, написаний на мові програмування Kotlin, призначений для взаємодії з хмарним бекендом з метою отримання та візуалізації телеметричної інформації, що надходить від мережі IoT-пристроїв. Основною метою додатку є надання користувачеві зручного інтерфейсу для моніторингу стану системи в реальному часі.

Взаємодія мобільного додатку із сервером відбувається через REST API. Для інкапсуляції логіки мережевих запитів створено допоміжний клас *NetworkHelper*, який використовує бібліотеку *OkHttpClient* для виконання HTTP-запитів. Конфігурація додатку містить визначені URL-адреси для доступу до різних

кінцевих точок API, що дозволяє отримувати списки пристроїв та їхні вимірювання (рис. 3.38).

```
private val serverAddr = "https://34.207.219.250:8080"
private val apiInfo = "$serverAddr/info"
private val apiHubDevices = "$apiInfo/hub_devices"
private val apiUnitDevices = "$apiInfo/unit_devices"
private var apiMeasurements = "$apiInfo/measurements"
```

Рисунок 3.38 – Фрагмент програмного коду

Ключовим аспектом є забезпечення безпечної передачі даних між клієнтом та сервером, що реалізовано за допомогою протоколу HTTPS. З огляду на використання підписаного SSL-сертифіката на сервері, у додатку налаштовано спеціальний SSL-контекст. Це включає завантаження сертифіката з ресурсів додатку, створення *KeyStore* та ініціалізацію *TrustManagerFactory* для валідації. Такий підхід гарантує шифрування трафіку та забезпечує конфіденційність даних (рис. 3.39).

```
private fun createSecureClient(context: Context): OkHttpClient {
    try {
        val cf = CertificateFactory.getInstance("X.509")
        val certInput: InputStream = context.resources.openRawResource(R.raw.selfsigned)
        val ca: Certificate = cf.generateCertificate(certInput)
        certInput.close()
        val keyStore = KeyStore.getInstance(KeyStore.getDefaultType()).apply {
            load(null, null)
            setCertificateEntry("ca", ca)
        }
        val tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm()).apply {
            init(keyStore)
        }
        val sslContext = SSLContext.getInstance("TLS").apply {
            init(null, tmf.trustManagers, null)
        }
        return OkHttpClient.Builder()
            .sslSocketFactory(
                sslContext.socketFactory,
                tmf.trustManagers[0] as javax.net.ssl.X509TrustManager
            )
            .hostnameVerifier { _, _ -> true } // Disable hostname verification
            .build()
    } catch (e: Exception) {
        throw RuntimeException("Failed to create secure client", e)
    }
}
```

Рисунок 3.39 – Фрагмент програмного коду

Головний екран додатку реалізує користувацький інтерфейс для перегляду даних. Для уникнення блокування основного потоку, всі мережеві операції виконуються асинхронно за допомогою *Coroutines* у Kotlin. У методі *startMonitoringServer* запускається підпрограма, яка відповідає за періодичне оновлення даних. У циклі спершу перевіряється доступність сервера, після чого оновлюється статус з'єднання, що відображається користувачеві (рис. 3.40).

```
private fun startMonitoringServer() {
    CoroutineScope(Dispatchers.IO).launch {
        val networkHelper = NetworkHelper(this@MainActivity)

        while (isActive) {
            if (!networkHelper.isServerReachable(serverAddr)) {
                withContext(Dispatchers.Main) {
                    updateConnectionStateUI(false)
                }
                delay(1000)
                continue
            } else {
                withContext(Dispatchers.Main) {
                    updateConnectionStateUI(true)
                }
            }
            // ... business logic for updating UI and fetching data ...
            delay(500)
        }
    }
}
```

Рисунок 3.40 – Фрагмент програмного коду

Інтерфейс надає можливість фільтрації даних за конкретними IoT-пристроями. Два випадаючі списки (Spinner) дозволяють обрати концентратор (*hub*) та вузол (*unit*) за їх MAC-адресою. Додаток послідовно завантажує списки доступних пристроїв. На основі вибору користувача, формується запит до сервера для отримання останніх вимірювань, які потім відображаються у табличному вигляді. Таблиця оновлюється періодично, що забезпечує моніторинг показників у режимі, близькому до реального часу (рис. 3.41).

```

val queryParams = emptyMap<String, String>().toMutableMap()
if (selectedHubMac != anyItemInTheList) {
    queryParams += mapOf("hub_mac_addr" to selectedHubMac)
}
if (selectedUnitMac != anyItemInTheList) {
    queryParams += mapOf("unit_mac_addr" to selectedUnitMac)
}
queryParams += mapOf("last_n_measurements" to measurementsCnt.toString())

val measurements = parseMeasurements(networkHelper.get(apiMeasurements, queryParams) ?: "")

```

Рисунок 3.41 – Фрагмент програмного коду

Дані, отримані від сервера, надходять у форматі JSON. У додатку реалізовано функції для їх парсингу. Наприклад, функція *parseMeasurements* (рис. 3.42) обробляє JSON-масив вимірювань і перетворює його у структуру *Map* для подальшої обробки. На основі цих даних динамічно генеруються рядки таблиці (*TableLayout*) для візуалізації показників. Функція *updateTable* відповідає за очищення старих даних та заповнення таблиці новими, що гарантує актуальність відображеної інформації, (рис. 3.43).

```

fun parseMeasurements(response: String): Map<Int, String> {
    val result = mutableMapOf<Int, String>()
    val jsonArray = JSONArray(response)

    for (i in 0 until jsonArray.length()) {
        val entry = jsonArray.getJSONArray(i)
        val key = entry.getInt(0)
        val value = entry.getString(1)
        result[key] = value
    }

    return result
}

```

Рисунок 3.42 – Фрагмент програмного коду

Connection state: Connected		Connection state: Connected	
Hub Mac	14:2B:2F:9F:6F:F8	Hub Mac	14:2B:2F:9F:6F:F8
Unit Mac	14:2B:2F:9F:6F:F8	Unit Mac	Any
MEASUREMENTS			
Time	Data	Time	
203961	[84390, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9760, -31, 165, -27, 13, 6]	12133	14:2B:2F:9F:6F:01 00,
200916	[84380, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9760, -15, 15, -27, 13, 0]	9089	14:2B:2F:9F:6F:04 00,
197871	[84370, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9760, -29, 164, -27, 13, 3]	6045	[84340, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9710, -29, 164, -27, 13, 3]
194826	[84360, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9740, -31, 165, -27, 13, 6]	3001	[84330, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9790, -15, 15, -27, 13, 0]
191781	[84350, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9760, -31, 165, -27, 13, 6]	69981	[84350, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9760, -31, 165, -27, 13, 6]
188736	[84340, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9710, -29, 164, -27, 13, 3]	66936	[84340, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9710, -29, 164, -27, 13, 3]
185691	[84330, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9790, -15, 15, -27, 13, 0]	63891	[84330, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9790, -15, 15, -27, 13, 0]
182646	[84420, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9760, -29, 164, -27, 13, 3]	60846	[84420, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21800, 9760, -29, 164, -27, 13, 3]

Рисунок 3.43 - Вигляд інтерфейсу мобільного додатку

3.5 Висновки до розділу 3

У третьому розділі виконано побудову наскрізного каналу передачі телеметрії від FPGA-вузла збору даних (Розділ 2) до хмарної платформи через бездротову мережу LoRa із можливістю моніторингу у мобільному застосунку. Основні результати розділу полягають у такому:

1) Автором запропоновано архітектуру IoT-системи з топологією «зірка», в якій FPGA-вузли виконують роль периферійних одиниць багатоканального збору та первинної обробки даних з підключених сенсорів (частотних перетворювачів, датчиків температури, акселерометрів). Передача даних здійснюється через бездротові модулі LilyGO LoRa32 на центральний концентратор. На відміну від багатострибкових mesh-топологій, запропонована архітектура забезпечує простоту управління, високу енергоефективність вузлів та легке масштабування системи без необхідності переналаштування існуючих пристроїв.

2) Розроблено програмну реалізацію LoRa-передавача FPGA-вузла на базі модуля LilyGO LoRa32 (ESP32) у середовищі FreeRTOS. Реалізовано прийом даних через UART-інтерфейс зі швидкістю 115200 бод (формат 8N1), багатозадачну архітектуру з ізоляцією критичних до часу операцій прийому від решти логіки застосунку, потокобезпечну чергу для буферизації вимірювань та механізм синтаксичного розбору ASCII-повідомлень. Задача прийому виконується на окремому ядрі процесора з виділеним стеком 7 кБ, що мінімізує затримки та ризик втрати байтів під час радіопередачі.

3) Автором запропоновано протокол передачі вимірювальних даних з використанням компактної серіалізації CBOR, що зменшує обсяг трафіку у радіоканалі LoRa. Реалізовано механізм фрагментації великих повідомлень з урахуванням обмеження корисного навантаження LoRa-пакета (100 байт). До складу пакета включено ідентифікатори вузла та концентратора, що забезпечує коректну агрегацію даних на серверній стороні.

4) Розроблено серверну частину системи на основі REST API (Python/FastAPI [95]) з базою даних SQLite для зберігання вимірювань. Передача даних між концентратором та сервером здійснюється захищеним каналом HTTPS, що забезпечує конфіденційність, автентичність та цілісність інформації при роботі через публічні мережі. Серверна архітектура мінімізує бар'єри інтеграції та спрощує підключення клієнтських застосунків.

5) Автором розроблено мобільний Android-застосунок мовою Kotlin для візуалізації та моніторингу вимірювальних даних у режимі реального часу. Реалізовано асинхронне виконання мережевих запитів за допомогою Coroutines, налаштовано SSL-контекст для роботи з HTTPS, створено інтерфейс фільтрації даних за MAC-адресами концентраторів та вузлів. Застосунок забезпечує періодичне оновлення табличного відображення вимірювань з інтервалом 500 мс.

6) Основні наукові та практичні результати розділу опубліковані в роботах [7, 8, 14, 15, 21].

РОЗДІЛ 4

МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ТА FPGA-РЕАЛІЗАЦІЯ ГЕНЕРАТОРІВ ДЕТЕРМІНОВАНОГО ХАОСУ

4.1 Уніфікована конвеєрна архітектура генераторів хаотичних сигналів на FPGA

На основі аналізу, виконаного у підрозділі 1.4, для FPGA-реалізації обрано п'ять хаотичних систем третього порядку: Лоренца, Росслера, Нозе-Гувера, Рікітакі та модифіковану систему Аніщенко-Астахова. Вибір обґрунтовано різноманітністю механізмів хаотичної динаміки (дволопатекий, спіральний, тороїдальний, двопетлевий аттрактори) та різним ступенем складності нелінійностей, від простих поліноміальних добутків до трансцендентної функції $\tanh$. Апаратною платформою обрано мікросхему EP4CE15F23C8 сімейства Cyclone IV E з тактовою частотою 50 МГц. Для реалізації всіх п'яти систем розроблено уніфіковану конвеєрну архітектуру, яка забезпечує спільний підхід до дискретизації, арифметики та синхронізації обчислень.

Реалізація хаотичних систем на FPGA потребує переходу від безперервних диференціальних рівнянь до дискретних різницеких схем. Для чисельного інтегрування обрано метод Ейлера першого порядку, за яким оновлення змінної стану на кожному кроці обчислюється за формулою

$$x[k + 1] = x[k] + f(x[k], y[k], z[k]) \times dt \quad (4.1)$$

де dt є фіксованим кроком інтегрування. Метод Ейлера потребує лише одного обчислення правої частини системи на крок, тоді як метод Рунге-Кутти четвертого порядку (RK4) вимагає чотирьох таких обчислень із зберіганням проміжних коефіцієнтів k_1, k_2, k_3, k_4 . Для тривимірної хаотичної системи це означає збільшення кількості операцій множення у 4 рази та відповідне зростання

використання DSP-блоків FPGA. У працях [70, 77, 78, 79] показано, що метод Ейлера широко застосовується для FPGA-реалізації хаотичних генераторів, оскільки забезпечує достатню точність відтворення топології атракторів при помірних апаратних витратах. Для апаратної реалізації обрано крок $dt = 0.02$, а для верифікації (підрозділ 4.5) додатково використано $dt = 0.005$, що дозволяє оцінити вплив кроку дискретизації на точність.

Для представлення числових значень використано формат з фіксованою комою Q16.16. Це 32-бітний формат зі знаком, де старші 16 біт (включно зі знаковим) відводяться під цілу частину, а молодші 16 біт під дробову. Діапазон представлення становить ± 32767 , що повністю покриває амплітуди коливань досліджуваних систем (зазвичай не більше 50 одиниць). Абсолютна точність формату дорівнює $2^{-16} \approx 1.5 \times 10^{-5}$. Використання фіксованої коми замість плаваючої дозволяє замінити складні блоки IEEE-754 на цілочисельну арифметику, що суттєво зменшує кількість логічних елементів та підвищує досяжну тактову частоту. Апаратна реалізація множення у форматі Q16.16 потребує обчислення 64-бітного добутку (формат Q32.32) з подальшою нормалізацією арифметичним зсувом вправо на 16 біт. Параметри диференціальних рівнянь та крок інтегрування конвертуються у Q16.16 заздалегідь, тобто $dt = 0.02$ представляється як 32-бітна константа `0x0000051E` ($0.02 \times 2^{16} \approx 1310$).

Усі розроблені модулі генераторів хаотичних сигналів реалізовано мовою опису апаратури SystemVerilog/Verilog з уніфікованим інтерфейсом. Кожен модуль має стандартні входні сигнали тактування (*clk*) та скидання (*rst*), а також три 32-бітні вихідні порти (*x, y, z*), що видають поточні значення змінних стану у форматі Q16.16. Параметри конкретної хаотичної системи (коефіцієнти рівнянь, крок інтегрування, початкові умови) задаються як Verilog-параметри, що дозволяє змінювати їх без переписування коду модуля.

Внутрішня структура генераторів організована за принципом конвеєрної обробки даних (рис. 4.1). Логіка роботи поділяється на два основні процеси: процес керування станами та арифметичні конвеєри. Процес керування ініціює обчислення наступного кроку, встановлюючи мітки запуску (*start_flag*) для трьох арифметичних блоків, та очікує сигнали завершення (*done_flag*) від кожного з них. Лише коли всі три нові значення (x_{next} , y_{next} , z_{next}) обчислені, відбувається синхронне оновлення регістрів стану. Це гарантує, що всі змінні оновлюються одночасно на основі одного й того ж попереднього стану, що є критичним для коректного чисельного інтегрування зв'язаних рівнянь. При подачі сигналу скидання (*rst*) регістри ініціалізуються ненульовими початковими умовами (x_0, y_0, z_0).

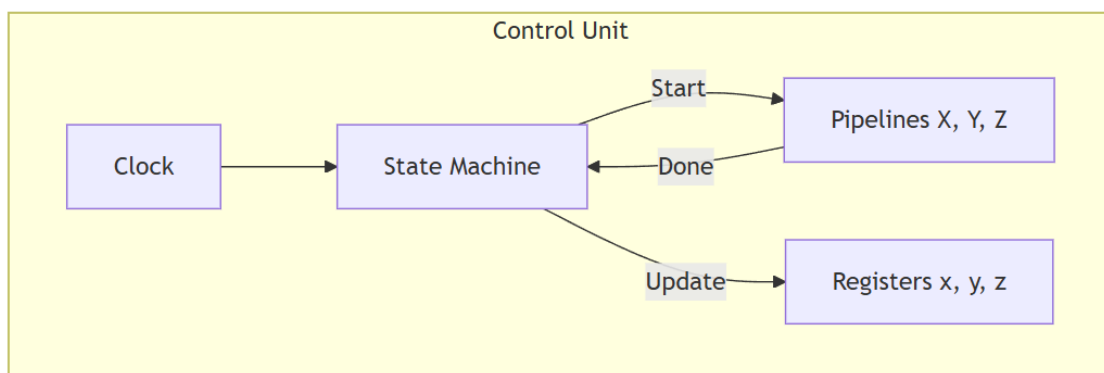


Рисунок 4.1 - Схема керування потоком даних у FPGA-генераторі хаотичних сигналів

Для кожної змінної стану виділено окремий арифметичний конвеєр (рис. 4.2), реалізований як блок *always* у Verilog. Конвеєр виконує послідовність операцій відповідно до дискретизованого рівняння: множення поточних значень змінних на коефіцієнти системи (з отриманням 64-бітного результату), нормалізацію зсувом на 16 біт, додавання або віднімання проміжних результатів, множення на крок dt та фінальне оновлення. Проміжні результати множення зберігаються у 64-бітних регістрах для запобігання переповненню перед зсувом. Вбудовані множники 9×9

(DSP-блоки) мікросхеми Cyclone IV E використовуються синтезатором автоматично для операцій множення. Після завершення всіх обчислень конвеєр виставляє сигнал `done_flag`.

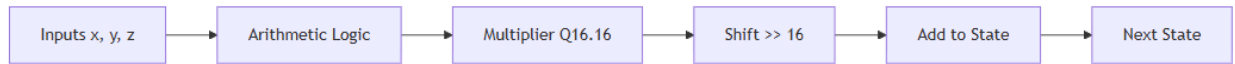


Рисунок 4.2 - Деталізація арифметичного конвеєра для однієї змінної

Наведений нижче фрагмент коду ілюструє процес керування станами, спільний для всіх п'яти реалізованих систем. Різниця між модулями генераторів полягає лише у вмісті арифметичних конвеєрів, тобто у конкретних операціях, що відповідають правим частинам відповідної системи рівнянь (рис. 4.3).

```

always @(posedge clk or posedge rst) begin
  if (rst) begin
    x <= x_0; y <= y_0; z <= z_0;
    {x_start_flag1, y_start_flag1, z_start_flag1} <= 3'b000;
    {x_start_flag2, y_start_flag2, z_start_flag2} <= 3'b000;
  end else if (x_done_flag && y_done_flag && z_done_flag) begin
    x <= x_next_calc;
    y <= y_next_calc;
    z <= z_next_calc;
    if ({x_start_flag1, y_start_flag1, z_start_flag1} == 3'b000) begin
      {x_start_flag1, y_start_flag1, z_start_flag1} <= 3'b111;
      {x_start_flag2, y_start_flag2, z_start_flag2} <= 3'b000;
    end else begin
      {x_start_flag1, y_start_flag1, z_start_flag1} <= 3'b000;
      {x_start_flag2, y_start_flag2, z_start_flag2} <= 3'b111;
    end
  end
end
end

```

Рисунок 4.3 – Фрагмент програмного коду

Подвійна буферизація міток (*start_flag1*, *start_flag2*) забезпечує коректне чергування запусків конвеєрів: після оновлення стану інвертується активна група міток, що ініціює наступний цикл обчислень.

Для перевірки коректності апаратних реалізацій розроблено схему верифікації, яка включає декілька етапів. На першому етапі виконується

симуляція HDL-модуля у середовищі ModelSim з тактовою частотою 50 МГц, а результати (значення x , y , z на кожному такті) записуються у текстовий файл у шістнадцятковому форматі Q16.16. На другому етапі ці дані імпортуються у середовище Python, де перетворюються з hex у float та порівнюються з еталонною програмною моделлю, реалізованою методом Ейлера з арифметикою *float64* при ідентичних параметрах та початкових умовах. Як метрики використовуються середньоквадратична похибка (RMSE), максимальна абсолютна похибка та відносна максимальна похибка. Результати верифікації для всіх п'яти систем наведено у підрозділі 4.5.

4.2 Математичне моделювання хаотичної динаміки

Математичні моделі п'яти обраних хаотичних систем описано у підрозділі 1.4 у вигляді систем звичайних диференціальних рівнянь із визначеними параметрами. Однак аналітичний опис не гарантує, що при конкретних значеннях параметрів та обраному кроці інтегрування система демонструє стійкий хаотичний режим, а не періодичні або розбіжні розв'язки. Тому перед FPGA-реалізацією виконано чисельне моделювання динаміки всіх п'яти систем у середовищі Python з використанням методу Ейлера (підрозділ 4.1) та арифметики подвійної точності (*float64*). Отримані результати використовуються як еталонна база для верифікації апаратних реалізацій у підрозділі 4.5.

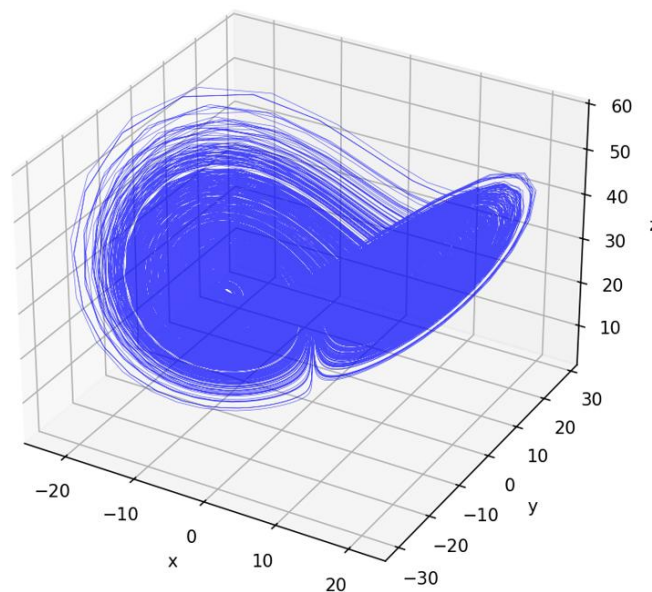
Параметри кожної системи підібрано для забезпечення стійких хаотичних режимів при кроці інтегрування $dt = 0.02$ (табл. 4.1). Рівняння чотирьох поліноміальних систем (Лоренца, Росслера, Нозе-Гувера, Рікітакі) наведено у підрозділі 1.4 і тут не повторюються. Рівняння модифікованої системи Аніщенка-Астахова описано нижче у цьому підрозділі, оскільки воно потребує введення допоміжних нелінійних функцій.

Таблиця 4.1 - Параметри хаотичних систем та початкові умови

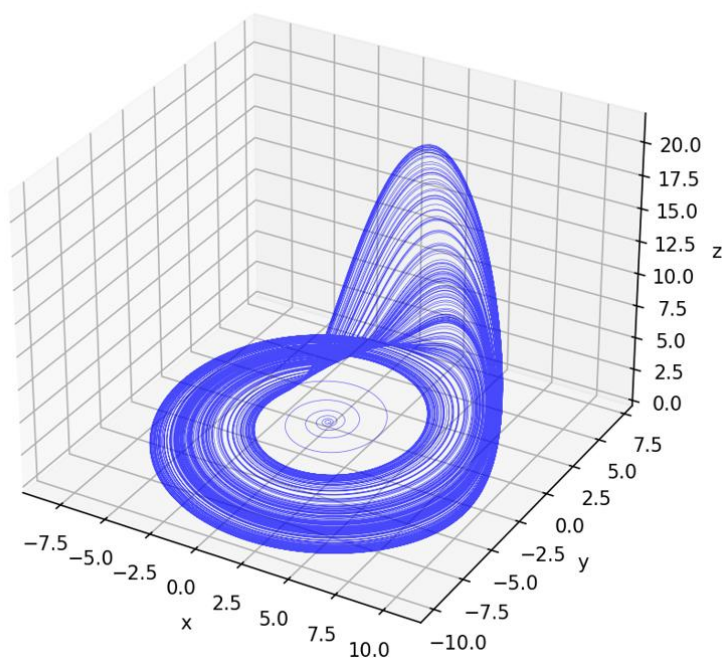
Система	Параметри	Початкові умови (x_0, y_0, z_0)
Лоренца	$\sigma = 10, \rho = 28, \beta = 8/3$	(1.0, 1.0, 1.0)
Росслера	$a = 0.19, b = 0.19, c = 5.0$	(0.0, 0.0, 0.0)
Нозе-Гувера	без параметрів	(0.1, 0.2, 0.0)
Рікітакі	$\mu = 0.8, a = 0.3$	(0.1, 0.0, 0.0)
Модифікована А-А	$m = 0.9, d = 0.01, g = 0.5$	(1.0, 1.0, 1.0)

Для кожної з чотирьох поліноміальних систем реалізовано Python-функцію, що обчислює праві частини відповідної системи рівнянь та виконує інтегрування методом Ейлера. Структура кода однакова для всіх систем: у циклі по кроках обчислюються похідні dx, dy, dz у поточній точці, після чого стан оновлюється за формулою $x[i + 1] = x[i] + dt \cdot dx$. Тривалість моделювання становить від 400 до 2000 одиниць часу залежно від системи, що забезпечує достатнє заповнення атрактора після перехідного процесу. На рисунках 4.4-4.7 наведено тривимірні фазові портрети, побудовані за результатами моделювання.

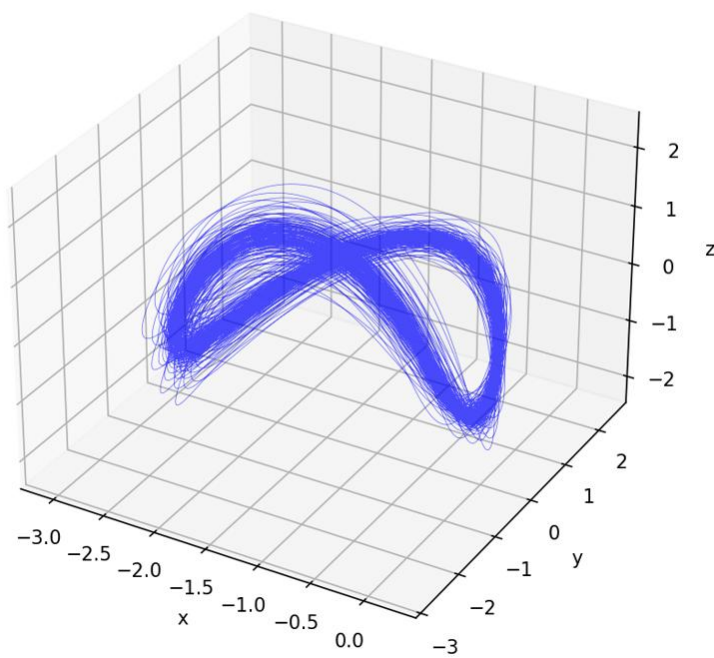
Python (еталон)

Рисунок 4.4 - Фазовий портрет системи Лоренца (Python, метод Ейлера, $dt = 0.02$)

Python (эталон)

Рисунок 4.5 - Фазовый портрет системы Росслера (Python, метод Ейлера, $dt = 0.02$)

Python (эталон)

Рисунок 4.6 - Фазовый портрет системы Нозе-Гувера (Python, метод Ейлера, $dt = 0.02$)

Python (еталон)

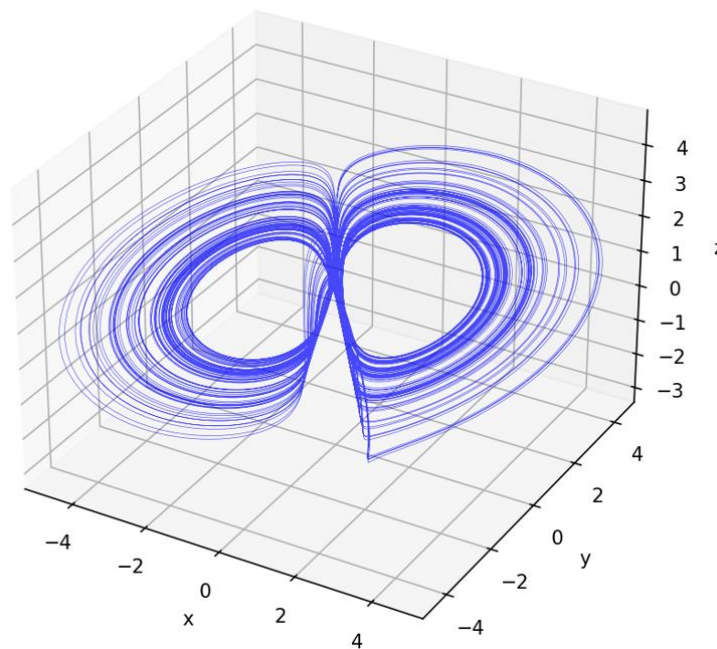


Рисунок 4.7 - Фазовий портрет системи Рікітакі (Python, метод Ейлера, $dt = 0.02$)

Система Лоренца формує характерний дволопатевий аттрактор, між «крилами» якого траєкторія хаотично переключається (рис. 4.4). Атрактор Росслера має спіральну структуру з імпульсоподібними викидами вздовж координати z (рис. 4.5). Система Нозе-Гувера демонструє тороїдальну структуру аттрактора з хаотичним заповненням (рис. 4.6), а система Рікітакі формує двопетлевий аттрактор з нерегулярними переходами між петлями (рис. 4.7). Усі чотири системи демонструють обмежені аперіодичні коливання, що підтверджує наявність стійких хаотичних режимів при обраних параметрах.

Модифіковану математичну модель автоколивального генератора з інерційною нелінійністю на основі транзисторної структури з від'ємним опором запропоновано у [97, 98] та детально досліджено у [99]. На відміну від оригінальної системи Аніщенко-Астахова [80], яка містить просту квадратичну нелінійність, модифікована модель використовує функцію $G(x_1)$, що описує вольт-амперну характеристику транзисторної структури і включає трансцендентну

нелінійність типу $\tanh$. Система задається трьома звичайними диференціальними рівняннями:

$$\dot{x}_1 = mx_1 + x_2 - x_1x_3 - dG(x_1) \quad (4.2)$$

$$\dot{x}_2 = -x_1 \quad (4.3)$$

$$\dot{x}_3 = -gx_3 + gI(x_1)F(x_1) \quad (4.4)$$

де $x_1(t)$ відповідає нормованій напрузі в коливальному контурі генератора, $x_2(t)$ є сполученою змінною, пов'язаною зі швидкістю, а $x_3(t)$ описує стан інерційного перетворювача. Параметр $m = 0.9$ задає ефективне підсилення активного елемента, $d = 0.01$ визначає вагу статичної нелінійності $G(x_1)$, а $g = 0.5$ характеризує швидкість релаксації інерційної змінної.

У рівняннях (4.2-4.4) використовуються три допоміжні нелінійні функції (рис. 4.8). Функція $G(x)$ описує робочу вітку сімейства статичних ВАХ біполярної транзисторної структури з від'ємним опором і задається виразом

$$G(x) = 2.029 \cdot 10^{-3}x + 1.114(1 - 1.002)\tanh\left(\frac{25x}{1-1.002x}\right) \quad (4.5)$$

що поєднує лінійну частину з нелінійністю типу $\tanh$. Графік $G(x)$ має виражений максимум у діапазоні малих напруг, після чого прямує до нуля (рис. 4.8, верхній лівий), що відповідає характерній формі ВАХ приладу з від'ємним опором. Функція $F(x)$ моделює властивості інерційного перетворювача та задається кубічним поліномом

$$F(x) = 2.88 \cdot 10^{-6}x + 2.074 \cdot 10^{-3}x^2 + 0.995x^3 \quad (4.6)$$

графік якого є монотонно зростаючою кривою (рис. 4.8, верхній правий). Функція $I(x)$ реалізує діодну умову пропускання струму:

$$I(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases} \quad (4.7)$$

тобто ввімкнення інерційного каналу лише за прямої полярності (рис. 4.8, нижній лівий). На основі $I(x)$ визначається функція $\Phi(x) = I(x) \cdot x^2$, яка збігається з x^2 у правій півплощині та дорівнює нулю для від'ємних значень x (рис. 4.8, нижній правий).

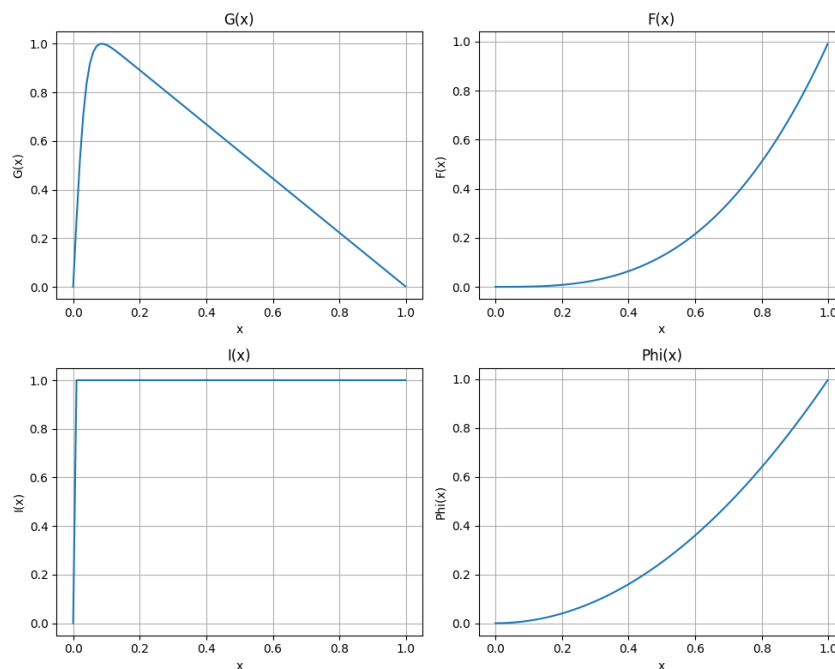


Рисунок 4.8 - Допоміжні нелінійні функції $G(x)$, $F(x)$, $I(x)$ та $\Phi(x)$ модифікованої системи Аніщенка-Астахова

Чисельне моделювання модифікованої системи А-А реалізовано функцією *solve_aa*, яка інтегрує рівняння (4.2-4.4) методом Ейлера з використанням допоміжних функцій G , F , I (рис. 4.9).

```

def G(x):
    return 2.029e-3 * x + 1.114 * (1 - 1.002 * x) * np.tanh(25 * x / (1 - 1.002 * x))

def F(x):
    return 2.88e-6 * x + 2.074e-3 * x**2 + 0.995 * x**3

def I(x):
    return 1.0 if x > 0 else 0.0

def solve_aa(m=0.9, d=0.01, g=0.5, x10=1.0, x20=1.0, x30=1.0, dt=0.02, N=25000):
    x1 = np.zeros(N); x2 = np.zeros(N); x3 = np.zeros(N)
    x1[0], x2[0], x3[0] = x10, x20, x30
    for i in range(N - 1):
        dx1 = m * x1[i] + x2[i] - x1[i] * x3[i] - d * G(x1[i])
        dx2 = -x1[i]
        dx3 = -g * x3[i] + g * I(x1[i]) * F(x1[i])
        x1[i+1] = x1[i] + dt * dx1
        x2[i+1] = x2[i] + dt * dx2
        x3[i+1] = x3[i] + dt * dx3
    return x1, x2, x3

```

Рисунок 4.9 – Фрагмент програмного коду

Структура коду ідентична моделям поліноміальних систем, однак включає виклики допоміжних функцій $G(x_1)$, $F(x_1)$, $I(x_1)$, що відповідають нелінійним елементам генератора з інерційною нелінійністю. Саме наявність функції $\tanh$ у $G(x)$ створює основну складність при подальшій FPGA-реалізації (підрозділ 4.4). Лістинги Python-моделей для чотирьох поліноміальних систем не наводяться, оскільки їхня структура аналогічна.

На рисунку 4.10 наведено тривимірний фазовий портрет модифікованої системи А-А, побудований за результатами моделювання при параметрах $m = 0.9$, $d = 0.01$, $g = 0.5$. Атрактор має характерну спірально-фрактальну форму: траєкторія багаторазово намотується в околі нижньої області фазового простору, після чого здійснює «викиди» в область великих значень x_3 та повертається до нижньої частини атрактора. Коливання залишаються обмеженими та аперіодичними, що підтверджує стійкий хаотичний режим генератора при обраних параметрах.

Python (еталон)

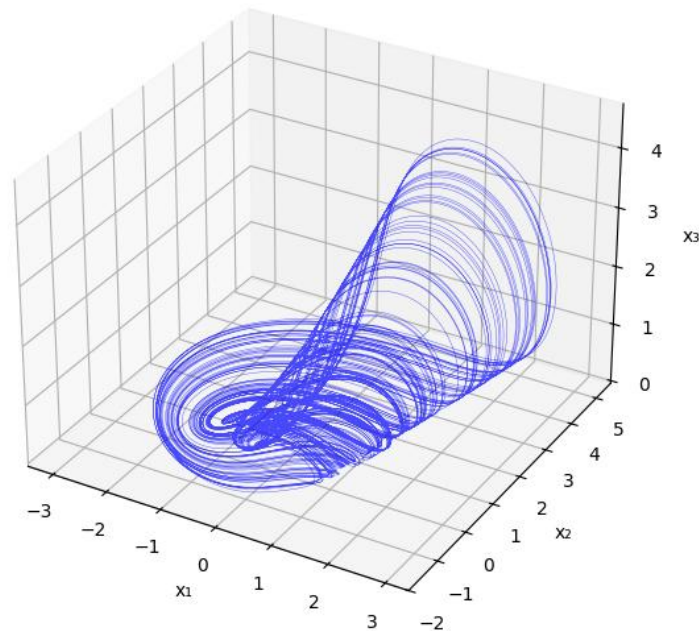


Рисунок 4.10 - Фазовий портрет модифікованої системи Аніщенко-Астахова
(Python, метод Ейлера, $dt = 0.02$, $m = 0.9$, $d = 0.01$, $g = 0.5$)

У підсумку виконано повний цикл чисельного моделювання п'яти хаотичних систем методом Ейлера у середовищі Python. Побудовані фазові портрети підтверджують наявність стійких хаотичних режимів при параметрах, зазначених у табл. 4.1. Отримані програмні моделі та числові результати використовуються як еталон для верифікації апаратних реалізацій у підрозділі 4.5. Параметри моделювання (коефіцієнти систем, початкові умови, крок інтегрування) ідентичні параметрам апаратних модулів, описаних у підрозділах 4.3 та 4.4.

4.3 FPGA-реалізація поліноміальних хаотичних систем

Системи Лоренца, Росслера, Нозе-Гувера та Рікітакі містять лише поліноміальні нелінійності: добутки змінних стану (xu , uz , xz), квадрати (y^2) та лінійні члени. Такі операції безпосередньо реалізуються стандартними 32-бітними множниками у форматі Q16.16 (підрозділ 4.1) і не потребують додаткових

апроксимаційних блоків. Усі чотири модулі побудовано на основі уніфікованої конвеєрної архітектури з підрозділу 4.1: машина станів (*start_flag/done_flag*), три паралельні арифметичні конвеєри, синхронне оновлення регістрів. Різниця між модулями полягає лише у вмісті арифметичних конвеєрів, тобто у послідовності операцій, що відповідають правим частинам конкретної системи рівнянь.

Систему Лоренца обрано як базову модель для детального розгляду підходу до FPGA-реалізації. Модуль *lorenz_chaos_wave.v* реалізує дискретизовані рівняння:

$$x[k + 1] = x[k] + \sigma \cdot (y[k] - x[k]) \cdot dt \quad (4.8)$$

$$y[k + 1] = y[k] + (x[k] \cdot (\rho - z[k]) - y[k]) \cdot dt \quad (4.9)$$

$$z[k + 1] = z[k] + (x[k] \cdot y[k] - \beta \cdot z[k]) \cdot dt \quad (4.10)$$

з параметрами у форматі Q16.16 (табл. 4.1): $\sigma = 10$ (0x000A0000), $\rho = 28$ (0x001C0000), $\beta = 8/3$ (0x0002AAAB), $dt = 0.02$ (0x0000051E). Конвеєр для змінної x обчислює вираз $\sigma \cdot (y - x) \cdot dt$ за два послідовних множення з проміжною нормалізацією (рис. 4.11).

```
reg signed [63:0] x_reg0, x_reg1;
always @(posedge x_start_flag1 or posedge x_start_flag2) begin
    x_done_flag = 1'b0;
    x_reg0 = $signed(sigma) * $signed(y - x);
    x_reg1 = $signed(x_reg0 >>> 16) * $signed(dt);
    x_next_calc = x + (x_reg1 >>> 16);
    x_done_flag = 1'd1;
end
```

Рисунок 4.11 – Фрагмент програмного коду

Спочатку обчислюється різниця $(y - x)$ у 32-бітному форматі, результат множиться на σ з отриманням 64-бітного добутку (Q32.32), який нормалізується зсувом на 16 біт. Далі нормалізований результат множиться на dt , знову нормалізується та додається до поточного значення x .

Конвеєр для змінної y реалізує вираз $(x \cdot (\rho - z) - y) \cdot dt$. Обчислення починається з множення x на різницю $(\rho - z)$, після нормалізації від результату віднімається y , а отримана різниця множиться на dt (рис. 4.12).

```
reg signed [63:0] y_reg0;
reg signed [31:0] y_reg1;
always @(posedge y_start_flag1 or posedge y_start_flag2) begin
    y_done_flag = 1'b0;
    y_reg0 = $signed(x) * $signed(r - z);
    y_reg1 = (y_reg0 >>> 16) - y;
    y_reg0 = $signed(y_reg1) * $signed(dt);
    y_next_calc = y + (y_reg0 >>> 16);
    y_done_flag = 1'd1;
end
```

Рисунок 4.12 – Фрагмент програмного коду

Конвеєр для змінної z обчислює вираз $(x \cdot y - \beta \cdot z) \cdot dt$. На відміну від попередніх конвеєрів, тут два множення ($x \times y$ та $\beta \times z$) не мають залежності за даними і можуть бути синтезовані паралельно (рис. 4.13).

```
reg signed [63:0] z_reg0, z_reg1;
reg signed [31:0] z_reg2;
always @(posedge z_start_flag1 or posedge z_start_flag2) begin
    z_done_flag = 1'b0;
    z_reg0 = $signed(x) * $signed(y);
    z_reg1 = $signed(b) * $signed(z);
    z_reg2 = (z_reg0 >>> 16) - (z_reg1 >>> 16);
    z_reg0 = $signed(z_reg2) * $signed(dt);
    z_next_calc = z + (z_reg0 >>> 16);
    z_done_flag = 1'd1;
end
```

Рисунок 4.13 – Фрагмент програмного коду

Після нормалізації обох добутків обчислюється їхня різниця, яка множиться на dt та додається до поточного z . Таким чином, кожен конвеєр системи Лоренца виконує 2–3 множення, 2–3 зсуви та 1–2 операції додавання/віднімання.

Система Росслера (модуль *rossler_chaos_wave.v*) реалізує дискретизовані рівняння:

$$x[k + 1] = x[k] + (-y[k] - z[k]) \cdot dt \quad (4.11)$$

$$y[k + 1] = y[k] + (x[k] + a \cdot y[k]) \cdot dt \quad (4.12)$$

$$z[k + 1] = z[k] + (b + z[k] \cdot (x[k] - c)) \cdot dt \quad (4.13)$$

з параметрами (табл. 4.1): $a = 0.19, b = 0.19, c = 5.0$. Система є дещо простішою за Лоренца за рахунок меншої кількості нелінійних членів. Конвеєри для змінних x та y містять лише лінійні операції та одне множення на параметр. Основна нелінійність зосереджена у конвеєрі для змінної z (рис. 4.14).

```
z_reg0 = $signed(z) * $signed(x - c);
z_reg2 = $signed(b) + (z_reg0 >>> 16);
z_reg0 = $signed(z_reg2) * $signed(dt);
z_next_calc = z + (z_reg0 >>> 16);
```

Рисунок 4.14 – Фрагмент програмного коду

Добуток $z \cdot (x - c)$ реалізується одним множенням з подальшим додаванням константи b . Система Нозе-Гувера (модуль *nose_hoover_chaos_wave.v*) реалізує дискретизовані рівняння:

$$x[k + 1] = x[k] + y[k] \cdot dt \quad (4.14)$$

$$y[k + 1] = y[k] + (y[k] \cdot z[k] - x[k]) \cdot dt \quad (4.15)$$

$$z[k + 1] = z[k] + (1 - y[k]^2) \cdot dt \quad (4.16)$$

Система не містить явних параметрів у правих частинах рівнянь, тобто єдиним числовим параметром є крок інтегрування dt . Конвеєр для змінної x обчислює вираз $y \cdot dt$, що зводиться до одного множення. Конвеєр для змінної z реалізує вираз $1 - y^2$, де константа 1.0 представляється у Q16.16 як $0x00010000$ (рис. 4.15).

```

z_reg0 = $signed(y) * $signed(y);
z_reg1 = (1 <<< 16) - (z_reg0 >>> 16);
z_reg2 = z_reg1 * $signed(dt);
z_next_calc = z + (z_reg2 >>> 16);

```

Рисунок 4.15 – Фрагмент програмного коду

Обчислення y^2 потребує лише одного множення з подальшим відніманням від константи. Система Рікітакі (модуль *rikitake_chaos_wave.v*) реалізує дискретизовані рівняння:

$$x[k + 1] = x[k] + (-\mu \cdot x[k] + y[k] \cdot z[k]) \cdot dt \quad (4.17)$$

$$y[k + 1] = y[k] + (-\mu \cdot y[k] + x[k] \cdot (z[k] - a)) \cdot dt \quad (4.18)$$

$$z[k + 1] = z[k] + (1 - x[k] \cdot y[k]) \cdot dt \quad (4.19)$$

з параметрами (табл. 4.1): $\mu = 0.8, a = 0.3$. Система містить три перехресні нелінійності: $y \cdot z, x \cdot (z - a)$ та $x \cdot y$. Найскладнішим є конвеєр для змінної x , який потребує двох незалежних множень (рис. 4.16).

```

x_reg0 = $signed(mu) * $signed(x);
x_reg1 = $signed(y) * $signed(z);
x_reg2 = (-x_reg0 >>> 16) + (x_reg1 >>> 16);
x_reg0 = $signed(x_reg2) * $signed(dt);
x_next_calc = x + (x_reg0 >>> 16);

```

Рисунок 4.16 – Фрагмент програмного коду

Конвеєр для змінної y має аналогічну структуру ($y' = -\mu \cdot y + x \cdot (z - a)$), а конвеєр для z обчислює $z' = 1 - x \cdot y$, що за структурою збігається з конвеєром z системи Нозе-Гувера.

Реалізація чотирьох поліноміальних хаотичних систем на Verilog підтверджує ефективність уніфікованої конвеєрної архітектури з підрозділу 4.1. Усі модулі мають ідентичну структуру керування (машина станів, *start_flag/done_flag*) та

відрізняються лише арифметичним вмістом конвеєрів. Кожен конвеєр містить від 1 до 3 множень, що ефективно використовує вбудовані DSP-блоки мікросхеми Cyclone IV E. Верифікація та кількісне порівняння з еталонними Python-моделями (підрозділ 4.2) наведено у підрозділі 4.5.

4.4 FPGA-реалізація модифікованої системи Аніщенка-Астахова

Серед п'яти реалізованих хаотичних систем модифікована система Аніщенка-Астахова є найбільш складною для апаратної реалізації. На відміну від поліноміальних систем (підрозділ 4.3), які містять лише стандартні арифметичні операції, модифікована система А-А включає трансцендентну нелінійність $\tanh$ у функції $G(x_1)$, що описує вольт-амперну характеристику транзисторної структури з від'ємним опором. Модифіковану математичну модель автоколивального генератора з інерційною нелінійністю на основі транзисторної структури з від'ємним опором запропоновано у [97, 98] та детально досліджено у [99]. На відміну від систем Лоренца, Росслера, Нозе-Гувера та Рікітакі, для яких існують FPGA-реалізації (підрозділ 1.4), модифікована система А-А раніше не була реалізована в цифровій формі. Навіть для оригінальної системи Аніщенка-Астахова [80], яка використовує просту квадратичну нелінійність замість $\tanh$, FPGA-реалізацій у літературі не виявлено. Наявність функції $\tanh$ у $G(x_1)$ створює основну складність, оскільки ця функція не має прямого апаратного еквівалента на FPGA і потребує спеціальних методів обчислення.

Цифрова FPGA-реалізація, на відміну від аналогових схем на дискретних компонентах, забезпечує детерміновану відтворюваність хаотичних послідовностей: за однакових початкових умов генератор формує ідентичну послідовність значень на кожному запуску. Це є критичною властивістю для застосувань, де потрібна синхронізація двох генераторів, наприклад у схемах хаотичного маскування сигналу. Крім того, цифрова реалізація вільна від дрейфу параметрів, характерного для аналогових генераторів хаосу, де старіння

компонентів (операційних підсилювачів, діодів) та температурні зміни призводять до поступового зсуву робочої точки та можливого виходу з хаотичного режиму.

Рівняння модифікованої системи А-А та допоміжні функції $G(x)$, $F(x)$, $I(x)$ детально описано у підрозділі 4.2 (формули 4.2–4.7). Параметри у форматі Q16.16: $m = 0.9$ (0x0000E666), $d = 0.01$ (0x0000028F), $g = 0.5$ (0x00008000). Ці параметри задаються як Verilog-параметри модуля *anishchenko_astakhov_chaos_wave.v* і можуть бути змінені без переписування коду.

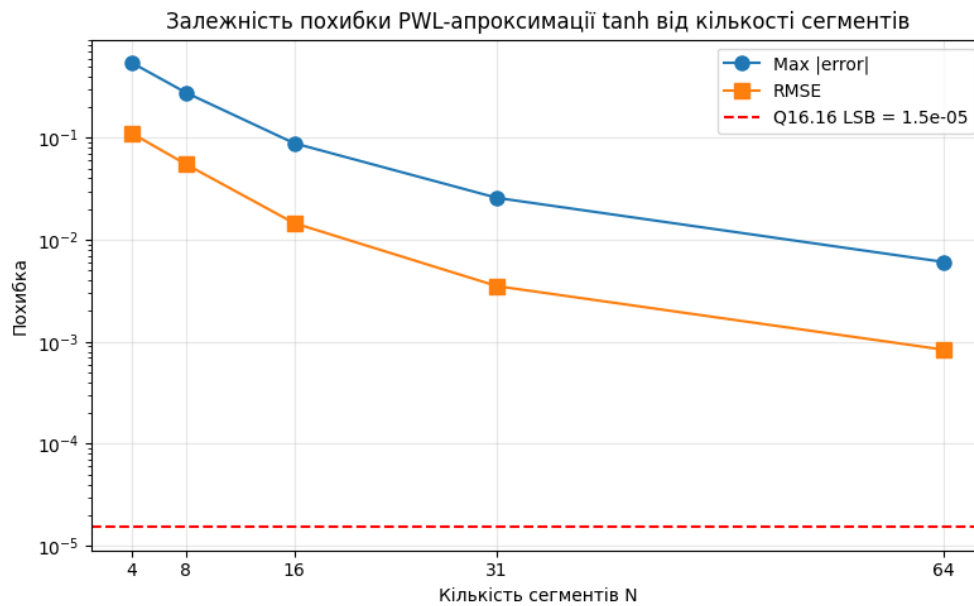
Для апаратного обчислення $\tanh(x)$ на FPGA існують три основні підходи (підрозділ 1.5): алгоритм CORDIC, табличний метод (LUT) та кусково-лінійна апроксимація (PWL). Алгоритм CORDIC забезпечує високу точність, однак потребує 10–20 ітерацій [84], що обмежує швидкість генерації. Табличний метод дозволяє отримати результат за один такт, але для досягнення високої точності потребує значного обсягу блокової пам'яті [83, 85]. Кусково-лінійна апроксимація також обчислює результат за один такт, використовуючи помірні ресурси комбінаторної логіки [86]. Для задачі високошвидкісної генерації хаотичних сигналів обрано метод PWL-апроксимації як такий, що забезпечує оптимальне співвідношення між швидкістю, точністю та апаратними витратами.

Метод PWL-апроксимації полягає у розбитті діапазону аргументу на N рівномірних сегментів з наступним обчисленням лінійної функції $y = M[i] \cdot x + B[i]$ у межах кожного сегмента. Діапазон апроксимації обрано $x \in [-10, +10]$, оскільки за його межами значення $\tanh(x)$ невідрізняється від ± 1 у форматі Q16.16. Для кожного сегмента зберігаються три 32-бітні константи: точка розриву $BREAK[i]$, нахил $M[i]$ та зсув $B[i]$. Обчислення виконується за один прохід: послідовне порівняння аргументу з точками розриву визначає активний сегмент, після чого результат обчислюється одним множенням та одним додаванням.

Для обґрунтування вибору кількості сегментів проведено порівняльний аналіз похибки PWL-апроксимації для $N = 4, 8, 16, 31$ та 64 рівномірних сегментів на діапазоні $x \in [-10, +10]$ (рис. 4.17). Результати наведено у табл. 4.2.

Таблиця 4.2 - Похибка PWL-апроксимації $\tanh$ для різних N

N	Ширина сегмента	$\text{Max } \text{error} $	RMSE
4	5.0000	5.441×10^{-1}	1.107×10^{-1}
8	2.5000	2.771×10^{-1}	5.506×10^{-2}
16	1.2500	8.847×10^{-2}	1.457×10^{-2}
31	0.6452	2.575×10^{-2}	3.514×10^{-3}
64	0.3125	6.067×10^{-3}	8.373×10^{-4}

Рисунок 4.17 - Залежність максимальної похибки та RMSE PWL-апроксимації $\tanh$ від кількості сегментів N

Максимальна похибка апроксимації зосереджена у зоні переходу функції $\tanh$, тобто при $|x| \approx 0.5 - 2.5$, де кривизна функції найбільша (рис. 4.18). У зонах насичення ($|x| > 3$) похибка практично відсутня. Обрано $N = 31$, оскільки у рівнянні системи (4.2) функція $G(x_1)$ множиться на коефіцієнт $d = 0.01$. Ефективний вплив PWL-похибки на динаміку системи становить $\text{max_error} \times d = 2.575 \times 10^{-2} \times 0.01 = 2.575 \times 10^{-4}$, що менше за похибку методу Ейлера першого порядку при $dt = 0.02$ (порядку 4×10^{-4} за крок). Збільшення кількості

сегментів до $N = 64$ подвоїло б апаратні витрати на зберігання коефіцієнтів та комбінаторну логіку пошуку без суттєвого покращення точності системи в цілому.

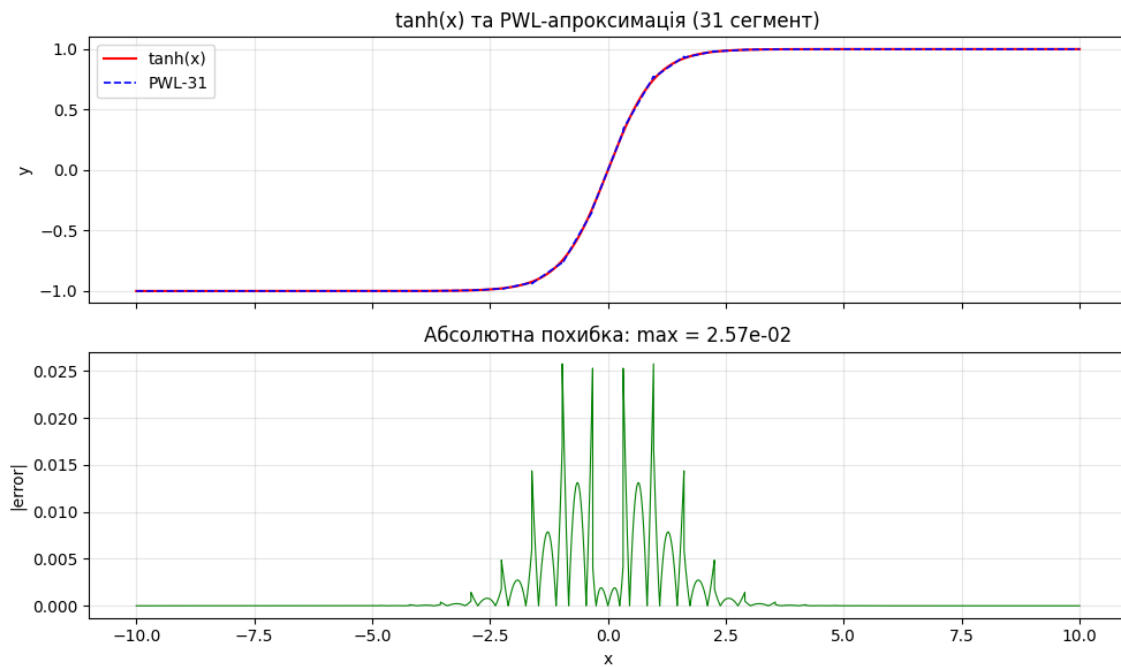


Рисунок 4.18 - Порівняння $\tanh(x)$ та PWL-апроксимації (31 сегмент) з розподілом абсолютної похибки

Функція *tanh_pwl* реалізована як Verilog-функція, що виконує лінійний пошук сегмента та обчислює результат за один прохід (рис. 4.19).

```
function [31:0] tanh_pwl;
input signed [31:0] x;
reg signed [31:0] m_sel, b_sel;
reg signed [63:0] prod;
begin
  m_sel = 32'sd0;
  b_sel = 32'h0001_0000;
  for (i=0; i<31; i=i+1) begin
    if (x < BREAK[i]) begin
      m_sel = M[i];
      b_sel = B[i];
    end
  end
  prod = x * m_sel;
  tanh_pwl = (prod >>> 16) + b_sel;
end
endfunction
```

Рисунок 4.19 – Фрагмент програмного коду

Масиви *BREAK*, *M* та *B* (по 31 елементу кожен) визначено у файлі *utils.vh* як 32-бітні константи у форматі Q16.16. За замовчуванням (якщо аргумент перевищує всі точки розриву) функція повертає значення насичення 1.0 (0x00010000). Цикл *for* розгортається синтезатором Quartus у комбінаторну логіку з 31 паралельним компаратором, що забезпечує обчислення за один комбінаторний прохід без тактової затримки.

Функція $G(x)$ реалізує повний вираз (4.5) шляхом послідовного обчислення лінійної та нелінійної складових (рис. 4.20).

```
function [31:0] G;
input signed [31:0] x;
reg signed [63:0] prod64;
reg signed [31:0] one_minus, lin_part, tanh_arg, tanh_val, nl_part;
begin
    prod64 = x * G_p4;
    one_minus = G_p3 - (prod64 >>> 16);
    prod64 = x * G_p1;
    lin_part = prod64 >>> 16;
    prod64 = x * G_p5;
    if (one_minus == 32'd0)
        tanh_arg = prod64 >>> 16;
    else
        tanh_arg = prod64 / one_minus;
    tanh_val = tanh_pwl(tanh_arg);
    prod64 = one_minus * G_p2;
    prod64 = (prod64 >>> 16) * tanh_val;
    nl_part = prod64 >>> 16;
    G = lin_part + nl_part;
end
endfunction
```

Рисунок 4.20 – Фрагмент програмного коду

Константи G_{p1} – G_{p5} відповідають коефіцієнтам формули (4.5): $G_{p1} = 2.029 \times 10^{-3}$, $G_{p2} = 1.114$, $G_{p3} = 1.0$, $G_{p4} = 1.002$, $G_{p5} = 25$. Функція містить операцію ділення ($prod64 / one_minus$) для обчислення аргументу $tanh$, що є найбільш ресурсоємною операцією і синтезується як комбінаторна схема. Обробка граничного випадку $one_minus = 0$ запобігає діленню на нуль.

Функція $F(x)$ реалізує кубічний поліном (4.6) через послідовне обчислення степенів $x^2 = (x \times x) \ggg 16$, $x^3 = (x^2 \times x) \ggg 16$ та підсумовування

зважених доданків з коефіцієнтами F_{p1}, F_{p2}, F_{p3} . Функція $I(x)$ реалізує умову (4.7) як просте порівняння: якщо $x > 0$, повертається константа 1.0 (0x00010000), інакше 0. Обидві функції не потребують складних апаратних блоків. Конвеєр для змінної x_1 є найскладнішим, оскільки обчислює чотири доданки рівняння (4.2), включаючи виклик функції $G(x_1)$ (рис. 4.21).

```
always @(posedge x1_start_flag1 or posedge x1_start_flag2) begin
    x1_done_flag = 1'b0;
    x1_reg0 = ($signed(m) * $signed(x1)) >>> 16;
    x1_reg1 = ($signed(x1) * $signed(x3)) >>> 16;
    x1_reg2 = ($signed(d) * $signed(G(x1))) >>> 16;
    x1_reg3 = x1_reg0 + x2 - x1_reg1 - x1_reg2;
    x1_next_calc = x1 + (($signed(x1_reg3) * $signed(dt)) >>> 16);
    x1_done_flag = 1'd1;
end
```

Рисунок 4.21 – Фрагмент програмного коду

Виклик $G(x_1)$ у рядку $x1_reg2$ розгортається синтезатором у повний комбінаторний ланцюг: обчислення one_minus , лінійної частини, аргументу $tanh$, PWL-пошук сегмента, обчислення $tanh_pwl$, комбінування результату. Саме цей конвеєр є основним споживачем логічних елементів та вбудованих множників.

Конвеєр для змінної x_3 обчислює вираз $-g \cdot x_3 + g \cdot I(x_1) \cdot F(x_1)$ з послідовним викликом функцій $I(x_1)$ та $F(x_1)$ (рис. 4.22).

```
always @(posedge x3_start_flag1 or posedge x3_start_flag2) begin
    x3_done_flag = 1'b0;
    x3_reg0 = ($signed(-g) * $signed(x3)) >>> 16;
    x3_reg1 = ($signed(g) * $signed(I(x1))) >>> 16;
    x3_reg2 = ($signed(x3_reg1) * $signed(F(x1))) >>> 16;
    x3_reg3 = x3_reg0 + x3_reg2;
    x3_next_calc = x3 + (($signed(x3_reg3) * $signed(dt)) >>> 16);
    x3_done_flag = 1'd1;
end
```

Рисунок 4.22 – Фрагмент програмного коду

Виклик $I(x_1)$ обнуляє добуток $g \cdot F(x_1)$ при $x_1 \leq 0$, що відповідає фізиці діодної провідності інерційного перетворювача. Конвеєр для змінної x_2 обчислює тривіальний вираз $x_2' = -x_1$ і тому не наводиться.

Модульна структура реалізації (*tanh_pwl* - *G* - конвеєр) дозволяє змінювати параметри PWL-апроксимації (масиви *BREAK*, *M*, *B* у файлі *utils.vh*) без переписування основної логіки генератора. За результатами синтезу (підрозділ 4.5) реалізація модифікованої системи А-А потребує 4990 логічних елементів (32% EP4CE15) та 90 вбудованих множників 9×9 (80%), що у 8-10 разів перевищує апаратні витрати поліноміальних систем (476-610 LEs). Основними споживачами ресурсів є PWL-логіка (31 компаратор, таблиці коефіцієнтів) та операція ділення у функції $G(x)$.

4.5 Верифікація та експериментальні результати

Для перевірки коректності розроблених FPGA-реалізацій створено програмно-апаратний комплекс верифікації. На першому етапі HDL-модуль симулюється у середовищі ModelSim з тактовою частотою 50 МГц, а значення змінних стану (x , y , z) записуються у текстовий файл у шістнадцятковому форматі Q16.16 на кожному такті. На другому етапі ці дані імпортуються у середовище Python, перетворюються у формат *float* та порівнюються з еталонною програмною моделлю (підрозділ 4.2), реалізованою методом Ейлера з арифметикою *float64* при ідентичних параметрах та початкових умовах. Як метрики використовуються середньоквадратична похибка (RMSE), максимальна абсолютна похибка та відносна максимальна похибка.

Верифікацію виконано для всіх п'яти систем на платформі EP4CE15F23C8 (Cyclone IV E) при двох кроках інтегрування: $dt = 0.02$, що відповідає апаратній реалізації, та $dt = 0.005$, що дозволяє оцінити вплив дискретизації на точність. Порівняння проводилось на двох горизонтах: короткостроковому (2000 кроків) та довгостроковому (10000 кроків).

На рисунках 4.23-4.27 наведено тривимірні фазові портрети, побудовані безпосередньо з даних FPGA-симуляції при $dt = 0.02$ для кожної з п'яти систем.

FPGA (Q16.16)

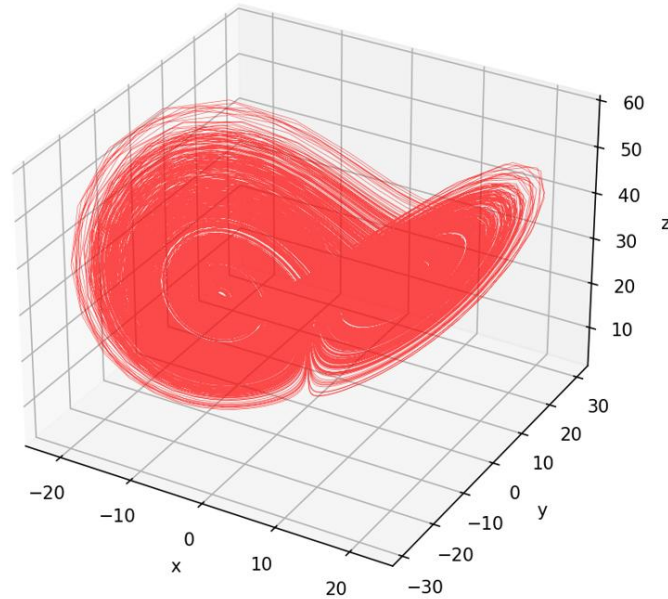


Рисунок 4.23 - Атрактор системи Лоренца, отриманий з FPGA ($dt = 0.02$)

FPGA (Q16.16)

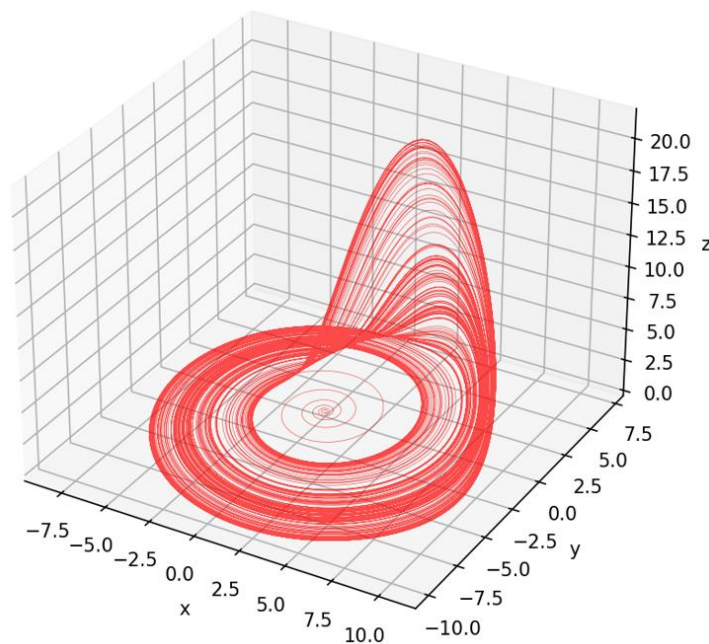
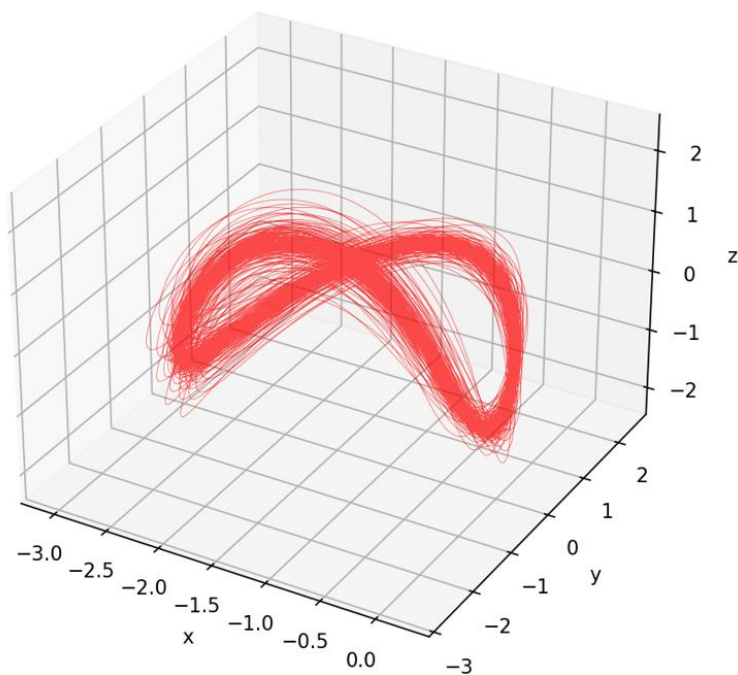
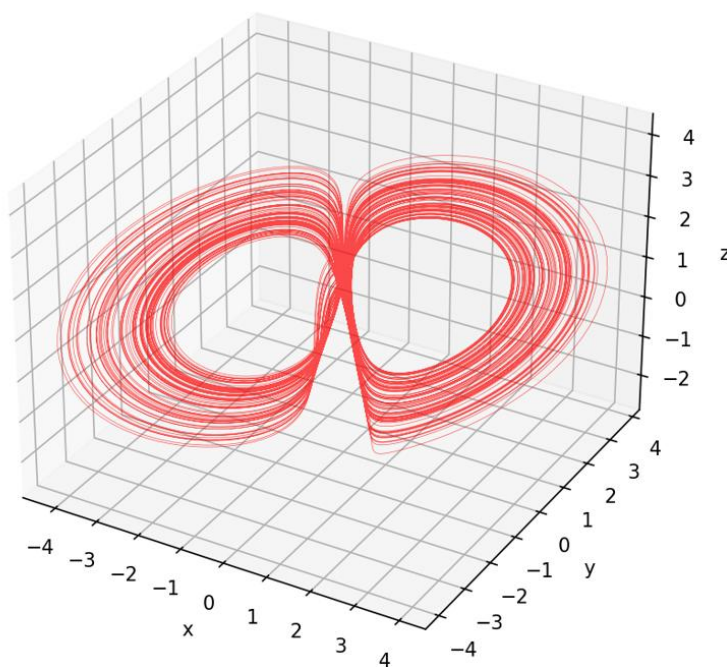


Рисунок 4.24 - Атрактор системи Росслера, отриманий з FPGA ($dt = 0.02$)

FPGA (Q16.16)

Рисунок 4.25 - Атрактор системи Нозе-Гувера, отриманий з FPGA ($dt = 0.02$)

FPGA (Q16.16)

Рисунок 4.26 - Атрактор системи Рікітакі, отриманий з FPGA ($dt = 0.02$)

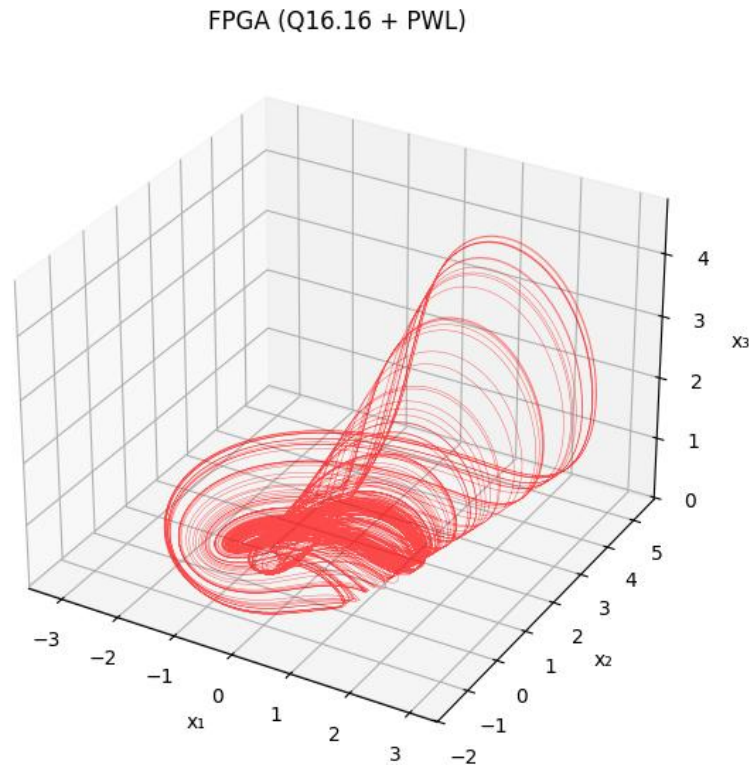


Рисунок 4.27 - Атрактор модифікованої системи Аніщенко-Астахова, отриманий з FPGA ($dt = 0.02$)

Форми атракторів з FPGA відповідають еталонним Python-моделям (рис. 4.4-4.8): система Лоренца формує дволопатевий атрактор, Росслера спіральний, Нозе-Гувера тороїдальний, Рікітакі двопетлевий. Фазовий портрет модифікованої системи А-А з FPGA (рис. 4.27) має характерну спірально-фрактальну форму, ідентичну еталону (рис. 4.8). На рисунку 4.28 наведено порівняння атракторів А-А при $dt = 0.02$ (Python та FPGA на одному рисунку), а на рисунку 4.29 аналогічне порівняння при $dt = 0.005$. В обох випадках форми атракторів візуально збігаються, що підтверджує коректність FPGA-реалізації, включаючи PWL-апроксимацію функції $\tanh$ у $G(x_1)$.

Порівняння атракторів: Python vs FPGA

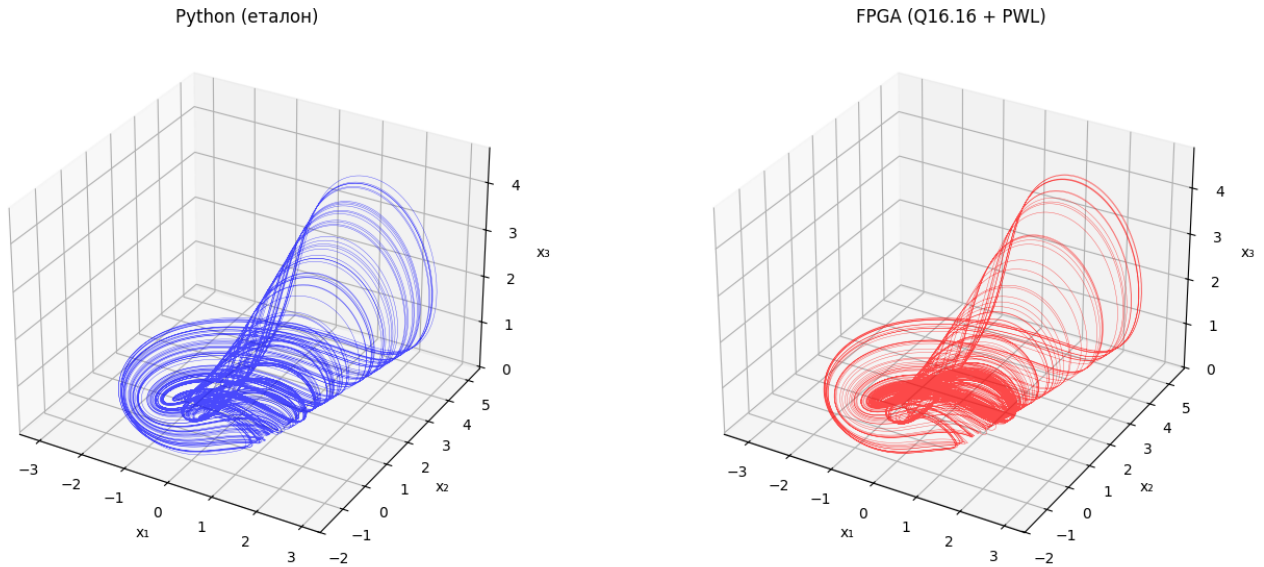


Рисунок 4.28 - Порівняння атракторів модифікованої системи А-А: Python (ліворуч) та FPGA (праворуч), $dt = 0.02$

Порівняння атракторів: Python vs FPGA

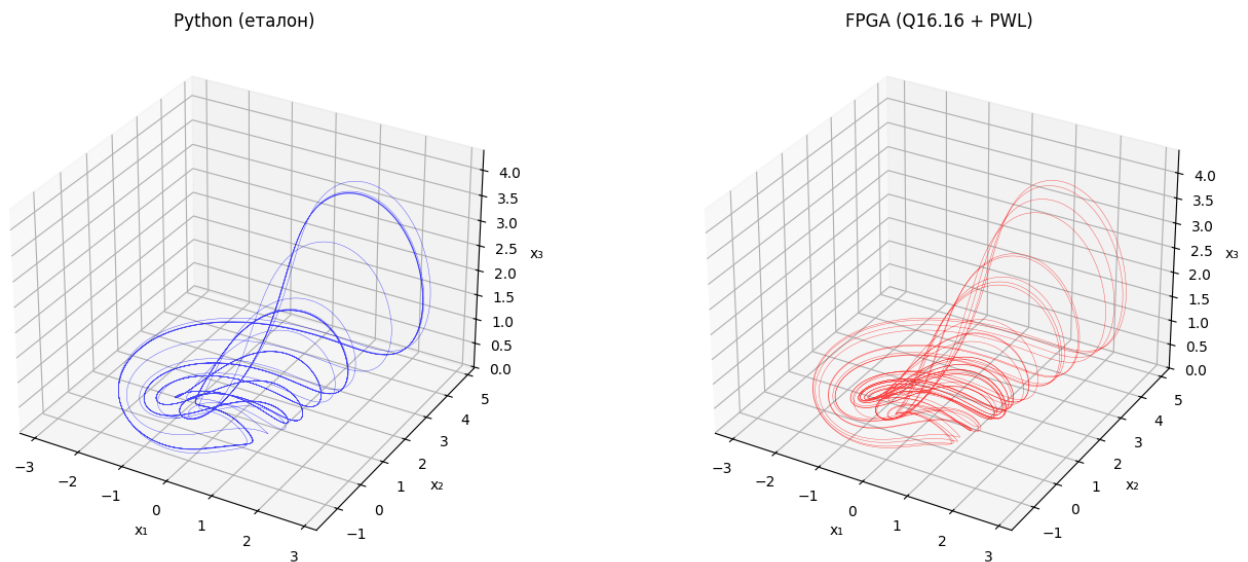


Рисунок 4.29 - Порівняння атракторів модифікованої системи А-А: Python (ліворуч) та FPGA (праворуч), $dt = 0.005$

Для натурних випробувань сигнали з FPGA виведено через 14-бітний цифро-аналоговий перетворювач (ЦАП) на цифровий осцилограф Siglent. Перетворення

формату Q16.16 у 14-бітний код виконується апаратним блоком *q16_16_to_14_bits* з масштабуванням динамічного діапазону сигналу. Тактова частота становить 50 МГц, крок інтегрування $dt = 0.02$.

На рисунку 4.30 наведено структурну схему експериментального зразка для чотирьох поліноміальних систем. Кожен генератор (*nose_hoover_chaos_wave*, *rossler_chaos_wave*, *lorenz_chaos_wave*, *rikitake_chaos_wave*) виводить три 32-бітні координати у форматі Q16.16, які перетворюються у 14-бітний код блоками *q16_16_to_14_bits*. Мультиплексори *mux_3_bus* забезпечують вибір координати (x, y або z) для кожної системи, а мультиплексор *mux_4_bus* дозволяє обрати систему за допомогою апаратної кнопки. Блок *clock_divider* формує тактовий сигнал для генераторів, а блоки *debounce* усувають брязкіт механічних контактів.

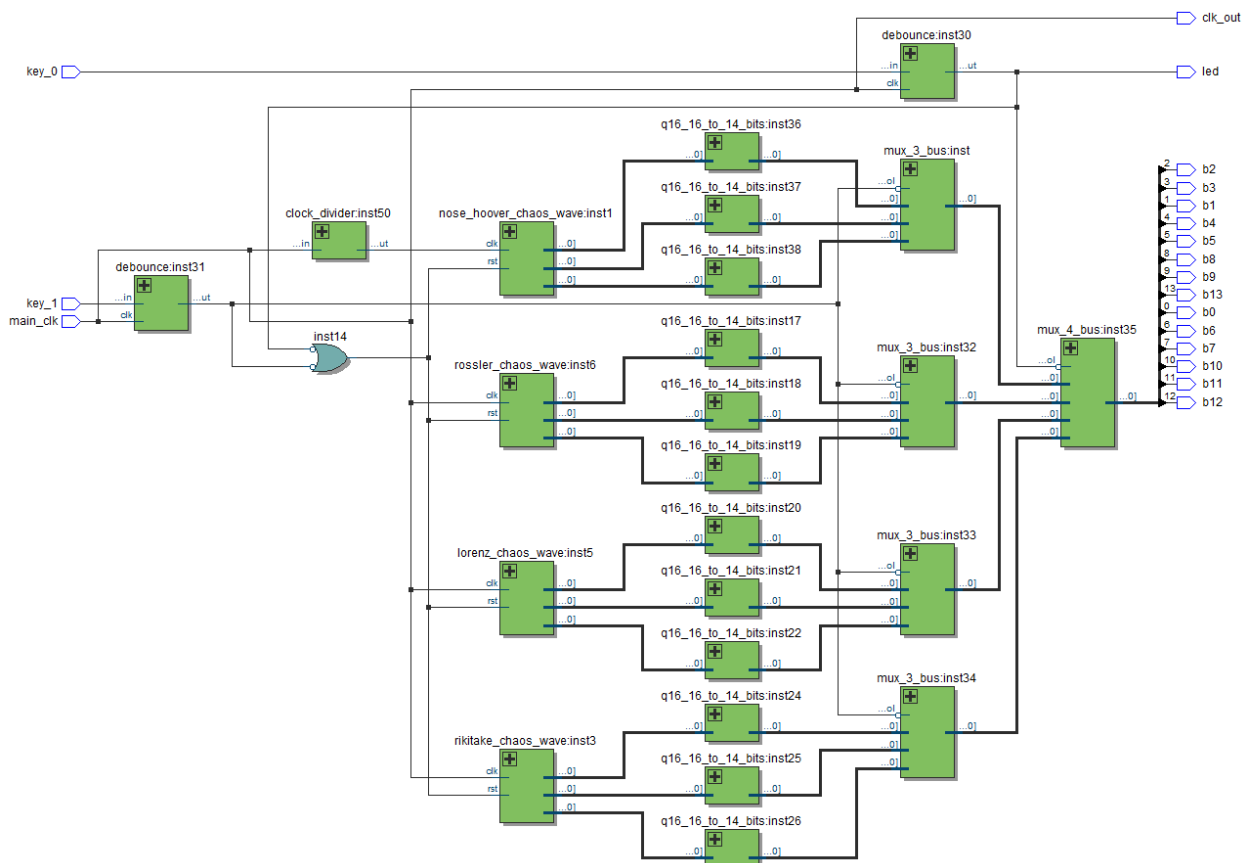


Рисунок 4.30 - Структурна схема експериментального зразка для чотирьох поліноміальних хаотичних систем з виведенням через 14-бітну шину

На рисунку 4.31 наведено аналогічну схему для модифікованої системи Аніщенко-Астахова. Схема містить один генератор *anishchenko_astakhov_chaos_wave* з трьома перетворювачами *q16_16_to_14_bits* та мультиплексором *mux_3_bus* для вибору координати. Спрощена структура порівняно з рисунком 4.30 пояснюється тим, що тестування А-А виконувалось окремо від поліноміальних систем через значне споживання ресурсів FPGA (підрозділ 4.4).

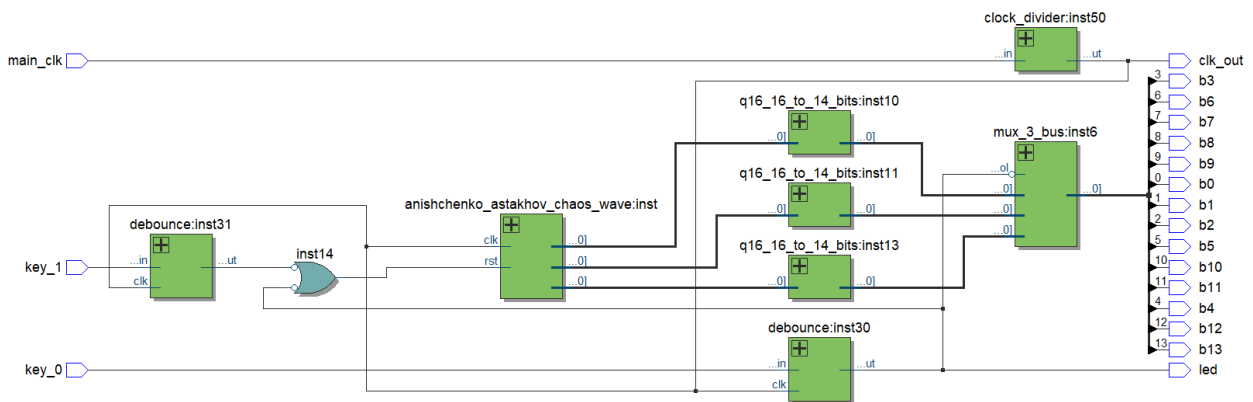


Рисунок 4.31 - Структурна схема експериментального зразка для модифікованої системи Аніщенко-Астахова з виведенням через 14-бітну шину

На рисунках 4.32-4.35 наведено осцилограми координати x для чотирьох поліноміальних систем. Осцилограма системи Лоренса (рис. 4.32) демонструє характерні аперіодичні коливання з різкими переходами, що відповідають переключенню траєкторії між «крилами» атратора. Система Росслера (рис. 4.33) формує квазіперіодичні коливання з помітною модуляцією амплітуди. Система Нозе-Гувера (рис. 4.34) генерує сигнал з регулярнішою структурою, проте з хаотичними модуляціями огинаючої. Система Рікітакі (рис. 4.35) демонструє виражену аперіодичність з нерегулярними переходами між двома режимами коливань.

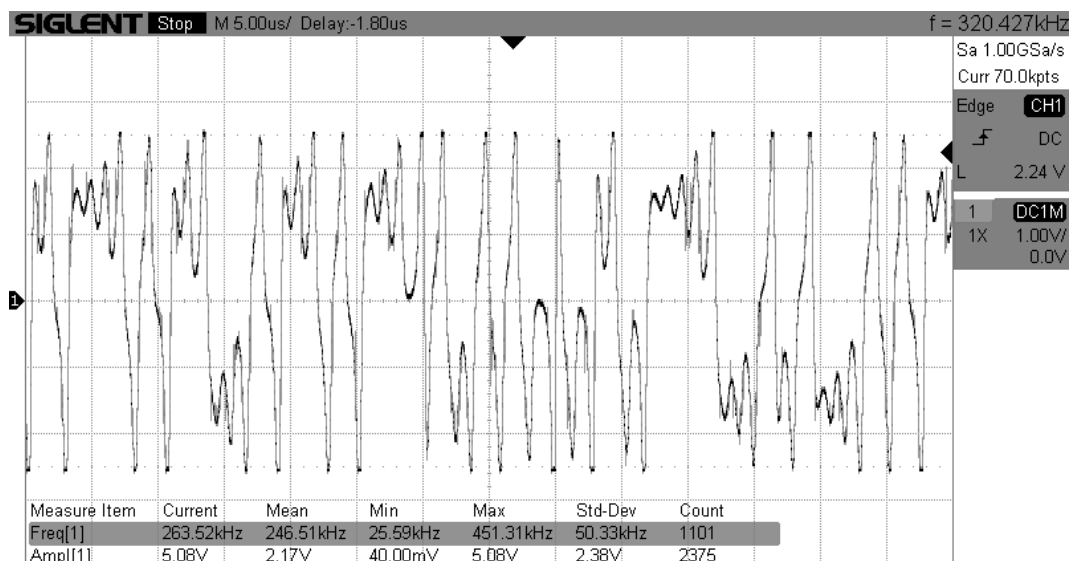


Рисунок 4.32 - Осцилограма координати x системи Лоренца з FPGA

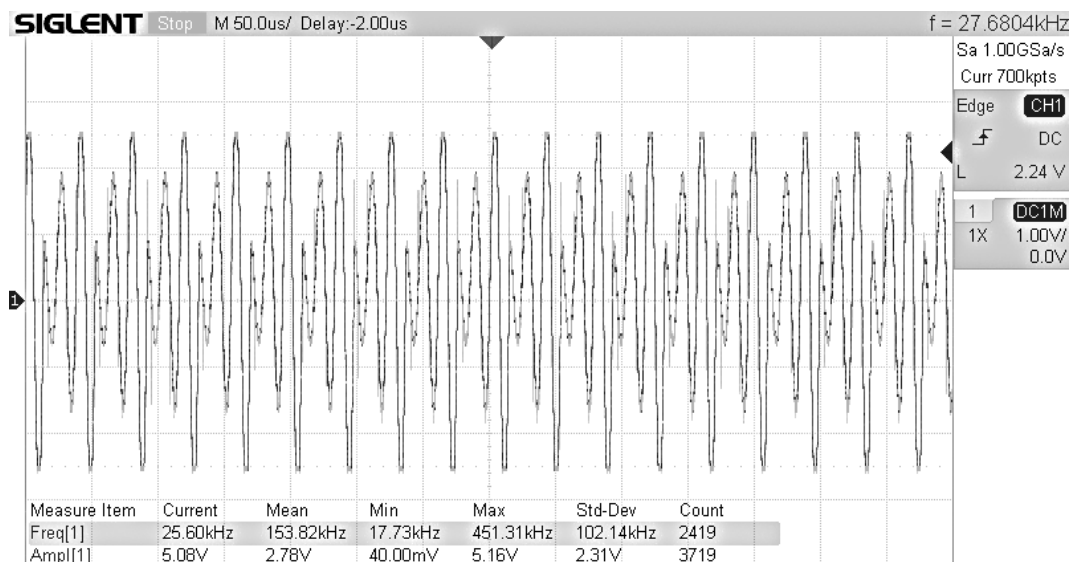


Рисунок 4.33 - Осцилограма координати x системи Росслера з FPGA

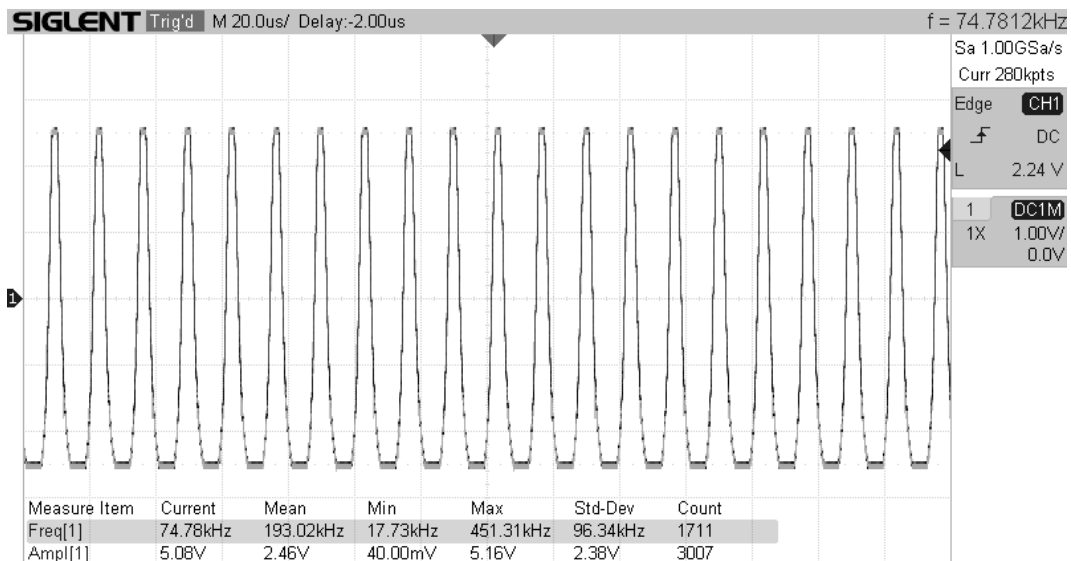


Рисунок 4.34 - Осцилограма координати x системи Нозе-Гувера з FPGA

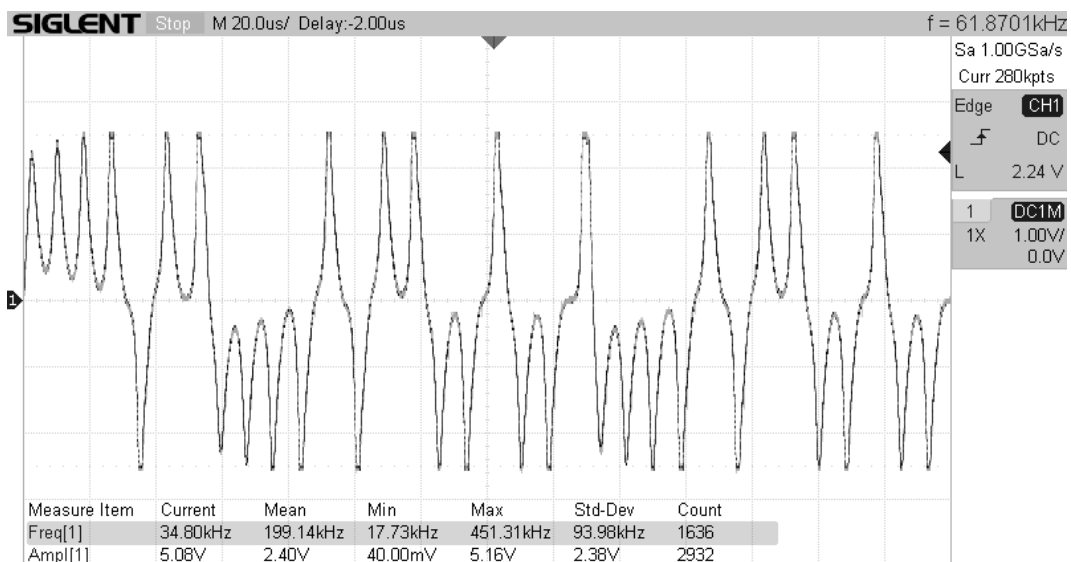


Рисунок 4.35 - Осцилограма координати x системи Рікітакі з FPGA

Для модифікованої системи А-А наведено осцилограми всіх трьох координат (рис. 4.36–4.38). Координата x_1 (рис. 4.36) демонструє характерні імпульсоподібні сплески з аперіодичними інтервалами, що відповідає нормованій напрузі в коливальному контурі генератора. Координата x_2 (рис. 4.37) має повільніший характер коливань, пов'язаний зі швидкістю генератора. Координата x_3 (рис. 4.38) відображає стан інерційного перетворювача та має виражений імпульсний

характер. Усі три координати залишаються обмеженими та демонструють стійкий хаотичний режим на реальній апаратній платформі.

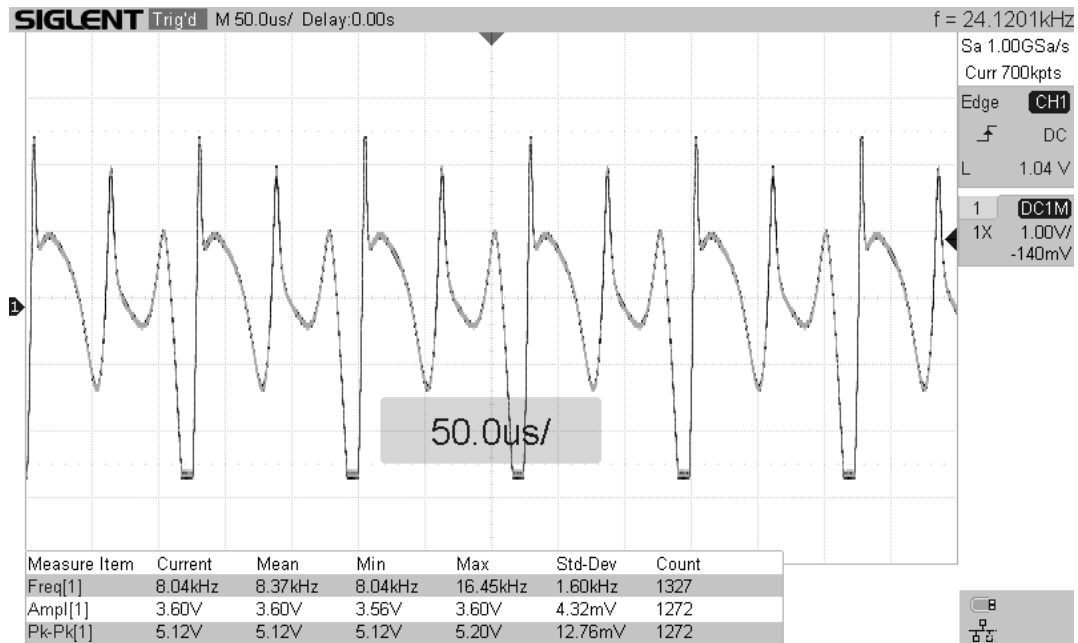


Рисунок 4.36 - Осцилограма координати x_1 модифікованої системи А-А з FPGA

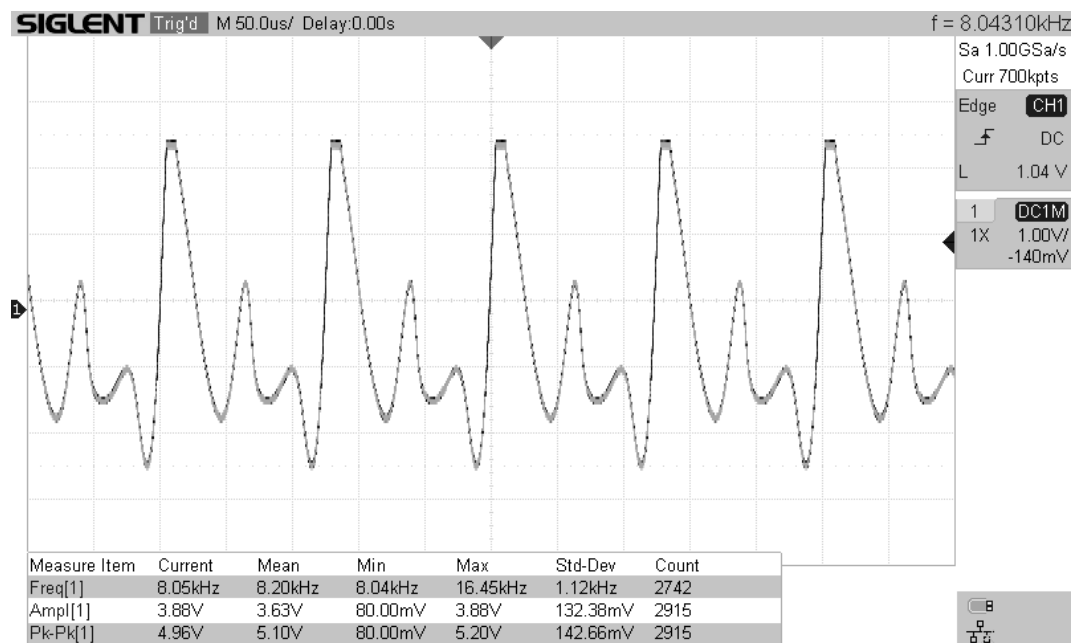


Рисунок 4.37 - Осцилограма координати x_2 модифікованої системи А-А з FPGA

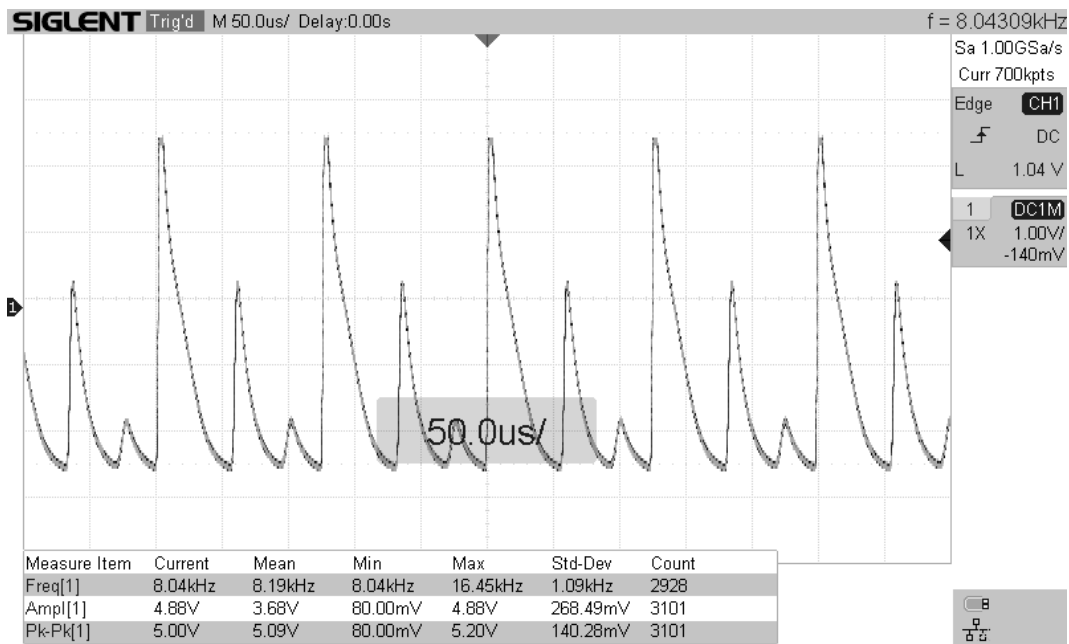


Рисунок 4.38 - Осцилограма координати x_3 модифікованої системи А-А з FPGA

Для кількісної оцінки точності FPGA-реалізацій обчислено RMSE порівняно з Python-еталоном на короткостроковому горизонті (2000 кроків) для двох значень кроку інтегрування. Результати при $dt = 0.02$ наведено у табл. 4.3.

Таблиця 4.3 - RMSE при $dt = 0.02$ (короткостроковий, 2000 кроків)

Система	RMSE x	RMSE y	RMSE z	Rel max x [%]
Росслер	1.672×10^{-2}	1.883×10^{-2}	2.475×10^{-4}	2.93
Нозе-Гувєр	4.103×10^{-2}	5.801×10^{-2}	5.861×10^{-2}	3.86
Мод. А-А	4.349×10^{-1}	4.098×10^{-1}	3.414×10^{-1}	47.94
Рікітакі	1.606	1.382	8.384×10^{-1}	135.97
Лоренц	1.216×10^1	1.386×10^1	1.149×10^1	160.00

Значення RMSE варіюються від 10^{-2} (Росслер) до 10^1 (Лоренц), тобто на три порядки. Ця різниця зумовлена різними показниками Ляпунова досліджуваних систем: чим вищий максимальний показник Ляпунова (λ_{max}), тим швидше

відбувається декореляція траєкторій FPGA та Python і, відповідно, тим більшим є RMSE на тому ж горизонті.

Результати при $dt = 0.005$ наведено у табл. 4.4.

Таблиця 4.4 - RMSE при $dt = 0.005$ (короткостроковий, 2000 кроків)

Система	RMSE x	RMSE y	RMSE z	Rel max x [%]
<i>Росслер</i>	4.118×10^{-3}	4.167×10^{-3}	3.376×10^{-4}	15.02
<i>Нозе-Гувер</i>	5.579×10^{-3}	9.261×10^{-3}	9.215×10^{-3}	0.70
<i>Мод. А-А</i>	6.095×10^{-2}	1.026×10^{-1}	7.229×10^{-2}	5.64
<i>Рікітакі</i>	8.003×10^{-2}	8.131×10^{-2}	9.911×10^{-2}	7.41
<i>Лоренц</i>	1.106	1.740	1.901	35.00

При зменшенні кроку інтегрування з $dt = 0.02$ до $dt = 0.005$ RMSE суттєво зменшується для всіх п'яти систем. Кількісне порівняння покращення наведено у табл. 4.5.

Таблиця 4.5 - Покращення RMSE при зменшенні dt (RMSE x, 2000 кроків)

Система	RMSE (dt=0.02)	RMSE (dt=0.005)	Покращення
<i>Росслер</i>	1.672×10^{-2}	4.118×10^{-3}	4.1×
<i>Нозе-Гувер</i>	4.103×10^{-2}	5.579×10^{-3}	7.4×
<i>Мод. А-А</i>	4.349×10^{-1}	6.095×10^{-2}	7.1×
<i>Рікітакі</i>	1.606	8.003×10^{-2}	20.1×
<i>Лоренц</i>	1.216×10^1	1.106	11.0×

Зменшення кроку інтегрування у 4 рази забезпечило покращення RMSE від 4.1 разу (Росслер) до 20.1 разу (Рікітакі), із середнім значенням ~ 9.9 . Це перевищує теоретичне значення $4\times$ для методу Ейлера першого порядку (похибка якого пропорційна dt). Різниця пояснюється нелінійним характером хаотичних

систем: менший крок dt не лише зменшує локальну похибку дискретизації, а й відкладає момент декореляції траєкторій, що подовжує горизонт, на якому FPGA-реалізація зберігає корельованість з еталоном. Цей результат підтверджує, що домінуючим джерелом похибки є метод Ейлера першого порядку, а не PWL-апроксимація $\tanh$ чи арифметика фіксованої коми Q16.16.

Модифікована система А-А при $dt = 0.005$ має RMSE координати $x_1 = 6.095 \times 10^{-2}$, що порівнянне з Рікітакі (8.003×10^{-2}). Обидві системи не використовують PWL (Рікітакі є поліноміальною), що підтверджує: PWL-похибка, послаблена коефіцієнтом $d = 0.01$, не є вузьким місцем точності. Покращення RMSE для А-А при зменшенні dt ($7.1\times$) також порівнянне з іншими системами.

На рисунку 4.39 наведено точкове відхилення координати x_1 між FPGA та Python для модифікованої системи А-А при $dt = 0.02$.



Рисунок 4.39 - Точкове відхилення x_1 між FPGA та Python для модифікованої системи А-А ($dt = 0.02$, 10000 кроків)

Декореляція траєкторій настає при $t \approx 25$ одиниць часу, після чого похибка набуває хаотичного характеру з амплітудою, порівнянною з амплітудою самого сигналу. При $dt = 0.005$ (рис. 4.40) момент декореляції відкладається до $t \approx 50$, а амплітуда похибки на початковому інтервалі значно менша. Така поведінка є фундаментальною властивістю хаотичних систем з додатним показником

Ляпунова: навіть мінімальна різниця між арифметикою *float64* (53 біти мантиси) та Q16.16 (16 біт дробової частини) з часом підсилюється експоненційно.

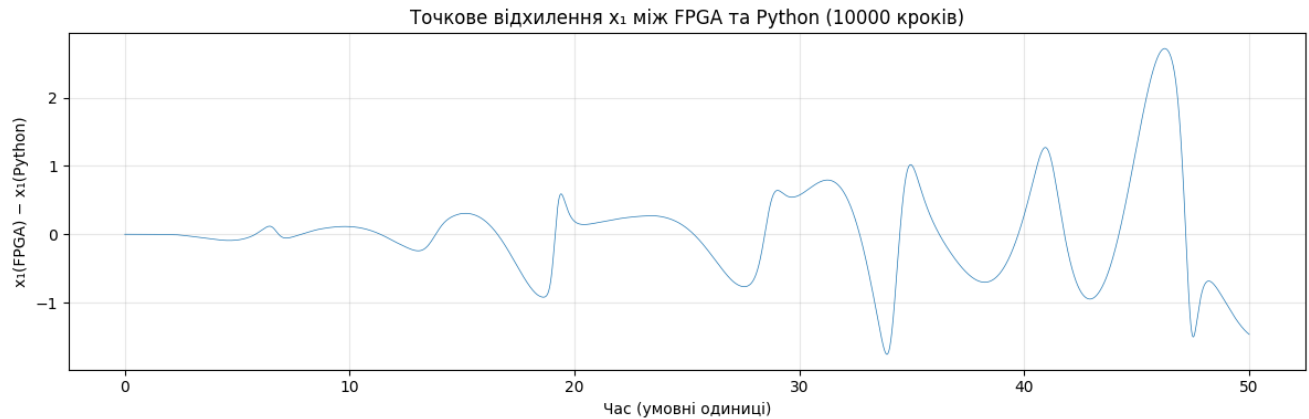


Рисунок 4.40 - Точкове відхилення x_1 між FPGA та Python для модифікованої системи A-A ($dt = 0.005$, 10000 кроків)

Попри зростання RMSE на довгому горизонті, базові статистичні характеристики сигналів залишаються практично ідентичними для обох реалізацій. У табл. 4.6 наведено порівняння для модифікованої системи A-A при $dt = 0.02$ на інтервалі 100 000 кроків.

Таблиця 4.6 - Базові статистичні характеристики модифікованої A-A ($dt = 0.02$, 100 000 кроків)

Змінна		mean	std	min	max
x_1	<i>Python</i>	0.0003	1.0628	-3.04	2.96
	<i>FPGA</i>	0.0007	1.0501	-3.14	3.00
x_2	<i>Python</i>	0.4439	1.2157	-1.52	5.11
	<i>FPGA</i>	0.4289	1.1777	-1.52	5.30
x_3	<i>Python</i>	0.8680	0.8290	0.05	4.64
	<i>FPGA</i>	0.8853	0.7856	0.06	4.76

Відхилення середнього значення не перевищує 3.4%, стандартного відхилення 5.2%, мінімуму та максимуму 4.5%. Аналогічне збігання спостерігається для всіх п'яти систем. Це означає, що FPGA-реалізація генерує той самий хаотичний атрактор з тим самим інваріантним розподілом, незважаючи на неминучу довгострокову розбіжність окремих траєкторій. Для хаотичних систем, де RMSE на довгому горизонті завжди зростає через додатний показник Ляпунова, критерієм коректності є саме збігання статистичних характеристик та форми атрактора. Обидва критерії виконуються для всіх п'яти систем при обох значеннях dt . Результати синтезу кожної системи окремо на мікросхемі EP4CE15F23C8 наведено у табл. 4.7.

Таблиця 4.7 - Ресурси FPGA для кожної хаотичної системи (EP4CE15F23C8, Cyclone IV E)

Система	LEs	Registers	Mult 9×9	LEs [%]	Mult [%]
<i>Росслер</i>	476	96	24	3%	21%
<i>Нозе-Гувер</i>	518	96	32	3%	29%
<i>Лоренц</i>	573	96	36	4%	32%
<i>Рікітакі</i>	610	96	44	4%	39%
<i>Мод. А-А</i>	4990	96	90	32%	80%

Кількість регістрів (96) однакова для всіх систем, що дорівнює $3 \text{ змінні} \times 32 \text{ біти}$ і підтверджує уніфіковану архітектуру (підрозділ 4.1). Чотири поліноміальні системи займають від 476 до 610 логічних елементів (3-4% ресурсів мікросхеми), а різниця між ними визначається кількістю множень у рівняннях. Модифікована система А-А потребує 4990 логічних елементів (32%), що у 8-10 разів перевищує поліноміальні системи. Основними споживачами ресурсів є PWL-логіка (31 компаратор та таблиці коефіцієнтів *BREAK*, *M*, *B*) та операція ділення у функції $G(x)$. Використання 90 із 112 вбудованих множників 9×9 (80%) означає, що

модифікована система А-А працює близько до межі ресурсів мікросхеми EP4CE15. На одному кристалі можна розмістити модифіковану А-А разом з однією поліноміальною системою (наприклад, Росслером: $90 + 24 = 114 > 112$, тобто за множниками не вміщається, однак за логічними елементами залишається значний запас). Ресурсний аналіз також демонструє можливість інтеграції генератора хаотичних сигналів з компонентами IoT-вузла, описаними у розділах 2 та 3, на одному кристалі EP4CE15. Система збору даних (підрозділ 2.2) використовує процесорне ядро NIOS II, апаратний частотомір та контролер I²C. Поліноміальні генератори хаосу (476-610 LEs, 24-44 множники) потребують незначних додаткових ресурсів, що дозволяє розмістити генератор поруч із IoT-вузлом для потенційного захисту каналу передачі телеметричних даних хаотичними послідовностями.

Верифікація підтвердила коректність усіх п'яти FPGA-реалізацій хаотичних генераторів. Фазові портрети з FPGA відповідають еталонним Python-моделям, осцилограми з реальної плати демонструють стійкі хаотичні коливання, а базові статистичні характеристики збігаються з точністю до 5%. Зменшення кроку інтегрування з 0.02 до 0.005 покращило RMSE у 4-20 разів для всіх систем, що підтверджує домінуючий вплив методу Ейлера як джерела похибки. Для апаратної реалізації обрано $dt = 0.02$, що забезпечує вищу ефективну частоту генерації хаотичного сигналу при прийнятній точності відтворення атратора.

4.6 Висновки до розділу 4

У розділі наведено результати математичного моделювання та FPGA-реалізації генераторів детермінованого хаосу на основі п'яти хаотичних систем третього порядку. Основні результати полягають у такому:

- 1) Розроблено уніфіковану конвеєрну архітектуру для FPGA-реалізації генераторів хаотичних сигналів на основі методу Ейлера з арифметикою фіксованої коми Q16.16. Архітектура забезпечує паралельне обчислення трьох

змінних стану з синхронним оновленням на основі міток готовності (*start_flag*, *done_flag*) та працює на тактовій частоті 50 МГц на мікросхемі EP4CE15F23C8 (Cyclone IV E).

2) Виконано математичне моделювання динаміки п'яти хаотичних систем (Лоренца, Росслера, Нозе-Гувера, Рікітакі та модифікованої системи Аніщенко-Астахова) методом Ейлера у середовищі Python. Для кожної системи підібрано параметри, що забезпечують стійкі хаотичні режими при кроках інтегрування $dt = 0.02$ та $dt = 0.005$, та побудовано еталонні фазові портрети для верифікації апаратних реалізацій.

3) Реалізовано на FPGA чотири поліноміальні хаотичні системи (Лоренца, Росслера, Нозе-Гувера, Рікітакі) мовою Verilog з використанням уніфікованої конвеєрної архітектури. Апаратні витрати становлять від 476 логічних елементів (система Росслера, 3% EP4CE15) до 610 логічних елементів (система Рікітакі, 4% EP4CE15) та від 24 до 44 вбудованих множників 9×9 .

4) Вперше реалізовано на FPGA модифіковану систему Аніщенко-Астахова, математичну модель якої запропоновано у [97, 98, 99]. На відміну від поліноміальних систем, модифікована система А-А містить трансцендентну нелінійність $\tanh$ у функції $G(x_1)$, що описує вольт-амперну характеристику транзисторної структури з від'ємним опором. Для апаратного обчислення $\tanh$ запропоновано кусково-лінійну апроксимацію з 31 сегментом (максимальна похибка апроксимації 2.575×10^{-2}), що забезпечує обчислення за один такт. З урахуванням коефіцієнта $d = 0.01$ у рівнянні системи ефективний вплив PWL-похибки на динаміку не перевищує 2.6×10^{-4} , що менше за похибку методу Ейлера. Реалізація потребує 4990 логічних елементів (32% EP4CE15) та 90 вбудованих множників 9×9 (80%), що у 8–10 разів перевищує поліноміальні системи.

5) Верифікація підтвердила коректність усіх п'яти FPGA-реалізацій: базові статистичні характеристики (середнє значення, стандартне відхилення, мінімум,

максимум) збігаються з Python-еталоном з точністю до 5%, а форми атракторів візуально ідентичні. Зменшення кроку інтегрування з $dt = 0.02$ до $dt = 0.005$ покращило RMSE від 4.1 разу (система Росслера) до 20.1 разу (система Рікітакі), що підтверджує домінуючий вплив методу Ейлера першого порядку як джерела похибки.

6) Проведено випробування всіх п'яти генераторів на платформі EP4CE15F23C8 з виведенням сигналів через 14-бітний ЦАП на цифровий осцилограф. Осцилограми підтверджують генерацію стійких аперіодичних обмежених коливань з характерною для кожної системи структурою хаотичного сигналу.

7) Основні наукові та практичні результати розділу опубліковані в роботах [2, 12].

ВИСНОВКИ

Дисертаційне дослідження присвячено розв'язанню актуального науково-прикладного завдання, яке полягає у забезпеченні одночасного багатоканального вимірювання частотних сенсорів та відтворюваного цифрового генерування хаотичних сигналів у радіотехнічних приладах на FPGA для IoT-систем збору даних.

Проведений аналіз публікацій, присвячених сучасним підходам до побудови IoT-систем збору даних на основі FPGA та FPGA-реалізацій хаотичних систем, дозволив визначити перспективні способи інтеграції програмованих логічних матриць у багаторівневі IoT-архітектури та встановити відсутність цифрових FPGA-реалізацій модифікованої системи Аніщенка-Астахова. Обґрунтовано вибір FPGA як центрального обчислювального елемента для систем збору даних від частотних сенсорів завдяки перевагам паралельної обробки, детермінованості часових інтервалів та можливості інтеграції програмного ядра на одному кристалі.

У науковій роботі досягнуто мету, яка полягає у забезпеченні одночасного багатоканального вимірювання частотних сенсорів та відтворюваного цифрового генерування хаотичних сигналів у радіотехнічних приладах на FPGA для IoT-систем збору даних за рахунок розробки нових та удосконалення існуючих методів апаратної реалізації на програмованій логіці із застосуванням кусково-лінійної апроксимації нелінійних функцій.

Основні результати досліджень дисертаційної роботи полягають у такому:

1. Здійснено систематизацію та аналіз сучасних підходів до побудови IoT-систем збору даних на основі FPGA. Визначено переваги використання програмованих логічних матриць порівняно з мікроконтролерами для задач паралельного збору даних від частотних сенсорів: одночасна обробка даних від декількох сенсорів, детерміновані затримки, можливість реконфігурації апаратної логіки. Проаналізовано стандартні інтерфейси обміну даними (I²C, SPI, UART) та бездротові протоколи (LoRa, Wi-Fi) для їх інтеграції в FPGA-рішення.

Систематизовано існуючі FPGA-реалізації хаотичних генераторів та встановлено, що модифікована система Аніщенко-Астахова раніше не була реалізована в цифровій формі.

2. Вперше розроблено багатоканальну систему збору даних від частотних сенсорів на FPGA, яка, на відміну від існуючих мікроконтролерних рішень, що використовують послідовне опитування каналів з обмеженою кількістю апаратних таймерів, забезпечує одночасне паралельне вимірювання частоти 12 незалежних каналів із синхронною часовою базою та детермінованим вікном вимірювання 100 мс. Модуль керування `block_form` синтезує синхронні сигнали `qRst`, `qCnt`, `qLd` із детермінованим таймінгом (ширина імпульсу скидання 500 мкс, вікно підрахунку 99.8 мс, імпульс завантаження 5 мс). Результати зберігаються у 32-розрядних регістрах, а похибка вимірювання визначається квантованістю підрахунку (± 1 імпульс), що забезпечує відносну похибку порядку 10^{-4} - 10^{-5} для діапазону сотень кілогерц.

3. Удосконалено структурну організацію IoT-вузла на FPGA шляхом апаратно-програмного розподілу функцій, яка, на відміну від суто апаратних або суто програмних підходів, забезпечує одночасну роботу апаратного багатоканального частотоміра та контролера I²C для цифрових сенсорів під керуванням програмного ядра NIOS II на одному кристалі з циклом опитування 10 Гц. Конфігурація SoC на мікросхемі EP4CE15F23C8 включає: NIOS II (20 КБ RAM, 50 МГц), апаратний 12-канальний частотомір, контролер I²C (400 кГц), інтерфейс UART (115200 бод, обрано відповідно до вимог трафіку, не є фізичним обмеженням) та контролер переривань. Розроблено контролер шини I²C мовою VHDL у вигляді синхронного цифрового автомата з дев'ятьма станами, що підтримує механізм `clock stretching`, та `Repeated Start` відповідно до специфікації I²C.

4. Набув подальшого розвитку метод побудови периферійних вузлів IoT-сенсорних мереж шляхом застосування FPGA як вузла збору даних з наскрізною

передачею телеметрії до хмарної платформи через LoRa, який, на відміну від типових MCU-базованих IoT-архітектур, забезпечує наскрізну доставку телеметрії від FPGA-вузла з частотними сенсорами до хмарного сервісу з можливістю масштабування кількості вузлів у топології «зірка». Реалізовано протокол фрагментації пакетів для передачі даних обсягом понад 100 байт із серіалізацією у компактному бінарному форматі CBOR. Розроблено програмну реалізацію FPGA-вузла та центрального хабу на базі модулів LilyGO LoRa32 (ESP32) у середовищі FreeRTOS із багатозадачною архітектурою та передачею даних на хмарну платформу через захищене HTTPS-з'єднання.

5. Розроблено серверну частину IoT-системи на основі REST API (Python/FastAPI) з базою даних SQLite та мобільний Android-застосунок мовою Kotlin для моніторингу сенсорних даних. Серверна архітектура забезпечує прийом телеметричних даних через HTTPS та надає ендпоінти для фільтрації вимірювань за ідентифікаторами пристроїв. Мобільний застосунок реалізує асинхронне оновлення табличного відображення вимірювань з інтервалом 500 мс із захищеним TLS-з'єднанням.

6. Розроблено уніфіковану конвеєрну архітектуру для FPGA-реалізації генераторів хаотичних сигналів на основі методу Ейлера з арифметикою фіксованої коми Q16.16. Архітектура забезпечує паралельне обчислення трьох змінних стану з синхронним оновленням на основі міток готовності (start_flag, done_flag) та працює на тактовій частоті 50 МГц. Реалізовано на FPGA чотири поліноміальні хаотичні системи (Лоренца, Росслера, Нозе-Гувера, Рікітакі) з апаратними витратами від 476 логічних елементів (3% EP4CE15) до 610 логічних елементів (4% EP4CE15) та від 24 до 44 вбудованих множників 9×9 . Виконано математичне моделювання динаміки усіх п'яти систем методом Ейлера у середовищі Python та побудовано еталонні фазові портрети для верифікації.

7. Вперше реалізовано на FPGA модифіковану систему Аніщенка-Астахова, математичну модель якої запропоновано у [97, 98, 99]. На відміну від існуючих

досліджень цієї системи, що обмежуються теоретичним аналізом та аналоговими схемами на дискретних компонентах, цифрова FPGA-реалізація забезпечує детерміновану відтворюваність хаотичних послідовностей за заданих початкових умов та відсутність дрейфу параметрів. Для апаратного обчислення трансцендентної нелінійності $\tanh$ у функції $G(x_1)$, що описує вольт-амперну характеристику транзисторної структури з від'ємним опором, запропоновано кусково-лінійну апроксимацію з 31 сегментом (максимальна похибка 2.575×10^{-2}). З урахуванням коефіцієнта $d = 0.01$ у рівнянні системи ефективний вплив PWL-похибки на динаміку не перевищує 2.6×10^{-4} . Реалізація потребує 4990 логічних елементів (32% EP4CE15) та 90 вбудованих множників 9×9 (80%), що у 8-10 разів перевищує поліноміальні системи.

8. Проведено верифікацію та випробування всіх п'яти генераторів хаотичних сигналів на платформі EP4CE15F23C8 з виведенням сигналів через 14-бітний ЦАП на цифровий осцилограф. Базові статистичні характеристики (середнє значення, стандартне відхилення, мінімум, максимум) збігаються з Python-еталоном з точністю до 5% для всіх п'яти систем, а форми атракторів візуально ідентичні. Зменшення кроку інтегрування з $dt = 0.02$ до $dt = 0.005$ покращило RMSE від 4.1 разу (система Росслера) до 20.1 разу (система Рікітакі), що підтверджує домінуючий вплив методу Ейлера першого порядку як джерела похибки. Ресурсний аналіз демонструє можливість інтеграції генератора хаотичних сигналів з компонентами IoT-вузла на одному кристалі EP4CE15 для захисту каналу передачі телеметричних даних.

Достовірність теоретичних положень і висновків підтверджена збігом результатів математичного моделювання у середовищі Python з результатами апаратних реалізацій на FPGA, а також експериментальними дослідженнями розроблених прототипів IoT-вузлів та генераторів хаотичних сигналів. Результати дисертаційної роботи впроваджені у Sinai RnD LLC, м. Вінниця, а також у навчальний процес Вінницького національного технічного університету.

Отримані в дисертаційній роботі нові наукові та практичні результати є корисними для розробки, серійного виготовлення та експлуатації IoT-систем на основі FPGA для промислового моніторингу та збору телеметричних даних від частотних сенсорів, а також генераторів хаотичних сигналів для задач захисту інформації, для науково-дослідної роботи та навчального процесу з підготовки фахівців галузі 17 «Електроніка та телекомунікації» у вищих навчальних закладах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. O. V. Osadchuk, I. O. Osadchuk, V. K. Skoshchuk. IoT system with frequency converters of physical quantities on FPGA. *IAPGOS*, vol. 15, iss. 3, pp. 59–66, 2025. DOI: <https://doi.org/10.35784/iapgos.7076>
2. O. Osadchuk, I. Osadchuk, V. Skoshchuk. Using FPGA for modelling and generating chaotic processes. *IAPGOS*, vol. 16, iss. 1, pp. 72–77, 2026. DOI: <https://doi.org/10.35784/iapgos.7583>
3. O. Osadchuk, C. Kaczmarek, I. Osadchuk, O. Nanaka, O. Melnyk, V. Skoshchuk, V. Petrenko, N. Smailov. Optical-electronic generator of determinative chaos based on bipolar field transistor structure with negative differential resistance. *Proc. SPIE* 14009, 140091A, 2025. DOI: <https://doi.org/10.1117/12.3098870>
4. О. В. Осадчук, Я. О. Осадчук, В. К. Скощук. Багатоканальний частотомір на ПЛІС для радіовимірювальної системи з частотними сенсорами фізичних величин. *Вісник Хмельницького національного університету*. Технічні науки, № 6, 2021, С. 186–194. DOI: <https://doi.org/10.31891/2307-5732-2021-303-6-186-194>
5. О. Осадчук, Я. Осадчук, В. Скощук. Використання ядра NIOS II у багатоканальному частотомірі на FPGA для радіотехнічної системи з частотними сенсорами фізичних величин. *Вимірювальна та обчислювальна техніка в технологічних процесах*, (1), 2023, С. 137–148. DOI: <https://doi.org/10.31891/2219-9365-2023-73-1-19>
6. О. Осадчук, Я. Осадчук, В. Скощук. Удосконалення багатоканальної радіотехнічної системи на FPGA з підтримкою цифрових сенсорів. *Вимірювальна та обчислювальна техніка в технологічних процесах*, (2), 2023, С. 72–82. DOI: <https://doi.org/10.31891/2219-9365-2023-74-10>
7. О. Осадчук, Я. Осадчук, В. Скощук. Інтеграція LILYGO LORA32 у багатоканальну радіотехнічну систему на FPGA для частотних перетворювачів

фізичних величин. *Вимірювальна та обчислювальна техніка в технологічних процесах*, № 4, 2024, С. 74–83. DOI: <https://doi.org/10.31891/2219-9365-2024-80-10>

8. О. Осадчук, В. Скощук. Бездротова система збору даних на FPGA з використанням LoRa. *Вісник Хмельницького національного університету. Технічні науки*, Том 347, № 1, 2025, С. 375–386. URL: <https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/1047>

9. О. В. Осадчук, Я. О. Осадчук, В. І. Петренко, В. К. Скощук, К. В. Шикун. Експериментальне дослідження генератора детермінованого хаосу на основі транзисторної структури. *Оптико-електронні інформаційно-енергетичні технології*, Том 49, № 1, 2025, С. 235–246. DOI: <https://doi.org/10.31649/1681-7893-2025-49-1-235-246>

10. О. В. Осадчук, Я. О. Осадчук, В. І. Петренко, В. К. Скощук. Розробка генератора детермінованого хаосу на основі транзисторної структури з від'ємним опором. *Оптико-електронні інформаційно-енергетичні технології*, Том 49, № 1, 2025, С. 247–256. DOI: <https://doi.org/10.31649/1681-7893-2025-49-1-247-256>

11. О. Осадчук, Я. Осадчук, В. Скощук, В. Петренко. Генератор детермінованого хаосу на основі біполярно-польової транзисторної структури з від'ємним диференціальним опором. *Вимірювальна та обчислювальна техніка в технологічних процесах*, 82(2), 2025, С. 240–249. URL: <https://vottp.khmnu.edu.ua/index.php/vottp/article/view/511>

12. О. В. Осадчук, Я. О. Осадчук, В. К. Скощук, В. І. Петренко, К. В. Шикун. FPGA-реалізація хаотичного атрактора на основі структури з від'ємним опором за допомогою модифікованої моделі Аніщенка-Астахова. *Оптико-електронні інформаційно-енергетичні технології*, Том 50, № 2, 2025, С. 311–321. DOI: <https://doi.org/10.31649/1681-7893-2025-50-2-311-321>

13. В. К. Скощук, О. В. Осадчук. Передачі накопичених даних радіотехнічною системою на FPGA для частотних перетворювачів фізичних величин використовуючи технологію WiFi. *Матеріали Міжнародної науково-*

практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)», 15–16 червня 2025 р., Вінниця, ВНТУ, 2025. С. 1–4. [Електронний ресурс]. URL:

<https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/download/23195/19197>

14. В. К. Скощук, О. В. Осадчук. Збір вимірювань із багатоканальних радіотехнічних систем на FPGA для частотних перетворювачів фізичних величин використовуючи LoRa концентратор на базі LilyGO LoRa32. *Тези доповідей LIV Всеукраїнської науково-технічної конференції факультету інформаційних електронних систем (ВНТКІП ВНТУ-2025)*, 24–27 березня 2025 р., Вінниця, ВНТУ, 2025. С. 1–4. [Електронний ресурс]. URL:

<https://conferences.vntu.edu.ua/index.php/all-frtzip/all-frtzip-2025/paper/download/23194/19193>

15. В. К. Скощук, О. В. Осадчук. Проектування структури системи передачі даних для багатоканальної радіотехнічної системи на FPGA з використанням LoRa. *Матеріали IX Міжнародної науково-практичної конференції «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем (СПІРН-2023)»*, 15–17 листопада 2023 р., Вінниця, ВНТУ, 2023. С. 1–3. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/spirn/spirn2023/paper/view/19141>

16. В. К. Скощук, О. В. Осадчук, Я. О. Осадчук. Використання ядра NIOS II як шлях до удосконалення багатоканальної радіотехнічної система на FPGA для частотних перетворювачів фізичних величин. *Матеріали LII Науково-технічної конференції факультету інформаційних електронних систем (2023)*, 21–23 червня 2023 р., Вінниця, ВНТУ, 2023. С. 1–4. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/all-frtzip/all-frtzip-2023/paper/download/17111/14402>

17. В. К. Скощук, О. В. Осадчук, Я. О. Осадчук. Використання цифрових сенсорів у багатоканальній радіотехнічній системі на FPGA для частотних перетворювачів фізичних величин із ядром NIOS II. *Матеріали Міжнародної*

науково-практичної інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)», 22–23 червня 2023 р., Вінниця, ВНТУ, 2023. С. 1–4. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/mn/mn2023/paper/view/17110>

18. О. В. Осадчук, В. К. Скощук. Багатоканальний частотомір на FPGA для радіовимірювальної системи з частотними сенсорами. *8-а Міжнародна науково-практична конференція «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем» (СПІРН-2021)*, 03–05 листопада 2021 р., Вінниця, ВНТУ, 2021. С. 1–3. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/spirn/spirn2021/paper/download/13742/11583>

19. О. В. Осадчук, Я. О. Осадчук, В. К. Скощук. Багатоканальна радіовимірювальна система на ПЛІС для частотних перетворювачів фізичних величин. *Матеріали І Науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ-2021)*, м. Вінниця, 10–12 березня 2021 р., Вінниця: ВНТУ, 2021. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/all-frtzip/all-frtzip-2021/paper/view/11435>

20. Oleksandr Osadchuk, Iaroslav Osadchuk, Paweł Komada, Aliya Kalizhanova, Valentyn Skoshchuk, Vitaliy Petrenko, Yaroslav Yaroslavskyy, Mikhailo Vatso. Self-oscillating parametric optical power transducer with photosensitive bipolar transistor. *Proceedings of SPIE - The International Society for Optical Engineering*, 30 December 2025. DOI: <https://doi.org/10.1117/12.3100537>

21. В. К. Скощук, О. В. Осадчук. Реалізація системи передачі даних для багатоканальної радіотехнічної системи на FPGA з використанням LoRa. *IX Міжнародна науково-технічна конференція «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем» (СПІРН-2023)*, 15–17 листопада 2023 р., Україна, м. Вінниця, ВНТУ, 2023. С. 1–4. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/spirn/spirn2023/paper/view/19142>

22. T. Qin, S. J. Bleiker, S. Rana, F. Niklaus et al. Performance Analysis of Nanoelectromechanical Relay-Based Field-Programmable Gate Arrays. *IEEE Access*, vol. 6, pp. 15997–16009, 2018. DOI: <https://doi.org/10.1109/ACCESS.2018.2816781>.
23. A. Magyari, Y. Chen. Review of State-of-the-Art FPGA Applications in IoT Networks. *Sensors*, vol. 22, no. 19, Art. 7496, 2022. DOI: <https://doi.org/10.3390/s22197496>
24. D. M. Pham, S. M. Aziz. FlexiS—A Flexible Sensor Node Platform for the Internet of Things. *Sensors*, vol. 21, no. 15, Art. 5154, 2021. DOI: <https://doi.org/10.3390/s21155154>
25. N. K. Jumaa. Survey: Internet of Thing Using FPGA. *Iraqi Journal for Electrical and Electronic Engineering*, vol. 13, no. 1, pp. 38–45, 2017. DOI: <https://doi.org/10.37917/ijeee.13.1.5>
26. M. Vaithianathan et al. FPGA-Based Smart Health Monitoring Systems for Wearable Devices. *International Journal of ISAE*, vol. 12, no. 4, pp. 537–546, 2024.
27. J. Chen, S.-W. Jun. Myrmec: FPGA-Accelerated SmartNIC for Cost and Power Efficient IoT Sensor Networks. *Proc. SAMOS 2023*, LNCS 14385, pp. 57–71, 2023.
28. S. N. Silva, M. A. C. Fernandes. High-Throughput and Low-Power FPGA Implementation of SHA-256 for IoT Devices. *Sensors*, vol. 24, no. 12, Art. 3908, 2024. DOI: <https://doi.org/10.3390/s24123908>
29. A. Braeken, A. Touhafi. Sensor Systems Based on FPGAs and Their Applications: A Survey. *Sensors*, vol. 12, no. 9, pp. 12335–12364, 2012. DOI: <https://doi.org/10.3390/s120912335>
30. P. D. Nguyen et al. An IoT Hardware Platform Architecture for Monitoring Power Grid Systems Based on Heterogeneous Multi-Sensors. *Sensors*, vol. 20, no. 21, Art. 6082, 2020. DOI: <https://doi.org/10.3390/s20216082>
31. R. Chéour et al. Recent Trends of FPGA Used for Low-Power Wireless Sensor Network. *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, no. 10, pp. 28–38, 2019. DOI: <https://doi.org/10.1109/MAES.2019.2938663>

32. S. Y. Yurish. Sensors and Transducers: Frequency Output versus Voltage Output. *Sensors and Transducers Journal*, vol. 49, no. 12, pp. 302–305, 2004.
33. F. N. Murrieta-Rico et al. Sensors with frequency output. *Applied Mechanics and Materials*, vol. 756, pp. 575–581, 2015. DOI: <https://doi.org/10.4028/www.scientific.net/AMM.756.575>
34. A. V. Osadchuk, I. A. Osadchuk. Pressure transducer on the basis of reactive properties of transistor structure with negative resistance. *Proc. SPIE* 9816, 98161C, 2015. DOI: <https://doi.org/10.1117/12.2229211>
35. O. V. Osadchuk, I. O. Osadchuk. Frequency pressure transducer with a sensitivity of mem capacitor based on microelectronic transistor structure with negative resistance. *Proc. SPIE* 10445, 104452B, 2017. DOI: <https://doi.org/10.1117/12.2280958>
36. A. V. Osadchuk et al. Noncontact infrared thermometer based on a self-oscillating lambda type system. *23rd International Crimean Conference*, 2013.
37. A. V. Osadchuk et al. Temperature transducer based on metal-pyroelectric-semiconductor structure with NDR. *Proc. SPIE* 10808, 108085D, 2018. DOI: <https://doi.org/10.1117/12.2501625>
38. O. V. Osadchuk et al. Optical transducers with frequency output. *Proc. SPIE* 10445, 2017. DOI: <https://doi.org/10.1117/12.2280892>
39. A. V. Osadchuk et al. The mathematical model of frequency gas transducer based on transistor structure with NDR. *Proc. SPIE* 11176, 111762T, 2019. DOI: <https://doi.org/10.1117/12.2536850>
40. A. V. Osadchuk et al. Optical-frequency gas flow meter on the basis of transistor structures with NDR. *Proc. SPIE* 11456, 1145616, 2020. DOI: <https://doi.org/10.1117/12.2569771>
41. A. V. Osadchuk et al. Self-Oscillating Parametric Humidity Sensor with Frequency Output Signal. *IAPGOS*, 2023. DOI: <https://doi.org/10.35784/iapgos.3476>

42. V. S. Osadchuk, A. V. Osadchuk. The Microelectronic Radiomeasuring Transducers of Magnetic Field with a Frequency Output. *Elektronika ir Elektrotechnika*, ISSN 1392-1215, 2011. No. 4(110).
43. A. V. Osadchuk et al. Numerical method for processing frequency measuring signals from microelectronic sensors based on transistor structures with NDR. *Proc. SPIE* 11176, 111765Y, 2019. DOI: <https://doi.org/10.1117/12.2536942>
44. Y.-Y. Fang, X.-J. Chen. Design of Equal Precision Frequency Meter Based on FPGA. *Engineering*, vol. 4, no. 10, pp. 696–700, 2012. DOI: <https://doi.org/10.4236/eng.2012.410089>
45. A. O. Triqadafi et al. Four-Channels High-Resolution Frequency Counter for QCM Sensor Array. *Journal of Electrical and Computer Engineering*, vol. 2023, Art. 5182455, 2023. DOI: <https://doi.org/10.1155/2023/5182455>
46. A. Bourenane et al. Accurate Multi-Channel QCM Sensor Measurement Enabled by FPGA-Based Embedded System Using GPS. *Electronics*, vol. 12, no. 12, Art. 2666, 2023. DOI: <https://doi.org/10.3390/electronics12122666>
47. M. Ferrari et al. Strategies for the Accurate Measurement of the Resonance Frequency in QCM-D Systems via Low-Cost Digital Electronics. *Sensors*, vol. 22, no. 15, Art. 5728, 2022. DOI: <https://doi.org/10.3390/s22155728>
48. A. Gazis et al. Comprehensive Review: Sensor Technologies, Instrumentation, and Signal Processing in Low-Power IoT. *Academia Engineering*, vol. 2, no. 1, 2025. DOI: <https://doi.org/10.20935/AcadEng7541>
49. A. Sharma et al. A Review of Sensors and Their Application in Internet of Things (IoT). *International Journal of Computer Applications*, vol. 174, no. 24, 2021. DOI: <https://doi.org/10.5120/ijca2021921148>
50. M. Javaid et al. Sensors for daily life: A review. *Sensors International*, vol. 2, 2021. Elsevier.

51. J. A. Gastmaier Marques et al. Open-Source FPGA Implementation of an I3C Controller. *Chips (MDPI)*, vol. 4, no. 1, Art. 6, 2025. DOI: <https://doi.org/10.3390/chips4010006>
52. Mishra, Rawat. FPGA Improved I2C Protocol for Multiple Slave Device Using VHDL. *International Journal of Microelectronics*, 2025.
53. S. Sawant et al. Interfacing of Sensors with FPGA Board Using I2C Protocol. *IOSR Journal of Engineering*, pp. 16–21, 2021.
54. G. Beniwal, A. Singhrova. A systematic literature review on IoT gateways. *Journal of King Saud University — Computer and Information Sciences*, vol. 34, no. 10, pp. 9541–9563, 2022. DOI: <https://doi.org/10.1016/j.jksuci.2021.11.007>
55. S. Hamdan, M. Ayyash, S. Almajali. Edge-Computing Architectures for Internet of Things Applications: A Survey. *Sensors*, vol. 20, no. 22, Art. 6441, 2020. DOI: <https://doi.org/10.3390/s20226441>
56. P. Pierleoni, R. Concetti, A. Belli, L. Palma. Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison. *IEEE Access*, vol. 8, pp. 5455–5470, 2020. DOI: <https://doi.org/10.1109/ACCESS.2019.2961511>
57. C. Gemirter, S. Senturca, S. Baydere. A Comparative Evaluation of AMQP, MQTT and HTTP Protocols Using Real-Time Public Smart City Data. *Proceedings of the 6th International Conference on Computer Science and Engineering (UBMK)*, pp. 542–547, 2021. DOI: <https://doi.org/10.1109/UBMK52708.2021.9559032>
58. Industrial Internet Consortium. The Industrial Internet of Things Volume G1: Reference Architecture. IIC-PUB-G1-V1.80-20170131, pp. 10–46, 2017.
59. L. Kane, V. Liu, M. McKague, G. Walker. An Experimental Field Comparison of Wi-Fi HaLow and LoRa for the Smart Grid. *Sensors*, vol. 23, no. 17, Art. 7409, 2023. DOI: <https://doi.org/10.3390/s23177409>
60. C. Horne, N. Peters, M. Ritchie. Classification of LoRa Signals With Real-Time Validation Using the Xilinx Radio Frequency System-on-Chip. *IEEE Access*, vol. 11, pp. 26211–26223, 2023. DOI: <https://doi.org/10.1109/ACCESS.2023.3252170>

61. Z. Zheng, W. Cui, L. Qiao, J. Guo. Performance and Power Consumption Analysis of IEEE802.11ah for Smart Grid. *Wireless Communications and Mobile Computing*, vol. 2018, Art. 5286560, 8 p., 2018. DOI: <https://doi.org/10.1155/2018/5286560>
62. Q. H. Ta, V. K. Ta, T. T. Nguyen, H. Oh. A WSN and LoRa Hybrid Multimedia Transmission Protocol for Scalar Data and Image Transmission. *Sensors*, vol. 24, no. 24, Art. 8165, 2024. DOI: <https://doi.org/10.3390/s24248165>
63. M. E. Gazi, H. Keshmiri, K. A. Wahid. LoRa Resource Allocation Algorithm for Higher Data Rates. *Sensors*, vol. 25, no. 2, Art. 518, 2025. DOI: <https://doi.org/10.3390/s25020518>
64. S. Ould, N. S. Bennett. Energy Performance Analysis and Modelling of LoRa Prototyping Boards. *Sensors*, vol. 21, no. 23, Art. 7992, 2021. DOI: <https://doi.org/10.3390/s21237992>
65. P.-D. Gleonec, J. Ardouin, M. Gautier, O. Berder. Energy Allocation for LoRaWAN Nodes with Multi-Source Energy Harvesting. *Sensors*, vol. 21, no. 8, Art. 2874, 2021. DOI: <https://doi.org/10.3390/s21082874>
66. A. Magyari, Y. Chen. Integrating Lorenz Hyperchaotic Encryption with Ring Oscillator PUFs for IoT. *Sensors*, vol. 23, no. 3, Art. 1536, 2023.
67. Y. Guang, Y. Yang, Y. Wu. Chaos Based Lightweight Cryptographic Algorithm Design and FPGA Implementation. *Applied Sciences*, vol. 12, no. 8, Art. 4023, 2022.
68. T. M. Youssef et al. A Secure Chaos-Based Lightweight Cryptosystem for the Internet of Things. *IEEE Access*, vol. 11, pp. 52399–52413, 2023.
69. M. F. Tasdemir et al. FPGA-Based Chaotic Oscillator Designs and Performance Analysis. *Chaos and Fractals*, vol. 2, no. 1, pp. 8–13, 2025. DOI: <https://doi.org/10.69882/adba.chf.2025012>

70. I. Damaj et al. Optimizing FPGA implementation of high-precision chaotic systems for improved performance. *PLoS ONE*, vol. 19, no. 4, e0299021, 2024. DOI: <https://doi.org/10.1371/journal.pone.0299021>
71. S. Sadoudi et al. Design and FPGA implementation of a wireless hyperchaotic communication system for secure real-time image transmission. *EURASIP Journal on Image and Video Processing*, 2013:43, 2013. DOI: <https://doi.org/10.1186/1687-5281-2013-43>
72. J. Schmitz, L. Zhang. Rössler-based chaotic communication system implemented on FPGA. *IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2017. DOI: <https://doi.org/10.1109/CCECE.2017.7946729>
73. F. N. Kemdoun et al. FPGA based implementation of a perturbed Chen oscillator for random number generation. *Scientific Reports*, vol. 14, Art. 21262, 2024. DOI: <https://doi.org/10.1038/s41598-024-71531-y>
74. L. Renteria et al. FPGA-Based Numerical Simulation of the Chaotic Synchronization of Chua Circuits. *MDPI Computation*, vol. 12, no. 9, Art. 174, 2024. DOI: <https://doi.org/10.3390/computation12090174>
75. W. A. Al-Musawi et al. Implementation of Chaotic System using FPGA. *6th Asia-Pacific Conference on Applied Information and Communication Research Systems (ACIRS)*, IEEE, 2021. DOI: <https://doi.org/10.1109/ACIRS52449.2021.9519360>
76. S. Vaidyanathan et al. FPGA-Based Implementation of a New 3-D Multistable Chaotic Jerk System with Two Unstable Balance Points. *MDPI Technologies*, vol. 11, no. 4, Art. 92, 2023. DOI: <https://doi.org/10.3390/technologies11040092>
77. T. Bonny, A. S. Elwakil. FPGA realizations of high-speed switching-type chaotic oscillators using compact VHDL codes. *Nonlinear Dynamics*, vol. 93, no. 2, pp. 819–833, 2018. DOI: <https://doi.org/10.1007/s11071-018-4229-7>
78. M. Tuna et al. High speed FPGA-based chaotic oscillator design. *Microprocessors and Microsystems*, vol. 66, pp. 72–80, 2019. DOI: <https://doi.org/10.1016/j.micpro.2019.02.012>

79. A. Garg et al. An FPGA based Real time Implementation of Nosé-Hoover Chaotic System and its Application in Communication. *7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, IEEE, 2021. DOI: <https://doi.org/10.1109/ICACCS51430.2021.9441923>

80. V. S. Anishchenko, V. V. Astakhov, A. B. Neiman, T. E. Vadivasova, L. Schimansky-Geier. Nonlinear Dynamics of Chaotic and Stochastic Systems: Tutorial and Modern Developments. *Springer Series in Synergetics*, 2nd ed., 2007. ISBN: 978-3-540-38164-6.

81. C. Hu et al. A new chaotic system with novel multiple shapes of two-channel attractors. *Chaos, Solitons and Fractals*, vol. 162, Art. 112454, 2022. DOI: <https://doi.org/10.1016/j.chaos.2022.112454>

82. A. Kilina et al. Multi-dimensional chaos initiated by short pulses in non-autonomous radio-physical generator. *Communications in Nonlinear Science and Numerical Simulation*, vol. 134, Art. 108041, 2024. DOI: <https://doi.org/10.1016/j.cnsns.2024.108041>

83. S. Sorayaasa. Different Implementation Methods of Tanh on FPGAs for Neural Networks Application. *M.Sc. thesis, University of Windsor*, 2022.

84. Z. Hajduk, G. R. Dec. Very High Accuracy Hyperbolic Tangent Function Implementation in FPGAs. *IEEE Access*, vol. 11, pp. 23701–23713, 2023. DOI: <https://doi.org/10.1109/ACCESS.2023.3253668>

85. Z. Hajduk. Hardware implementation of hyperbolic tangent and sigmoid activation functions. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, vol. 66, no. 5, pp. 563–577, 2018. DOI: <https://doi.org/10.24425/124272>

86. Z. Li et al. FPGA Implementation for the Sigmoid with Piecewise Linear Fitting Method Based on Curvature Analysis. *MDPI Electronics*, vol. 11, no. 9, Art. 1365, 2022. DOI: <https://doi.org/10.3390/electronics11091365>

87. NXP Semiconductors. I²C-bus specification and user manual. *UM10204*, Rev. 7.0, October 2021. URL: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>

88. *InvenSense (TDK)*. MPU-6000 and MPU-6050 Product Specification. Revision 3.4, 2013. URL: <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>
89. *FreeRTOS*. FreeRTOS Real-Time Operating System. Amazon Web Services. URL: <https://www.freertos.org/>
90. *Zephyr Project*. Zephyr RTOS Documentation. The Linux Foundation. URL: <https://docs.zephyrproject.org/>
91. Semtech Corporation. SX1276/77/78/79 — 137 MHz to 1020 MHz Low Power Long Range Transceiver. *Datasheet, Rev. 7*, March 2020.
92. C. Bormann, P. Hoffman. Concise Binary Object Representation (CBOR). *RFC 8949, IETF, December 2020*. DOI: <https://doi.org/10.17487/RFC8949>
93. Espressif Systems. ESP-IDF Programming Guide. *Version 5.x*, 2024. URL: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/>
94. J. Grosz. RadioLib — Universal wireless communication library for embedded devices. *Version 6.x*, 2024. URL: <https://github.com/jgromes/RadioLib>
95. S. Ramírez. FastAPI — Modern, fast web framework for building APIs with Python. URL: <https://fastapi.tiangolo.com/>
96. *SQLite Consortium*. SQLite Documentation. URL: <https://www.sqlite.org/docs.html>
97. А. О. Семенов. Хаотична автоколивальна система з кубічною інерційною нелінійністю на основі приладу з від'ємним опором. *Вимірювальна та обчислювальна техніка в технологічних процесах*, № 3(56), 2016, С. 77–81. ISSN 2219-9365.
98. А. О. Семенов, О. В. Осадчук. Генератор динамічного хаосу з інерційною нелінійністю на основі біполярної транзисторної структури з від'ємним опором. *Вісник Вінницького політехнічного інституту*, № 6, 2017, С. 147–152. ISSN 1997-9266.

99. А. О. Семенов. Дослідження генератора детермінованого хаосу на основі біполярної транзисторної структури з від'ємним опором. *Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки*, Том 29(68), № 1, Частина 1, 2018, С. 50–57. ISSN 1606-3721.

ДОДАТКИ

ДОДАТОК 1

Акти впровадження результатів дисертаційної роботи



Sinai RnD LLC

34 N Franklin Ave Ste 687, #5046, Pinedale, WY
82941, USA
RN: 2022-001121818

№ D25/12-01

від 10 грудня 2025 р.

Довідка

про впровадження результатів дисертаційної роботи Скощука Валентина Костянтиновича
на тему «Радіотехнічні прилади на FPGA з частотними сенсорами та генераторами
детермінованого хаосу для IoT-систем збору даних»

Цією довідкою підтверджується, що за результатами дисертаційної роботи аспіранта кафедри інформаційних радіоелектронних технологій і систем Вінницького національного технічного університету Скощука В.К. у Sinai RnD LLC були використані у проєктно-конструкторській діяльності з розробки програмного забезпечення для вбудованих систем та IoT-рішень такі результати:

1. Архітектуру периферійного IoT-вузла на базі FPGA Cyclone IV EP4CE15F23C8 з апаратно-програмним розподілом функцій, що забезпечує детерміноване зчитування даних від частотних та цифрових сенсорів, формування та підготовку телеметрії до передавання, використано при проєктуванні вбудованих систем збору даних.

2. Наскрізну гетерогенну архітектуру збору та передачі телеметрії в системі «сенсор - FPGA - LoRa - шлюз (ESP32) - хмара (AWS)», включаючи серверне рішення на FastAPI та мобільний застосунок на Android/Kotlin, використано як основу при розробці програмного забезпечення для систем дистанційного моніторингу IoT-пристроїв.

3. Методи апаратної реалізації генераторів детермінованого хаосу на FPGA (системи Лоренца, Росслера, Нозе-Гувера, Рікітакі та модифікованої системи Аніщенко-Астахова) на основі методу Ейлера першого порядку із використанням формату фіксованої коми Q16.16 та кусково-лінійної апроксимації для модифікованої системи Аніщенко-Астахова, використано при дослідженні можливостей апаратного захисту телеметрії у вбудованих IoT-системах.

Використання результатів дисертаційної роботи Скощука В.К. дозволило скоротити час проєктування вбудованих IoT-систем збору даних за рахунок застосування готових апаратних модулів (частотомір, ІС-контролер) та програмних компонентів (бібліотека LoraTransport, серверний REST API), а також розширити функціональні можливості систем дистанційного моніторингу.

Дана довідка видана Скощуку В.К. для пред'явлення за місцем вимоги та не може бути підставою для фінансових розрахунків між вище зазначеними організаціями.

Директор
Sinai RnD LLC

Роман КРИНОЧКІН

**ЗАТВЕРДЖУЮ**

Проректор з науково-педагогічної роботи та організації освітнього процесу
Вінницького національного технічного університету

к.т.н. доц. Олександр ПЕТРОВ

« 12 » 01 2026 р.

АКТ

про впровадження результатів дисертаційної роботи на здобуття наукового ступеня доктора філософії Скощука Валентина Костянтиновича на тему «Радіотехнічні прилади на FPGA з частотними сенсорами та генераторами детермінованого хаосу для IoT-систем збору даних»

Комісія у складі Тимчика С.В. (голова комісії), Осадчука О.В. і Савицького А.Ю. (члени комісії) підтверджує, що у Вінницькому національному технічному університеті для забезпечення навчального процесу кафедри інформаційних радіоелектронних технологій і систем з дисциплін «Сенсори фізичних величин», «Радіовимірювальні перетворювачі інформації», «Вимірювання в телекомунікації та радіотехніці», «Конструювання та технологія телекомунікаційних та радіотехнічних систем», «Радіотехнічні прилади з від'ємним опором», «Радіоелектронні системи» для студентів спеціальності 172 – Телекомунікації та радіотехніка першого (бакалаврського) рівня підготовки, а також при виконанні бакалаврських і магістерських кваліфікаційних робіт студентів спеціальності 172 – Телекомунікації та радіотехніка впроваджено такі результати дисертаційної роботи Скощука В.К.:

- лабораторні макети периферійного IoT-вузла на базі FPGA Cyclone IV EP4CE15F23C8 з апаратно-програмним розподілом функцій, що включають 12-канальний апаратний частотомір для вимірювання параметрів частотних сенсорів фізичних величин, апаратний ІС-контролер для підключення цифрових сенсорів та soft-core процесор NIOS II для програмного керування та обробки даних, а також методичне забезпечення для проведення лабораторних робіт;
- лабораторні макети генераторів детермінованого хаосу на FPGA (системи Лоренца, Росслера, Нозе-Гувера, Рікітакі та модифікованої системи Аніщенка-Астахова) з виведенням сигналів через ЦАП та спостереженням на осцилографі, а також методичне забезпечення для проведення лабораторних робіт з дисципліни «Радіотехнічні прилади з від'ємним опором»;
- архітектуру наскрізного каналу передавання телеметрії в гетерогенній IoT-системі «сенсор – FPGA – LoRa – шлюз (ESP32) – хмара (AWS)» та принципи побудови бездротових сенсорних мереж на базі технології LoRa з хмарною інфраструктурою;
- методику проєктування апаратних модулів на FPGA мовами Verilog HDL та VHDL для задач збору та обробки сенсорних даних.

Голова комісії

декан ФІЕС к.т.н., доцент

Сергій ТИМЧИК

Члени комісії:

завідувач кафедри ІРТС д.т.н., професор

доцент кафедри ІРТС, к.т.н., доцент

Олександр ОСАДЧУК

Антон САВИЦЬКИЙ

ДОДАТОК 2

Список публікацій здобувача. Відомості про апробацію

Наукові праці, в яких опубліковані основні наукові та практичні результати дисертації:

- матеріали дисертаційної роботи, викладені у періодичних виданнях, що індексовані в наукометричних базах Scopus та Web of Science:

1. Oleksandr V. Osadchuk, Iaroslav O. Osadchuk, Valentyn K. Skoshchuk. IoT system with frequency converters of physical quantities on FPGA. *Informatyka, Automatyka, Pomiarы w Gospodarce i Ochronie Środowiska*, Volume 15, Issue 3, Pages 59–66, 2025. DOI: <https://doi.org/10.35784/iapgos.7076> (Scopus) ISSN 2083-0157 (print); eISSN 2391-6761 (online)

2. Oleksandr Osadchuk, Iaroslav Osadchuk, Valentyn Skoshchuk. Using FPGA for modelling and generating chaotic processes. *Informatyka, Automatyka, Pomiarы w Gospodarce i Ochronie Środowiska*, Volume 16, Issue 1, Pages 72–77, 2026. DOI: <https://doi.org/10.35784/iapgos.7583> (Scopus) ISSN 2083-0157 (print); eISSN 2391-6761 (online)

3. Oleksandr Osadchuk, Cezary Kaczmarek, Iaroslav Osadchuk, Olena Nanaka, Oleksandr Melnyk, Valentyn Skoshchuk, Vitaliy Petrenko, Nurzhigit Smailov. Optical-electronic generator of determinative chaos based on bipolar field transistor structure with negative differential resistance. *Proceedings of SPIE - The International Society for Optical Engineering*, Volume 14009, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2025, Article 140091A, 2025. DOI: <https://doi.org/10.1117/12.3098870> (Scopus, Web of Science). ISSN: 0277-786X; eISSN: 1996-756X. ISBN: 9781510699748; eISBN: 9781510699755

- матеріали дисертаційної роботи, викладені у виданнях, що внесені до переліку наукових фахових видань України категорії Б для захисту дисертацій з технічних наук:

4. Осадчук О.В., Осадчук Я.О., Скощук В.К. Багатоканальний частотомір на програмованій логічній інтегральній схемі для радіовимірювальної системи з частотними сенсорами фізичних величин. *Вісник Хмельницького національного університету*. Серія: Технічні науки. № 6, 2021 (303). С. 186–194. DOI: <https://www.doi.org/10.31891/2307-5732-2021-303-6-186-194> ISSN 2307-5732

5. Осадчук О., Осадчук Я., Скощук В. Використання ядра NIOS II у багатоканальному частотомірі на FPGA для радіотехнічної системи з частотними сенсорами фізичних величин. *Вимірювальна та обчислювальна техніка в технологічних процесах*, (1), 2023, 137–148. DOI: <https://doi.org/10.31891/2219-9365-2023-73-1-19> ISSN 2219-9365

6. Осадчук Олександр, Осадчук Ярослав, Скощук Валентин. Удосконалення багатоканальної радіотехнічної системи на FPGA для частотних перетворювачів фізичних величин підтримкою цифрових сенсорів. *Вимірювальна та обчислювальна техніка в технологічних процесах*, (2), 2023, С. 72–82. DOI: <https://doi.org/10.31891/2219-9365-2023-74-10> ISSN 2219-9365

7. Осадчук О., Осадчук Я., Скощук В. Інтеграція LILYGO LORA32 у багатоканальну радіотехнічну систему на FPGA для частотних перетворювачів фізичних величин. *Вимірювальна та обчислювальна техніка в технологічних процесах*, № 4, 2024, С. 74–83. DOI: <https://doi.org/10.31891/2219-9365-2024-80-10> ISSN 2219-9365

8. Осадчук О., Скощук В. Бездротова система збору даних на FPGA для частотних перетворювачів фізичних величин із використанням LoRa. *Herald of Khmelnytskyi National University. Technical Sciences (Вісник Хмельницького національного університету. Серія: Технічні науки)*. Том 347 № 1, 2025. С. 375–

386. URL: <https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/1047> ISSN 2307-5732

9. Осадчук О.В., Осадчук Я.О., Петренко В.І., Скощук В.К., Шикун К.В. Експериментальне дослідження генератора детермінованого хаосу на основі транзисторної структури. *Оптико-електронні інформаційно-енергетичні технології*. Том 49 № 1, 2025, С. 235–246. DOI: <https://doi.org/10.31649/1681-7893-2025-49-1-235-246> ISSN 2311-2662 (Online), ISSN 1681-7893 (Print)

10. Осадчук О.В., Осадчук Я.О., Петренко В.І., Скощук В.К. Розробка генератора детермінованого хаосу на основі транзисторної структури з від'ємним опором. *Оптико-електронні інформаційно-енергетичні технології*. Том 49 № 1, 2025. С. 247–256. DOI: <https://doi.org/10.31649/1681-7893-2025-49-1-247-256> ISSN 2311-2662 (Online), ISSN 1681-7893 (Print)

11. Осадчук О., Осадчук Я., Скощук В., Петренко В. Генератор детермінованого хаосу на основі біполярно-польової транзисторної структури з від'ємним диференціальним опором. *Вимірювальна та обчислювальна техніка в технологічних процесах*, 82(2), 2025, С. 240–249. URL: <https://vottp.khmnu.edu.ua/index.php/vottp/article/view/511> ISSN 2219-9365

12. Осадчук О.В., Осадчук Я.О., Скощук В.К., Петренко В.І., Шикун К.В. FPGA-реалізація хаотичного атрактора на основі структури з від'ємним опором за допомогою модифікованої моделі Аніщенка-Астахова. *Оптико-електронні інформаційно-енергетичні технології*. Том 50, № 2 (2025). С. 311–321. DOI: <https://doi.org/10.31649/1681-7893-2025-50-2-311-321> ISSN 2311-2662 (Online), ISSN 1681-7893 (Print)

- наукові праці, які засвідчують апробацію матеріалів дисертації:

13. Скощук В.К., Осадчук О.В. Передачі накопичених даних радіотехнічною системою на FPGA для частотних перетворювачів фізичних величин використовуючи технологію WiFi. *Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи*

(МН-2025)», 15–16 червня 2025 р., Вінниця, ВНТУ, 2025. С. 1–4. [Електронний ресурс]. URL:

<https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/download/23195/19197>

14. Скощук В.К., Осадчук О.В. Збір вимірювань із багатоканальних радіотехнічних систем на FPGA для частотних перетворювачів фізичних величин використовуючи LoRa концентратор на базі LilyGO LoRa32. *Тези доповідей LIV Всеукраїнської науково-технічної конференції факультету інформаційних електронних систем (ВНТКІП ВНТУ-2025)*, 24–27 березня 2025 р., Вінниця, ВНТУ, 2025. С. 1–4. [Електронний ресурс]. URL:

<https://conferences.vntu.edu.ua/index.php/all-frtzip/all-frtzip-2025/paper/download/23194/19193>

15. Скощук В. К., Осадчук О. В. Проектування структури системи передачі даних для багатоканальної радіотехнічної системи на FPGA з використанням LoRa. *Матеріали IX Міжнародної науково-практичної конференції «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем (СПІРН-2023)»*, 15–17 листопада 2023 р., Вінниця, ВНТУ, 2023. С. 1–3. [Електронний ресурс]. URL:

<https://conferences.vntu.edu.ua/index.php/spirn/spirn2023/paper/view/19141>

16. Скощук В.К., Осадчук О.В., Осадчук Я.О. Використання ядра NIOS II як шлях до удосконалення багатоканальної радіотехнічної система на FPGA для частотних перетворювачів фізичних величин. *Матеріали LII Науково-технічної конференції факультету інформаційних електронних систем (2023)*, 21–23 червня 2023 р., Вінниця, ВНТУ, 2023. С. 1–4. [Електронний ресурс]. URL:

<https://conferences.vntu.edu.ua/index.php/all-frtzip/all-frtzip-2023/paper/download/17111/14402>

17. Скощук В.К., Осадчук О.В., Осадчук Я.О. Використання цифрових сенсорів у багатоканальній радіотехнічній системі на FPGA для частотних перетворювачів фізичних величин із ядром NIOS II. *Матеріали Міжнародної науково-практичної інтернет-конференції студентів, аспірантів та молодих*

науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)», 22–23 червня 2023 р., Вінниця, ВНТУ, 2023. С. 1–4. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/mn/mn2023/paper/view/17110>

18. Осадчук О.В., Скощук В.К. Багатоканальний частотомір на FPGA для радіовимірювальної системи з частотними сенсорами. *8-а Міжнародна науково-практична конференція «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем» (СПІРН-2021)*, 03–05 листопада 2021 р., Вінниця, ВНТУ, 2021. С. 1–3. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/spirn/spirn2021/paper/download/13742/11583>

19. Осадчук О.В., Осадчук Я.О., Скощук В.К. Багатоканальна радіовимірювальна система на ПЛІС для частотних перетворювачів фізичних величин. *Матеріали І Науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ-2021)*, м. Вінниця, 10–12 березня 2021 р., Вінниця: ВНТУ, 2021. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/all-frtzp/all-frtzp-2021/paper/view/11435>

20. Oleksandr Osadchuk, Iaroslav Osadchuk, Paweł Komada, Aliya Kalizhanova, Valentyn Skoshchuk, Vitaliy Petrenko, Yaroslav Yaroslavskyy, Mikhailo Vatso. Self-oscillating parametric optical power transducer with photosensitive bipolar transistor. *Proceedings of SPIE - The International Society for Optical Engineering*, 30 December 2025. DOI: <https://doi.org/10.1117/12.3100537> (Scopus, Web of Science). ISSN: 0277-786X; eISSN: 1996-756X. ISBN: 9781510699748; eISBN: 9781510699755

21. Скощук В. К., Осадчук О. В. Реалізація системи передачі даних для багатоканальної радіотехнічної системи на FPGA з використанням LoRa. *IX Міжнародна науково-технічна конференція «Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем» (СПІРН-2023)*, 15–17 листопада 2023 р., Україна, м. Вінниця, ВНТУ, 2023. С. 1–4. [Електронний ресурс]. URL: <https://conferences.vntu.edu.ua/index.php/spirn/spirn2023/paper/view/19142>