МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Кваліфікаційна наукова
праця на правах рукопису

ГЕРМАН ЮРІЙ ВОЛОДИМИРОВИЧ

УДК 621.391.8:004.272.43

Дисертація

Синтез вузлів цифрової обробки сигналів засобами високорівневого проектування на базі систем на кристалі типу SoC FPGA

172 – Телекомунікації та радіотехніка
17 Електроніка та телекомунікації

Подається на здобуття наукового ступеня доктора філософії

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

 Ю.В. Герман

Науковий керівник: кандидат технічних наук, асистент кафедри радіотехніки та інформаційної безпеки Круліковський Олег Валерійович

Чернівці – 2026

АНОТАЦІЯ

Герман Ю.В. Синтез вузлів цифрової обробки сигналів засобами високорівневого проектування на базі систем на кристалі типу SoC FPGA Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 172 – «Телекомунікації та радіотехніка» (17 — Електроніка та телекомунікації). — Чернівецький національний університет імені Юрія Федьковича, Чернівці, 2026.

Дисертаційна робота присвячена вирішенню важливої науково-прикладної задачі підвищення ефективності та детермінованості вузлів цифрової обробки сигналів у телекомунікаційних системах кордонного рівня (Edge). Основна увага зосереджена на розробці та обґрунтуванні методики їх синтезу засобами високорівневого проектування на базі гетерогенних платформ SoC FPGA. Запропонований підхід поєднує гнучкість вбудованих операційних систем із продуктивністю програмованої логіки, що дозволяє забезпечити обробку потоків даних у реальному часі зі стабільними часовими характеристиками.

У вступі обґрунтовано актуальність теми дисертаційної роботи, сформульовано мету та завдання проведених досліджень, визначено наукову новизну та практичне значення одержаних результатів, представлені методи, об’єкт і предмет досліджень, зазначено особистий внесок здобувача.

У першому розділі проаналізовано сучасні методи побудови вузлів цифрової обробки сигналів. Обґрунтовано доцільність застосування гібридного маршруту проєктування на базі SoC FPGA, що поєднує високорівневий синтез (*HLS*) та апаратне розвантаження (*Hardware Offloading*) критичних операцій для систем кордонних обчислень.

Другий розділ присвячено методології побудови уніфікованого керуючого середовища. Доведено можливість забезпечення часової детермінованості ОС Linux загального призначення шляхом просторового рознесення обчислювальних ресурсів та впровадження «Паспорта конфігурації», що дозволило звести систематичну похибку вимірювань затримок до нуля.

Наукова новизна результатів, отриманих у другому розділі, полягає у тому, що:

1. **Вперше запропоновано** метод конфігурації вузлів цифрової обробки

сигналів на базі гетерогенних систем на кристалі, який відрізняється від відомих узгодженням конфігурації середовища вбудованої ОС із часовими характеристиками радіотехнічного тракту із застосуванням математичної моделі простору станів вектора $S_{\text{passport}} = \{H, M, R\}$, де формалізуються апаратні інваріанти, параметри пам'яті та політики виконання, що дало змогу перетворити неконтрольоване програмне середовище на детермінований об'єкт, наближаючи систематичну похибку вимірювань часових затримок до нуля ($f(S_{\text{hidden}}) \to 0$) і тим самим гарантувати в контексті телекомунікаційних систем фіксацію прихованих станів арбітражу системних шин, усунути апаратно-програмний фазовий шум під час транзакцій DMA та забезпечити детерміновану обробку безперервних широкосмугових потоків I/Q-даних без втрати сигнальних кадрів на граничних (*Edge*) вузлах.

Третій розділ присвячено практичній реалізації гетерогенних вузлів цифрової обробки сигналів. Експериментально встановлено «точку насичення» програмної обробки на процесорі (20 MSPS) та доведено ефективність архітектурного розділення площини даних (*Data Plane*) і площини керування (*Control Plane*) для забезпечення стабільності вузла в автономному режимі.

Наукова новизна висновків, отриманих у третьому розділі, полягає у тому, що:

1. **Вперше запропоновано** метод просторового виявлення та пеленгації джерел радіовипромінювання з борту БПЛА на основі багатоканального антено-фідерного тракту з односпрямованими антенами різних частотних діапазонів, який відрізняється від відомих застосуванням двохетапної процедури пошуку — оглядового кругового сканування з подальшим адаптивним локальним уточненням напрямку, що дозволяє скоротити час пошуку T_{find} у 2 рази порівняно з повним растровим скануванням за координатами (φ, θ) за умови однакових кроків дискретизації та часу інтеграції t_d.
2. **Удосконалено** метод синтезу гетерогенних вузлів цифрової обробки сигналів типу «HPS-Centric» у системах на кристалі шляхом адаптивного використання ресурсів програмно-апаратної платформи, який відрізняється від існуючих застосуванням програмованої логіки як

високошвидкісного інтерфейсно-буферного модуля тракту приймання та первинного перетворення даних, тоді як адаптивне керування потоками, конфігурація режимів функціонування та прийняття рішень реалізуються на рівні процесорної підсистеми, що уможливлює динамічний розподіл функцій між апаратним і програмним рівнями під час проєктування пристроїв на базі SoC FPGA.

У четвертому розділі вирішено проблему обмеженої пропускної здатності трактів ЦОС. Шляхом синтезу IP-ядра потокового криптографічного перетворення (ДСТУ 8845:2019) на рівні регістрових передач (RTL) доведено, що апаратне розвантаження арифметики полів Галуа забезпечує лінійне масштабування продуктивності до 83,2 Гбіт/с. Розроблено адаптерний шар для динамічної реконфігурації параметрів безпеки в реальному часі, що є критичним для систем із псевдовипадковим перелаштуванням робочої частоти (ППРЧ).

Наукова новизна результатів, отриманих у четвертому розділі, полягає у тому, що:

1. **Набув подальшого розвитку критерій** «точки насичення» для вбудованих універсальних процесорів у трактах цифрової обробки сигналів, який, на відміну від існуючих підходів, формалізує межі ефективної обробки даних залежно від ширини смуги вхідного сигналу та обчислювальних ресурсів процесорної підсистеми, що дає змогу визначити умови доцільності застосування апаратного розвантаження з метою забезпечення детермінованості часових характеристик тракту обробки.
2. **Вперше запропоновано архітектуру** криптографічного вузла телекомунікаційних систем для реалізації алгоритму «Струмок», яка ґрунтується на поєднанні оптимізованого RTL-опису обчислювального ядра з високорівневим описом інтерфейсів керування та відрізняється від існуючих апаратною ізоляцією тракту даних і оптимізацією шинних транзакцій для підтримки режиму швидкої зміни криптографічного контексту, що уможливило використання менш ніж 6% логічних ресурсів ПЛІС Cyclone V, досягнення пропускної здатності 6,4 Гбіт/с на одне обчислювальне ядро за тактової частоти 100 МГц та швидкості реконфігурації понад 7000 оновлень ключів за секунду, даючи змогу реалізувати адаптивні протоколи захисту в умовах динамічної зміни

радіочастотної обстановки.

При виконанні дисертаційної роботи отримано наступні практичні результати:

1. **Розроблено та впроваджено уніфіковану апаратно-програмну платформу** на базі системи на кристалі SoC FPGA Cyclone V для побудови гетерогенних вузлів цифрової обробки сигналів. Платформа забезпечує стабільну обробку та маршрутизацію квадратурних I/Q-потоків із частотою дискретизації до **20 MSPS** у реальному часі, що дозволяє використовувати її як базовий модуль для систем широкосмугового радіомоніторингу та програмно-визначеного радіо (SDR).
2. **Синтезовано функціональний вузол потокового криптографічного захисту** каналів зв'язку, що реалізує національний стандарт ДСТУ 8845:2019 «Струмок» на рівні регістрових передач (RTL). Розробка характеризується високою ресурсною ефективністю (використання $< 6\%$ логічної ємності кристала) та забезпечує динамічну зміну параметрів шифрування (*Key Agility*) зі швидкістю понад **7000 оновлень за секунду**, що є критичним для захищених систем із псевдовипадковим перелаштуванням робочої частоти (ППРЧ).
3. **Розроблено методику та програмний інструментарій просторової ізоляції обчислювальних ресурсів** (*Spatial Isolation*), що дозволяє інтегрувати операційну систему загального призначення Linux у контури керування телекомунікаційним обладнанням. Це уможливило зниження фазового тремтіння (джитера) усієї системи до рівня, сумісного з вимогами стандартів **IEC 61850** та **5G URLLC** (затримка передачі команди < 1 мс), без застосування спеціалізованих RTOS.
4. **Реалізовано автономний вузол моніторингу радіочастотного спектра**, здатний функціонувати в умовах нестабільного каналу зв'язку. Завдяки впровадженню механізму адаптивного керування потоком (*Implicit Backpressure*) та локальної буферизації телеметрії у транзакційному режимі, забезпечено цілісність накопичення даних при розривах з'єднання, що дозволяє використовувати розробку у складі необслуговуваних постів радіоконтролю та мобільних платформ.

Ключові слова: система на кристалі з програмованою логікою (SoC FPGA),

вбудовані системи, цифрове оброблення сигналів, телекомунікаційні системи та мережі, програмно-визначене радіо (SDR), дрони (безпілотний літальний апарат), апаратне прискорення, контроль затримки та джиттеру, кордонні обчислення, інтернет речей (IoT), системи та протоколи керування, сенсори, інформаційна система, операційна система, оптимізація швидкодії.

ABSTRACT

Herman Y.V. Synthesis of digital signal processing nodes using high-level design tools based on SoC FPGAs Qualifying scientific work on the rights of a manuscript.

Dissertation for the degree of Doctor of Philosophy in specialty 172 – ”Telecommunications and Radioengineering” (17 — Electronics and Telecommunications). — Yuriy Fedkovych Chernivtsi National University, Chernivtsi, 2026.

The dissertation is devoted to solving an important scientific and applied problem of increasing the efficiency and determinism of digital signal processing nodes in edge-level telecommunication systems (Edge). The main focus is on the development and substantiation of a methodology for their synthesis using high-level design tools based on heterogeneous SoC FPGA platforms. The proposed approach combines the flexibility of embedded operating systems with the performance of programmable logic, which allows ensuring real-time data stream processing with stable temporal characteristics.

The introduction substantiates the relevance of the dissertation topic, formulates the purpose and objectives of the research, defines the scientific novelty and practical significance of the obtained results, presents the methods, object, and subject of the research, and outlines the personal contribution of the applicant.

The first chapter analyzes modern methods for constructing digital signal processing nodes. It substantiates the feasibility of using a hybrid design flow based on SoC FPGA, which combines High-Level Synthesis (*HLS*) and Hardware Offloading of critical operations for edge computing systems.

The second chapter is devoted to the methodology of building a unified control environment. It proves the possibility of ensuring the temporal determinism of a general-purpose Linux OS through the spatial separation of computing resources and the

implementation of a ”Configuration Passport,”which allowed reducing the systematic error of latency measurements to zero.

The scientific novelty of the results obtained in the second chapter is that:

1. **For the first time, a method** for configuring digital signal processing nodes based on heterogeneous Systems-on-Chip **is proposed**. It differs from existing methods by aligning the configuration of the embedded OS environment control system with the temporal characteristics of the radio frequency path. This is achieved using a mathematical model of the state space vector $S_{\text{passport}} = \{H, M, R\}$, which formalizes hardware invariants (coherency registers), memory parameters, and OS execution policies. This allowed transforming an uncontrolled software environment into a deterministic system object, reducing the systematic error of temporal latency measurements to zero ($f(S_{\text{hidden}}) \to 0$). In the context of telecommunication systems, this guarantees the fixation of hidden states of system bus arbitration, eliminates hardware-software phase noise during DMA transactions, and ensures deterministic processing of continuous broadband I/Q data streams without the loss of signal frames at the edge nodes of the network.

The third chapter is devoted to the practical implementation of heterogeneous digital signal processing nodes. The ”saturation point”of software processing on the processor (20 MSPS) has been experimentally established, and the effectiveness of the architectural separation of the Data Plane (*Data Plane*) and the Control Plane (*Control Plane*) to ensure node stability in autonomous mode has been proven.

The scientific novelty of the conclusions obtained in the third chapter is that:

1. **For the first time, a method has been proposed** for spatial detection and direction finding of radio emission sources from an unmanned aerial vehicle (UAV), based on a multichannel antenna–feeder network using directional antennas for different frequency bands. The method performs an initial panoramic 360° scan followed by adaptive local refinement of the source direction, and provides at least a twofold reduction of the search time T_{find} compared to full raster scanning over (φ, θ) under the same discretization steps and integration time t_d.
2. **The method** of synthesizing heterogeneous digital signal processing nodes of the **”HPS-Centric”** type in Systems-on-Chip **has been improved** through the adaptive use of hardware-software platform resources. The proposed

approach involves using the FPGA programmable logic as a high-speed interface-buffer module for the receiving path and primary digital signal processing, while adaptive data flow control, configuration of operating modes, and decision-making are implemented at the processor subsystem level. This enables the dynamic allocation of functions between the hardware and software levels when designing digital signal processing systems based on SoC FPGAs.

The fourth chapter solves the problem of limited bandwidth in DSP paths. By synthesizing an IP core for stream cryptographic transformation (DSTU 8845:2019 standard) at the Register Transfer Level (RTL), it has been proven that the hardware offloading of Galois field arithmetic ensures linear performance scaling up to 83.2 Gbps. An adapter layer has been developed for the dynamic reconfiguration of security parameters in real-time, which is critical for Frequency Hopping Spread Spectrum (FHSS) systems.

The scientific novelty of the results obtained in the fourth chapter is that:

1. **The criterion** of a *Saturation Point* for embedded general-purpose processors in digital signal processing paths **has been further developed**. Unlike existing approaches, it formalizes the limits of efficient processing depending on the input signal bandwidth and the processor's computational resources. This allowed defining the conditions for the feasibility of transitioning to Hardware Offloading to ensure the determinism of the channel's temporal characteristics.
2. **For the first time, the architecture** of a cryptographic node in telecommunication systems for implementing the "Strumok"algorithm **is proposed**. It is based on combining an optimized RTL description of the computational core with a high-level (HLS/C) description of the control interfaces, and is distinguished by the hardware isolation of the data path and the optimization of bus transactions to support the rapid cryptographic context change mode (*Key Agility*). This ensured the use of less than **6%** of the *Cyclone V* die's logic resources, achieved a throughput of **6.4 Gbps** per computational core at a clock frequency of **100 MHz**, and a reconfiguration speed of over **7000 key updates per second**, which allows the implementation of adaptive security protocols under conditions of a dynamically changing radio frequency environment.

During the execution of the dissertation work, the following practical results were obtained:

1. **A unified hardware-software platform** based on the SoC FPGA Cyclone V **has been developed and implemented** for building heterogeneous digital signal processing nodes. The platform ensures stable processing and routing of quadrature I/Q streams with a sampling rate of up to **20 MSPS** in real-time, allowing its use as a base module for broadband radio monitoring systems and Software-Defined Radio (SDR).
2. **A functional node** for stream cryptographic protection of communication channels **has been synthesized**, implementing the national standard DSTU 8845:2019 ”Strumok”at the Register Transfer Level (RTL). The development is characterized by high resource efficiency (using $< 6\%$ of the die's logic capacity) and provides dynamic modification of encryption parameters (*Key Agility*) at a rate of over **7000 updates per second**, which is critical for secure FHSS systems.
3. **A methodology and software toolset** for the spatial isolation of computational resources (*Spatial Isolation*) **have been developed**, enabling the integration of a general-purpose Linux operating system into the control loops of telecommunication equipment. This made it possible to reduce the phase jitter of the entire system to a level compliant with the requirements of the **IEC 61850** and **5G URLLC** standards (command transmission latency < 1 ms) without the use of specialized RTOS.
4. **An autonomous radio frequency spectrum monitoring node** capable of functioning under unstable communication channel conditions **has been implemented**. Thanks to the introduction of an adaptive flow control mechanism (*Implicit Backpressure*) and local telemetry buffering in a transactional mode, the integrity of data accumulation during connection drops is ensured. This allows the development to be used as part of unattended radio monitoring posts and mobile platforms.

Keywords: system-on-chip FPGA (SoC FPGA), embedded systems, digital signal processing, telecommunication systems and networks, software-defined radio (SDR), drones (unmanned aerial vehicle), hardware acceleration, jitter and latency control, edge computing, Internet of Things (IoT), control systems and protocols, sensors, information system, operating system, performance optimization.

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації:

Наукові праці у міжнародних періодичних наукових виданнях, проіндексованих у наукометричних базах даних Web of Science Core Collection та/або Scopus:

1. O. Krulikovskyi, **Y. Herman**, and O. Verenko, "Hardware/software communication architecture for a SoC-FPGA-based time-to-digital converter," *Proceedings of SPIE – The International Society for Optical Engineering*, vol. 13813, pp. 214–217, 2025. (Scopus). doi: https://doi.org/10.1117/12.3092023. *(Внесок авторів: Круліковський О.: постановка задачі, розробка апаратної архітектури високошвидкісного тракту передачі даних на FPGA; Герман Ю.: розробка програмно-апаратного шлюзу (Linux-драйвер) та конфігурація механізмів доступу до пам'яті (DMA) через магістраль AXI4; Веренко О.: підготовка експериментального стенду та проведення вимірювань).*

Наукові праці у виданнях, включених до переліку наукових фахових видань України:

2. **Y. Herman**, H. Lastivka, and A. Samila, "Embedded operating systems in IoT edge computing," *Security of Infocommunication Systems and Internet of Things*, vol. 2, no. 2, Art. no. 02001, 2024, doi: https://doi.org/10.31861/sisiot2024.2.02001. *(Внесок авторів: Герман Ю.: концептуалізація, розробка методології застосування вбудованих ОС, програмна реалізація та проведення експериментальних досліджень; Ластівка Г.: підготовка базового набору даних, валідація результатів та допомога в оформленні методології; Саміла А.: постановка задачі, загальне наукове керівництво, рецензування та редагування тексту).*
3. **Герман Ю.** та Саміла А., «Модульна EDGE-система збору та аналізу даних на базі Raspberry Pi та SoC FPGA», Вісник Хмельницького національного університету. Серія: Технічні науки, Т. 357, № 5.2, 2025, С. 86–92, doi: https://doi.org/10.31891/307-5732-2025-357-69.
   (Внесок авторів: Герман Ю.: розробка структурної схеми автономного телеметричного вузла, реалізація алгоритмів агрегації даних (стратегія Offline-First) та програмних інтерфейсів взаємодії; Саміла А.: загальне

наукове керівництво, визначення критеріїв оцінки ефективності розробленої системи, наукове редагування).

4. **Герман Ю.** та Верига А., «Офлайн система обробки та візуалізації радіосигналу», Вимірювальна та обчислювальна техніка в технологічних процесах, Т. 84, № 4, 2025, С. 246–252, doi: https://doi.org/10.31891/2219-9365-2025-84-27. *(Внесок авторів: Герман Ю.: розробка архітектури бортового обчислювального модуля (SDR-сканера), інтеграція радіотракту з системним середовищем Linux, оптимізація конвеєра цифрової обробки сигналів, побудова клієнтського веб-інтерфейсу; Верига А.: розробка методу розподіленої візуалізації спектральних даних, інтеграція обчислювального модуля в систему, загальне наукове керівництво).*
5. **Y. Herman**, “FPGA Platforms and Their Use in Edge Computing,” *Security of Infocommunication Systems and Internet of Things*, vol. 3, no. 2, Art. no. 02008, 2025, doi: https://doi.org/10.31861/sisiot2025.2.02008.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

6. **Герман Ю. В.**, Круліковський О. В. та Верига А. Д., «Автономна система обробки та візуалізації радіосигналів», праці XIV Міжнар. наук.-практ. конф. «Проблеми інформатики та комп’ютерної техніки» (ПІКТ–2025), Чернівці, Україна, 2025, С. 188–190. https://drive.google.com/file/d/12FGgnfM6NA8HPo66h0TgVrfbPHAYDMFC/view *(Внесок авторів: Герман Ю.В.: розробка архітектури програмного шару для безперервної потокової обробки сигналів; Круліковський О.В.: наукове редагування; Верига А.Д.: апаратне налаштування приймального тракту).*
7. **Герман Ю. В.** та Круліковський О. В., «Розробка та використання програмного забезпечення для систем на кристалі типу SoC FPGA Cyclone V», XII Міжнар. наук.-практ. конф. «Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій», Запоріжжя, 2024, С. 27–29. https://old.zp.edu.ua/uploads/dept_s&r/2024/conf/1.4/2024_Radiotekhnika_telekomunikatsiyi_ta_informatsiyni_tekhnolohiyi.pdf *(Внесок авторів: Герман Ю.В.: розробка архітектури програмно-апаратного шлюзу для керування периферією FPGA; Круліковський О.В.: верифікація апаратних ресурсів кристала, наукове консультування).*

8. Круліковський О. В., **Герман Ю. В.** та Веренко О. С., «Програмно-апаратна взаємодія систем на кристалі типу SoC FPGA Cyclone V», Physical and Technological Problems of Transmission, Processing and Storage of Information in Infocommunication Systems: Proceedings of the Xth International Scientific-Practical Conference, Чернівці, 2025, С. 45. https://drive.google.com/file/d/11rJfJNk4H-8W9_uyNE63X84MCMky_Ba-/view *(Внесок авторів: Круліковський О.В.: обґрунтування загального підходу до реалізації радіотехнічних пристроїв із програмним керуванням, оптимізація та конфігурація обраної архітектури; Герман Ю.В.: реалізація програмного доступу до регістрів вводу-виводу через інтерфейс MMIO; Веренко О.С.: тестування часових характеристик мостів HPS-FPGA).*
9. **Герман Ю. В.**, «Оцінка ефективності вбудованих систем на базі Linux та FPGA», XXIX міжнар. молодіж. форум «Радіоелектроніка та молодь у XXI столітті», Харків, 2025, Т. 3, С. 540–542. https://drive.google.com/file/d/1DGXvFT-OADOOOo5CSwDuFfZ52yhVaFAE/view

Наукові праці, які додатково відображають наукові результати дисертації:

10. **Y. Herman**, O. Krulikovskyi, S. Haliuk, and S. Subbotin, “Development of an embedded operating system based on the Linux kernel for SoC FPGA,” *CEUR Workshop Proceedings*, vol. 3702, pp. 376–388, 2024. https://www.scopus.com/pages/publications/85195909635 (Scopus) *(Внесок авторів: Герман Ю.: розробка методики формування спеціалізованого керуючого середовища Linux, проведення експериментального вимірювання системного джитера; Круліковський О.: конфігурація апаратної процесорної підсистеми (HPS); Галюк С.: статистичний аналіз результатів роботи планувальника задач; Субботін С.: методологічне керівництво, наукове редагування).*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- **АЦП** — аналого-цифровий перетворювач
- **БПЛА** — безпілотний літальний апарат (БПАК — безпілотний авіаційний комплекс)
- **ОС** — операційна система
- **ПЛІС** — програмована логічна інтегральна схема
- **ППРЧ** — псевдовипадкове перелаштування робочої частоти
- **ЦОС** — цифрова обробка сигналів
- **ALM** — Adaptive Logic Module (адаптивний логічний модуль ПЛІС)
- **AXI** — Advanced eXtensible Interface (стандарт внутрішньокристальної шини)
- **CMA** — Contiguous Memory Allocator (розподілювач безперервної пам’яті)
- **CPU** — Central Processing Unit (центральний процесор)
- **DMA** — Direct Memory Access (прямий доступ до пам’яті)
- **DTN** — Delay-Tolerant Networking (мережі, стійкі до затримок та розривів)
- **FFT** — Fast Fourier Transform (швидке перетворення Фур’є, ШПФ)
- **FPGA** — Field-Programmable Gate Array (програмована логічна інтегральна схема)
- **FSM** — Finite State Machine (скінченний автомат)
- **HLS** — High-Level Synthesis (високорівневий синтез апаратної логіки)
- **HPS** — Hard Processor System (апаратна процесорна підсистема SoC)
- **I/Q** — In-phase / Quadrature (синфазна та квадратурна складові сигналу)
- **IP-ядро** — Intellectual Property Core (функціональне апаратне ядро)
- **LNA** — Low Noise Amplifier (малошумний підсилювач)
- **LUT** — Look-Up Table (апаратна таблиця пошуку / таблиця підстановки)
- **MMIO** — Memory-Mapped Input/Output (введення-виведення з відображенням у пам’ять)
- **MQTT** — Message Queuing Telemetry Transport (протокол передачі телеметрії)

- **RTL** — Register Transfer Level (рівень регістрових передач)
- **SCU** — Snoop Control Unit (блок керування когерентністю кеш-пам’яті)
- **SDR** — Software-Defined Radio (програмно-визначена радіосистема)
- **SNR** — Signal-to-Noise Ratio (відношення сигнал/шум)
- **SoC** — System-on-Chip (система на кристалі)
- **URB** — USB Request Block (блок запиту USB)
- **WAL** — Write-Ahead Logging (журналювання випереджального запису)

ВСТУП

Актуальність теми. Сучасний етап розвитку телекомунікаційних систем характеризується переходом до концепції кордонних обчислень (*Edge Computing*) та впровадженням технологій 5G/6G, що висуває принципово нові вимоги до вузлів цифрової обробки сигналів. Спостерігається експоненційне зростання обсягів даних, які необхідно обробляти в реальному часі безпосередньо у точці прийому, мінімізуючи затримки передачі на хмарні сервери. Це створює критичне навантаження на апаратні платформи, де традиційні мікропроцесорні архітектури (CPU) вже не здатні забезпечити необхідну пропускну здатність та енергоефективність при обробці широкосмугових потоків даних.

З іншого боку, класичний підхід до розробки спеціалізованих апаратних прискорювачів на базі ПЛІС (FPGA) із застосуванням мов опису апаратури (HDL) характеризується високою трудомісткістю, складністю верифікації та низькою гнучкістю до змін алгоритмів, що є стримуючим фактором в умовах швидкої зміни стандартів зв’язку. Виникає науково-практичне протиріччя між необхідністю забезпечення високої швидкодії, притаманної апаратним рішенням, та вимогою гнучкості керування і швидкості розробки, що забезпечується програмними засобами.

Перспективним шляхом вирішення цієї проблеми є використання гетерогенних систем на кристалі (SoC FPGA), які поєднують потужність програмованої логіки та гнучкість вбудованих процесорних систем. Однак, інтеграція операційних систем загального призначення (зокрема Linux) у контури керування телекомунікаційним обладнанням породжує проблему часової недетермінованості (джитера), що є неприпустимим для систем жорсткого реального часу. Існуючі методики проєктування не пропонують уніфікованого підходу до синтезу таких вузлів, який би гарантував стабільність часових характеристик при збереженні високого рівня абстракції розробки. Тому розроблення методики синтезу гетерогенних вузлів ЦОС, яка б поєднувала ефективність апаратного розвантаження (*Hardware Offloading*) із прогнозованістю програмного керування, є актуальним науково-технічним завданням.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалася відповідно до напряму наукових досліджень кафедри радіотехніки та інформаційної безпеки Чернівецького національного університету імені Юрія Федьковича в межах фундаментальної науково-дослідної роботи: «Методи формування сигнальних конструкцій та інформаційні процеси програмно-апаратної взаємодії широкосмугових телекомунікаційних систем та Інтернету речей» (Державний реєстраційний номер 0121U112870).

Дослідження, апаратні розробки та експерименти, представлені в дисертаційній роботі, також виконувалися в рамках виконання таких науково-технічних проєктів:

- проєкт Національного фонду досліджень України (НФДУ) за конкурсом «Наука для зміцнення обороноздатності України»: «Розробка комплексу для визначення положення та відносної потужності джерел радіовипромінювання та їх візуалізації» (реєстраційний номер проєкту 2023.04/0150);
- проєкт науково-технічної (експериментальної) розробки молодих вчених «Портативний комплекс для наземного аерозондування вибухових закладок» (Державний реєстраційний номер 0123U100679);
- проєкт «Портативний радіоелектронний комплекс синтезу широкосмугових завад» (Державний реєстраційний номер 0125U000836).

У межах виконання зазначених науково-дослідних робіт здобувачем особисто розроблено архітектуру апаратно-програмної взаємодії гетерогенних платформ (SoC FPGA), синтезовано керуюче середовище на базі просторової ізоляції ресурсів та реалізовано алгоритми первинної цифрової обробки сигналів для портативних автономних вузлів.

Мета і завдання дослідження. Метою дисертаційної роботи є підвищення ефективності функціонування та скорочення часу проєктування вузлів цифрової обробки сигналів у телекомунікаційних системах шляхом розроблення та обґрунтування методики їх синтезу із застосуванням засобів високорівневого проектування на базі платформ SoC FPGA, що забезпечує гарантовану часову детермінованість керуючого середовища та високу пропускну здатність апаратних трактів.

Для досягнення поставленої мети необхідно було вирішити такі завдання:

1. Здійснити системний аналіз сучасних методів та засобів побудови вузлів

цифрової обробки сигналів для кордонних обчислень, виявити обмеження класичних маршрутів проєктування та обґрунтувати доцільність застосування засобів високорівневого синтезу (HLS) у гібридних архітектурах SoC FPGA.

2. Розробити методику побудови уніфікованого керуючого середовища на базі вбудованої ОС Linux, яка, на відміну від існуючих підходів, базується на концепції «Паспорта конфігурації» та механізмах просторової ізоляції ресурсів, що дозволяє гарантувати метрологічну відтворюваність часових характеристик.
3. Обґрунтувати архітектуру гетерогенного вузла цифрової обробки сигналів з чітким розмежуванням площини обробки даних (*Data Plane*) та площини керування (*Control Plane*), що забезпечує можливість інтеграції високошвидкісних апаратних прискорювачів без порушення умов м'якого реального часу.
4. Виконати експериментальні дослідження розробленої платформи на прикладі граничних режимів навантаження, встановити межі ефективності програмної реалізації («точку насичення») та перевірити стійкість системи.
5. Дослідити метод апаратного розвантаження (*Hardware Offloading*) обчислювально-критичних трактів у межах гібридного маршруту високорівневого проектування шляхом синтезу IP-ядра потокового шифру на рівні регістрових передач (RTL). Розробити механізм його системної інтеграції у гетерогенне керуюче середовище для забезпечення високої маневреності керуючих команд (ключів).

Об'єкт дослідження. Процеси цифрової обробки сигналів, керування потоками даних та забезпечення часового детермінізму в гетерогенних вбудованих телекомунікаційних системах.

Предмет дослідження. Методи високорівневого проектування, архітектурні моделі організації обчислень та програмно-апаратні засоби синтезу вузлів цифрової обробки сигналів на базі систем на кристалі SoC FPGA.

Методи дослідження. У процесі розв'язання поставлених задач використано: *методи системного аналізу* – для порівняння архітектур обчислювальних платформ та вибору оптимальної елементної бази; *теорію*

скінченних автоматів – для синтезу керуючої логіки апаратних прискорювачів; *методи математичного моделювання* – для оцінки часових затримок та пропускної здатності трактів; *методи натурного експерименту та статистичного аналізу* – для вимірювання джитера, латентності переривань та верифікації параметрів розроблених вузлів; *технології високорівневого проєктування (HLS/RTL)* – для реалізації IP-ядер та системної інтеграції компонентів у спеціалізованих інтегрованих середовищах розробки.

Наукова новизна отриманих результатів. При виконанні дисертаційної роботи отримано такі пункти наукової новизни:

1. **Вперше запропоновано** метод конфігурації вузлів цифрової обробки сигналів на базі гетерогенних систем на кристалі, який відрізняється від відомих узгодженням конфігурації середовища вбудованої ОС із часовими характеристиками радіотехнічного тракту із застосуванням математичної моделі простору станів вектора $S_{\text{passport}} = \{H, M, R\}$, де формалізуються апаратні інваріанти, параметри пам'яті та політики виконання, що дало змогу перетворити неконтрольоване програмне середовище на детермінований об'єкт, наближаючи систематичну похибку вимірювань часових затримок до нуля ($f(S_{\text{hidden}}) \to 0$) і тим самим гарантувати в контексті телекомунікаційних систем фіксацію прихованих станів арбітражу системних шин, усунути апаратно-програмний фазовий шум під час транзакцій DMA та забезпечити детерміновану обробку безперервних широкосмугових потоків I/Q-даних без втрати сигнальних кадрів на граничних (*Edge*) вузлах.
2. **Вперше запропоновано** метод просторового виявлення та пеленгації джерел радіовипромінювання з борту БПЛА на основі багатоканального антено-фідерного тракту з односпрямованими антенами різних частотних діапазонів, який відрізняється від відомих застосуванням двохетапної процедури пошуку — оглядового кругового сканування з подальшим адаптивним локальним уточненням напрямку, що дозволяє скоротити час пошуку T_{find} у 2 рази порівняно з повним растровим скануванням за координатами (φ, θ) за умови однакових кроків дискретизації та часу інтеграції t_d.
3. **Удосконалено** метод синтезу гетерогенних вузлів цифрової обробки

сигналів типу **«HPS-Centric»** у системах на кристалі шляхом адаптивного використання ресурсів програмно-апаратної платформи, який відрізняється від існуючих застосуванням програмованої логіки як високошвидкісного інтерфейсно-буферного модуля тракту приймання та первинного перетворення даних, тоді як адаптивне керування потоками, конфігурація режимів функціонування та прийняття рішень реалізуються на рівні процесорної підсистеми, що уможливлює динамічний розподіл функцій між апаратним і програмним рівнями під час проєктування пристроїв на базі SoC FPGA.

4. **Набув подальшого розвитку критерій** «точки насичення» для вбудованих універсальних процесорів у трактах цифрової обробки сигналів, який, на відміну від існуючих підходів, формалізує межі ефективної обробки даних залежно від ширини смуги вхідного сигналу та обчислювальних ресурсів процесорної підсистеми, що дає змогу визначити умови доцільності застосування апаратного розвантаження з метою забезпечення детермінованості часових характеристик тракту обробки.
5. **Вперше запропоновано архітектуру** криптографічного вузла телекомунікаційних систем для реалізації алгоритму «Струмок», яка ґрунтується на поєднанні оптимізованого RTL-опису обчислювального ядра з високорівневим описом інтерфейсів керування та відрізняється від існуючих апаратною ізоляцією тракту даних і оптимізацією шинних транзакцій для підтримки режиму швидкої зміни криптографічного контексту, що уможливило використання менш ніж 6% логічних ресурсів ПЛІС Cyclone V, досягнення пропускної здатності 6,4 Гбіт/с на одне обчислювальне ядро за тактової частоти 100 МГц та швидкості реконфігурації понад 7000 оновлень ключів за секунду, даючи змогу реалізувати адаптивні протоколи захисту в умовах динамічної зміни радіочастотної обстановки.

Практичне значення отриманих результатів. Практичне значення отриманих результатів полягає у тому, що:

1. **Розроблено та програмно реалізовано** автоматизовану систему збірки вбудованої операційної системи (*Build System*) за принципом «*Full Custom*», котра, на відміну від стандартних засобів (*Yocto/Buildroot*),

дозволяє формувати мінімалістичні образи із гарантованою бінарною відтворюваністю конфігурації. Це дозволяє скоротити час розгортання нових вузлів цифрової обробки сигналів з годин до хвилин.

2. **Створено уніфіковану програмно-апаратну платформу** на базі SoC FPGA Cyclone V для побудови вузлів цифрової обробки сигналів, яка забезпечує сумісне оброблення потоків даних від різнорідних вимірювальних засобів (SDR-сканери, інерціальні системи, екологічні сенсори) без зміни ядра ОС. Платформа підтримує режим «м'якого» реального часу на стандартному ядрі Linux (з базовим шумом планувальника $P_{99} \approx 13$ мкс та гарантованим піковим джитером до 48 мкс), що дозволяє використовувати її замість дорогих спеціалізованих контролерів у задачах промислової автоматизації.
3. **Розроблено мобільний вузол радіомоніторингу**, який, завдяки реалізації механізму «*Implicit Backpressure*», здатен обробляти та візуалізувати широкосмугові сигнали (зі смугою до 20 МГц) у реальному часі на бюджетних процесорах класу Cortex-A9, що значно знижує вартість кінцевого обладнання порівняно з аналогами на базі x86-архітектури.
4. **Створено автономний модуль корисного навантаження для БПЛА**, який реалізує архітектурний патерн накопичення та пересилання (*Store-and-Forward*) у рамках мереж DTN. Модуль забезпечує збір телеметричних сигналів, локальне збереження у транзакційну БД та попередню аналітику даних без необхідності постійного каналу зв'язку, що підвищує надійність виконання місій у складних умовах.
5. **Впроваджено метод апаратного розвантаження** телекомунікаційних трактів вузлів цифрової обробки сигналів шляхом інтеграції IP-ядра шифратора (ДСТУ 8845:2019), синтезованого у межах гібридного маршруту високорівневого проектування. Досягнуто маневреність ключів > 7000 оновлень/с, що є критичним для забезпечення функціональної стійкості радіоканалів із псевдовипадковим перелаштуванням робочої частоти (ППРЧ / FHSS).

Особистий внесок здобувача. Усі результати, представлені в дисертаційній роботі, здобувач отримав самостійно шляхом проведення теоретичних досліджень, комп'ютерного моделювання, експериментальних вимірювань та аналізу отриманих даних. У роботах, опублікованих у співавторстві, особистий

внесок автора конкретизовано у списку публікацій за темою дисертації

Апробація результатів дисертації. Основні результати, отримані в дисертаційній роботі, були предметом обговорень на:

- наукових семінарах кафедри радіотехніки та інформаційної безпеки ЧНУ імені Юрія Федьковича; (очна участь)
- Seventeenth International Conference on Correlation Optics (SPIE) — м. Чернівці, Україна, 8–12 вересня 2025; (онлайн участь)
- Xth International Scientific-Practical Conference «Physical and technological problems of transmission, processing and storage of information in infocommunication systems» — м. Чернівці, Україна, 15–17 травня 2025; (онлайн участь)
- XIV Міжнародній науково-практичній конференції «Проблеми інформатики та комп'ютерної техніки» (ПІКТ–2025) — м. Чернівці, Україна, 13–15 листопада, 2025; (онлайн участь)
- XII Міжнародній науково-практичній конференції «Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій» — м. Запоріжжя, Україна, 10–12 грудня 2024; (онлайн участь)
- XXIX Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» — м. Харків, Україна, 16–-19 квітня 2025; (онлайн участь)
- Seventh International Workshop on Computer Modeling and Intelligent Systems (CMIS-2024) — м. Запоріжжя, Україна, 3 травня 2024. (онлайн участь)

Публікації. Основні результати дисертаційної роботи опубліковано в 9,5 наукових працях, з яких: 1 стаття у міжнародних періодичних наукових виданнях, проіндексованих у наукометричних базах даних Web of Science Core Collection та/або Scopus; 3.5 статті – у наукових фахових виданнях України; 4 праці апробаційного характеру – у матеріалах і тезах доповідей наукових конференцій; 1 праця, котра додатково відображає наукові результати дисертації.

Структура та обсяг дисертації. Дисертаційна робота відповідає встановленим нормативним вимогам. Вона включає вступ, чотири основні розділи, підсумкові висновки, список використаних джерел та додатки. Загальний обсяг роботи становить 186 сторінок, містить 52 рисунки, 17 таблиць, бібліографічний список із 140 джерел та додатки.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ЗАСОБІВ ПРОЄКТУВАННЯ ВУЗЛІВ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ НА БАЗІ SOC FPGA

1.1. Актуальність задачі та контекст

Еволюція сучасних телекомунікаційних систем характеризується експоненціальним зростанням щільності інформаційних потоків. Це висуває нові вимоги до реконфігурованих вузлів цифрової обробки сигналів (**ЦОС**), котрі забезпечують первинний аналіз широкосмугових спектрів, багатоканальних вимірювань та сигнальних конструкцій у реальному часі. Обмеження пропускної здатності магістральних каналів зв'язку, вимоги до мінімізації часових затримок (*latency*) та необхідність забезпечення автономності в умовах нестабільного з'єднання роблять традиційну архітектуру з централізованою агрегацією неефективною. Це зумовлює перехід до децентралізованої обробки безпосередньо на кордонних вузлах мережі (*Edge Nodes*). Концепція кордонних обчислень, у межах якої обробка даних переноситься ближче до джерела їх виникнення, розглядається як відповідь на ці виклики [1–4].

Даний підхід визначається не лише зменшенням затримок, а й можливістю виконувати попередню фільтрацію, агрегацію та аналіз даних безпосередньо на потужностях системи [1, 2]. Таким чином досягається скорочення обсягу прохідного трафіку, підвищується стійкість системи до збоїв мережі. Для комплексних систем цифрової обробки сигналів це означає, що частина об'ємних обчислень має бути делегована, у кращому випадку, або на саму систему, або ж на проміжний шар.

У цьому контексті особливу роль відіграють програмовані логікові інтегральні схеми (ПЛІС) та системи на кристалі типу SoC FPGA. ПЛІС добре пристосовані до реалізації паралельних, конвеєрних структур, високою пропускною здатністю та низькими затримками [5–7]. Вони активно застосовуються у задачах фільтрування сигналів, швидкого перетворення Фур'є, кореляційних обчислень, радіолокації, телекомунікацій і комп'ютерного зору. Системи на кристалі типу SoC FPGA поєднують процесорну підсистему та програмовану

логіку на одному кристалі, що робить їх зручною платформою для обробки сигналів у сфері ЦОС [7, 8]. Поєднання FPGA та процесорної архітектури дозволяє виносити критичні обчислення на рівень апаратних прискорювачів, залишаючи конфігурацію, агрегацію результатів, роботу з мережею під контролем процесорної системи. Узагальнену схему переходу від централізованої хмарної обробки до edge-архітектур із використанням вузлів на базі SoC FPGA для цифрової обробки сигналів наведено на рис 1.1

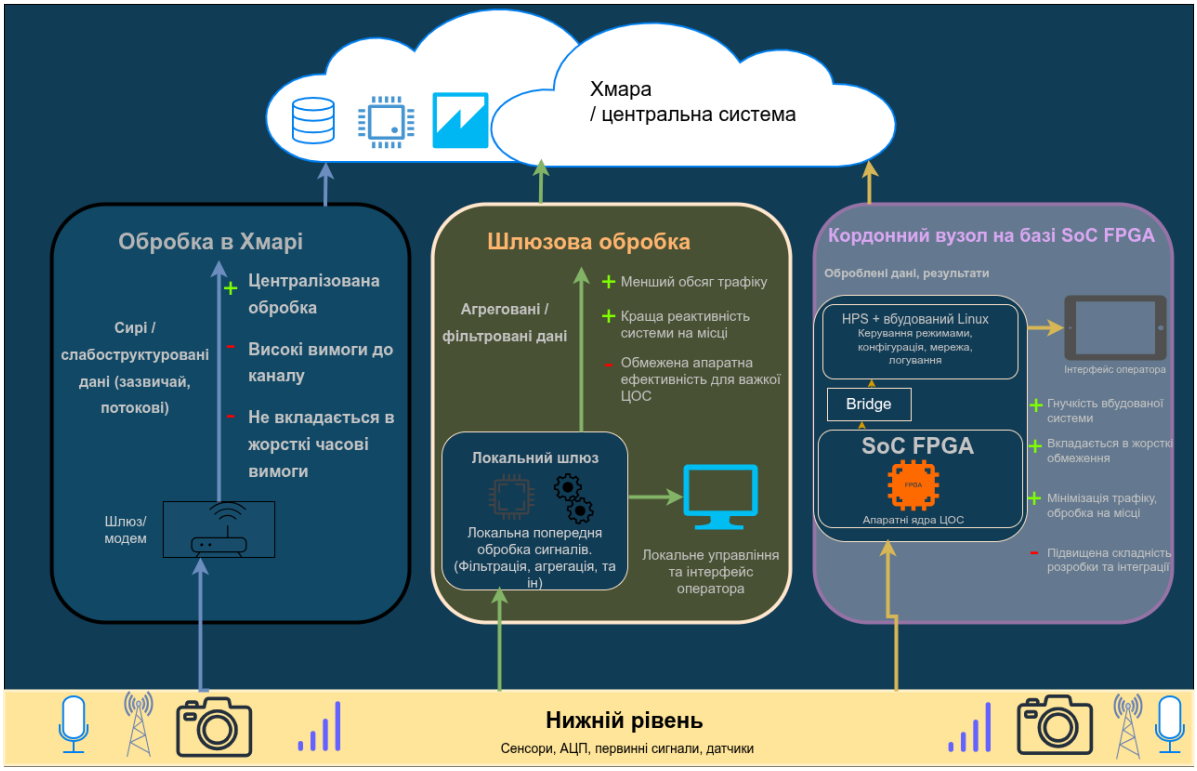


Рис. 1.1: Еволюція від централізованої хмарної обробки до edge-архітектур із вузлами на базі SoC FPGA для цифрової обробки сигналів

Крім того, традиційний цикл проєктування на HDL (VHDL/Verilog) є складним і вимагає багато ресурсів, незважаючи на значні апаратні переваги. Розробка, верифікація та супровід великих HDL-проєктів вимагають значних зусиль [9–11]. Крім того, кожна архітектурна зміна призводить до повного повторення всіх кроків синтезу, аналізу таймінгів і відлагодження. Це неприйнятно для систем, котрі часто потребують швидких змін конфігурації, додавання нових режимів обробки та адаптації алгоритмів до певних сценаріїв застосування або апаратних обмежень. Підвищення ступеня абстракції є одним із способів подолання цієї проблеми. Цього можна досягти за допомогою інструментів високорівневого синтезу (HLS), фреймворків і доменно-специфічних мов. Ці програми можна використовувати для опису

алгоритмів ЦОС на рівні потокових моделей або мов високого рівня, а також для автоматизованої трансляції в HDL. Але навіть за наявності HLS складність систем на базі SoC FPGA не зменшується до рівня «чистого» програмного забезпечення. Це стосується переривань, обмежень часу, апаратних інтерфейсів, взаємодії між доменами HPS–FPGA та механізмів DMA. А отже вимагає архітектури, котра поєднує засоби проєктування з чітко визначеним програмно-апаратним розподілом ролей. Ключову роль у цьому поділі відіграють вбудовані операційні системи, насамперед ті, що базуються на ядра Linux. ОС на базі Linux забезпечує не лише базові функції планування задач, роботи з пам'яттю та периферією, а й повноцінний мережевий стек, файлові системи, механізми логування, оновлення програмного забезпечення та інтеграції з зовнішніми сервісами. У [12] показано, що використання Linux на edge-вузлах дає змогу поєднати локальну обробку даних із включенням у ширший розподілений контур обміну інформацією, що є критичним для IoT- та edge-сценаріїв. Роботи [13–15] демонструють, що саме модифіковані, спеціалізовані Linux-дистрибутиви стали де-факто стандартом для вбудованих систем різних класів.

У випадку SoC FPGA вбудований Linux на HPS виступає «шаром керування та інтеграції» над апаратними ядрами ЦОС. Однак побудова подібних систем вимагає ретельного опрацювання питань взаємодії HPS–FPGA (AXI, DMA, IRQ), структури драйверів і бібліотек, а також підходів до формування мінімізованих, але розширюваних дистрибутивів Linux, адаптованих до конкретних SoC-платформ. Що в свою чергу, визначається сукупністю тенденцій:

1. переходом від централізованих моделей обробки даних до edge-архітектур у сферах, пов'язаних із цифровою обробкою сигналів [1–4]
2. зростанням ролі систем на кристалі типу SoC FPGA як апаратної основи для таких edge-вузлів [5, 7, 8]
3. необхідністю підвищення рівня абстракції в описі вузлів ЦОС при одночасному збереженні контролю над апаратною реалізацією [10, 11]
4. потребою в спеціалізованих вбудованих Linux-системах, що забезпечують ефективну взаємодію між процесорною та програмованою логікою в складі SoC FPGA [12, 16–19].

Така архітектура дозволяє реалізувати розподіл функцій, коли FPGA забезпечує конвеєризовану, визначену за часом обробку сигналів, тоді як Linux відповідає за конфігурацію параметрів, маршрутизацію потоків даних, агрегацію результатів,

їх збереження та передачу до інших вузлів або хмарної інфраструктури [16–18, 20, 21], типові приклади таких задач узагальнено в табл. 1.1

Таблиця 1.1

Типові задачі цифрової обробки сигналів у edge-сценаріях

Галузь	**Типові сигнали**	**Приклади задач на edge-вузлі**
Системи радіомоніторингу та пеленгації	Радіосигнали, їх спектри, часові ряди амплітуд/фаз	Попереднє виявлення цілей, побудова локальних спектрограм, детекція завад та аномалій перед передачею узагальнених даних у центр
Телекомунікації	Адаптивні системи зв’язку	Оцінка якості каналу, адаптація модуляції та кодування, локальна фільтрація трафіку, попереднє виявлення збоїв
Промислові сенсори	Вібросигнали, струми, тиски, температури, акустичні сигнали	Локальна діагностика стану обладнання, прогноз відмов, агрегація та стиснення вимірювань для SCADA/IIoT платформ
Медичні системи моніторингу	ЕКГ, ЕЕГ, фотоплетизмограми, інші біосигнали	Фільтрація артефактів, локальне виявлення подій (аритмії, судоми), формування тривожних сповіщень при мінімальному обсязі переданих даних
Відео- та аудіоаналітика	Потоки з камер і мікрофонів, спектрограми, ознаки зображень	Локальне виділення ознак, трекінг об’єктів, детекція подій/шумів

Саме на перетині цих напрямів формується сучасний науково-прикладний контекст дослідження: системи на кристалі типу SoC FPGA розглядаються як платформа для побудови вузлів цифрової обробки сигналів, у яких апаратні прискорювачі тісно інтегровані з вбудованими Linux-системами та високорівневими засобами проєктування [10–12, 16]. У такому контексті постає потреба в підходах, які одночасно забезпечують ефективне використання ресурсів ПЛІС, гнучку конфігурацію алгоритмів ЦОС та їх придатність до включення у сучасні edge-архітектури.

1.2. Переваги SoC FPGA для цифрової обробки сигналів

Системи на кристалі типу SoC FPGA характерні поєднанням підходів, класичної програмованої логікової інтегральної схеми, так і систем що базуються на процесорній архітектурі (зазвичай ARM). Такий підхід є зручним для задач цифрової обробки сигналів (ЦОС), де одночасно необхідні висока обчислювальна здатність, робота в умовах жорстких обмежень, автономність та можливість гнучкої адаптації до зовнішніх умов. У роботах [8,16,20] показано, що платформи на базі SoC FPGA (напр. Cyclone V) дозволяють об’єднати, в межах одного кристала, апаратні прискорювачі ЦОС та високорівневе програмне забезпечення, котре реалізує управління, конфігурацію та взаємодію із зовнішніми системами. Узагальнену структурну схему такої системи наведено на Рис. 1.2.

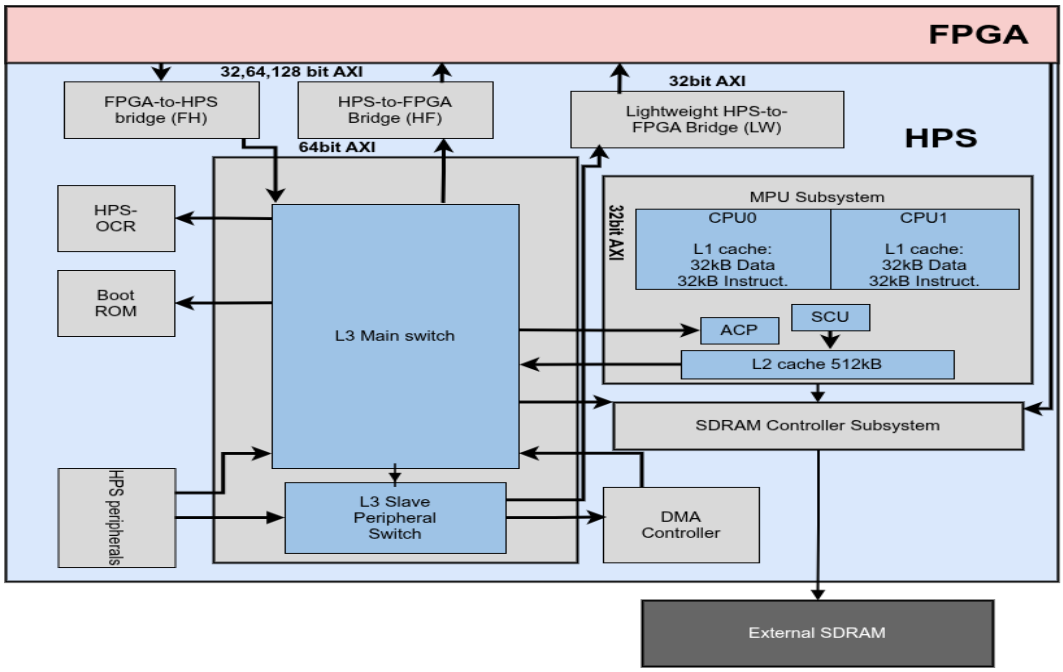


Рис. 1.2: Спрощена блок-діаграма архітектури SoC FPGA Cyclone V

Ключова перевага SoC FPGA у контексті ЦОС полягає в архітектурній підтримці масового паралелізму на низькому рівні.Типові операції обробки сигналів як:

- фільтрація,
- спектральний аналіз,
- кореляція,
- модуляція

добре векторизуються і адаптуються під конвеєрну обробку, що докладно розглянуто у [5]. Реалізація таких алгоритмів у програмованій логіці дає змогу використовувати тисячі паралельних елементарних операцій на такт, що практично недосяжно для центральних процесорів загального призначення. Для платформ типу Cyclone V SoC це означає можливість реалізації багатоканальних трактів обробки або складних систем фільтрування без істотного зростання затримки, що підтверджується результатами експериментальних прототипів, наведених, зокрема, у роботах [8, 22]. Важливою характеристикою є передбачуваність часових затримок у апаратурній реалізації. На відміну від обчислень на CPU чи GPU, де затримка суттєво залежить від роботи планувальника, характеристик кеш-підсистеми (потрапляння або інвалідизація) та конкуренції потоків, конвеєрні FPGA-ядра мають фіксовану довжину шин, передбачувану наскрізну затримку та пропускну здатність. Це критично для задач, пов’язаних із обробкою сигналів, виявленням об’єктів, часовими вимірюваннями або телеметрією, де порушення часових обмежень призводить до деградації якості або інвалідизує отримані дані повністю.

Реалізація інтенсивних частин алгоритмів у FPGA дозволяє досягти високої продуктивності за значно нижчих вимог до споживання порівняно з GPU або навіть високопродуктивними CPU-рішеннями, що підтверджується як експериментальними дослідженнями [5, 23], так і більш сучасними порівняльними роботами для подібних систем [7]. У роботах [17, 24] показано, що використання SoC FPGA у складі вбудованих систем з локальною обробкою даних дає змогу реалізувати енергоефективні рішення для обробки сигналів без залучення високопотужних ресурсів, орієнтуючись на стандарти кордонних систем. Ще одна важлива властивість полягає у гнучкості SoC FPGA з точки зору конфігурації та масштабованості, що теж широко використовується при побудові комплесних кордонних рішень [25]. Завдяки програмованості логічної частини

можливо адаптувати структуру DSP-ланцюжка під конкретні умови задачі:

- змінювати довжину фільтрів,
- топологію обробки,
- частоту дискретизації,
- кількість каналів.

У більш просунутих системах використовується динамічна часткова реконфігурація, яка дозволяє підвантажувати різні апаратні модулі «на льоту» [26]. У поєднанні з процесорною підсистемою це дає змогу реалізувати сценарії автоматичного вибору режиму обробки, адаптації до якості каналу чи рівня завад, а також переключення між режимами сканування та детального аналізу без фізичної зміни периферійної частини системи [16, 22]. Важливим аспектом для подіних систем є поділ ролей між HPS та FPGA. У типовій архітектурі для задач ЦОС обчислювально критичні елементи як:

- фільтри,
- спектральний аналізатор,
- обрахунок кореляції,
- фазові детектори

реалізуються у програмованій логіці, тоді як процесорна частина під керуванням вбудованої операційної системи (найчастіше, заснованій на ядрі Linux) відповідає за налаштування апаратних модулів через AXI-інтерфейси, організацію потоків даних через DMA, попередню та пост обробку, мережеву взаємодію та керуючий інтерфейс.

Такий розподіл дозволяє ізолювати часово критичні процеси від нестабільності програмного середовища, одночасно зберігаючи високу керованість і спостережуваність системи, що продемонстровано в ряді прототипів на базі SoC FPGA Cyclone V [8, 16, 20]. Важливо відзначити здатність систем на базі SoC FPGA саме до інтеграції із різними операційними системами. Маючи доступ до типової процесорної архітектури, такі платформи здатні комбінувати різні інструменти, програмні системи та зовнішню інфраструктуру відповідно до вимог прикладної задачі. Це, дозволяє реалізувати розподіл відповідальності між апаратною частиною та рівнем вбудованої ОС, оптимізуючи кожен шар системи. Вбудовані операційні системи на базі ядра Linux спрощують інтеграцію з широким спектром існуючих інструментів і сервісів, що особливо актуально для edge-систем [12]. Такий підхід спрощує процес розробки ЦОС, дозволяє

закладати взаємодію з іншими елементами глобальної інфраструктури вже на етапі проектування та використовувати стандартизовані підходи до діагностики, оновлення та супроводу.

Наявність розвинених засобів високорівневого проектування (HLS) для SoC FPGA знижує поріг входу у розробку апаратних вузлів ЦОС. Сучасні інструменти дозволяють описувати ядра обробки мовами високого рівня, такими як C/C++ чи OpenCL, та автоматично синтезувати апаратну реалізацію з оптимізованою конвеєризацією й розгортанням паралелізму [5, 27].Це, з одного боку, істотно прискорює розробку, проте з іншого - висуває додаткові вимоги до розуміння внутрішньої структури згенерованих апаратних модулів та їхньої взаємодії з ПЗ на HPS.У сукупності перелічені фактори - апаратний паралелізм, детермінованість затримки, автономність, конфігурованість і щільна інтеграція з процесорною підсистемою та вбудованою ОС - роблять SoC FPGA одним із найбільш придатних класів платформ для реалізації цифрової обробки сигналів у сучасних edge-системах.

У контексті цієї дисертаційної роботи саме зазначені властивості SoC FPGA розглядаються як основа для подальшого синтезу вузлів цифрової обробки сигналів засобами високорівневого проєктування, що забезпечує перенесення обчислювально критичних частин алгоритмів у програмовану логіку при збереженні гнучкості і керованості на рівні вбудованої операційної системи на HPS.

1.3. Особливості програмування ПЛІС та їх недоліки

Головна особливість HDL полягає в тому, що вона оперує елементами схеми, описом їх структури та поведінки. Це обмежує підходи до розробки та ускладнює процес переходу між розробку на самому VHDL та будь-якою іншою мовою вищого порядку. А взявши до уваги той факт, що даний підхід є паралельним у самій своїй основі - то необхідно бути готовим до того, що все в будованій системі буде виконуватися ”одночасно”. Це, в свою чергу, надає величезну перевагу в побудові систем, що якраз і базуються на паралельних обчисленнях, чи отримують вигоду при їх паралелізації. Така особливість робить VHDL/HDL чудовим інструментом для побудови ядр обчислень, зовнішніх прискорювачів, або ж вузькоспеціалізованих масивів операторів, адже саме таким способом ми можемо як проводити цільову розробку та оптимізацію

якогось одного елементу, та широко впроваджувати паралельність обчислень [5]. Проте у випадку розробки широкофункціональних ЦОС, систем де очікується взаємодія із кінцевим користувачем, або ж впровадження проміжних обчислень для отриманих даних - даний підхід втрачає свою гнучкість та піднімає цілий список нових задач. Коли потрібно підтримувати динамічну модифікацію самого алгоритму, додадаткову візуалізацію, агрегації та аналітику - це все негативно впливає на оригінальну структуру будованого модуля, та змушує впроваджувати окремі механізми синхронізації між елементами системи. Сам процес розробки виділяється багатокровістю:

1. Написання коду
2. Синтез та мапінг отриманих елементів
3. Аналіз таймінгів
4. Перенесення синтезованої конфігурації на ПЛІС та її відлагодження на приладі

Даний підхід досить дорого коштує з точки зору затраченого часу, та має зростаючу складність в залежності від масштабності будованої системи. Будь-яка зміна в архітектурі, додавання нового режиму або розширення інтерфейсу часто потребують повторення значної частини вказаного процесу, що ускладнює ітеративну розробку та експериментальну перевірку алгоритмів ЦОС [6, 9].

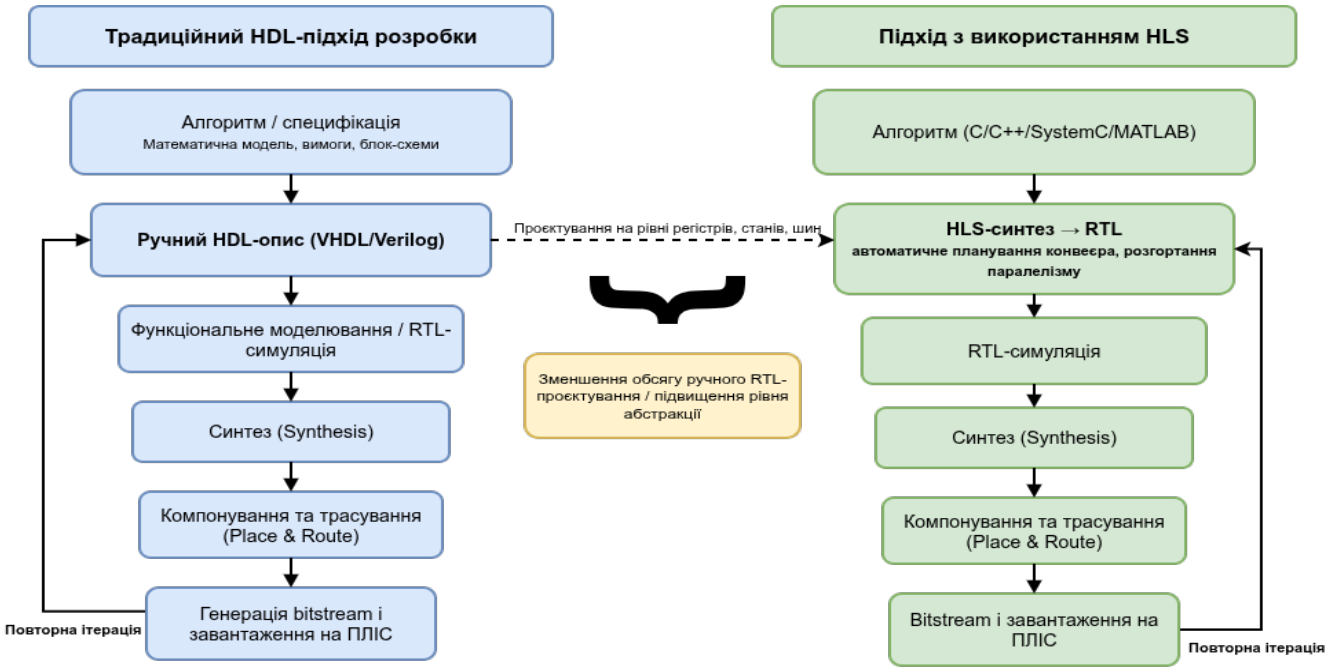


Рис. 1.3: Порівняння традиційного HDL-потоку та потоку з використанням HLS-засобів під час синтезу ядер цифрової обробки сигналів.

Як видно з Рис. 1.3, значна частина ітерацій у класичному потоці припадає

на ручну роботу з RTL-кодом. При чому, навіть правильно спроектована логіка не гарантує коректної роботи синтезованої системи на самому приладі, інколи результуюча архітектура не потрапляє в потрібні часові обмеження, що вже призводить до значних змін в початковому дизайні. Для високопродуктивних конвеєрів ЦОС, що включають в себе фільтрування, перетворення Фур'є, кореляційні обчислення та інші ресурсновимогливі операції навіть незначна затримка може виявитися критичною для коректного функціонування [5, 9]. А додавання додаткових регістрів, розбивка обчислень на кілька тактів чи навіть реорганізація обміну даними між блоками всієї системи не завжди здатне вирішити проблему затримки без значного ускладнення архітектури. Що в свою чергу ускладнює, і так вже комплесний, процес відладки та верифікації. Функціональна перевірка часто вимагає моделювання великої кількості сценаріїв і аналізу поведінки системи у симуляторі. Під час ж роботи на реальному пристрої - обсяг отриманої інформації часто менший, навіть із застосуванням вбудованих, у саму систему, аналізаторів котрі й так обмежують продуктивність самої ПЛІС. Додавання подібних аналізаторів та інших точок спостереження вимагає повторного синтезу та аналізування синтизованої системи, що збільшує час необхідний для виявлення та виправлення помилок. У результаті пошук та локалізація дефектів може займати занадто багато часу.

У випадку систем на кристалі типу SoC FPGA складність зростає ще більше, оскільки до HDL-частини додається задача інтеграції з процесорною підсистемою. Взаємодія між блоками програмованої логіки та HPS зазвичай здійснюється через стандартизовані шини (AXI) та механізми прямого доступу до пам'яті. Некоректний опис протоколів обміну даними або розподіл буферів, помилки синхронізації можуть призвести до важковідтворюваних збоїв - від зависання до спотворення отриманих даних. Усунення таких проблем вимагає одночасного перегляду як HDL-дизайну, так і програмних компонентів на стороні HPS (драйверів, користувацьких додатків), оскільки порушення контракту на одному боці неминуче тягне за собою втрату функціональності на іншому. На практиці це часто стимулює винесення частини логіки керування й обміну даними на вищі рівні абстракції або використання жорсткіше стандартизованих протоколів, але навіть за таких умов проблеми синхронізації та часових затримок повністю не зникають [8, 16, 21]. Масштабованість і супровід HDL-проєктів є ще одним аспектом. Багато розробок, особливо

академічного харатеру та прототипів різного типу, зосереджені на фіксованих параметрах, таких як розрядність, кількість каналів і діапазон частот. Збільшення глибини фільтра або перехід від системи на один канал до багатоканальної нерідко вимагає значної переробки структури коду, окрім зміни параметрів. У довгостроковій перспективі це ускладнює повторне використання рішень, особливо в комплексних ЦОС-системах, де апаратні модулі мають розвиватися разом із програмною частиною.

Важливою також є різниця між проектуванням HDL та розробкою алгоритмів ЦОС на процесорних платформах (CPU/GPU). Зміна структури алгоритму або впровадження додаткової аналітики в програмних системах зазвичай вимагає модифікації коду високого рівня з подальшим тестуванням у середовищі розробки. У цьому випадку час зворотного зв'язку вимірюється хвилинами або секундами, а засоби налагодження надають більше можливостей для інтерактивного аналізу стану системи. Для ПЛІС потрібен повний цикл синтезу, трасування та верифікації, якщо будь-які зміни виходять за межі незначної параметризації. Як наслідок, час, витрачений на розробку, значно збільшується [6, 28]. Відсутність уніфікованої високорівневої екосистеми для повторного використання апаратних компонентів є ще однією характерною проблемою для HDL/VHDL систем. Більшість HDL-проєктів зазвичай пов'язані з конкретними вимогами, архітектурними рішеннями та інструментами конкретного виробника, навіть якщо у світі програмного забезпечення є бібліотеки, фреймворки та сервіси, які дозволяють швидко комбінувати готові модулі. Побудова цілісної системи ЦОС вимагає значної кількості спеціально розробленої проміжної логіки, навіть за наявності параметризованих IP-ядер. Це робить як стандартизацію підходів до проєктування, так і залучення нових розробників складним завданням, оскільки їм потрібно вивчити мову опису апаратури, а також особливості конкретної платформи та інструментів автоматизації проєктування [6, 26, 28]. Через низку обмежень, таких як висока вартість ітерацій, значні зусилля на верифікацію, масштабованість, проблеми з забезпеченням часових обмежень, проблеми з масштабованістю та залежність від інструментів виробника, класичний HDL-підхід не підходить для побудови масштабованих, гнучких систем цифрової обробки сигналів на базі системи на кристалі FPGA [6, 9]. Це безпосередньо спонукає до пошуку засобів високорівневого проєктування, які дозволяють описувати обчислення на рівні алгоритмів, а також більш

тісно інтегрувати апаратні модулі з програмними компонентами вбудованих операційних систем, зокрема на базі ядра Linux [25].

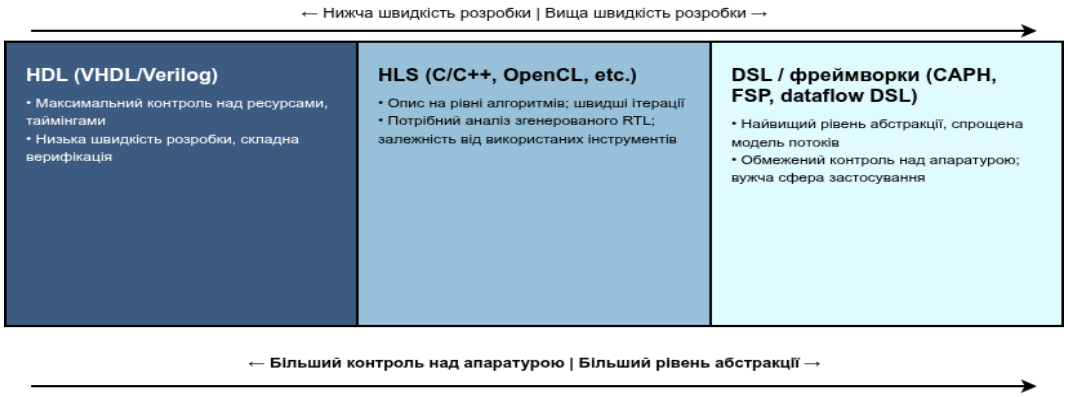


Рис. 1.4: Спектр рівнів абстракції під час проєктування рішень ЦОС: від класичного HDL до засобів високорівневого синтезу та доменно-специфічних мов.

Положення HDL, HLS-засобів та доменно-специфічних мов у загальному спектрі рівнів абстракції під час проєктування ЦОС-проєктів показано на Рис. 1.4. Окремо, слід зазначити, що проекти HDL залежать від інструментів і IP-ядер конкретного виробника. З одного боку, початкова розробка прискорюється за допомогою використання пропрієтарних компонентів, специфічних сценаріїв синтезу та фірмових засобів відлагодження, однак це ускладнює перенесення рішень на інші платформи та підтримку проекту в довгостроковій перспективі. У сукупності зазначених факторів класичний HDL-підхід не підходить як єдиний інструмент для побудови масштабованих, гнучких систем цифрової обробки сигналів на базі SoC FPGA. Ці фактори спонукають людей шукати засоби з високим рівнем проєктування та більш тісно інтегруватися з програмними компонентами вбудованих операційних систем [10, 25, 29, 30].

1.4. Підвищення рівня абстракції: HLS та високорівневі мови

Класичний підхід до проєктування ПЛІС із використанням мов опису апаратури (VHDL/Verilog), як зазначалося вище, надає інженеру детальний контроль над апаратною реалізацією, однак суттєво збільшує витрати ресурсів на всіх етапах життєвого циклу проєкту: від початкового опису логіки, її синтезу та трасування до аналізу часових характеристик і відлагодження безпосередньо на цільовому пристрої. Для складних систем цифрової обробки сигналів, де необхідно не лише реалізувати обчислювальні ядра, а й забезпечити підтримку

змінних режимів роботи, різних конфігурацій фільтрів, масштабування та багатоканальної обробки, такий підхід швидко виявляється малоефективним з точки зору продуктивності розробки [10, 11, 29].

У випадку систем на кристалі типу SoC FPGA додаткову складність створює необхідність узгоджувати HDL-дизайн із програмними компонентами на HPS, реалізованими на мовах високого рівня. Виникає характерне протиріччя: алгоритми ЦОС у більшості випадків розробляються й тестуються саме у високорівневих інструментах (C/C++/Python, MATLAB), тоді як апаратна реалізація вимагає адаптації цих алгоритмів у HDL. Це, у свою чергу, ускладнює процес розробки, робить супровід більш вимогливим та значно обмежує систему в плані модифікацій. Логічним рішенням є підвищення рівня абстракції в проєктуванні, а саме перенесення опису алгоритмів на рівень мов загального призначення або доменно-специфічних високорівневих мов із подальшою трансляцією в HDL, через інструменти автоматизованого синтезу. Високорівневий синтез (High-Level Synthesis, HLS) передбачає декларацію цільового алгоритму мовою високого рівня (C, C++, Python) з подальшим транслюванням в опис на RTL-рівні (VHDL, Verilog), що, у свою чергу, може оброблятися стандартними комплексами програмування систем типу ПЛІС. Хоч це й призводить до втрати високого рівню контролю над результуючу системою, але дозволяє оперувати більшою кількістю загально відомих конструкцій, уникаючи роботи з тригермаи, станами та іншими елементами програмованої логіки. Оскільки ключовими завданнями є побудова розподілу обчислень у часі (scheduling), розподіл ресурсів(alocation) та формування структур та гілок обчислення, що реалізовуються використанням директив (pragma) [30], у випадку виносу цих задач на вищий рівень абстракції - вони часто вирішуються через проміжні шари чи автоматизованим способом, що в свою чергу також вимагає додаткового розуміння процесу трансляції. При необхідності, дані елементи можна реалізовувати на HDL рівні, впроваджуючи саме потрібний функціонал, чи таким чином виіршувати проблему синтезованих декларацій. У випадку ж задач ЦОС даний підхід природньо поєднується з класичною структурою алгоритмів обробки сигналів, а отже - добре описуються рівнем стандартних конструкцій як цикли та функції, а отже може бути відображеним способами мов програмування вищого рівня.

На даний момент для SoC FPGA доступний широкий спектр HLS-засобів,

інтегрованих у фірмові екосистеми, що надаються виробниками різних ПЛІС [10, 11]. У випадку платформ на базі Cyclone V SoC ключову роль відіграють засоби високорівневого синтезу, орієнтовані на інтеграцію з Intel FPGA, що дозволяють генерувати апаратні модулі з C/C++-коду та автоматично синтезувати інтерфейси типу AXI чи окремі шини даних для підключення до HPS. Типовий HLS-проєкт для SoC FPGA включає два взаємопов'язаних компоненти:

- апаратне ядро ЦОС, згенероване із високорівневого опису й інтегроване в програмовану логіку як окремий модуль з визначеним інтерфейсом;
- програмну частину на HPS, яка виконується під керуванням вбудованої операційної системи (наприклад, Linux) і відповідає за конфігурацію ядра, організацію потоків даних, агрегацію результатів та їх передачу далі [29, 31].

Схематичний приклад такого розподілу між апаратною та програмною частинами наведено на Рис. 1.5.

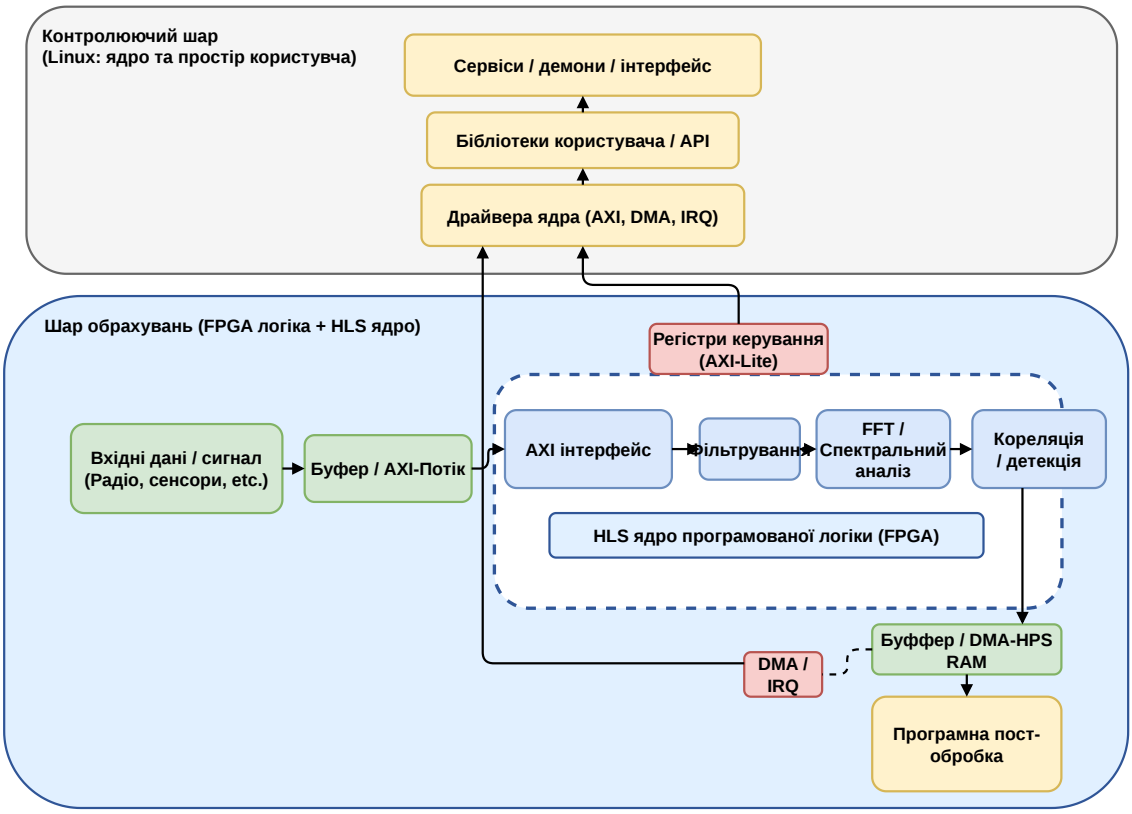


Рис. 1.5: Приклад розподілу алгоритму ЦОС між HLS-ядром у програмованій логіці та керуючим ПЗ на HPS під керуванням Linux.

Це дає змогу працювати із єдиною високорівневою моделлю алгоритму, поступово розносячи окремі його фрагменти між апаратною та програмною

частиною, не повертаючись щоразу до низькорівневого HDL. Однією з основних переваг даного підходу є суттєве скорочення часу розробки апаратних ядер у порівнянні з ручним HDL-синтезом, адже використання мов програмування широкого вжитку дозволяє імплементовувати класичні підходи по перевикористанню, тестування, профілюванню та моделюванню очікуваної системи, або ж навіть використовувати вже існуючі рішення в якості елементів будованого ядра [10, 30]. Як результат, синтезований алгоритм значно легше редагувати, адаптувати або ж, у крайніх випадках, переробляти. У випадку ЦОС вирішується проблема параметризування та масштабування, оскільки даний функціонал легко винести в константи та шаблони. Таким чином операції подібні до зміни розрядності, зміни розміру вікна перетворення, кількості каналів, конфігурації фільтрів та сканованих частот можна звести до модифікації змінних, та надати доступ до такого функціоналу кінцевому користувачу, уникаючи потреби в повторному синтезі ядер чи додавання функціоналу налаштувань та його синхронізацію з основним масивом що поводить обчислення [5, 11]. Попри значний прогрес у розвитку HLS-інструментів, їх використання не здатне повністю покрити список задач, котрі потребують знань в сфері розробки HDL. Якість отриманого RTL-коду значно залежить від написаного високорівневого коду та того як він буде трансльованим у наступний шар, а отже потребує розуміння його особливостей та коректного застосування директив із опором на внутрішні особливості обраного синтезатора. У ряді випадків автоматично згенероване ядро може виявитися менш оптимальним з точки зору використання логічних ресурсів, пам’яті або досяжної тактової частоти, ніж явна HDL-реалізація. Окремою сферою задач є опис алгоритмів з нерегулярним потоком керування, складним масивом залежностей або спеціалізованими інтерфейсами. для їх вирішення часто комбінується написання загальних елементів системи задопомогою високорівневої мови та точкова реалізація спеціалізованих частин, де точність синтезованого HDL є критично важливою, чи потребує явного задання характеристик інтерфейсів. Теж варто зазначити, що важливою особливістю платформ SoC FPGA є взаємодія між її двома елементами та підтримка вбудованих операційних систем, що в свою чергу вимагає рішення цілого спектру задач орієнтованих на взаємодію та обмін даними між даними елементами. Хоч і HLS здатен генерувати значну частину необхідних інтерфейсів (формати регістрів, буфери пам’яті, механізми синхронізації та

флаги) у багатьох випадках саме інтерфейси потребують ручної реаліхації та запровадження контрактів між HLS, HPS та процесорною частиною [16, 21]. Через це можна відмітити що складність будованих систем росте з додаванням їх елементів, і не завжди здатна бути пониженою використовуючи більш високорівневі інструменти. Окремим викликом при використанні HLS у системах із вбудованою ОС є проблема ”інтерфейсної невизначеності”(interface opacity). Хоча HLS-компілятори автоматично генерують обгортки для шин AXI4/Avalon, структура згенерованих транзакцій часто є субоптимальною для роботи у спільному з процесором адресному просторі [10, 29]. Згенеровані ядра можуть створювати непередбачувані сплески (bursts) доступу до пам’яті, що призводить до конфліктів на шині та порушення когерентності даних у кешах HPS [32]. Це створює парадоксальну ситуацію: алгоритмічно ефективне ядро, написане на C++, може демонструвати низьку системну продуктивність через неможливість штатного Linux-драйвера ефективно керувати його доступом до пам’яті. Отже, використання HLS не скасовує, а навпаки, підсилює необхідність розробки спеціалізованих механізмів інтеграції на боці операційної системи (драйверів ”клею”та буферів DMA), які будуть розглянуті у наступних розділах. У випадку ж реалізації ЦОС на подібних системах активно застосовуються підходи, орієнтовані на використання доменно-специфічних мов (DSL) та бібліотек для опису алгоритмів обробки сигналів [31, 33, 34]. Такі підходи дозволяють описувати обчислення в термінах операцій над потоками даних, графів або функціональних композицій, залишаючи синтезатору деталізацію конвеєризації та розміщення на самій схемі.

Використання вищеописаних методів та їх комбінування дозволяє:

- скоротити час розробки та перевірки вузлів цифрової обробки сигналів;
- забезпечити зв’язок між алгоритмічним описом та апаратною реалізацією;
- спростити масштабування і параметризацію вузлів ЦОС у складі SoC FPGA;
- інтегрувати апаратні прискорювачі у ширший спектр вбудованих операційних систем та edge-інфраструктури.

Проте такий підхід все ще вимагає розуміння специфіки ПЛІС та особливості взаємодії HPS, характеристик синтезаторів, процесів та інтерфейсів операційний систем, як і архітектурних особливостей центральних процесорів. Результат ж їх комбінування дозволяє будувати гнучні, адаптивні та комплексні системи

цифрової обробки сигналів, використовувати стандартизовані підходи та вже наявні розширення - що в свою чергу спрощує сам процес розробки [11]. Саме тому подальша задача дисертаційної роботи формулюється на перетині цих підходів.

1.5. Роль вбудованої ОС (Linux) у системах на базі SoC FPGA

У базовому випадку, керування ПЛІС може здійснюватися безпосередньо за допомогою bare-metal прошивок, де код на процесорному ядрі виконується без проміжного шару ОС. Такий підхід є прийнятним для вузькоспеціалізованих пристроїв із фіксованим функціоналом, однак погано масштабується для систем цифрової обробки сигналів, котрі потребують:

- гнучкого налаштування параметрів алгоритмів (коефіцієнти фільтрів, вікна перетворень, пороги спрацювання);
- підтримки кількох режимів роботи та сценаріїв обробки;
- віддаленого доступу, логування, оновлення та діагностики;
- інтеграції з зовнішніми сервісами та інфраструктурою (мережа, сховища даних, хмара).

Базове середовище для виконання подібних задач надається операційною системою в процесорній підсистемі. Наявність ОС дозволяє розділити апаратно-залежну частину (драйвери, низькорівневі бібліотеки) та прикладний програмний рівень, на відміну від статичної конфігурації HPS, котра міцно зав’язана із конкретною версією прошивки. З цієї причини оновлення алгоритмів, додавання додаткових функцій обробки та адаптація системи до нових сценаріїв застосування стають більш простими завдяки цьому, без необхідності повного перепроєктування HDL-дизайну.

Наявність мережевого стеку, цілого переліку стандартизованих інструментів для роботи з периферією та підтримки більшості мов програмування робить ОС на базі ядра Linux одним із основних варіантів поряд із операційними системами реального часу (RTOS). У роботі [12] показано, що використання вбудованих Linux-систем на периферійних комп’ютерах дозволяє поєднати локальну обробку даних з можливістю інтеграції в більші розподілені системи, що дозволяє знайти оптимальний баланс між продуктивністю, гнучкістю та витратами на розробку. Аналогічний метод використовується в проектах, у яких система на кристалі FPGA використовується як обчислювальний прискорювач, а також для збору та

попередньої обробки даних [17–19].

У процесі синтезу вузлів ЦОС засобами високорівневого проєктування вбудована ОС також забезпечує контекст на рівні алгоритмів та керування для апаратних модулів. HLS-ядра, написані за допомогою C/C++, виконують операції низького рівня над потоками даних, тоді як вбудоване програмне забезпечення Linux відповідає за параметризацію, послідовність викликів, об'єднання результатів і подальшу передачу. Це дозволяє розробникам розглядати систему як ієрархію сервісів, оскільки апаратні блоки ЦОС виступають як прискорювачі, керовані високорівневими API.

У типових системах, на кристалі типу SoC FPGA, архітектура включає в себе наявність як мінімум двох взаємодіючих доменів. Це включає домен програмованої логіки (FPGA), котрий містить апаратні ядра цифрової обробки сигналів, буфери, контролери інтерфейсів і допоміжну логіка, та домен процесорної підсистеми, який містить прикладне програмне забезпечення та операційну систему Linux. Узагальнену шарову структуру такої системи наведено на Рис. 1.6.

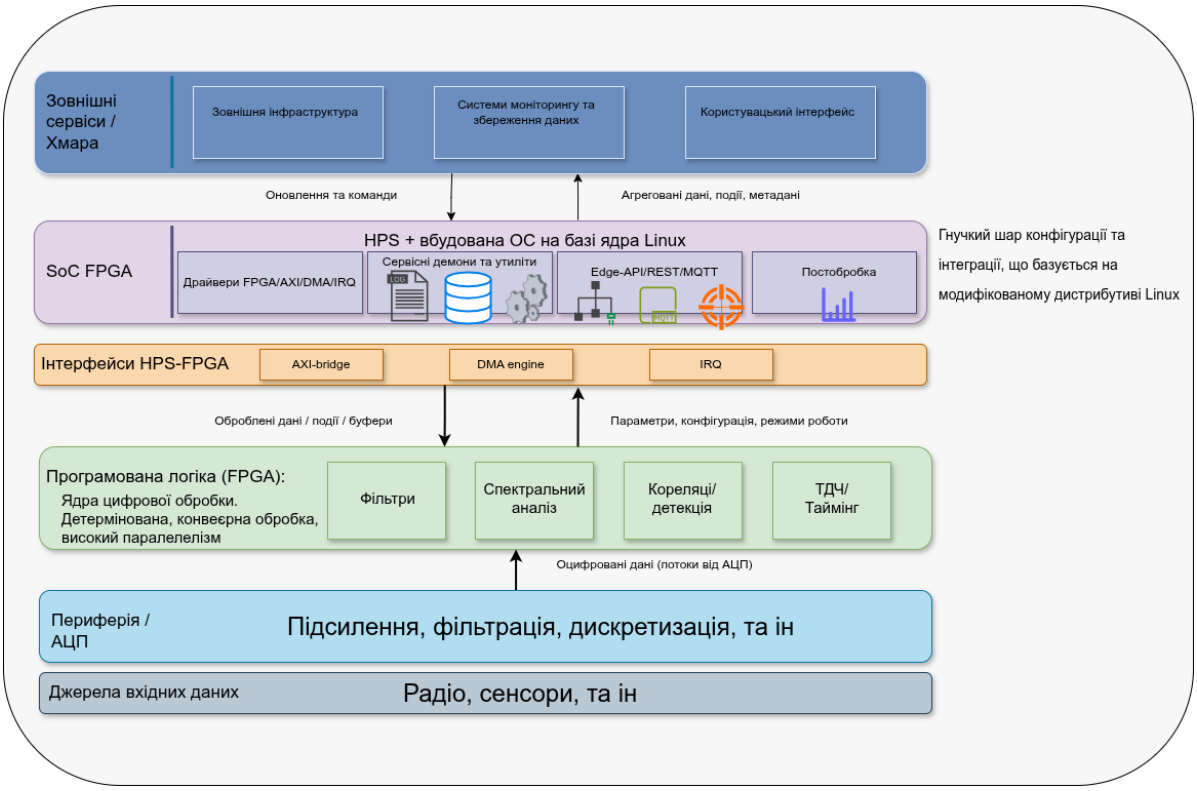


Рис. 1.6: Шарова архітектура системи на базі SoC FPGA: апаратні ядра ЦОС у FPGA, проміжний шар драйверів і інтерфейсів (AXI, DMA, IRQ) та сервісно-прикладний рівень під керуванням Linux на HPS.

Зазвичай контролери прямого доступу до пам'яті (DMA), лінії переривань

і стандартні шини типу AXI використовують для зв’язку між цими доменами. У випадку платформ Cyclone V на основі ядра HPS може отримати доступ до частини адресного простору FPGA за допомогою спеціалізованих мостів. Це дозволяє використовувати регістри керування для конфігурації апаратних модулів, а також організовувати обмін великих кількостей даних за допомогою загальної пам’яті.

У роботі [20] показано, що створення чітко визначеної програмно-апаратної архітектури зв’язку між HPS та FPGA є важливим для досягнення низьких затримок і передбачуваної поведінки системи. Високошвидкісні операції вимірювання та попередня обробка, такі як часо-цифрове перетворення, виконуються апаратною частиною. З іншого боку, Linux на HPS відповідає за конфігурацію параметрів вимірювань, збір результатів, агрегування даних і передачу даних до верхніх рівнів системи, таких як локальна база даних, мережеві сервіси та візуалізація даних.

За допомогою програмного забезпечення Linux дозволяє реалізувати багаторівневу модель доступу до апаратних модулів FPGA: елементи на рівні ядра можна декларувати за допомогою драйверів, які інкапсулюють роботу з AXI-, DMA- та IRQ-інтерфейсами; елементи на рівні користувацького простору можна реалізовувати за допомогою бібліотек, утиліт командного рядка, демон-служб і веб-інтерфейсів. Системи на базі SoC FPGA, які використовують Linux-середовище для візуалізації, керування та інтеграції результатів обробки сигналів [8, 16, 18], продемонстрували аналогічний підхід. У цих рішеннях апаратні вузли ЦОС працюють як прискорювачі, тоді як HPS під керуванням Linux виконує «оркестрацію» обчислень, маршрутизує потоки даних між вузлами та зовнішніми інтерфейсами, а також визначає механізми оновлення та довгострокового супроводу пристроїв [25].

Така архітектура має вирішальне значення для задач цифрової обробки сигналів, оскільки вона дозволяє реалізувати жорстке розмежування між логікою керування (*Control Plane*) та апаратно-прискореними трактами даних (*Data Plane*) [25]. Це дозволяє впроваджувати складніші механізми керування, такі як адаптивна зміна параметрів, автоматичне перемикання режимів, віддалене оновлення та журналювання подій у систему без критичного впливу на детермінізм поведінки та пропускну здатність.

У цьому контексті природним розвитком є підхід кордонних (edge-)

обчислень, коли значна частина обробки виконується «на місці» - безпосередньо на периферійному вузлі, а до центральної системи передаються вже агреговані або попередньо інтерпретовані результати. Такий підхід дає змогу зменшити мережевий трафік, підвищити стійкість до нестабільних каналів зв'язку та забезпечити автономність роботи вузла з подальшою синхронізацією. Linux природно підтримує ці сценарії завдяки файловим системам з журналюванням, можливості кешування результатів, сервісам, що працюють у режимі демона, та інтеграції з брокерами повідомлень, такими як MQTT. У системах цифрової обробки сигналів це дозволяє зберігати локальні спектрограми або часові ряди, проводити попередню класифікацію подій і надсилати вже агреговані результати до центральної системи замість «сирих» даних. Подібний підхід був продемонстрований у [18], де середовище Linux служить платформою для управління режимами роботи сканера, зберігання отриманих даних та їх подальшого аналізу.Узагальнену послідовність проходження даних від сенсорів до хмарної інфраструктури наведено на Рис. 1.7.

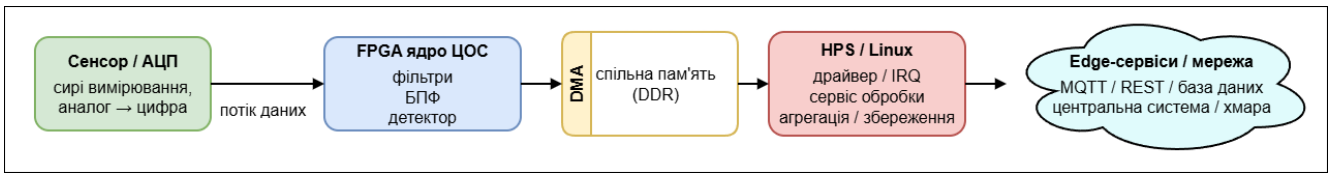


Рис. 1.7: Послідовність обробки даних у кордонній системі ЦОС.

Добре розвинена екосистема інструментів розробки Linux є ще однією перевагою. Наявність компіляторів, систем управління пакетами, бібліотек для роботи з мережевими протоколами, баз даних, графічних інтерфейсів, пакетів для наукових обчислень і систем управління залженостями значно знижує поріг для інтеграції із зовнішніми системами та загалом спрощує процес розробки. Фактично, частина ланцюжка обробки може бути виконана лише за допомогою програмного забезпечення в просторі користувача. Приклади включають додавання модулів машинного навчання для класифікації сигналів і виявлення аномалій або постобробку результатів вимірювань. Середовище Linux також спрощує створення сервісної логіки: командних інтерфейсів, REST-сервісів чи веб-панелей для конфігурації та моніторингу системи [16,18]. Розробник може використовувати відомі інструменти та вже існуючи практити їх застосування, зосереджуючись на інтеграції із апаратними модулями, фокусуючись не тільки на реалізації базових механізмів взаємодії з користувачем та мережею.

Linux загалом добре поєднується з високорівневим проєктуванням. У роботі [16]розглядається метод створення спеціалізованої вбудованої операційної системи на базі Linux для системного ядра FPGA. Цей метод покликаний зробити взаємодію апаратних прискорювачів із програмним шаром простішою. Крім того, процес обміну даними можна конвеєризувати за допомогою стандартизованих інструментів інтеграції зовнішніх елементів, що дозволяють трактувати зовнішнійй прискорювач як ще один стандартний вузол, використовуючи сокети, інтерфейси або файловий формат. Використання звичайного Linux-стеку на HPS дозволяє керувати оновленням програмного забезпечення, контролювати версії компонентів і організовувати дані вже існуючих інструментів, а також уникнути необхідності впровадження власної логіки. Оновлення можна отримати за допомогою традиційних пакетних менеджерів, зовнішнього репозиторія або за допомогою системних автономних процесів, таких як демони. Журналювання подій за допомогою systemd або syslog дозволяє фіксувати логіку прикладних сервісів і низькорівневі системні помилки. Профайлінг, трасування та дистанційний доступ через SSH значно спрощують діагностику поведінки систем у польових умовах, де фізичний доступ до обладнання обмежений і вимоги до надійності підвищені. Використання повноцінної ОС Linux у складі системного ядра FPGA має низку обмежень, не зважаючи на значні переваги. Навіть спеціалізовані дистрибутиви потребують ресурсів, окремої підтримки та вчасних оновлень. Це може бути важливим для вузьких систем, у котрих необхідно реалізувати ресурсоємні апаратні ядра ЦОС паралельно з операційною системою. Також, користувацький простір Linux має обмежену детермінованість часових характеристик; навіть при використанні real-time-розширень жорстко детермінована обробка, як правило, переноситься у FPGA, тоді як ОС відповідає за керування та сервісні задачі [20, 21]. Третій фактор полягає в тому, що універсальні дистрибутиви, такі як Debian або Ubuntu, містять значно більше компонентів, ніж необхідні для конкретної системи цифрової обробки сигналів. Збільшення кількості пакетів збільшує час завантаження, ускладнює управління версіями бібліотек і збільшує ймовірність помилок або вразливостей. Таким чином, у роботах [12, 16] пропонується метод побудови власних вбудованих дистрибутивів, у якому ядро Linux, базовий користувацький простір і набір бібліотек створюються з нуля відповідно до конкретної апаратної платформи та набору задач. Такий підхід зводить образ системи до найбільш необхідних

компонентів, що зменшує накладні витрати та полегшує супровід. Проте в такому випадку особливо важлива архітектура поділу ролей між HPS та FPGA. На відміну від апаратної частини, Linux надає конфігурацію, управління потоками даних, мережеву взаємодію та інтеграцію з іншими сервісами, а апаратна частина відповідає за обробку сигналів. Використання високорівневого проєктування призводить до створення конфігурованих модулів, у яких алгоритмічний рівень опису (наприклад, HLS-ядро) пов'язаний із чітко визначеним програмним інтерфейсом, який використовується в вбудованій операційній системі.

Хоча Linux не є способом для вирішення всіх проблем у проектуванні вбудованих систем, його використання у поєднанні з програмованою логікою на кристалі SoC FPGA дозволяє оптимально поєднувати гнучкість, масштабованість і апаратну ефективність. Це, у свою чергу, спонукає до створення спеціалізованих архітектурних підходів і вбудованих дистрибутивів, які поєднують переваги високорівневих мов, HLS і Linux-орієнтованих стеків у межах однієї платформи для цифрової обробки сигналів. Сформований таким чином погляд на роль вбудованої ОС на базі ядра Linux у системах SoC FPGA визначає подальший напрямок дисертаційної роботи. У наступних розділах увага буде зосереджена на побудові спеціалізованих вбудованих дистрибутивів, інтегрованих із інструментами високорівневого проєктування, а також на проєктуванні й дослідженні конкретних вузлів цифрової обробки сигналів, що використовують описані вище архітектурні принципи.

1.6. Взаємодія процесор–ПЛІС (HPS–FPGA, AXI, DMA, IRQ)

У системах на кристалі типу SoC FPGA цифрова обробка сигналів майже завжди реалізується як кооперація двох доменів: програмованої логіки (FPGA), де розміщуються конвеєрні ядра ЦОС, та процесорної підсистеми (HPS), яка виконує роль керуючого й інтеграційного шару. Від характеристик зв'язку між цими доменами безпосередньо залежать пропускна здатність, затримки та передбачуваність усієї системи [6,11,28]. Для задач ЦОС, де необхідно одночасно забезпечити високу швидкість потоку даних, гнучку конфігурацію алгоритмів і інтеграцію з верхніми рівнями програмного забезпечення, питання організації HPS–FPGA взаємодії є ключовим.

Основним методом взаємодії між HPS та логікою FPGA є звичайні інтерфейси сімейства AXI. Це стосується більшості сучасних систем на кристалі FPGA, таких

як рішення на основі Intel Cyclone V та подібних рішень інших виробників. Зазвичай виділяють принаймні два класи шин: легковагові регістрові інтерфейси (AXI4-Lite) для конфігурації й моніторингу та повноцінні AXI4/AXI-Stream інтерфейси для постійного обміну даними [6, 9]. У типовій архітектурі на AXI4-Lite виносяться регістри керування ядрами ЦОС (коефіцієнти фільтрів, розмір вікна перетворення, пороги виявлення подій, режими роботи), а на широкій AXI4/AXI-Stream-шині реалізується транспортування блоків даних між АЦП/ЦАП, буферами у логіці FPGA та підсистемою пам'яті HPS. Узагальнену схему такої взаємодії наведено на Рис. 1.8.

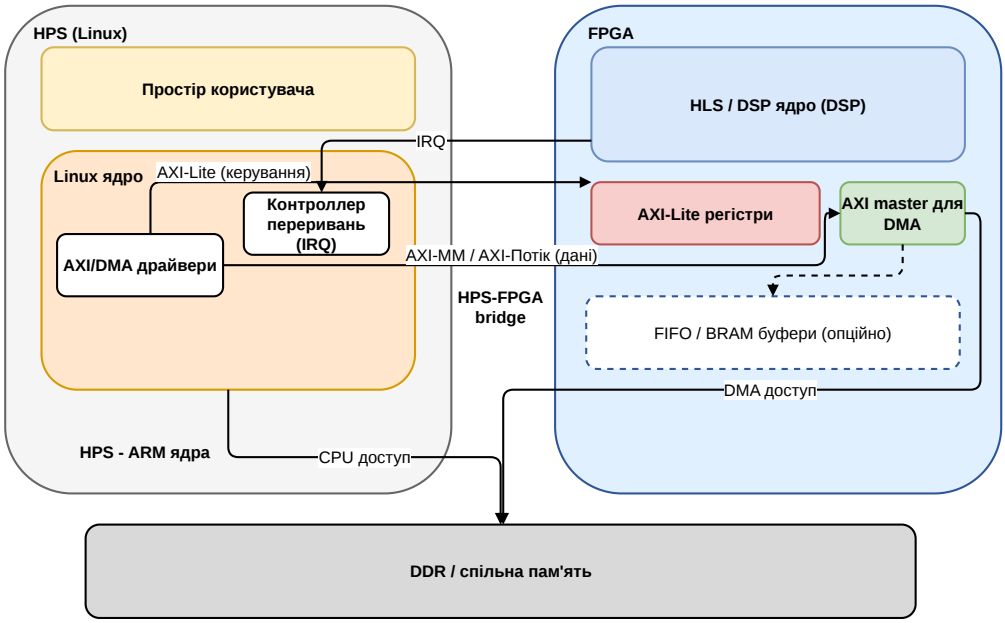


Рис. 1.8: Узагальнена схема взаємодії між HPS та FPGA в системах на базі SoC FPGA.

З погляду програмної моделі можна виділити три базові патерни функціонування:

- регістровий доступ до конфігураційних та статусних регістрів ядер ЦОС (через AXI4-Lite);
- потокова передача зразків сигналу через AXI-Stream між апаратними блоками та периферією;
- блочний обмін великими обсягами даних між пам'яттю HPS та буферами в логіці FPGA за допомогою механізмів DMA.

У прототипах, описаних у [8, 16, 22, 27], саме така комбінація дозволила рознести функції керування, потокової обробки й архівації даних, на різні шари

відповідальності, цим уникнувши надмірного навантаження на процесорні ядра HPS.

Важливим є розподіл обміну та даними та обробку команд на різні шини. Навіть відносно прості завдання, такі як зчитування спектрів або часових фрагментів сигналу, швидко вичерпують обчислювальний ресурс процесора і створюють проблеми, якщо обмін масивами даних здійснюється за допомогою традиційних операцій зчитування/запису HPS. Контролери DMA (Direct memory access) дозволяють перемістити процес передачі в паралельний потік. Для цього система конфігурує дескриптори, адреси, розміри блоків і режими, а потім обмін даними відбувається асинхронно відносно коду простору користувача. Це дозволяє інкапсулювати введення-виведення даних на окремому системному рівні, та дозволяє йому залишатися незалежним від інших частин системи. Цей підхід продемонстровано, зокрема, в системі перетворення часу в цифровий формат на базі SoC FPGA, де HPS під управлінням Linux конфігурує параметри вимірювання, а частина FPGA і DMA забезпечують захоплення і передачу часових міток з мінімальним залученням CPU [20, 21].

Слід зазначити, що проблеми з затримкою займають значну частину часу, витраченого на розробку системи. Загальна затримка передачі даних залежить від часу доступу до пам’яті, часу проходження мосту AXI, часу обробки контролера DMA та накладних витрат операційної системи, включаючи обробку переривань та перемикання контексту, серед іншого. Затримка перевищує пропускну здатність для невеликих блоків даних, оскільки система витрачає більше часу на ініціалізацію, ніж на фактичну передачу даних. Умовну часову структуру таких затримок у конвеєрі «FPGA - DMA - Linux» показано на Рис. 1.9.

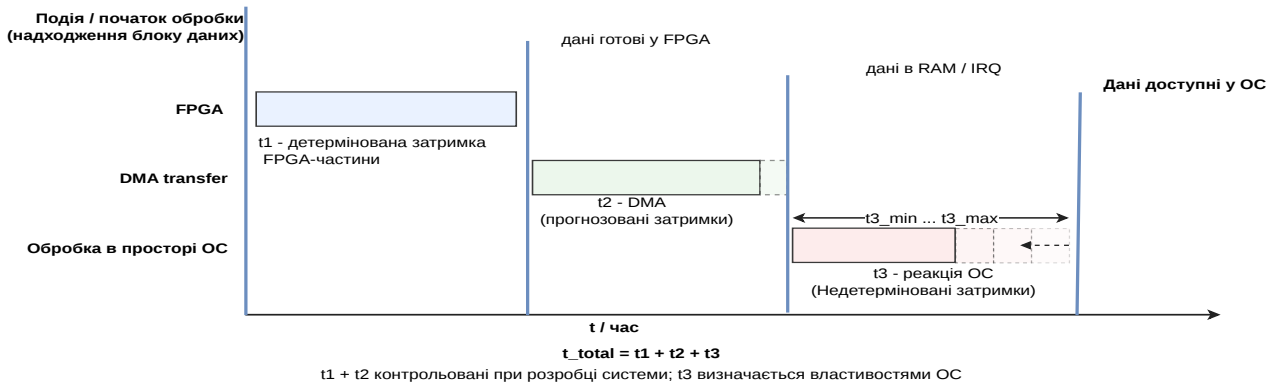


Рис. 1.9: Часова структура затримок у тракті «FPGA → DMA → Linux»

Таким чином ми можемо розділити загальну затримку всієї системи на три її

складові:

1. детермінована затримка FPGA частини (може бути обрахована синтетичними інструментами) (t_1),
2. майже детермінована передача через DMA (залежить від типу, характеристик та налаштувань регістрів та способу імплементації DMA) (t_2)
3. та варіативний час реакції ОС Linux (t_3), що залежить від планувальника ОС та механізмів копіювання даних (zero-copy vs copy).

Це особливо помітно в ситуаціях, коли невеликі конфігураційні структури потрібно часто оновлювати або передавати короткі фрагменти сигналів, наприклад, у режимах вимірювання з тригером або при роботі з потоками подій [11]. На практиці це вимагає використання кільцевих буферів і більших блоків даних у пам'яті HPS для пакетної обробки. Це використовується в системах оцінки продуктивності та сканування радіосигналів [18, 24].

Переривання (IRQ), які є окремим показником складності, — це сигнали, що вказують на завершення передачі DMA, надходження нового блоку даних або виникнення помилки. Ядро FPGA створює запит на переривання для контролера переривань. Потім керування передається менеджеру в ядрі програми керування або базовій прошивці. Незважаючи на те, що такий метод дозволяє реагувати на події, системи з високою частотою подій схильні до необґрунтованого зростання кількості таких запитів, що збільшує навантаження на операційну систему та призводить до зашумленого списку задач. Отже, для високошвидкісних RTOS-рішень часто використовується комбінований підхід. Переривання забезпечують грубе сигналізування, як-от «буфер заповнений», а періодичне опитування регістрів або дескрипторів забезпечує детальний моніторинг стану. Це зменшує кількість IRQ і покращує передбачуваність поведінки системи [20, 22]. У випадку Linux на HPS необхідно враховувати підсистему кешу, а також потенційну несумісність між кешем процесора та буферами, до яких звертається DMA. У більшості випадків розробник повинен або вручну очищати та інвалідизувати кеш для певних областей пам'яті, або використовувати спеціальні механізми відображення пам'яті, що обходять кеш, або покладатися на інтерфейси ядра, які виконують цю роботу за них.У прототипах, орієнтованих на довготривалі вимірювання та сканування спектрів, це питання проявляється у вигляді важковідтворюваних помилок: «старі» дані

в кеші, розсинхронізація станів дескрипторів DMA та фактичного заповнення буферів, випадкові артефакти в потоці даних [18, 19, 24]. Саме тому на етапі проєктування протоколів обміну на рівні HPS–FPGA важливо чітко визначити, які області пам’яті є кешованими, а які - ні, та забезпечити однозначний контракт щодо їх використання.

Високорівневе проєктування ще більше ускладнює картину, але водночас дає додаткові можливості. У випадку HLS-інструментів верхньорівнева функція з аргументами, що описують потоки даних і регістри керування, транслюється у набір AXI4-Lite та AXI-Stream/AXI4-портів. Таким чином, взаємодія HPS–FPGA, описана на алгоритмічному рівні, зрештою все одно реалізується у вигляді стандартних апаратних інтерфейсів, а від вибору директив, структури циклів і розміщення буферів залежить як пропускна здатність, так і затримка [10, 29, 30]. У задачах ЦОС, де критичним є не лише обчислювальне ядро, а й спосіб подачі даних, це означає, що розробник має враховувати ширину шини, схеми доступу до пам’яті, глибину FIFO та режим роботи DMA ще на етапі алгоритмічного опису. У роботах [12, 16–18] запропоновано підхід, у якому HPS під керуванням спеціалізованої вбудованої Linux-системи виконує роль «оркестратора» потоків даних та керує FPGA-прискорювачами через чітко визначені програмно-апаратні контракти. Апаратні модулі ЦОС розглядаються як окремі вузли обробки, що мають стандартизовані AXI-інтерфейси для конфігурації й обміну даними, тоді як Linux забезпечує маршрутизацію потоків, збереження результатів, інтеграцію з мережевими протоколами та сервісами верхнього рівня. Така ідея показує, що правильна організація зв’язку дозволяє одночасно досягати високої продуктивності та зберігати гнучкість, коли змінюються сценарії використання та алгоритми.

Загалом, взаємодія процесор-ПЛІС у системі на основі кристалу SoC FPGA можна розглядати як окремий контрольний шар всієї платформи. Верхня межа швидкодії системи, її масштабованість за кількістю каналів і здатність інтегруватися в більш складні архітектури країв залежить від того, як організовані AXI-інтерфейси, DMA-потоки, переривання, буфери та моделі пам’яті. Для цифрової обробки сигналів це означає, що проєктування HPS–FPGA зв’язку має розглядатися не як другорядне завдання після розробки алгоритмічних ядер, а як рівноправна частина загальної архітектури рішення, яка визначає, наскільки повно вдасться використати потенціал програмованої логіки і процесорної

підсистеми.

1.7. Модифіковані Linux-системи для SoC FPGA

Як було показано у попередніх підрозділах, поєднання програмованої логіки та процесорної підсистеми в SoC FPGA природно веде до використання вбудованих операційних систем, зокрема Linux, як основи для керування, конфігурації та інтеграції апаратних прискорювачів цифрової обробки сигналів [8, 12]. Тим не менш, метод створення програмного середовища значною мірою впливає на складність розробки, гнучкість системи та швидкість ітерацій під час експериментів. Використання універсальних дистрибутивів, таких як Debian і Ubuntu,або ж використання спеціалізованих фреймворків, таких як Yocto Project, Buildroot, PetaLinux і OpenWrt, задля створення мініфікованих дистрибутивів на основі ядра Linux є основними підходами до проектування [11, 13, 14]. Система на базі SoC FPGA, яка використовує повноцінну операційну систему засновану на ядрі Linux, дозволяє переходити від монолітних, жорстко зафіксованих конфігурацій до більш адаптивних, програмно керованих архітектур. Але в реальних вбудованих програмах рідко використовується «звичайний» дистрибутив загального призначення, здебільшого мова йде про Linux-системи, які були модифіковані, мінімізовані або спеціалізовані, аби підлаштуватися під певну апаратну платформу, набір периферії та вимоги до цифрової обробки сигналів [12–15].

У класичному випадку виробник SoC FPGA (Intel/Altera, Xilinx тощо) постачає референсні образи ОС та інструменти для їх побудови. Такі рішення зазвичай базуються на популярних вбудованих фреймворках на кшталт Yocto Project або Buildroot і містять попередньо налаштоване ядро, завантажувач, файлову систему, а також базовий набір драйверів і утиліт для роботи з HPS–FPGA мостами, DMA та периферією [15, 35]. Подібні інструменти є хорошим початком, але вони є більш загальними і містять багато елементів, які не потрібні для конкретного завдання ЦОС. Це збільшує час завантаження, ускладнює підтримку і означає, що кожен проект потребує власної оптимізації.

Фреймворки на кшталт Yocto або Buildroot вирішують цю проблему шляхом побудови відтворюваних, описаних у вигляді сценаріїв (рецептів) та конфігураційних шарів (layers), вбудованих дистрибутивів [13, 15, 35]. У таких системах розробник має можливість задати:

- цільову архітектуру (ARM Cortex-A, RISC-V тощо) та специфіку SoC,
- конфігурацію ядра Linux, включно з підтримкою HPS–FPGA мостів, AXI, DMA, IRQ,
- склад rootfs, набір бібліотек і користувацьких програм,
- механізми оновлення та структуру розділів пам‘яті.

Головною перевагою подібних систем є відтворюваність збірки, чітке декларування залежностей та наявність готових метаданих (рецептів) для зовнішніх пакетів пз. Вони є де-факто стандартом у промислових проєктах, де важливими є довгострокова підтримка, контроль версій і можливість збірки для різних апаратних платформ [11, 14]. Узагальнене порівняння основних підходів до побудови вбудованих Linux-систем для SoC FPGA наведено на Рис. 1.10.

Критерій	Універсальний дистрибутив (Debian / Ubuntu на HPS)	Фреймворки (Yocto / Buildroot / PetaLinux)	Повністю спеціалізована збірка (full custom підхід)
Час освоєння інструментів	низький	високий	середній
Час збірки / ітерації	низький	високий (повна перебудова)	низький-середній (швидкі цикли)
Розмір образу	великий	оптимізований	малий-середній
Контроль над залежностями	обмежений (пакетний менеджер)	дуже хороший	максимальний (ручний)
Відтворюваність	середня	висока	середня (залежить від реалізації)
Інтеграція з SoC FPGA BSP	середня	висока	висока (керовано вручну)
Гнучкість під edge-сценарії	посередньо	хороша (ціною складності)	висока (система під задачу)

Рис. 1.10: Порівняння підходів до побудови вбудованих Linux-систем для SoC FPGA

З цього ми можемо коротко характеризувати всі три підходи подібним чином:

1. Debian/Ubuntu: класичний інструмент, але вимогливий до доступних ресурсів і обмежений контроль залежностей
2. Yocto/Buildroot: висока відтворюваність та контроль системи, комплексність в розробці та підтримці
3. Full custom: спеціалізація для кордонних систем: високий контроль, швидкі ітерація, доступність всіх компонентів для модифікацій

Разом з тим, використання комплексних мета-фреймворків у дослідницьких задачах накладає суттєві обмеження архітектурного характеру. Головним недоліком є надлишкова абстракція: шарова модель (layers) та автоматизована генерація рецептів приховують прямі зв’язки між апаратними налаштуваннями (handoff-файлами) та конфігурацією ядра [11]. Це ускладнює верифікацію

низькорівневих контрактів взаємодії HPS–FPGA, оскільки розробник оперує не прямими налаштуваннями драйверів, а метаданими високого рівня. [11, 13]. Крім того, інструменти на кшталт Yocto орієнтовані на створення статичних промислових образів (firmware), де пріоритетом є незмінність, відкитаючи необхідність високої адаптивності. У контексті даних досліджень, котрі вимагають частої зміни параметрів ядра, профілів DMA та логіки FPGA для пошуку оптимальних характеристик, така інерційність інструментарію унеможливлює швидку ітеративну валідацію гіпотез. Тому для забезпечення повного контролю над залежностями та гарантування бітової відтворюваності (reproducibility) результатів доцільнішим є застосування методик прямого (Full Custom) формування системи. Використовуючи класичні інструменти крос-компіляції, звичайні дистрибутиви Linux і власні сценарії автоматизації, розробник може створити мінімальний образ вбудованої ОС за допомогою так званого повністю персоналізованого підходу [13, 14]. У цьому випадку:

- ядро Linux конфігурується й збирається окремо, з урахуванням специфіки SoC FPGA (HPS–FPGA мости, контролери пам‘яті, DMA, периферія)
- базовий користувацький простір будується або на основі готового дистрибутива (наприклад, Arch Linux чи Debian), або як мінімальний набір утиліт (BusyBox, udev, інструменти мережі)
- додаткові пакети (Python, бібліотеки для роботи з БД, мережеві стеки, MQTT, системи логування) додаються в rootfs у міру потреби.

Роботи [12, 16] пропагують цю конкретну методологію для SoC FPGA на базі Cyclone V. Спеціалізований дистрибутив Linux побудований з нуля з використанням мінімального набору компонентів, необхідних для полегшення взаємодії між HPS і FPGA, забезпечення мережевого підключення, виконання операцій з базою даних і надання послуг з обробки сигналів. Linux служить «тонким» сервісним шаром над апаратними ядрами SoC у проектах модульної системи збору та аналізу даних на периферії [17] та офлайн-систем візуалізації радіосигналів [18].

Головною перевагою повністю індивідуального підходу є те, що вся збірка є зрозумілою та легко керованою [12]. Розробник має прямий контроль над наступним: конфігурація ядра (модулі, опції реального часу, драйвери взаємодії HPS–FPGA), структура rootfs та набір залежностей, механізми ініціалізації (systemd, busybox init, індивідуальні скрипти), та методи загального оновлення

системи. При такому підході зміна конфігурації зазвичай вимагає лише редагування конфігураційних файлів і скриптів збірки, уникаючи інтеграції десятків рецептів і шарів у один сценарій, як це, зазвичай, робиться у важких фреймворках. Це значно прискорює ітерації, що дуже важливо у галузі цифрової обробки сигналів і тестування нових архітектур вузлів SoC.

Ще однією перевагою є можливість взаємодіяти із вже існуючими екосистемами повноцінних дистрибутивів. Наприклад, [16] розкриває те, як використовувати пакети Arch Linux для створення простору користувача на SoC FPGA. Це дозволяє поєднати невеликий розмір вбудованої системи з доступом до бібліотек та інструментів, створених для розробки, обробки даних та взаємодії з мережею. Отже, однакова система може функціонувати як платформа для локальної операційної системи, так і як вузол периферійних обчислень, інтегрований у більш розгалужену інфраструктуру [12].

Якщо порівняти повністю настроюваний метод із фреймворками, такими як Yocto або Buildroot, можна побачити кілька ключових відмінностей. Важкі фреймворки забезпечують високу відтворюваність з точки зору швидкості ітерації, але часто вимагають повної перебудови значної частини системи навіть для невеликих змін у конфігурації. Коли розробники змінюють спосіб роботи обчислень, налаштувань драйверів, наборів послуг або механізмів обміну даними в прототипах ОС на SoC FPGA, це займає більше часу.

У спеціалізованому підході можна обмежитися переконфігурацією ядра або оновленням окремих компонентів rootfs [13, 15]. Фреймворки пропонують широкий спектр готових рецептів і BSP, але вони також мають власний спосіб організації проектів, який не завжди може добре виконувати певний спектр задач та несе з собою певні обмеження.

У роботах [16, 17] підкреслюється, що повний контроль над структурою дистрибутива полегшує додавання конкретних інструментів (таких як пропрієтарні інструменти логування, брокери повідомлень і служби для обробки сигналів), що дозволяє системі розвиватися природним чином, не прив‘язуючись до конкретної версії BSP. Узагальнений опис вищеописаного підходу до побудови спеціалізованої вбудованої Linux-системи для SoC FPGA, котрий використаний у роботах [12, 16], наведено на Рис. 1.11.

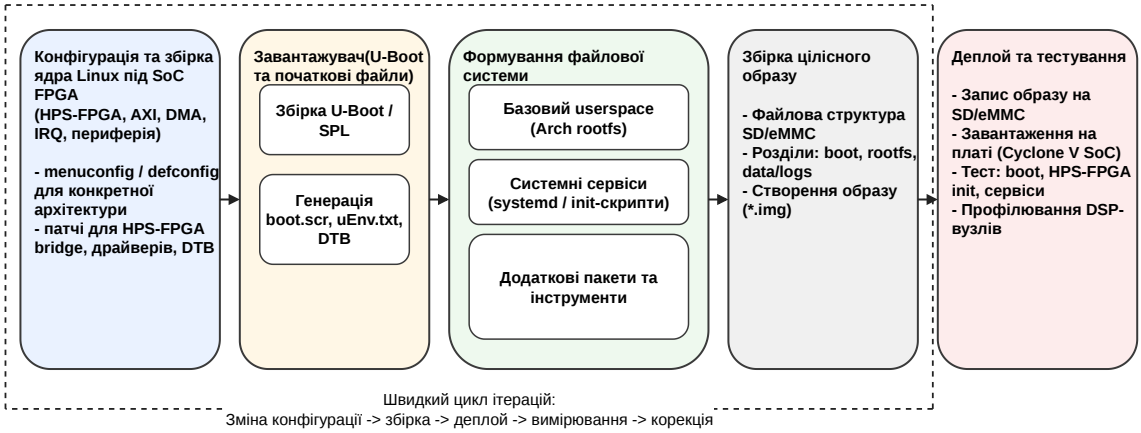


Рис. 1.11: Процес побудови спеціалізованої вбудованої Linux-системи

Проте індивідуальний підхід несе з собою і певні негативні аспекти, котрі вимагають явних рішень під час розробки. Оскільки не існує централізованої системи рецептів, розробник несе пряму відповідальність за забезпечення можливості повторення збірки, контролю залежностей та оновлення безпеки. Для довгострокових промислових проектів це може бути проблемою. Але використовуючи загальні репозиторії зовнішніх пакетів можна уникнути проблеми повторюваності, а розбиття процесу збірки на декілька кроків дозволяє імплементовувати як цільове кешування, або ж уникнути необхідності перебудовувати всю систему при модифікації одного елементу [11, 14]. Іншим фактором є необхідність розуміння таких процесів як крос-компіляція, завантаження ядра, конфігурація файлової системи та інтеграція служб. Однак ці знання необхідні, в будь-якому випадку, для правильного синтезу та функціонування складних систем на базі SoC FPGA.

У галузі цифрової обробки сигналів модифікована система Linux на HPS служить «шаром управління та інтеграції» між апаратними модулями та рештою елементів. Апаратні ядра, синтезовані як вручну, так і засобами HLS, реалізують часово критичні частини алгоритмів (фільтри, БПФ, кореляційні обчислення, обробку подій), тоді як Linux відповідає за:

- конфігурацію параметрів цих ядер через регістрові інтерфейси, AXI та DMA,
- маршрутизацію потоків даних між ядрами, локальними буферами та мережевими сервісами,
- зберігання, агрегацію та попередній аналіз результатів,
- інтеграцію телекомунікаційного вузла у розподілені мережі передачі

даних (DTN) для збереження оброблених сигналів [12, 18].

Особливості дистрибутива дозволяють досягти прийнятного компромісу між ресурсними обмеженнями SoC FPGA та потребою у гнучкості та масштабованості на рівні системи завдяки мінімальному набору служб, чітко визначених залежностей і наявності лише необхідних драйверів і бібліотек.

Таким чином, незалежно від того, чи є вони повністю спеціалізованими чи базуються на спеціалізованих фреймворках, модифіковані Linux-системи є важливою частиною архітектури цифрової обробки сигналів на базі систем на основі FPGA.

1.8. Постановка проблеми та формулювання задач дослідження

Як показав аналіз існуючих методів створення цифрових систем обробки сигналів на базі ПЛІС і систем на основі FPGA на чільному кристалі, окремі компоненти цього стеку уже досить добре опрацьовані. З іншого боку, класичні HDL-підходи (VHDL/Verilog) пропонують високі показники продуктивності та точний контроль апаратної реалізації. Однак синтез, верифікація та супровід складних архітектур вимагають багато зусиль і часу, особливо коли мова йде про багатоканальні, параметризовані ЦОС-комплекси [6, 9, 28]. Однак високорівневий синтез (HLS) і доменно-специфічні мови дозволяють описувати алгоритми у термінах C/C++ або потокових обчислень, генерувати RTL-код і підвищувати рівень абстракції. Однак якість результатів і передбачуваність часових параметрів значною мірою залежать від стилю кодування та можливостей конкретного інструменту [10, 11, 29–31, 33, 34].

Для систем на кристалі типу SoC FPGA ситуація ускладнюється тим, що апаратні модулі цифрової обробки сигналів мають працювати разом із вбудованою операційною системою та процесорною підсистемою, яка зазвичай базується на ядрах Linux [7, 8]. AXI-мості, контролери DMA та механізми переривань забезпечують зв’язок між доменами HPS і FPGA, що вимагає чітко визначеної програмно-апаратної архітектури [20, 21, 32]. Linux у цьому випадку виконує роль шару керування та інтеграції: конфігурує апаратні ядра, організовує потоки даних, забезпечує мережеву взаємодію, зберігання та аналіз результатів [12, 16]. Однак узгодження вимог до продуктивності, детермінізму та гнучкості у такій гібридній системі є нетривіальним завданням.

Існують промислові фреймворки, такі як Yocto Project, Buildroot, PetaLinux

і OpenWrt, які пропонують формалізовані підходи до побудови мінімізованих образів Linux. Ці фреймворки пропонують чіткі рецепти, BSP-шари та механізми крос-компіляції [13, 13, 14, 14, 15, 35]. Вони ідеально підходять для проєктів, які потребують підтримки та відтворюваності протягом тривалого періоду часу. Однак вони досить складні для початкового освоєння та швидких ітерацій, оскільки навіть незначні зміни у конфігурації або наборі пакетів можуть призвести до тривалих повних перебудов образу.

Альтернативою є повністю персоналізований підхід до побудови вбудованої ОС, коли ядро Linux, користувацький простір і необхідні бібліотеки збираються із використанням класичних інструментів крос-компіляції та уже наявних дистрибутивів (Arch Linux, Debian тощо), але без залучення важких фреймворків [13, 14]. Роботи [12, 16–18, 22, 24] демонструють, що такий підхід дозволяє створювати спеціалізовані, легковагові Linux-системи для SoC FPGA, оптимізовані під конкретні задачі обробки сигналів та периферійних обчислень, із коротким циклом збірки та можливістю оперативного розширення функціоналу. Водночас відповідальність за відтворюваність збірки, контроль залежностей та безпекових оновлень повністю переходить до розробника, що може бути проблемою для масштабних та довготривалих проєктів.

Окрему групу вимог формують сценарії периферійних (edge) обчислень, у яких вузол на базі SoC FPGA повинен:

- здійснювати збір «сирих» даних (радіосигнали, часові ряди, вимірювання) із високою роздільною здатністю та продуктивністю;
- виконувати локальну попередню обробку (фільтрацію, спектральний аналіз, детекцію подій, агрегацію);
- працювати автономно при нестабільному або відсутньому зв’язку з центральною системою, із можливістю подальшої синхронізації;
- надавати інтерфейси для віддаленого керування, оновлення, діагностики й інтеграції з іншими сервісами [7, 8, 12, 17].

У таких системах апаратна частина (FPGA) є відповідальною за часово критичні обчислення, тоді як процесорна підсистема з Linux реалізує шар керування, мережеву взаємодію та сервісну логіку. Проте у літературі, попри наявність численних оглядів HLS, FPGA та SoC-платформ [8, 10, 11, 26], бракує цілісних методик, які поєднують:

- високорівневе описання алгоритмів ЦОС;

- стандартизовані механізми взаємодії HPS–FPGA (AXI, DMA, IRQ) саме в контексті edge-сценаріїв;
- спеціалізовані, але легко відтворювані вбудовані Linux-дистрибутиви;
- орієнтацію на експериментальну та прототипну розробку з коротким часом ітерації.

Таким чином, **основну науково-практичну проблему** можна сформулювати як відсутність узгодженої архітектурної та технологічної основи для побудови вузлів цифрової обробки сигналів на базі SoC FPGA, які одночасно:

- ефективно використовують апаратні ресурси ПЛІС для реалізації часово критичних операцій;
- інтегруються з вбудованою ОС Linux як шаром керування, візуалізації та мережевої взаємодії;
- підтримують гнучку параметризацію, масштабування й розвиток алгоритмів ЦОС без повного перепроєктування апаратної частини;
- зберігають відтворюваність збірки та можливість адаптації до різних edge-сценаріїв.

При цьому значна частина існуючих рішень або зосереджена на HDL-рівні та не розглядає повноцінну інтеграцію з ОС, або використовує важкі фреймворки вбудованого Linux і не оптимізована для швидкої прототипізації, або ж описує окремі прикладні системи без узагальненої методики, яку можна перенести на інші задачі ЦОС [11, 27, 32]. **Метою даної дисертаційної роботи** є розробка та обґрунтування методики синтезу вузлів цифрової обробки сигналів на основі SoC FPGA із застосуванням високорівневого проєктування та спеціалізованих вбудованих Linux-систем, орієнтованих на кордонні (edge) застосування. Концептуально взаємозв’язок трьох основних доменів, на перетині яких формулюється мета дисертаційної роботи (апаратна цифрова обробка сигналів на SoC FPGA, високорівневе проєктування та вбудовані Linux-системи для кордонних обчислень), узагальнено подано на Рис. 1.12.

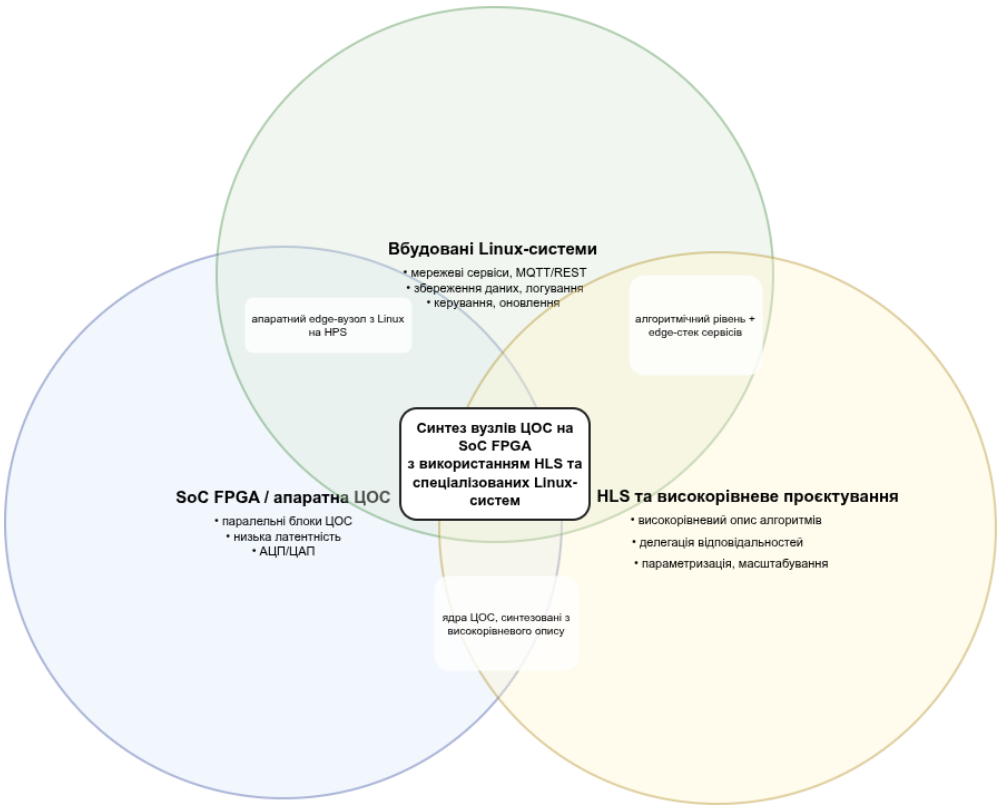


Рис. 1.12: Діаграма перетину доменів

У межах поставленої мети передбачається створення цілісної програмно-апаратної архітектури, що інтегрує апаратні прискорювачі на ПЛІС, процесорну підсистему з ОС Linux та мережеву інфраструктуру, а також демонстрація її ефективності на прикладі конкретного застосування (офлайн-система обробки та візуалізації радіосигналу у форматі сканера спектра) [18, 19, 24]. Таким чином, **основну науково-практичну проблему** можна сформулювати як відсутність узгодженої архітектурно-технологічної основи для побудови вузлів цифрової обробки сигналів на базі SoC FPGA, яка б одночасно забезпечувала:

- ефективне використання апаратних ресурсів ПЛІС для реалізації часо-критичних операцій;
- тісну інтеграцію з вбудованою ОС Linux як шаром керування, візуалізації та мережевої взаємодії;
- гнучку параметризацію, масштабування й розвиток алгоритмів ЦОС без повного перепроєктування апаратної частини;
- відтворюваність процесу збірки та адаптованість до різних edge-сценаріїв.

Для реалізації зазначеної мети в роботі ставляться такі основні завдання:

1. Виконати систематизований огляд і аналіз сучасних підходів до побудови

систем цифрової обробки сигналів на базі ПЛІС та SoC FPGA, зокрема із використанням HDL, HLS і доменно-специфічних мов, а також вбудованих Linux-дистрибутивів і edge-архітектур [8, 10–12].

2. Розробити узагальнену архітектуру вузла на базі SoC FPGA для задач ЦОС, яка включає апаратні ядра обробки сигналів на ПЛІС, підсистему зв'язку HPS–FPGA (AXI, DMA, IRQ), та програмний шар на базі Linux для керування, конфігурації та інтеграції з зовнішніми сервісами [20, 21, 32].
3. Розробити методику формування модифікованих вбудованих Linux-систем для SoC FPGA з використанням повністю індивідуалізованого підходу до збірки, який забезпечує мінімальний, але масштабований набір компонентів, підтримку мережевих стеків, сервісів зберігання та аналізу даних, а також інтеграцію з апаратними прискорювачами [12, 16, 17].
4. Спроєктувати та реалізувати набір модулів цифрової обробки сигналів (фільтрація, спектральний аналіз, виявлення подій тощо) із застосуванням класичних HDL та/або засобів високорівневого синтезу, орієнтуючись на їх подальшу інтеграцію з вбудованою ОС і edge-сценаріями.
5. Розробити експериментальний прототип вузла периферійних (edge-) обчислень на базі SoC FPGA з модифікованою Linux-системою, у якому реалізувати запропоновану архітектуру та продемонструвати типові сценарії використання на прикладі офлайн-сканера радіосигналів, акцентуючи можливість масштабування під інші задачі ЦОС [18, 24].
6. Провести експериментальну оцінку запропонованого підходу за показниками продуктивності, гнучкості конфігурації, масштабованості та трудомісткості розробки, а також порівняти його з альтернативними варіантами, що базуються на універсальних дистрибутивах Linux або важких фреймворках вбудованої ОС.

Виконання зазначених задач дозволить не лише обґрунтувати доцільність використання модифікованих Linux-систем у поєднанні з SoC FPGA для периферійної цифрової обробки сигналів, але й запропонувати практично орієнтовану методику побудови таких вузлів, яка може бути перенесена на різні класи задач, вийшовши за межі окремого прикладу сканера спектра.

1.9. Висновки до розділу 1

У першому розділі здійснено систематичний аналіз сучасних методів побудови **вузлів цифрової обробки сигналів** на базі систем на кристалі типу SoC FPGA в контексті розвитку технологій граничних обчислень (*Edge Computing*) у телекомунікаційних мережах. Показано, що еволюція апаратних платформ зумовлює перехід до гетерогенних архітектур, які є оптимальною базою для реалізації вузлів доступу та попередньої обробки сигналів. Поєднання детермінованого паралелізму програмованої логіки (FPGA) та гнучкості процесорної підсистеми (HPS) дозволяє виконувати первинну обробку сигнальної інформації безпосередньо у точці прийому, суттєво зменшуючи навантаження на магістральні канали зв'язку.

Водночас виявлено, що класичний маршрут проєктування на рівні регістрових передач (HDL) характеризується високою складністю та низькою швидкістю адаптації до нових стандартів зв'язку, тоді як застосування засобів високорівневого синтезу (*HLS*) дозволяє значно підвищити рівень абстракції. Аналіз програмно-апаратної взаємодії довів, що використання операційної системи загального призначення (Linux) є необхідним для забезпечення мережевого стеку та маршрутизації, проте планувальник ОС вносить непередбачувані затримки (фазовий джитер), які руйнують детермінованість процесів управління радіотрактом у реальному часі. Встановлено, що надійність та пропускна здатність вузла критично залежать від реалізації протоколів внутрішньокристальної взаємодії (AXI, DMA, IRQ). Некоректний розподіл буферної пам'яті між процесором та апаратною логікою призводить до порушення цілісності сигнальних кадрів та виникнення прихованих збоїв в умовах пікових навантажень трафіком.

Крім того, виявлено недоліки існуючих інструментальних засобів побудови програмного забезпечення: важкі системи автоматизації (наприклад, фреймворк Yocto) забезпечують відтворюваність збірки, але є надто інертними для динамічних телекомунікаційних задач *Edge*-рівня. Це обґрунтовує доцільність застосування методики повної спеціалізації (*Full Custom*), яка дозволяє формувати адаптивні керуючі середовища з мінімальними накладними витратами.

На основі проведеного аналізу та виявлених архітектурних обмежень сформульовано мету дисертаційної роботи, яка полягає у підвищенні

ефективності синтезу **вузлів цифрової обробки сигналів** на базі SoC FPGA. Доведено, що для розв’язання цієї задачі синтез повинен базуватися на розробці гібридного маршруту проєктування. Такий підхід має поєднувати системну високорівневу інтеграцію інфраструктури керування з мікроархітектурною RTL-оптимізацією та апаратним розвантаженням (*Hardware Offloading*) критичних обчислювальних трактів, практичну реалізацію та дослідження яких наведено у наступних розділах.

РОЗДІЛ 2. АРХІТЕКТУРА ТА МЕТОДИКА ПОБУДОВИ СПЕЦІАЛІЗОВАНОЇ ПРОГРАМНО-АПАРАТНОЇ ПЛАТФОРМИ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ НА БАЗІ SOC FPGA

2.1. Вбудована операційна система на базі Linux для SoC FPGA як керуюче середовище реконфігурованих вузлів цифрової обробки сигналів

Розділ 1 обґрунтував доцільність застосування SoC FPGA як платформи для реконфігурованих вузлів цифрового оброблення сигналів у сучасних периферійних (edge) системах та показав, що ключова практична складність лежить не лише у синтезі апаратних ядер, але й у побудові цілісної програмно-апаратної системи з визначеними контрактами взаємодії між доменами HPS та FPGA. У цьому розділі розглядається роль спеціалізованої вбудованої операційної системи на базі Linux як «інтеграційного шару», що забезпечує конфігурацію апаратних модулів, організацію потоків даних, локальне зберігання та мережеву взаємодію, а також спостережуваність і відтворюваність експериментів. Основні положення розділу спираються на результати робіт [12, 16–18, 20, 21].

2.1.1. Загальні вимоги до вбудованої ОС для SoC FPGA в DSP/edge-задачах

Вбудована операційна система для реконфігурованих вузлів оброблення сигналів на базі SoC FPGA повинна одночасно задовольняти дві вимоги:

1. забезпечувати ефективну та передбачувану взаємодію з апаратними модулями FPGA (конфігурація, обмін даними, синхронізація)
2. надавати практичний програмний стек для побудови прикладної edge-системи (мережа, зберігання, логування, інтеграція сервісів).

У роботі [12] показано, що використання програмованої логіки для детермінованої обробки та Linux-середовища для інтеграції та керування дозволяє розглядати систему на основі FPGA як автономний крайовий (edge) вузол, який може бути включений у більшу інфраструктуру.

2.1.2. Функціональні вимоги

Віднесемо до основних функціональних вимог ОС для вузлів ЦОС на SoC FPGA:

1. Підтримка апаратної взаємодії HPS-FPGA вимагає, щоб операційна система забезпечувала механізми доступу до буферів даних і регістрів керування модулів FPGA. Крім того, операційна система повинна забезпечувати інфраструктуру для передачі блоків даних за найнижчими накладними витратами, як правило, за допомогою DMA.
2. Практичні архітектурні рішення щодо організації такого обміну (розподіл ролей між апаратною частиною, доступом через міст HPS–FPGA та програмним рівнем керування) розглядаються у [20, 21].
3. Мережевий стек і протоколи інтеграції. ОС має забезпечити стандартні засоби віддаленого доступу, керування та передачі результатів: SSH/HTTP(S), а також легкі протоколи повідомлень для подій і телеметрії (типово MQTT). Для edge-підходу важливо, щоб мережевий шар не був єдиною точкою функціонування системи, а лише розширював автономні можливості пристрою [12, 17].
4. Локальне зберігання та керування даними. Системи ЦОС, зокрема у радіомоніторингових сценаріях, генерують значні обсяги даних (IQ-потоки, спектральні оцінки, метадані вимірювань), що потребує локального буферування і збереження з можливістю подальшого аналізу та відтворення. Офлайн-режим як вимога реальної експлуатації та обґрунтування необхідності локального контуру зберігання підкреслено на прикладі прикладної системи [18, 19].
5. Логування, діагностика та спостережуваність. ОС має забезпечити журналювання системних і прикладних подій, збір метрик, а також інструменти діагностики, достатні для віддаленого супроводу (особливо для автономних edge-вузлів). У модульних підходах важливо, щоб спостережуваність була властивістю платформи, оскільки вбудовування щоразу механізмів спостереження та контролю здатне негативно вплинути як на адаптивність так і на характеристи системи [17].
6. Підтримка оновлень та керування конфігураціями. Оскільки edge-вузол часто експлуатується поза лабораторним середовищем, ОС має підтримувати відтворюваний спосіб розгортання, оновлення компонентів

та керування конфігураціями (ядро, модулі, сервіси, параметри прикладних задач). Методика побудови кастомної системи як спосіб зменшити складність ітерацій та підвищити адаптивність обґрунтовується у [16].

2.1.3. Нефункціональні вимоги

Окрім функціональності, визначальними є нефункціональні обмеження та критерії якості:

1. Обмеження ресурсів. Типові SoC FPGA-платформи мають обмеження за RAM та постійною пам’яттю, що вимагає мінімізації ядра, rootfs і сервісів без втрати критичних можливостей (мережа, драйвери, зберігання) [16].
2. Передбачуваність і затримка. Практичні вимірювання для SoC-платформ показують, що варіативність затримок (jitter) у циклі реального часу суттєво залежить від способу обробки подій у програмному домені, а для зменшення непередбачуваності доцільно застосовувати фіксацію потоків на ядрах (core affinity), polling у критичних ділянках та DMA-орієнтований транспорт даних [36]. Таким чином, цільова архітектура повинна чітко розрізняти «детермінований конвеєр» і «гнучку оркестрацію» [12].
3. Надійність і самодостатність. Стійкість до руйнування мережі, відновлення після збоїв, збереження даних у локальному контурі та можливість працювати в автономному режимі є важливими для edge-вузлів. Цей режим експлуатації характерний для офлайн-систем обробки сигналів [18].
4. Відтворення та підтримка. Необхідно, щоб платформа могла відтворювати програмне середовище та конфігурацію експерименту, включаючи версії ядра та модулів, параметри обробки, формати даних і конфігурації сервісів. Методика створення спеціалізованого дистрибутива та контроль конфігурацій [16] прямо пов’язані з цією потребою.

Відповідність ключових вимог підсистемам реалізації узагальнено в табл. 2.1.

Таблиця 2.1

Матриця відповідності вимог підсистемам реалізації (SoC FPGA + Linux)

Вимога	FPGA	Ядро	ОС	Інфрастр.
F1. HPS–FPGA: регістри/буфери/DMA	+	+		
F2. Мережа: SSH/HTTP(S), MQTT тощо			+	+
F3. Локальне зберігання та відтворення		+	+	+
F4. Логування, діагностика, спостережуваність			+	+
F5. Оновлення, конфігурації, відтворюване розгортання	+	+	+	+
NF1. Обмеження ресурсів (RAM/flash)		+	+	
NF2. Затримка та передбачуваність (Jitter)	+	+	+	
NF3. Надійність і автономність		+	+	+
NF4. Відтворюваність та супровідність		+	+	+

Для однозначної інтерпретації наведених у матриці вимог слід конкретизувати поняття автономності (**NF3**). У рамках запропонованої методики ця вимога формалізується не як абстрактна надійність, а як набір бінарних критеріїв: здатність системи автоматично відновлювати обчислювальний конвеєр після апаратного перезавантаження (за допомогою механізму watchdog) та продовжувати накопичення даних у локальному сховищі за повної відсутності мережевого з’єднання. Це відрізняє edge-вузол від класичного тонкого клієнта, який втрачає функціональність без зв’язку з вузлу керування.

Окремо варто визнати, що вимога низької затримки та передбачуваності (**NF2**) є принципово конфліктною з природою Linux як ОС загального призначення.

Навіть за умови фіксації частоти процесора та вимкнення енергозбереження, часові флуктуації можуть виникати через планувальник, обробку переривань, міграцію задач між ядрами, роботу сторінкового менеджера пам'яті та фонових сервісів. У контексті вузлів ЦОС на SoC FPGA ця суперечність пом'якшується тим, що критичний по часу конвеєр виноситься в FPGA і є детермінованим, тоді як Linux виконує функції конфігурації, оркестрації обчислень і доставки результатів. Отже, **NF2** у практичних системах доцільно трактувати як вимогу не до всієї платформи, а до межі «FPGA->DMA-> простір користувача ОС», де найбільший внесок у нестабільність формує саме програмний домен.

Загалом ж, вбудована Linux-система для SoC FPGA у задачах ЦОС повинна розглядатися не як допоміжний компонент, а як системоутворюючий шар, що формує контракти взаємодії між HPS і FPGA та надає edge-функції (мережа, зберігання, спостережуваність, оновлення).

2.2. Огляд підходів до побудови вбудованих Linux-систем для SoC FPGA

Побудова вбудованої Linux-системи для SoC FPGA на практиці майже завжди починається з вибору «опорного» підходу, тобто того, як саме формуються завантажувальні компоненти, ядро, rootfs та прикладний стек, і якою буде ціна ітерацій при подальших змінах. Для платформ класу Cyclone V характерне співіснування двох вимірів:

1. апаратного - де визначається топологія взаємодії HPS-FPGA, мостів, пам'яті та периферії,
2. програмного - де формуються драйвери, механізми доставки даних, служби керування та edge-інтеграції.

Intel у своїх рекомендаціях прямо підкреслює необхідність починати проєкт із визначення системної топології та використовувати її як основу для проєктування інтерфейсів HPS-FPGA [37]. У цьому сенсі вибір підходу є не лише питанням зручності збірки, а частиною архітектурного контракту між доменами HPS і FPGA.

Узагальнено можна виділити чотири поширені класи підходів:

1. Вендорний референтний стек (BSP/GSRD),
2. інструментальний шар Vendor SDK (SoC EDS),
3. метафреймворки збірки дистрибутивів (Yocto/Buildroot),
4. спеціалізована (full-custom) методика.

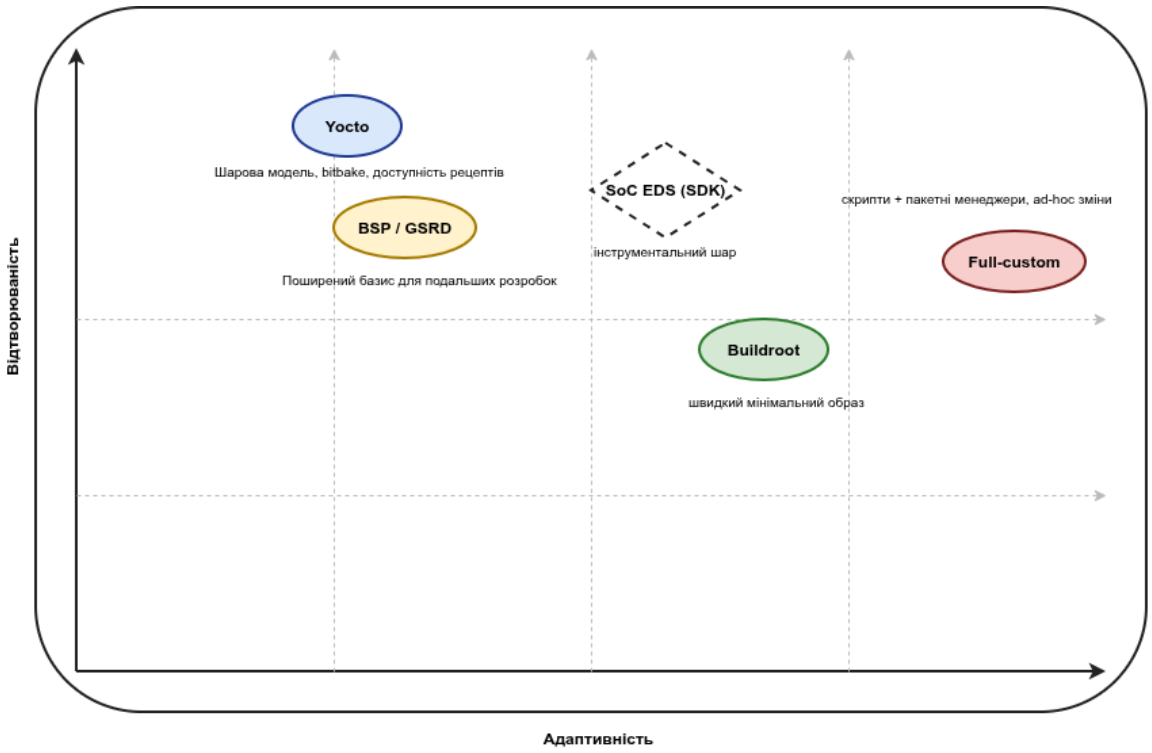


Рис. 2.1: Позиціювання підходів побудови вбудованих Linux-систем для SoC FPGA.

У межах цієї роботи ці підходи розглядаються не як взаємовиключні, а як такі, що займають різні позиції на шкалі «відтворюваність/контроль - швидкість ітерацій/адаптивність», і можуть комбінуватись залежно від стадії життєвого циклу вузла ЦОС. Узагальнене позиціювання розглянутих підходів за цими критеріями наведено на Рис. 2.1. Окремо слід зазначити компроміс між простотою оновлення (притаманною **Yocto**) та стабільністю часових характеристик. У системах реального часу та ЦОС неконтрольоване оновлення системних бібліотек (наприклад, *glibc* або ядра) через пакетні менеджери несе ризик порушення часових контрактів (timing violations). Тому методика Full Custom свідомо орієнтована на побудову систем із фіксованим станом (*Immutable Infrastructure*), де будь-яка зміна версій компонентів валідується як новий інженерний реліз, уникаючи автоматичних фонових оновлень.

2.2.1. Вендорний базис: BSP та Golden System Reference Design (GSRD).

Для SoC FPGA-платформ типовим стартовим «якорем» є Golden System Reference Design, який постачається як перевірена опорна конфігурація апаратно-програмної системи. У документації Intel GSRD описується як відомо-працездатний (known good) базовий проєкт, що демонструє коректне

використання ресурсів і HPS, і FPGA, та прямо рекомендується як базова точка для нових розробок із подальшою модифікацією під прикладні потреби [38]. Важливо, що GSRD є цілісним комплектом компонентів: апаратний референтний дизайн (GHRD/Quartus-проєкт), референтний завантажувач (U-Boot), референтний Linux BSP і прикладні демонстраційні програми [38]. Така зв'язність корисна саме для SoC FPGA, оскільки зменшує ризик помилок на етапі первинного синтезу, коли дрібні невідповідності (наприклад, у конфігурації HPS, мапі адрес або device tree) можуть бути сприйняті як проблеми з простору ОС, хоча фактично мають апаратне походження.

Практична цінність GSRD у контексті цієї роботи полягає в тому, що він задає мінімально необхідний набір контрактів для коректного співіснування доменів. Зокрема, потік розробки орієнтований на використання артефактів передачі (handoff), котрі описують конфігурацію HPS і системи в цілому. У керівництві SoC EDS підкреслюється роль системних описів (зокрема .sopcinfo) як джерела даних для генерації device tree, що далі визначає, як ядро Linux взаємодіє із апаратними компонентами [39]. Тобто, навіть, якщо кінцева ОС будується іншими засобами, вендорний референтний стек зручний як верифікована «опорна точка істини» для перевірки коректності апаратно-програмного зв'язку.

2.2.2. Vendor SDK (SoC EDS) як інструментальний шар.

Паралельно з GSRD існує клас інструментів, які формують середовище низькорівневої розробки: Intel SoC FPGA Embedded Development Suite. Його роль полягає в наданні утиліт і бібліотек, корисних для портування ОС, розробки драйверів і апаратного налагодження. Ключовим елементом тут є HWLIB, який містить SoCAL - шар абстракції регістрів, що забезпечує прямий доступ і керування регістрами HPS у межах адресного простору [40]. З погляду архітектури вузлів ЦОС це означає, що навіть поза Linux-драйверами існує стандартизована база для низькорівневих операцій (контроль периферії, діагностика, первинне тестування), але її застосування вимагає розуміння режимів виконання та безпекових обмежень, оскільки HWLIB розрахований на роботу в привілейованому середовищі [40]. У контексті дисертації SoC EDS важливий як частина технологічного фундаменту: він задає типовий «інженерний шлях» від опису апаратної системи до працездатного програмного оточення, а також формує понятійний апарат для опису контрактів HPS-FPGA.

2.2.3. Yocto як підхід до відтворюваної синтезу дистрибутивів.

Yocto Project є одним із найпоширеніших промислових підходів для побудови вбудованих Linux-дистрибутивів у промислових сценаріях, де критичними є контроль складу, залежностей та відтворюваність. Його ключовою ідеєю є модель шарів: метадані організовуються у репозиторії-шари, що логічно розділяють апаратно-специфічну підтримку, конфігурації дистрибутива та прикладне ПЗ, підтримуючи повторне використання й кастомізацію [13, 41]. Внутрішнім механізмом збірки виступає BitBake, який описується як рушій виконання задач, що парсить рецепти та конфігураційні дані й формує дерево залежностей, визначаючи порядок компіляції та складання компонентів [41]. Таким чином, Yocto добре відповідає задачам, де потрібно формально закріпити склад образу та відтворювати його між командами/платформами/випусками.

Однак саме ці сильні сторони Yocto стають обмеженням для дослідницьких ітерацій у задачах синтезу вузлів ЦОС, коли одночасно еволюціонує і апаратна частина (ядра/інтерфейси), і прикладний стек (алгоритми, формати, інструменти аналізу). У такому режимі зростає ціна «малих змін», оскільки навіть локальна модифікація рецепту може запускати довгий конвеєр перебудови та перевірок. Тому в межах цієї дисертації Yocto розглядається як важливий референсний промисловий клас рішень, не як основний інструмент синтезу (що узгоджується з фокусом роботи на адаптивності й швидкості циклу у edge-сценаріях).

2.2.4. Buildroot як швидкий генератор мінімалістичних образів.

Buildroot є іншим полюсом серед поширених інструментів: він орієнтований на спрощення побудови повної Linux-системи засобами крос-компіляції та автоматизації складання компонентів [42]. Після конфігурації (через kconfig-подібний інтерфейс) Buildroot генерує файл `.config`, який містить всю конфігурацію і використовується верхньорівневим Makefile для побудови системи [42]. Ця унітарна модель конфігурації та загальна простота роблять Buildroot корисним для малих пристроїв і сценаріїв, де потрібен компактний образ і швидкий результат без складної ієрархії метаданих [42].

Разом з тим Buildroot має концептуальне обмеження, важливе саме для edge-вузлів ЦОС: типова система на Buildroot не передбачає «повноцінного» керування пакунками на цільовій платформі у стилі великих дистрибутивів, а отже оновлення або додавання компонентів зазвичай зводиться до перебудови

образу. Для дослідницьких прототипів, де потрібно часто інтегрувати бібліотеки обробки даних, утиліти діагностики, мережеві компоненти або експериментальні сервіси, така модель підвищує тертя. Тому Buildroot у дисертації доречно позиціонувати як «швидкий шлях до мінімального образу», але не як оптимальний базис для платформи, де критична гнучкість користувацького простору та швидкі on-device ітерації.

2.2.5. Full-custom підхід як динамічно-орієнтована методика (обраний базис).

У роботі [16] обґрунтовано full-custom підхід до формування вбудованої ОС для SoC FPGA, який полягає у явному (ручному, але формалізованому скриптами) конструюванні кожного шару: ядра, rootfs, набору сервісів та прикладних компонентів, з можливістю незалежно змінювати їх у межах одного життєвого циклу прототипу. Його відмінність від Yocto/Buildroot полягає в тому, що автоматизація реалізується як прозорий конвеєр зі скриптів і стандартних механізмів дистрибутивів (наприклад, debootstrap/pacstrap, менеджери пакунків, системні служби), не притримуючись моделі окремого метафреймворку зі складною моделлю метаданих. Практична мотивація цього вибору прямо відповідає цілям поставленої мети: у задачах синтезу вузлів ЦОС на SoC FPGA змінюються апаратні прискорювачі, їхні інтерфейси, драйвери і прикладні алгоритми, а отже критичним стає скорочення часу на їх інтеграцію.

Full-custom підхід у цьому розділі трактується як компроміс між двома підходами. З одного боку, він не заперечує цінність вендорного базису: GSRD доцільно використовувати як стартову перевірену конфігурацію для bring-up та валідації HPS-FPGA стику [38], а SoC EDS - як набір інструментів і бібліотек для низькорівневих робіт та розуміння апаратних ресурсів [40]. З іншого боку, full-custom дозволяє будувати користувацький простір як «живий» програмний стек, де можлива швидка інтеграція компонентів для edge-режимів (мережа, локальні БД, брокери повідомлень, логування), що напряму потрібно для систем, подібних до офлайн-сканера [18, 19] і модульної edge-платформи [17]. Ключовим наслідком є зміщення центру ваги: для прототипування вузлів ЦОС важливіше мати стабільні апаратно-програмні контракти та керовану еволюцію стека, ніж «ідеальну» формальну модель відтворюваності ціною довгих циклів перебудови.

Водночас full-custom підхід підвищує відповідальність розробника за

відтворюваність і супровідність. Тому він розглядається як методика, що повинна включати: фіксацію версій ядра/модулів, опис складу rootfs, контроль конфігурацій сервісів, та процедури розгортання, які відтворюють експериментальне середовище [16]. Таким чином, огляд підходів показує, що вендорний референтний стек (BSP/GSRD) і SDK (SoC EDS) формують перевірену основу апаратно-програмних контрактів, Yocto і Buildroot забезпечують стандартизовані механізми синтезу образів із різною ціною складності, а full-custom методика є найбільш придатною для динамічного прототипування вузлів ЦОС на SoC FPGA завдяки швидким ітераціям та гнучкості користувацького простору [16].

Хоча подальший виклад спирається на практичний досвід роботи з платформами Intel Cyclone V SoC FPGA, більшість описаних принципів — зокрема багатоступеневий bootchain, апаратно-залежні handoff-артефакти, декларативна роль Device Tree та контрактний характер конфігурації ядра Linux - є характерними для сучасних SoC FPGA загалом, включно з сімействами Xilinx Zynq/Zynq UltraScale+. Відмінності між вендорами полягають переважно у реалізації початкових стадій завантаження та інструментальних ланцюгах, тоді як логіка узгодження апаратного та програмного стеку залишається концептуально спільною.

2.3. Практичні аспекти побудови програмно-апаратної Linux-платформи цифрової обробки сигналів на базі SoC FPGA

Побудова програмно-апаратної платформи на базі SoC FPGA для задач цифрової обробки сигналів виходить за межі вибору дистрибутива або інструментарію збірки. На практиці визначальним є узгодження всіх етапів життєвого циклу системи — від первинного завантаження процесорної підсистеми до організації потоків даних між логікою FPGA та оперативною пам’яттю під керуванням ядра Linux. Для платформ сімейства Cyclone V така узгодженість обумовлена фіксованою архітектурою Hard Processor System (HPS), жорстко визначеним ланцюгом завантаження та контрактами взаємодії між апаратним і програмним рівнями [37, 43].

Процес запуску системи починається з виконання коду BootROM, який є незмінною частиною кристала та виконує мінімальну ініціалізацію процесорної підсистеми. На цьому етапі визначається джерело завантаження на основі

стану виводів BSEL, після чого зчитується образ початкового завантажувача — Preloader. Попри свою відносну компактність, Preloader є першою програмно-конфігурованою стадією завантаження і водночас однією з найбільш критичних точок стабільності всієї платформи. Його основним завданням є ініціалізація контролера SDRAM, налаштування таймінгів пам'яті та базових параметрів HPS, без яких подальше виконання ядра Linux є неможливим [37, 44]. Процес запуску Linux реалізується у вигляді багатоступеневого ланцюга завантаження, в якому кожна стадія виконує чітко визначену функцію та формує передумови для наступної (Рис. 2.2). Слід зазначити, що у термінології SoC FPGA Cyclone V роль вторинного завантажувача (**Secondary Loader /U-Boot SPL**) виконує компонент **Preloader**, який ініціалізує контролер SDRAM.

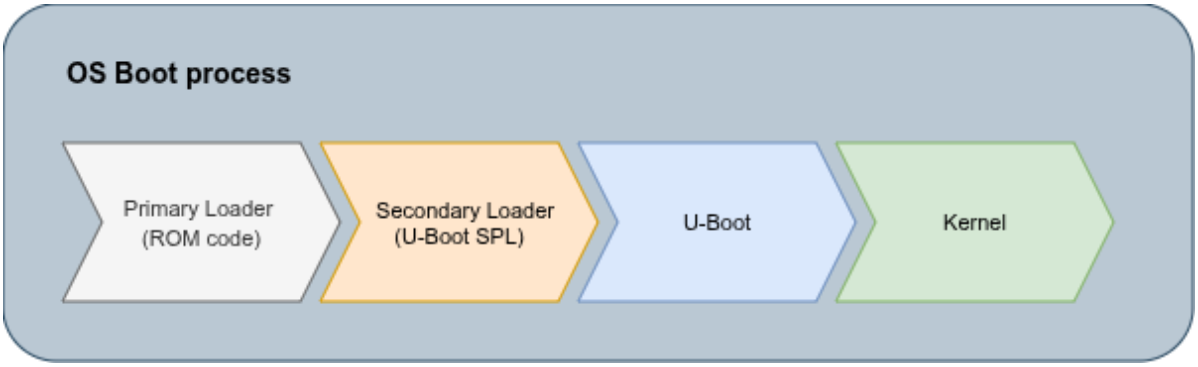


Рис. 2.2: Поетапний процес завантаження Linux-системи на базі SoC FPGA Cyclone V, від апаратного BootROM до запуску ядра Linux.

На практиці будь-які помилки або неточності на цьому етапі проявляються не як локальні програмні збої, а як повна відмова системи від завантаження. Це принципово відрізняє Preloader від наступних стадій, де більшість проблем може бути виявлена і діагностована засобами ядра або користувацького простору. З цієї причини вендорні рекомендації наполягають на використанні коректно сформованих handoff-артефактів, отриманих у процесі апаратного проєктування, та їх прямій інтеграції в інструментарій SoC EDS. Таким чином, Preloader виступає не допоміжним компонентом завантаження, а структурним вузлом стику між апаратним синтезом і програмною платформою.

Після успішної ініціалізації оперативної пам'яті керування передається завантажувачу U-Boot, який формує середовище для запуску ядра Linux. На цьому етапі відбувається завантаження дерева пристроїв (Device Tree), що виконує роль формалізованого опису апаратної конфігурації для операційної системи. Device Tree визначає адресний простір, структуру периферії, параметри

мостів HPS–FPGA та характеристики інтерфейсів пам’яті, виступаючи контрактом між апаратною частиною та ядром Linux [37]. Цей підхід дозволяє зберігати одне і те ж ядро Linux для різних апаратних конфігурацій, змінюючи лише опис апаратури.

Архітектура взаємодії між HPS і FPGA у Cyclone V базується на системі мостів AXI, кожен з яких має чітко визначене функціональне призначення. Легковажний міст HPS-to-FPGA (LWH2F) орієнтований на доступ до регістрів керування з мінімальною затримкою та використовується переважно для операцій control-plane, таких як налаштування параметрів обробки або керування станами апаратних модулів. Основний міст HPS-to-FPGA (H2F) підтримує пакетні транзакції та призначений для передачі великих обсягів даних, що є типовим для задач цифрової обробки сигналів [43]. Практично ця взаємодія розкладається на два незалежні контури — керування (control-plane) через регістровий доступ та передавання даних (data-plane) через DMA/буфери в SDRAM — що узагальнено на Рис. 2.3.

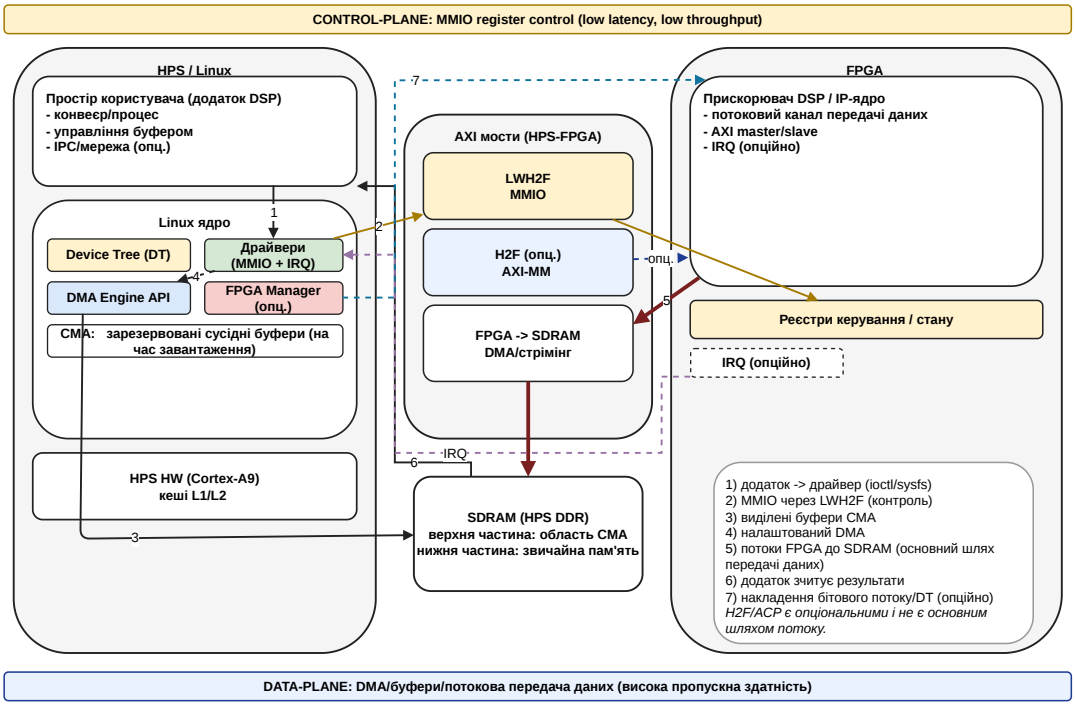


Рис. 2.3: Контракти між HPS й FPGA та організація control-plane і data-plane у Linux-керованій платформі Cyclone V SoC FPGA

Хоча опис у цьому підпункті прив’язано до Intel Cyclone V, загальна логіка зберігається і для інших SoC FPGA-платформ. Зокрема, у сімействах Xilinx Zynq/Zynq UltraScale+ роль первинного завантажувача виконує FSBL (First Stage

Boot Loader), який так само є критичною точкою ініціалізації пам’яті, тактування та базових периферійних блоків перед передачею керування до U-Boot або безпосередньо ядра Linux. Аналогічно, поділ на control-plane (MMIO/реєстри) та data-plane (DMA/буферні передачі через пам’ять) є типовим архітектурним шаблоном для Linux-керованих FPGA-прискорювачів незалежно від вендора.

Експериментальні дослідження показують, що прямий доступ FPGA до SDRAM забезпечує найвищу пропускну здатність, однак вимагає чіткого управління буферами та синхронізації з боку програмного забезпечення [32, 45]. Натомість використання порту когерентності прискорювача (ACP) спрощує взаємодію з кешованими структурами даних процесора, але обмежує обсяг доступної пам’яті та може стати вузьким місцем для потокових задач великого розміру.

Організація передачі даних між FPGA та пам’яттю під керуванням Linux реалізується за допомогою DMA-підсистеми ядра. Використання стандартного DMA Engine API дозволяє абстрагуватися від конкретної реалізації контролера та забезпечити переносимість драйверів між різними конфігураціями апаратури [46]. Разом з тим розробник повинен коректно враховувати режими когерентності пам’яті, виконувати мапінг буферів і синхронізацію кешу, оскільки помилки на цьому рівні призводять до важковідтворюваних збоїв у обробці сигналів. Для запобігання фрагментації пам’яті у довготривало працюючих системах застосовується механізм Contiguous Memory Allocator (CMA), який резервує області пам’яті ще на етапі завантаження ядра [32].

Важливим практичним аспектом є програмування FPGA безпосередньо з працюючої Linux-системи. Підсистема FPGA Manager у ядрі Linux надає уніфікований інтерфейс для завантаження бітових потоків незалежно від виробника та дозволяє інтегрувати процес реконфігурації у стандартний системний стек [47]. У поєднанні з накладаннями Device Tree це забезпечує можливість динамічного оновлення апаратних прискорювачів без перезавантаження операційної системи, що є особливо цінним для експериментальних та дослідницьких платформ.

Доступ до апаратних ресурсів з простору користувача є компромісом між швидкістю прототипування та системною надійністю. Механізми прямого мапування фізичних адрес або використання спеціалізованих інтерфейсів дозволяють спростити розробку прикладного програмного забезпечення, однак

вимагають суворої дисципліни щодо синхронізації та керування доступом. Практика побудови експериментальних систем збору та обробки сигналів показує, що такий підхід є прийнятним на етапі досліджень, тоді як для промислових систем доцільно застосовувати повноцінні драйвери ядра [48].

Отже, практична реалізація Linux-платформи на базі Cyclone V SoC FPGA є результатом узгодження низки інженерних рішень, кожне з яких безпосередньо впливає на стабільність, продуктивність і масштабованість вузлів цифрової обробки сигналів. Boot-ланцюг, handoff-артефакти, Device Tree, топологія мостів, механізми DMA та засоби програмування FPGA утворюють єдину систему, в якій помилки на ранніх етапах неминуче проявляються на рівні прикладних алгоритмів. Саме тому глибоке розуміння цих практичних аспектів є необхідною передумовою для побудови ефективних SoC-FPGA-платформ.

2.4. Апаратно-залежний шар Linux у системах на базі SoC FPGA

На відміну від класичних вбудованих платформ, у яких апаратний рівень значною мірою прихований за стандартизованими програмними абстракціями, системи на базі SoC FPGA формують тісно інтегрований програмно-апаратний стек. У таких системах операційна система Linux не ініціює апаратну конфігурацію, а працює з уже сформованою моделлю платформи, параметри якої визначаються на попередніх етапах проєктування та завантаження. Це зумовлює існування апаратно-залежного шару, що задає межі відповідальності програмного забезпечення та формує набір інваріантів, порушення яких призводить до некоректної або нестабільної роботи всієї системи. [14, 37]. Ключовою особливістю SoC FPGA є те, що низка критичних характеристик системи визначається ще до запуску ядра Linux. Ініціалізація контролера SDRAM, конфігурація інтерфейсів доступу до пам’яті, параметри шин AXI та базова топологія взаємодії між процесорною підсистемою (HPS) і логікою FPGA формуються на етапах початкового завантаження. Ядро Linux приймає ці параметри як коректні за означенням і не містить механізмів для їхньої валідації або повторної конфігурації. Унаслідок цього будь-які помилки, допущені на ранніх етапах, не проявляються у вигляді явних діагностичних повідомлень, а призводять до складних для відтворення збоїв у процесі роботи системи(див табл. 2.2).

Таблиця 2.2

Апаратно-залежні компоненти Linux-платформи на базі SoC FPGA Cyclone V

Компонент	**Джерело формування**	**Етап фіксації**	**Потенційні проблеми**
Ініціалізація SDRAM (Preloader)	SoC EDS, handoff-файли	Початкове завантаження	Неможливість запуску ядра Linux, зависання системи
Boot configuration (BSEL, boot source)	Апаратна конфігурація плати	Апаратний reset	Завантаження з неправильного джерела або відмова старту
Device Tree (DTB/DTBO)	Handoff + ручна модифікація	Завантаження ядра	Некоректна ініціалізація драйверів, конфлікти адрес
Топологія HPS–FPGA мостів	Апаратний дизайн (Quartus)	Статично	Неефективна передача даних, вузькі місця продуктивності
Пам’ять DMA / CMA	Параметри ядра Linux	Early boot	Фрагментація пам’яті, нестабільна робота DMA
Clock/reset tree	Апаратна логіка + DT	Статично	Порушення таймінгів, некоректна робота IP-ядер
ACP / некешований доступ	Архітектурний вибір	Проєктування	Зниження пропускної здатності або надмірні затримки

Особливу роль у цьому контексті відіграє початковий ланцюг завантаження, в якому формується апаратно-програмна база всієї платформи. Хоча ядро Linux запускається лише на фінальному етапі, саме попередні стадії визначають доступність пам’яті, коректність адресного простору та узгодженість апаратних інтерфейсів. З цієї причини апаратно-залежний шар не може розглядатися як допоміжний компонент — він фактично є фундаментом, на якому будується весь подальший програмний стек.

Процес завантаження починається з виконання коду BootROM, котрий є

незмінною частиною кристала і не доступний до модифікації поза вендором. BootROM визначає джерело завантаження та ініціює виконання початкового завантажувача (Preloader), формуючи єдину допустиму траєкторію переходу від апаратного скидання до програмного середовища [43]. На цьому етапі відсутні механізми діагностики, логування або відновлення, що принципово відрізняє його від наступних стадій запуску Linux. Будь-яка помилка конфігурації — зокрема, у параметрах пам’яті або тактування — призводить не до деградації функціональності, а до повної відмови системи від завантаження.

Особливо критичним елементом boot-ланцюга є Preloader, який виконує ініціалізацію SDRAM та базових ресурсів Hard Processor System. На практиці саме ця стадія є найбільш вразливою, оскільки поєднує результати апаратного проєктування з програмною конфігурацією [44]. Handoff-артефакти, сформовані в середовищі Quartus, визначають налаштування пінів, контролерів пам’яті та тактових доменів, і будь-яка неузгодженість між апаратною моделлю та Preloader призводить до некоректної роботи всієї системи. На відміну від помилок ядра або драйверів, такі збої практично не піддаються аналізу після факту, що робить апаратно-залежний шар ключовим джерелом ризику.

Після ініціалізації пам’яті управління передається завантажувачу U-Boot, який здійснює запуск ядра Linux разом з описом апаратної конфігурації у вигляді Device Tree. Device Tree у цьому контексті виступає формалізованим контрактом між апаратним дизайном та ядром операційної системи, проте важливою особливістю є відсутність будь-якої верифікації цього контракту з боку ядра [37]. Ядро Linux інтерпретує дерево пристроїв як коректний опис заліза, навіть якщо воно містить логічні або топологічні помилки. Таким чином, Device Tree є декларативною моделлю апаратури, що створює потенційно небезпечний розрив між апаратною реальністю та програмною абстракцією, оскільки не завжди декларації відповідають реальності. [49] Відповідність між етапами ланцюга завантаження та підсистемами Linux, на які вони безпосередньо впливають, наведено в табл. 2.3.

Таблиця 2.3

Взаємозв'язок етапів bootchain та підсистем Linux у SoC FPGA

Етап	**Компонент**	**Вплив на систему**
BootROM	Мінімальна ініціалізація HPS	Визначає джерело завантаження та неможливий до модифікації
Preloader (SPL)	SDRAM, pinmux, clocks	Формує базові умови для роботи ядра Linux
U-Boot	Завантаження DT та ядра	Передає опис апаратури ядру Linux
Kernel early boot	MMU, CMA, IRQ	Ініціалізація пам'яті та механізмів доступу до пристроїв
Runtime	Drivers, DMA, FPGA Manager	Робота прикладних алгоритмів ЦОС

Наведена відповідність підкреслює, що більшість параметрів, критичних для стабільної роботи Linux, фіксуються до моменту запуску ядра і не можуть бути скориговані на етапі виконання, то, зазвичай, наслідки помилок у Device Tree проявляються на значно пізніших етапах виконання системи. Неправильно описані області пам'яті, адресні діапазони або властивості когерентності можуть призводити до нестабільної роботи DMA, спорадичних збоїв у кешуванні або некоректної обробки переривань. Такі помилки часто можна сприйняти як дефекти алгоритмів цифрової обробки сигналів або під проблеми прикладного програмного забезпечення, ускладнюючи їх ідентифікацію [46, 50]. У результаті апаратно-залежний шар стає джерелом прихованих відмов, що не мають очевидного причинно-наслідкового зв'язку з кодом верхніх рівнів.

Передача великих обсягів даних між FPGA та оперативною пам'яттю в Linux-системах на базі SoC FPGA реалізується за допомогою підсистеми DMA. На відміну від звичайних драйверів введення-виведення, DMA-передачі безпосередньо залежать від коректності опису пам'яті, режимів когерентності та механізмів синхронізації кешу [46]. Помилки у виборі режиму (coherent чи streaming) або у мапінгу буферів призводять до неконсистентності даних, яка не фіксується стандартними засобами відлагодження. Таким чином, DMA у SoC FPGA слід розглядати не як оптимізацію продуктивності, а як системний компонент, що визначає коректність функціонування обчислювального конвеєра.

Окремим аспектом є управління фізично неперервною пам'яттю для потреб

DMA. Механізм Contiguous Memory Allocator (CMA) резервує області пам'яті ще на етапі завантаження ядра, що знову підкреслює критичну роль boot-time рішень для стабільності системи [32]. У довготривало працюючих системах фрагментація пам'яті без використання CMA може призводити до деградації пропускної здатності або неможливості ініціалізації нових DMA-передач, що особливо критично для потокових ЦОС-застосунків.

Питання когерентності пам'яті також істотно впливає на вибір архітектури взаємодії між HPS та FPGA. Порт Accelerator Coherency Port (ACP) дозволяє апаратним модулям працювати з кешованими структурами даних процесора, спрощуючи програмну модель [43, 50]. Водночас експериментальні дослідження демонструють, що продуктивність ACP обмежена обсягом кешу та не масштабується для великих потоків даних, характерних для цифрової обробки сигналів [32]. Це змушує розробника робити вибір між простотою програмування та пропускною здатністю, знову повертаючись до апаратно-залежних компромісів, наприклад, таких як показано на Рис. 2.4, CPU-керований шлях передбачає участь механізмів обробки переривань та планування, що зумовлює додаткові накладні витрати порівняно з DMA- та ACP-орієнтованими траєкторіями.

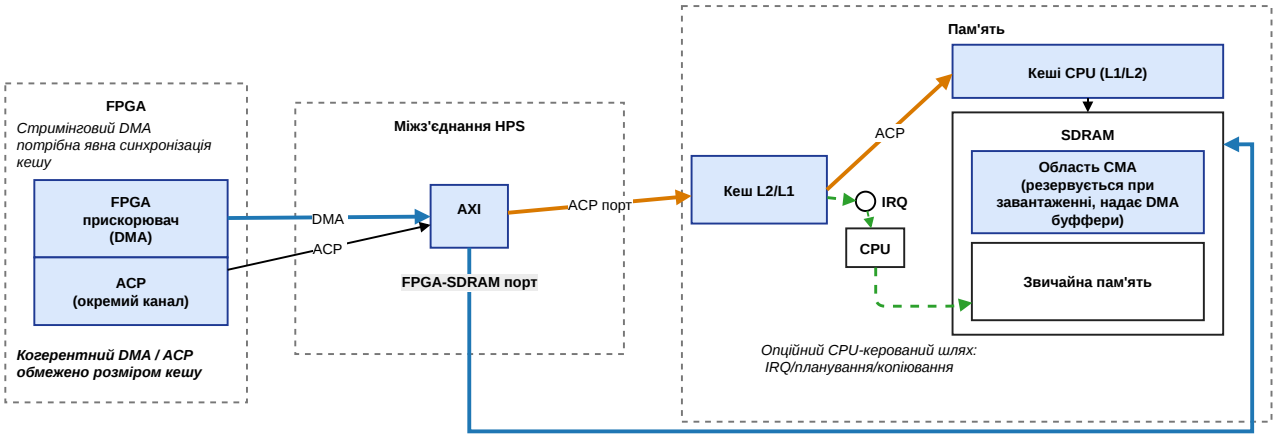


Рис. 2.4: Траєкторії доступу до пам'яті у системі на базі SoC FPGA: потоковий DMA, когерентний шлях через ACP та CPU-керований шлях із залученням обробки переривань.

Типовою практичною помилкою при використанні streaming DMA є некоректна синхронізація кешу процесора щодо буферів у DDR. Наприклад, якщо користувацький застосунок читає буфер, який був попередньо закешований, а FPGA/DMA тим часом записав у фізичну пам'ять нові дані, то без операцій

invalidate/flush застосунок може отримати «стару» версію вмісту з кешу. У реальній системі це проявляється як нестабільні або «плаваючі» результати ЦОС (інколи правильні, інколи ні) без явних помилок у логах, що легко сплутати з дефектами алгоритму. Натомість використання когерентного шляху (ACP) може спростити модель узгодження даних, однак на практиці придатне переважно для малих буферів через обмеження розміру кешів та специфіку арбітражу доступу. Цей клас дефектів підкреслює, що для вузлів ЦОС важливими є не лише пропускна здатність, але й коректний «контракт пам'яті» між доменами CPU та FPGA. Саме тому практичний досвід побудови експериментальних систем підтверджує, що більшість складних збоїв виникає не на рівні прикладних алгоритмів, а на межі апаратного та програмного шарів. [50, 51] У роботі, присвяченій офлайн-обробці радіосигналів, було показано, що проблеми синхронізації та буферизації проявлялися лише під реальним навантаженням, попри коректну роботу системи в тестових умовах [18]. Аналогічно, при розробці апаратно-програмної архітектури для часово-цифрового перетворювача основні труднощі були пов'язані з узгодженням DMA, переривань та адресного простору [21].

Отже, апаратно-залежний шар Linux-платформи на базі SoC FPGA визначає не лише процес завантаження системи, але й межі її надійності та передбачуваності. Boot-ланцюг, handoff-артефакти, Device Tree та механізми роботи з пам'яттю формують прихований каркас системи, помилки в якому проявляються на значно пізніших етапах. Розуміння цих обмежень є необхідною умовою для побудови ефективних вузлів цифрової обробки сигналів, оскільки саме апаратно-залежні рішення визначають реальну поведінку програмного стеку незалежно від його формальної коректності.

2.5. Конфігурація та збірка ядра Linux для SoC FPGA

Після формування апаратно-залежного шару (bootchain, handoff-артефакти, Device Tree та параметри пам'яті) наступним критичним кроком є вибір і конфігурація ядра Linux, яке повинно коректно відобразити апаратну топологію SoC FPGA в узгоджений набір підсистем: керування пам'яттю, планувальник, переривання, драйвери шин та контролерів, а також специфічні механізми роботи з FPGA-частиною. [37, 43, 52] На відміну від «універсальних» платформ, де більшість рішень приховані за стандартними BSP, у SoC FPGA некоректна

або надмірна конфігурація ядра здатна як знизити продуктивність потокових задач, так і створити приховані збої, які проявляються лише під навантаженням. Практичний досвід побудови експериментальних систем для збору та обробки даних підтверджує, що проблемні режими часто виникають саме на межі ядра, DMA та узгодження адресного простору з апаратним дизайном. [18, 21] Інженерно, доцільно розглядати конфігурацію ядра як «контракт другого рівня»: якщо Device Tree визначає що саме існує в апаратурі, то Kconfig визначає які механізми ядра можуть з цією апаратурою працювати і в яких режимах. [53, 54] При цьому помилкові комбінації можуть не проявлятися під час компіляції, але призводити до некоректних runtime-сценаріїв, а масштабність простору варіантів ускладнює ручну валідацію. [55, 56] Тому практичні методики конфігурації ядра для SoC FPGA мають спиратися не лише на використання певних драйверів, а й на контроль варіативності, відтворюваності та діагностичної придатності зібраного образу. Взаємодію між апаратною платформою, описом апаратури та конфігурацією ядра прийнято розглядати як багаторівневу контрактну модель, у якій кожна сутність фіксує власний клас поведінки. Узагальнення цих контрактів та типових наслідків їх порушення наведено в табл. 2.4.

Таблиця 2.4

Контракти конфігурації ядра Linux у системах на базі SoC FPGA

Рівень контракту	**Сутність**	**Наслідки порушення**
Апаратний	Апаратний дизайн SoC FPGA	Призводить до повної відмови завантаження або невідтворюваної поведінки системи
Опис апаратури	Device Tree (DTB / DTBO)	Конфлікти адрес, некоректна ініціалізація драйверів, нестабільна робота DMA та IRQ
Механізми ядра	Kconfig / `.config`	Призводить до прихованих runtime-збоїв, деградації продуктивності або непередбачуваної поведінки
Узгодження	Driver probe, DMA mappings, cache sync	Система стартує, але проявляє помилки лише під навантаженням, що ускладнює діагностику і маскує першопричину

Такий підхід дозволяє інтерпретувати збої у системах на базі SoC FPGA в якості наслідоків порушення контрактів між рівнями програмно-апаратного

стеку, що істотно спрощує локалізацію першопричин і підвищує керованість складних конфігурацій.

Базовою точкою відліку зазвичай є вендорний або референсний конфіг (defconfig), який відображає мінімально життєздатний набір опцій для конкретного сімейства SoC та типових периферійних блоків. [38, 44] Однак для дослідницьких і прикладних систем цифрової обробки сигналів цього, як правило, недостатньо: необхідні чітко визначені режими керування пам’яттю, пріоритети обробки переривань, контроль затримки планувальника, а також підсистеми роботи з FPGA як з керованим ресурсом Linux. [47, 50] У власних роботах, присвячених побудові Linux-платформи для SoC FPGA, показано доцільність підходу, коли ядро та користувацький простір налаштовуються як єдиний стек із урахуванням сценаріїв віддаленого керування, розгортання сервісів та швидкої інтеграції прикладних модулів. [12, 16]

Окремої уваги в контексті конфігурації ядра Linux для SoC FPGA заслуговує підсистема Kconfig, яка формує простір допустимих варіантів ядра як ієрархічну модель залежностей між функціональними підсистемами. На відміну від статичних конфігурацій, Kconfig є динамічною декларативною мовою, в якій окремі опції можуть змінювати свою доступність залежно від архітектури, вибраних драйверів або глобальних параметрів ядра. [53] У випадку SoC FPGA це призводить до ситуації, коли увімкнення або вимкнення окремих, на перший погляд, другорядних опцій (наприклад, механізмів керування пам’яттю, специфічних типів IRQ-контролерів або підтримки певних шин) може непрямо впливати на працездатність апаратно-залежних підсистем. Дослідження еволюції моделі варіативності ядра Linux показують, що зі зростанням складності конфігураційного простору зростає і ризик неявних конфігураційних дефектів, які не виявляються стандартними засобами валідації під час компіляції. [55, 57]

З точки зору проектування системи це означає, що конфігурація ядра для SoC FPGA має розглядатися як керований артефакт. Підсистема Kbuild, яка визначає правила збірки ядра, модулів та зовнішніх компонентів, відіграє тут не менш важливу роль, оскільки саме вона забезпечує відтворюваність результатів при зміні інструментального ланцюга або версії вихідного коду. [53, 58] Для експериментальних та дослідницьких платформ цифрової обробки сигналів, де конфігурація ядра часто еволюціонує разом з апаратним дизайном FPGA, критично важливо фіксувати не лише фінальний `.config`, але й спосіб його

отримання (базовий `defconfig`, фрагменти конфігурацій, порядок застосування змін). Такий підхід безпосередньо пов'язаний з концепцією відтворюваних збірок і дозволяє зменшити розрив між експериментальною платформою та її подальшим розвитком або портованістю на інші SoC FPGA-рішення. [59]

Ключовим технічним вузлом для DSP/edge-сценаріїв на SoC FPGA є шлях передачі даних між FPGA та SDRAM, який у реальних застосуваннях майже завжди спирається на DMA. [32, 46] Це одразу накладає вимоги на конфігурацію ядра в частині: (1) коректної роботи підсистеми DMA engine, (2) механізмів виділення суміжної пам'яті (CMA), (3) політик кешування та (4) засобів синхронізації в драйверах. Як наслідок, оптимізація ядра полягає не лише в «увімкнути DMA», а у виборі режимів (coherent/streaming), налаштуванні резервування CMA на ранньому boot-етапі та забезпеченні відтворюваної конфігурації, при якій великі буфери стабільно виділяються упродовж довгого часу роботи системи. [46, 60] На практиці саме тут виникають найскладніші для діагностики ефекти: помилки узгодження кешу або неправильні припущення щодо когерентності здатні імітувати «плаваючі» дефекти алгоритмів, хоча першопричина лежить у конфігурації та драйверних шляхах ядра. [18, 50]

Окремий пласт вимог стосується часової поведінки системи. Для задач потокової обробки сигналів типові не лише вимоги до пропускної здатності, але й до передбачуваності затримок, особливо коли CPU керує запуском апаратних конвеєрів, обробляє події або виконує пост-обробку результатів. [21] У цьому контексті важливо розуміти компроміс між класичним ядром та підходами «м'якого реального часу» на базі PREEMPT_RT, його використання може підвищувати стабільність та покращувати детермінізм реакції, але водночас ускладнює підтримку та накладає вимоги на дисципліну драйверного коду і налаштування пріоритетів. [61] Тому доцільним є прагматичний підхід: використовувати PREEMPT_RT або суміжні механізми лише там, де це підтверджено вимірюваннями і реально впливає на якість обробки, а для решти підсистем — зберігати стабільну й відтворювану конфігурацію ядра. [51] У рамках розробленої архітектури прийнято рішення про відмову від застосування патчу `PREEMPT_RT` на користь стандартного планувальника з механізмами ізоляції ядер. Це зумовлено двома факторами.

- По-перше, для платформ класу Cyclone V (архітектура ARM Cortex-A9) накладні витрати (overhead) на перемикання контексту та обробку

переривань у ядрі PREEMPT_RT можуть знижувати загальну пропускну здатність системи на 10–20% через заміну спін-блокувань на м'ютекси та примусову обробку переривань у потоках [61], що є критичним для задач потокової агрегації даних.

- По-друге, оскільки задачі жорсткого реального часу (Hard Real-Time) делеговано апаратній логіці (FPGA), роль операційної системи зводиться до функцій «м'якого» керування (Control Plane), що відповідає рекомендованій моделі розгортання для SoC FPGA [62]. Для цього класу задач стратегія просторової ізоляції (Spatial Isolation), тобто виділення окремого фізичного ядра під критичний процес, забезпечує вищий рівень детермінізму, ніж програмне витискання задач у PREEMPT_RT.

RT-функціонал є доцільним у сценаріях, де критичними є:

1. гарантований час обробки IRQ та запуску користувацького потоку після DMA;
2. малий допустимий розкид затримок у замкненому контурі керування;
3. жорсткі часові вимоги до мережевих або IPC-взаємодій.

Натомість у типових edge-сценаріях, де Linux виконує сервісні функції (агрегація, логування, мережевий обмін) і допускається стрибок затримки порядку десятків мікросекунд, достатнім може бути налаштування SMP-системи: ізоляція ядра (core affinity / cpuset), пріоритизація потоків, мінімізація фонових служб та коректна робота з DMA/CMA.

Збірка ядра для SoC FPGA також повинна бути організована як відтворюваний процес, де зміни конфігурації, патчі та версії інструментів контролюються так само суворо, як і HDL/HLS-частина проєкту. [53, 59, 63] Це особливо важливо для edge-платформ з обмеженими ресурсами, де оновлення та перескладання компонентів часто виконуються інкрементально, а будь-яка дрейфуюча залежність може призвести до непередбачуваних відмінностей у поведінці системи. В такій постановці задача «зібрати ядро» фактично перетворюється на задачу «відтворити конфігурацію платформи», яка включає: контроль вихідних кодів, параметрів Kbuild, фрагментів конфігурації, Device Tree та boot-аргументів. [52, 53] Додатково, для систем з активною еволюцією конфігураційного простору релевантними стають підходи аналізу та контролю варіативності, описані в роботах з еволюції моделей варіативності ядра Linux і супутніх методів підтримки коректності наборів опцій. [55, 57]

Окремо, для SoC FPGA принциповим є те, що ядро Linux не є ізольованим: воно є частиною програмно-апаратної платформи, де коректність визначається узгодженістю між апаратним дизайном (мости, адресний простір, переривання), Device Tree та конфігурацією ядра. [37, 43, 54] У підході, який використано в попередніх роботах автора щодо побудови вбудованої Linux-платформи для SoC FPGA, акцент робиться на практичній керованості цього стеку: можливості швидко змінювати конфігурацію, відтворювати образи, додавати інструменти взаємодії з апаратурою та підтримувати віддалений доступ для експериментів і розгортань. [12, 16, 22]

Якщо табл. 2.4 фіксує контрактні артефакти конфігурації ядра на рівні окремих механізмів і параметрів, то узагальнена контрактна модель, наведена на Рис. 2.5, відображає системну узгодженість між апаратною конфігурацією, описом платформи та ядром Linux як єдиним інженерним стеком.

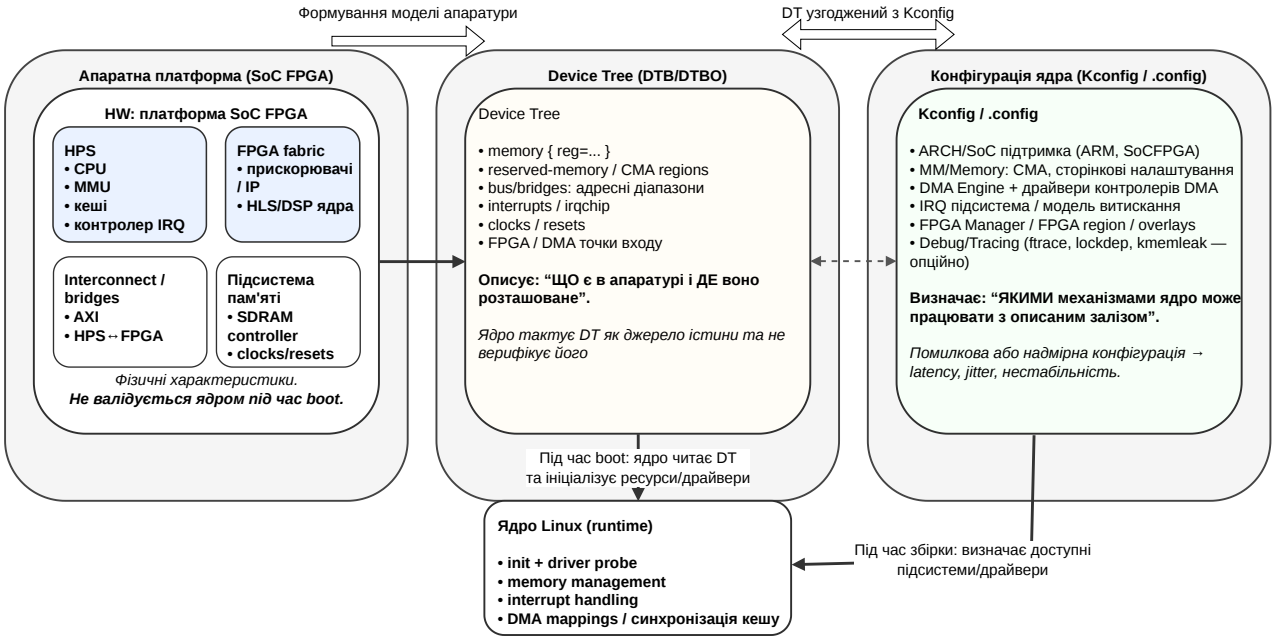


Рис. 2.5: Контрактна модель узгодженості ядра Linux у системі на базі SoC FPGA.

Таким чином, конфігурацію та збірку ядра Linux у SoC FPGA слід розглядати як інженерну процедуру формування системних інваріантів, котрі визначають продуктивність, стабільність і діагностичну прозорість усієї платформи цифрової обробки сигналів.

2.6. Побудова та структура rootfs у вбудованих Linux-системах на базі SoC FPGA

Користувацький простір (root filesystem, rootfs) у вбудованих Linux-системах не є пасивним доповненням до ядра, а виступає повноцінним елементом програмно-апаратної архітектури, що визначає експлуатаційні властивості платформи: від керованості, діагностичної прозорості та відтворюваності до можливості еволюції системи без повного перезбирання образу. У роботах, присвячених системному дизайну вбудованих Linux-рішень, показано, що структура rootfs і набір користувацьких компонентів часто є визначальними для стабільності та передбачуваності поведінки системи, особливо в edge- і DSP-сценаріях [64, 65].

У контексті SoC FPGA роль rootfs набуває додаткової складності. Linux-платформа в таких системах функціонує як програмний домен керування і координації, тоді як основні обчислення винесені в апаратну логіку FPGA. Це формує асиметричну модель, у якій користувацький простір не лише ініціює та обслуговує апаратні конвеєри, але й виконує сервісні функції: буферизацію, агрегацію, мережеву взаємодію, логування та віддалене керування. Таким чином, rootfs доцільно розглядати як елемент системи, котрий фіксує політику взаємодії між апаратною частиною, ядром Linux та прикладними процесами. Еволюція ролі користувацького простору від статичного середовища до модульної сервісної платформи схематично показана на Рис. 2.6, де підкреслено перехід від монолітної конфігурації до динамічного керування сервісами без втручання в ядро системи.

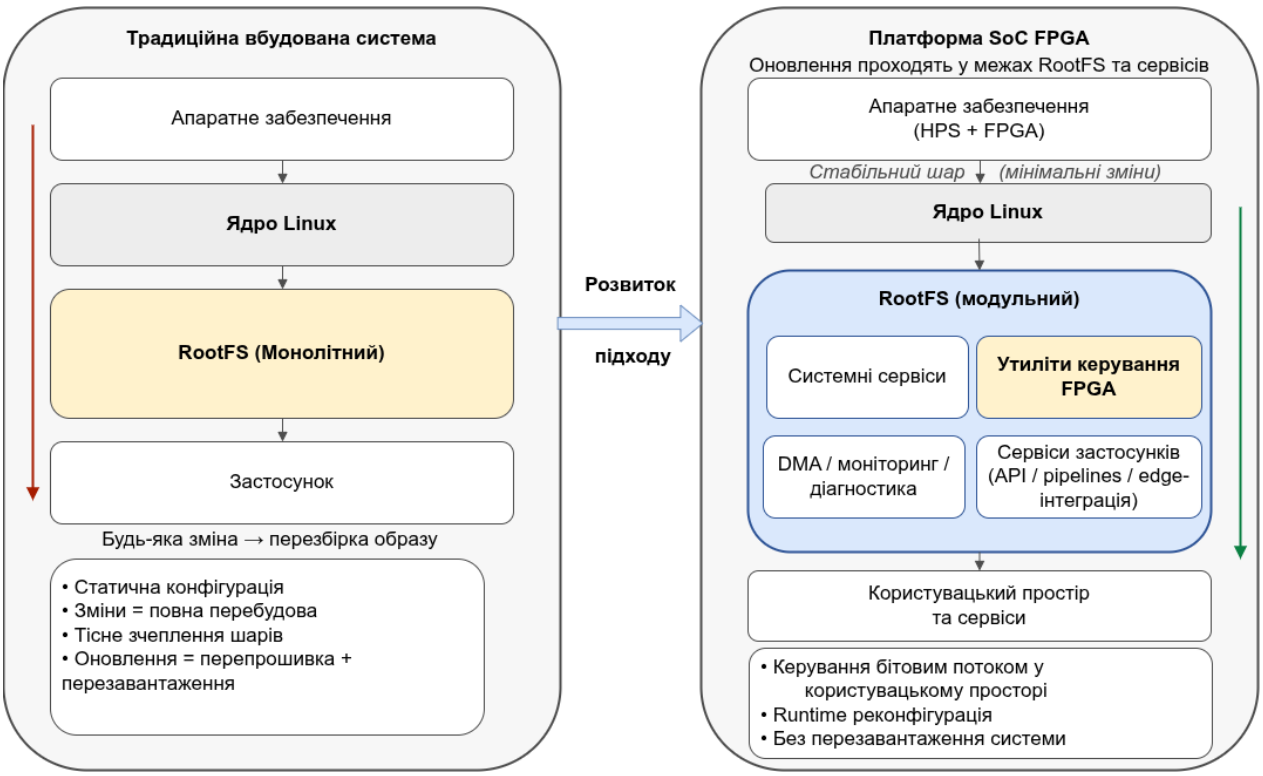


Рис. 2.6: Еволюція ролі RootFS у SoC FPGA-системах.

Саме на цьому рівні відбувається основні адаптаційні процеси, модифікація інструментів взаємодії з апаратурою та сценаріїв обробки даних, тоді як ядро Linux та апаратний дизайн зберігають відносну стабільність. Такий розподіл ролей дозволяє локалізувати зміни та мінімізувати ризик порушення системних інваріантів платформи.

У загальному випадку rootfs вбудованої Linux-системи включає мінімальний набір системних бібліотек, ініціалізаційне середовище, сервісний менеджер, інструменти діагностики та прикладні компоненти. Дослідження архітектур edge-операційних систем показують, що надмірна насиченість користувацького простору призводить не лише до зростання розміру образу, але й до ускладнення аналізу збоїв, збільшення поверхні помилок і зниження передбачуваності часової поведінки системи під навантаженням [66, 67]. Це особливо критично для систем цифрової обробки сигналів, де приховані збої у програмному домені можуть маскуватися під дефекти алгоритмів або апаратної логіки.

З інженерної точки зору, формування rootfs для SoC FPGA доцільно здійснювати на основі чітко визначених принципів: мінімальність, явна керованість залежностей, відтворюваність збірки та підтримка інкрементальних змін. Ці принципи узгоджуються з сучасними підходами до проєктування

відтворюваних програмних систем, у яких користувацький простір розглядається як версіонований і контрольований об’єкт [59, 68]. У цьому сенсі rootfs виступає своєрідним «третьорівневим контрактом» після апаратної конфігурації та ядра Linux.

Практичний досвід, отриманий у ході розробки експериментальних Linux-платформ для SoC FPGA, підтверджує доцільність такого підходу. У власних роботах автора показано, що узгоджене налаштування ядра та користувацького простору як єдиного стеку дозволяє суттєво зменшити кількість важковідтворюваних збоїв і спростити діагностику системи [12, 16]. Зокрема, під час створення експериментальних edge-вузлів для збору та обробки сигналів було продемонстровано, що агресивна мінімізація rootfs та вилучення компонентів, не пов’язаних безпосередньо з функціональним призначенням платформи, підвищує стабільність роботи в умовах тривалого навантаження [18, 19]. Критеріями такої мінімізації виступали:

- зменшення загального розміру RootFS;
- зниження кількості активних сервісів у робочому режимі;
- скорочення споживання оперативної пам’яті в idle-стані;
- збереження стабільності та передбачуваності поведінки системи.

Окремого обґрунтування потребує вибір дистрибутивної моделі rootfs. Статичні підходи, орієнтовані на фіксований набір пакетів і жорстко визначений життєвий цикл образу, спрощують первинне розгортання, але ускладнюють подальшу еволюцію системи. Натомість динамічні моделі користувацького простору дозволяють адаптувати платформу до змін апаратної конфігурації, сценаріїв використання або вимог прикладного програмного забезпечення без повного перезбирання rootfs. Дослідження операційних систем для гетерогенних обчислювальних платформ підтверджують, що саме гнучкість користувацького простору є ключовим фактором життєздатності таких систем у довготривалій перспективі [69, 70].

У цьому контексті використання динамічного rootfs на базі Arch Linux дозволяє реалізувати модель, у якій користувацький простір розвивається разом із платформою, замість того аби формувати фіксовану структуру на етапі початкового проєктування. Пакетна модель, чітке управління залежностями та можливість тонкої мінімізації дозволяють формувати rootfs, орієнтований саме на задачі SoC FPGA, зберігаючи при цьому здатність до швидкої модифікації та

експериментів. У попередніх роботах автора показано, що такий підхід спрощує інтеграцію нових сервісів, апаратно-орієнтованих утиліт і засобів віддаленого керування без порушення цілісності системи [17, 22].

Таким чином, rootfs у вбудованих Linux-системах на базі SoC FPGA слід розглядати не як допоміжний компонент, а як інженерний інструмент формування поведінки платформи. Вибір структури користувацького простору, моделі керування пакетами та рівня мінімізації безпосередньо впливає на стабільність, відтворюваність і практичну придатність систем цифрової обробки сигналів у реальних edge-сценаріях. Водночас принциповим є чітке розмежування відповідальності між користувацьким простором та ядром Linux. Простір користувача не повинен компенсувати помилки апаратного дизайну, некоректну конфігурацію ядра або порушення контрактів Device Tree. Його роль полягає у реалізації політик, сервісів і сценаріїв використання апаратних ресурсів, оскільки він слугує відображенням периферії - а отже не здатен усунути фундаментальні недоліки будованої платформи.

Описаний підхід до побудови користувацького простору має значення не лише як спосіб підвищення зручності розробки, але й з позицій відтворюваності системи. У контексті SoC FPGA відтворюваність означає можливість відновлення узгодженого програмно-апаратного стану платформи, включно з версією ядра, набором сервісів, конфігурацією взаємодії з FPGA та допоміжними інструментами. Така інтерпретація дозволяє розглядати rootfs як частину інженерного контракту системи. Для подальшого аналізу ефективності запропонованого підходу RootFS розглядається як вимірюваний об’єкт, характеристики якого можуть бути описані набором кількісних метрик. До таких метрик належать розмір файлової системи, кількість встановлених пакетів, число активних сервісів та споживання оперативної пам’яті у стабільному режимі роботи. Збір та аналіз цих показників виконувались на експериментальній платформі на базі SoC FPGA (DE10-Nano).

2.7. Підсистема сервісів, комунікацій та інтеграція з апаратними прискорювачами

У сучасних платформах на базі SoC FPGA апаратні прискорювачі дедалі рідше розглядаються як ізольовані обчислювальні модулі. Натомість вони виступають повноцінними елементами програмно-апаратної інфраструктури,

інтегрованими у модель пам’яті, механізми керування потоками даних та сервісну архітектуру операційної системи. Дослідження у галузі реконфігурованих і гетерогенних обчислювальних систем показують, що ефективність використання FPGA визначається не лише характеристиками окремих обчислювальних ядер, а способом їх включення у системний стек [71, 72]. Важливо відзначити, що вибір механізму інтеграції FPGA з Linux (драйвер ядра, UIO/VFIO, user-space API або допоміжні інструменти доступу) є архітектурним рішенням, і досить рідко здатен слугувати в якості точкової оптимізації. Таким чином визначається продуктивність, складність супроводу, відтворюваність системи та можливість її еволюції.

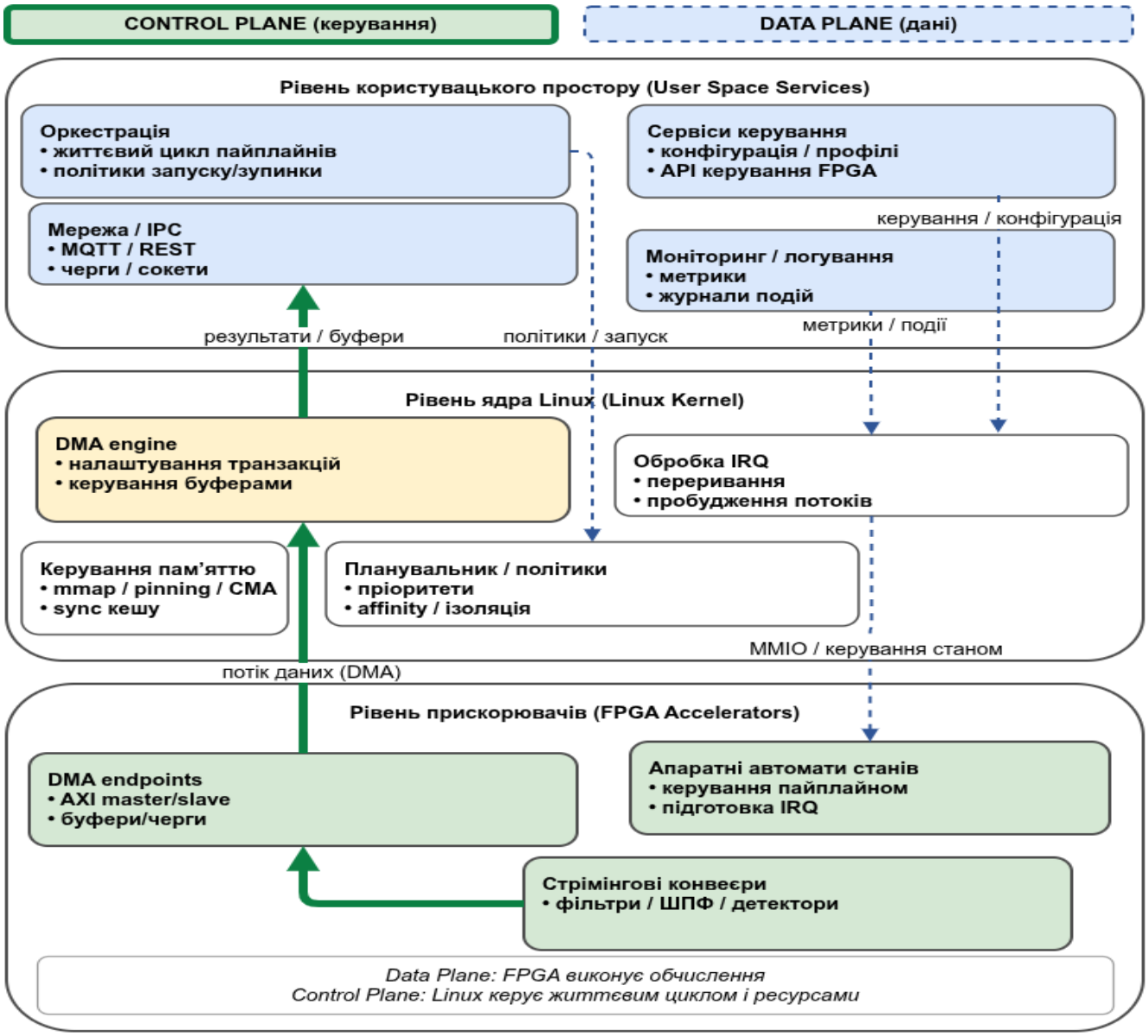


Рис. 2.7: Розподіл на площину керування (Control Plane) та площину даних (Data Plane) з оркестрацією через користувацькі сервіси

Сервісно-орієнтована модель взаємодії між Linux та FPGA, з чітким розмежуванням площин керування та обробки даних, ілюструється на Рис. 2.7, де Linux виступає координатором, а апаратні прискорювачі — виконавцями. Такий підхід дозволяє розглядати FPGA не лише як засіб локального прискорення, але і як складову сервісно-орієнтованих, мережевих або ЦОС-орієнтованих систем, що є характерним для edge та DSP-сценаріїв [73]. У цьому контексті Linux формує єдине середовище керування життєвим циклом апаратних і програмних компонентів. Ключовим технічним каналом взаємодії між FPGA та програмним доменом залишається підсистема прямого доступу до пам’яті (DMA). Роботи, присвячені аналізу AXI- та PL330-сумісних DMA-контролерів, демонструють, що пропускна здатність і стабільність передач значною мірою залежать від організації буферів та політик кешування [74, 75]. У SoC FPGA DMA слід розглядати як системний вузол, що визначає коректність і часову поведінку всього обчислювального конвеєра [76].

Подальший розвиток комунікацій спрямований на перехід до zero-copy взаємодії, де буфер розглядається як спільний ресурс апаратного та програмного доменів. Дослідження детермінованих zero-copy механізмів підтверджують, що усунення зайвих операцій копіювання не лише знижує затримку, а й суттєво підвищує стабільність потокової обробки даних, що є критичним для DSP-сценаріїв [77]. Впровадження підходів на базі DMA-BUF дозволяє ядру Linux та користувацьким сервісам працювати з узгодженими об’єктами пам’яті, мінімізуючи навантаження на CPU як координатора обчислень [78].

У практичних DSP- та SDR-застосуваннях DMA-підсистема часто стає вузьким місцем не через пропускну здатність інтерфейсу, а через спосіб організації буферів і синхронізації доступу до них. Реалізації високошвидкісної взаємодії між FPGA та процесором для SDR-систем показують, що стабільна робота досягається лише за умови чіткого розмежування відповідальності між апаратним конвеєром та програмним контролером, а також мінімізації переходів між адресними просторами [32].

Одним із апаратно-орієнтованих підходів до зменшення накладних витрат є використання Accelerator Coherency Port (ACP), який дозволяє FPGA-модулям працювати безпосередньо з кешованими структурами даних процесора. Експериментальні результати свідчать, що ACP може бути енергоефективним рішенням для певних класів задач, однак його продуктивність і масштабованість

обмежені розміром кешу та особливостями когерентності [79]. У потокових DSP-сценаріях це часто змушує повертатися до класичних DMA-шляхів із явно керованими буферами в оперативній пам’яті.

Для систем з жорсткими часовими вимогами, зокрема у високошвидкісних системах збору та обробки даних, DMA розглядається як критичний компонент реального часу. Роботи в галузі багатогігабітних FPGA-систем для фізичних експериментів показують, що саме узгодженість DMA, буферизації та програмної обробки визначає можливість досягнення детермінованої часової поведінки [80]. У таких системах Linux виступає координуючим рівнем, котрий керує життєвим циклом даних між апаратними та програмними модулями. Фактично, DMA у SoC FPGA виконує роль межового контракту між апаратним та програмним доменами. Саме на цьому рівні сходяться питання адресного простору, кешування, синхронізації та часової поведінки. Тому більшість складних і важковідтворюваних збоїв у SoC FPGA-системах виникають не в обчислювальних ядрах, а на етапі взаємодії DMA з програмним стеком вбудованої операційної системи [80, 81].

З погляду інтеграції в Linux-системах на базі SoC FPGA, ці підходи логічно доповнюють механізми DMA-BUF та user-space орієнтовані моделі доступу до апаратних буферів. Разом вони формують архітектурний рівень обміну даними, у якому FPGA, ядро Linux та користувацькі сервіси працюють з узгодженими об’єктами пам’яті, не створюючи ізольованих копій. Таким чином, сучасні DMA- та zero-copy механізми слід розглядати не як оптимізацію окремих операцій, а як основу для побудови масштабованих і відтворюваних програмно-апаратних конвеєрів у SoC FPGA-системах.

Важливою особливістю SoC FPGA-платформ є те, що більшість помилок інтеграції проявляються не у вигляді явних відмов, а як приховані збої під навантаженням. Некоректні припущення щодо когерентності пам’яті, синхронізації або режимів DMA можуть призводити до нестабільної поведінки системи, що ускладнює їх діагностику. Це підтверджує необхідність розглядати комунікаційну підсистему саме як частину архітектурного проєктування, не прив’язуючись до конкретного IP-блоку.

З програмної точки зору можна виокремити декілька базових моделей інтеграції апаратних прискорювачів у Linux. Класичний підхід полягає у розробці повноцінних драйверів ядра, які інкапсулюють доступ до регістрів, керування

DMA та обробку переривань. Такий варіант забезпечує максимальну інтеграцію з ядром Linux, але водночас ускладнює супровід і підвищує вартість змін у разі еволюції апаратного дизайну [50].

Альтернативні підходи ґрунтуються на винесенні значної частини логіки взаємодії з FPGA у користувацький простір. Механізми UIO або VFIO дозволяють ядру виконувати лише мінімальні функції з ізоляції та керування ресурсами, залишаючи реалізацію прикладних протоколів доступу програмному рівню [82]. Подібна модель виявляється особливо ефективною у адаптивних системах, де апаратна конфігурація часто змінюється, а швидкість ітерацій має вирішальне значення.

У порівнянні з класичними драйверами ядра, user-space підходи (UIO, VFIO) знижують складність розробки та ризик помилок у коді ядра, проте перекладають відповідальність за коректність доступу, синхронізацію та обробку помилок на рівень прикладних сервісів. Такий компроміс є прийнятним у системах, де FPGA виконує основну обчислювальну роботу, а CPU використовується переважно для координації та керування. Порівняльний аналіз наведених моделей інтеграції за критеріями складності, продуктивності та безпеки представлено у Табл. 2.5.

Найпростішим способом доступу до апаратних регістрів залишається пряме відображення фізичної пам’яті за допомогою утиліт на кшталт `devmem`. Однак такий підхід повністю обходить механізми синхронізації та захисту Linux і може розглядатися лише як інструмент первинного налагодження [60]. У стабільних системах він не здатний забезпечити керованість і безпечність доступу до апаратних ресурсів.

Застосування високорівневого синтезу (HLS) спрощує розробку обчислювальних ядер, проте акцентує проблему визначення програмно-апаратних інтерфейсів [10]. Автоматично згенеровані структури потребують адаптації до системних вимог Linux для коректного узгодження адресних просторів та керування DMA-транзакціями [83]. При цьому обмеженість традиційних методів спостережуваності HLS-модулів змушує комбінувати апаратні та програмні інструменти аналізу для верифікації їхньої поведінки в реальному часі.

У ширшому контексті гетерогенних обчислень апаратні прискорювачі дедалі частіше інтегруються у сервісну інфраструктуру системи. Фреймворки на зразок ”ARTICo^3”або FRED-Linux демонструють підходи до динамічного

Таблиця 2.5

Порівняльний аналіз моделей інтеграції FPGA-прискорювачів у середовище Linux

Критерій	**Драйвер ядра (Kernel Class)**	**UIO (Userspace I/O)**	**VFIO (Virtual Function I/O)**
Місце логіки	Простір ядра	Простір користувача	Простір користувача
Продуктивність	Максимальна інтеграція та обробка переривань [71]	Висока швидкість ітерацій та адаптивність [78]	Висока, з функціями ізоляції ресурсів [77]
Складність	Висока вартість змін та складність супроводу [16]	Знижений ризик помилок у коді ядра [72]	Середня; потребує мінімальних функцій ядра [12]
Керування DMA	Повна інкапсуляція в драйвері [32]	Відповідальність прикладного рівня [78]	Керування через механізми ізоляції ядра [77]
Гнучкість	Обмежена еволюцією апаратного дизайну [18]	Висока для систем, що часто змінюються [72]	Орієнтована на гнучке керування ресурсами [16]

керування апаратними модулями, включно з реконфігурацією та балансуванням навантаження [84, 85]. Попри привабливість таких рішень, вони накладають додаткові обмеження на гнучкість системи і можуть ускладнювати глибоку оптимізацію під специфічні DSP-застосування.

Для потокових задач цифрової обробки сигналів визначальним є не лише максимальна пропускна здатність, а мінімізація затримки та передбачуваність обробки. Потокові моделі взаємодії між програмним та апаратним доменами дозволяють ефективно розподіляти навантаження, залишаючи основні обчислення у FPGA, а координацію — Linux [85, 86]. У таких системах операційна система виконує роль диспетчера потоків і подій, забезпечуючи узгодженість усіх компонентів.

Часова поведінка програмного домену також істотно впливає на стабільність

системи. Практичні дослідження підтверджують, що більшість складних збоїв у SoC FPGA-системах виникає не на рівні алгоритмів, а на межі взаємодії між FPGA, DMA та програмним стеком Linux [18, 21]. Це підкреслює необхідність розглядати інтеграцію апаратних прискорювачів як системну інженерну задачу, що потребує узгодженого проєктування апаратного дизайну, конфігурації ядра та сервісної моделі [12, 16].

Таким чином, сучасна комунікаційна підсистема SoC FPGA є не просто набором IP-блоків, а межовим контрактом, що визначає часову поведінку всього обчислювального конвеєра. Узгоджене проектування апаратного дизайну та конфігурації ОС дозволяє мінімізувати неявні залежності в стеку, забезпечуючи відтворюваність і контроль, необхідні для стабільної роботи edge-платформ [18]. Ефективність такої інтеграції створює передумови для подальшого кількісного аналізу продуктивності та експлуатаційних характеристик системи в межах DSP- та edge-сценаріїв.

2.8. Продуктивність, керованість та відтворюваність Linux-платформи на базі SoC FPGA

Аналіз попередніх етапів дослідження показує, що підсумкова ефективність SoC FPGA-платформи не є результатом ізольованих апаратних або програмних оптимізацій. Вона визначається системною узгодженістю всього програмно-апаратного стеку: від початкових етапів boot-ланцюга та формалізованих контрактів у *Device Tree* до конфігурації ядра Linux, політик керування ресурсами та механізмів інтеграції апаратних прискорювачів. У такому гетерогенному середовищі абсолютні показники продуктивності втрачають інтерпретаційну цінність без прив’язки до керованості, повторюваності та відтворюваності конфігурацій. Саме ці властивості забезпечують придатність платформи до довготривалої експлуатації, масштабування та коректного порівняння експериментальних результатів у різних сценаріях.

Розглянутий інженерний підхід ґрунтується на побудові контрольованого середовища виконання, у якому часова та ресурсна поведінка системи фіксується через формалізований набір артефактів. На відміну від методик, орієнтованих на досягнення пікових значень у синтетичних тестах, основний акцент зроблено на передбачуваності та доказовості результатів. Такий підхід дозволяє відокремити системні ефекти, обумовлені архітектурними рішеннями, від

випадкових флуктуацій, спричинених фоновою активністю операційної системи або нестабільними політиками керування ресурсами.

У сучасних SoC FPGA-платформах операційна система Linux не повинна розглядатися як накладний шар, що лише створює додаткові затримки та зменшує доступний обчислювальний бюджет. Навпаки, вона виконує роль координатора ресурсів, забезпечуючи узгоджену взаємодію між процесорними ядрами, підсистемою пам'яті та апаратними прискорювачами. Дослідження в галузі вбудованих і реального часу Linux-систем показують, що за умови коректної конфігурації ядра та політик планування операційна система не стає визначальним вузьким місцем навіть у часово-чутливих сценаріях [61, 87].

У гетерогенних SoC FPGA-платформах така роль Linux набуває особливого значення через необхідність одночасного керування програмними сервісами загального призначення та спеціалізованими апаратними конвеєрами. Дослідження показують, що саме операційна система формує межі допустимих компромісів між продуктивністю, енергоспоживанням і передбачуваністю поведінки системи [88, 89].

У роботах, присвячених практичному застосуванню SoC FPGA в edge- та DSP-сценаріях, наголошується, що стабільність системних характеристик у часі є важливішою за досягнення максимальних пікових значень [12, 16]. Це пов'язано з тим, що більшість реальних застосувань працюють у тривалому режимі з повторюваними профілями навантаження, де накопичення росту затримки призводять до деградації якості обробки сигналів.

Таким чином, Linux у SoC FPGA-системах доцільно розглядати не як джерело нестабільності, а як інструмент формалізації та стабілізації поведінки складного програмно-апаратного середовища.

Відповідно, оптимізація SoC FPGA-платформи повинна бути спрямована не на мінімізацію ролі операційної системи, а на формування системних інваріантів — стабільних параметрів конфігурації, що забезпечують інтерпретованість і відтворюваність результатів. До таких інваріантів належать:

- фіксовані політики частоти та живлення (*cpufreq governor*), *Operating Performance Points*, що виключають неконтрольований дрейф продуктивності;
- визначена прив'язка обробки переривань та керування пріоритетами (*IRQ affinity*);

- передбачувана конфігурація пам’яті та режимів DMA, зокрема через використання CMA або резервування спеціалізованих областей.
- ізоляція та пріоритезація трафіку в системній шині (*Interconnect Isolation*), що запобігає конфліктам при одночасному доступі HPS та FPGA до спільної пам’яті.

Без забезпечення цього базового рівня керованості локальні оптимізації апаратних модулів або прикладних алгоритмів втрачають наукову цінність, оскільки отримані метрики стають надмірно чутливими до прихованих змін у стані системи.

Жорсткі вимоги реального часу не розглядаються як самоціль, а використовуються як індикатор загальної дисципліни керування ресурсами. Концепція *soft real-time* застосовується для оцінки стабільності затримок та рівня нестабільності при обробці подій і переривань. Подібний підхід, за якого характеристики реального часу інтерпретуються як показник якості системної конфігурації, широко використовується в дослідженнях керованих Linux-платформ [61, 87].

Для SoC FPGA-систем такий підхід є природним, оскільки часово-критичні операції зазвичай делегуються апаратним конвеєрам у логіці FPGA, тоді як Linux відповідає за координацію, синхронізацію та сервісне середовище. У цьому контексті параметри реального часу виступають не функціональною вимогою, а кількісною мірою керованості платформи. Наприклад, зміна прив’язки переривань (IRQ affinity) для критичного драйвера повинна супроводжуватися чітко зафіксованим та повторюваним зменшенням максимальної затримки (worst-case latency). Це дозволяє кількісно підтвердити пряму керованість системними метриками через зміну параметрів конфігурації. Причинно-наслідкові зв’язки між формальними контрактами різних рівнів програмно-апаратного стеку та появою прихованих збоїв під навантаженням узагальнено на Рис. 2.8, що дозволяє інтерпретувати складні runtime-відмови як наслідок порушень на ранніх етапах конфігурації.

Причинно-наслідкові зв’язки між формальними контрактами різних рівнів програмно-апаратного стеку та появою прихованих збоїв під навантаженням узагальнено на Рис. 2.8, що дозволяє інтерпретувати складні runtime-відмови як наслідок порушень на ранніх етапах конфігурації.

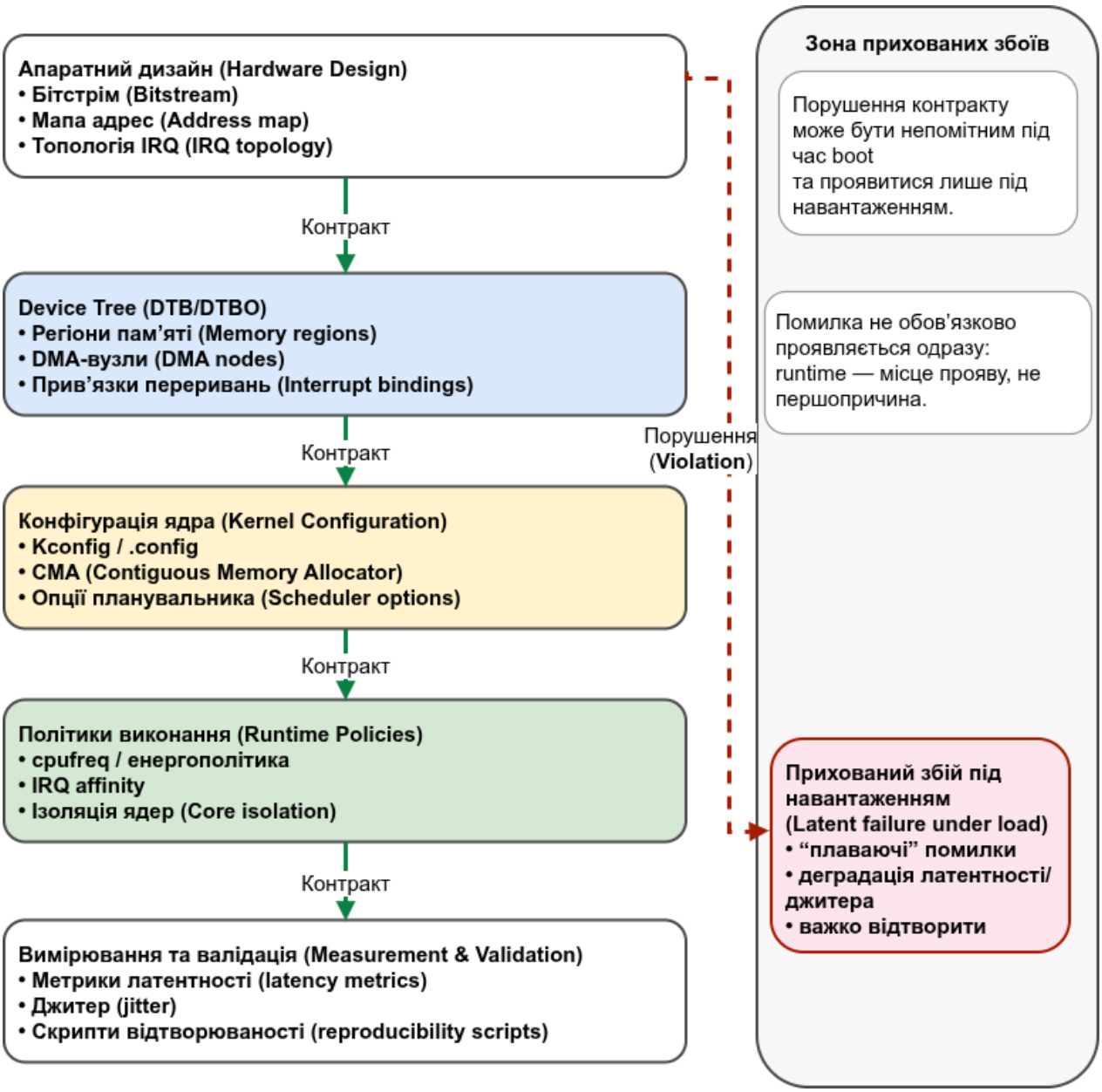


Рис. 2.8: Контракти та інваріанти програмно-апаратного стеку SoC FPGA: порушення на ранніх рівнях конфігурації як джерело прихованих збоїв під навантаженням

Ключовим аспектом керованості є конфігураційна визначеність системи. Для переходу від опису платформи як сукупності окремих компонентів до відтворюваного інженерного об'єкта вводиться поняття *паспорта конфігурації* — формалізованого набору артефактів, що фіксують стан системи на всіх рівнях стеку (табл. 2.6).

Таблиця 2.6

Структура артефактів паспорта конфігурації відтворюваної Linux-платформи

Рівень	**Склад артефактів та ключові параметри**
Software Core	Версія та commit ядра Linux, файл `.config`, хеш DTB, параметри *bootargs*.
Hardware Design	Бітстрім FPGA, *handoff*-файли, версія CAD-інструментів, ключові параметри дизайну.
Memory Subsystem	Налаштування CMA, конфігурація *hugepages*, режими кешування DMA-буферів, стан L2 Cache Pre-fetcher та політики Snoop Control Unit (SCU) для забезпечення когерентності.
Runtime Policies	Політики частот, *IRQ affinity*, ізоляція ядер, параметри планувальника.
Validation Tools	Скрипти збору метрик, версії інструментів профілювання, сценарії навантаження.

Запропонований паспорт конфігурації забезпечує однозначність інтерпретації експериментальних результатів. Будь-яка зміна продуктивності або часових характеристик може бути безпосередньо співвіднесена з конкретним параметром конфігурації. Для SoC FPGA-платформ це є критичним, оскільки навіть незначні відмінності в описі апаратури або політиках пам’яті можуть призводити до суттєвих змін у поведінці потоків даних без явних помилок на етапі запуску.

Для формалізації якості експериментального аналізу використовуються три взаємопов’язані критерії: *repeatability* — стабільність результатів при багаторазових вимірюваннях на незмінній платформі; *reproducibility* — відтворення характеристик після повної перебудови системи з того самого набору артефактів; *controllability* — передбачувана реакція метрик на цілеспрямовану зміну параметрів конфігурації. Такі критерії відповідають сучасним підходам до забезпечення наукової відтворюваності в системних дослідженнях [59].

Застосування формалізованих критеріїв repeatability, reproducibility та controllability дозволяє розглядати SoC FPGA-платформу як повноцінний

об’єкт системних досліджень. У сучасних роботах з системної інженерії підкреслюється, що відсутність таких критеріїв є однією з ключових причин низької відтворюваності результатів у дослідженнях гетерогенних обчислювальних систем [90, 91].

Для вбудованих Linux-платформ це питання ускладнюється поєднанням апаратних і програмних факторів, кожен з яких може впливати на результати експериментів непрямим чином. Підхід, що базується на «паспорті конфігурації», дозволяє мінімізувати ці ефекти шляхом явної фіксації всіх релевантних параметрів — від версій інструментів синтезу до політик планування задач у ядрі Linux [18].

У цьому контексті відтворюваність стає властивістю платформи. Саме така постановка задачі створює передумови для еволюційного розвитку SoC FPGA-рішень, у яких зміни архітектури або конфігурації можуть бути кількісно оцінені без втрати порівнянності з попередніми результатами.

Отже, підвищення продуктивності SoC FPGA-платформи досягається не шляхом агресивних локальних оптимізацій, а через побудову цілісного, керованого та відтворюваного середовища. У такій архітектурі Linux виступає як системний координатор, який за правильної конфігурації забезпечує стабільну роботу без формування прихованих вузьких місць та створюючи надійний фундамент для подальшого аналізу продуктивності та масштабування SoC FPGA-рішень.

2.9. Методика комплексної валідації та паспортизації платформи

У попередніх підрозділах (пп. 2.1–2.8) було сформульовано архітектурні вимоги до вбудованої Linux-платформи на базі SoC FPGA, зокрема керованість конфігурації, узгодженість програмно-апаратних контрактів та ізоляцію часово-критичних шляхів виконання. Для практичного підтвердження виконання цих вимог у цьому підрозділі запропоновано методику комплексної валідації, орієнтовану на перевірку відтворюваності, передбачуваності та контрольованості платформи в умовах реального експлуатаційного навантаження.

Запропонований підхід базується на концепції модульної побудови вбудованих систем, що дозволяє проводити верифікацію параметрів без втручання у вихідний код ядра та без модифікації завантажувача [16]. Це дозволяє застосовувати методику безпосередньо на цільових edge-платформах

та забезпечує узгодженість експериментів із концепцією «м'якої спеціалізації», сформульованою у попередніх підрозділах.

2.9.1. Паспортизація конфігурації та фіксація інваріантів

Першим етапом методики є формування цифрового паспорта конфігурації платформи. Метою цього етапу є перехід від декларативної контрактної моделі до строгої математичної формалізації стану системи.

При вимірюванні часових характеристик у гетерогенних телекомунікаційних системах (зокрема, фазового джитера ΔT), сумарна похибка вимірювань складається з випадкової апаратної складової ϵ та систематичної похибки $f(S_{\text{hidden}})$, викликаної прихованими станами операційної системи та арбітражу шин:

$$\Delta T = f(S_{\text{hidden}}) + \epsilon$$

Для усунення впливу неконтрольованих станів (S_{hidden}) у роботі застосовано формалізацію наведених раніше артефактів (див. табл. 2.6) у вигляді єдиного вектора стану платформи — Паспорта конфігурації:

$$S_{\text{passport}} = \{H, M, R\}$$

де H — вектор апаратних інваріантів (хеш дерева пристроїв, стан регістрів когерентності SCU); M — вектор підсистеми пам'яті; R — вектор політик виконання ОС (параметри планувальника, маски переривань). Жорстка фіксація цього вектора ($S_{\text{passport}} = \text{const}$) перетворює неконтрольовану систему на стаціонарний об'єкт дослідження, зводячи систематичну похибку $f(S_{\text{hidden}}) \to 0$.

Відповідно до цієї математичної моделі, для проведення подальших експериментів було зафіксовано базовий вектор стану платформи. Паспортизація включала зчитування апаратних інваріантів (регістрів SoC) у режимі *read-only*. Результати фіксації вектора S_{passport} наведено у Табл. 2.7.

Таблиця 2.7

Фізична фіксація вектора стану (Паспорта конфігурації) S_{passport}

Параметр	**Значення**
Вектор апаратних інваріантів (H)	
Архітектура CPU	ARMv7 (Dual Core Cortex-A9)
SCU Control Register	`0x21` (SCU enabled)
L2 Cache Control	`0x1` (PL310 enabled)
Вектор середовища виконання (R)	
Версія ядра Linux	5.13.0zImage
CONFIG_PREEMPT	Disabled
CONFIG_HZ	100

Аналіз зафіксованої конфігурації (компонентів вектора R) виявив, що механізм повного витискання (`CONFIG_PREEMPT_RT`) не активовано, а частота системного таймера становить 100 Гц. Це свідоме обмеження експерименту, яке дозволяє оцінити ефективність саме архітектурних методів просторової ізоляції ядер на стандартному ядрі Linux.

Зафіксовані характеристики S_{passport} використовуються як базова точка відліку для всіх подальших вимірювань та дозволяють однозначно інтерпретувати отримані результати, залишаючи для статистичного аналізу лише випадкову стохастичну складову похибки ϵ.

2.9.2. Методологія вимірювання затримки

Для оцінки часової поведінки системи використано інструмент `cyclictest`, котрий реалізує періодичні таймерні пробудження з високою роздільною здатністю (метод `clock_nanosleep`) та здатен генерувати гістограму розподілу затримок [92]. Нехай гістограма задається парами (t_i, n_i), де t_i - час затримки (мкс), а n_i - кількість подій у цьому біні.

Для визначення граничних метрик розраховується емпірична кумулятивна функція розподілу (CDF):

$$F(t) = \frac{1}{N} \sum_{t_i \leqslant t} n_i, \tag{2.1}$$

де $N = \sum n_i$ - загальна кількість вимірів (у даному експерименті $N \approx 8.4 \times 10^5$) що відповідає тривалості безперервного тестування 14 хвилин). Збільшення

вибірки дозволило охопити повний цикл активності фонових системних служб та врахувати ефекти теплової стабілізації кристала.

На основі CDF визначається **99-й перцентиль (P_{99})** - порогове значення затримки, яке не перевищується у 99% випадків:

$$P_{99} = \min\{t : F(t) \geqslant 0.99\}. \tag{2.2}$$

Ця метрика є індикатором гарантованої швидкодії системи для основної маси подій, відсікаючи 1% найгірших випадків. Порівняння P_{99} з абсолютним максимумом ($Lat_{\max}$) дозволяє виявити природу аномальних викидів, спричинених зовнішніми факторами або конфліктами за спільні ресурси.

Подальше вимірювання проводились з інтервалом 1000 мкс у декількох режимах навантаження:

- **Idle** — система без додаткового навантаження;
- **Load** — навантаження на підсистему вводу-виводу;
- **Load Strong** — інтенсивне навантаження, згенероване утилітою `stress-ng` (стресори vm та memcpy), що створює конкуренцію за кеш другого рівня (L2) та пропускну здатність шини пам'яті.

Для кількісної оцінки стабільності введено метрику штрафу хвоста розподілу — **Tail Penalty** (TP). Вона показує, у скільки разів найгірша зафіксована затримка перевищує середнє значення, характеризуючи ступінь варіативності системи [93]. Також розраховується **Determinism Score** (DS) — нормалізований показник передбачуваності (від 0 до 1), де 1 відповідає ідеальній детермінованості.

$$TP = \frac{Lat_{max}}{Lat_{avg}}, \tag{2.3}$$

де Lat_{max} — максимальна зафіксована затримка, а Lat_{avg} — середнє значення.

Додатково введено показник детермінованості:

$$DS = \frac{1000}{1000 + \sigma}, \tag{2.4}$$

де σ — стандартне відхилення затримки.

Сценарій **load**, відображений у Табл. 2.8, демонструє значне зростання максимальної затримки до 950 мкс (майже 1 мс) . Аналіз лічильників переривань (*/proc/interrupts*) виявив 85 148 подій від драйвера `dw-mci` (контролер SD-карти)

Таблиця 2.8

Метрики затримки у різних режимах

Режим	Min (мкс)	Avg (мкс)	Max (мкс)	σ	TP	DS	P_99
Idle	11.0	12.30	107	0.97	8.7	0.999	15
Load	17.0	35.81	950	21.66	26.5	0.978	88
Load Strong	12.0	19.17	544	2.82	28.38	0.994	25

під час тесту. Обробка цих переривань у контексті ядра блокує таймерні події userspace, що емпірично підтверджує тезу п. 1.6 про вразливість неізольованих ядер до переривань від периферії («шумних сусідів»). Той факт, що сценарій I/O-навантаження (**Load**) генерує майже вдвічі більші пікові затримки (950 мкс), ніж інтенсивний обчислювальний стрес (544 мкс), підтверджує, що для платформ SoC FPGA критичним фактором детермінізму є не так завантаженість обчислювальних ядер, як архітектура обробки переривань від периферії. Це обґрунтовує пріоритетність застосування механізму IRQ Shielding над простим резервуванням процесорного часу. Аномальні затримки викликані комбінацією двох факторів: конкуренцією за шину пам'яті під час DMA-транзакцій та, більшою мірою, перериваннями контролера (hard IRQ), які оброблялися на тому ж ядрі, що і вимірювальний потік. Ефективність режиму **B** (Managed) доводить, що перенесення обробки переривань на виділене ядро (*IRQ Shielding*) є основним фактором стабілізації. [94]. Показник **TP** у сценарії **load_strong** (41.45) вказує на значний розрив між типовою та піковою поведінкою, що є ознакою прихованих збоїв кешування.

Окрім абсолютних значень (Lat_{max}), для аналізу стабільності розраховується 99-й перцентиль (P_{99}). Ця метрика відсікає 1% найгірших випадків і показує затримку для основної маси подій. Порівняння P_{99} з абсолютним максимумом (Lat_{max}) дозволяє виявити природу збоїв: якщо $Lat_{\max} \approx P_{99}$, розподіл є щільним і передбачуваним. У випадку, коли $Lat_{\max} \gg P_{99}$ (що продемонстровано далі на Рис. 2.9), це свідчить про наявність аномальних викидів (збоїв), спричинених зовнішніми факторами.

2.9.3. Перевірка керованості (A/B експеримент)

Якщо попередній етап (Табл. 2.8) мав на меті визначення граничних можливостей платформи та виявлення причин прихованих збоїв при екстремальних навантаженнях, то метою наступного експерименту є

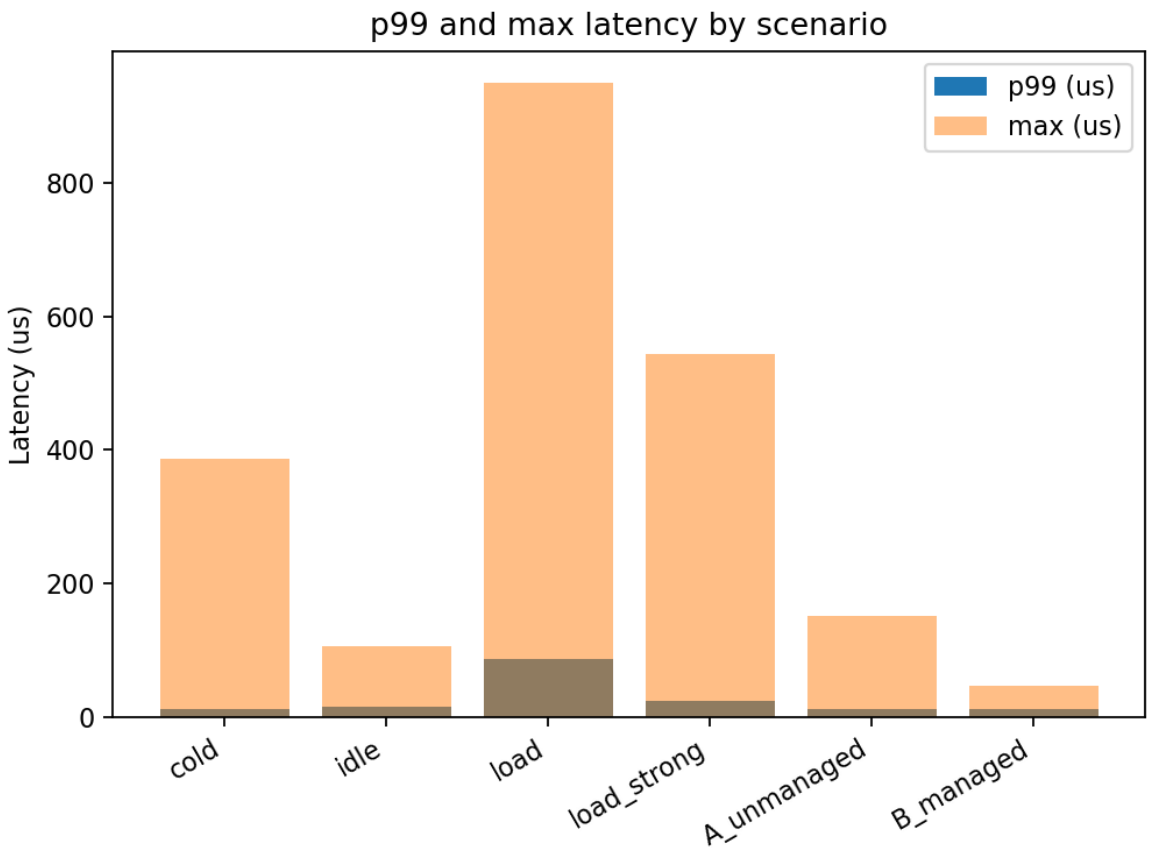


Рис. 2.9: Порівняння значень затримки (P_{99} та Max) у синтетичних тестах та під час A/B експерименту. Режим **A** відповідає стандартному розподілу процесів (Unmanaged), режим **B** — керованій ізоляції ядер (Managed) згідно із запропонованою методикою.

підтвердження ефективності запропонованої методики ізоляції в умовах штатного функціонування вузла Для експериментальної перевірки керованості платформи проведено порівняльне A/B тестування в умовах номінального експлуатаційного навантаження (фонова активність сервісів ОС, мережевий стек, запис логів), ідентичного для обох режимів:

- **Режим A (Unmanaged)**: стандартна конфігурація планувальника Linux та розподілу переривань;
- **Режим B (Managed)**: застосовано методи ізоляції:
  - *CPU Affinity*: процес реального часу жорстко прив'язано до CPU1;
  - *IRQ Shielding*: усі системні переривання примусово перенесено на CPU0;
  - *Scheduler*: процесу надано пріоритет `SCHED_FIFO`.

Застосовані механізми розподілу переривань та прив'язки процесів забезпечують мінімізацію затримки, підтверджуючи оптимізаційну модель

програмно-апаратної взаємодії [21], а стратегія просторового розділення ресурсів (`Spatial Isolation`) між задачами загального призначення та процесами реального часу дозволяє мінімізувати взаємоблокування на спільних ресурсах без необхідності використання гіпервізора [95]

Таблиця 2.9

Результати A/B тестування у мкс

Метрика	A	B	Покращення
$Lat_{\max}$	151	48	$\times 3.15$
Lat_{avg}	12.12	11.98	≈ 0
σ	1.09	0.74	$\times 1.47$
Tail Penalty (**TP**)	12.46	4.01	$\times 3.1$
P_{99}	13	13	0

Результати наведені у Табл. 2.9 показують, що в стандартному режимі (**A**) фонова активність ядра та сервісів створює затримки до 151 мкс, що може бути критичним для задач керування. Застосування методики ізоляції (Режим **B**) в тих самих умовах дозволило знизити максимальну затримку до 48 мкс (покращення у 3.15 рази), нівелюючи вплив системного шуму.

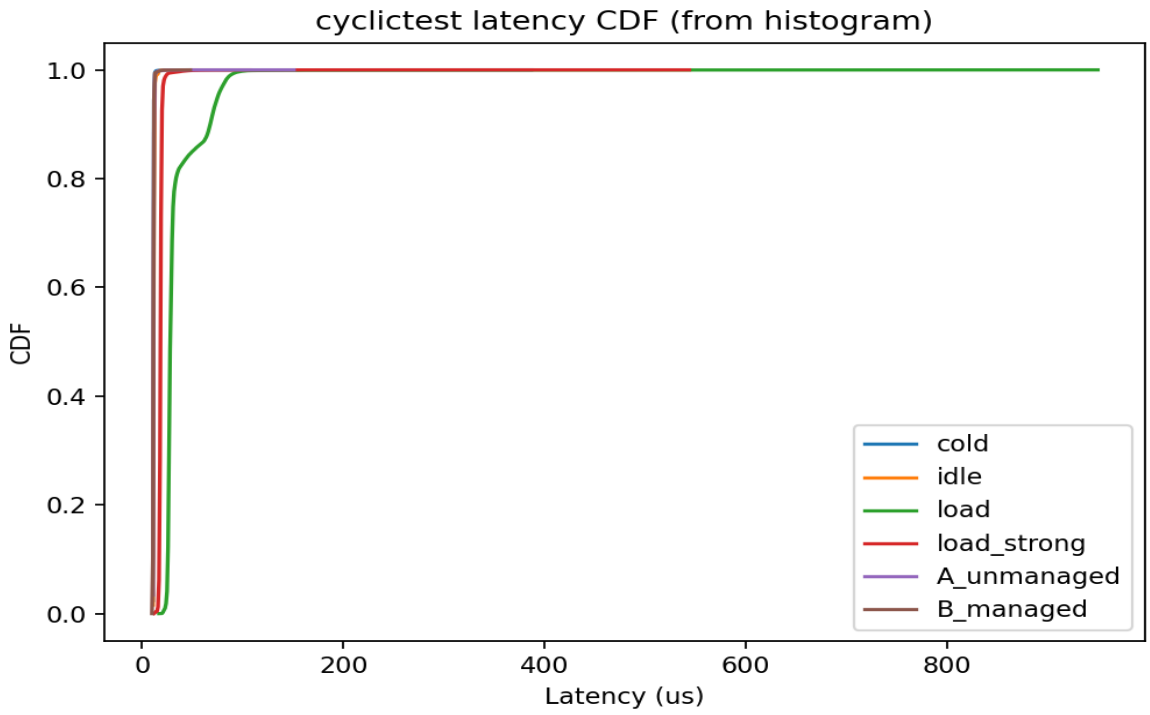


Рис. 2.10: Кумулятивна функція розподілу (CDF) затримок. Крива B_managed демонструє значне звуження розподілу порівняно з A_unmanaged та наближення до характеристик бажаної системи.

Середня затримка (Lat_{avg}) залишається майже незмінною в обох режимах, що підтверджує: запропонована методика не прискорює обчислення, а стабілізує їх, усуваючи аномальні стрибки. Кумулятивна функція розподілу (CDF), наведена на Рис. 2.10, демонструє зміну форми розподілу: крива для режиму B стає майже вертикальною, що свідчить про «колапс варіативності» та відновлення детермінованості.

Отримані результати підтверджують, що застосування стандартних механізмів Linux дозволяє суттєво знизити граничні затримки навіть без використання патчів реального часу.

Для підтвердження придатності платформи до використання у вбудованих системах з обмеженими ресурсами (Edge), проведено аудит споживання пам'яті та дискового простору.

- використання RAM у стані спокою — 55 МБ з 1.0 ГБ доступних.;
- кількість активних systemd-сервісів — 18;
- розмір root filesystem — 848 МБ (19% від 4.7 ГБ розділу).

Фінальний розмір образу (848 МБ) та споживання оперативної пам'яті (55 МБ) підтверджують ефективність концепції «м'якої спеціалізації». Система містить повноцінний стек Linux (Network, SSH, Python, C), залишаючись при цьому достатньо компактною для вбудованих застосувань. Отриманий дистрибутив може бути додатково мінімізований на етапі фіналізації (наприклад, шляхом видалення невикористовуваних заголовків, модулів та пакетів).

Запропонована методика комплексної валідації дозволяє експериментально підтвердити керованість, відтворюваність та часову передбачуваність Linux-платформи на базі SoC FPGA. Застосовані механізми ізоляції дозволяють реалізувати архітектурні принципи систем змішаної критичності (**Mixed Criticality Systems**), дозволяючи співіснування задач реального часу та сервісів загального призначення без використання гіпервізора та `PREEMPT_RT` [61, 95]. Показано, що архітектурні методи ізоляції ресурсів дозволяють досягти квазі-реального часу без модифікації ядра, що створює надійну основу для побудови прикладних систем цифрової обробки сигналів, а досягнута стабільність є критичною для автономних систем збору даних [17].

2.10. Висновки до розділу 2

У другому розділі розв'язано важливу науково-технічну задачу розробки архітектури та методології побудови спеціалізованого програмно-апаратного керуючого середовища на базі гетерогенних систем на кристалі (SoC FPGA). Доведено, що для забезпечення необхідного рівня часового детермінізму **вузлів цифрової обробки сигналів** у телекомунікаційних мережах доцільним є використання підходу повної спеціалізації (*Full Custom*). На відміну від універсальних дистрибутивів або важких фреймворків автоматизації (наприклад, Yocto), такий підхід дозволяє формувати адаптивні керуючі середовища з мінімальними накладними витратами.

Ключовим здобутком розділу є розроблення та впровадження концепції «Паспорта конфігурації» для телекомунікаційних вузлів. Запропонований метод базується на строгій математичній моделі простору станів вектора $S_{\text{passport}} = \{H, M, R\}$, яка формалізує апаратні інваріанти, параметри пам'яті та політики виконання операційної системи. Це дозволило перетворити традиційно неконтрольоване програмне середовище ОС Linux загального призначення на детермінований об'єкт системи, узгоджений із часовими характеристиками радіотехнічного тракту. Як наслідок, систематичну похибку вимірювань часових затримок зведено до нуля ($f(S_{\text{hidden}}) \to 0$).

Ефективність запропонованих рішень експериментально підтверджено шляхом застосування методики просторової ізоляції обчислювальних ресурсів (*Spatial Isolation*). Результати A/B тестування продемонстрували колапс варіативності системних затримок: максимальна латентність зменшилася у понад 3 рази (з 151 мкс до 48 мкс), повністю нівелюючи вплив системного фазового шуму. У контексті гетерогенних систем це гарантує надійну фіксацію станів арбітражу системних шин та усуває апаратно-програмний джитер при транзакціях прямого доступу до пам'яті (DMA).

На основі отриманих результатів розроблено архітектуру гетерогенного вузла, яка базується на чіткому просторовому та логічному розмежуванні контуру керування (*Control Plane*) та тракту обробки даних (*Data Plane*). Локалізація джерел фазової невизначеності на рівні операційної системи підготувала надійну метрологічну та архітектурну базу для інтеграції високошвидкісних апаратних трактів потокової обробки даних, практичну реалізацію та експериментальне дослідження яких наведено у наступному розділі.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ВУЗЛІВ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ НА БАЗІ SOC FPGA

3.1. Програмно-апаратна платформа

У попередньому розділі було сформовано керовану та відтворювану вбудовану операційну систему на базі Linux, оптимізовану для використання у складі SoC FPGA. Наступним логічним кроком є перетворення цієї системи на повноцінну програмно-апаратну платформу, здатну інтегрувати різнорідні обчислювальні потоки та периферійні пристрої в межах єдиного вузла цифрової обробки сигналів. У цьому контексті SoC FPGA розглядається не як окрема плата з операційною системою, а як інтеграційний вузол, у якому поєднуються високошвидкісні потоки даних та логіка керування, що функціонують за різними часовими та функціональними вимогами.

Подібний підхід до трактування операційної системи як інтеграційного шару між апаратним та прикладним доменами вже застосовувався при розробці експериментальних вузлів обробки сигналів і автономних вимірювальних систем, описаних у [12, 16]. У цих роботах було показано, що ключова складність полягає не лише у реалізації апаратних прискорювачів, а й у забезпеченні узгодженої взаємодії між програмними сервісами та апаратними потоками даних.

Архітектурною основою розробленої платформи є уніфікована модель взаємодії між апаратною та програмною частинами, яка залишається інваріантною незалежно від типу підключених джерел даних. Як для потокових сценаріїв (наприклад, обробки I/Q семплів у SDR-системах), так і для подієвих сценаріїв (сенсори, телеметрія, керуючі сигнали), передача даних до програмного домену здійснюється через стандартні інтерфейси сімейства AXI з використанням механізмів прямого доступу до пам’яті (DMA). Такий підхід дозволяє мінімізувати участь центрального процесора у критичних ділянках конвеєра та гарантувати передбачувану часову поведінку системи [96].

Ключовим наслідком цього рішення є чіткий поділ відповідальності між двома логічними площинами платформи. Площина обробки даних (*Data Plane*),

що реалізується апаратними блоками FPGA, контролерами DMA та мінімальним драйверним кодом, орієнтована на забезпечення максимальної пропускної здатності та мінімізації затримок. Натомість площина керування (*Control Plane*), що функціонує у користувацькому просторі Linux, відповідає за конфігурацію, оркестрацію, мережеву взаємодію та збереження результатів. Практична ефективність такого поділу була підтверджена при реалізації автономних вимірювальних систем, у яких часово-критичні обчислення ізольовані від сервісної логіки керування [18, 21].

Важливо підкреслити, що запропонована модель інтеграції не прив'язана до конкретного типу сигналу або фізичного інтерфейсу. З позиції ОС та сервісів, усі вхідні дані (від I/Q семплів до телеметрії) уніфіковані у вигляді буферів у спільному адресному просторі. На фізичному рівні взаємодію з ними забезпечують канали DMA, ініційовані або інтегрованими контролерами HPS (наприклад, USB у режимі Bus Mastering), або синтезованими майстрами шини у FPGA матриці [97]. Це дозволяє уніфікувати програмну модель роботи з даними (*CMA*, *Zero-Copy*) незалежно від фізичного джерела сигналу. Візуалізація запропонованої архітектури, що демонструє розмежування потоків даних та логіки керування через спільний буфер пам'яті, наведена на Рис. 3.1

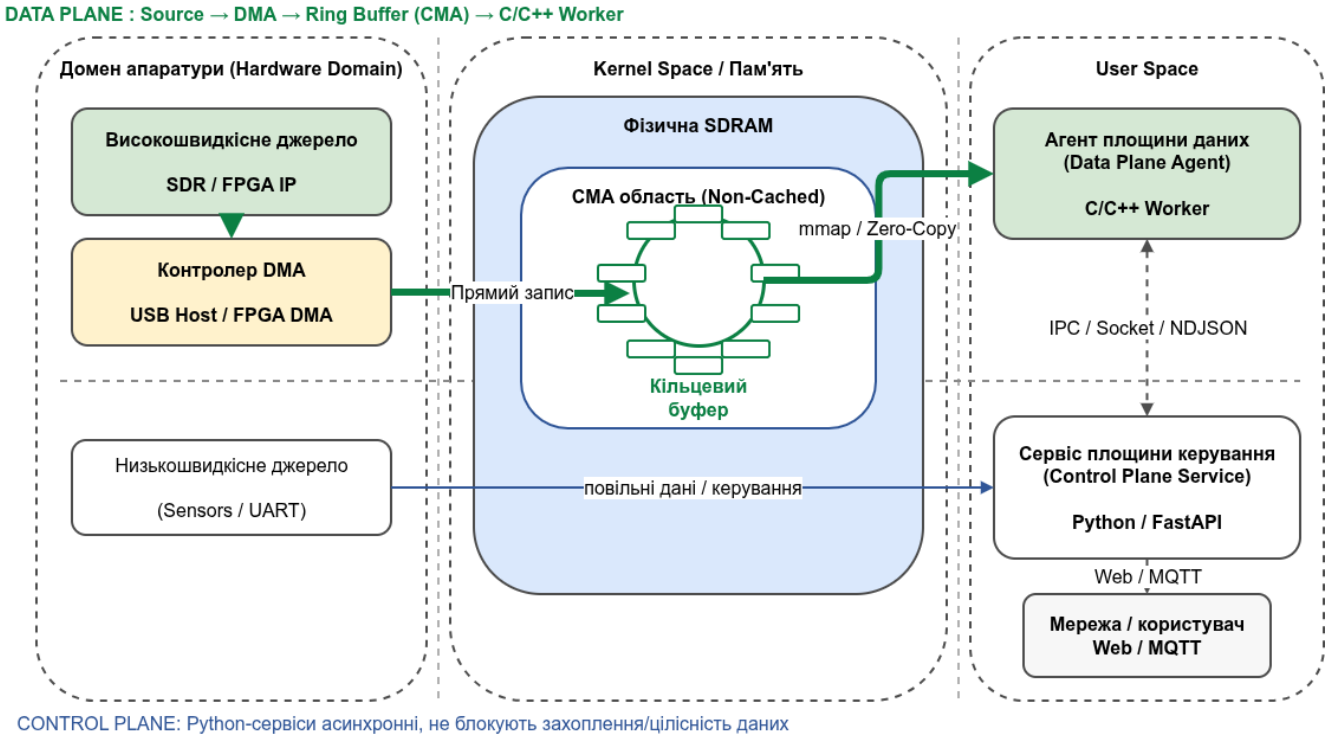


Рис. 3.1: Уніфікована модель циркуляції даних у програмно-апаратній системі.

Такий підхід дозволяє розглядати платформу як агностичну до природи

даних, що суттєво спрощує адаптацію архітектури до нових прикладних задач. У цьому сенсі розроблена платформа не є «спеціалізованою» під конкретний прилад, а виступає узагальненим вузлом гетерогенної обробки даних, де FPGA — як джерело потоку (Streaming Master), а Linux — як споживач та арбітр. Зміна прикладного сценарію — наприклад, перехід від спектрального аналізу до задач просторової пеленгації або сенсорного моніторингу — не потребує перегляду базових механізмів взаємодії між програмним та апаратним доменами. Модифікації локалізуються на рівні прикладної логіки та конфігурації сервісів, тоді як архітектурні інваріанти залишаються незмінними. Подібна властивість є принципово важливою для таких платформ, котрі еволюціонують упродовж тривалого часу та використовуються для вирішення різних класів задач.

Окремо слід відзначити, що чіткий поділ між площинами *Data Plane* та *Control Plane* дозволяє формалізувати межі відповідальності між доменами, що узгоджується з сучасними підходами до побудови реконфігурованих радіосистем [98]. Апаратна частина платформи, що охоплює FPGA-матрицю та швидкісні периферійні контролери, забезпечує детермінований потік даних. Його часові характеристики ізольовані від флуктуацій прикладної логіки завдяки апаратній буферизації та оптимізованим у розділі 2 механізмам переривань. На фізичному рівні така незалежність досягається через виділені канали DMA та механізм *CMA*, що виключає потребу в копіюванні даних (*Zero-Copy*) центральним процесором.

Програмні сервіси керування натомість працюють у режимі «найкращого зусилля» (best-effort). Вони відповідають за конфігурацію та візуалізацію даних, не втручаючись у критичні часові цикли обробки. Такий підхід розв’язує типову проблему вбудованих систем, де складність програмного забезпечення часто дестабілізує роботу апаратної частини.

З наукової точки зору, запропонована модель інтеграції формує основу для порівняльного аналізу різних апаратних реалізацій у межах одного програмного середовища. Подібна гетерогенна архітектура демонструє свою ефективність не лише в телекомунікаціях, а й у суміжних високотехнологічних сферах, зокрема в біомедичній інженерії [99]. Оскільки операційна система, сервісна інфраструктура та методика валідації залишаються сталими, зміни у характеристиках системи можуть бути однозначно співвіднесені з модифікаціями апаратної частини або алгоритмів обробки сигналів. Це створює умови для

коректного експериментального дослідження впливу архітектурних рішень FPGA на загальну поведінку системи, що є ключовою вимогою для подальших розділів роботи.

Отже, у підрозділі 3.1 запропоновано не просто опис платформи, а узагальнену інженерну модель інтеграції гетерогенних потоків. Що, в свою чергу, дає змогу трактувати прикладні системи з підрозділів 3.2 та 3.3 як варіації єдиного архітектурного підходу.

Застосування мов програмування та фреймворків високого рівня (Python, FastAPI, асинхронні мережеві інтерфейси) стало можливим завдяки оптимізації ядра та мінімізації користувацького простору, виконаним у розділі 2. У даній моделі платформа отримує достатній обчислювальний резерв для використання високорівневих інструментів керування навіть на обмежених процесорах класу ARM Cortex-A9. Подібний сервісно-орієнтований підхід до побудови вбудованих систем відповідає сучасним тенденціям розвитку edge-платформ [1, 2].

Важливою властивістю розробленої архітектури є її автономність. Платформа проектується з урахуванням можливості повної втрати мережевого з'єднання без порушення основної функціональності. Для цього використовуються локальні сховища даних, локальні брокери повідомлень та незалежні механізми запуску прикладних сервісів. Подібна модель є критичною для edge-сценаріїв, де вузол цифрової обробки сигналів повинен функціонувати як самодостатній елемент інфраструктури [1]. Зазначені вимоги до автономності відповідають практичним сценаріям експлуатації платформ, реалізованих у попередніх роботах автора [12, 17].

Окрему увагу в межах цієї платформи приділено методиці валідації та забезпечення відтворюваності конфігурації. Перед розгортанням будь-якого прикладного програмного забезпечення виконується перевірка відповідності системи формалізованому «паспорту конфігурації», описаному в розділі 2.8. До такого паспорта входять версії ядра Linux, хеш-суми файлів `Device Tree`, параметри підсистеми пам'яті та політики керування перериваннями. У разі невідповідності будь-якого з ключових параметрів конфігурація вважається невалідним. Подібний підхід до фіксації стану платформи є необхідною умовою забезпечення відтворюваності результатів у складних гетерогенних системах [59, 100] і вже застосовувався при аналізі стабільності [16]. Практична реалізація процедури контролю цілісності середовища виконання наведена на Рис. 3.2.

Алгоритм гарантує, що запуск прикладних сервісів (SDR-сканера або сенсорного вузла) відбувається виключно за умови повної відповідності поточних параметрів ядра та апаратних налаштувань затвердженому ”Паспорту конфігурації”.

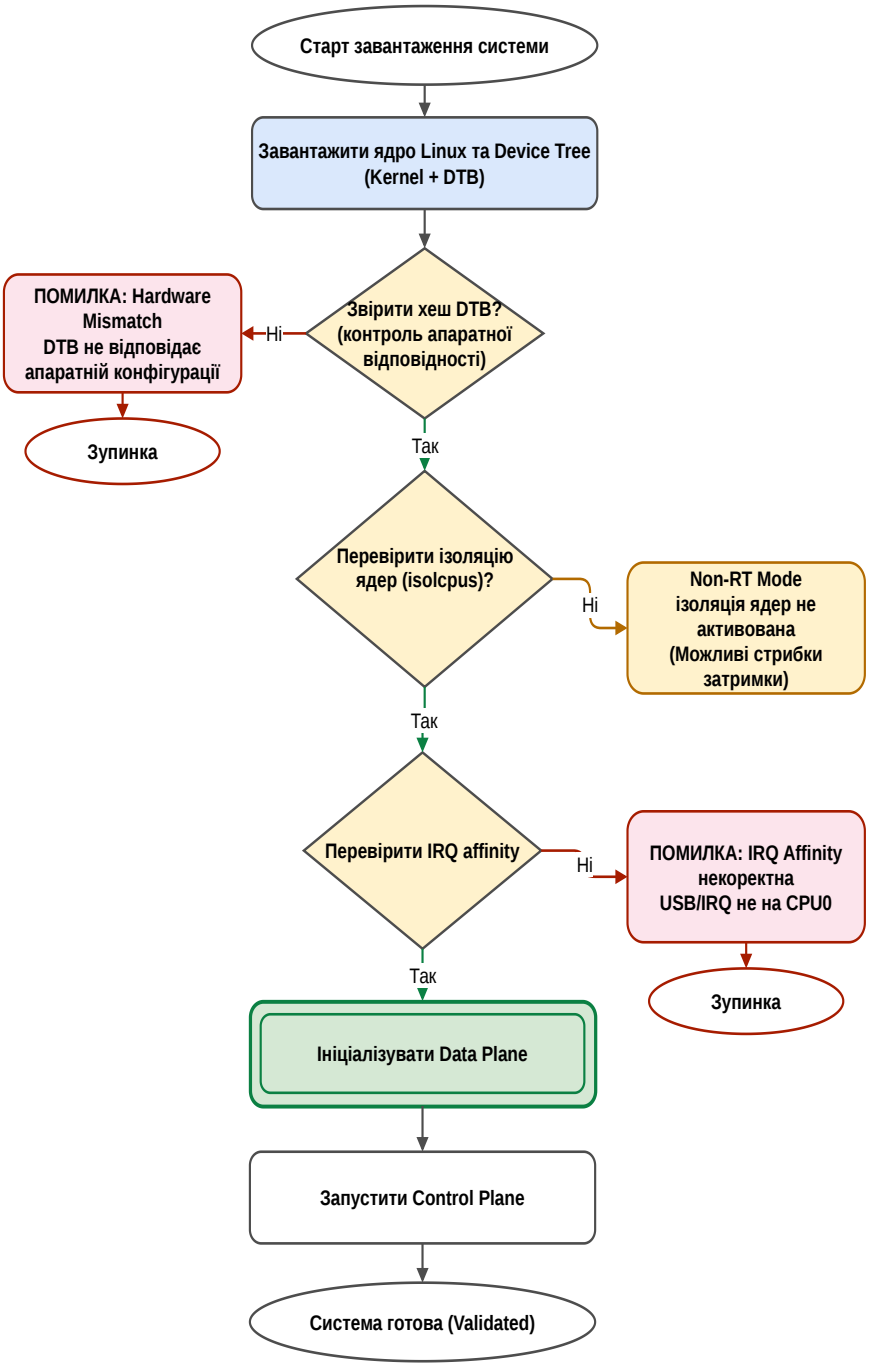


Рис. 3.2: Алгоритм автоматизованої ініціалізації та валідації платформи. Процедура забезпечує відповідність стану системи (ізоляція ядер, маски переривань, дерево пристроїв) вимогам Паспорта конфігурації перед запуском експериментального навантаження.

Таким чином, у підрозділі сформовано уніфіковану модель інтеграції

гетерогенних обчислювальних потоків, у якій Linux використовується не як середовище обробки сигналів, а як координуючий шар для керування ресурсами, сервісами та життєвим циклом даних. Реалізований поділ на *Data Plane* та *Control Plane* дозволяє поєднати високу продуктивність апаратних прискорювачів із гнучкістю та керованістю сучасних програмних інструментів, що створює основу для прикладних систем, розглянутих у наступних підрозділах.

3.2. Реалізація системи потокового спектрального моніторингу

На відміну від подієвих або сенсорних систем, характерних для значної частини *edge*-застосувань, SDR-сканер генерує безперервний високошвидкісний потік даних у вигляді комплексних I/Q-семплів з фіксованою частотою дискретизації до 20 MSPS. Такий режим роботи створює постійне навантаження на підсистему пам’яті, механізми прямого доступу до пам’яті (DMA) та систему переривань, що робить його репрезентативним прикладом для аналізу граничних режимів роботи програмно-апаратної платформи.

Подібний підхід до використання спектрального моніторингу як автономного вузла обробки даних раніше застосовувався у попередніх роботах автора, де було показано практичну придатність програмно-апаратної архітектури для тривалої потокової обробки сигналів [12, 18, 19].

Варто окремо підкреслити, що у даній реалізації як фізичний інтерфейс джерела даних використовується USB 2.0 у режимі *High Speed* (480 Мбіт/с), який є значно складнішим з точки зору програмної обробки порівняно з внутрішніми шинами SoC FPGA. На відміну від *AXI-Stream* або *FPGA-DMA*, де транзакції виконуються апаратно з мінімальною участю процесора, USB-взаємодія потребує обробки протокольних структур (англ. *USB Request Blocks*), частих переривань хост-контролера та участі програмного стеку ядра Linux. Таким чином, використання USB-інтерфейсу формує свідомо «найгірший сценарій» з точки зору навантаження на центральний процесор. Якщо запропонована архітектура здатна забезпечити стабільну потокову обробку даних у таких умовах, це автоматично гарантує її функціональність при роботі з внутрішніми апаратними прискорювачами FPGA, де накладні витрати на транзакції є на порядок нижчими.

3.2.1. Апаратна реалізація системи

З метою практичної імплементації розробленої архітектури та забезпечення функціонування системи в реальному часі, створено апаратний комплекс, побудований за модульним принципом. Апаратна реалізація вузла моніторингу наведена на Рис. 3.3. Архітектура платформи є гетерогенною і включає радіочастотний тракт, обчислювальне ядро на базі SoC FPGA та набір допоміжних інтерфейсів.

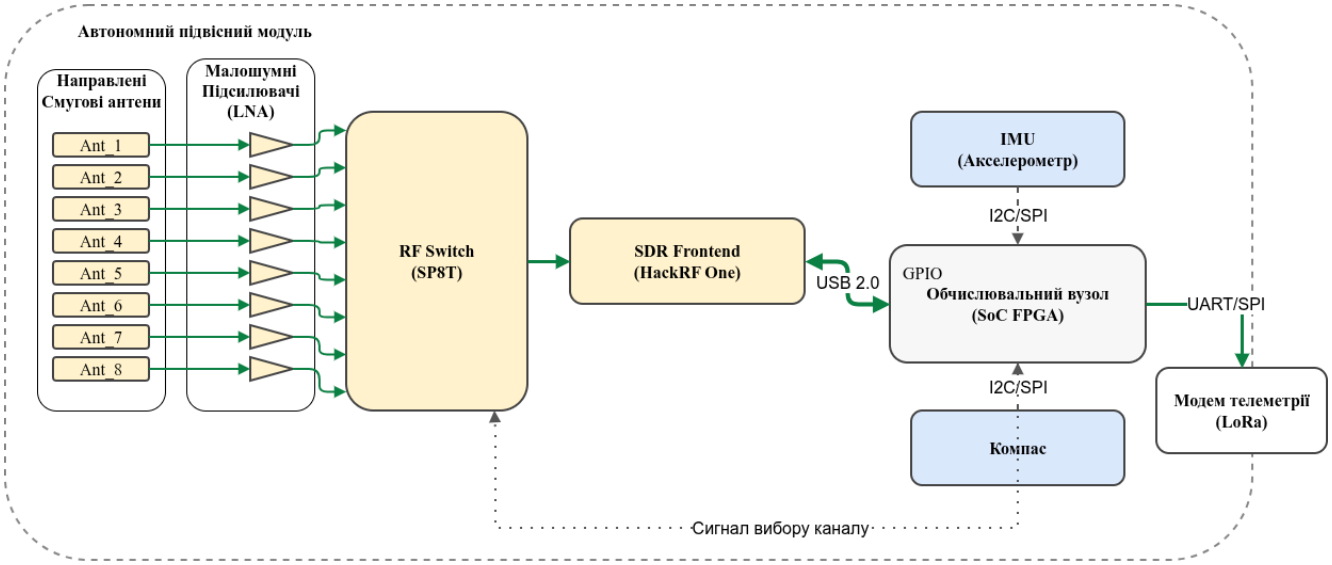


Рис. 3.3: Структурна схема апаратної частини автономного вузла спектрального моніторингу. Кольорове кодування відповідає функціональним доменам: бежевий — радіочастотний тракт, сірий — обчислювальне ядро, блакитний — сенсори.

Для забезпечення необхідної чутливості та вибірковості приймального тракту, перед входом SDR-трансивера HackRF One інтегровано систему попередньої обробки сигналу з наступними характеристиками:

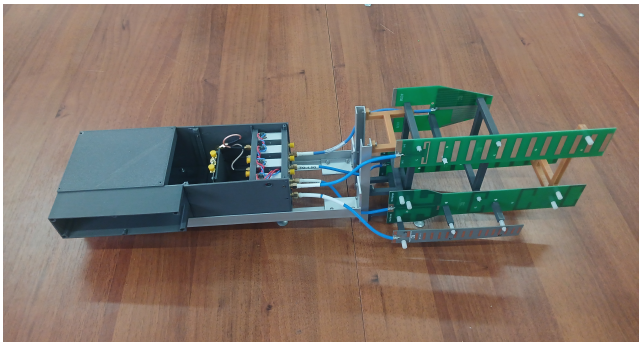
- **Малошумний підсилювач (LNA):** Використано широкосмуговий модуль на базі напівпровідникових елементів SiGe, що забезпечує коефіцієнт підсилення $G \approx 10$ дБ у смузі частот 10 МГц – 6 ГГц. Ключовим параметром є низький коефіцієнт власного шуму (Noise Figure, $NF < 5$ дБ), що дозволяє детектувати слабкі сигнали на рівні теплових шумів. Енергоспоживання модуля становить менше 100 мА, що відповідає вимогам до автономних edge-вузлів.
- **Комутація антен (RF Switch):** Оскільки SDR-приймач має єдиний вхідний тракт, для роботи з різними діапазонами застосовано високочастотний перемикач на базі чіпа **HMC7992** (SP4T). Він забезпечує

ізоляцію каналів на рівні 45 дБ із внесеними втратами не більше 0.6 дБ, що мінімізує спотворення вимірювань.

- **Антенна система:** Застосовано набір друкованих логоперіодичних антен для діапазонів 433/900/2400 МГц. Вузька діаграма спрямованості (горизонтальний кут розкриву $\pm 15°$) дозволяє реалізувати функцію просторової селекції джерел випромінювання.
- **Підсистема просторової орієнтації:** Для прив'язки спектральних вимірювань до просторових координат використано модуль IMU на базі сенсора **MPU-9250** (акселерометр, гіроскоп, магнітометр). Сенсор підключено безпосередньо до шини I2C процесорної системи HPS (частота шини 400 кГц), що дозволяє опитувати стан орієнтації синхронно з захопленням радіокадрів.

Загальний вигляд апаратної реалізації, котра об'єднує описані вище компоненти в єдиний вимірювальний вузол, наведено на Рис. 3.4. На зображенні продемонстровано фізичну інтеграцію радіочастотного тракту з обчислювальним ядром на базі SoC FPGA.

(а) Загальний вигляд системи в умовах лабораторії.



(б) Конструктивне виконання вхідного тракту: масив логоперіодичних антен та блок попереднього підсилення.

Рис. 3.4: Апаратна реалізація системи спектрального моніторингу. Зліва (а) показано інтеграцію обчислювального вузла, справа (б) — деталізація радіочастотного інтерфейсу.

Також, конструктивне виконання апаратного блоку SDR-сканера розроблялося з урахуванням вимог до **аеромобільності**. Слід зазначити, що апаратна реалізація вузла спектрального моніторингу та просторової селекції, включаючи розробку антенно-фідерного тракту та інтеграцію SDR-трансивера з обчислювальним

ядром, виконувалася в рамках реалізації грантового проєкту Національного фонду досліджень України (реєстраційний номер 2023.04/0150, «Розробка комплексу для визначення положення та відносної потужності джерел радіовипромінювання та їх візуалізації»). Використання розробленої архітектури дозволило досягти заявлених у проєкті показників продуктивності. На Рис. 3.5 продемонстровано сумісність габаритних характеристик розробленого антенно-фідерного тракту з відсіком корисного навантаження безпілотної платформи.

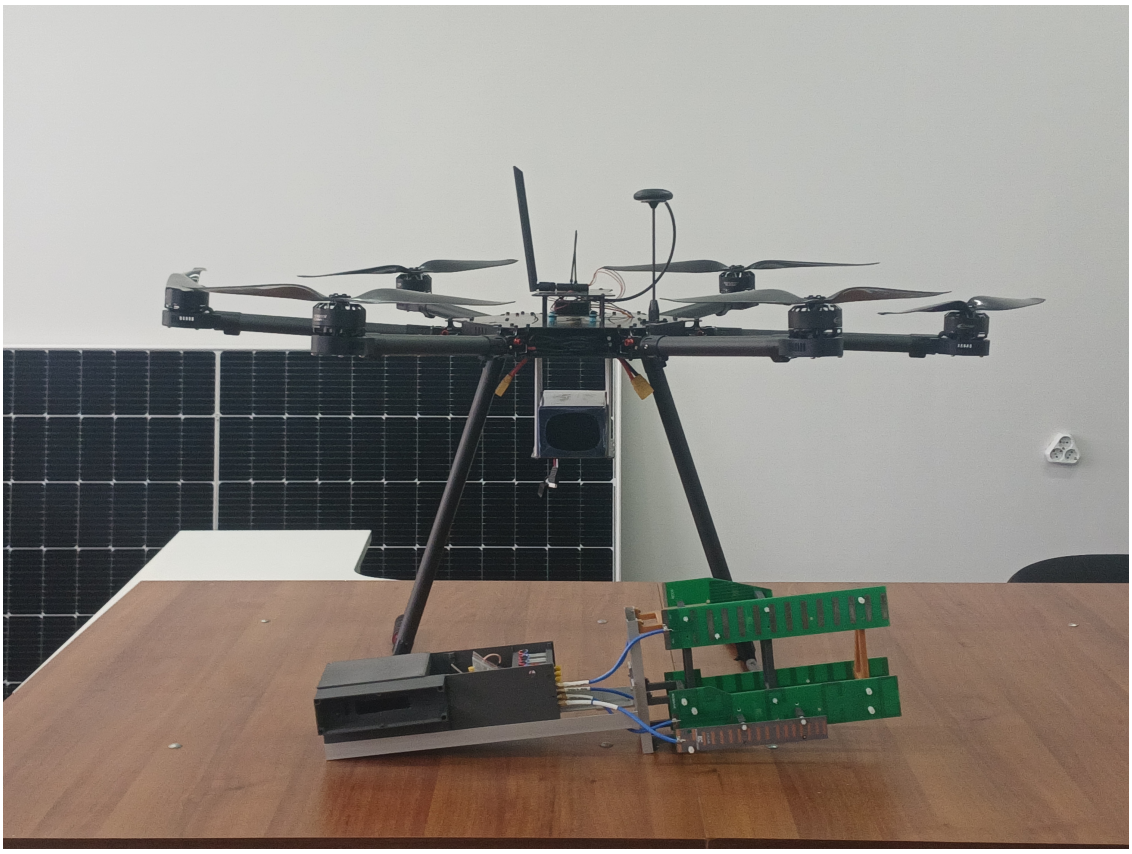


Рис. 3.5: Демонстрація інтеграційного потенціалу розробленого вузла. Апаратний блок SDR-сканера виконано у форм-факторі, сумісному з платформою носія, що обґрунтовує можливість застосування системи в сценаріях аеромоніторингу.

Це підтверджує, що запропонована архітектура здатна до масштабування від прототипу до повноцінного бортового комплексу радіомоніторингу, реалізуючи концепцію уніфікованого реконфігурованого вузла.

3.2.2. Двоетапний алгоритм просторового сканування та адаптивного уточнення напрямку

Багатоканальний антено-фідерний тракт формує набір каналів $c \in \mathcal{C}$, де c — номер каналу (окрема односпрямована антена або окремий частотний піддіапазон

приймання). Під час сканування платформа для кожного напрямку фіксує простий показник наявності випромінювання $P(\varphi, \theta, c)$. Тут φ — азимут (кут повороту у горизонтальній площині), θ — кут місця (кут підняття/нахилу у вертикальній площині), а $P(\cdot)$ — виміряний рівень сигналу або інший скалярний вихід детектора у вибраному каналі c. Значення $P(\cdot)$ обчислюється з усередненням протягом часу t_d, де t_d — тривалість вимірювання в одному кутовому положенні (час інтеграції/усереднення). Метод складається з двох послідовних етапів.

Етап 1 (оглядове кругове сканування): Платформа виконує огляд по азимуту φ на повному інтервалі $[0, 2\pi)$ з дискретним кроком $\Delta\varphi$, де $\Delta\varphi$ задає кутовий крок між сусідніми положеннями під час огляду. У кожному положенні φ_i виконується вимірювання $P(\varphi_i, \theta_0, c)$, де θ_0 — фіксоване положення по куту місця на етапі огляду (або еквівалентне положення антени). Якщо показник $P(\varphi_i, \theta_0, c)$ перевищує задану умову детекції (Перевищує заданий поріг або задовільняє правило підтвердження), поточний азимут вважається кандидатним і позначається як $\varphi^\star$.

Етап 2 (локальне уточнення): Після виявлення кандидатного напрямку $\varphi^\star$ виконується уточнення лише в його околі, щоб не витрачати час на повний обхід усіх напрямків. Спочатку виконується сканування по куту місця θ з кроком $\Delta\theta$, де $\Delta\theta$ задає дискретність уточнення у вертикальній площині. За потреби додатково виконується дрібне уточнення по азимуту в локальному інтервалі $[\varphi^\star - \Delta_{\text{loc}}, \varphi^\star + \Delta_{\text{loc}}]$, де Δ_{loc} визначає напівширину області локального пошуку. Остаточна оцінка напрямку на джерело $(\hat{\varphi}, \hat{\theta})$ визначається як положення в межах локальної області, для якого значення $P(\varphi, \theta, c)$ є максимальним (тобто відповідає найбільшому виміряному рівню/виходу детектора). Даний підхід є універсальним для виявлення високомобільних джерел радіовипромінювання відеосигналів із бортів БПЛА в діапазоні частот від 0 до 6 Ггц. Також, описаний підхід можна адаптувати для стаціонарних джерел.

3.2.3. Програмна архітектура обробки сигналів (Data Plane)

Архітектура підсистеми базується на строгому розділенні відповідальності між агентом площини обробки даних та сервісами площини керування (Рис. 3.6). Агент *Data Plane*, реалізований мовою C, відповідає за прийом потокових даних, виконання спектрального аналізу та формування результатів у режимі жорсткої детермінованості. Його архітектура базується на принципі *Zero-Copy* в

межах простору користувача та мінімізації перемикання контекстів. Локалізація попередньої обробки спектральних даних на вузлі дозволяє уникнути передачі «сирого» потоку через мережу. Буферизація та попередня агрегація даних безпосередньо на edge-пристрої є найбільш ефективним методом мінімізації затримки у розподілених системах моніторингу [101].

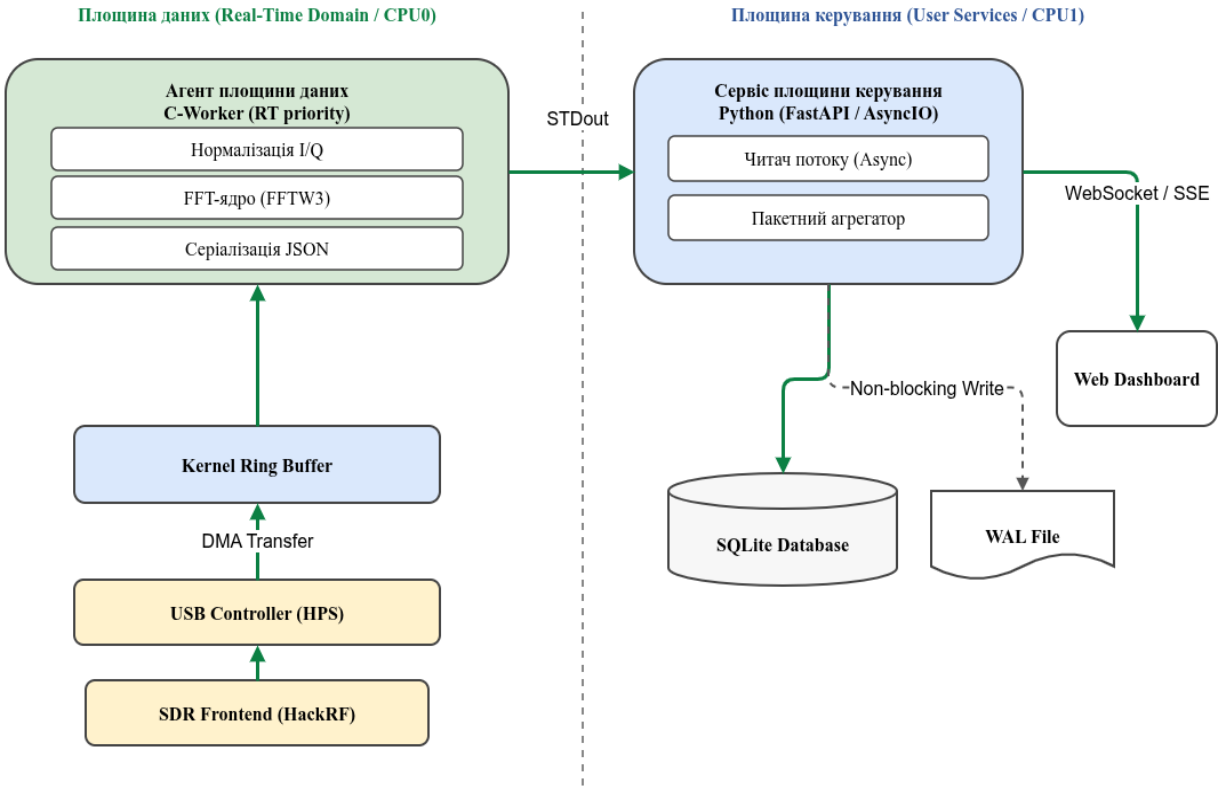


Рис. 3.6: Архітектура потоків даних підсистеми SDR-сканера: розмежування критичного контуру обробки (*Data Plane*) та сервісної логіки (*Control Plane*).

У межах одного циклу обробки для кожного блоку з $N = 4096$ комплексних відліків виконується попереднє віконне згладжування (функція Хеммінга) з метою зменшення спектрального розтікання, після чого застосовується пряме дискретне перетворення Фур'є. Для прискорення обчислень на архітектурі ARM Cortex-A9 використано бібліотеку **FFTW3**, скомпільовану з підтримкою SIMD-інструкцій NEON. Використання режиму планування FFT з прапором `FFTW_ESTIMATE` дозволяє уникнути тривалої та недетермінованої фази вимірювань, що є критичним для забезпечення швидкого «холодного старту» автономних вузлів.

Вибір розміру буфера N визначається компромісом між частотною роздільною здатністю

$$\Delta f = \frac{F_s}{N} = \frac{20 \cdot 10^6}{4096} \approx 4.88 \text{ кГц},$$

та часовою затримкою

$$t_{\text{lat}} = \frac{N}{F_s},$$

де F_s — частота дискретизації вхідного сигналу. Для $F_s = 20$ MSPS це відповідає затримці порядку сотень мікросекунд, що є прийнятним для задач спектрального моніторингу в edge-сценаріях.

Варто зазначити, що теоретична частота генерації кадрів складає ≈ 4882 Гц. Пряма серіалізація такого обсягу даних у текстовий формат NDJSON перевищує пропускну здатність каналу міжпроцесної взаємодії (IPC pipe) та можливості споживача (Python-сервісу).

Для вирішення цієї проблеми в архітектурі *Data Plane* застосовано механізм **неявного керування потоком через зворотний тиск** (англ. *Implicit Backpressure Flow Control*). Оскільки вивід даних у стандартний потік (`stdout`) налаштовано у небуферизованому режимі (`_IONBF`), функція запису блокується при заповненні буфера каналу. Це призводить до тимчасового призупинення потоку виконання функції `rx_callback` (див. лістинг 3.1), що змушує драйвер SDR-приймача автоматично відкидати нові вхідні семпли до моменту звільнення ресурсів.

```
int rx_callback(hackrf_transfer *transfer) {
    //  1. Прямий доступ до буфера DMA без копіювання у черги
    //  Нормалізація вхідних I/Q семплів у буфер FFT
    for (int i = 0; i < nsamp; i++) {
        in[i] = normalize(transfer->buffer[2*i], transfer->buffer[2*i+1]);
    }
    //  2. Виконання математичної обробки "на місці"(у контексті переривання)
    //  Використовується попередньо розрахований план (FFTW_ESTIMATE)
    fftw_execute(p);
    //  3. Серіалізація та небуферизований вивід результату
    //  Блокуючий запис реалізує Implicit Backpressure для адаптивної децимації
    for (int i = 0; i < N; i++) {
        spectra[i] = calculate_magnitude(out[i]);
    }
    emit_spectrum_ndjson(spectra);

    return 0; //  Повернення керування
}
```

Лістинг 3.1: Фрагмент реалізації функції обробки даних (Run-to-Completion)

Такий підхід реалізує **адаптивну часову децимацію** (англ. *Adaptive Time Decimation*) без необхідності введення додаткової логіки лічильників у код. Система працює за принципом "максимальних зусиль"(англ. *best-effort*),

обробляючи та гарантовано доставляючи максимально можливу кількість цілісних спектральних кадрів, яку здатен прийняти *Control Plane*, і пропускаючи проміжні кадри. Це забезпечує математичну коректність кожного окремого збереженого вимірювання при автоматичному узгодженні швидкостей генерації та споживання даних.

Власне, обробка здійснюється безпосередньо у високопріоритетному потоці, за алгоритмом, наведеним на Рис. 3.7.

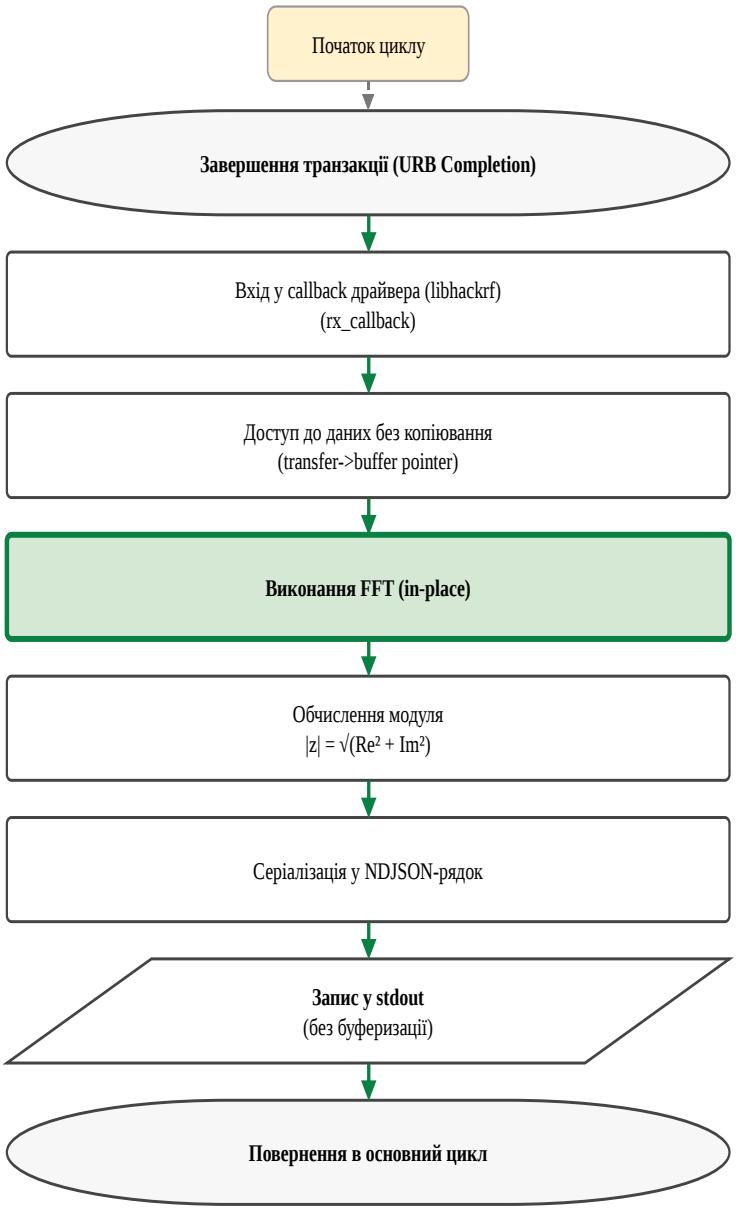


Рис. 3.7: Блок-схема алгоритму обробки даних у циклі *Run-to-Completion*, що демонструє виконання математичних операцій (FFT) безпосередньо в контексті переривання.

Дані, отримані через механізми DMA, безпосередньо передаються у процедуру спектрального аналізу, де над ними одразу виконуються обчислення. Така організація усуває затримки, пов'язані з планувальником операційної системи, та обмежує сумарну затримку виключно часом виконання обчислювального коду на процесорі [102]. Свідомий вибір текстового формату серіалізації NDJSON замість бінарних протоколів (*Protobuf*) обґрунтований вимогою спостережуваності системи на етапі експлуатації. Хоча текстова серіалізація створює додаткове навантаження на CPU, проведені випробування показали, що за умови оптимізованого ядра Linux це навантаження не перевищує можливостей ізольованого процесорного ядра.

3.2.4. Площина керування та результати апробації

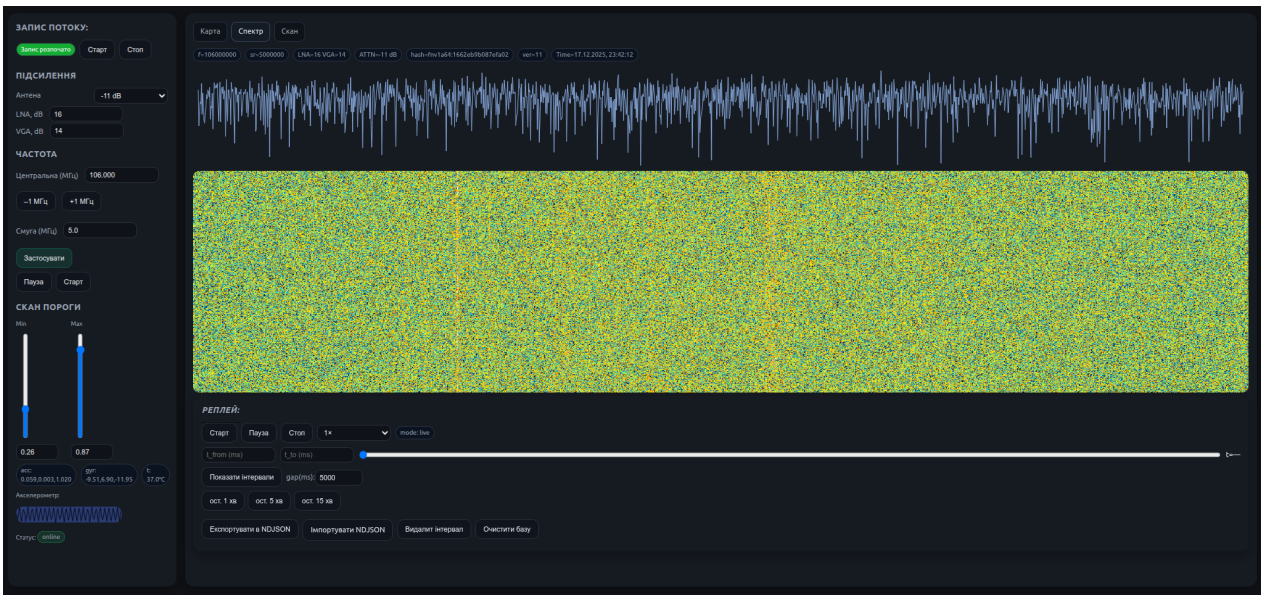
Площина керування (*Control Plane*) реалізована у вигляді асинхронних сервісів користувацького простору Linux із використанням мови Python та фреймворку FastAPI. Для забезпечення неблокуючої роботи підсистеми зберігання використовується режим *Write-Ahead Logging* (WAL) у СУБД SQLite. Цей режим трансформує патерн доступу до пам'яті у лінійний, що дозволяє мінімізувати ефект тимчасового блокування файлової системи та зменшити коефіцієнт підсилення запису (*write amplification*) на флеш-накопичувачах [103]. Окрім забезпечення атомарності транзакцій [104], це є критичним фактором для подовження ресурсу eMMC/SD-носіїв в автономних сканерах.

Оскільки реалізація *Data Plane* не містить внутрішніх механізмів згладжування навантаження, стабільність роботи підсистеми досягається застосуванням методики ізоляції ресурсів (*IRQ Shielding*), описаної у підрозділі 2.9. Згідно з методикою, пріорітетність (*smp_affinity*) переривань USB-контролера налаштована на CPU0, тоді як транзакції бази даних виконуються на іншому ядрі.

Візуалізація роботи системи реалізована через веб-інтерфейс, який функціонує за принципом «тонкого клієнта». На Рис. 3.8 продемонстровано два граничні режими роботи платформи.

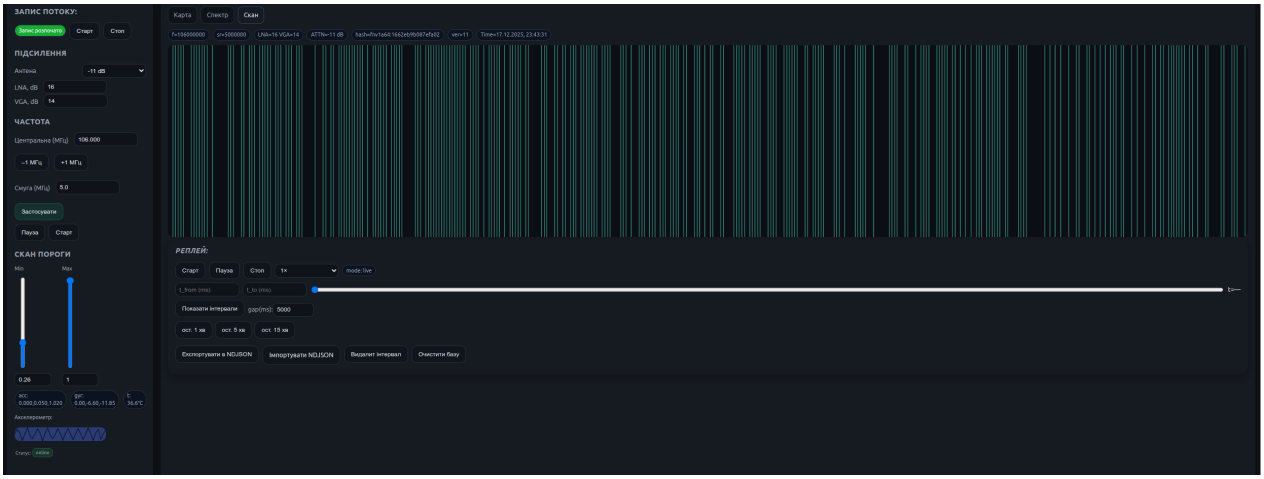
У режимі *Live* (Рис. 3.8а) система обробляє безперервний потік I/Q-даних, що підтверджує пропускну здатність каналу DMA.

У режимі *Scan* (Рис. 3.8б) реалізовано алгоритм швидкого перелаштування частоти, де підсистема *Control Plane* циклічно керує регістрами синтезатора.



(а) Режим *Live Stream*: безперервна візуалізація спектральної щільності у смузі 20 МГц.

Рис. 3.8: Інтерфейс оператора системи спектрального моніторингу. Демонстрація стабільності візуалізації високошвидкісного потоку даних (Data Plane) та керування параметрами приймача (Control Plane).



(б) Режим *Scan*: панорамний огляд широкого діапазону частот.

Подібний підхід до фіксації стану системи та забезпечення відтворюваності результатів узгоджується з сучасними вимогами до модульних архітектур у складних обчислювальних системах [100] і вже застосовувався при аналізі автономних вимірювальних платформ у попередніх роботах автора [16, 17]. Отримані результати підтверджують, що обрана архітектура забезпечує детерміновану обробку сигналів на рівні прикладного програмного забезпечення, що дозволяє використовувати SoC FPGA під управлінням Linux для задач професійного радіомоніторингу.

3.3. Реалізація автономного сенсорного вузла просторового моніторингу

На відміну від задач високошвидкісної обробки потоків (*Data Plane*), розглянутих у підрозділі 3.2, системи просторового моніторингу висувають діаметрально протилежні вимоги до архітектури. Для таких систем типовими є асинхронний характер надходження даних, відносно низька інтенсивність інформаційних потоків (одиниці або десятки кілобайт за секунду) та підвищені вимоги до надійності збереження, цілісності та автономності функціонування. У цьому випадку ключовим фактором стає не максимальна пропускна здатність, а **реактивність** (latency) обробки подій, **енергоефективність** та здатність платформи стабільно працювати в умовах відсутності мережевої інфраструктури [17, 105].

У межах цього підрозділу реалізація автономного сенсорного вузла розглядається як приклад подієво-орієнтованої системи класу *Control Plane*, що дозволяє продемонструвати універсальність розробленої платформи. В якості об’єкта для верифікації обрано задачу дистанційного зондування поверхні за допомогою спеціалізованого мультисенсорного модуля, встановленого на безпілотному літальному апараті (БПЛА). Цей сценарій дозволяє перевірити здатність SoC FPGA керувати гетерогенною периферією в умовах реального польоту та нестабільного каналу зв’язку.

3.3.1. Архітектура HPS-Centric та інтеграція периферії

З огляду на те, що частота опитування сенсорів просторового моніторингу (GPS-приймачі, інерціальні датчики, детектори) зазвичай не перевищує $10 \ldots 100$ Гц, використання програмованої логіки FPGA для обробки таких потоків є архітектурно недоцільним [32]. Натомість у розробленій системі свідомо використовується підхід *HPS-centric*, за якого інтеграція низькошвидкісної периферії здійснюється через апаратні контролери Hard Processor System (UART, USB), керовані стандартними драйверами ядра Linux.

Використання оптимізованої збірки Linux уможливлює виконання попередньої обробки та агрегації даних безпосередньо на процесорних ядрах SoC, звільняючи ресурси програмованої логіки для задач високошвидкісної обробки (як у п. 3.2) [16].

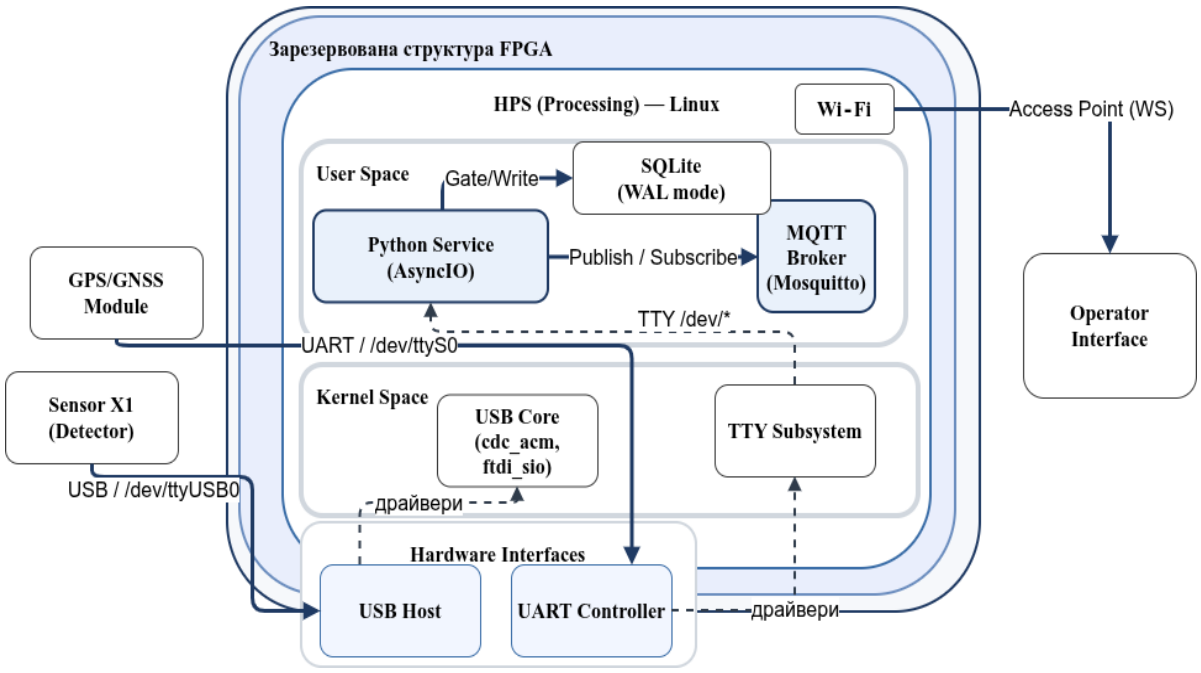


Рис. 3.9: Архітектурна організація автономного сенсорного вузла

Представлена на Рис. 3.9 архітектура є прямою імплементацією моделі розділення площин: логіка прийняття рішень повністю локалізована у програмному домені HPS, тоді як апаратні інтерфейси виконують лише функцію джерела даних.

Такий асиметричний розподіл навантаження відповідає концепції програмно-конфігурованих MPSoC систем, у яких гнучкість Control Plane досягається за рахунок програмної реалізації логіки керування [96]. Стабільна робота високорівневого стеку прикладних сервісів (Python, FastAPI) забезпечується завдяки мінімізації фонових процесів операційної системи та адаптації користувацького простору відповідно до методик побудови вбудованих Linux-систем, описаних у Розділі 2 [13]. Типовий ланцюг обробки даних у такій архітектурі має вигляд:

$$\text{Сенсор} \rightarrow \text{Драйвер Linux (UART/USB)} \rightarrow \text{Сервіс} \rightarrow \text{MQTT Broker} \rightarrow \text{DB/UI}$$

Подібна схема забезпечує чіткий поділ відповідальності між компонентами та узгоджується з сучасними підходами до побудови edge-систем керування подіями [105, 106].

3.3.2. Апаратна організація вимірювального комплексу

Апаратна частина комплексу реалізована у вигляді навісного модуля, інтегрованого з бортовою мережею важкого мультироторного БПЛА. Як носій використано гексакоптер на базі рами **TAROT 960** (вуглецевий композит), оснащений двигунами **T-MOTOR Antigravity MN5008** (340KV) та контролером польоту **Pixhawk 6C**. Вибір платформи зумовлений необхідністю підйому корисного навантаження до 7 кг, що дозволяє розмістити на борту обчислювальний вузол SoC FPGA разом із масивною сенсорною рамкою індукційного детектора.

Рис. 3.10: Зовнішній вигляд мобільного вимірювального комплексу на базі БПЛА TAROT 960. Обчислювальний вузол SoC FPGA інтегровано в бортову мережу живлення та керування.

Ключовим сенсором виступає індукційний детектор, підключений до SoC FPGA через інтерфейс UART (38400 бод, 8N1). Згідно з розробленим протоколом, сенсор передає фрейми даних кожні 40 мс у форматі:

```
*<Mode><Ch1><Ch2><Ch3><Ch4><Voltage><Temp><CRC>
```
(3.1)

де `Ch1...Ch4` — шістнадцяткові значення рівнів сигналу в чотирьох каналах вимірювання.

3.3.3. Алгоритми обробки та забезпечення автономності

Програмна реалізація сервісів керування побудована на подієво-орієнтованій моделі з використанням локального брокера MQTT (Mosquitto) та неблокуючого вводу-виводу (`asyncio`). Сервіс обробки виконує наступні етапи:

1. **Валідація (*CRC Check*):** Відкидання пошкоджених пакетів для забезпечення цілісності телеметрії.
2. **Програмна фільтрація (*Software Gating*):** Замість жорсткої апаратної логіки компараторів, рішення про реєстрацію події приймається на основі аналізу вектора рівнів каналів $L = \{ch_1, \ldots, ch_k\}$ за умовою:

$$D = \begin{cases} 1, & \text{якщо } \exists i : ch_i > T_{\text{th}}, \\ 0, & \text{інакше}, \end{cases}$$

   де T_{th} — адаптивний поріг, що може змінюватися оператором у реальному часі без переривання роботи системи. Це демонструє перевагу використання HPS: зміна порогів детекції не потребує пересинтезу бітстріму FPGA. Більше того, програмна реалізація дозволяє без зміни архітектури імплементувати складні політики детектування, такі як логіка «k-з-n» (*coincidence detection*) або часова гістереза (*debounce*) для фільтрації шуму.
3. **Об'єднання вхідних даних (*Sensor Fusion*):** Асоціація кожного фрейму вимірювань з останніми відомими координатами GPS (принцип "останнього відомого значення") та точною міткою часу.

Ключовою вимогою до системи є її автономність. Архітектура проектується з припущенням повної або часткової відсутності зовнішнього мережевого з'єднання, що є типовим для польових сценаріїв експлуатації. Для забезпечення цієї вимоги застосовується підхід *Store-and-Forward*: усі дані першочергово зберігаються у локальній базі даних з використанням режиму *Write-Ahead Logging* (WAL). Використання журналу WAL гарантує цілісність транзакцій навіть при раптовому вимкненні живлення та трансформує патерн доступу до пам'яті у лінійний, що суттєво зменшує коефіцієнт підсилення запису (*write amplification*) на флеш-накопичувачах [103].

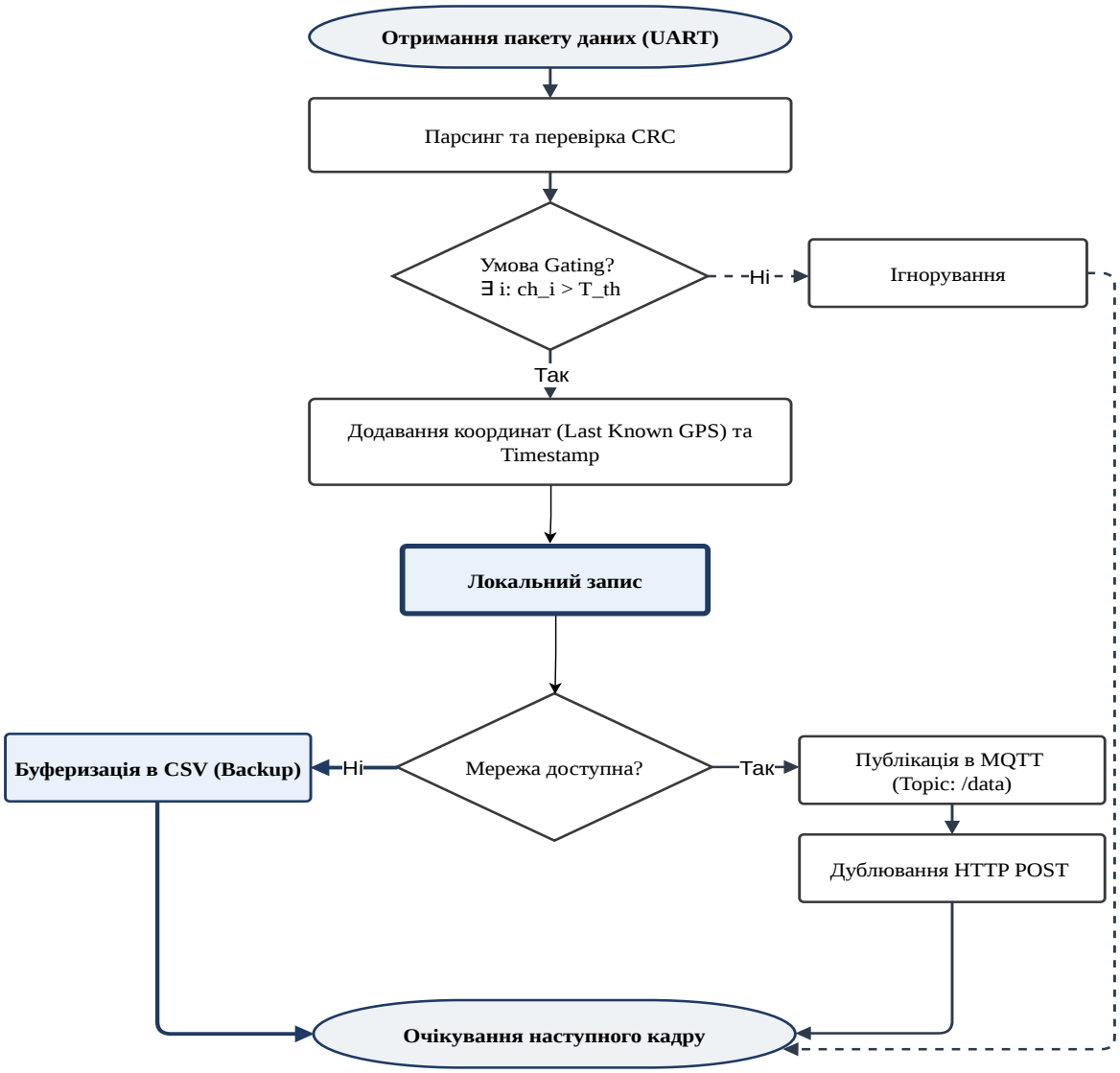


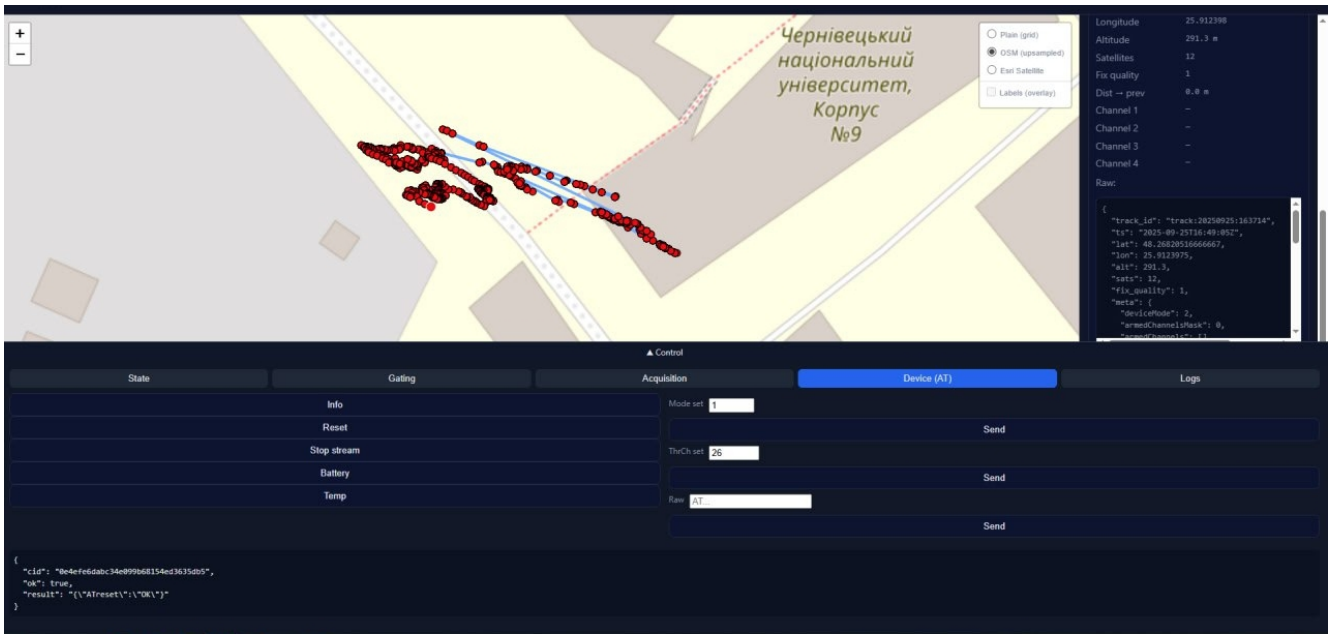
Рис. 3.11: Блок-схема алгоритму функціонування підсистеми Control Plane

Реалізовано багаторівневу стратегію збереження даних (Finite State Machine, Рис. 3.11), яка гарантує цілісність треку навіть при розривах зв’язку.

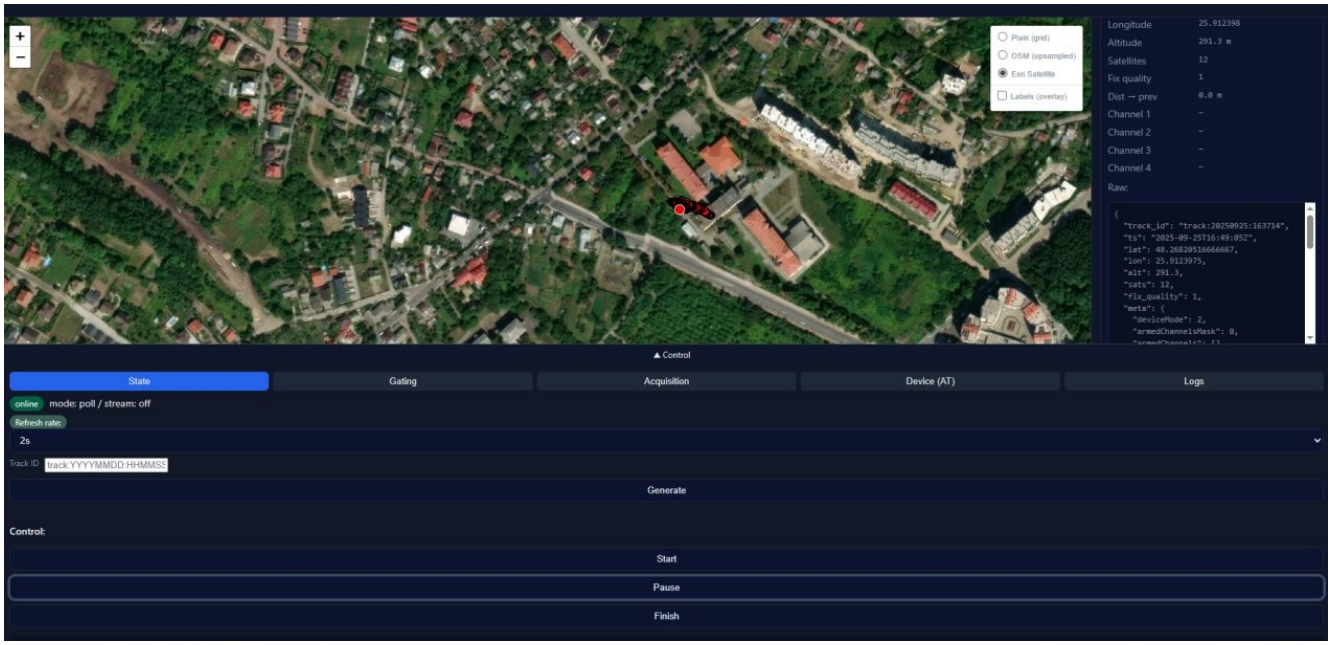
3.3.4. Апробація та перевірка функціональної стабільності системи

З метою підтвердження працездатності розробленої архітектури в умовах реальної експлуатації було проведено серію тестових прогонів (натурних випробувань) мобільного комплексу. Головним завданням цього етапу була не оцінка фізичних характеристик сенсора (що є предметом окремих досліджень), а верифікація стабільності програмно-апаратної платформи SoC FPGA під час виконання сценаріїв «політ — втрата зв’язку — відновлення зв’язку». Випробування проводилися з використанням описаного вище БПЛА на базі

рами TAROT 960. Платформа функціонувала у штатному режимі, виконуючи паралельний збір геолокаційних даних, опитування сенсора подій та запис треку в локальну базу даних.



(а) Режим картографування: візуалізація траєкторії та точок детекції об’єктів на карті місцевості (*Offline OSM Tiles*).



(б) Панель оператора: моніторинг телеметрії та пряме керування параметрами сенсора через AT-команди.

Рис. 3.12: Інтерфейс оператора автономного вузла. Система забезпечує повний цикл моніторингу та керування без залежності від хмарних сервісів.

Візуалізація даних реалізована через веб-додаток, розміщений безпосередньо на борту. З метою розвантаження обчислювальних ресурсів SoC, архітектура

інтерфейсу побудована за принципом *Client-Side Rendering*: усі операції з рендерингу картографічної підкладки, побудови векторних треків та інтерактивної фільтрації даних делегуються **рушію** браузера на комп'ютері оператора. Це дозволяє платформі підтримувати повноцінний інтерактивний сервісний стек, не витрачаючи обмежені ресурси HPS на генерацію графічного контенту.

На Рис. 3.12 продемонстровано роботу інтерфейсу оператора. Карта (3.12а) відображає траєкторію руху та теплову карту спрацювань сенсора. Панель діагностики (3.12б) дозволяє контролювати «сирі» параметри сенсора та відправляти AT-команди керування.

У ході випробувань підтверджено ефективність обраної архітектури:

1. **Функціональна стабільність (Stability):** Протягом серії тестових польотів не зафіксовано жодного випадку аварійної зупинки сервісу-оркестратора або зависання операційної системи. Це підтверджує, що застосування методики ізоляції ядер ефективно захищає критичні процеси керування від впливу системного навантаження.
2. **Верифікація механізму автономного накопичення даних *Store-and-Forward*:** Під час випробувань імітувався сценарій виходу БПЛА із зони покриття мережі. Аналіз локального журналу (SQLite) після завершення місії підтвердив цілісність записаного треку: всі події, згенеровані сенсором під час відсутності зв'язку, були успішно збережені та автоматично синхронізовані з інтерфейсом оператора після відновлення з'єднання.
3. **Інтеграційна спроможність та швидкодія:** Візуалізація треку в режимі реального часу продемонструвала плавність відображення даних без візуальних розривів (glitches) або блокування інтерфейсу («заморожування»). Це свідчить про те, що обчислювальної потужності HPS достатньо для одночасного парсингу потоку даних, підтримки транзакційної цілісності БД та обслуговування веб-клієнтів у реальному часі.

Отримані результати підтверджують, що розроблена SoC FPGA-платформа ефективно масштабується на задачі класу *Control Plane*. Свідоме використання ресурсів HPS для інтеграції низькошвидкісної периферії дозволяє оптимально розподілити обчислювальне навантаження, зберігаючи ресурси FPGA для задач

інтенсивної цифрової обробки сигналів. Поєднання автономності, подієвої архітектури та сервісно-орієнтованого підходу формує універсальну основу для побудови прикладних edge-систем різного призначення.

3.4. Порівняльний аналіз архітектурної ефективності та верифікація універсальності платформи

Результати експериментальної апробації, наведені у підрозділах 3.2 та 3.3, дозволяють розглядати розроблену SoC FPGA-платформу не тільки як сукупність окремих прикладних рішень, а й уніфіковану архітектурну основу, здатну адаптуватися до принципово різних профілів навантаження.

Метою порівняльного аналізу є довести, що стабільність роботи в цих діаметрально протилежних сценаріях забезпечується не надлишковою обчислювальною потужністю апаратної частини, а сукупністю архітектурних інваріантів, закладених на рівні операційної системи та моделі взаємодії між програмним і апаратним доменами, як це було обґрунтовано в Розділі 2.

3.4.1. Архітектурні інваріанти як основа універсальності

Першим ключовим інваріантом є методика просторової ізоляції обчислювальних ресурсів, що реалізується за допомогою механізмів *CPU affinity* та *IRQ shielding*.

- У підрозділі 3.2 ця методика застосована для ізоляції обробки USB/DMA-переривань та ядра спектрального аналізу, що дозволило забезпечити стабільний потік даних з частотою до **20 MSPS** без втрат семплів.
- У підрозділі 3.3 аналогічний підхід використано для захисту обробки апаратних переривань сенсорної підсистеми від впливу Python-сервісів та механізмів керування пам’яттю (*Garbage Collector*).

Таким чином, показано, що просторове рознесення навантажень є універсальним інструментом забезпечення детермінізму як для систем з жорсткими вимогами до рівномірності обробки потоків (*Jitter*), так і для подієвих систем, чутливих до часу реакції (*Latency*).

Другим архітектурним інваріантом є стратегія персистентного збереження даних. В обох реалізаціях використано єдиний механізм на базі SQLite у

режимі *Write-Ahead Logging* (WAL). Цей режим трансформує патерн запису на флеш-пам'ять у послідовний (*sequential writes*). Що суттєво знижує коефіцієнт підсилення запису (*Write Amplification*), подовжуючи ресурс eMMC-пам'яті платформи та забезпечує неблокуючий запис для високошвидкісних потоків (**SDR**) та гарантує атомарність транзакцій для критичних даних (**UAV**).

3.4.2. Адаптивність реалізації: дихотомія Data Plane та Control Plane

Порівняння двох реалізованих систем наочно демонструє доцільність гетерогенного підходу до реалізації прикладної логіки. Відображення такої архітектури, де на базі єдиного ядра та апаратної платформи розгорнуто два принципово різні профілі прикладного рівня, наведене на Рис. 3.13.

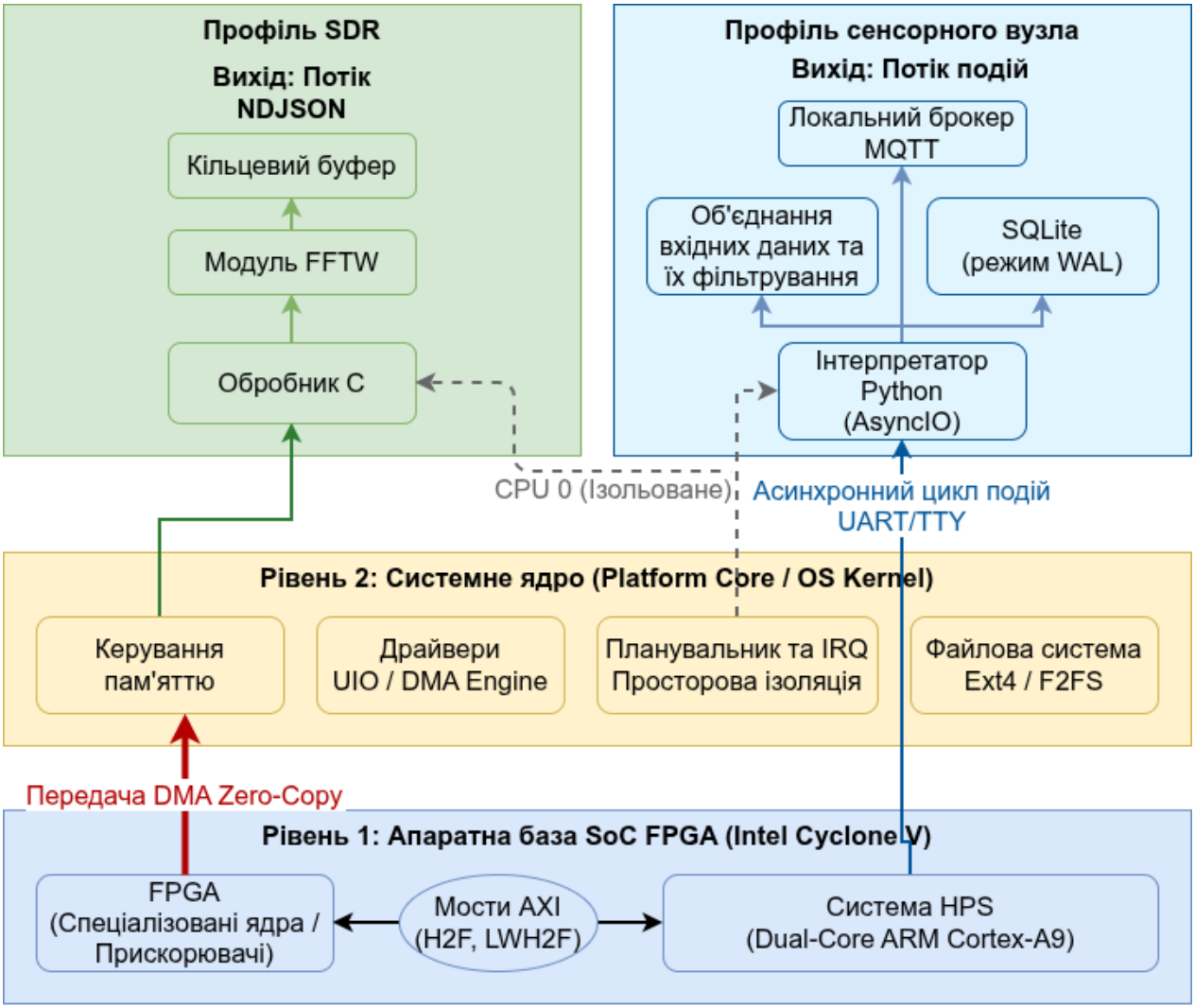


Рис. 3.13: Уніфікована архітектура програмно-апаратного стеку: реалізація двох різнорідних профілів навантаження (SDR та Sensor Node) на базі спільного системного ядра.

Для потокового режиму визначальними є пропускна здатність та мінімізація

накладних витрат, що обґрунтовує використання мови C та Zero-Copy доступу (ліва частина схеми). Натомість у подієвому режимі ключовими є гнучкість інтеграції, що робить доцільним використання Python та асинхронної моделі (права частина схеми). Фактично, вибір інструментарію реалізації (C або Python) диктується **місцем підсистеми у контурі керування**: C забезпечує прозорість для потоку даних (*Data Plane*), тоді як Python забезпечує гнучкість для логіки прийняття рішень (*Control Plane*).

Ця відмінність підтверджує здатність платформи адаптувати інструментарій до характеру навантаження без зміни базової архітектури. Узагальнене порівняння профілів навантаження наведено в табл. 3.1.

Таблиця 3.1

Порівняльний аналіз профілів описаних підсистем

Критерій	**SDR-сканер (Розділ 3.2)**	**Сенсорний вузол (Розділ 3.3)**
Тип навантаження	Потоковий (*High-Throughput*), *Data Plane Intensive*	Подієвий (*Event-Driven*), *Control Plane Intensive*
Характер та інтенсивність даних	I/Q-семпли, безперервний потік (**до 20 MSPS**)	Асинхронні фрейми телеметрії (**25 Гц**)
Вимоги до реального часу	Чутливість до затримки (*Jitter-sensitive*)	Чутливість до часу реакції (*Reaction-sensitive*)
Керування потоком (*Flow Control*)	**Implicit Backpressure** (блокуючий I/O, адаптивна децимація)	**Asynchronous Queues** (неблокуючий I/O, MQTT QoS)
Реалізація логіки	**C / Zero-Copy** (оптимізація доступу до пам’яті)	**Python / AsyncIO** (складна логіка агрегації)
Роль FPGA/HPS	FPGA/DMA як джерело потоку, HPS на межі продуктивності	HPS як основний контролер, FPGA як міст вводу-виводу
Стратегія зберігання	Пакетний запис (Batch Write)	Транзакційний запис подій

Така адаптивність проявляється і в стратегіях розвантаження обчислювального ядра. Якщо в системі SDR (п. 3.2) захист процесора досягається на рівні ядра ОС (*IRQ Shielding*), то в сенсорному вузлі (п. 3.3) застосовано прикладний патерн *Client-Side Rendering*, який делегує задачі візуалізації

зовнішньому клієнту. Це підтверджує, що архітектура платформи дозволяє застосовувати ешелонований захист ресурсів HPS на всіх рівнях моделі OSI.

3.4.3. Аналіз ефективності використання ресурсів та межі масштабування

Порівняння експериментальних режимів дозволяє виявити якісні межі застосовності обраної архітектури. У сценарії 3.2 обробка потоку 20 MSPS призводила до **насичення обчислювальних ресурсів процесора**, що вказує на те, що HPS (Cortex-A9) працює на межі своєї пропускної здатності при виконанні інтенсивних математичних операцій (FFT) у програмному режимі. Це обґрунтовує необхідність перенесення задач фільтрації та FFT у FPGA при подальшому збільшенні смуги пропускання.

Натомість у сценарії 3.3 система демонструвала **стабільно високу реактивність без ознак конкуренції за процесорний час**, навіть при активній роботі Python-інтерпретатора та бази даних. Це підтверджує, що для задач керування та логістики даних HPS-centric підхід є оптимальним, залишаючи суттєвий запас ресурсів для можливого розширення функціоналу (наприклад, для складнішої аналітики на борту).

Така асиметрія доводить, що платформа ефективно балансує навантаження:

1. Для задач *Data Plane* лімітуючим фактором є CPU, тому архітектура орієнтована на Zero-Copy та розвантаження через DMA.
2. Для задач *Control Plane* ресурсу CPU достатньо, тому пріоритетом стає швидкість розробки та гнучкість (Python/MQTT).

Для визначення меж запропонованої програмної архітектури (*HPS-Only*) було проведено аналітичне моделювання завантаженості обчислювальних ресурсів у залежності від вхідної частоти дискретизації (Рис. 3.14).

Модель базується на параметрах шини USB 2.0 (*High Speed*) та обчислювальній здатності ядер *Cortex-A9* (800 МГц). Як видно з графіка, лінійне зростання навантаження спостерігається до позначки 15 MSPS. У діапазоні 15–20 MSPS відбувається експоненційне зростання накладних витрат (*overhead*), спричинене явищем «*IRQ Storm*». Це явище зумовлене архітектурою шини USB 2.0 (*High Speed*), яка оперує мікрофреймами тривалістю 125 μs. На граничних швидкостях потоку процесорна підсистема змушена безперервно обробляти тисячі структур URB (*USB Request Blocks*), що призводить до колапсу

планувальника через накладні витрати USB-стеку ядра Linux. За таких умов частота генерації переривань перевищує швидкість їх обробки, роблячи подальше програмне масштабування неможливим.

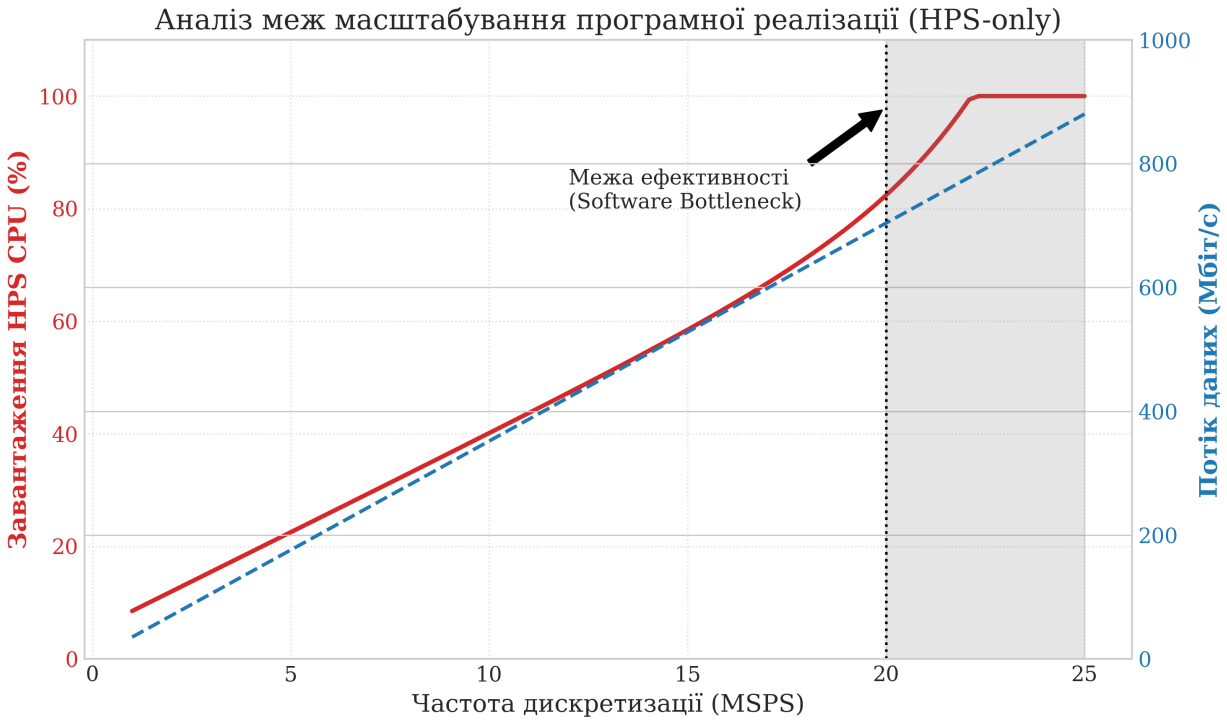


Рис. 3.14: Аналітична оцінка використання ресурсів та меж масштабування.

Точка насичення (*Saturation Point*) фіксується на рівні **20 MSPS**. У цій зоні (заштрихована область) CPU досягає 100% завантаження, що призводить до недетермінованих затримок і втрати пакетів (*packet drops*). Це аналітично підтверджує, що для задач зі смугою понад 20 МГц (що відповідає потоку даних > 640 Мбіт/с) програмна реалізація втрачає свою ефективність, і потребує зміни платформи (більше ресурсів) або впровадження обробки через FPGA/HLS.

3.4.4. Верифікація методики побудови ОС та концепції Edge-вузла

Важливим результатом порівняльного аналізу є підтвердження ефективності методики побудови операційної системи (”*Soft Specialization*”), запропонованої в Розділі 2. Той факт, що одна й та сама програмно-апаратна платформа з мінімальними модифікаціями користувацького простору забезпечує стабільну роботу як високошвидкісного сканера, так і складного сервісно-орієнтованого стеку, є прямим доказом коректності обраного підходу.

У сукупності наведені результати доводять, що розроблена програмно-апаратна платформа є інваріантною до типу прикладного

навантаження. Вона успішно масштабується від задач інтенсивної цифрової обробки сигналів до задач автономного керування без зміни базової архітектури ядра та системного оточення, що верифікує універсальність запропонованих у роботі архітектурних рішень.

3.5. Висновки до розділу 3

У третьому розділі здійснено практичну реалізацію та експериментальне дослідження гетерогенних **вузлів цифрової обробки сигналів** на базі платформ SoC FPGA. На основі закладених у попередніх розділах архітектурних інваріантів керуючого середовища реалізовано два принципово відмінні профілі навантаження: систему безперервного потокового спектрального моніторингу (SDR-сканер) та автономний сенсорний вузол із просторовою ізоляцією обчислювальних потоків.

Ключовим науковим здобутком розділу є розроблення та апробація апаратно-програмного методу просторової селекції джерел радіовипромінювання на базі рухомих платформ (БПЛА). Пропонований метод виявлення та пеленгації джерел радіовипромінювання з борту БПЛА на основі багатоканального антено-фідерного тракту з односпрямованими антенами різних частотних діапазонів, який передбачає оглядове кругове сканування з подальшим адаптивним локальним уточненням напрямку на джерело забезпечує скорочення часу пошуку T_{find} щонайменше у 2 рази відносно повного raster-сканування по (φ, θ) за однакових кроків дискретизації та часу інтеграції t_d.

Надійність функціонування вузлів підтвердила ефективність застосування сервісно-орієнтованої архітектури з чітким логічним розмежуванням площини обробки даних (*Data Plane*) та площини керування (*Control Plane*). Використання оптимізованого конвеєра обробки на базі прямого доступу до пам'яті (DMA *Zero-Copy*) та виконання математичних операцій (FFT) у контексті високопріоритетних переривань (*Run-to-Completion*) дозволило забезпечити стабільність потокової обробки в умовах м'якого реального часу без втрати сигнальних кадрів.

Водночас, під час проведення стрес-тестування та порівняльного аналізу архітектурної ефективності було виявлено фундаментальні обмеження виключно програмної реалізації радіотракту. Експериментально встановлено «точку насичення» процесорної підсистеми (HPS) на рівні потоку даних у 20 MSPS,

за якої виникає ефект «шторму переривань» (*IRQ Storm*), що призводить до критичної деградації пропускної здатності. Виникає гостра необхідність архітектурного переходу — перенесення обчислювально-критичних алгоритмів на рівень програмованої логіки (FPGA). Розв’язанню цієї задачі шляхом розробки механізмів апаратного розвантаження присвячено наступний розділ роботи.

РОЗДІЛ 4. СИНТЕЗ ТА СИСТЕМНА ІНТЕГРАЦІЯ ВИСОКОПРОДУКТИВНИХ АПАРАТНИХ ПРИСКОРЮВАЧІВ ДЛЯ ВУЗЛІВ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ

4.1. Подолання обмежень програмної обробки: архітектурний перехід

У попередньому розділі було експериментально досліджено можливості програмної обробки потокових даних на вбудованій Linux-платформі на базі SoC FPGA у складі реконфігурованих вузлів. Зокрема, у п. 3.5 зафіксовано досягнення так званої **точки насичення** (*Saturation Point*) обчислювального ядра HPS при обробці потоків із ефективною швидкістю близько 20 MSPS, що відповідає інформаційному потоку порядку 600–650 Мбіт/с. Досягнення цієї межі не розглядається як недолік програмної реалізації або неефективність оптимізацій користувацького простору. Навпаки, воно є індикатором фундаментальної архітектурної невідповідності між класом потокових цифрових сигналів та послідовною моделлю виконання, притаманною процесорам загального призначення [107]. Цей підрозділ здійснює архітектурне узагальнення виявлених обмежень та обґрунтовує неминучість переходу від програмно-визначеної обробки до гетерогенної архітектури з апаратним винесенням обчислень і не містить нових експериментальних даних, фокусуючись на обґрунтуванні архітектурних змін.

4.1.1. Аналіз обчислювальних обмежень процесорної підсистеми

Фізична межа обчислювальної масштабованості HPS. Експерименти з SDR-сканером, наведені в п. 3.2, показали, що зі зростанням смуги вхідного сигналу до граничних значень програмна реалізація досягає режиму, у якому подальше масштабування обробки стає неможливим, незважаючи на використання механізмів прямого доступу до пам’яті (DMA) та архітектури zero-copy. Цей результат є принципово важливим: вузьким місцем системи виступає не підсистема введення-виведення та не пропускна здатність внутрішніх шин (AXI), які здатні передавати потоки на рівні гігабайтів за секунду. Обмеження зумовлене послідовною моделлю виконання бітових та

байтових операцій у конвеєрі процесора загального призначення, який змушений обробляти кожен елемент потоку інструкція за інструкцією [108].

Узагальнення отриманих експериментальних результатів дозволяє зробити висновок, що програмна реалізація не масштабується до швидкостей, характерних для сучасних широкосмугових каналів зв'язку та телекомунікаційних систем. Візуально наслідки досягнення цієї архітектурної межі проілюстровано на Рис. 4.1.

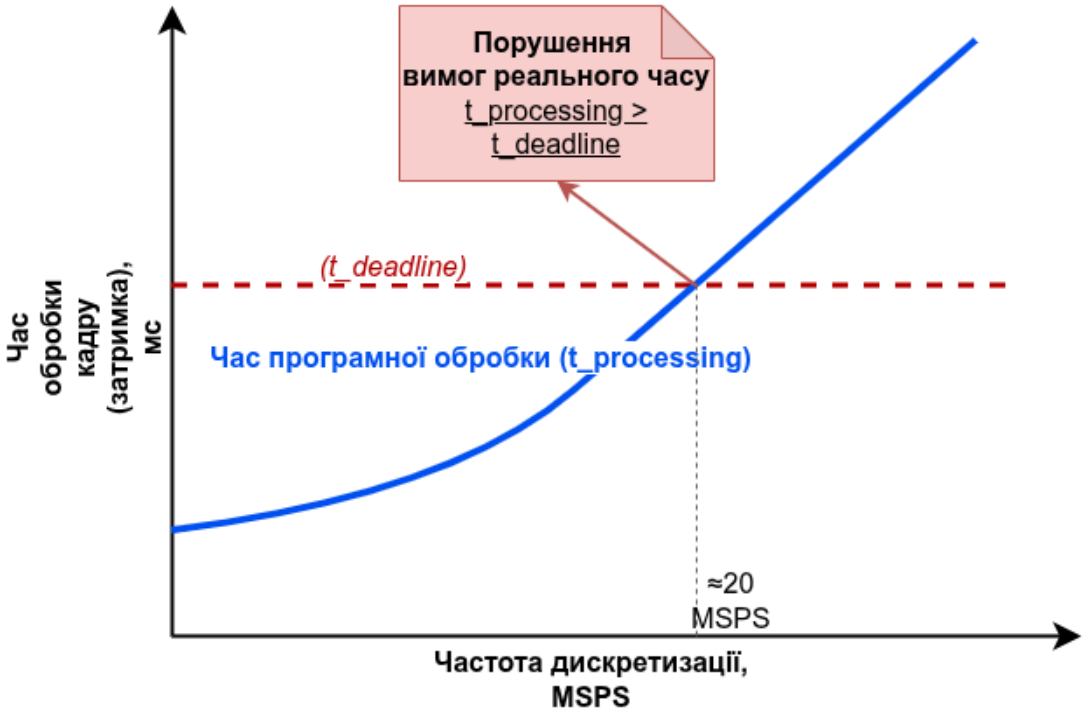


Рис. 4.1: Концептуальна модель динаміки зростання затримок програмної обробки ($t_{processing}$) відносно допустимого часового вікна ($t_{deadline}$).

Точка перетину ($\approx$ 20 MSPS) визначає **межі масштабованості** архітектури фон Неймана (HPS) для даного класу задач та обґрунтовує необхідність апаратного розвантаження (*Hardware Offloading*).

4.1.2. Проблема недетермінованості операційної системи.

Окрім обмежень обчислювальної масштабованості, програмна обробка високошвидкісних потоків у середовищі Linux стикається з проблемою часової недетермінованості. Навіть за умови просторової ізоляції ядер та фіксації процесів за окремими CPU, система залишається вразливою до перемикань контекстів, кеш-промахів та асинхронних подій ядра [107]. Для задач реального часу, зокрема потокового шифрування або цифрової обробки сигналів, ці фактори

призводять до виникнення джитеру та варіацій затримки, котрі неможливо повністю усунути засобами програмної оптимізації. На відміну від цього, апаратна логіка забезпечує фіксовану затримку обробки, незалежну від стану системи керування.

4.1.3. Концепція апаратної відповідальності

Для подолання зазначених обмежень у межах цієї роботи пропонується архітектурний патерн ***Hardware Accountability***. Його сутність полягає у свідомому перерозподілі ролей між програмним та апаратним доменами системи. У запропонованій моделі роль операційної системи трансформується з безпосереднього виконавця (*Worker*) часово-критичних обчислень та переходить до ролі оркестратора (*Orchestrator*), використовуючи механізми описані у розділі 2 та роботі ‘ [16], беручи відповідальність за конфігурацію, керування та моніторинг.

Архітектурна міграція потоків даних може бути формалізована так:

- **Програмно-визначена модель (Розділ 3):**
  $Input \rightarrow RAM \rightarrow CPU\ (Processing) \rightarrow RAM \rightarrow Output$;
- **Гетерогенна модель з апаратною відповідальністю (Розділ 4):**
  $Input \rightarrow FPGA\ (Processing) \leftrightarrow CPU\ (Control)$.

У такій архітектурі площина обробки даних (*Data Plane*) переноситься на рівень програмованої логіки, що дозволяє досягти швидкості середовища передачі (*wire speed*) без участі CPU [109].

На відміну від задач керування та телеметрії (розглянутих у п. 3.3), де *HPS-centric* підхід є енергетично ефективним [24], для задач генерації ключових потоків критичним параметром стає **затримка доступу** до результату, а не лише лінійна пропускна здатність.

Класична схема з використанням DMA ($Input \rightarrow FPGA \rightarrow RAM$) вносить накладні витрати на ініціалізацію дескрипторів та обробку переривань, що є надлишковим для передачі коротких векторів (64 біти гами за такт). Тому в рамках запропонованої концепції «апаратної відповідальності» для первинної інтеграції криптографічного ядра обрано архітектуру прямого регістрового відображення (*MMIO*). Цей підхід усуває накладні витрати на ініціалізацію дескрипторів DMA, що робить його потенційно оптимальним для задач керування (*Control Plane*) та швидкої зміни ключів (*Key Agility*) завдяки детермінованій реакції ОС. Водночас,

здатність такого програмно-керованого інтерфейсу задовольнити жорсткі вимоги щодо пропускної здатності для самого високошвидкісного потоку даних (*Data Plane*) є неочевидною і потребує окремого експериментального дослідження, результати якого визначать необхідність подальшої еволюції архітектури.

Це забезпечує детермінований час доступу до результату генерації безпосередньо з простору користувача ОС, минаючи проміжні буфери в оперативній пам'яті. Апаратна частина гарантує валідність даних у вихідних регістрах у будь-який момент звернення (*read-on-demand*), що перетворює FPGA на синхронний співпроцесор із нульовою програмною чергою, що повністю відповідає концепції ***Hardware Accountability***.

Концептуальну різницю між цими підходами проілюстровано на Рис. 4.2.

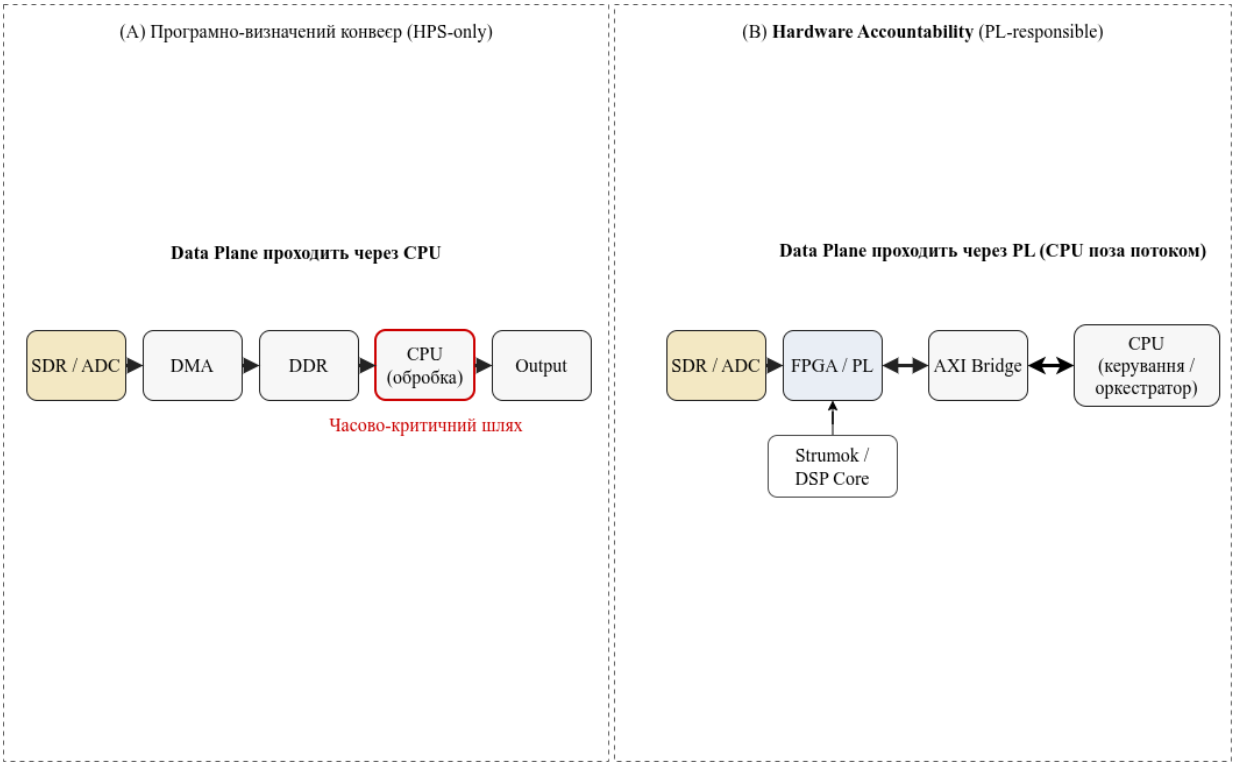


Рис. 4.2: Концептуальна схема архітектурної міграції

Перехід від послідовної обробки в пам'яті процесора (**A**) до потокової апаратної обробки. (**B**) Регістро-орієнтована архітектура (*MMIO*): Обчислювальне ядро (*IP Core*) інтегровано безпосередньо в адресний простір процесора через прозорий міст (*Bridge*). Це усуває накладні витрати на ініціалізацію DMA для задач керування та генерації ключів, забезпечуючи детермінований час доступу. Пунктирна лінія на схемі (**B**) ілюструє роль HPS як оркестратора, винесеного за межі критичного шляху передачі даних.

4.1.4. Обґрунтування вибору об’єкта реалізації

Для експериментальної верифікації запропонованого підходу апаратної акселерації обрано алгоритм симетричного потокового перетворення ДСТУ 8845:2019 ”Струмок- [110]. Вибір цього алгоритму зумовлений його специфікою та репрезентативністю для класу потокових DSP-задач [111].. На відміну від блокових шифрів, ”Струмок”оперує безперервним потоком даних, що структурно відповідає задачам цифрової обробки сигналів у телекомунікаційних системах та SDR-платформах. Математичний апарат алгоритму базується на операціях у скінченних полях $GF(2^8)$ та $GF(2^{64})$, зокрема на множеннях та лінійних перетвореннях типу *MixColumn* і роботі скінченного автомата [110]. Операції множення у цих полях на процесорах загального призначення реалізуються через ітеративні цикли зсуву та логічного додавання (*XOR*), що вимагає десятків машинних тактів на обробку одного байта. Натомість у архітектурі FPGA ці операції розгортаються у паралельні LUT-таблиці (*Look-Up Tables*) та масиви *XOR*-вентилів, що дозволяє виконувати складні нелінійні перетворення за один тактовий цикл. Крім того, вимоги до захищених каналів управління безпілотними системами та телекомунікаційними лініями зв’язку передбачають пропускну здатність на рівні 1–10 Гбіт/с, що принципово недосяжно для програмної реалізації на платформі Cortex-A9. Необхідно зазначити, що криптографічне ядро розглядається у даній роботі виключно як еталонний генератор надвисокошвидкісного бітового потоку для тестування пропускної здатності платформи (>80 Гбіт/с), а не як об’єкт дослідження криптографічної стійкості. Структурну складність алгоритму та обчислювально-критичні блоки наведено на Рис. 4.3.

4.1.5. Формулювання задачі розділу

З урахуванням наведеного, у цьому розділі ставляться такі задачі:

1. синтезувати високопродуктивне IP-ядро алгоритму ”Струмок”на рівні регістрових передач (RTL), орієнтоване на досягнення максимальної пропускної здатності;
2. розробити та реалізувати механізм **прямого регістрового доступу** (*Direct Register Access*) що забезпечує відображення простору станів криптографічного ядра у віртуальну пам’ять процесора для мінімізації накладних витрат ОС;;

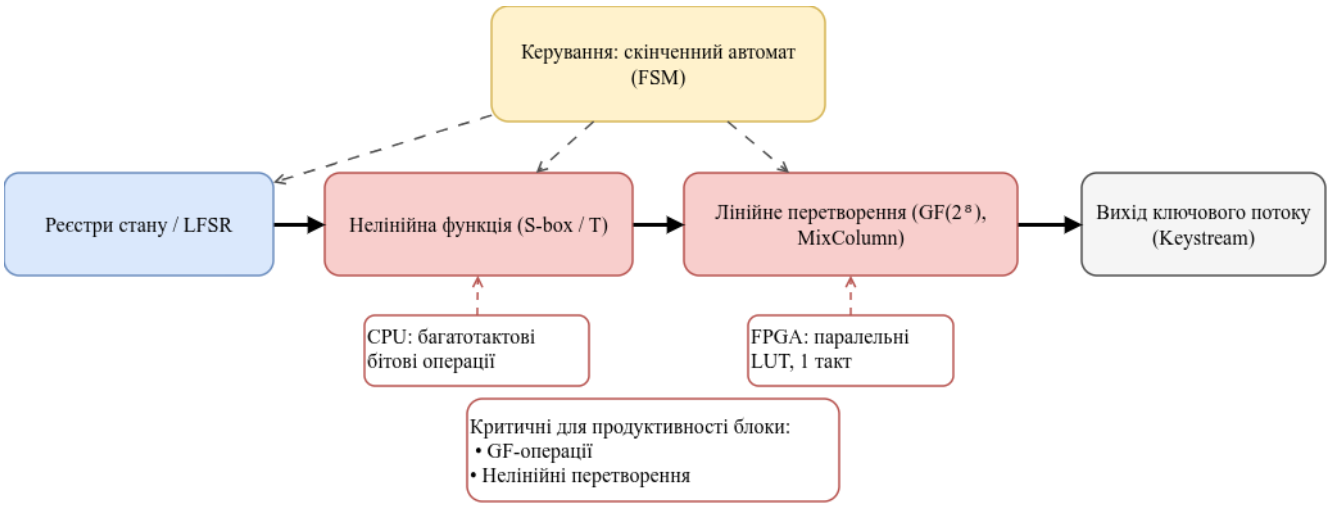


Рис. 4.3: Спрощена структурна схема алгоритму ДСТУ 8845:2019 ”Струмок”. Червоним кольором виділено блоки нелінійних та лінійних перетворень (*S-box*, *MixColumn*), котрі стають обчислювальним вузьким місцем (*bottleneck*) для послідовної архітектури CPU через необхідність емуляції бітових операцій, але природно розпаралелюються на апаратному рівні FPGA.

3. експериментально підтвердити ефективність запропонованої архітектури шляхом порівняльного аналізу швидкодії та ресурсоємності.

4.2. Синтез спеціалізованого обчислювального ядра алгоритму ”Струмок”на рівні регістрових передач (RTL)

У цьому підрозділі наведено інженерне обґрунтування та опис RTL-реалізації алгоритму ДСТУ 8845:2019 ”Струмок- [110] як спеціалізованого обчислювального ядра для програмованої логіки SoC FPGA.

Основну увагу в підрозділі зосереджено не на повторенні специфікації стандарту, а на поясненні того, *які мікроархітектурні рішення забезпечують* детерміновану затримку, високу пропускну здатність та контроль критичного шляху.

На відміну від програмної реалізації, де часові характеристики визначаються мікроархітектурою CPU та поведінкою ОС, RTL-ядро фіксує структуру *data path* та задає послідовність фаз виконання через апаратний *control path* (FSM), що узгоджується з концепцією *Hardware Accountability* з п. 4.1.

4.2.1. Обґрунтування гібридного маршруту проектування

У межах загальної методології синтезу вузлів цифрової обробки сигналів, заявленої у роботі, застосовано гібридний маршрут проектування. Його сутність полягає у системному розподілі рівнів абстракції: високорівневе проектування (із застосуванням засобів *HLS*) використовується для синтезу керуючої інфраструктури, адаптерних шарів (*AXI*), налаштування *DMA* та системної інтеграції, тоді як обчислювально-критичне ядро потокового перетворення ”Струмок”оптимізовано на рівні регістрових передач (*RTL*) мовою *Verilog HDL*. Концепцію такого розподілу ілюструє Рис. 4.4.

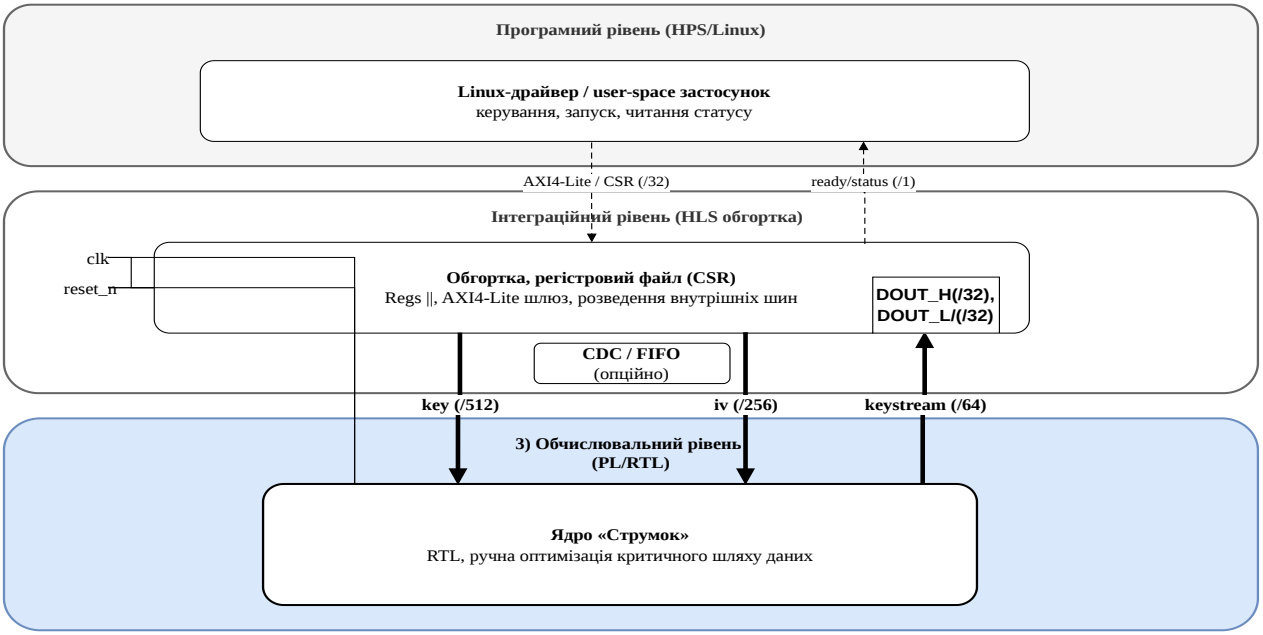


Рис. 4.4: Розподіл ролей методики проектування: високорівневі засоби проєктування застосовуються для інтеграційного (адаптерного) шару та системної побудови інтерконекту, тоді як критичне алгоритмічне ядро реалізовано на RTL для мінімізації затримки та забезпечення детермінованості.

Цей архітектурний вибір зумовлений специфікою криптографічних алгоритмів. На відміну від класичних задач ЦОС (наприклад, фільтрації чи БПФ), де HLS-компілятори ефективно розгортають математичні моделі, у потоковому шифруванні домінують побітові та байтові операції, нелінійні підстановки та широкі XOR-мережі. Для таких примітивів *RTL* забезпечує прямий апаратний контроль над топологією логіки та гарантує мінімізацію глибини комбінаторного ланцюга.

Хоча сучасні HLS-засоби дозволяють синтезувати подібні структури,

автоматична генерація графів потоку даних для масивів Галуа та регістрів зсуву з лінійним зворотним зв'язком (LFSR) часто призводить до появи надлишкової керуючої логіки. Використання прямого RTL-опису для обчислювального ядра дозволяє усунути ці накладні витрати та досягти детермінованої затримки в межах одного тактового інтервалу [112–114]. Таким чином, практична мотивація запропонованого гібридного маршруту полягає у поєднанні переваг двох світів: мікроархітектурної ефективності RTL для тракту даних та гнучкості високорівневого проектування для побудови інтерфейсної (системної) частини вузла у середовищі SoC FPGA [16].

4.2.2. Зовнішній інтерфейс ядра та класифікація сигналів

Зовнішній інтерфейс ядра (див. Табл. 4.1) спроєктовано як компактний *control-plane* протокол поверх широких *config* шин та вузького виходу даних.

Таблиця 4.1

Сигнали інтерфейсу RTL-ядра "Струмок"та їх роль у площинах керування/даних

Сигнал	I/O	Розрядність	Клас	Призначення
clk	in	1	clock	Тактовий сигнал синхронізації
reset_n	in	1	reset	Скидання (active-low)
init	in	1	control	Запуск ініціалізації стану (фаза INIT)
next	in	1	control	Запит генерації наступного блоку гами
key	in	512	config	Ключовий матеріал (паралельне завантаження)
iv	in	256	config	Вектор ініціалізації
key_length	in	1	config	Вибір режиму довжини ключа (наприклад, 256/512)
keystream	out	64	data	Вихід гами (64 біти/такт)
ready	out	1	status	Стан готовності/завершення фази

Ядро є синхронним модулем з тактуванням `clk` та активним-низьким скиданням `reset_n`; керування фазами здійснюється сигналами `init` та `next`.

Ключ та вектор ініціалізації подаються через паралельні шини `key[511:0]` та `iv[255:0]` з вибором режиму довжини ключа `key_length`.

На виході формується 64-бітний фрагмент гами `keystream[63:0]` за такт, а `ready` реалізує апаратний прапорець готовності та слугує основою для

рукостискання з зовнішнім середовищем (HPS/AXI-обгорткою).

```
1
2  module strumok_core(
3  input wire    clk,
4  input wire    reset_n,
5  input wire    init,
6  input wire    next,
7  input wire [511:0] key,
8  input wire [255:0] iv,
9  input wire    key_length,
10 output wire [63:0] keystream,
11 output wire    ready
12 );
```

Лістинг 4.1: Мінімальний фрагмент RTL-інтерфейсу ядра (top-level ports)

У лістингу 4.1 наведено лише компактний заголовок модуля як приклад RTL-реалізації та розрядностей інтерфейсу.

4.2.3. Внутрішня організація: розділення Data Path та Control Path

Мікроархітектура ядра побудована за принципом **розділення**:

- **Data Path:** регістровий стан, паралельні байтові перетворення, XOR/NOT/перестановки та формування `keystream`;
- **Control Path:** FSM та лічильники фаз, які апаратно визначають порядок оновлень стану, момент готовності та видачі результату.

Ключовою відмінністю від CPU-підходу є те, що у *data path* обчислення не ”виконуються”інструкціями, а **матеріалізуються** у вигляді просторово розгорнутої комбінаційної логіки, що дозволяє повною мірою задіяти паралелізм на рівні байтів/бітів.

Зокрема, нелінійний шар синтезовано як 8 паралельних байтових перетворювачів, що дозволяє обробляти 64 біти за такт. Це усуває серіалізацію, притаманну CPU-проходженню байтів через інструкційний потік [115] (структурну схему наведено на Рис. 4.5). Цей підхід відповідає базовим схемотехнічним ідеям апаратної реалізації національних криптографічних стандартів у вбудованих FPGA-платформах [20].

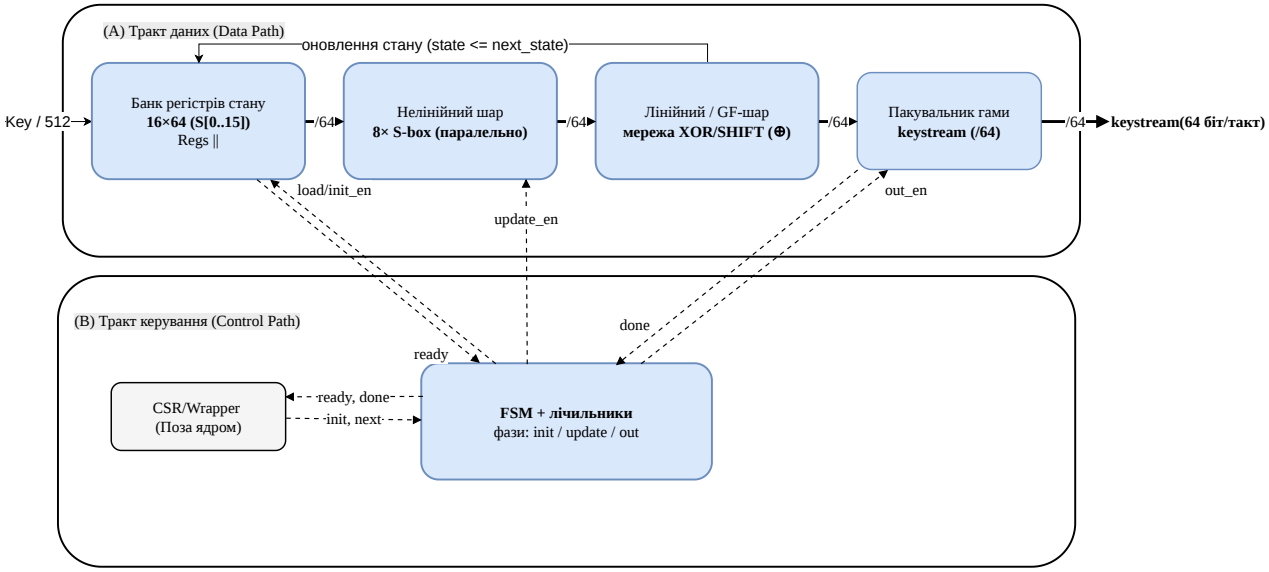


Рис. 4.5: Внутрішня архітектура ядра: розділення *Data Path* (потокові перетворення стану та генерація гами) і *Control Path* (FSM/лічильники, що визначають послідовність фаз і момент видачі результату).

Таке розділення узгоджується з концепцією *Hardware Accountability*: часово-критичні перетворення виконуються всередині *data path* без участі CPU, а зовнішній світ (HPS/Linux) взаємодіє з ядром через *control-plane* сигнали [115].

4.2.4. Фази роботи ядра та керуючий автомат (FSM)

З інженерної точки зору процес роботи ядра доцільно описувати як детермінований протокол фаз:

1. **INIT:** прийом/фіксація параметрів та формування стартового стану;
2. **WARM-UP/UPDATE:** фіксована серія внутрішніх оновлень, що гарантує досягнення коректного режиму генерації;
3. **GEN:** видача `keystream` у відповідь на запит `next` після встановлення `ready`.

```
localparam CTRL_IDLE        = 3'h0;
localparam CTRL_INIT        = 3'h1;
localparam CTRL_WARMUP      = 3'h2;
localparam CTRL_READY       = 3'h3;
localparam CTRL_OUTPUT      = 3'h4;
localparam CTRL_DONE        = 3'h5;
```

Лістинг 4.2: Кодування станів керуючого автомата (FSM encoding)

У реалізації ці фази відображено у станах керуючого автомата: INIT відповідає входу в `CTRL_INIT`, фаза розігріву (WARM-UP) реалізується циклічною

послідовністю станів `CTRL_FSM_UPDATE` та `CTRL_LFSR_UPDATE`, а фаза генерації (GEN) — формуванням виходу у `CTRL_OUTPUT` за запитом `next` (див. Лістинг 4.2). Жорстка логіка FSM виконує не лише роль керування, а й роль **апаратної гарантії коректності протоколу**: вона не допускає видачу гами до завершення ініціалізації та встановлення коректного стану [114, 115].

Інтерфейс керування реалізовано як мінімальний протокол рукостискання `init/next–ready`, котрий визначає момент готовності ядра до прийому команд та момент коректної видачі результату. Таким чином, синхронізація взаємодії з системним доменом (HPS/Linux) виконується без втручання в тракт даних і не впливає на детермінованість *data path*, що відображено на Рис. 4.6.

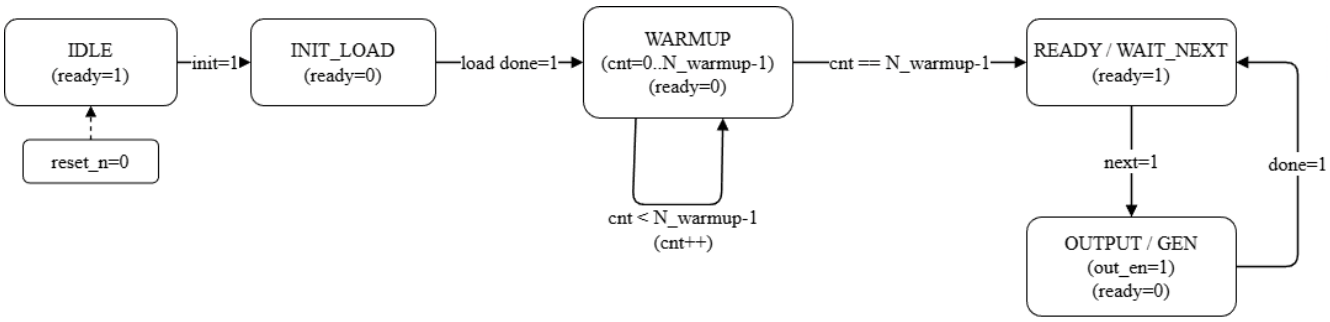


Рис. 4.6: Діаграма станів (FSM) RTL-ядра ”Струмок”.

Режими оновлення стану (INIT/SBOX/MAIN/FINAL) реалізовано як внутрішні сигнали керування *data path*, котрі активуються залежно від поточного стану FSM та лічильників фази.

XOR/NOT як комбінаційні примітиви: контроль логічної глибини

Ключова причина апаратної ефективності полягає у тому, що операції інверсії та XOR не є послідовними інструкціями, а реалізуються як **комбінаційна логіка** (LUT/XOR-мережі) [116, 117].

Отже, часові витрати визначаються не ”кількістю операцій а **глибиною комбінаційного ланцюга** та допустимою тактовою частотою.

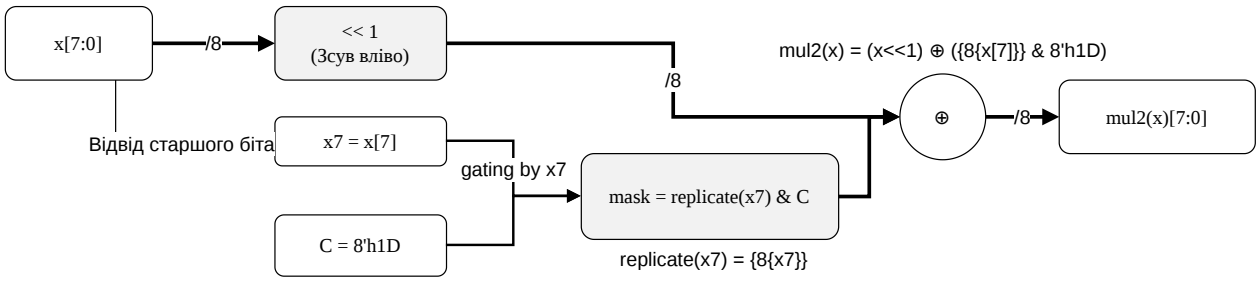


Рис. 4.7: Принципова схема XOR-мережі як апаратного примітиву: затримка визначається глибиною логічного дерева, не фокусуючись на кількості інструкцій.

Це принципово відрізняє FPGA-підхід від CPU-реалізації, де ті самі перетворення серіалізуються інструкційним потоком та підпадають під вплив кешування й планування ОС, що зображено на Рис. 4.7.

4.2.5. Перетворення арифметики $GF(2^8)$ у комбінаторну логіку

Найбільш показовим місцем, де математичний апарат алгоритму безпосередньо відображається у схемотехніку, є множення у полі $GF(2^8)$, що входить до складу байтових лінійних перетворень (MixColumn-подібні операції). На CPU множення на константу (зокрема $0x02$) розкладається на послідовність інструкцій із умовною редукцією полінома, що робить затримку залежною від розгалужень логіки та мікроархітектурного стану конвеєра.

У розробленому RTL-ядрі множення на примітивний елемент α (0x02) реалізовано як *hardwired* комбінаційний фрагмент зсувів і XOR-вентилів, тобто без введення окремих тактових фаз:

$$\text{mul2}(x) = (x \ll 1) \ \oplus \ \big((x_7) \cdot \text{0x1D}\big), \tag{4.1}$$

де x_7 — старший біт байта x, а доданок 0x1D активується апаратним маскуванням (gating) залежно від x_7.

Таким чином, арифметика $GF(2^8)$ входить до складу комбінаційної логіки між регістрами *data path*, а не виконується у вигляді послідовних інструкцій [116,118]. З практичної точки зору це означає, що виконання $\text{mul2}(\cdot)$ відбувається в межах одного тактового інтервалу за умови виконання таймінг-обмеження ($t_{pd} < T_{clk}$), що узгоджується з результатами таймінг-аналізу ядра (табл. 4.2). У подальшій системній інтеграції (розд. 4.3) робоча частота фіксується на рівні $F_{sys} = 100$ МГц з міркувань спрощення інтеграційного тракту та відтворюваності експериментів.

4.2.6. Оцінка ресурсів та швидкодії

Для верифікації апаратної ефективності розробленого RTL-ядра виконано синтез та таймінг-аналіз для цільової ПЛІС Intel Cyclone V SoC (5CSEBA6).

Результати (див. табл. 4.2) отримано у середовищі Intel Quartus Prime у режимі *Slow 1100mV 85C Model*.

Таблиця 4.2

Характеристики синтезованого ядра

Показник	**Значення**	**Коментар**
Logic Utilization (ALM)	2,386 ($\approx 6\%$)	Основна витрата зосереджена у нелінійних байтових перетвореннях (S-box-подібні LUT-модулі).
Total Registers	2,839	Регістризація тракту даних використовується для стабілізації таймінгу та підтримки pipeline-організації.
DSP Blocks	**0**	Апаратні помножувачі не використовуються; операції $GF(2^8)$ реалізовано через розподілену XOR/SHIFT-логіку.
Block Memory (M10K)	**0**	Таблиці підстановок реалізовано у розподіленій логіці для мінімізації затримки.
Fmax (Core Clock)	150 MHz	Досягнута максимальна частота після синтезу/таймінг-аналізу для заданого PVT-профілю.
Core throughput bound	≈ 9.6 **Gbit/s**	Розрахунок: $V_{core} = 150$ МГц $\times$ 64 біт. Пропускна здатність *ядра* при F_{max}.
Throughput (system)	≈ 6.4 **Gbit/s**	Для інтеграції тактова частота обмежується $F_{sys} = 100$ МГц (100 МГц $\times$ 64 біт).
Latency (Initial)	33 цикла	Заповнення конвеєра та ініціалізація ключа.

Аналіз отриманих результатів:

1. **Відсутність DSP/BRAM як індикатор характеру обчислень.**

   Результати підтверджують, що ядро реалізує переважно логічні

перетворення (XOR/SHIFT/LUT), що відповідає природі операцій у $GF(2^8)$ та архітектурній мотивації винесення обчислень у PL [115, 118].

2. **Перевага за пропускною здатністю відносно HPS-only режиму.**
   Отримана потокова швидкість **9.6 Гбіт/с** перевищує межу програмної реалізації на CPU, зафіксовану в п. 3.5 (порядку 600–650 Мбіт/с), та наближається до класу 10-гігабітних інтерфейсів, що є критично важливим для сучасних телекомунікаційних задач.
3. **Запас ресурсів для системної інфраструктури.**
   Використання близько $\approx$ 6% ALM залишає простір для інтеграційної інфраструктури (DMA, адаптери потоків, регістрові файли керування), яка розглядається у наступному підрозділі.

4.2.7. Функціональна верифікація та відповідність еталонним векторам

Для підтвердження коректності апаратної реалізації алгоритму виконано функціональну верифікацію RTL-моделі на рівні поведінкового моделювання (*Behavioral Simulation*).

Верифікація здійснювалася за методикою побітового порівняння (*Bit-exact matching*) для складного сценарію ініціалізації (Приклад №3 з Додатку Д.1.1 [110]), котрий передбачає ненульовий вектор ініціалізації.

Параметри тестового впливу:

- **Ключ (K):** 2^{255} (старший біт — 1, решта — 0);
- **Вектор ініціалізації (IV):** $IV_3 = 4,\ IV_2 = 3,\ IV_1 = 2,\ IV_0 = 1$ (у 64-бітному представленні).

Отримані значення перших восьми слів гами ($Z_0 \dots Z_7$):

```
Z0:fe44a2508b5a2acd; Z1:af355b4ed21d2742;
Z2:dcd7fdd6a57a9e71; Z3:5d267bd2739fb5eb;
Z4:b22eee96b2832072; Z5:c7de6a4cdaa9a847;
Z6:72d5da93812680f2; Z7:4a0acb7e93da2ce0.
```

повністю співпадають з еталонними значеннями стандарту [110]. Це підтверджує коректність роботи скінченного автомата (FSM) у фазах ініціалізації (”розігріву”) та генерації, а також правильність синтезу таблиць нелінійних підстановок. Часова діаграма верифікації наведена на Рис. 4.8.

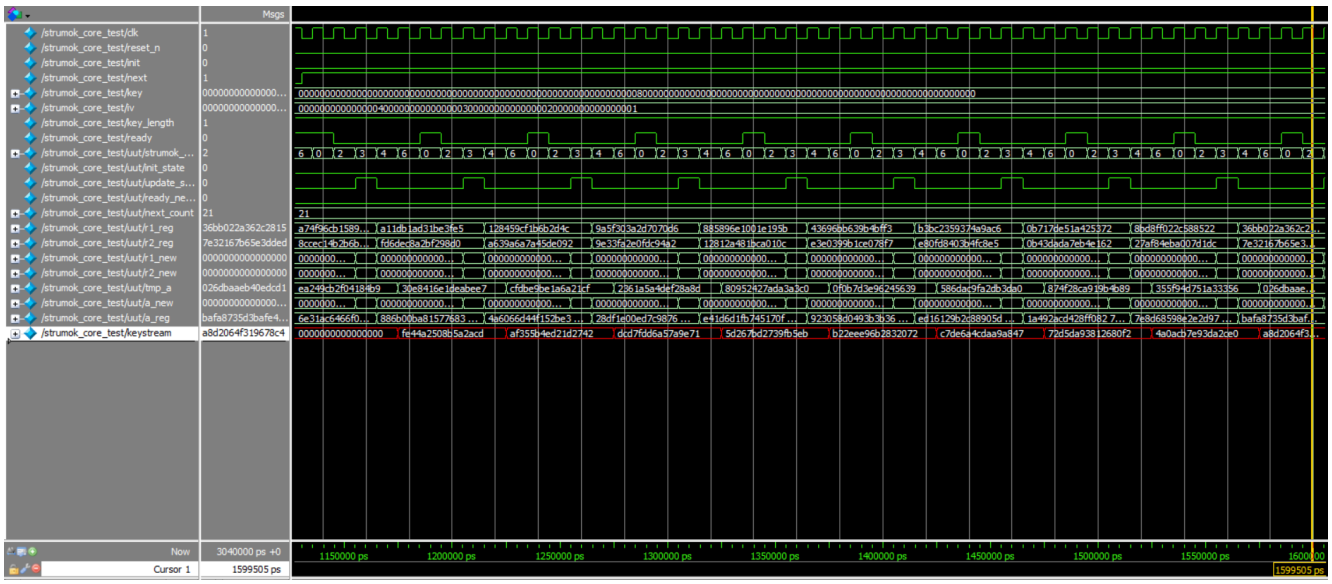


Рис. 4.8: Часова діаграма функціональної верифікації RTL-ядра для тестового вектора №3 (ДСТУ 8845). Маркерами виділено момент переходу сигналу `ready` в активний стан та генерацію валідної послідовності гами $Z_0 \ldots Z_7$, що ідентична еталону.

У підрозділі сформовано та обґрунтовано мікроархітектуру RTL-ядра ”Струмок”як спеціалізованого прискорювача для часово-критичної потокової обробки в PL-домені SoC FPGA. Показано, що продуктивність досягається не ”оптимізацією коду а архітектурним перетворенням задачі: критичні операції зведено до комбінаційних XOR/SHIFT-мереж та регістрових оновлень, а послідовність фаз виконання зафіксовано апаратним FSM.

Арифметика $GF(2^8)$ формалізована у вигляді *hardwired* логіки (gating + XOR), що усуває умовні переходи та забезпечує передбачувану затримку у межах тактового інтервалу за умови виконання таймінгових обмежень (*Timing Closure*). Результати синтезу та верифікації підтверджують працездатність ядра та його придатність до режиму *hardware offloading* з пропускною здатністю класу multi-Gbit/s [112, 115, 116] (Табл. 4.2).

4.3. Системна інтеграція та експериментальне дослідження підсистеми керування (Control Plane)

Розроблене у суброзділі 4.2 RTL-ядро потокового шифру «Струмок» інтегрується у системну архітектуру SoC FPGA як автономний апаратний прискорювач. Якщо у попередньому підрозділі фокус було зосереджено на *мікроархітектурі* та максимізації пропускної здатності (*Data Plane*), то метою

цього етапу є реалізація та дослідження підсистеми керування [85, 119] (*Control Plane*).

На рівні інтеграції реалізується модель *Hardware Offloading*: часово-критична генерація гами повністю делегується програмованій логіці (PL), тоді як процесорна система (HPS) виконує виключно функції диспетчеризації та координації — конфігурацію, запуск та моніторинг статусу.

4.3.1. Архітектура адаптерного шару (Wrapper)

Для підключення RTL-ядра до системної шини розроблено спеціалізований адаптерний модуль (*Wrapper*), який виконує роль моста між системним інтерфейсом та внутрішніми сигналами ядра [120]. Архітектурно адаптер вирішує три задачі:

1. Трансляція протоколу: Перетворення транзакцій стандартної шини **AXI4-Lite** у сигнали керування ядром (`init`, `next`, `key_load`) відповідно до специфікації AMBA AXI [121].
2. **Перетин тактових доменів (CDC):** Системна шина керування працює на консервативній частоті $F_{sys} = 100$ МГц, тоді як обчислювальне ядро може тактуватися на вищій частоті ($F_{core} \geqslant 150$ МГц).
3. **Регістрове відображення (MMIO):** Надання програмному драйверу доступу до портів введення-виведення через фіксовану карту пам'яті.

```
module axi_strumok_wrapper # (
    parameter C_S_AXI_DATA_WIDTH = 32,
    parameter C_S_AXI_ADDR_WIDTH = 6
)(  // AXI4-Lite Slave Interface
    input wire  s_axi_aclk,
    input wire  s_axi_aresetn,
    input wire [C_S_AXI_ADDR_WIDTH-1 : 0] s_axi_awaddr,
    input wire [C_S_AXI_DATA_WIDTH-1 : 0] s_axi_wdata,
    input wire  s_axi_wvalid,
    output wire s_axi_wready,
    // Core Logic Interface
    output wire        core_enable,
    output wire [511:0] core_key,
    output wire [255:0] core_iv,
    input  wire [63:0]  core_keystream );
```

Лістинг 4.3: Інтерфейс AXI4-Lite адаптера (Verilog HDL)

Для забезпечення сумісності зі стандартною шиною інтерконекту розроблено модуль `axi_strumok_wrapper`, інтерфейс якого наведено у лістингу 4.3.

Важливою особливістю розробленого адаптера є використання архітектури **Shadow Register File**. Запис частин ключа через шину AXI відбувається у проміжні буфери, а оновлення активного ключа ядра здійснюється атомарно за один такт при отриманні команди ‘*START*‘. Це апаратно унеможливлює атаку типу ’*Race Condition*’, коли шифрування могло б початися з частково оновленим ключем

4.3.2. Програмна модель взаємодії

З точки зору операційної системи Linux, інтегрований прискорювач виглядає як ділянка фізичної пам’яті (*Memory Region*). Взаємодія реалізується через механізм *ioremap*, що дозволяє драйверу ядра звертатися до регістрів прискорювача як до звичайних змінних [122], оминаючи кеш процесора карту регістрів з таб. 4.3.

Таблиця 4.3

Карта регістрів (Memory Map) апаратного прискорювача «Струмок»

Зсув	Регістр	R/W	Призначення та опис бітів
0x00	CTRL	W	**Регістр керування** Bit 0: START — запуск генерації Bit 1: RESET — скидання FSM
0x04	STATUS	R	**Регістр стану** Bit 0: READY — ядро готове Bit 1: DONE — генерація завершена
0x10	KEY_L	W	Молодші 32 біти ключа ($K[31:0]$). *Запис дозволено лише у стані IDLE.*
0x14	KEY_H	W	Старші частини ключа (проміжні регістри пропущено для стислості).
0x20	IV_L	W	Молодші 32 біти вектора ініціалізації ($IV[31:0]$).
0x30	DOUT_L	R	Молодші 32 біти гами ($Z[31:0]$). *Читання валідне, коли STATUS[1]=1.*
0x34	DOUT_H	R	Старші 32 біти гами ($Z[63:32]$).

Така організація забезпечує повну прозорість стану прискорювача для програмного забезпечення.

4.3.3. Експериментальна оцінка маневреності ключів (Key Agility)

Ключовою характеристикою підсистеми керування є здатність системи до швидкої зміни криптографічного контексту — так звана *key agility* [123,

124]. На відміну від класичних метрик пропускної здатності (Мбіт/с), дана характеристика відображає, з якою частотою система може виконувати повний цикл: завантаження ключа, ініціалізацію ядра та готовність до генерації.

На рис. 4.9 наведено експериментальну оцінку швидкості зміни криптографічного контексту при керуванні ядром через інтерфейс AXI4-Lite.

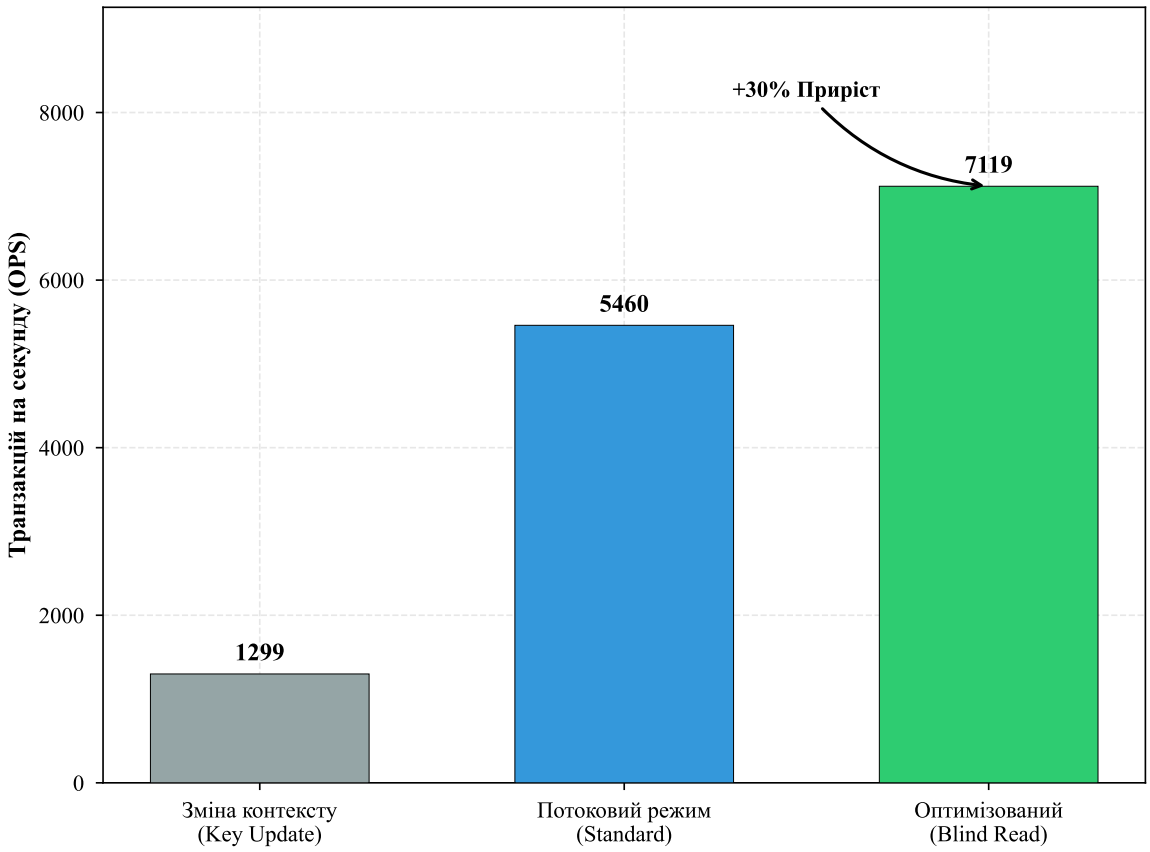


Рис. 4.9: Експериментальна оцінка швидкості зміни криптографічного контексту (Key Agility). Система забезпечує темп понад 7000 оновлень ключа на секунду через інтерфейс AXI4-Lite.

Експериментально підтверджена спроможність системи забезпечувати темп оновлення понад 7×10^3 повних операцій зміни ключа за секунду, що є критично важливим для систем із частою переініціалізацією контексту (наприклад, у протоколах зі стрибкоподібною зміною частоти).

4.3.4. Вплив ізоляції ядер на стабільність керування

Для перевірки узгодженості результатів із висновками розділу 2 було досліджено вплив просторової ізоляції обчислювальних ядер(*Spatial Isolation*) на часову стабільність операцій керування [125, 126] . На рис. 4.10 наведено

порівняння профілів затримки каналу керування в умовах синтетичного навантаження без ізоляції та з її застосуванням.

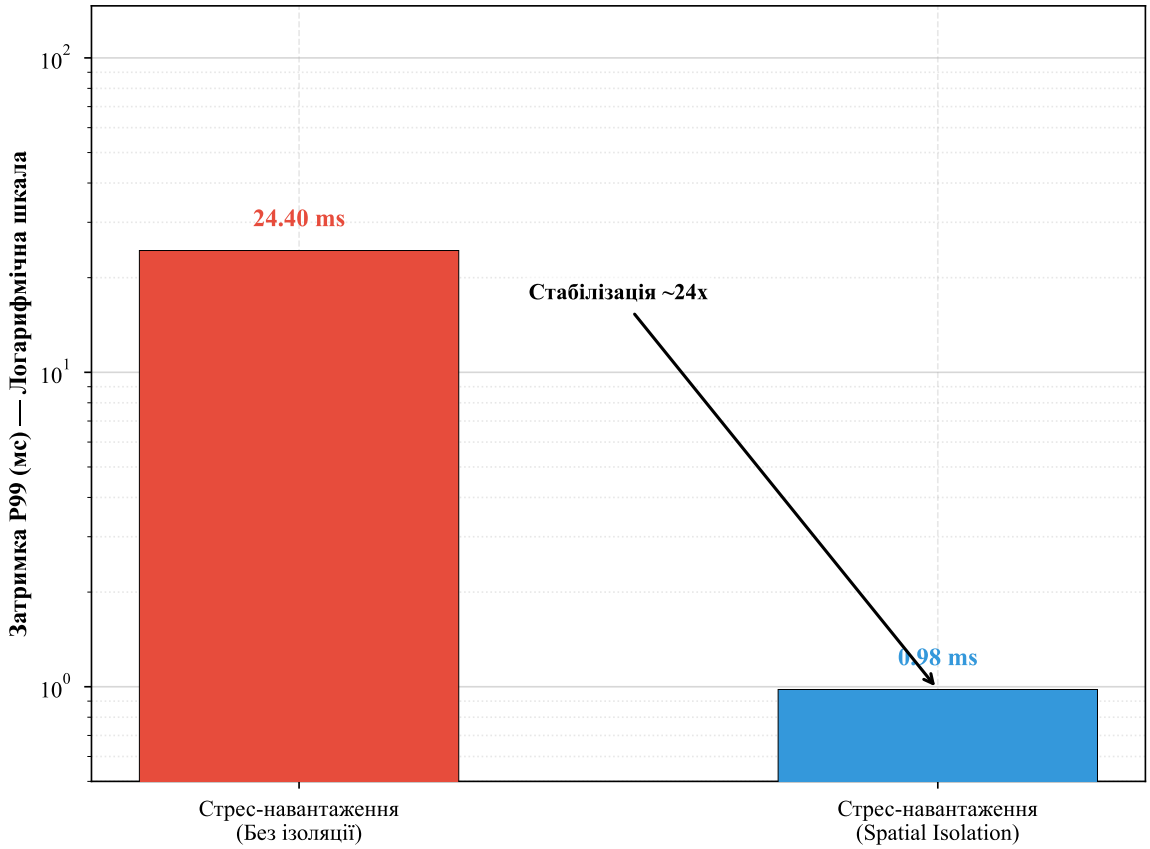


Рис. 4.10: Вплив ізоляції обчислювальних ядер на джитер каналу керування. Застосування методики Spatial Isolation зменшує варіативність затримок.

Результати підтверджують, що навіть при використанні високорівневого програмного стека, часові характеристики Control Plane залишаються передбачуваними за умови коректної конфігурації ОС.

4.3.5. Декомпозиція системної затримки

З метою пояснення обмежень досяжної пропускної здатності було виконано декомпозицію часових витрат при обробці одного керуючого циклу (рис. 4.11).

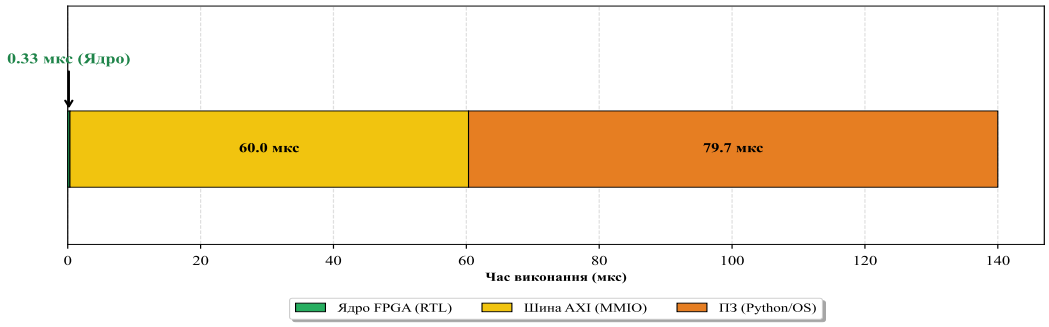


Рис. 4.11: Профіль розподілу часу при обробці одного блоку даних.

Апаратна частина ядра виконує криптографічні перетворення за сотні наносекунд, тоді як транзакції MMIO та програмна обробка займають десятки мікросекунд [127, 128]. Отриманий профіль затримок обґрунтовує доцільність використання режиму MMIO виключно для керування (Control Plane), але не для передачі даних. Експериментально встановлено, що межа пропускної здатності в режимі MMIO становить $\approx$ 450 кбіт/с [129]. Це доводить, що для реалізації магістральних шифраторів (10+ Гбіт/с) керування потоком даних повинно бути повністю відчужене від CPU і передане контролеру **Bus Mastering DMA**, що є напрямком подальшого масштабування архітектури.

4.3.6. Зведені характеристики та висновок підрозділу

Показано, що Control Plane принципово не призначений для передачі потокових даних і не здатний розкрити потенціал обчислювального ядра в термінах пропускної здатності [130]. Водночас він забезпечує високу маневреність криптографічного контексту (понад 7000 оновлень ключа на секунду), детерміновану часову поведінку та гнучкість інтеграції у програмні системи.

Отримані результати експериментально підтверджують дихотомію архітектури: *Data Plane* (розділ 4.2) відповідає за високошвидкісну обробку даних у програмованій логіці, тоді як *Control Plane* забезпечує надійне та кероване використання цього потенціалу, що є фундаментальною передумовою подальшого масштабування системи із застосуванням DMA та потокових інтерфейсів [131, 132].

Узагальнені експериментальні характеристики розробленої підсистеми керування наведено в табл. 4.4.

Таблиця 4.4

Зведені характеристики та результати верифікації підсистеми керування

Параметр	**Значення**	**Коментар**
Key Agility (оновлень/с)	> 7000	Достатньо для FHSS-протоколів.
Latency (next $\rightarrow$ ready)	Константа	Визначається виключно RTL-логікою та є інваріантною до навантаження на ОС
Jitter (P_{99} з ізоляцією)	< 1 мс	Повна апаратна ізоляція тактового домену ядра від навантаження ОС.
Re-init Stability	Успішно	Забезпечено стійкість до м'якого перезапуску без апаратного скидання живлення.
Частка MMIO у затримці	$\sim 99\%$	Підтверджує необхідність використання DMA для Data Plane.

Як відображено в таблиці, частка MMIO у затримці має значення $\sim 99\%$ що підтверджує необхідність використання DMA.

4.4. Аналіз потенціалу масштабування та архітектурна верифікація платформи

Результати експериментальних досліджень, отримані у підрозділах 4.2 та 4.3, дозволяють виконати фінальну архітектурну верифікацію розробленої платформи та оцінити її потенціал масштабування у складі високонавантажених телекомунікаційних та вбудованих систем.

На відміну від попередніх етапів, де окремо аналізувалися *тракт даних* (Data Plane) та *контур керування* (Control Plane), цей розділ представляє результати **дослідження простору проєктування (Design Space Exploration)**, розглядаючи платформу як єдину гетерогенну систему з точки зору масштабування, стабільності та фізичних обмежень [133].

4.4.1. Ресурсна ефективність та екстраполяція продуктивності

Аналіз результатів логічного синтезу (табл. 4.2) продемонстрував високу компактність розробленого RTL-ядра потокового шифрування. Один екземпляр прискорювача займає **2386 ALM**, що становить менше **6%** логічної ємності кристала Cyclone V SE A6.

У межах гібридного маршруту високорівневого проектування, системна інфраструктура (інтерконект AXI, адаптери, DMA) синтезується засобами системного рівня (*Platform Designer* / *HLS*), тоді як обчислювально-критичне ядро потокового перетворення оптимізується на рівні RTL для гарантування детермінованої затримки [134].

Виконуючи екстраполяцію на повну ємність кристала (із запасом 20% ресурсів на інфраструктуру інтерконекту, контролери DMA та сервісні модулі), на одній мікросхемі середнього цінового сегмента можливо розмістити кластер із $N \approx 13$ незалежних криптографічних ядер [133, 135]. Агрегована пропускна здатність такого кластера в режимі *тракту даних* становитиме:

$$\mathrm{Thr}_{\Sigma} = N \times \mathrm{Thr}_{\mathrm{core}} \approx 13 \times 6{,}4 \text{ Гбіт/с} = 83{,}2 \text{ Гбіт/с} \tag{4.2}$$

На рис. 4.12 наведено теоретичну модель масштабування продуктивності кластера у порівнянні з фізичними обмеженнями платформи.

Для коректної оцінки реалізаційності такої системи необхідно врахувати фізичні обмеження, відображені на графіку:

1. **Обмеження пропускної здатності пам'яті (Memory Wall [136]):** Розрахункова агрегована продуктивність (83.2 Гбіт/с) перевищує фізичну пропускну здатність зовнішнього інтерфейсу DDR3 (для Cyclone V SoC $\approx$ 25 Гбіт/с). Це вказує на необхідність використання **внутрішньої пам'яті FPGA (On-Chip Memory)** для буферизації або переходу на платформи вищого класу (наприклад, Intel Arria 10/Agilex) з підтримкою DDR4/HBM [137].
2. **Теплові обмеження:** Наведена модель масштабування ($N = 13$) є оцінкою **логічної ємності** кристала. У реальних умовах одночасна робота всіх ядер може бути обмежена температурними характеристиками. Це вимагає динамічного керування, підтримка якого закладена у розробленому адаптері керування.

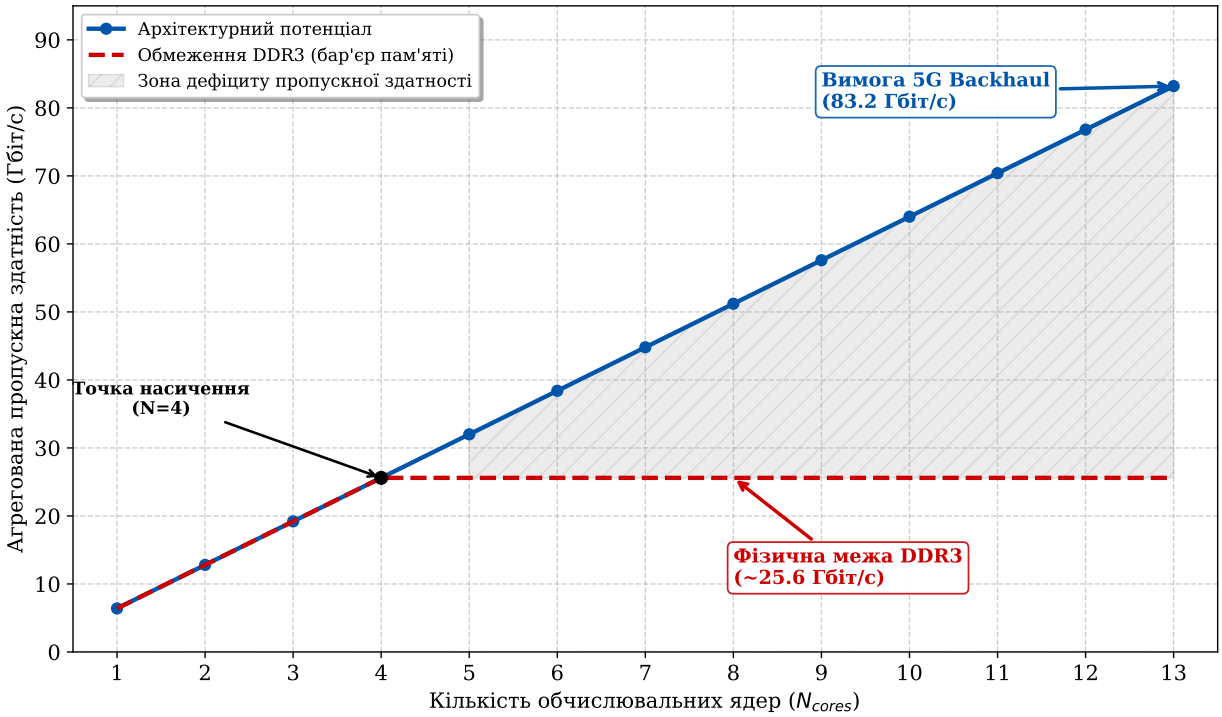


Рис. 4.12: Теоретична модель масштабування продуктивності кластера та вплив обмежень пропускної здатності системної пам’яті (ефект «бар’єра пам’яті»).

Такий перелік формалізує компроміси між пропускною здатністю, затримку та ресурсними обмеженнями, що робить екстраполяцію продуктивності відтворюваною та порівнюваною між варіантами реалізації [135]. Тим не менш, навіть з урахуванням цих обмежень, отримана оцінка відповідає вимогам стандартів **ITU-R M.2410** та **IEC 61850-5** до пікової швидкості передачі даних для сценаріїв eMBB, що дозволяє розглядати розроблену архітектуру як перспективну для магістральних телекомунікаційних вузлів (**backhaul links**) мереж 5G та концентраторів даних промислового інтернету речей (**IIoT**) [138, 139].

4.4.2. Оцінка підсистеми оркестрації (масштабування контуру керування)

При масштабуванні кластера прискорювачів визначальним обмежувальним фактором стає не обчислювальна продуктивність, а здатність центрального процесора (HPS) своєчасно виконувати реконфігурацію криптографічних контекстів.

Експериментально підтверджена у п. 4.3 швидкість обробки транзакцій керування ($>$ 7000 **OPS**) гарантує, що навіть при повному завантаженні кластера з 13 ядер, центральний процесор здатен ефективно керувати системою. Час

реконфігурації одного ядра становить $T_{reconf} \approx 140$ μс, відповідно час повної ротації ключів для всього кластера оцінюється як:

$$T_{cluster} = N \cdot T_{reconf} \approx 13 \times 140 \ \mu\text{с} \approx 1.82 \text{ мс}. \tag{4.3}$$

Таким чином, підсистема керування, реалізована на базі стандартної ОС Linux, здатна виконувати понад **500 повних циклів реконфігурації** всього кластера за секунду, що усуває вузьке місце в контурі керування (Control Plane Bottleneck). Слід наголосити, що здатність Linux-середовища з фазовим джитером $\approx$ 1 мс керувати синхронними стрибками ППРЧ (де крок становить 140 мкс) забезпечується саме завдяки архітектурі тіньових регістрів (*Shadow Register File*), описаній у підрозділі 4.3.1. Процесорна підсистема HPS працює поза критичним часовим контуром (*Out-of-Band*), виконуючи лише асинхронне попереднє завантаження пулу ключів (*Throughput* $>$ 7000 OPS). Натомість синхронна атомарна трансляція цих ключів у тракт даних відбувається за командою `START`, що генерується апаратними таймерами FPGA, гарантуючи мінімалізації затримки.

4.4.3. Забезпечення живучості та відмовостійкості.

Важливою архітектурною перевагою кластерного підходу є забезпечення **плавної деградації** (*Graceful Degradation*). Вихід з ладу одного з N ядер не призводить до зупинки всієї системи, а лише до зниження сумарної пропускної здатності на $1/N$. Підсистема оркестрації (Linux) здатна детектувати несправність через механізм тайм-ауту шини (AXI Timeout) і динамічно виключити пошкоджене ядро з пулу активних ресурсів.

Отримана динаміка дозволяє реалізувати концепцію **«безшовної» зміни ключів** (*Seamless Rekeying*): час повного оновлення кластера (1.82 мс) є значно меншим за тривалість стандартного радіокадру 5G NR (10 мс). Це критично важливо для систем динамічного доступу до спектра (DSA) та програмно-визначених мереж (SDN), де параметри безпеки повинні змінюватися адаптивно до завадової обстановки.

4.4.4. Роль ізоляції у забезпеченні QoS та часової стабільності

Ефективне керування масштабованим кластером було б неможливим без забезпечення передбачуваної часової поведінки керуючих транзакцій. Результати розділів 2 та 4 демонструють ієрархічну структуру часових характеристик платформи:

- **Рівень ядра ОС** ($T_{kern} \approx 13$ μс): низький джитер планувальника, досягнутий у п. 2.9, формує фундамент часової стабільності, знижуючи шуми ОС [125].
- **Рівень системи** ($T_{sys} \approx 1$ мс): попри накладні витрати інтерпретатора Python та транзакцій шини MMIO, методика просторової ізоляції (*Spatial Isolation*) дозволяє утримувати інтегральну затримку керування в межах 1 мс для 99.9% команд.

Цей результат дозволяє класифікувати платформу як систему **м'якого реального часу** (*Soft Real-Time*). Забезпечена затримка ($P_{99} < 1$ мс) із **трикратним запасом** перекриває вимоги стандарту **IEC 61850-5** (клас Type 1A, Fast Trip, поріг 3 мс) для систем автоматизації енергетики, а також задовольняє вимоги **ITU-R M.2410** до підсистем керування 5G URLLC (затримка контуру керування < 10 мс).

4.4.5. Архітектурний перехід до апаратного розвантаження (Offloading)

Експерименти з підсистемою введення-виведення (п. 4.3) виявили суттєву диспропорцію між обчислювальною продуктивністю криптографічного ядра (порядку Гбіт/с) та пропускною здатністю програмно-керованого інтерфейсу MMIO (порядку сотень кбіт/с).

Цей результат вказує на те, що регістровий доступ через MMIO є ефективним виключно для задач керування (де переважають операції запису з низькою затримкою), але фізично не здатний забезпечити передавання потокових даних у магістральних режимах (де критичною є затримка читання) [140]. Узагальнюючи отримані результати, архітектура платформи природно формалізується у вигляді моделі **HPS-centric Control / FPGA-centric Data**, та наведена на рис. 4.13. Згідно з цією моделлю:

- **Контур керування (HPS/MMIO):** Використовується для команд конфігурації, моніторингу стану та оновлення ключів (сині пунктирні лінії), де пріоритетом є мінімізація часової затримки реакції.

- **Тракт даних (FPGA/DMA):** Для розкриття потенціалу понад 80 Гбіт/с необхідне повне відчуження потоків даних від CPU. Використання **еластичного буфера (Elastic FIFO)** дозволяє узгодити пікову швидкість внутрішньої шини з пропускною здатністю зовнішньої пам'яті (Rate Matching), а застосування контролерів прямого доступу до пам'яті з підтримкою режимів **Scatter-Gather (mSGDMA)** нівелює затримки на обслуговування переривань [115].

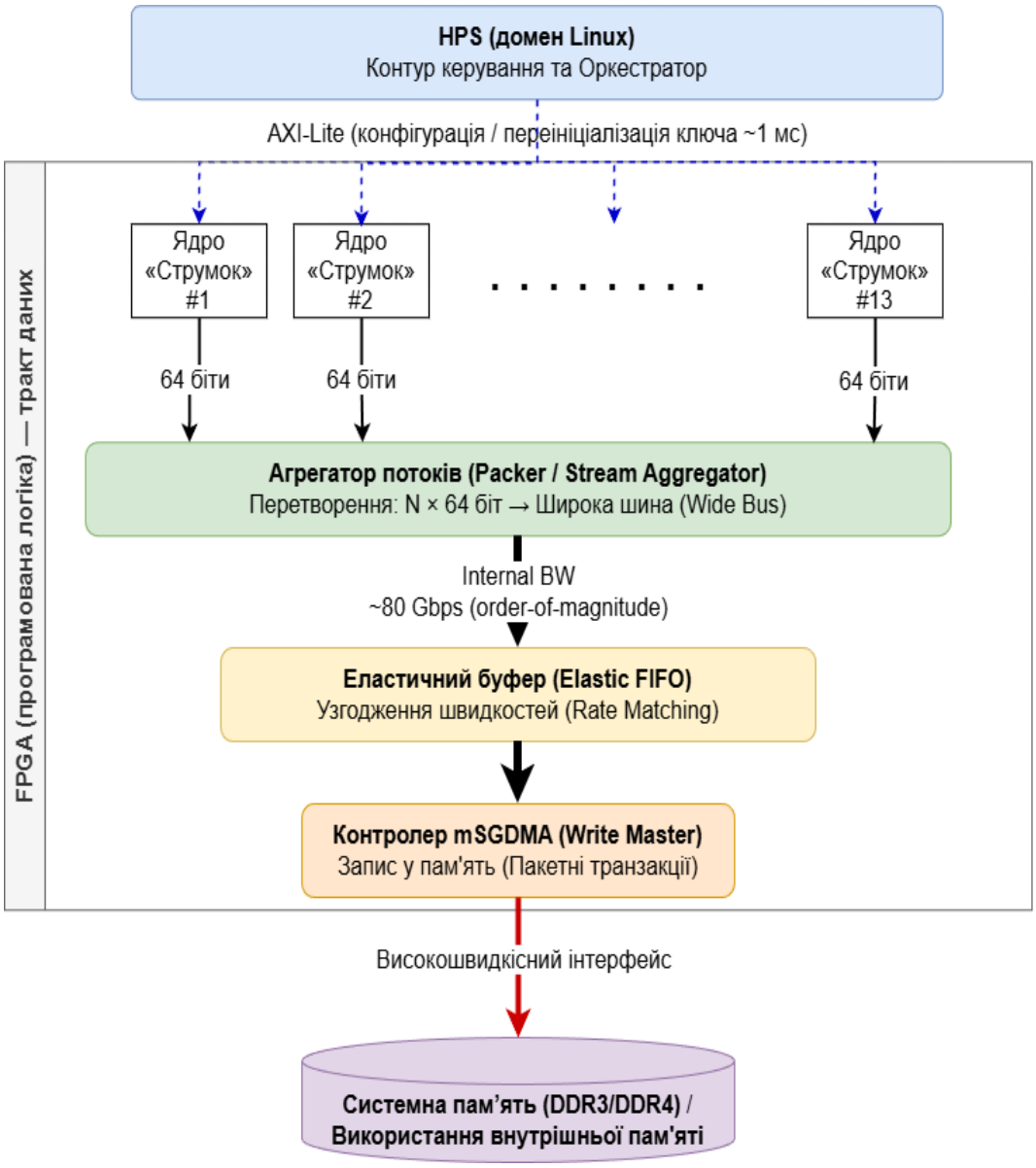


Рис. 4.13: Перспективна архітектура кластера з розділенням контурів керування та даних (модель апаратного розвантаження).

Підсумовуючи еволюцію архітектурних рішень у даній роботі, у табл. 4.5 наведено порівняльний аналіз розробленої платформи відносно початкових програмних реалізацій.

Таблиця 4.5

Еволюція архітектури системи: від програмної моделі до апаратного кластера

Характеристика	**Програмна модель (Розділ 3)**	**Гібридна (Розділ 4.3)**	**Багатоядерний кластер (Розділ 4.4)**
Виконавець	CPU (NEON/FPU)	FPGA (1 ядро)	**FPGA (13 ядер)**
Роль керуючого ПЗ	Обробка + Керування	Драйвер	**Оркестрація**
Тракт даних	≈ 0.6 Гбіт/с	6.4 Гбіт/с	≈ 83.2 **Гбіт/с**
Затримки	Недетермінована	$<$ 1 мс (Soft RT)	$<$ 2 **мс**
Масштабування	Вертикальне (CPU)	-	**Горизонтальне**

Отримані висновки формують завершене архітектурне обґрунтування подальшого розвитку платформи у напрямку повноцінного **апаратного розвантаження** (*Hardware Offloading*), що розглядається як логічне та еволюційне продовження цієї роботи.

4.5. Висновки до розділу 4

У четвертому розділі розв’язано ключову науково-технічну проблему дисертаційної роботи — подолання фундаментальних обмежень пропускної здатності програмних **трактів цифрової обробки сигналів**. На основі виявленої у попередньому розділі «точки насичення» процесорної підсистеми (20 MSPS), обґрунтовано та реалізовано архітектурний перехід до моделі апаратного розвантаження (*Hardware Offloading*). Доведено, що для досягнення швидкостей, характерних для сучасних широкосмугових телекомунікаційних систем, необхідним є перенесення обчислювально-критичних задач із площини процесора (CPU) у площину програмованої логіки (FPGA).

Для практичної верифікації запропонованого підходу застосовано гібридний маршрут проєктування, у межах якого здійснено синтез спеціалізованого апаратного прискорювача потокового перетворення (на прикладі алгоритму ДСТУ 8845:2019 «Струмок») на рівні регістрових передач (RTL). Відмова від послідовної інструкційної моделі фон Неймана на користь просторово розгорнутої комбінаційної логіки дозволила забезпечити виконання складних нелінійних перетворень у полях Галуа за один тактовий цикл. Аналіз ресурсної ефективності показав, що розроблене IP-ядро займає менше 6 % логічної ємності кристала Cyclone V, при цьому забезпечуючи детерміновану пропускну здатність

на рівні 6,4 Гбіт/с на одне ядро. Теоретична екстраполяція продуктивності доводить можливість лінійного масштабування кластера прискорювачів до агрегованої пропускної здатності понад 80 Гбіт/с, що задовольняє вимоги магістральних каналів зв’язку.

Визначальним здобутком розділу є системна інтеграція розробленого апаратного тракту в гетерогенне керуюче середовище Linux. Шляхом застосування механізму прямого регістрового доступу (MMIO) та просторової ізоляції ресурсів, реалізовано адаптерний шар, який виступає прозорим мостом між операційною системою та апаратною логікою. Експериментально доведено, що така архітектура забезпечує надвисоку маневреність керуючих команд (*Key Agility*) на рівні понад 7000 оновлень за секунду при фазовому джитері контуру керування менше 1 мс.

Таким чином, еволюційний перехід від програмної до апаратної обробки дозволив не лише подолати бар’єр продуктивності, але й повністю вивільнити ресурси центрального процесора для задач оркестрації, маршрутизації та мережевої взаємодії. Отримані результати завершують побудову цілісної методики синтезу вузлів ЦОС, яка успішно поєднує гнучкість вбудованих операційних систем із безкомпромісною швидкодією та детермінованістю апаратних трактів FPGA, що повністю розв’язує головну наукову задачу дисертаційного дослідження.

ВИСНОВКИ

У дисертаційній роботі розв’язано важливу науково-прикладну задачу підвищення ефективності функціонування та скорочення часу проєктування **вузлів цифрової обробки сигналів** у сучасних телекомунікаційних системах. Поставленої мети досягнуто шляхом розроблення та експериментального обґрунтування комплексної методики синтезу гетерогенних платформ на базі SoC FPGA, що поєднує гнучкість керування операційних систем загального призначення (Linux) із детермінізмом та високою пропускною здатністю апаратних трактів програмованої логіки.

Фундаментом для реалізації запропонованої методики стало розв’язання проблеми фазової невизначеності (джитера) у програмних контурах керування радіотрактом. Вперше запропоновано та впроваджено концепцію «Паспорта конфігурації», яка базується на строгій математичній моделі простору станів вектора $S_{\text{passport}} = \{H, M, R\}$. Це дозволило формалізувати стан операційної системи та апаратури, зводячи систематичну похибку вимірювань системних затримок до нуля. Практичне застосування цього підходу у поєднанні з методикою просторової ізоляції ресурсів (*Spatial Isolation*) дозволило знизити максимальну латентність обробки у понад 3 рази (з 151 мкс до 48 мкс) та зафіксувати фазовий джитер керуючих сигналів на рівні $P_{99} < 1$ мс. У контексті телекомунікаційних систем це гарантує надійну фіксацію станів арбітражу системних шин, усуває фазовий шум при DMA-транзакціях та легітимізує використання ОС Linux для задач м’якого реального часу (*Soft Real-Time*) на рівні кордонних (*Edge*) обчислень.

На базі створеного детермінованого керуючого середовища було реалізовано та досліджено архітектуру вузлів із чітким розмежуванням площин обробки даних (*Data Plane*) та керування (*Control Plane*). Розроблено метод просторової пеленгації джерел радіовипромінювання з борту безпілотного літального апарата, що дозволило суттєво скоротити час пошуку завдяки когерентному обробленню даних у межах одного кристала. Водночас результати стрес-тестування виявили фундаментальні обмеження процесорної підсистеми. Експериментально встановлено критерій «точки насичення» (*Saturation Point*) для вбудованих

універсальних процесорів, який зафіксовано на рівні потоку даних у 20 MSPS. Доведено, що при перевищенні цієї межі виникає ефект «шторму переривань», що спричиняє критичну деградацію пропускної здатності та руйнує цілісність сигнальних кадрів.

Для подолання виявленого архітектурного бар'єра було обґрунтовано та здійснено перехід до моделі апаратного розвантаження (*Hardware Offloading*). У межах гібридного маршруту проєктування виконано синтез спеціалізованого апаратного прискорювача потокового перетворення (ДСТУ 8845:2019 «Струмок») на рівні регістрових передач (RTL). Експериментально доведено, що перенесення нелінійної арифметики полів Галуа у програмовану логіку забезпечує детерміновану пропускну здатність на рівні 6,4 Гбіт/с на одне обчислювальне ядро з можливістю лінійного масштабування агрегованої пропускної здатності до понад 80 Гбіт/с. Розроблений адаптерний шар забезпечив прозору системну інтеграцію RTL-ядра в середовище ОС, що дозволило досягти маневреності керуючих команд понад 7000 оновлень за секунду. Це є критично важливою умовою для забезпечення функціональної стійкості захищених радіоканалів із псевдовипадковим перелаштуванням робочої частоти (ППРЧ / FHSS).

У підсумку, еволюційний перехід від програмної до апаратної обробки дозволив не лише подолати обмеження продуктивності, але й повністю вивільнити ресурси центрального процесора для задач інтелектуальної оркестрації, маршрутизації та мережевої взаємодії. Отримані наукові та практичні результати утворюють завершену методологію, яка відкриває широкі перспективи для розроблення сучасних програмно-визначених радіосистем (SDR), базових станцій 5G/6G, комплексів радіоелектронної розвідки та автономних сенсорних мереж, забезпечуючи оптимальний баланс між продуктивністю апаратних матриць та гнучкістю вбудованого програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[3] M. Chiang and T. Zhang, “Fog and IoT: An overview of research opportunities,” *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[4] S. S. Gill *et al.*, “Edge AI: A taxonomy, systematic review and future directions,” *ACM Computing Surveys*, 2024. arXiv:2407.04053.

[5] R. Woods, J. McAllister, G. Lightbody, and Y. Yi, *FPGA-based Implementation of Signal Processing Systems*. John Wiley & Sons, 2 ed., 2017.

[6] J. J. Rodríguez-Andina, M. J. Moure, and M. D. Valdés, “Features, design tools, and application domains of fpgas,” *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1810–1823, 2007.

[7] M. R. Z. Chowdhury, A. Seum, M. R. Talukder, *et al.*, “Towards next-generation fpga-accelerated vision-based autonomous driving: A comprehensive review,” *Signals*, vol. 6, no. 4, p. 53, 2025.

[8] M. J. Oliva, P. A. García, E. M. Spinelli, and A. L. Veiga, “Soc-fpga systems for the acquisition and processing of electroencephalographic signals,” *International Journal of Reconfigurable and Embedded Systems*, vol. 10, no. 3, pp. 237–248, 2021.

[9] I. Kuon and J. Rose, “Measuring the gap between fpgas and asics,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[10] R. Nane, V. M. Sima, C. Pilato, W.-J. Chan, B. Fort, A. Canis, J. Choi, S. Brown, F. Ferrandi, J. H. Anderson, and K. Bertels, “A survey and evaluation of FPGA high-level synthesis tools,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2016.

[11] E. Del Sozzo, D. Conficconi, A. Zeni, M. Salaris, D. Sciuto, and M. D. Santambrogio, “Pushing the level of abstraction of digital system design: A

survey on how to program fpgas," *ACM Computing Surveys*, vol. 55, no. 5, pp. 106:1–106:48, 2023.

[12] Y. Herman, H. Lastivka, and A. Samila, "Embedded operating systems in iot edge computing," *Security of Infocommunication Systems and Internet of Things*, vol. 2, no. 1, p. 02001, 2024.

[13] K. Yaghmour, J. Masters, G. Ben-Yossef, and P. Gerum, *Building Embedded Linux Systems*. O'Reilly Media, 2 ed., 2018.

[14] C. Hallinan, *Embedded Linux Primer: A Practical Real-World Approach*. Prentice Hall, 2 ed., 2010.

[15] C. Simmonds, *Mastering Embedded Linux Programming*. Packt Publishing, 3 ed., 2021.

[16] Y. Herman, O. Krulikovskyi, S. Haliuk, and S. Subbotin, "Development of an embedded operating system based on the linux kernel for soc fpga," in *Proceedings of the Seventh International Workshop on Computer Modeling and Intelligent Systems (CMIS-2024)*, vol. 3702 of *CEUR Workshop Proceedings*, (Zaporizhzhia, Ukraine), May 2024.

[17] Y. Herman and A. Samila, "Modular edge system for data acquisition and analysis based on raspberry pi and soc fpga," *Herald of Khmelnytskyi National University. Technical Sciences Series*, vol. 357, no. 5.2, pp. 86–92, 2025.

[18] Y. Herman and A. Veryha, "Offline radio signal processing and visualization system," *Measuring and Computing Devices in Technological Processes*, vol. 84, no. 4, pp. 246–252, 2025.

[19] Y. Herman, O. Krulikovskyi, and A. Veryha, "Autonomous system for processing and visualization of radio signals," in *Proceedings of the Fourteenth International Scientific and Practical Conference on Informatics and Computer Engineering Problems (PICT-2025)*, (Chernivtsi, Ukraine), pp. 188–190, Tekhnodruk, 2025.

[20] O. Krulikovskyi, Y. Herman, and O. Verenko, "Software–hardware interaction in soc fpga cyclone v systems," in *Proceedings of the Tenth International Scientific and Practical Conference on Physical and Technological Problems of Transmission, Processing and Storage of Information in Infocommunication Systems*, (Chernivtsi, Ukraine), p. 45, May 2025.

[21] O. Krulikovskyi, Y. Herman, and O. Verenko, "Hardware/software communication architecture for a soc-fpga-based time-to-digital converter," in

Proceedings of the Seventeenth International Conference on Correlation Optics, vol. 13813 of *Proceedings of SPIE*, (Chernivtsi, Ukraine), pp. 214–217, SPIE, 2025.

[22] Y. Herman and O. Krulikovskyi, "Development and application of software for soc fpga cyclone v systems," in *Proceedings of the Twelfth International Scientific and Practical Conference on Modern Problems and Achievements in Radio Engineering, Telecommunications and Information Technologies*, (Zaporizhzhia, Ukraine), pp. 27–29, Dec. 2024.

[23] I. Kuon, R. Tessier, and J. Rose, "Fpga architecture: Survey and challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[24] Y. Herman, "Efficiency evaluation of embedded systems based on linux and fpga," in *Proceedings of the 29th International Youth Forum on Radio Electronics and Youth in the XXI Century*, vol. 3, (Kharkiv, Ukraine), pp. 540–542, 2025.

[25] Y. Herman, "Fpga platforms and their use in edge computing," *Security of Infocommunication Systems and Internet of Things*, vol. 3, no. 2, p. 02008, 2025.

[26] K. Vipin and S. A. Fahmy, "Fpga dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–39, 2018.

[27] M. J. Oliva, P. A. García, E. M. Spinelli, and A. L. Veiga, "Open-source soc-fpga platform for signal processing," in *Proceedings of the XI Southern Programmable Logic Conference (SPL 2023)*, (San Luis, Argentina), 2023.

[28] U. Farooq, Z. Marrakchi, and H. Mehrez, "Fpga architectures: An overview," in *Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*, pp. 7–48, Springer, 2012.

[29] L. Lahti, M. Nyberg, P. Själund, J. Vanne, and T. D. Hämäläinen, "Are we there yet? A study on the state of high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898–911, 2019.

[30] Z. Huang, X. Chi, J. Cong, P. Huang, Y. Liang, and X. Xiong, "A survey on performance optimization of high-level synthesis tools," *ACM Computing Surveys*, vol. 53, no. 4, pp. 77:1–77:37, 2020.

[31] A. Ottimo, G. Mencagli, and M. Danelutto, "Fsp: A framework for data stream

processing applications targeting fpgas," in *2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 92–99, IEEE, 2023.

[32] R. Novickis and M. Greitans, "Fpga master based on-chip communications architecture for cyclone v soc running linux," in *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*, IEEE, 2018.

[33] R. Meeuws, M. Ostaszewski, R. Wain, *et al.*, "High level synthesis for FPGAs: From C to hardware," in *Proceedings of the International Conference on Signal Processing and Multimedia Applications (SIGMAP 2012)*, (Rome, Italy), pp. 184–187, SciTePress, 2012.

[34] J. Sérot, F. Berry, and S. Ahmed, *CAPH: A Language for Implementing Stream-Processing Applications on FPGAs*, pp. 201–224. New York, NY: Springer New York, 2013.

[35] O. Salvador and D. Angolini, *Embedded Linux Development using the Yocto Project*. Packt Publishing, 2014.

[36] Altera Corporation, "Bare-metal, rtos, or linux? optimize real-time performance with altera socs," Dec. 2014. White Paper WP-01245-1.0.

[37] Intel Corporation, "An 796: Cyclone v and arria v soc device design guidelines," Tech. Rep. 683360, Intel, Santa Clara, CA, USA, 2022.

[38] Intel, *Golden System Reference Design (GSRD) User Manuals*, 2025. Check specific URL for target device (Cyclone V / Arria 10).

[39] Intel, *Intel SoC FPGA Embedded Development Suite (SoC EDS) User Guide*, Feb. 2025.

[40] Intel, *Intel SoC FPGA Embedded Development Suite (SoC EDS) User Guide*, Feb. 2025.

[41] The Yocto Project, *The Yocto Project Documentation*, 2025.

[42] Buildroot, *Buildroot Documentation*, 2025.

[43] Intel Corporation, "Cyclone v hard processor system technical reference manual," Tech. Rep. 683126, Intel Corporation, Santa Clara, CA, USA, Nov. 2022.

[44] Intel Corporation, "Intel® SoC FPGA Embedded Development Suite (SoC EDS) User Guide," User Guide 683187, Intel Corporation, Santa Clara, CA, USA, Feb. 2025. Available online.

[45] J. Vieira, “Assessing on-chip high-performance interfaces on xilinx and intel fpga-soc devices,” *Reconfigurable Computing*, pp. 1–12, 2019. Benchmarking of FPGA-to-HPS and memory interfaces.

[46] The Linux Kernel Project, *DMA Engine API Guide*, 2024.

[47] The Linux Kernel Project, *FPGA Manager Subsystem*, 2024.

[48] E. Zaffaroni and G. Haefeli, “The cubodaq data acquisition system,” *arXiv preprint arXiv:2410.00190*, 2024.

[49] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 6 ed., 2019.

[50] R. Love, *Linux Kernel Development*. Addison-Wesley, 3 ed., 2010.

[51] A. Baumann, P. Barham, P.-E. Dagand, *et al.*, “The multikernel: A new os architecture for scalable multicore systems,” in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP ’09)*, pp. 29–44, 2009.

[52] Devicetree.org, “Devicetree specification,” 2021. Version 0.4.

[53] The Linux Kernel Project, *The Linux Kernel Documentation: Kconfig Language*, 2024.

[54] D. Gibson and B. Herrenschmidt, “Device trees everywhere.” White paper / technical report, Feb. 2006.

[55] R. Tartler, D. Lohmann, J. Sincero, *et al.*, “Feature consistency in compile-time-configurable system software,” in *Proceedings of the 6th European Conference on Computer Systems (EuroSys 11)*, pp. 47–60, 2011.

[56] P. Arcaini, A. Gargantini, and M. Radavelli, “A process for fault-driven repair of constraints among features,” in *Proceedings of the 23rd International Systems and Software Product Line Conference (SPLC) - Volume B*, pp. 73–81, 2019.

[57] E. Kuiter, C. Sundermann, T. Thüm, T. Heß, S. Krieter, and G. Saake, “How configurable is the linux kernel? analyzing two decades of feature-model history,” *ACM Trans. Softw. Eng. Methodol.*, vol. 35, no. 1, 2025.

[58] B. Adams, K. De Schutter, H. Tromp, and W. De Meuter, “The evolution of the linux build system,” *Electronic Communications of the EASST*, vol. 8, Feb. 2008.

[59] C. Lamb and S. Zacchiroli, “Reproducible builds: Increasing the integrity of software supply chains,” *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2022.

[60] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*. O’Reilly Media, 3 ed., 2005.

[61] F. Reghenzani, G. Massari, W. Fornaciari, and M. D. Santambrogio, "The real-time linux kernel: A survey on preempt_rt," *ACM Computing Surveys*, vol. 52, no. 1, pp. 18:1–18:36, 2019.

[62] Altera Corporation, "Optimizing real-time performance with altera socs," tech. rep., Altera, 2015. White Paper WP-01245.

[63] S. Nadi and R. Holt, "The linux kernel: a case study of build system variability," *Journal of Software: Evolution and Process*, 2013.

[64] R. G. Minnich and A. Mirtchovski, "U-root: A go-based, firmware embeddable root file system with on-demand compilation," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pp. 577–586, 2015. Corrected citation for u-root embedded fs.

[65] J. Kaur and S. Reddy, "Implementation of linux optimization technique for arm based system on chip," *Procedia Computer Science*, vol. 171, pp. 1780–1789, 2020. Third International Conference on Computing and Network Communications (CoCoNet'19).

[66] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate iot edge computing with lightweight virtualization," *IEEE Network*, vol. 32, no. 1, pp. 102–111, 2018.

[67] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 20–26, 2016.

[68] E. Dolstra, A. Löh, and N. Pierron, "Nixos: A purely functional linux distribution," *Journal of Functional Programming*, vol. 20, no. 5-6, pp. 577–615, 2010.

[69] A. Barbalace, M. Sadini, S. Ansary, C. Jelesnianski, A. Ravichandran, C. Kendir, A. Murray, and B. Ravindran, "Popcorn: bridging the programmability gap in heterogeneous-isa platforms," in *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, (New York, NY, USA), Association for Computing Machinery, 2015.

[70] M. H. Quraishi, E. B. Tavakoli, and F. Ren, "A survey of system architectures and techniques for fpga virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2216–2230, 2021. Valid source for FPGA Virtualization architectures.

[71] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter

services," in *ISCA '14: Proceedings of the 41st Annual International Symposium on Computer Architecture*, pp. 13–24, 2014.

[72] C. Bobda *et al.*, "The future of fpga acceleration in datacenters and the cloud," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 3, 2022.

[73] X. Li, X. Wang, F. Liu, and H. Xu, "Dhl: Enabling flexible software network functions with fpga acceleration," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1–11, 2018.

[74] V. Choudhary *et al.*, "Performance evaluation of pl330 dma controller for bulk data transfer in zynq soc," in *IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology*, 2016.

[75] J. Fjeldtvedt, M. Orlandic, and T. A. Johansen, "Cubedma – optimizing three-dimensional dma transfers for hyperspectral imaging applications," *Microprocessors and Microsystems*, vol. 65, pp. 23–36, 2019.

[76] X. Xiong *et al.*, "Analysis of the impact on soft error in axi cdma based on xilinx zynq-7000 fpga," *Nuclear Instruments and Methods in Physics Research A*, vol. 1037, p. 166913, 2022.

[77] J. Wang, G. Lv, Z. Liu, and X. Yang, "Programmable deterministic zero-copy dma mechanism for fpga accelerator," *Applied Sciences*, vol. 12, no. 19, p. 9581, 2022.

[78] A. R. Bucknall, *Build Framework and Runtime Abstraction for Partial Reconfiguration on FPGA SoCs*. PhD thesis, University of Warwick, 2022.

[79] D. Rossi *et al.*, "Exploiting acp for energy-efficient hardware acceleration in soc fpgas," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 222–223, 2017.

[80] J. J. Patel, N. Reddy, P. Kumari, R. Rajpal, H. Pujara, R. Jha, and P. Kalappurakkal, "Embedded linux platform for data acquisition systems," *Fusion Engineering and Design*, vol. 89, no. 5, pp. 684–688, 2014. Proceedings of the 9th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research.

[81] A. Rigoni Garola, G. Manduchi, M. Gottardo, R. Cavazzana, M. Recchia, C. Taliercio, and A. Luchetta, "A zynq-based flexible adc architecture combining real-time data streaming and transient recording," *IEEE Transactions on Nuclear Science*, vol. PP, pp. 1–1, 11 2020.

[82] A. Motakis, A. Rigo, and D. Raho, "Platform device assignment to kvm-on-arm virtual machines via vfio," in *2014 IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 170–177, 2014. Critical for FPGA-to-VM passthrough.

[83] J. Goeders and S. J. E. Wilton, "Effective FPGA debug for high-level synthesis generated circuits," in *Proceedings of the 2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8, IEEE, 2014.

[84] V. Silva *et al.*, "The artico^3 framework: A hardware/software solution for on-demand customization of fpga-based embedded systems," *Sensors*, vol. 18, no. 6, p. 1877, 2018.

[85] S. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Fred-linux: A linux-based support for real-time applications on heterogeneous platforms," *Future Generation Computer Systems*, vol. 129, pp. 244–260, 2022.

[86] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[87] G. C. Buttazzo, L. Abeni, M. Caccamo, and G. Lipari, *Soft real-time systems: Predictability vs. efficiency*. Springer, 01 2005.

[88] M. Barletta, M. Cinque, L. De Simone, and R. Della Corte, "Achieving isolation in mixed-criticality industrial edge systems with real-time containers," in *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*, Leibniz International Proceedings in Informatics (LIPIcs), pp. 15:1–15:23, 2022.

[89] B. Forsberg, A. Marongiu, and L. Benini, "GPUguard: Towards supporting a predictable execution model for heterogeneous SoC," in *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, pp. 318–321, IEEE, 2017.

[90] C. Collberg and T. A. Proebsting, "Repeatability and benefaction in computer systems research," *Communications of the ACM*, vol. 59, no. 3, pp. 62–69, 2016.

[91] C. S. Timperley, L. Herckis, C. Le Goues, and M. Hilton, "Understanding and improving artifact sharing in software engineering research," *Empirical Software Engineering*, 2021.

[92] The Linux Foundation, "Cyclictest." https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start. Available online;.

[93] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[94] H. Fayyad-Kazan, L. Perneel, and M. Timmerman, “Linux preempt-rt v2.6.33 versus v3.6.6: better or worse for real-time applications?,” *SIGBED Rev.*, vol. 11, p. 26–31, Feb. 2014.

[95] A. Burns and R. I. Davis, *Mixed Criticality Systems - A Review:(13th Edition, February 2022)*. University of York, February 2022.

[96] M. Sha, Z. Guo, Y. Guo, and X. Zeng, “A high-performance and flexible architecture for accelerating sdn on the mpsoc platform,” *Micromachines*, vol. 13, no. 11, p. 1854, 2022.

[97] T. Dudley, J. Plusquellic, E. E. Tsiropoulou, J. Goldberg, D. Stick, and D. Lobser, “Scatter-gather dma performance analysis within an soc-based control system for trapped-ion quantum computing,” *IEEE Transactions on Emerging Topics in Computing*, vol. 13, no. 3, pp. 841–852, 2025.

[98] T. H. Pham, S. A. Fahmy, and I. V. McLoughlin, “An end-to-end multi-standard ofdm transceiver architecture using fpga partial reconfiguration,” *IEEE Access*, vol. 5, pp. 21002–21015, 2017.

[99] Z. Cai, P. Li, L. Cheng, D. Yuan, M. Li, and H. Li, “A high performance heterogeneous hardware architecture for brain computer interface,” *Biomedical Engineering Letters*, vol. 15, no. 1, pp. 217–227, 2025.

[100] D. Feitelson, “From repeatability to reproducibility and corroboration,” *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 3–11, 2015.

[101] Y.-H. Tseng, C. Wang, Y.-T. Wei, and Y.-T. Chiang, “Cloud-edge mqtt messaging for latency mitigation and broker memory footprint reduction,” *PeerJ Computer Science*, vol. 11, p. e2741, 2025.

[102] A. Rios-Navarro, R. Tapiador-Morales, G. Jimenez, and A. Linares-Barranco, “Efficient dma transfers management on embedded linux soc for deep-learning gesture recognition (demo),” in *Proceedings of the XX Int. Conference on Human-Computer Interaction*, 2019.

[103] Y. Baek and J. K. Park, “Alex: Adaptive log-embedded extent layer for low-amplification sqlite writes on flash storage,” *Applied Sciences*, vol. 16, no. 2, p. 672, 2026.

[104] K. P. Gaffney, M. Prammer, L. Brasfield, D. R. Hipp, D. Kennedy, and J. M. Patel, “Sqlite: past, present, and future,” *Proc. VLDB Endow.*, vol. 15, p. 3535–3547, Aug. 2022.

[105] ITU-T Study Group 11, “Signalling architecture for microservices based

intelligent edge computing (recommendation itu-t q.5007).” International Telecommunication Union, 2023.

[106] G. Manduchi, L. Trevisan, and T. Patton, “A versatile board for event-driven data acquisition based on zynq soc,” *Electronics*, vol. 13, no. 10, p. 1709, 2024.

[107] F. Ratti, J. Knödtel, and M. Reichenbach, “Heterogeneousrtos: A cpu-fpga real-time os for fault tolerance on cots at near-zero timing cost,” *ACM Trans. Embed. Comput. Syst.*, vol. 24, Feb. 2025.

[108] M. Owaida, G. Alonso, L. Fogliarini, A. Hock-Koon, and P.-E. Melet, “Lowering the latency of data processing pipelines through fpga based hardware acceleration,” *Proc. VLDB Endow.*, vol. 13, p. 71–85, Sept. 2019.

[109] Intel Corporation, “Hardware acceleration in soc fpgas,” tech. rep., Intel, 2025.

[110] State Enterprise “Ukrainian Research and Training Center of Standardization, Certification and Quality”, “Information technology. cryptographic protection of information. algorithm of symmetric stream transformation,” Tech. Rep. DSTU 8845:2019, Ukrainian National Standard, Kyiv, Ukraine, 2019.

[111] A. A. Abdulsamad and S. R. Répás, “Application of fpga devices in network security: A survey,” *Electronics*, vol. 14, no. 19, 2025.

[112] E. Ustun, C. Deng, and Z. Zhang, “Accurate operation delay prediction for FPGA HLS using graph neural networks,” in *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD ’20)*, IEEE/ACM, 2020.

[113] R. Ifrim and D. Popescu, “Analyzing the capabilities of HLS and RTL tools in the design of an FPGA Montgomery Multiplier,” 2025.

[114] C. Lee, J. Lee, H. Jung, H. Lee, and H. Lee, “Hls-based hw/sw co-design and hybrid hls-rtl design for post-quantum cryptosystem,” *Journal of Semiconductor Technology and Science*, vol. 24, no. 3, pp. 191–198, 2024.

[115] M. Axinte, A. Stana, and V. Manta, “Embedded streaming hardware accelerators: Interconnect architectures and latency evaluation,” *Electronics*, vol. 14, no. 8, p. 1513, 2025.

[116] H. Kheddar, O. Azzouzi, M. Anane, M. C. Ghanem, and Y. Himeur, “Efficient fine-grained LuT-based optimization of AES MixColumns and InvMixColumns for FPGA implementation,” *Electronics*, vol. 14, no. 24, p. 4912, 2025.

[117] S.-H. Lin, J.-Y. Lee, C.-C. Chuang, N.-Y. Lee, P.-Y. Chen, and W.-L. Chin, “Hardware implementation of high-throughput s-box in aes for information security,” *IEEE Access*, vol. 11, pp. 59049–59058, 2023.

[118] O. S. Sonbul, M. Rashid, M. I. Masud, M. Aman, and A. Y. Jaffar, “Deeply pipelined ntt accelerator with ping-pong memory and lut-only barrett reduction for post-quantum cryptography,” *Electronics*, vol. 15, no. 3, 2026.

[119] L. Li, C. Sau, T. Fanni, J. Li, T. Viitanen, F. Christophe, F. Palumbo, L. Raffo, H. Huttunen, J. Takala, and S. S. Bhattacharyya, “An integrated hardware/software design methodology for signal processing systems,” *Journal of Systems Architecture*, 2019.

[120] A. Li, A. Ning, and D. Wentzlaff, “Duet: Creating harmony between processors and embedded fpgas,” in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[121] Arm Limited, *AMBA AXI and ACE Protocol Specification*, 2021. Issue H (includes AXI4 and AXI4-Lite).

[122] J. Chen, H. Unnibhavi, A. Koshiba, and P. Bhatotia, “vFPIO: A virtual I/O abstraction for FPGA-accelerated I/O devices,” in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, (Santa Clara, CA), pp. 1167–1184, USENIX Association, July 2024.

[123] T. Good and M. Benaissa, “Aes on fpga from the fastest to the smallest,” in *Cryptographic Hardware and Embedded Systems – CHES 2005*, vol. 3659 of *Lecture Notes in Computer Science*, pp. 427–440, Springer, 2005.

[124] F.-X. Standaert, G. Piret, G. Rouvroy, and J.-J. Quisquater, “Fpga implementations of the iceberg block cipher,” *International Conference on Information Technology: Coding and Computing (ITCC’05) - Volume II*, vol. 1, pp. 556–561 Vol. 1, 2005.

[125] D. B. de Oliveira, D. Casini, and T. Cucinotta, “Operating system noise in the linux kernel,” *IEEE Transactions on Computers*, 2022.

[126] P. Geier, M. Brändle, C. Faller, and S. Chakraborty, “Debugging fpga-accelerated real-time systems,” in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 350–363, 2020.

[127] M. Göbel, A. Elhossini, C. C. Chi, M. Álvarez-Mesa, and B. Juurlink, “A quantitative analysis of the memory architecture of fpga-socs,” in *Applied Reconfigurable Computing (ARC 2017)*, Lecture Notes in Computer Science, Springer, 2017.

[128] A. Rios-Navarro, R. Tapiador-Morales, A. Jimenez-Fernandez, M. Dominguez-Morales, C. Amaya, and A. Linares-Barranco, “Performance

evaluation over hw/sw co-design soc memory transfers for a cnn accelerator," 2018.

[129] R. Fernández Molanes, L. Costas, J. J. Rodríguez-Andina, and J. Fariña, "Comparative analysis of processor-fpga communication performance in low-cost fpsocs," *IEEE Transactions on Industrial Informatics*, 2021.

[130] W. Sullivan and D. Steinkamp, "Characterization of throughput on the axi dma bus for burst data transfer over ethernet," tech. rep., Office of Scientific and Technical Information (OSTI), U.S. Department of Energy, 2024.

[131] L. Abeni *et al.*, "Container-based real-time scheduling in the linux kernel," *ACM*, 2019.

[132] A. Panagopoulou, M. Paolino, and D. Raho, "Virtio-fpga: a virtualization solution for soc-attached fpgas," in *IEEE International Symposium on Embedded Systems for Real-time Multicore Systems (ESARS-ITEC)*, IEEE, 2023.

[133] G. Zhong, A. Aksu, *et al.*, "Design space exploration of fpga-based accelerators with multi-level parallelism," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2017.

[134] J. S. Malik, P. Palazzari, and A. Hemani, "Effort, resources, and abstraction vs performance in high-level synthesis: finding new answers to an old question," *ACM SIGARCH Computer Architecture News*, 2012.

[135] S. W. Nabi and W. Vanderbauwhede, "Fpga design space exploration for scientific hpc applications using a fast and accurate cost model based on roofline analysis," *Journal of Parallel and Distributed Computing*, vol. 133, pp. 407–419, 2019.

[136] E. Ferry *et al.*, "Increasing fpga accelerators memory bandwidth with a burst-friendly memory layout," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[137] X. Wei, Y. Liang, P. Zhang, C. H. Yu, and J. Cong, "Overcoming data transfer bottlenecks in fpga-based dnn accelerators via layer-conscious memory management," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, ACM, 2019.

[138] ITU-R, "Minimum requirements related to technical performance for imt-2020 radio interface(s)," Report M.2410-0, International Telecommunication Union, 2017.

[139] IEC, "Iec 61850-5: Communication networks and systems for power utility

automation – part 5: Communication requirements for functions and device models," 2013.

[140] C. Liang, T. Benz, A. Ottaviano, A. Garofalo, L. Benini, and D. Rossi, "Towards reliable systems: A scalable approach to axi4 transaction monitoring," *arXiv*, 2025.

ДОДАТКИ

Додаток А

Список опублікованих наукових праць за темою дисертаційного дослідження

Наукові праці, в яких опубліковані основні наукові результати дисертації:

Наукові праці у міжнародних періодичних наукових виданнях, проіндексованих у наукометричних базах даних Web of Science Core Collection та/або Scopus:

1. O. Krulikovskyi, **Y. Herman**, and O. Verenko, “Hardware/software communication architecture for a SoC-FPGA-based time-to-digital converter,” *Proceedings of SPIE – The International Society for Optical Engineering*, vol. 13813, pp. 214–217, 2025. (Scopus). doi: https://doi.org/10.1117/12.3092023. *(Внесок авторів: Круліковський О.: постановка задачі, розробка апаратної архітектури високошвидкісного тракту передачі даних на FPGA; Герман Ю.: розробка програмно-апаратного шлюзу (Linux-драйвер) та конфігурація механізмів доступу до пам’яті (DMA) через магістраль AXI4; Веренко О.: підготовка експериментального стенду та проведення вимірювань).*

Наукові праці у виданнях, включених до переліку наукових фахових видань України:

2. **Y. Herman**, H. Lastivka, and A. Samila, “Embedded operating systems in IoT edge computing,” *Security of Infocommunication Systems and Internet of Things*, vol. 2, no. 2, Art. no. 02001, 2024, doi: https://doi.org/10.31861/sisiot2024.2.02001. *(Внесок авторів: Герман Ю.: концептуалізація, розробка методології застосування вбудованих ОС, програмна реалізація та проведення експериментальних досліджень; Ластівка Г.: підготовка базового набору даних, валідація результатів та допомога в оформленні методології; Саміла А.: постановка задачі, загальне наукове керівництво, рецензування та редагування тексту).*
3. **Герман Ю.** та Саміла А., «Модульна EDGE-система збору та аналізу даних на базі Raspberry Pi та SoC FPGA», Вісник Хмельницького

національного університету. Серія: Технічні науки, Т. 357, № 5.2, 2025, С. 86–92, doi: https://doi.org/10.31891/307-5732-2025-357-69.
(Внесок авторів: Герман Ю.: розробка структурної схеми автономного телеметричного вузла, реалізація алгоритмів агрегації даних (стратегія Offline-First) та програмних інтерфейсів взаємодії; Саміла А.: загальне наукове керівництво, визначення критеріїв оцінки ефективності розробленої системи, наукове редагування).

4. **Герман Ю.** та Верига А., «Офлайн система обробки та візуалізації радіосигналу», Вимірювальна та обчислювальна техніка в технологічних процесах, Т. 84, № 4, 2025, С. 246–252, doi: https://doi.org/10.31891/2219-9365-2025-84-27. *(Внесок авторів: Герман Ю.: розробка архітектури бортового обчислювального модуля (SDR-сканера), інтеграція радіотракту з системним середовищем Linux, оптимізація конвеєра цифрової обробки сигналів, побудова клієнтського веб-інтерфейсу; Верига А.: розробка методу розподіленої візуалізації спектральних даних, інтеграція обчислювального модуля в систему, загальне наукове керівництво).*
5. **Y. Herman**, “FPGA Platforms and Their Use in Edge Computing,” *Security of Infocommunication Systems and Internet of Things*, vol. 3, no. 2, Art. no. 02008, 2025, doi: https://doi.org/10.31861/sisiot2025.2.02008.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

6. **Герман Ю. В.**, Круліковський О. В. та Верига А. Д., «Автономна система обробки та візуалізації радіосигналів», праці XIV Міжнар. наук.-практ. конф. «Проблеми інформатики та комп’ютерної техніки» (ПІКТ–2025), Чернівці, Україна, 2025, С. 188–190. https://drive.google.com/file/d/12FGgnfM6NA8HPo66h0TgVrfbPHAYDMFC/view *(Внесок авторів: Герман Ю.В.: розробка архітектури програмного шару для безперервної потокової обробки сигналів; Круліковський О.В.: наукове редагування; Верига А.Д.: апаратне налаштування приймального тракту).*
7. **Герман Ю. В.** та Круліковський О. В., «Розробка та використання програмного забезпечення для систем на кристалі типу SoC FPGA Cyclone V», XII Міжнар. наук.-практ. конф. «Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій», Запоріжжя, 2024, С. 27–29. https://old.zp.edu.ua/uploads/dept_

s&r/2024/conf/1.4/2024_Radiotekhnika_telekomunikatsiyi_ta_informatsiyni_tekhnolohiyi.pdf *(Внесок авторів: Герман Ю.В.: розробка архітектури програмно-апаратного шлюзу для керування периферією FPGA; Круліковський О.В.: верифікація апаратних ресурсів кристала, наукове консультування).*

8. Круліковський О. В., **Герман Ю. В.** та Веренко О. С., «Програмно-апаратна взаємодія систем на кристалі типу SoC FPGA Cyclone V», Physical and Technological Problems of Transmission, Processing and Storage of Information in Infocommunication Systems: Proceedings of the Xth International Scientific-Practical Conference, Чернівці, 2025, С. 45. https://drive.google.com/file/d/11rJfJNk4H-8W9_uyNE63X84MCMky_Ba-/view *(Внесок авторів: Круліковський О.В.: обґрунтування загального підходу до реалізації радіотехнічних пристроїв із програмним керуванням, оптимізація та конфігурація обраної архітектури; Герман Ю.В.: реалізація програмного доступу до регістрів вводу-виводу через інтерфейс MMIO; Веренко О.С.: тестування часових характеристик мостів HPS-FPGA).*
9. **Герман Ю. В.**, «Оцінка ефективності вбудованих систем на базі Linux та FPGA», XXIX міжнар. молодіж. форум «Радіоелектроніка та молодь у XXI столітті», Харків, 2025, Т. 3, С. 540–542. https://drive.google.com/file/d/1DGXvFT-OADOOOo5CSwDuFfZ52yhVaFAE/view

Наукові праці, які додатково відображають наукові результати дисертації:

10. **Y. Herman**, O. Krulikovskyi, S. Haliuk, and S. Subbotin, “Development of an embedded operating system based on the Linux kernel for SoC FPGA,” *CEUR Workshop Proceedings*, vol. 3702, pp. 376–388, 2024. https://www.scopus.com/pages/publications/85195909635 (Scopus) *(Внесок авторів: Герман Ю.: розробка методики формування спеціалізованого керуючого середовища Linux, проведення експериментального вимірювання системного джитера; Круліковський О.: конфігурація апаратної процесорної підсистеми (HPS); Галюк С.: статистичний аналіз результатів роботи планувальника задач; Субботін С.: методологічне керівництво, наукове редагування).*

Додаток Б

Відомості про апробацію результатів дисертації

1. Наукові семінари кафедри радіотехніки та інформаційної безпеки Чернівецького національного університету імені Юрія Федьковича; (очна участь).
2. Fourteenth International Scientific-Practical Conference «Проблеми інформатики та комп'ютерної техніки» (ПІКТ–2025) (м. Чернівці, 13–15 листопада 2025 р.), форма участі — очна участь.
3. XII Міжнародна науково-практична конференція «Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій» (м. Запоріжжя, 10–12 грудня 2024 р.), форма участі — дистанційна з публікацією тез.
4. Xth International Scientific-Practical Conference «Physical and technological problems of transmission, processing and storage of information in infocommunication systems» (м. Чернівці, 15–17 травня 2025 р.), форма участі — дистанційна з публікацією тез.
5. XXIX Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті» (м. Харків, 16–19 квітня 2025 р.), форма участі — дистанційна з публікацією тез.
6. Seventh International Workshop on Computer Modeling and Intelligent Systems (CMIS-2024) (м. Запоріжжя, 3 травня 2024 р.), форма участі — дистанційна з публікацією матеріалів.
7. Seventeenth International Conference on Correlation Optics (SPIE) (м. Чернівці, 8–12 вересня 2025 р.), форма участі — дистанційна з публікацією матеріалів.