Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем

(повна назва інституту/факультету)

Кафедра телекомунікацій

(повна назва кафедри)

«На правах рукопису»

УДК _____

До захисту допущено
Завідувач кафедри

_____ Сергій КРАВЧУК

(підпис) (Ім'я, прізвище)

"___"_____2020_р.

Магістерська дисертація

на здобуття освітнього ступеня «магістр»

Спеціальність 172 Телекомунікації та радіотехніка,

(код і назва)

За освітньо-професійною програмою Інженерія та програмування інфокомунікацій.

на тему: Дослідження структури організації програмного забезпечення SDR систем на базі SoC

Виконав (-ла): студент (-ка) _2_ курсу, групи ТЗ – з91_мп

(шифр групи)

Бондаренко Катерина Сергіївна

(прізвище, ім'я, по батькові) (підпис)

Науковий керівник к.т.н. доцент Явіся В. С.

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____

(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент_____ _____

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Київ – 2020 рік

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем

(повна назва)

Кафедра телекомунікацій

(повна назва)

Спеціальність 172 Телекомунікації та радіотехніка

(код і назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою Інженерія та програмування інфокомунікацій.

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Валерій ЯВІСЯ

(підпис) (Ім’я, прізвище)

« 20 » січня 2020 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Бондаренко Катерини Сергіївни

(прізвище, ім’я, по батькові)

1. Тема дисертації Дослідження структури організації програмного забезпечення SDR систем на базі SoC

науковий керівник дисертації к.т.н. доцент Явіся Валерій Сергійович,

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «04» листопада 2020 р. № 3218-с

2. Строк подання студентом дисертації _____

3. Об’єкт дослідження - програмно-визначувані радіосистеми (SDR) на базі системи на кристалі (SoC).

4. Предмет дослідження - ефективність Bare metal, RTOS (Real Time Operating Systems) та GPOS (General Purpose Operating Systems) а саме юнікс-подібних операційних систем для програмно-визначуваних радіосистем (SDR) на базі системи на кристалі (SoC).

5. Перелік завдань, які потрібно розробити:

Зібрати основні дані про програмно-визначуване радіо, а саме розкрити поняття SDR, історію його появи, особливості архітектури та програмного забезпечення. Розглянути підходи до реалізації SDR на SoC та основні складові програмного забезпечення SDR системи. Проаналізувати існуючі рішення для SDR SoC: Bare metal, RTOS та GPOS. Для кожного із існуючих рішень розглянути архітектурні

особливості, особливості розробки та провести оглядовий аналіз процесу розробки для таких рішень. За результатами аналізу обрати одне із висвітлених рішень і провести детальний огляд існуючих інструментів та підходів для такого рішення. Провести практичне дослідження обраного рішення з використанням розглянутого підходу.

6. Орієнтовний перелік ілюстративного матеріалу _____

_____

_____

7. Орієнтовний перелік публікацій _____

_____

_____

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Старший науковий співробітник Кайденко М.М.		
2	Старший науковий співробітник Кайденко М.М.		
3	Старший науковий співробітник Кайденко М.М.		
4	Кандидат технічних наук доцент Явіся В. С		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської	Примітка

		дисертації	
1	Збір матеріалів та основних відомостей про SDR		
2	Збір матеріалів та аналіз Bare metal рішення		
3	Збір матеріалів та аналіз RTOS рішення		
4	Збір матеріалів та аналіз GPOS рішення		
5	Порівняльний аналіз існуючих рішень та вибір рішення на основі проведеного аналізу		
6	Збір матеріалів та аналіз існуючих підходів для обраного рішення		
7	Практичне дослідження обраного рішення SDR на SoC		

Студент _____ Бондаренко К.С.

(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Явіся В. С.

(підпис) (ініціали, прізвище)

РЕФЕРАТ

Мета - спрощення вибору рішення для SDR систем на базі SoC у SDR системах у контексті ефективності обраного рішення та часу розробки програмного забезпечення для таких систем.

Практичні задачі, на вирішення яких спрямовано проект:

- детальний аналіз ефективності роботи RTOS (Real Time Operating Systems), Bare metal та юнікс-подібних операційних систем у SDR + SoC;
- оцінка орієнтовного часу розробки програмного забезпечення для існуючих рішень SDR систем на базі SoC;
- Практичне дослідження обраної структури організації програмного забезпечення SDR систем на базі SoC.

Значимість проекту для розв'язання економічних і соціальних проблем: підвищення ефективності розробки та використання програмного забезпечення для SDR систем на базі SoC.

Об'єктом дослідження є програмно-визначувані радіосистеми (SDR) на базі системи на кристалі (SoC).

Предметом дослідження є ефективність RTOS (Real Time Operating Systems), Bare metal та юнікс-подібних операційних систем для програмно-визначуваних радіосистем (SDR) на базі системи на кристалі (SoC).

Проблема, що вирішується - вибір програмного забезпечення (операційної системи або відсутність операційної системи) для забезпечення оптимальної роботи SDR систем на базі SoC.

Ключові слова: *SoC, SDR, Bare metal, OS, RTOS, GPOS, libiio.*

ABSTRACT

The **aim** of this work is to simplify the choice of solution for SoC-based SDR systems in SDR systems in the context of the efficiency of the chosen solution and the software development time for such systems.

Practical tasks to be solved by the project:

- detailed analysis of the efficiency of RTOS (Real Time Operating Systems), Bare metal and Unix-like operating systems in SDR + SoC;
- estimation of time spent for software development for existing SoC-based SDR solutions;
- study of the selected structure of the organization of software SDR systems based on SoC.

Significance of the project for solving economic and social problems: increase the efficiency of development and use of software for SDR systems based on SoC.

The **object** of the study is SDR systems based on a SoC.

The **subject** of the study is the efficiency of RTOS, Bare metal and Unix-like operating systems for SoC based SDR.

The **problem to be solved** is the choice of software (OS or no OS) to ensure optimal operation of SDR systems based on SoC.

Keywords: *SoC, SDR, Bare metal, OS, RTOS, GPOS, libiio*.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

API – Application Programming Interface;

DMA – Direct memory access;

GPOS – операційна система широкого призначення;

FPGA - Field Programmable Gate Array;

HIL - Hardware-in-the-Loop Simulation;

HPS - Hard Processor System;

RTOS – Real Time Operating System;

SDR - Software defined Radio;

SoC – System-on-a-Chip;

АЗ – апаратне забезпечення;

ПЗ – програмне забезпечення;

ОС – операційна система.

ВСТУП

Розробка програмно-визначуваних радіосистем (SDR - Software-Defined Radio) на базі технологій програмовані системи на кристалі (SoC) це складний та трудоємкий процес, який потребує використання декількох середовищ для прототипування, розробки та тестування таких систем. Слід зазначити, що SDR системи в тому числі і на основі SoC, відносяться до класу вбудованих систем (embedded system) і для їх проектування використовуються методи та засоби, які в загальному вигляді відрізняються від тих, що застосовуються для проектування комп'ютерних систем загального призначення. Перш за все, дана робота спрямована на аналіз та оцінку продуктивності роботи існуючих рішень.

У роботі проводиться порівняльний аналіз використання Bare metal, RTOS (Real Time Operating Systems), та юнікс-подібних операційних систем для програмно-визначуваних радіосистем (SDR) на базі системи на кристалі (SoC). Проводиться практичне дослідження обраної структури організації програмного забезпечення SDR систем на базі SoC. Також проводиться орієнтовна оцінка вартості використання існуючих рішень.

Детальний аналіз, практичне дослідження та орієнтовна оцінка вартості використання існуючих рішень спрямовані на підвищення ефективності розробки та використання програмного забезпечення для SDR систем на базі SoC спрощуючи вибір між існуючими рішеннями згідно цілей та бюджету проекту заздалегідь до початку етапу впровадження та імплементації проекту.

ОСНОВНІ ВІДОМОСТІ ПРО SDR

1.1 Поняття програмно-визначуваного радіо

Концепція програмного радіо була придумана Joseph Mitola для доступу до його оригінальної роботи на початку 90-х років [1]. Не зважаючи на те, що реалізація всієї радіосистеми повністю за допомогою програмного забезпечення на даний час все ще є недосяжною, в останні роки з`явилась велика кількість архітектурних рішень, що включають в себе все більшу ступінь програмованості. При цьому досі немає єдиного узгодженого рішення про апаратну архітектуру, вбудовану в мобільний термінал з засобами SDR. Використовуються різноманітні технології, які часто змішуються і тому іноді термін «конфігурований» більше підходить до них, ніж програмований. Вивчення архітектурних рішень для нових систем SDR має визначальне значення в силу необхідності визначення рівня абстрагованості апаратних засобів систем SDR: рівня абстрагованості радіообладнання. В 2010 році були опубліковані дві важливі роботи [2, 3]. В роботі [2] надано повний огляд проблем SDR, які пов`язані з архітектурою програмного забезпечення, вимогами до обчислювальних ресурсів, безпекою, сертифікацією та бізнес-моделями для систем SDR. В роботі [3] було приведено результати порівняльного дослідження різних важливих багатоядерних архітектурних рішень SDR та програмних рішень.

SDR форум, який був перейменований в Wireless Innovation Forum запропонував виділити п`ять рівнів радіосистем. Рівень 0 відповідає апаратним радіосистемам, рівень 1 відповідає програмно-керованим радіосистемам (в програмному забезпеченні реалізовані тільки функції управління), а рівень 2 відповідає програмно-визначуваним радіосистемам. Рівні 3 та 4 на сьогодні нереалізовані. Співпрацюючи з Інститутом інженерів електротехніки і електроніки (IEEE - Institute of

Electrical and Electronics Engineers) (робоча група P1900.1), SDR форум працював над створенням визначення SDR, яке забезпечує узгодженість та чіткий огляд технології та пов'язаних з цим переваг. SDR було визначено, як "Радіо, в якому деякі або всі функції фізичного рівня визначені програмно" [4].

В більш широкому розумінні **програмно-визначувана радіосистема** – це система, яка використовує набір апаратних і програмних технологій, при цьому деякі, чи усі функції роботи радіосистеми на фізичному рівні реалізуються за допомогою програмного забезпечення. Програмне забезпечення може модифікуватись, або може бути вбудованим, яке працює на програмованих пристроях обробки сигналів. Ці пристрої можуть включати в себе програмовані користувачем логічні матриці (FPGA- Field Programmable Gate Array), цифрові сигнальні процесори (DSP - Digital signal processor), процесори загального призначення (GPP - General purpose processor), програмовані системи на кристалі (SoC - System-on-chip) та інші спеціалізовані програмовані процесори. Використання цих технологій при побудові SDR дозволяє додавати нові безпроводові функції та можливості в існуючі радіосистеми без зміни апаратної платформи.

Завдяки передовим технологіям SDR надають можливість покращувати використання спектру, відкривати нові ринки телекомунікацій та розширювати перелік доступних послуг. SDR може виступати ключовою технологією для багатьох інших реконфігурованих радіосистем, які зазвичай розробляються з використанням передових технологій бездротового зв'язку. Хоча SDR не вимагає впровадження жодного з цих типів радіотехнологій, технології SDR можуть забезпечити цим типам радіозасобів необхідну їм гнучкість для досягнення повного потенціалу, переваги якого можуть допомогти зменшити витрати на розробку та підвищити ефективність системи: адаптивне радіо, конгнітивне радіо та інтелектуальне радіо [4]. Ці типи не обов'язково визначають окремий елемент обладнання, але можуть натомість включати компоненти, що розподілені по всій мережі.

Програмно-визначувані радіосистеми поєднують в собі передові технології для забезпечення гнучкості при створенні радіосистем. До цих технологій відносяться широкосмугові технології, радіочастотні компоненти, програмовані радіочастотні компоненти, технології цифрової

обробки сигналів, аналогово-цифрові та цифро-аналогові перетворювачі, синтезатори частоти, технології побудови системи на кристалі, технології обробки інформації.

Внаслідок гнучкості SDR технологій виникає ряд проблем при конфігуруванні та управлінні системою: збірка в єдине ціле обладнання та програмних компонент різних платформ в працюючу радіосистему; управління радіосистемою під час її роботи; робота радіосистеми у умовах фізичних та нормативних обмежень регулюючих органів різних країн; програмна самоконфігурація радіокомпонентів та адаптація; визначення загальних інтерфейсів для додатків та програмного забезпечення управління; використання бібліотек компонентів та радіоконфігурацій для різних апаратних та програмних платформ (переносимість).

Завдяки передовим технологіям SDR надають можливість покращувати використання спектру, відкривати нові ринки телекомунікацій та розширювати перелік доступних послуг, зокрема це відноситься до нових технологій побудови Cognitive Radio System.

Адаптивне радіо - це радіо, в якому системи зв’язку мають засоби для моніторингу власних характеристик та зміни своїх робочих параметрів для підвищення цих характеристик. Використання технологій SDR в адаптивній радіосистемі дозволяє отримати більший ступінь свободи при адаптації, а отже, більш високий рівень продуктивності та кращу якість обслуговування в каналі зв’язку.

Когнітивне радіо - це радіо, в якому системи зв'язку знають про свій внутрішній стан та оточуюче середовище, наприклад власне місцезнаходження та використання радіочастотного спектра в цьому місці. Вони можуть приймати рішення про свою поведінку в радіоефірі, зіставляючи цю інформацію відповідно до попередньо визначених цілей. Когнітивне радіо також визначається багатьма для використання програмно-визначуваного радіо, адаптивного радіо та інших технологій для автоматичного налаштування його поведінки чи операцій для досягнення бажаних цілей. Використання цих елементів має вирішальне значення для надання кінцевим споживачам можливості оптимально використовувати наявний частотний ресурс та бездротові мережі із загальним набором радіоапаратних засобів. Як зазначалося раніше, це знизить вартість для кінцевого споживача, одночасно дозволяючи йому

або їй спілкуватися з ким завгодно, коли завгодно і будь-яким доступним способом.

Інтелектуальне радіо - це конгнітивне радіо, яке здатне до машинного навчання. Це дозволяє когнітивному радіо вдосконалити способи адаптації до змін оточуючого середовища та продуктивності системи для кращого задоволення потреб кінцевого користувача.

Суть технології Software Defined Radio полягає в тому, що базові параметри приймально-передаваючого пристрою визначаються саме програмним забезпеченням, а не апаратною конфігурацією, як ми звикли бачити в класичних конструкціях. Таким чином, це словосполучення можна перевести, наприклад, як "радіо, яке визначається програмним забезпеченням", але можна піти далі і скоротити до двох слів: "програмне радіо", але з цим варіантом слід бути обережним і в контексті намагатися підкреслювати, що незважаючи на згадку епітета "програмний", ми маємо справу саме з апаратним забезпеченням, параметри якого визначаються програмно.

При використанні SDR персональний комп'ютер стає ядром любительської радіостанції, завдяки чому практично весь обсяг робіт із обробки сигналу перекладається на програмне забезпечення, яке запускається на персональному комп'ютері або керує роботою деяких конкретних спеціалізованих мікропроцесорних пристроїв, призначених для обробки сигналу. Мета такого підходу — створити систему, яка може приймати і передавати практично будь-які радіосигнали за допомогою програмного забезпечення, що є гнучким і адаптивним.

В даний час SDR широко застосовуються у військовому і стільниковому зв'язку, де в режимі реального часу потрібна підтримка різноманітних радіопротоколів, що змінюються.

У режимі прийому SDR може забезпечити вищу ефективність, ніж при використанні традиційних аналогових методів, оскільки при цифровій обробці сигналів їх фільтрація близька до ідеальної. Крім того, за допомогою програмних алгоритмів можуть бути реалізовані такі функції, які дуже складно отримати при аналоговій обробці.

Взагалі, ідея SDR зовсім недавно виглядала б відверто фантастично. Уявіть собі: ви набираєте певний код на конфігураціонній панелі, і

пристрій з прийомо-передавача Bluetooth перетворюється в ZigBee-систему. Зауважу, що мова йде не тільки про радіочастотні параметри системи - вид модуляції, потужність високочастотного сигналу, параметрів приймача (чутливість, вибірковість, придушення гармонік), що, взагалі, можна зробити за допомогою комутації відповідних вузлів приладу, але і про протокольну частину. Тобто в наведеному мною прикладі ми спостерігаємо повне переродження апаратури, яка раніше могла виконувати тільки одну жорстко задану функцію.

Наведений приклад трохи перебільшений, оскільки крім таких міжстандартних бездротових систем на даний момент активно розвиваються програмно реконфігуруємі пристрої SDR. Одним з пріоритетних напрямків розвитку систем SDR, безумовно, є створення багато-протокольних радіосистем. Тут активно працюють військові (зі зрозумілих причин), але не тільки вони. Цей напрямок має високу ступінь комерціалізації, завдяки чому він отримав активний розвиток.

Спрощена архітектура типового Software Defined Radio (рисунок 1.1) містить блоки аналого-цифрового, цифро-аналогового перетворення, антену, ланцюги обробки цифрових сигналів та інші допоміжні блоки. Як правило, крім цифрового сигнального процесора, радіо з архітектурою SDR містить мікроконтролер. Розглянемо докладніше кожен з блоків для випадку приймача з архітектурою SDR. Одним з найбільш важливих вузлів такого SDR-пристрою є аналого-цифровий перетворювач. В реальності АЦП безпосередньо підключається до антени, тобто перетворює безперервний в часі сигнал в дискретну двійково-кодовану форму. Очевидно, що характеристики АЦП будуть значною мірою визначати і параметри пристрою в цілому. Тому слід звернути увагу на такі важливі параметри аналого-цифрових перетворювачів, як відношення «сигнал - шум», розрішення (число біт за вибірку), динамічний діапазон при відсутності паразитних складових, і, нарешті, параметр, вкрай важливий для автономних систем - розсіювана потужність і наявність режимів енергозбереження.

Рисунок 1.1 - Спрощена архітектура типового Software Defined Radio.

Інший не менш важливий компонент архітектури SDR - цифровий сигнальний процесор. Саме він забезпечує гнучкість системи і використовується головним чином для проведення розрахунків, необхідних для виконання алгоритму обробки сигналу. Традиційно ЦСП використовувалися для виконання функцій перед-модуляційної обробки і обробки сигналу після детектування (в приймачах). Однак останнім часом вони використовуються в основному в трансиверах з розширеними комунікаційними можливостями для детектування, корекції, демодуляції, синтезу частот і фільтрації каналів. Перетворення Фур'є - одна з найбільш поширених функцій, які виконуються ЦСП мало не в кожному комунікаційному пристрої. Широко також використовується швидке перетворення Фур'є (ШПФ).

Однак ідеальний пристрій SDR в даний час не може бути реалізовано з причин надмірно високої вартості такої системи і досить відчутних обмежень технологій, що використовуються в основі систем SDR (наприклад, наявні на даний час цифрові сигнальні процесори мають недостатню швидкодію для реалізації одночасно всіх функцій радіо). Тому на даний момент існує кілька реалізацій бездротових платформ, які в більшій чи меншій мірі відповідають факторам архітектури SDR.

Історично однією з перших таких реалізацій була система SPEAKeasy, яка стала успішним проектом по використанню комунікаційних систем на базі технології Software Defined Radio у військовому обладнанні США. Система випробовувалася в США в 1970 році. SPEAKeasy дозволяла цифровій апаратній платформі загального призначення здійснювати зв'язок з іншими системами в широких діапазонах частот, видів модуляції, методів кодування даних і варіювання інших параметрів. Комерційні реалізації систем SDR підрозділяються на платформи для реалізації базових станцій і на пристрої і термінали споживчої категорії. Як правило, продуктивність останніх становить 1 млн операцій в секунду (зважаючи на постійне вдосконалення та здешевлення

ЦСП це значення постійно зростає) і вони в першу чергу орієнтовані на роботу від автономних джерел живлення (батареї, акумулятори).

1.2 Історія появи SDR

Відомо, що Джозеф Мітола у 1991-ому переосмислив термін програмного радіо для плану базової станції GSM, в якому реалізував прототип цифрового приймача базової смуги, що мав в собі масив процесорів, які виконували адаптивну фільтрацію для нівелювання ефектів, котрих зазнавав сигнал після подолання завад, та демодуляцію широкосмугових сигналів [5]. Одні з перших згадок подібного до SDR терміну були зроблені компанією “E-Systems” (з 1995-го “Raytheon Intelligence and Information Systems”, а з 2013-го — “Raytheon Intelligence, Information and Services”). Перед початком 90-их р. ПС США купили ідею у компанії. Тому існувала заборона Мітолі у своїй статті публікувати технічні деталі його прототипу, над котрим він працював у 90-91 роках, бо тоді це вважалося “конкурентною перевагою США”.

Отже, Мітола опублікував доповідь на тему: «Програмне радіо: опитування, критичний аналіз та майбутні напрямки» на національній конференції телекомунікаційних систем (IEEE) у 1992 р., в якій описав принципи архітектури без деталей щодо впровадження. Одразу ж за ним виступав Боб Прилл, який розпочав свою доповідь зі слів: “Джо абсолютно правий у теорії програмного радіо, і ми будуємо її”. Попри це, через два місяці було заперечно використання цього терміну. У відповідь віце-прем’єр-міністру маркетингу Алану Джексонну на запитання про наявність передавачів в пристроях, чи в лабораторії, віце-президент Гарленду сказав: “Ні, звичайно, ні — наш пристрій це програмний радіоприймач”. Тому справедливо визнати, що для того аби називати об’єкт статті “радіо” не вистачає трансмітеру. Через те публікація була зупинена, згідно з рішенням корпоративного керівництва.

У цілому, Мітола передбачав ідеальне радіо, фізичними компонентами якого на приймальній стороні були б лише антена та АЦП, а на

приймальній — ЦАП і передавальна антена. Усіма іншими функціями можна було б забезпечитись через процесор і керувати ними, перепрограмовуючи останній. [6] За цю утопічну концепцію радіотехнічна спільнота і вважає Мітолу “хрещеним батьком” SDR. А ще він відкрив дану концепцію технології для широкого використання в публічних інтересах.

Стівен Блюст був тим, хто ввів термін програмно-визначеного радіо у своєму патенті 1995 р. Хоча Мітола заперечував цей термін, згодом він погодився на нього із прагматичних міркувань. Зі роботою Стівена пов’язана одна із перших програм, що намагалась втілювати концепцію Мітоли.

1.3 Особливості архітектури SDR

Архітектура програмного забезпечення SDR зазвичай базується на архітектурі комунікаційного програмного забезпечення (SCA - Software Communications Architecture) [7], яка була розроблена в рамках програми Department of Defense Joint Tactical Radio Systems (JTRS) [8]. SCA була призвана забезпечити відкриту архітектуру, яка дозволяє розробникам радіозасобів поєднувати апаратні та програмні компоненти в одній системі. По суті SCA в деякому сенсі є «операційною системою», яка визначає взаємодію програмних компонентів SDR. Однак SCA рухається до SDR від програмного забезпечення, а не від реалізації радіосистеми та перспектив її життєвого циклу. Хоча підхід SCA забезпечує гнучкість для програмних компонент, однак перехід від програмних компонент до робочої радіосистеми не є однозначним. Вирішення цієї неоднозначності є дуже важливим при створенні конгнітивних радіосистем, в роботі [9] була запропонована модель архітектури програмно визначуваної радіосистеми для розробки обладнання.

Представлена архітектура, що схематично зображена на рисунку 1.2, покликана вирішити проблеми неоднозначності для розробників сучасних радіосистем, зокрема систем когнітивного радіо.

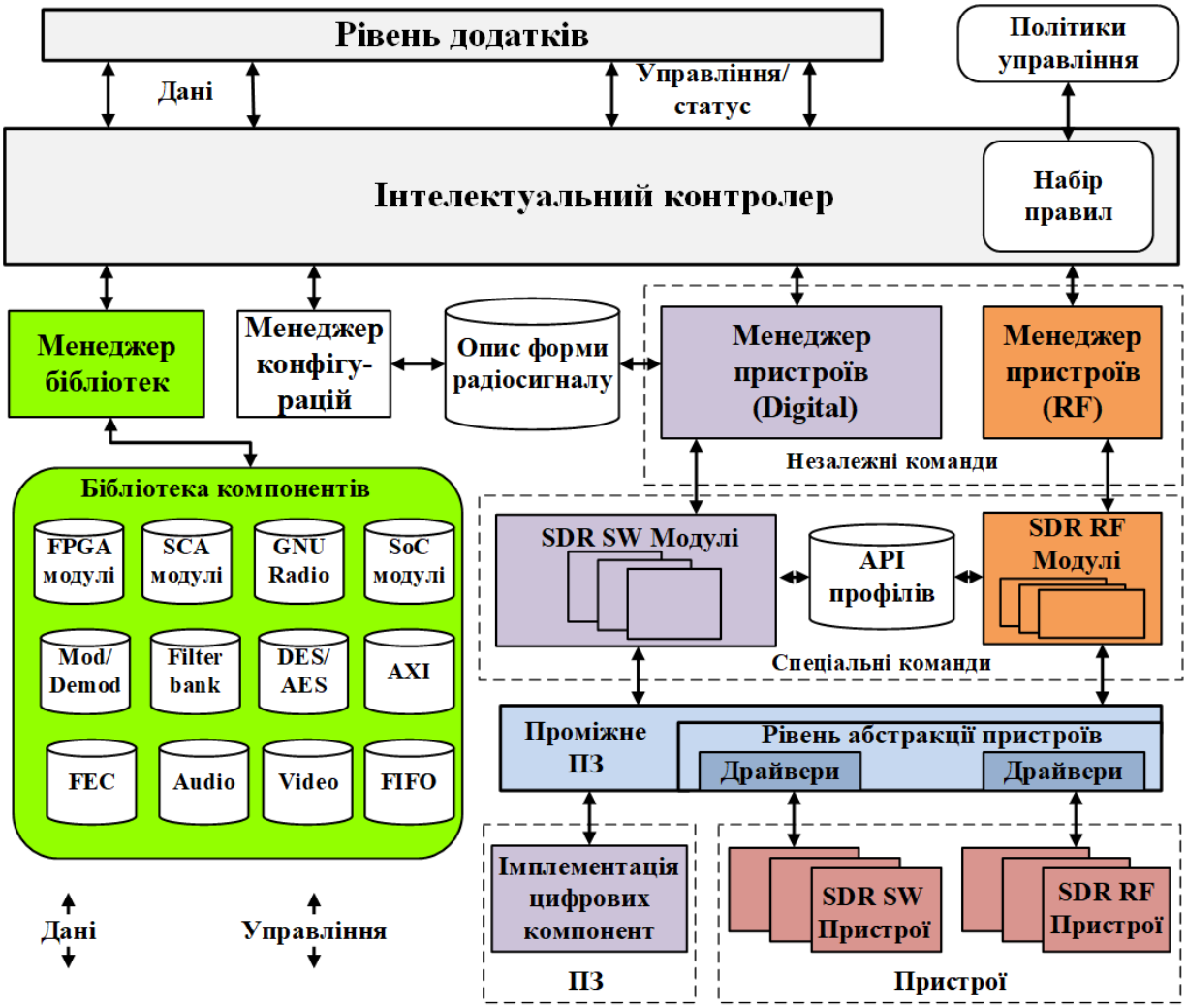

Рисунок 1.2 - Модель архітектури програмно-визначуваної радіосистеми для розробки радіотелекомунікаційного обладнання конгнітивної радіосистеми.

1.3.1 Модель архітектури SDR

Архітектура, яка була описана в [10], побудована на основі SCA специфікації і була використана для виявлення необхідних компонентів при реалізації реальної когнітивної радіосистеми, описаної [11-13]. Одним із основних компонентів, який використовується в цій моделі, є менеджер бібліотек для забезпечення організованого адміністрування компонент для безлічі файлів, що необхідні для програми SCA. Модель також демонструє використання класифікації компонентів, зокрема відповідних програмних інтерфейсів додатків (API - Application Programming Interface). API описує компонент для проекту таким чином, щоб дозволити компоненту інтерфейс з загальною архітектурою. Це покращує самосвідомість системи, яка в свою чергу покращує когнітивні здібності системи, дозволяючи радіосистемі визначати, які послуги надає доступний компонент і як отримати доступ до них просто знаючи тип класифікації.

1.3.2 Функції інтелектуального контролера

В архітектурі використовується Інтелектуальний контролер для зв’язку з рівнем додатків. До його складу входить також менеджер, що визначає набір правил для визначених політик управління. Політики та правила генеруються для визначеного призначення радіозасобу в межах доступних для його реалізації компонентів. Інтелектуальний контролер визначає, які функції доступні і які компоненти активні, або які потрібно активізувати. Додатково Інтелектуальний контролер повинен містити модуль перевірки рівня доступу до додатків для забезпечення захисту від несанкціонованого доступу.

1.3.3 Менеджери пристроїв

В архітектурі передбачено два менеджера пристроїв окремо для управління цифровими пристроями та радіочастотними пристроями. Це дозволяє забезпечити незалежне керування пристроями та покращити переносимість з платформи на платформу. Менеджери пристроїв створюють оперативну радіосистему інформація про яку знаходиться в описі програмного додатку (SAD - software application description). Динамічні радіосистеми можуть містити кілька SAD, які дозволять системі бути сумісними з різними стандартами та різними профілями передачі для одного стандарту. Після аналізу SAD цифрові пристрої та радіочастотні пристрої налаштовуються за допомогою менеджера пристроїв у відповідності до опису. Інформація для налаштування пристроїв (профілі) знаходиться в бібліотеці компонентів. Ця бібліотека зберігає зареєстровані компоненти, що доступні для радіосистеми, і керується менеджером бібліотек, який відповідає за внесення компонентів до бібліотеки, їх вилучення та підключення. Бібліотека компонентів використовується не тільки для забезпечення роботи радіосистеми і для покращення організаційного структура програмного забезпечення радіо, а й для поліпшення її автономності. У випадку конгнітивного радіо під керуванням Інтелектуального контролера визначається найбільш оптимальний варіант для побудови радіосистеми у відповідності до поставленої задачі. При створенні реальної радіосистеми, яка повинна адаптуватись як конгнітивна радіосистема, чи використовувати адаптацію до зміни умов роботи в умовах наявності нестаціонарних завад [14] інтелектуальний контролер у відповідності до згенерованих правил переносить їх набір до контролеру (процесору) управління системи разом з визначеними компонентами.

Логічні пристрої контролюють свої апаратні пристрої через відповідний інтерфейс. Використовуючи класифікацію API, логічний пристрій має лише знати тип пристрою, надаючи доступ до відповідної базової функції. Це дозволяє логічному пристрою, що представляє наприклад підсилювач, керувати ним до тих пір, доки він

використовується драйвером пристрою. Для конкретного компонента логічний пристрій - це простий шаблон, який не буде потрібним для створення системи в режимі реального часу.

1.4 Особливості побудови програмного забезпечення SDR

Структура програмного забезпечення для будь-якої довільної системи визначається призначенням системи та функціями, які вона виконує. Програмне забезпечення необхідне для роботи SDR системи повинне відповідати структурно-функціональним принципам побудови цієї системи. Програмне забезпечення для SDR може бути побудоване як з використанням операційної системи так і без неї (Bare Metal Application). SDR системи відносяться до класу вбудованих систем, тобто спеціалізованих систем управління, які розробляються таким чином, що такі системи працюють будучи вбудованими безпосередньо до пристрою, яким вони управляють. В якості операційної системи найчастіше використовується операційна система на базі Embedded Linux, оскільки для створенні сучасних та перспективних SDR систем найбільш передовою технологією є технологія систем на кристалі (System-on-a-chip — SoC). Принципова особливість SoC - це наявність програмованих блоків. Тому SoC - не просто інтегральна схема, а комплекс, до складу якого входять як апаратна частина (власне FPGA, HPS (Hard Processor System) і периферійні пристрої), так і програмна – вбудовується програмне забезпечення. Отже, при проектуванні SoC необхідно виконати операції по спільній верифікації та налагодженні взаємодії програмної і апаратної частин [15,16].

1.5 Реалізація SDR на SoC

У свій час мобільні пристрої перевернули ринок техніки, програмного забезпечення, ігор та інших сфер, пов'язаних з ними. Розібравши більшість з них сьогодні ми знайдемо інтегральну схему, у котру інтегровано більшість компонентів пристрою. Це і є прикладом системи на одному чипі (SoC).

На одній платі можна реалізувати увесь електронно-обчислювальний пристрій або більшу його частину, тому їх використання зручне для вбудованих систем та концепції "Інтернету речей", де вже існуючий пристрій треба доповнити з, часто, мінімальними змінами розміру. Окрім очевидних частин на кшталт CPU, GPU, RAM, Wi-fi, набору портів він повинен оперувати цифровими, аналоговими та змішаними сигналами, виконувати функції обробки частоти, бо інакше це лише "Прикладний процесор". Такі системи, здебільшого, виготовляють з використанням технологій метало-оксидного-напівпровідника (MOS).

Наслідком відмінності від дискретних схем (наприклад від материнських плат комп'ютера) полягає в покращенні продуктивності та зменшені енергоспоживання, поступаючись можливістю для компонентів бути заміненими, а отже і модульністю системи, що зменшує ще один параметр — займану платою площу. При цьому така плата, може бути модулем більш складної системи, але на місцях з'єднань збільшуються витрати енергії, площа пристрою та і продуктивність залежатиме від узгодженості модулів між собою. Такий варіант реалізації схеми підтримує ідею з концепції SDR, тому реалізація пристроїв на них привносить вище перераховані переваги.

1.6 Підходи до реалізації SDR на SoC

У розробці SoC систем існує два основні підходи [17]:

1. Створення пристрою без операційної системи (Bare Metal Application);
2. Використання операційної системи.

Перший підхід дає можливість отримання системи з максимальним рівнем продуктивності, але реалізація його вкрай складна, так як передбачає повну розробку всіх необхідних для SDR пристроїв, інтерфейсів, і т.п. на низькому рівні.

Другий підхід полегшує завдання, так як на операційну систему лягає завдання управління периферійними пристроями, реалізації високорівневих протоколів обміну, таких як IP, TCP, UDP і навіть протоколів доступу до пристрою, наприклад FTP, NTP, Telnet, SSH та ін. [18,19] Також, використання операційної системи дозволяє створювати універсальні пристрої, які набагато простіше інтегрувати в існуючі системи, ніж при використанні Bare Metal Application.

Кращим є використання операційної системи Linux для SoC пристроїв, так як вона має такі переваги [19]:

- широка апаратно-програмна підтримка розробника;

- величезна гнучкість в додатках готової системи;

- простота у використанні периферійних пристроїв;

- наявність розвиненої підтримки засобів розробки для вбудованих систем;

- на рівні ядра підтримка багатоядерних процесорів і процесорів ARM.

1.7 Складові програмного забезпечення SDR системи

Умовно програмне забезпечення (ПЗ) ділиться на системне, прикладне та інструментальне.

До **системного ПЗ** відноситься сама операційна система з графічною оболонкою (за необхідності її використання), стандартними драйверами та драйверами роботи з пам'яттю (підтримка специфіки роботи для SoC) і необхідні службові програми операційної системи.

Інструментальне ПЗ включає в себе необхідний набір драйверів для роботи з зовнішніми по відношенню до SoC пристроями та системами, а також драйверів внутрішнього обміну SoC. Ці драйвери не входять до складу операційної системи і повинні бути розроблені окремо. Підхід з використанням драйверів є найбільш доцільним, оскільки він є комплексним та дозволяє використовувати усі можливості операційної системи та FPGA при роботі SoC з периферійними пристроями. Цей підхід також дозволяє використовувати один драйвер для однотипних пристроїв, що істотно спрощує процес розробки системи.

Прикладне ПЗ використовується для виконання функцій управління, обробки даних, обміну даними та є оригінальним. Для створення прикладного ПЗ можуть використовуватись мови C, C++, Python.

Апаратна платформа SDR підключається до платформи SoC та взаємодіє з операційною системою через індустріальну підсистему вводу/виводу (IIO Subsystem). До складу ядра операційної для забезпечення взаємодії з SDR трансивером входять підсистеми GPIO, SPI, DMA та Clock.

Інтерфейс введення/виведення загального призначення (GPIO - General-purpose input/output) забезпечує зв'язок між компонентами комп'ютерної системи, (наприклад, ARM процесором) і периферійними пристроями**.** Контакти GPIO програмуються і як входи, і як виходи, і групуються в порти для обміну між FPGA та HPS.

Послідовний периферійний інтерфейс (SPI – Serial Peripheral Interface) є послідовним синхронним інтерфейсом передачі даних і забезпечую сполучення ARM процесора та SDR трансивера. На відміну від стандартного послідовного порту, SPI є синхронним інтерфейсом, в

якому кожна передача синхронізована з тактовим сигналом, що генерується ведучим пристроєм (ARM процесором).

Обмін даними між процесорним ядром та FPGA здійснюється в режимі прямого доступу до пам'яті (DMA – Direct Memory Access). DMA режим забезпечує обміну даними між пристроями без участі центрального процесора, в результаті чого швидкість передачі збільшується.

Згідно природі вбудованих систем, які часто виробляються тисячами або навіть мільйонами одиниць, зменшення витрат на АЗ на один долар може вилитися виробнику у значні заощадження. На сьогоднішній день, основною метою учасників ринку є зменшення вартості системи компонентів і зменшення складності конструкції. Саме ці переваги і надає використання SoC для реалізації SDR. Окрім того, на ринках, що стрімко розвиваються, таких як TVWS (TV White Space), де спритність в радіочастотному спектрі є необхідною, популярність рішень SoC SDR різко зросла. Настроюваний SoC RF також дуже привабливий для конструкцій 4G завдяки своїй здатності підтримувати глобальну частотну схему з однаковим дизайном АЗ.

1.8 Висновки

Програмно-визначувана радіосистема – це система, яка використовує набір апаратних і програмних технологій, при цьому деякі, чи усі функції роботи радіосистеми на фізичному рівні реалізуються за допомогою програмного забезпечення. Програмне забезпечення може модифікуватись, або може бути вбудованим, яке працює на програмованих пристроях обробки сигналів. Ці пристрої можуть включати в себе програмовані користувачем логічні матриці (FPGA- Field Programmable Gate Array), цифрові сигнальні процесори (DSP - Digital signal processor), процесори загального призначення (GPP - General purpose processor), програмовані системи на кристалі (SoC - System-on-chip) та інші спеціалізовані програмовані процесори. Завдяки передовим

технологіям, SDR надають можливість покращувати використання спектру, відкривати нові ринки телекомунікацій та розширювати перелік доступних послуг.

Суть технології Software Defined Radio полягає в тому, що базові параметри приймально-передаваючого пристрою визначаються саме програмним забезпеченням, а не апаратною конфігурацією.

АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Як було зазначено раніше, існує два підходи до реалізації SDR на SoC:

- Bare metal
- OS

1.9 Рішення на основі Bare metal

1.9.1 Історія розвитку підходу Bare metal

У початковий період розробки вбудованих систем програмне забезпечення було мінімальним. Зазвичай програмне забезпечення (ПЗ) розробляв той самий інженер, який розробляв і апаратне забезпечення (АЗ). Людина що займалася розробкою АЗ була ознайомлена з усіма нюансами поведінки обладнання, тому ПЗ дуже тісно взаємодіяло з електронікою і його розробка не вимагала надлишкових зусиль. [20]

По мірі вдосконалення і ускладнення систем, до розробки ПЗ стали залучатись спеціалісти з програмного забезпечення. З часом, зростання складності систем збільшувало і вимоги до таких спеціалістів. Таким чином, розробкою ПЗ почали займатися команди з декількох вузьконаправлених експертів.

Завдяки дедалі потужнішим мікропроцесорам та мікроконтролерам та більшим обсягам пам’яті, потреба в раціональній структурі програм спричинила прийняття операційних систем реального часу (RTOS), що

дозволило використовувати багатозадачність (багатопоточність або multi-threading). Драйвери для АЗ природно стали частиною RTOS.

З часом, ОС підхід набрав велику популярність витісняючи bare metal підхід. З приходом ОС на ринок ПЗ для АЗ, багато проектів мігрували з підходу «bare metal» до використання RTOS або GPOS (General Purpose Operating Systems, операційні системи широкої направленості).

Хоча більшість сучасних вбудованих програмних систем реалізована з використанням ОС, є кілька обставин при яких програмування bare metal - вигідне рішення. Сюди можна віднести ситуації, коли программа надзвичайно проста і реалізується на процесорі низького рівня. Також, програмування bare metal вигідне коли для роботи програми необхідне використання повної вичислювальної потужності системи і додаткові витрати на підтримку ОС є неприйнятними.

Хоча програмування bare metal на сьогоднішній день не є загальноприйнятим, інтерес до bare metal підходу починає зростати у індустріях в яких безпека використовується як основний критерій оцінки ПЗ. Ціль нового підходу до використання bare metal полягає в тому, щоб спостерігати та виявляти аномальну поведінку під час роботи системи, що дуже важливо для систем, у яких аномальна поведінка може спричинити несправність, пов'язану з безпекою людини. Створення безпечної та стабільної програми цілком можливе без контролю її поведінки на bare metal рівні, але у критичних системах найбільше занепокоєння викликає затримка та час, необхідний для виявлення аномальної поведінки, а ПЗ, що знаходиться на bare metal рівні, працює та реагує на такого роду помилки швидше ніж ПЗ що знаходиться на рівні ОС.

Вдалим прикладом для галузі, у якій програмування bare metal набирає популярності може бути автомобільна сфера. Потреба в програмуванні на bare metal в автомобільній галузі походить від допустимого рівня відмов (failure rates) який зазначається для деяких компонентів системи. Існують певні норми щодо того, як часто компоненти комплексних систем можуть відмовити. У багаторівневих системах, вірогідність відмови високорівневої системи являється добутком ймовірностей відмови усіх її компонентів, починаючи з самого низького рівню - вірогідності відмови АЗ на якому базується така система. Таким чином, використовуючи мінімальну кількість рівнів, а саме комбінацію АЗ + bare metal, можливо

досягти найменшої вірогідності відмови всієї системи що відіграє важливу роль для критичних систем. [20]

1.9.2 Особливості розробки bare metal

Програми, призначені для роботи без операційної системи (ОС), називаються програмами bare metal. У порівнянні з користувацькою програмою, якою керує ОС, програма bare metal може взаємодіяти безпосередньо з АЗ системи та працювати без ОС що зображене схематично на рисунку 2.1.

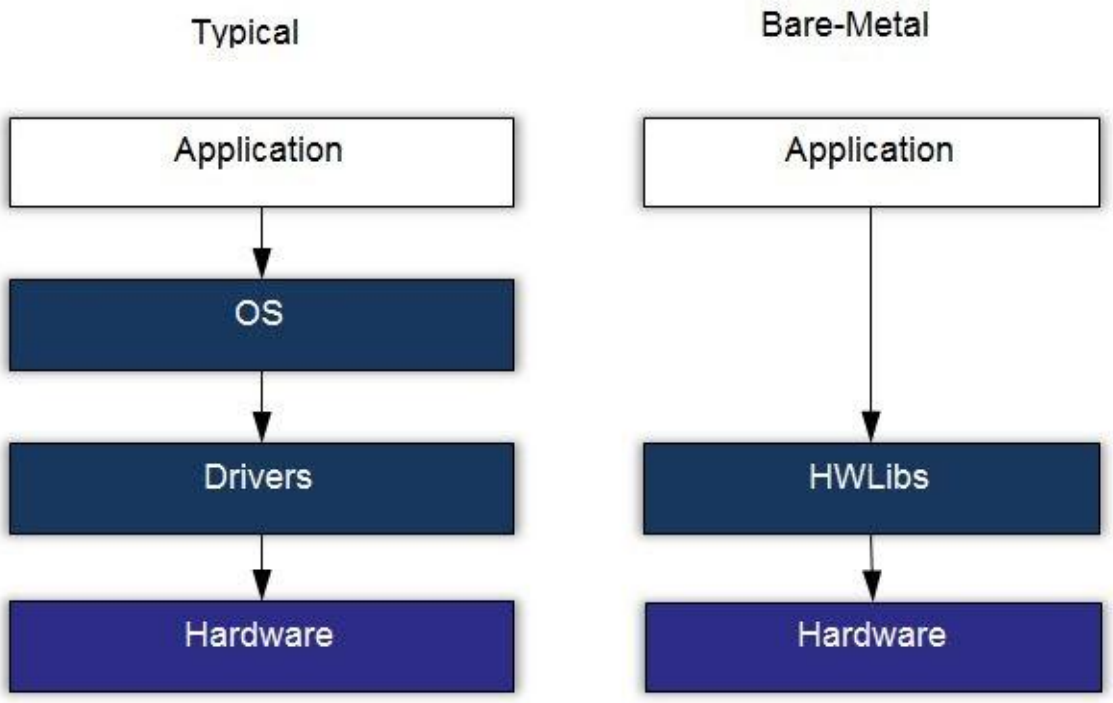

Рисунок 2.1 – Порівняльна схема типової архітектури та архітектури Bare metal.

Робота bare metal програми може бути настроєна одним із декількох способів. **Типова побудова** bare metal програми - програма запускається безпосередньо з попереднього завантажувача (Preloader) ознайомитися з якою можна на рисунку 2.2.

Рисунок 2.2 – Типова побудова bare metal програми.

Bootloader bare metal: програма bare metal запускається із завантажувача (Bootloader) ознайомитися з якою можна на рисунку 2.3.

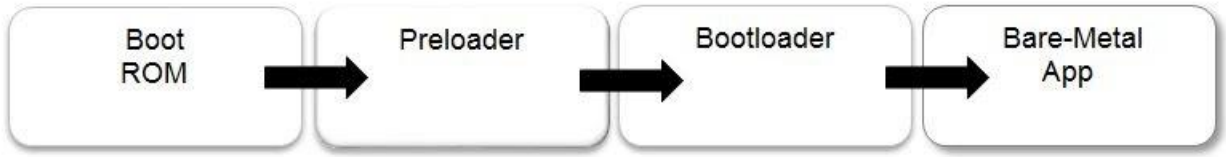

Рисунок 2.3 – Bootloader bare metal побудова bare metal програми.

Варто зазначити, що існує достатньо розповсюджена класифікація “RTOS bare metal” ознайомитися з якою можна на рисунку 2.4. У даній побудові, програма bare metal запускається з RTOS . [21]

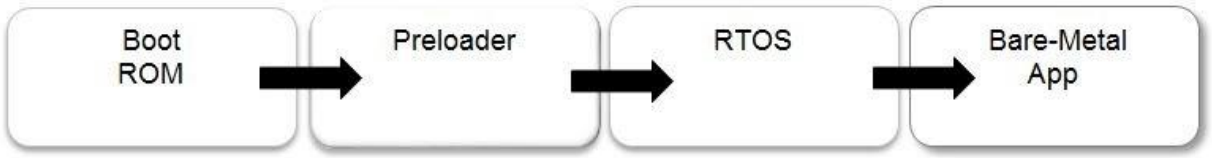

Рисунок 2.4 – RTOS bare metal побудова bare metal програми.

У даному випадку мається на увазі запуск ПЗ на прошарку попереднього завантажувача, не у прошарку ОС. Запуск ПЗ у прошарку ОС не відноситься до класифікації bare metal.

1.9.3 Оглядовий аналіз процесу розробки ПЗ для Bare metal

1.9.3.1 Збір інструментів необхідних для розробки ПЗ

Список необхідних компонентів для розробки ПЗ для Bare metal згідно [21] складається з:

- Altera SoC EDS
- USB-Blaster driver

Altera SoC EDS - Intel SoC FPGA Embedded Development Suite – набір інструментів для розробки Bare metal програм, в свою чергу включає в себе:

- ARM DS-5 Altera Edition
- ARM Compiler 5
- GNU Compiler Collection (GCC) Bare Metal Compiler
- HWLibs
- Mkpimage tool (required by BootROM)
- Mkimage tool (required by Preloader)
- SD card image tool
- Golden Hardware Reference Design (GHRD)

На момент створення даної роботи, вартість Altera SoC EDS ліцензії на одного програміста згідно [22] складає $2,995 USD на рік. У випадку

використання ОС, ціна необхідних компонентів для розробки ПЗ варіюється – починаючи від безпосередньої вартості ОС (як безкоштовно, так і $3-5 USD за одиницю згідно [23]). Для розробки на ОС, доступне велике розмаїття компонентів, і ціна на них починається від цілком безкоштовного стеку GNU технологій, і закінчується Matlab/Simulink, ціна на які може складати $2,150 для одного програміста (залежно від обраного пакету [24]). Таким чином, вартість необхідних компонентів для розробки на Bare metal можна вважати достатньо високою.

1.9.3.2 Підготовка до запуску тестового додатку

Для приблизної оцінки процесу розробки ПЗ для Bare metal буде використана примітивна програма з мінімальним функціоналом – «Hello world». Детальніше з даною програмою можна ознайомитись у Додатку А. Задача даної програми у виводі текстового блоку “Hello world!” у вихідний потік.

Для запуску тестової програми на Bare metal необхідно виконати декілька кроків.

Перш за все, необхідно створити файл для задання мапи пам’яті образу програми для лінкеру («**Scatter File**», формат файлу - .scat). Цей файл використовується лінкером ARM компілятору для визначення розміщення програми у пам’яті АЗ. Після створення файлу, його потрібно пов’язати з властивостями проекту. Після цього необхідно зібрати проект для того щоб впевнитись у його роботі. Опціонально, можна запустити у дебаг режимі – опис цього кроку буде опущений оскільки тестовий додаток достатньо примітивний. Необхідно також **скомпілювати** програму.

Наступним кроком буде завантаження та **запуск попереднього завантажувача**. Під час цього кроку, попередній завантажувач (Preloader) проводить настройку деяких компонентів плати, в тому числі і контролер пам’яті. Замість самостійного запуску попереднього завантажувача,

можливо також імпортувати вже існуючий, що цілком підходить для тестового додатку але не підходить для великого спеціалізованого проекту. Після цього, необхідно завантажити тестовий додаток до **обраного розділу у пам'яті**. Деякі розділи пам'яті потребують додаткової настройки для їх використання. Після виконання вищезазначених кроків, тестовий додаток може бути запущений на платі.

Варто зазначити, що тестовий додаток "Hello world" не вимагає створення додаткових пристроїв, інтерфейсів і т.п. оскільки виконує мінімальний можливий набір дій. Для запуску додатку була обрана найпростіша побудова bare metal програми - програма запускається безпосередньо з попереднього завантажувача (Preloader), і, таким чином, крок настройки завантажувача (Bootloader) був опущений. Також, виділення і настройка пам'яті для настільки примітивної програми може пройти автоматично, майже без людського втручання. Для кожної додаткової дії навіть у такому тестовому додатку необхідно буде виділяти та перерозподіляти пам'ять. Для кожного додаткового фунціоналу, за виключенням необхідного мінімального набору в який входить вивід даних у вихідний потік, необхідно буде створювати та настроювати відповідні пристрої на інтерфейси.

Навіть з використанням примітивного тестового додатку, можна помітити основну особливість підходу Bare metal – повний контроль над пам'яттю та над кожною дією. З Bare metal, програму виводу даних у вихідний потік неможливо запустити без попередньої настройки, а будь-які значні зміни у програмі неможливі без повторної перестройки багатьох компонентів. Ресурс АЗ буде цілком спрямований на виконання зазначених дій, а пам'ять виділена в точності як зазначено розробником.

1.10 Рішення на основі операційних систем

1.10.1 Використання ОС для SoC

Операційна система надає прикладне ПЗ з абстрактним представленням АЗ. Як частина абстракції вона забезпечує ініціалізацію та підтримку АЗ, управління завданнями (включаючи планування завдань), загальне управління ресурсами (пам’ять, периферичні пристрої тощо), міжпроцесорний зв’язок та контроль багатозадачності (блокування (locks), семафори (semaphores) тощо).

Через специфіку їх застосування, не всі SoC потребують наявність ОС. Натомість прикладне програмне забезпечення безпосередньо реалізує необхідну функціональність ОС. Причини відмови від ОС варіюються від проблем з продуктивністю до відсутності ОС для даної платформи SoC.

Аргументи для використання ОС на SoC, подібні до аргументів для запуску ОС в будь-якій системі. В першу чергу, використання існуючої ОС зменшує кількість часу, витраченого на реалізацію функціональних можливостей ОС. Крім того, використання ОС з добре розробленим та задокументованим API дозволяє програмістам підвищити свою продуктивність. ОС також забезпечує абстракції, які роблять використання апаратного забезпечення простішим та менш схильним до помилок для програміста. Ця абстракція також дозволяє повторно використовувати код програми, навіть якщо деталі базової апаратної архітектури змінюються. Більше того, використання раніше впровадженої та розгорнутої ОС зменшує кількість помилок у програмному забезпеченні. Нарешті, оскільки існує чітке розділення між функціональністю, схожою на ОС (наприклад, доступ до апаратного забезпечення, управління завданнями тощо) та кодом програми, існує більш чітке розділення проблем, що призводить до кращого структурованого коду з меншою кількістю помилок. [25]

1.10.2 Специфіка ОС для вбудованих систем

Можна зазначити декілька ключових характеристик операційних систем для SoC:

- Модульність
- Абстракціювання
- Переносимість

1.10.2.1 Модульність

Можливість реалізації традиційних програмних компонентів безпосередньо в АЗ відкриває нові можливості для ОС SoC. Зокрема, стає можливим впровадити функціональність ОС безпосередньо в апаратному забезпеченні. Наприклад, стек TCP / IP може бути впроваджений в АЗ для підвищення продуктивності, надійності та зменшення енергоспоживання.

Для підтримки таких модулів, ОС повинна підтримувати модульну структуру, яка дозволяє додавати, видаляти, заміняти або модифікувати частини функціоналу який традиційно вважається відповідальністю ОС. Наприклад, якщо стек TCP / IP потрібно перенести в апаратне забезпечення, ОС має надавати можливість видалити код TCP / IP з операційної системи та замінити його інтерфейсом для реалізації цього модулю на рівні АЗ. Також, якщо стек TCP / IP був попередньо забезпечений SoC, але він видаляється, щоб звільнити місце для іншого компонента, повинна бути можливість додати програмну реалізацію стека до ОС. В ОС без модульної структури взаємозалежності між різними

компонентами ОС такі модифікації надзвичайно складні та спричиняють багато помилок під час настройки роботи з SoC.

Модульність також потрібна для досягнення масштабованості існуючого функціоналу. Масштабована операційна система - це та, яку можна налаштувати на включення лише тих функціональних можливостей, які потрібні програмам, що її використовують. Оскільки SoC, що використовуються у вбудованих системах, як правило, обмежені ресурсами, масштабованість є особливо важливою характеристикою операційних систем SoC. ОС повинна забезпечувати полегшений інтерфейс (API), який забезпечує лише функціональність, необхідну додатку, і не більше. Надмірний інтерфейс створює непотрібні витрати, що негативно впливають на загальний розмір системи та її продуктивність. Крім того, надмірні API зменшують модульність, визначаючи велику кількість основних функціональних можливостей, які ОС повинна завжди забезпечувати.

У RTOS для SoC модульність є важливою особливістю, оскільки вона дозволяє масштабувати ОС. Більшість RTOS забезпечують масштабованість під час компіляції, дозволяючи включати або виключати набори функцій за потреби. Юнікс-подібні ОС, як правило, менш модульні і не масштабуються так само, як RTOS. Вони також мають більший розмір та вимагають більше ресурсів. Деякі гібридні системи поєднують невеликий RTOS з великою вбудованою ОС (наприклад, RT-Linux). Незважаючи на те, що у таких гібридних системах, частина RTOS невелика, вона тісно полягає на Linux реалізацію таких функціональних можливостей як ініціалізація, управління пам'яттю та драйвери.

1.10.2.2 Абстракціювання

Головне завдання ОС на користувацьких пристроях - абстракціювати АЗ на якому вона працює. У традиційній операційній системі акт абстрагування обладнання має три основні цілі щодо доступу до ресурсів: захист та безпека, мультиплексування, та спрощення використання АЗ. Віртуальна пам’ять і багатозадачність є головними прикладами перших двох цілей. Драйвери пристроїв є головними прикладами останньої.

Для ОС на SoC, ці три цілі не завжди бажані. Програмне забезпечення, що працює на SoC, використовує конкретне програмне забезпечення, яке є в мікросхемі. Тип абстракції, що виконується допоміжним програмним забезпеченням SoC, повинен підтримувати апаратний / програмний дизайн системи. Це означає, що операційна система не може абстрагувати все обладнання, натомість вона повинна абстрагувати лише загальне (не специфічне для програми) обладнання, інакше взаємодія з SoC буде неможлива.

Наприклад, у випадку з процесором, ОС відображає, що процесор підтримує управління адресним простором, паралельні дії (програми, що працюють паралельно), обмін інформацією між паралельними діями (міжпроцесовий зв'язок) та управління для серіалізації цієї віртуальної паралельності (планування, scheduling) . Оскільки ці характеристики не залежать від фактичної апаратної архітектури процесора, така ОС абстрагується від конкретних апаратних реалізацій. Наприклад, ОС не надає користувачу інформацію щодо того скільки регістрів загального призначення надає конкретна архітектура процесора, як виконується перемикання в режим супервізора і т. д. Це відображається кількістю часу або тактових циклів, необхідних для фактичного виконання певного обов'язку.

Серед RTOS та юнікс-подібних систем, найпоширенішою абстракцією є API POSIX. У згаданій раніше RTLinux, доступ до АЗ можна отримати безпосередньо через треди, що працюють у просторі ядра. Але для цього потрібен дуже обережний та надійний дизайн програми, оскільки ці програми можуть зашкодити всій функціональності системи. Альтернативою є робота через драйвери пристроїв Linux, які також мають дуже абстрактний інтерфейс. Таким чином, поточна мінімальна абстракція надає вибір між швидким виконанням та надійністю.

1.10.2.3 Переносимість

Оскільки SoC зазвичай спрямовані на реалізацію специфічних задач, кожна конструкція SoC привносить інше апаратне середовище, яким повинна керувати операційна система. Це означає, що операційну систему доведеться фактично переносити на нову апаратну платформу для кожного нового SoC, на якому вона працює. Отже, ОС, придатна для SoC, повинна бути надзвичайно портативною, забезпечуючи інкапсуляцію різних функцій ОС. Ця інкапсуляція необхідна, щоб уникнути перехресних залежностей, які ускладнюють процес перенесення.

Код повинен бути не тільки портативним. Модифікації коду операційної системи не повинні спричиняти змін API або абстракцій, наданих операційною системою. Це особливо важливо, коли зміни в розділі дизайну SoC під час розробки спричиняють зміни, пов'язані з рівнем ОС. Такі зміни не повинні впливати на код програми, якщо вони не впливають безпосередньо на функціональність програми. [25]

Більшість комерційних RTOS та юнікс-подібних систем забезпечують широкий спектр апаратної підтримки. Ця апаратна підтримка, як правило, надається постачальником операційної системи і охоплює існуючі центральні процесори, розповсюджені шини, тощо. Лише декілька комерційних RTOS (наприклад, uC / OS-II) надають засоби для адаптації ОС до дуже специфічного середовища та можливостей SoC. Некомерційні ОС, такі як maRTE-OS, як правило, не забезпечують широкого спектру АЗ, однак вони є більш доступними і, отже, забезпечують засоби для повного використання середовища SoC.

1.10.3 Архітектури ОС

Архітектура ОС впливає на надійність вбудованої системи та її здатність відновлюватись після несправностей.

Існує дві архітектури ОС:

- монолітна (монолітне ядро, monolithic kernel)
- мікроядерна (microkernel). [26]

1.10.3.1 Монолітна ОС

Монолітне ядро запускає всі компоненти операційної системи в просторі ядра. Наприклад, монолітна ОС включає драйвери пристроїв, управління файлами, мережу та графічний стек як частину простору ядра (kernel space). Однак програми працюють у просторі користувача (user space). Хоча запуск користувальницьких додатків як процесів захищених у контексті менеджменту пам'яті (memory management), захищає монолітне ядро від помилок коду програм прошарку користувача, одна помилка програмування у файловій системі, стеку протоколів або драйвері може призвести до цілковитого збою ОС. Крім того, будь-яка зміна драйвера або системного файлу вимагає модифікації та перекомпіляції ОС. [26]. Детальніше з перевагами та недоліками монолітної ОС можна ознайомитися на таблиці 2.1.

Таблиця 2.1

Переваги та недоліки монолітної ОС

Переваги	Недоліки
Планування процесів, управління пам'яттю та управління файлами виконуються як єдиний великий процес	Помилка будь-якої служби може призвести до збоїв в роботі ОС.

в одному і тому ж адресному просторі, що може поліпшити продуктивність.	
Вся ОС міститься в одному статичному двійковому (binary) файлі, який може працювати швидше і надійніше, ніж динамічно пов'язані бібліотеки.	Додавання або видалення служби вимагає модифікації та перекомпіляції ОС.
	Служби ядра ОС представляють велику поверхню для атаки - якщо служба скомпрометована, це може зробити всю систему вразливою.
	Важко налагоджувати та підтримувати.

1.10.3.2 Мікроядерна ОС

Мікроядерна ОС складається з крихітного ядра, що забезпечує мінімальний функціонал. Мікроядро працює з командою некритичних процесів, які працюють поза простором ядра (в просторі користувача), що забезпечує функціональність ОС вищого рівня. У самому мікроядрі відсутні файлові системи та багато інших служб, які зазвичай очікуються від ОС. Мікроядерна ОС втілює фундаментальну інновацію в підході до забезпечення функціональності ОС: модульність є ключовим фактором, а малий розмір - побічним ефектом.

У мікроядерній ОС, доступ до всієї системи надається лише основному ядру ОС, що підвищує надійність та безпеку такої системи. Мікроядро захищає і виділяє пам'ять для інших процесів та забезпечує переключення між завданнями. Усі інші компоненти, включаючи драйвери та компоненти системного рівня, містяться у власному ізольованому просторі процесів (process space).

Ізоляція запобігає впливу помилок у компоненті на інші частини системи - єдине, що може вийти з ладу це сам компонент. Такі збої можна легко виявити, а несправний компонент можна перезапустити під час роботи системи настільки швидко, що перезапуск не впливає на продуктивність.

Під час розробки коду ізоляція всіх процесів має дві суттєві переваги:

- Помилки виявляються раніше в процесі розробки, і їх легко знайти в рядку коду в процесі що чинить негаразди. Для порівняння, приховані помилки можуть залишатися в процесі розробки драйверів для монолітної ОС навіть після запуску системи.
- Драйвери розглядаються як процеси застосування, що робить їх набагато простішими для написання та налагодження. Для того, щоб писати драйвер для мікроядра, не потрібно бути спеціалістом з драйверів пристроїв або налагоджувачем ядра. [26]

Детальніше з перевагами та недоліками мікроядерної ОС можна ознайомитися на таблиці 2.2.

Таблиця 2.2

Переваги та недоліки мікроядерної ОС

Переваги	Недоліки
Ізоляція несправностей та швидке відновлення системи.	Потрібно більше переключення контексту (context switching, user space - kernel space), що істотно збільшує витрати ресурсів.
Легке розширення - можливість розробляти драйвери пристроїв та розширення ОС без перекомпіляції.	Драйвери працюють автономно – витрати надлишкових ресурсів на комунікацію
Можливість перезапуску системної служби динамічно, не впливаючи на ядро (без перезавантаження	

системи).	
Менше коду, що працює в просторі ядра, зменшує поверхню атаки та підвищує безпеку ядра.	

1.10.3.3 Порівняння ОС архітектур

Порівняння ОС архітектур представлено у виді таблиці 2.3.

Таблиця 2.3

Порівняння ОС архітектур.

Категорія	Мікроядерна ОС	Монолітна ОС
Продуктивність	Повільніша через більшу кількість переключення контексту.	Швидша завдяки меншій кількості переключення контексту.
Оновлення системи	Нові драйвери та оновлення служб ОС можуть виконуватися без змін у ядрі і, отже, не вимагають перезавантаження ОС.	Нові драйвери та оновлення служб ОС вимагатимуть перезавантаження ОС.
Відновлення у випадку збою	Може перезапустити будь-яку службу окремо, не перериваючи роботу ядра.	Відновлення служби після збою вимагає перезавантаження ОС

Сертифікація	Простіше і дешевше сертифікувати ядро. Більшість оновлень системи не вимагають повного циклу сертифікації, а обмежуються лише оновленою службою чи драйвером.	Важко та дорожче пройти сертифікацію. Оновлення системи вимагають повного циклу сертифікації, оскільки ОС, як правило, перебудовується
Підтримка	Простіше і менш трудомістке обслуговування та усунення несправностей існуючих систем. Користувачі можуть оновити, усунути неполадки та перезавантажити службу, без перезавантаження всієї ОС	Складні та трудомісткі для обслуговування та усунення несправностей. Користувачам необхідно перезавантажити ОС під час виконання більшості оновлень системи, кроків з усунення несправностей або перезавантаження служби.

1.10.4 Оглядовий аналіз процесу розробки ПЗ для ОС рішень

1.10.4.1 Збір інструментів необхідних для розробки ПЗ

Для розробки ПЗ на ОС незалежно від ОС що використовується розробником, необхідний лише інструмент для компіляції ПЗ. Якщо розробник планує компілювати програми безпосередньо на платі, такий

компілятор найчастіше надається виробником плати. Якщо розробник планує крос-компілювати (cross-compile) додаток на ПК, можливе користування будь-яким компілятором здатним до крос-компіляції. Одним із найпоширеніших компіляторів являється GCC, the GNU Compiler Collection – окрім широкого функціоналу і доступності на багатьох платформах, він також цілком безкоштовний. [27]

1.10.4.2 Підготовка до запуску тестового додатку

Для приблизної оцінки процесу розробки ПЗ для ОС буде використана примітивна програма з мінімальним функціоналом – «Hello world» як і для оцінки Bare metal підходу. Детальніше з даною програмою можна ознайомитись у Додатку А. Задача даної програми у виводі текстового блоку “Hello world!” у вихідний потік.

Для запуску тестової програми необхідно **скомпілювати** її. Якщо програма була крос-компільована на ПК, програму також необхідно завантажити на плату.

Одна з найбільших переваг використання ОС для вбудованих систем це підтримка великої кількості пристроїв, файлових систем, підключення до мережі та підтримки інтерфейсу користувача. Саме завдяки цьому, запуск тестового додатку “Hello world” з використанням ОС рішення практично не вимагає зусиль для розробника програми. Звісно складні програми потребують більшої настройки а можливо і підключення додаткових модулів. Незважаючи на це, використання ОС значно спрощує розробку ПЗ. Один із найбільших недоліків ОС підходу – ресурси що будуть виділені на підтримку роботи ОС. Варто зазначити і надлишкові витрати на комунікацію між АЗ та ПЗ – будь який сигнал від АЗ сперш пройде через ОС прошарок і тільки після цього досягне ПЗ і навпаки.

1.10.5 Типи ОС для вбудованих систем

1.10.5.1 RTOS

Основна відповідальність ОС полягає в управлінні апаратними ресурсами та діями в системі: планування прикладних програм, запис файлів на диск, надсилання даних через мережу тощо. Коли ОС повинна одночасно обробляти декілька подій і забезпечувати відповідь системи на ці події в чітко виражений срок, таку ОС називають RTOS.

Багато вбудованих систем вимагають поведінки або реакції на зміни у навколишньому середовищі в режимі реального часу, і через обмеження апаратних ресурсів продуктивність та ефективність таких систем є головними пріоритетами. RTOS забезпечує суворе управління ресурсами та планування, необхідне для задоволення потреб програм - за допомогою багатозадачності, потоків, превентивному плануванні задач по пріоритетності, і швидкого переключення контексту.

RTOS, як правило, має невеликий розмір, проте кожна RTOS повинна бути налаштована з урахуванням можливостей системи, яку ця ОС підтримує. Від простої конфігурації ядра, що управляє невеликою кількістю завдань, до повнофункціональної RTOS, що управляє сотнями завдань та підсистем, включаючи графіку, мережу, файлову систему, аудіо та багато іншого - RTOS повинна гнучко масштабуватися з урахуванням системних вимог та ресурсів. [26]

Найросповсюдженіші RTOS:

- QNX
- OS-9 (MacOS)
- VxWorks/Tornado
- IA-SPOX
- RTX
- Falcon

- Hyperkernel

1.10.5.2 GPOS

GPOS – General purpose operating systems – це операційні системи загального призначення такі як Windows, Linux, Unix та ін. У контексті розробки для вбудованих систем, під GPOS маються на увазі саме Юнікс-подібні операційні системи.

Юнікс-подібні операційні системи - операційні системи, які були створені під впливом Unix. Термін включає відкриті (open source) операційні системи, утворені від Unix, комерційні і запатентовані розробки, а також версії, засновані на вихідному коді (source code) Unix.

Ерік Рэймонд запропонував розділити Unix-подібні системи на 3 типи:

Генетичний Unix: Системи, історично пов'язані з кодовою базою (code base) AT&T. Більшість, але не всі комерційні дистрибутивні системи Unix-системи підпадають під цю категорію. Так само, як і в BSD-системі, які виконували результати університету Берклі в поздних 1970-х і ранніх 1980-х. У деяких із цих систем код AT&T відсутній, але до цих пір можливо відслідкувати походження коду від розробки AT&T.

Unix за товарним знаком, або брендом: такі системи, в основному комерційні, були визначені The Open Group як відповідні Єдиній специфікації UNIX. Більшість цих систем були створені з комерційної кодові бази UNIX System V у тій чи іншій формі.

Unix за функціональністю: У цілому, будь-яка система, яка дещо відповідає специфікації UNIX. До таких можна віднести Linux та Minix, які підтримують самі Unix-системи, але не пов’язані з кодовою базовою AT&T. Більшість відкритих реалізацій Unix не признані Юнікс-подібними офіційно, у зв'язку із дороговизною сертифікацією The Open Group.

Найросповсюдженіші Юнікс-подібні ОС:

- Linux
- Minix
- OpenSolaris
- Plan 9
- BSD

1.10.5.3 Linux

Linux - це сімейство Unix-подібних операційних систем на основі ядра Linux з відкритим кодом (open source). Дистрибутиви Linux включають в себе ядро Linux, системне програмне забезпечення та бібліотеки, багато з яких забезпечені GNU Проектом.

ОС на базі Linux - це модульна операційна система, схожа на Unix, створена на основі принципів, започаткованих в Unix протягом 1970-х та 1980-х років. Така система використовує монолітне Linux ядро, яке обробляє управління процесами, мережу, доступ до периферійних пристроїв та файлові системи. Драйвери пристрою або інтегровані безпосередньо з ядром, або додаються як модулі, завантажені під час роботи системи.

ОС Linux дотримуються POSIX, SUS, LSB, ISO та ANSI стандартів, наскільки можливо, хоча на сьогоднішній день лише один дистрибутив Linux був сертифікований POSIX.1 - Linux-FT.

Ядро Linux доступне для багатьох пристроїв, починаючи від мобільних телефонів до суперкомп'ютерів; воно працює на різноманітному спектрі комп'ютерних архітектур, включаючи портативний iPAQ на базі ARM та IBM mainframes System z9 або System z10. Спеціалізовані дистрибутиви та варіанти ядра існують для менш розповсюджених архітектур; наприклад, вилка ядра ELKS може працювати на 16-бітних мікропроцесорах Intel

8086 або Intel 80286, тоді як варіант ядра µClinux може працювати в системах без блоку управління пам'яттю.

Через низьку вартість і простоту налаштування Linux часто використовується у вбудованих системах. У секторі телекомунікаційного обладнання більшість апаратних засобів обладнання (CPE) працює на ОС на базі Linux.

1.10.5.4 Різниця між RTOS та GPOS

Основна різниця між RTOS та GPOS представлена у вигляді таблиці 2.4.

Таблиця 2.4

Основна різниця між RTOS та GPOS.

RTOS	GPOS
планування базується на пріоритетності задач	планування регулюється динамічно для оптимізації роботи
має передбачувану поведінку	поведінку неможливо передбачити
працює з розрахунку на “найгірший випадок” (worst case scenario)	робота оптимізована для “середнього випадку” (average case scenario)
“Нечесна” пріоритизація	“Чесна” пріоритизація

Linux - це ОС загального призначення (GPOS); її застосування у вбудованих системах, як правило, мотивовано наявністю підтримки великої кількості пристроїв, файлових систем, підключення до мережі та підтримки інтерфейсу користувача. Усі ці речі можуть бути доступні в

RTOS, але часто з менше підтримані зі сторони постачальника або потребують додаткових витрат чи зусилль на інтеграцію необхідних компонентів.

Багато RTOS не є повноцінними ОС у тому сенсі, як Linux, оскільки вони складаються із бібліотек статичного лінкування, які забезпечують лише планування завдань, IPC, синхронізацію та переривання - по суті лише ядро планування. Така бібліотека лінкується з кодом програми, щоб створити єдиний виконуваний файл, який система завантажує безпосередньо (або через бутлодер bootloader). Більшість RTOS безпосередньо не підтримують завантаження та вивантаження коду динамічно з файлової системи, як це було б у Linux - приєднання статичного виконуваного файлу відбувається при запуску пристрою та працює до вимкнення живлення.

Linux не надає підтримку виконання програми у реальному часі. RTOS надає гарантійне планування для забезпечення детермінованої поведінки та своєчасного реагування на події та переривання. У більшості випадків це відбувається за допомогою алгоритму попереджувального планування, заснованого на пріоритеті, де завдання з найвищим пріоритетом завжди готове до запуску перериваючи будь-яке завдання з нижчим пріоритетом.

Linux має ряд варіантів планування, включаючи планувальник (scheduler) у реальному часі, але Linux не може гарантувати повної відповідності реальному часу. Якщо для певної програми немає потреби в "жорсткому" режимі реального часу, Linux може задовольнити потреби такої програми, але типові затримки в реальному часі Linux становлять близько десятків або сотень мікросекунд, тоді як типове ядро реального часу RTOS може досягти затримок від нуля до декількох мікросекунд.

Інша проблема вбудованого Linux полягає в тому, що йому потрібні значні ресурси центрального процесора, і багато оперативної пам'яті, щоб завантажитися (що може зайняти кілька секунд). Потреби RTOS, з іншого боку, може завантажитися за мілісекунди, і потребує набагато менші ресурси.

Існують більші продукти RTOS, які демонструють деякі особливості GPOS, такі як динамічне завантаження, файлові системи, мережа, графічний інтерфейс (наприклад, в QNX), а багато RTOS надають API POSIX (зазвичай не такий вдалий як їх власний API реального часу)

наприклад, VxWorks і QNX, так що багато коду, розробленого для Linux та Unix, можна відносно легко перенести. Ці великі, всеохоплюючі продукти RTOS залишаються масштабованими, тому функціональність, яка не потрібна, не включається. [28]

1.11 Висновки

Bare metal надає розробнику повний контроль над кожним аспектом роботи АЗ, що надає як переваги, так і суттєві недоліки. Потенційно, такий підхід надає можливість отримання системи з максимальним рівнем продуктивності завдяки тому, що програма взаємодіє з АЗ без посередників, і ресурси не витрачаються на надлишковий функціонал. На практиці названого рівню продуктивності досягнути вкрай важко. Bare metal проект також виставляє високі вимоги щодо учасників такого проекту, оскільки такий проект потребує реалізації усіх необхідних для SDR пристроїв, інтерфейсів, і т.п. на низькому рівні, а значить, і спеціалістів які мають досвід у реалізації усіх необхідних компонентів для проекту.

На момент створення даної роботи, вартість Altera SoC EDS ліцензії на одного програміста згідно [22] складає $2,995 USD на рік. У випадку використання ОС, ціна необхідних компонентів для розробки ПЗ варіюється – починаючи від безпосередньої вартості ОС (як безкоштовно, так і $3-5 USD за одиницю згідно [23]). Для розробки на ОС, доступне велике розмаїття компонентів, і ціна на них починається від цілком безкоштовного стеку GNU технологій, і закінчується Matlab/Simulink, ціна на які може складати $2,150 для одного програміста (залежно від обраного пакету [24]). Таким чином, вартість необхідних компонентів для розробки на Bare metal можна вважати високою, а використання ОС може знизити витрати на інструментарій до мінімально необхідних.

Використання ОС рішень для вбудованих систем надає велику кількість переваг - ОС забезпечує ініціалізацію та підтримку АЗ, управління завданнями (включаючи планування завдань), загальне управління ресурсами (пам’ять, периферичні пристрої тощо), міжпроцесорний зв’язок

та контроль багатозадачності (блокування (locks), семафори (semaphores) тощо).

Серед існуючих ОС архітектур, саме монолітна ОС найбільше підходить для реалізації SDR, оскільки планування процесів, управління пам’яттю та управління файлами виконуються як єдиний процес в одному і тому ж адресному просторі, що позитивно позначиться на продуктивності системи. У випадку реалізації SDR, швидкість роботи може виступати основним критичним ресурсом.

RTOS забезпечує суворе управління ресурсами та планування, проте кожна RTOS повинна бути налаштована з урахуванням можливостей системи, яку ця ОС підтримує. Натомість, GPOS - Юнікс-подібні операційні системи, в особливості якщо ОС надана виробником SoC, працюють «з коробки», та не потребують додаткових налаштувань за виключенням вузькоспеціалізованих проектів. Також, типова Юнікс-подібна ОС найчастіше оптимізована для “середнього випадку” (average case scenario), тоді як RTOS працює з розрахунку на “найгірший випадок” (worst case scenario). Таким чином, в середньому GPOS забезпечить більш оптимальну роботу SoC SDR систем.

РОЗРОБЛЕННЯ ПРОГРАМ ДЛЯ SDR СИСТЕМ НА БАЗІ SOC З ВИКОРИСТАННЯМ GPOS UNIX-ПОДІБНОЇ ОС

1.12 Аналіз існуючих підходів для Розроблення програм для SDR систем на базі SoC з використанням GPOS Unix-подібної ОС

Як було зазначено раніше, існує два підходи до реалізації SDR на SoC:

- Bare metal
- OS

Основна перевага в використанні ОС для роботи з SDR в можливості використання існуючого програмного забезпечення.

Переваги використання існуючого ПЗ в роботі з SDR:

- Велика кількість користувачів.
- Добре протестоване у різноманітних випадках ПЗ.
- Підтримка користувача - форуми та офіційна документація, наявність прикладів для наслідування.
- Підтримка ПЗ.
- Відкритий код - можливість самостійно вносити модифікації до продукту для задоволення специфічних потреб проекту.

Недоліки використання існуючого ПЗ в роботі з SDR:

- Широка направленість - ПЗ не розроблялося для підтримки одного конкретного АЗ, і як наслідок не гарантує оптимального використання наявних ресурсів.
- Велика кодова база - внесення змін для задоволення специфічних потреб проекту може бути ускладнено великою кількістю наявного коду.

1.12.1 Linux індустріальна підсистема вводу/виводу

Індустріальна підсистема вводу/виводу (IIO Subsystem) призначена для підтримки пристроїв, які в деякому сенсі є аналогами аналогово-цифрових або цифро- аналогових перетворювачів (АЦП, ЦАП) [29]. Пристроями, що належать до цієї категорії, є:

- аналогово-цифрові перетворювачі (ADC);
- акселерометри (Accelerometers);
- гіроскопи (Gyros);
- інерційні вимірювальні пристрої (IMUs - Inertial Measurement Units);
- ємністно-цифрові переворювачі (CDCs - Capacitance to Digital Converters);
- датчики тиску (Pressure Sensors);
- датчики кольорів, світла та наближення Color, Light and Proximity Sensors);
- температурні датчики (Temperature Sensors);
- магнетометри (Magnetometers);
- цифро-аналогові перетворювачі (DACs);
- синтезатори прямого цифрового синтезу (DDS - Direct Digital Synthesis);
- ФАПЧ (PLLs - Phase Locked Loops);
- підсилювачі змінного / програмованого підсилення (VGA/PGA - Variable/Programmable Gain Amplifiers).

IIO підсистема заповнила розрив між дещо схожими підсистемами апаратного моніторингу (Hardware Monitoring - hwmon) та вхідними підсистемами (input). Hwmon в значній мірі спрямований на датчики низької швидкості вибірки, які використовуються в таких програмах, як контроль швидкості обертання вентилятора та вимірювання температури та ін. Input орієнтований на пристрої введення для взаємодії з людиною:

клавіатура, мишка, сенсорний екран, джойстик. У деяких випадках між ними та IIO є значне перекриття.

Типовий пристрій, що належить до категорії IIO, підключається через SPI або I2C. Однак типові пристрої, які працюють з прямим доступом до пам'яті (DMA), такі, як підключені до високошвидкісного синхронного послідовного порту (McBSP, SPORT), або до високошвидкісного синхронного паралельного порту (EPI, PPI), або периферійних пристроїв FPGA, також підпорядковуються цій підсистемі. Оскільки останні на відміну від SPI або I2C, як правило, не абстрагуються драйверами шин Linux, вони залежать від реалізації платформи процесора.

Структура IIO підсистеми показана на рисунку 3.1.

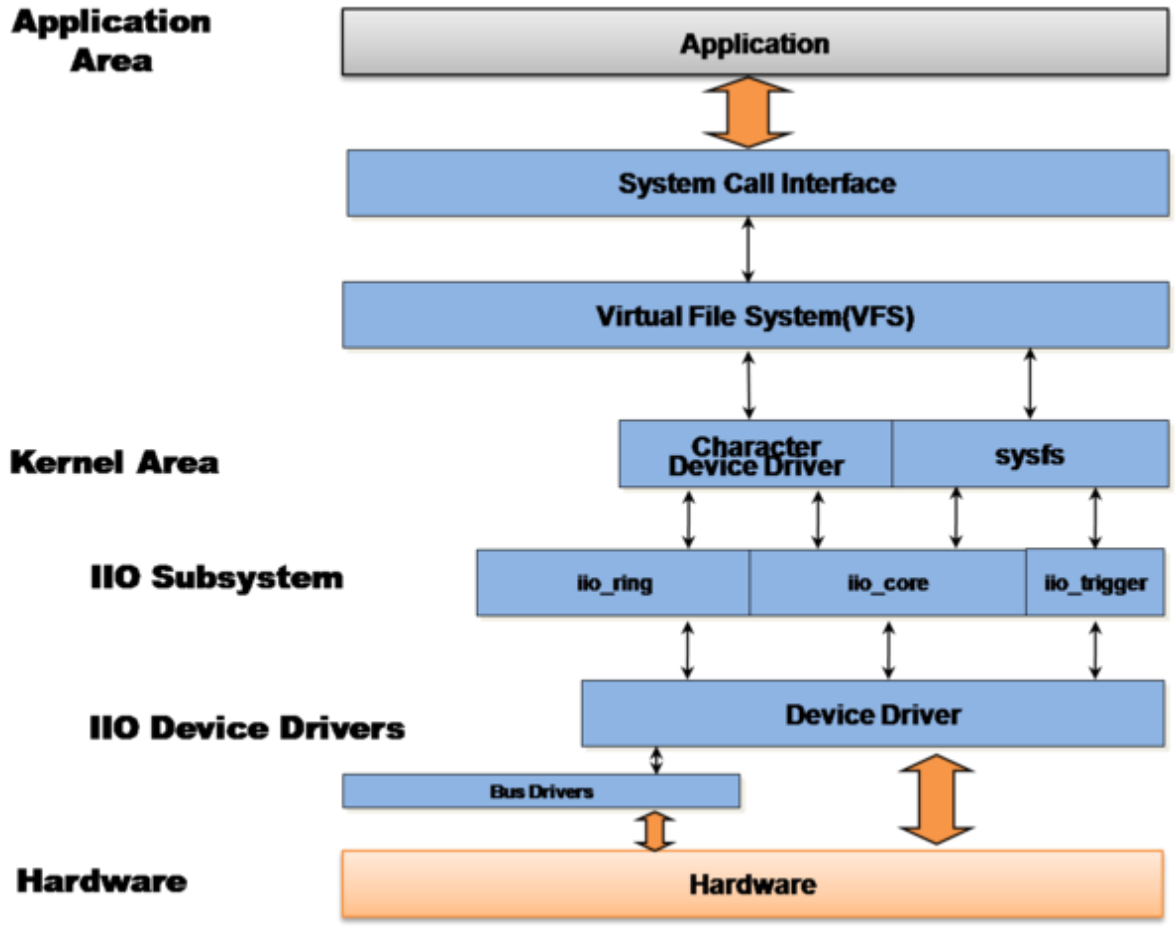

Рисунок 3.1 - Структура IIO підсистеми

sysfs — віртуальна файлова система в операційній системі Linux. Експортує до простору користувача інформацію ядра Linux про присутні в

системі пристрої та драйвери. Вперше з`явилась в ядрі версії 2.6. Необхідність створення була викликана застарілою системою роботи ядра з пристроями.

Функціональність IIO підсистеми включає в себе:

- базову реєстрацію пристрою та поводження з ним;
- доступ шляхом опитування (Polled access) до каналів пристроїв через sysfs;
- заходи chrdevs, які подібні до введення даних, вони надають маршрут до користувацького простору для подій, що запускаються апаратним забезпеченням;
- підтримку апаратного кільцевого буферу (Hardware ring buffer support);
- fifo/ring buffers, безпосередньо включені на мікросхемі датчика в багатьох останніх датчиках, що значно знижує навантаження на центральний процесор за рахунок буферизації відносно великої кількості виборок даних на основі внутрішніх тактових частот вибірки;
- підтримку в кожному ring buffer, як правило, події chrdev для передачі подій, таких як заповнення буферу на 50% та chrdev доступу, за допомогою якого можуть бути прочитані необроблені дані;
- підтримку тригерного та програмного ring buffer (kfifo).

Підтримка тригерного та програмного ring buffer полягає в наступному:

- у багатьох додатках для аналізу даних корисно мати можливість збору даних на основі якогось зовнішнього сигналу (тригера), в якості тригерів можуть виступати: сигнал готовності даних, лінія gpio підключена до якоїсь зовнішньої системи, періодичні переривання процесора (interrupt), читання простору користувача в визначений файл в sysfs;
- один тригер багатократно ініціалізує захват та читання даних з декількох датчиків;
- тригери використовуються в iio для заповнення програмних кільцевих буферів ring buffer, що діють дуже схоже з апаратними буферами;

- тригери можуть бути абсолютно не пов'язані з самим датчиком.

IIO підсистема може виступати в якості шару під іншими елементами ядра, забезпечуючи засоби отримання даних по типу АЦП, або керування виводом даних по типу ЦАП. Підтримка функціональності зростає по мірі необхідності з появою нових рішень для застосування IIO.

IIO використовує кільцевий буфер (*ring buffer*), який представляє собою структуру даних, що використовує єдиний буфер фіксованого розміру, як ніби він був з'єднаний кінцем до кінця *(end-to-end*). Ця структура добре підходить для буферизації потоків даних. Ці буфери зазвичай використовуються для вирішення проблеми джерело-отримувач. У деяких додатках джерело (наприклад, АЦП) може перезаписувати старі дані, якщо отримувач (наприклад, програма для користувацького простору) не в змозі миттєво їх обробити, тому буфер повинен мати достатній розмір, для того щоб такі обставини ніколи не виникали.

На рисунку 3.2 показано структуру IIO Ring Buffer / kfifo.

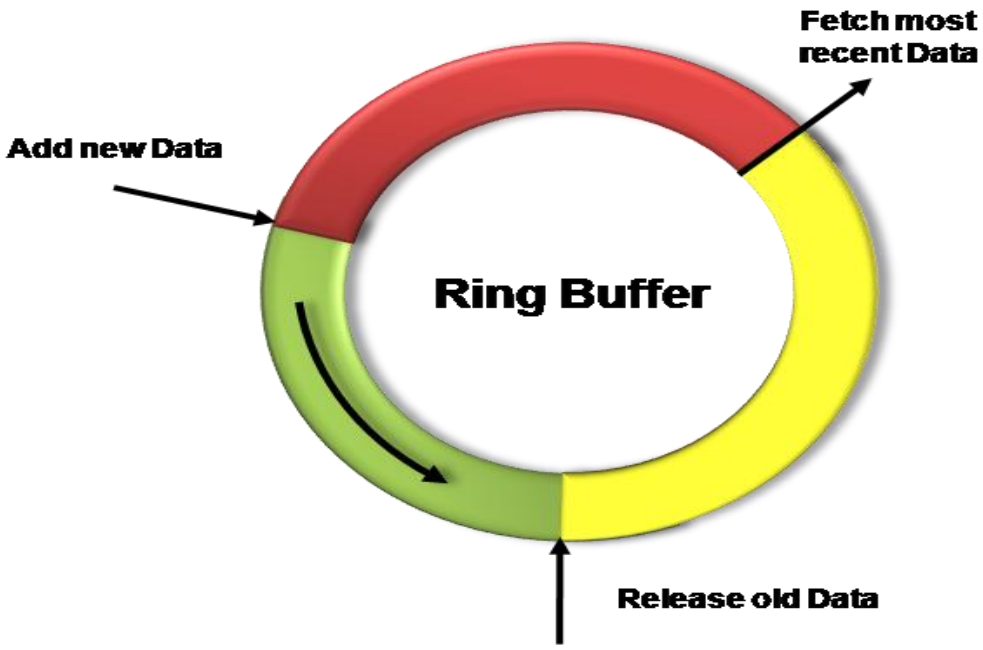

Рисунок 3.2 - Структура IIO Ring Buffer / kfifo

Більш детально з описом роботи з Industrial I/O можна познайомитись за посиланням [29]

1.12.2 Бібліотека libiio

Бібліотека libiio - це бібліотека, розроблена Analog Devices для полегшення розробки програмного забезпечення і сполучення (інтерфейсу) з Linux Industrial I/O (IIO) пристроями [30].

Бібліотека абстрагує деталі апаратного забезпечення низького рівня та забезпечує простий, але повний інтерфейс програмування, який можна використовувати при створенні нових сучасних проектів.

Бібліотека складається з одного API високого рівня та декількох програм:

- "локальний" бекенд (“local” backend), який є інтерфейсом ядра Linux через віртуальну файлову систему sysfs;
- "мережевий" бекенд (“network” backend,), який є інтерфейсом сервера iiod через мережеве з’єднання.

Сервер IIO Daemon (IIOD) - програма, яка використовує libiio. Він створює контекст libiio, який використовує "локальний" бекенд, а потім обмінюється створеним контентом у мережі з будь-яким підключеним до сервера клієнтським додатком, що використовує "мережевий" сервіс libiio. На рисунку 3.3 показано структуру обміну даними з використанням бібліотеки libiio IIOD серверу.

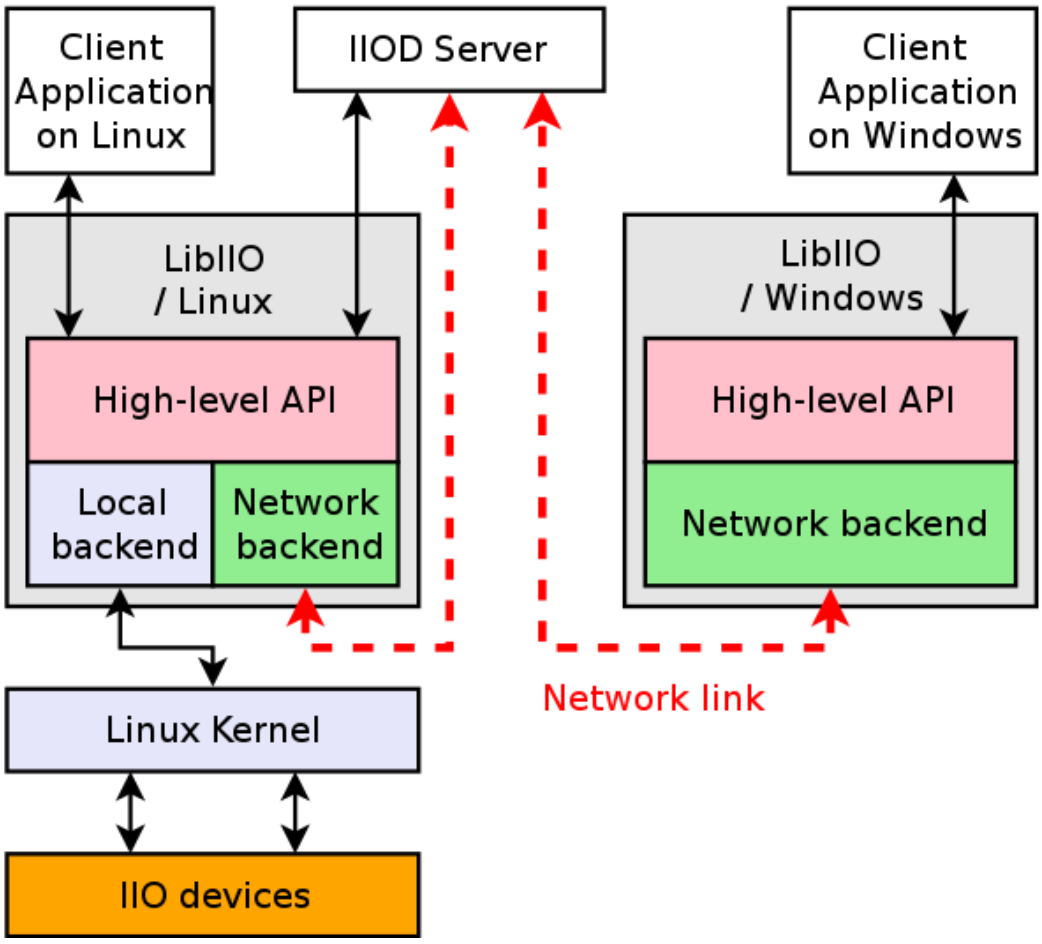

Рисунок 3.3 - Структура обміну даними з використанням бібліотеки libiio IIOD серверу.

Якщо ви просто хочете використовувати libiio та iiod, які можуть бути на поперелньо скомпільованому образі - є багато місць, до яких можна передавати дані: MATLAB & Simulink, Visual Analog, IIO Oscilloscope for Windows & Linux, GNU Radio

Незважаючи на те, що libiio була розроблена в основному компанією Analog Devices Inc., вона є активною бібліотекою з відкритим кодом. Бібліотека випускається під GNU Lesser General Public License (LGPL), що дозволяє кожному користуватися бібліотекою на будь-якому постачальнику процесора / FPGA / SoC, який може контролювати будь-які постачальники периферійних пристроїв (АЦП, ЦАП тощо) як локально, так і віддалено. Сюди входять програми закритого або відкритого коду,

комерційні або некомерційні програми (за умови ліцензії LGPL, свободи, зобов'язання та обмеження). [31]

1.12.3 Використання середовища Matlab Simulink для розроблення програм для SDR систем на базі SoC

1.12.3.1 Забезпечення взаємодії Matlab/Simulink та SDR

SDR системи в тому числі і на основі SoC, відносяться до класу вбудованих систем (embedded system), для їх проектування використовуються методи та засоби, які в загальному вигляді відрізняються від тих, що застосовуються для проектування комп’ютерних систем загального призначення. На рисунку 3.4 показано структуру процесу створення радіосистеми з використанням модельно-орієнтованого проектування з середовища Matlab/Simulink для подальшої компіляції в спеціалізованих середовищах для розробки та проектування [32].

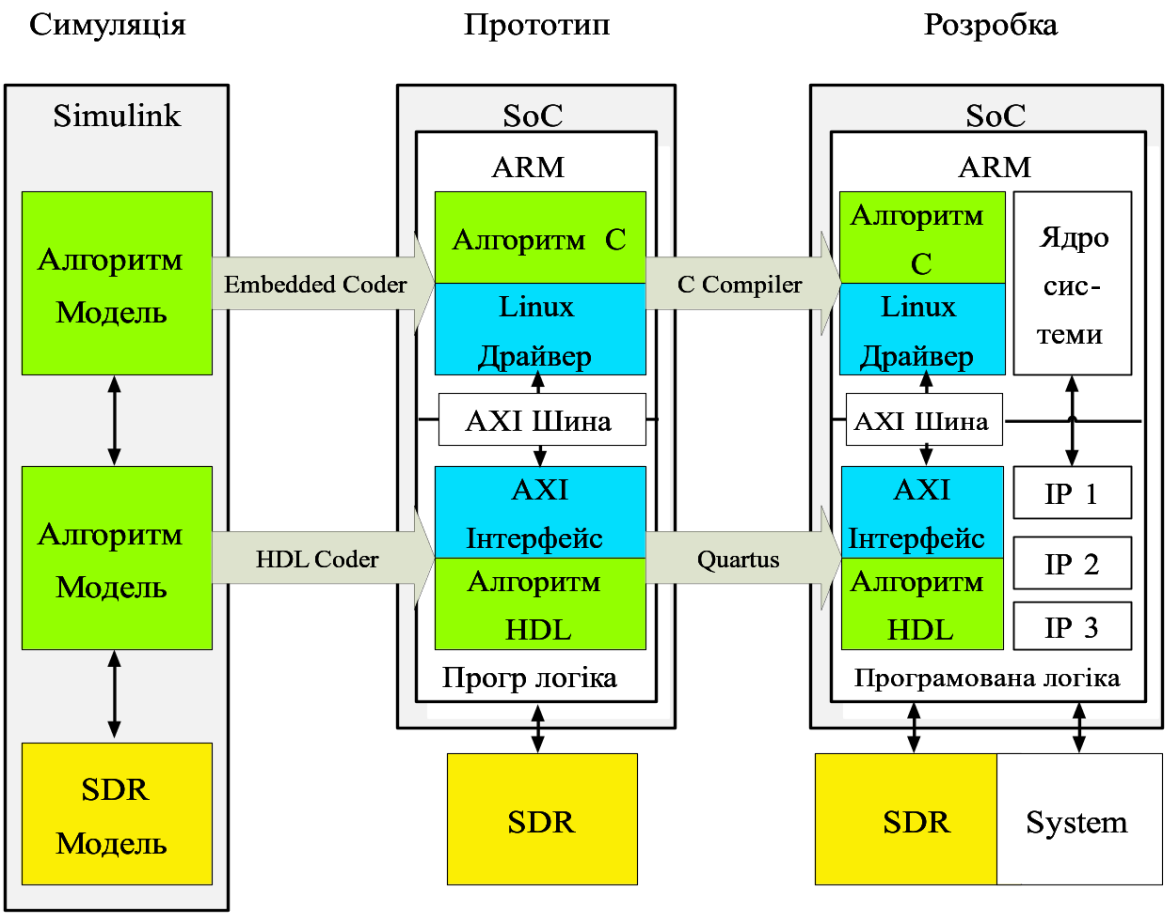

Рисунок 3.4 - Структура процесу створення радіосистеми з використанням модельно-орієнтованого проектування з середовища Matlab/Simulink для подальшої компіляції в спеціалізованих середовищах для розробки та проектування.

Першим етапом є створення моделі радіосистеми в середовищі Matlab/Simulink, при цьому в моделі повинні бути присутні модулі, які відповідають за взаємодію з апаратною платформою (SDR модель). При створенні моделі враховується яка частина моделі (алгоритми) будуть реалізуватись на FPGA, а яка на ARM процесорах. Етап закінчується програмно-апаратною симуляцією з використанням цільової платформи, яка організовується на базі відладочних засобів.

На другому етапі, після того, як модель відпрацьована, генерується HDL код на мові VHDL (Verilog) за допомогою HDL Coder та C код за допомогою Embedded Coder, які входять до пакету Matlab/Simulink. На цьому етапі здійснюється відладка з використанням апаратної платформи

для прототипування SDR, яка складається з модуля SDR радіотрансивера та SoC модуля. Перевіряється ефективність розроблених алгоритмів та моделей, здійснюються всі необхідні налаштування, система вже є працюючою як під управлінням з боку Matlab/Simulink, так і автономно. Етап закінчується створенням прототипу та внутрішньосхемною симуляцією і перевіркою з використанням цільової платформи, яка організовується на базі відладочних засобів..

Останнім етапом є автоматична компіляція С коду з використанням gcc linux компілятора та компіляція коду VHDL (Verilog) з використанням копмілятора, який входить до складу пакету Quartus (для SoC Altera), або Vivado (для SoC Xilinx). Етап закінчується перенесенням скомпільованого коду до цільової платформи з подальшою автономною роботою.

Процес створення радіосистеми з використання модельно-орієнтованого проектування гарантує, що після того, як алгоритм системи SDR буде імплементований в кінцевий продукт, він буде повністю перевірений і випробуваний і забезпечить впевненість в надійності системи. [32]

1.12.3.2 Програмно-апаратна симуляція з використанням середовища Matlab/Simulink

Програмно-апаратна симуляція (HIL - Hardware-in-the-Loop Simulation) являє собою метод, який використовується в розробці і випробуванні складних технічних і технологічних вбудованих систем реального часу. Використання програмно-апаратного моделювання дозволяє проводити внутрішньосхемне і внутрішньосистемне тестування розроблених рішень і математичних моделей, істотно скорочувати терміни і вартість розробки.

Процес HIL - симуляції в загальному вигляді складається з трьох основних етапів:

- створення математичної моделі реального середовища, в якій буде використовуватися апаратний пристрій;

- тестування розробленої математичної моделі на реальному апаратному пристрої;
- реалізація процесу в реальному пристрої з урахуванням результатів тестування моделі.

При HIL - симуляції можливі як варіанти з математичним описом (програмною моделлю) середовища і внутрішньосхемним моделюванням процесу, так і з математичним описом процесу (програмною моделлю) і внутрішньосистемним моделюванням його поведінки в реальному середовищі.

Програмно-апаратне моделювання телекомунікаційних процесів і систем в Matlab/Simulink базується на використанні IIO System Object [1], який в свою чергу базується на специфікації системних об'єктів Matlab [2]. IIO System Object працює як з Matlab, так і Simulink і призначений для обміну даними по мережі Ethernet з ADI апаратної платформою, підключеної до платформи FPGA / SoC, яка працює під управлінням ADI Linux.

IIO System Object побудований на бібліотеці libiio і дозволяє моделі Matlab або Simulink обмінюватися потоками даних з апаратною платформою, контролювати її налаштування та здійснювати моніторинг різних параметрів. На рисунку 3.5 представлена схема рівнів, яка схематично показує архітектуру процесу програмно-апаратного моделювання.

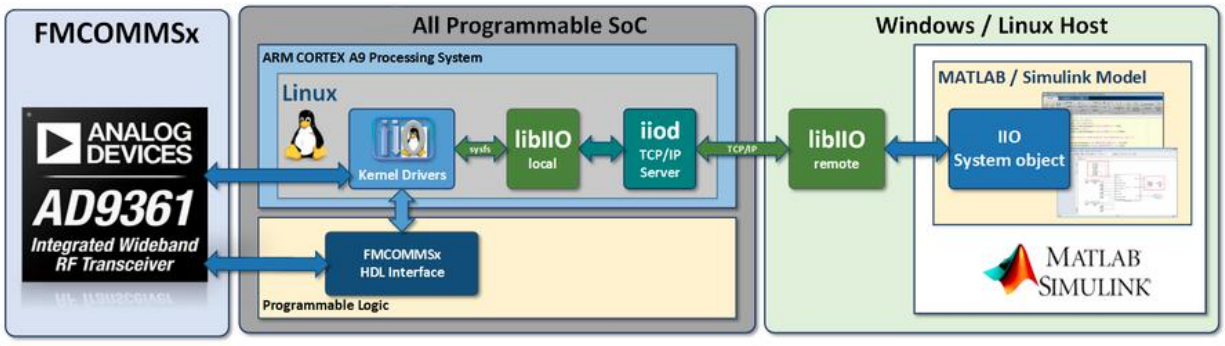

Рисунок 3.5 - Схема рівнів.

Для роботи середовища Matlab з програмно апаратної платформою створюється конфігураційний файл з розширенням *.cfg і ім'ям, що збігається з ім'ям пристрою, встановленим в діалоговому вікні

конфігурації блоку. Цей файл містить набір полів, які визначають драйвери Linux для цільового пристрою і канали конфігурації, які будуть використовуватися блоком Simulink, або скрипт файлом Matlab.

Вхідні і вихідні порти блоку Simulink, які відповідають IIO System Object, визначаються за допомогою його властивостей, а також за допомогою файлу конфігурації, специфічного для цільового пристрою. Вхідні і вихідні порти класифікуються як порти передачі даних і управління. Порти даних використовуються для прийому/передачі буферів безперервних даних з/в цільовий пристрій в режимі обробки кадрів, в той час як порти управління використовуються для конфігурації і моніторингу різних параметрів пристрою. Число і розмір портів даних налаштовується в діалоговому вікні конфігурації блоку, в той час як порти управління визначені в файлі конфігурації. [33,34]

1.12.3.3 Створення прототипу SDR системи в середовищі Matlab/Simulink

В середовищі Matlab/Simulink для переходу від процесу симуляції до прототипу використовуються як вбудовані засоби, так і засоби сторонніх розробників (third-party software) [35].

Для створення протопиту для FPGA повинне бути встановлено середовище розробки FPGA в залежності від SoC, який буде використовуватись в цільовій платформі: Altera Quartus II, Xilinx ISE, Xilinx Vivado.

Створення протопиту для FPGA полягає в створенні коду на мові verilog/VHDL, для цього в середовищі Matlab повинні були встановлені пакети розширень MATLAB Add-Ons для роботи з SDR системами та розробки SDR систем з використанням SoC та Embedded.

Процес створення прототипу закінчується (за необхідності внутрішньсхемної симуляції FIL — FPGA in the Loop) синтезом та

генерацією прошивки (bitstream) для вбудованої системи FPGA та її програмуванням через порти JTAG, Ethernet, або записом на носій. Згенерований проект IP Core може оптимізуватись та модернізуватись вже безпосередньо в середовищі розробки FPGA.

Створення протопиту для ARM процесора SoC полягає в генерації коду на мові C/C++. В результаті створюється прототип на мові C/C++ включаючи Makefile необхідний для компіляції. Після створення прототипу перенесення на цільову платформу здійснюється шляхом крос-компіляції.

1.12.4 Використання середовища GNU radio для розроблення програм для SDR систем на базі SoC

1.12.4.1 Основна інформація GNU Radio

GNU Radio - це безкоштовний набір інструментів з відкритим кодом для розробки програмного забезпечення, який надає блоки обробки сигналів для реалізації SDR. Його можна використовувати з радіочастотним обладнанням для створення програмно визначених радіоприймачів або без обладнання в симульованому середовищі. GNU Radio широко використовується в любительських, академічних та комерційних середовищах для підтримки досліджень бездротового зв'язку та радіосистем.

GNU Radio виконує всю обробку сигналу. GNU Radio може бути використаний для написання програм для отримання даних з цифрових потоків або для передачі даних у цифрові потоки, які потім передаються за допомогою АЗ. GNU Radio має фільтри, коди каналів, елементи синхронізації, еквалайзери, демодулятори, вокодери, декодери та багато інших елементів (у GNU Radio такі елементи називаються блоками), які

зазвичай зустрічаються в радіосистемах. Що ще важливіше, GNU Radio включає метод підключення цих блоків, а потім керує способом передачі даних з одного блоку в інший. Розширення GNU Radio також є досить простим; користувач має змогу створити відсутній блок та додати його у проект.

Оскільки GNU Radio - це програмне забезпечення, воно може обробляти лише цифрові дані. Зазвичай складні вибірки базової смуги є типом вхідних даних для приймачів і типом вихідних даних для передавачів. Потім аналогове обладнання використовується для зсуву сигналу на бажану центральну частоту. За винятком цієї вимоги, будь-який тип даних може передаватися з одного блоку в інший - будь то біти, байти, вектори, пакети або складніші типи даних.

Програми GNU Radio в основному написані з використанням мови програмування Python, тоді як критично важливий для продуктивності шлях обробки сигналу реалізований в C ++ за допомогою розширень з плаваючою комою процесора, де це можливо. Таким чином, розробник може реалізувати високопродуктивні радіосистеми в режимі реального часу в простому у використанні середовищі швидкого розробки додатків. [36]

1.12.4.2 Програмування з використанням GNU Radio

GNU Radio включає в себе GNU Radio Companion, що є графічним інтерфейсом користувача, подібним до Simulink. Це дозволяє створювати та виконувати програми обробки сигналів без навиків у програмуванні. Однак, якщо користувач планує розширити GNU Radio (тобто додати нову функціональність), тоді користувач мусить написати код. Для створення додатків, занадто складних для GNU Radio Companion, Python - це найпростіший шлях. Для критичного для продуктивності коду слід написати код C ++. [36]

Основна перевага GNU Radio – порівняно низький «поріг вступу» для розробника через наявність достатньо інтуїтивного користувацького інтерфейсу. Часто GNU Radio використовують для прототипування через швидкість розробки з його використанням. Також, GNU Radio надає широкий функціонал. Основний недолік GNU Radio – дещо урізані можливості для тонкої настройки та оптимізації ПЗ. Широкий функціонал що надається GNU Radio також може ввести і функціонал який не є необхідним для досягнення основної мети ПЗ, і, таким чином, використання надлишкових ресурсів.

1.13 Практична реалізація

Практична реалізація роботи з бібліотекою libiio була апробована в процесі наукових досліджень, які проводились в НДІ телекомунікацій в рамках виконання науково-дослідних робіт «Дослідження науково-технічних засад створення нового покоління програмно-визначуваних телекомунікаційних радіосистем» (д/б 2007) та «Системи зв’язку безпілотних апаратів з підвищеною стійкістю до впливу навмисних та ненавмисних завад» (д/б 2308). В процесі досліджень були розроблені програмні модулі на мові С та допоміжні програми на мові phyton, які були використані для тестування SDR трансиверів AD9361 (модуль HSMC ARRadio Daughter Card встановлений на Terasic DE10 — Standard) та AD9363 (вбудований в ADALM-PLUTO). Текст розроблених програм приведено в додатках.

З програмою вичитки та збереження прочитаних даних за допомогою libiio можна ознайомитись в додатку А.2. Варто відмітити специфіку роботи з SDR – великий потік даних потребує обережності в обробці та збереженні даних. В даному випадку, окрім стандартного збереження в файл, було знайдено доцільним надати можливість переадресувати дані у вихідний потік, побудувати канал між даним потоком і ПК, і зберігати дані безпосередньо на ПК.

З програмою відправлення даних за допомогою libiio можна ознайомитись у додатку А.3. Обидві вищезазначені програми – програма запису та програма вичитки даних – використовують у роботі спільний файл-помічник, з яким можна ознайомитись у додатку А.4.

Для апробації роботи обох програм вони були об’єднані у спільний комплекс і запущені одночасно. Програма для передачі даних передавала дані поки програма для вичитки даних їх вичитувала. Дані були переадресовані у вихідний потік, був встановлений канал між цим вихідним потоком і ПК. Після цього, дані були збережені і проаналізовані. Для спрощення аналізу отриманих даних, була написана програма на python3 з якою можна ознайомитись у Додатку А.5, вона візуалізує отримані дані. З візуалізацією можна ознайомитись на рисунку 3.6.

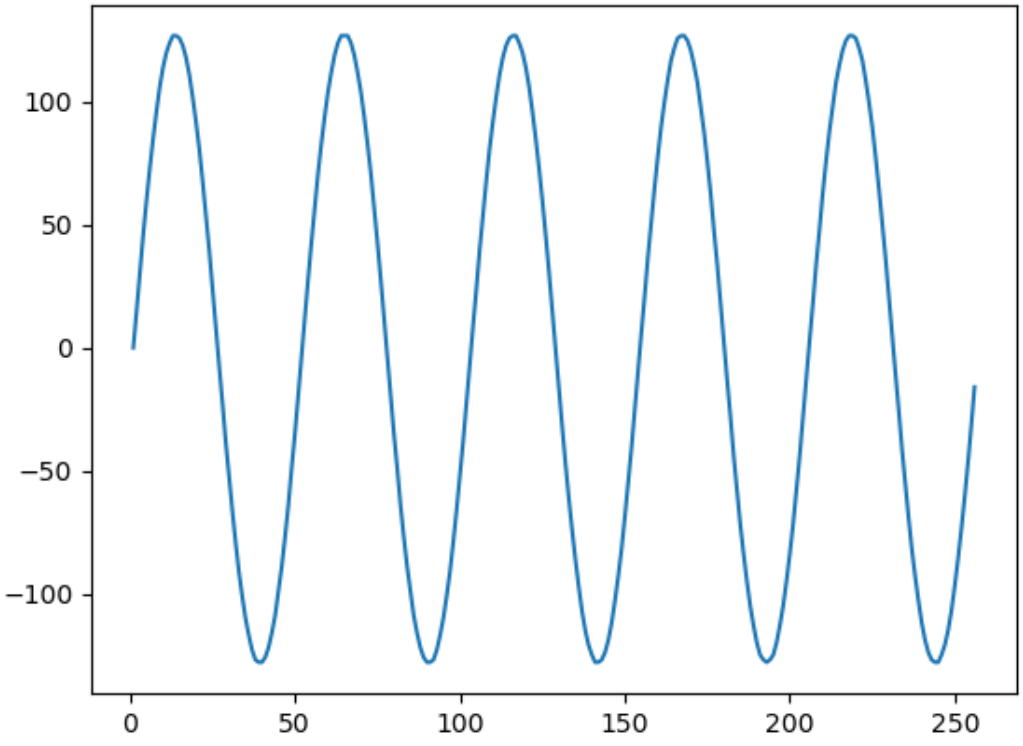

Рисунок 3.6 – Візуалізація отриманих даних.

1.14 Висновки

Індустріальна підсистема вводу/виводу (IIO Subsystem) призначена для підтримки пристроїв, які в деякому сенсі є аналогами аналогово-цифрових або цифро- аналогових перетворювачів (АЦП, ЦАП). Бібліотека libiio - це бібліотека, розроблена Analog Devices для полегшення розробки програмного забезпечення і сполучення (інтерфейсу) з Linux Industrial I/O (IIO) пристроями. Бібліотека складається з одного API високого рівня та декількох програм. Незважаючи на те, що libiio була розроблена в основному компанією Analog Devices Inc., вона є активною бібліотекою з відкритим кодом. Бібліотека випускається під GNU Lesser General Public License (LGPL), що дозволяє кожному користуватися бібліотекою на будь-якому постачальнику процесора / FPGA / SoC.

GNU Radio - це безкоштовний набір інструментів з відкритим кодом для розробки програмного забезпечення, який надає блоки обробки сигналів для реалізації SDR. GNU Radio широко використовується в любительських, академічних та комерційних середовищах для підтримки досліджень бездротового зв'язку та радіосистем. За рахунок «блокової» побудови та наявності GUI (Graphical User Interface), GNU Radio найліпше підходить для прототипування.

Simulink надає можливість спроектувати та змоделювати свою систему в Simulink перед тим, як переходити на апаратне забезпечення, надаючи змогу досліджувати та впроваджувати проекти без необхідності писати код C, C ++ або HDL. З іншого боку, робота з Simulink потребує вузьконаправлених спеціалізованих знань та навичок. Використання MATLAB та Simulink разом надає поєднання текстового та графічного програмування для проектування системи в середовищі імітації.

Взаємодія MATLAB та Simulink надає можливість безпосередньо використовувати тисячі алгоритмів, які вже є в MATLAB – можна додати свій код MATLAB у блок Simulink або діаграму Stateflow. MATLAB / Simulink надає фантастичні можливості та дуже добре підходить як для прототипування, так і реалізації повномасштабного проекту. Нажаль, ціна за MATLAB / Simulink у базовій комплектації починається від $5400

(MATLAB - $2150, Simulink - $3250) і в залежності від комплектації може досягати більше 10 тис. доларів на одного розробника.

КАК МОЖНО УПОМЯНУТЬ ПРАКТИКУ В ВЫВОДАХ?

РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

В даному розділі викладено маркетинговий аналіз перспектив реалізації комунікаційної системи на *???* короткій дистанції в умовах відсутності стільникового покриття.

1.15 Опис ідеї проекту

Проект направлений на створення програмного забезпечення на основі існуючого пристрою для забезпечення відео та аудіо комунікації в умовах відсутності стільникового покриття з використанням технології SDR SoC.

Таблиця 4.1

Таблиця 4.1. Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Пристрій який дає змогу спілкуватися по відео та аудіо зв’язку в умовах відсутності стільникового покриття	Мандрівки будь-якого призначення у віддалених локаціях (напр. туризм або наукове дослідження)	Користувачі зможуть спілкуватися між собою по відео зв’язку в умовах повної відсутності стільникового покриття, обмінюватися файлами та даними на відстані

1.16 Технологічний аудит ідеї проекту

Технологічна здійсненність ідеї проекту представлена у вигляді таблиці 4.2.

Таблиця 4.2

Таблиця 4.2. Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології реалізації	Наявність технологій	Доступність технологій
	Відео комунікація SDR SoC	Bare metal	Altera SoC EDS	Доступна, платна
1		OS	Стек програм Matlab/Simulink	Доступні, платні
2		OS	Програма GNU Radio	Доступна, безкоштовна
3		OS	Бібліотека libiio для C	Доступна, безкоштовна
4		OS	Бібліотека libiio для Python3	Доступна, безкоштовна
Обрана технологія реалізації POC: OS, а саме GNU Radio				
Обрана технологія реалізації проекту: OS, а саме мова програмування C				

Для реалізації проекту доступні два принципово різних рішення: Bare metal та OS. Використання Bare metal у стартап проектах виправдано тільки за умови наявності зацікавлених спеціалістів у роботі з Bare metal SDR SoC або наявності належного фінансування – розробка Bare metal дорога та потребує вузькоспрямованих спеціалістів. Розробка OS рішень, в

свою чергу, потребує менш спеціалізованих знань і надає можливість для використання безкоштовних продуктів.

Для створення прототипу продукту було обрано технологію GNU Radio. GNU Radio це проста у використанні програма, за допомогою якої розробка прототипу може відбутися достатньо швидко. В подальшому, для реалізації продукту буде використана інша технологія - мова програмування C, а саме libiio для C. Розробка на C займає більше часу і зусиль порівняно з GNU Radio, але дозволяє отримати більш продуктивну оптимізовану програму. Також, існуюча інтеграція між GNU Radio та C дозволить м’яко перейти від одного рішення до іншого, заміняючи модулі GNU Radio власними, оптимізованими модулями C коду.

1.17 Аналіз ринкових можливостей запуску стартап-проекту

При дослідженні ринкових можливостей, в першу чергу проведений аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку. Дані наведені у таблиці 4.3 нижче.

Таблиця 4.3

Таблиця 4.3. Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	4
2	Загальний обсяг продаж, грн/ум.од	?
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає

6	Середня норма рентабельності в галузі або по ринку,%	?

Характеристика потенційних клієнтів стартап-проекту представлена у вигляді таблиці 4.4.

Таблиця 4.4

Таблиця 4.4. Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Відео та аудіо зв'язок в умовах відсутності стільникового покриття	Туристи	Вимоги до якості та швидкості передачі даних - середні	• Передача відео та аудіо даних на *???* відстані • Стабільність у роботі ПЗ та АЗ • Простота у використанні • Відсутність необхідності самостійно займатись настройкою продукту
2		Науковці	Вимоги до якості та швидкості передачі даних - високі	

Проаналізовані фактори загроз (Таблиця 4.5) і проведений ступеневий аналіз конкуренції на ринку (Таблиця 4.6).

Таблиця 4.5

Таблиця 4.5. Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Наявність кваліфікованих кадрів	Потрібні люди з досвідом роботи в розробці систем SDR SoC	Пошук кваліфікованого персоналу
2	Наявність апаратного забезпечення за вигідною ціною	Вартість продукту сильно залежить від вартості АЗ	Укладання договорів з комерційними організаціями

Таблиця 4.6

Таблиця 4.6. Ступеневий аналіз конкуренції на ринку

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Тип конкуренції: олігополія	На ринку представлені декілька компаній, що поставляють подібні пристрої	Акцентування переваг продукту
2	Рівень конкурентної боротьби: національний / інтернаціональний	Першим етапом є боротьба за ринок України з подальшим виходом на ринки інших країн	Маркетингова компанія в першу чергу орієнтована на захоплення місцевого ринку
3	Галузева ознака: внутрішньогалузева	Економічна боротьба між різними товаровиробниками, які діють в одній галузі економіки, виробляють і	Слідкувати за новітніми технологіями, та розробками конкурентів

		реалізують однакові товари та технології, що задовольняють одну й ту саму потребу	
4	Конкуренція за видами товарів: товарно-видова	Конкуренція відбувається між послугами одного виду. За такої конкуренції значення набуває марка товару	Постійна робота над забезпеченням високого рівня іміджу компанії

Були обґрунтовані фактори конкурентоспроможності (Таблиця 4.7)

Таблиця 4.7

Таблиця 4.7. Обґрунтування факторів конкурентоспроможності

№ п/п	Фактори конкурентоспроможності	Обґрунтування
1	Ступінь задоволення потреб користувача	Необхідно надати високоефектий та зручний пристрій
2	Якість розробки з точки зору оптимальності показників надійності	Швидкодія та стабільність
3	Економічний	Середня ціна продукту

За визначеними факторами конкурентоспроможності проведений аналіз сильних та слабких сторін стартап-проекту (Таблиця 4.8).

Таблиця 4.8

Таблиця 4.8. Порівняльний аналіз сильних та слабких сторін продукту

№ п/п	Фактори конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з власним продукту						
			-3	-2	-1	0	1	2	3
1	Ступінь задоволення потреб користувача	20							+

2	Якість розробки з точки зору оптимальності показників надійності	20							+
3	Економічний	14					+		

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів що було виконано у таблиці 4.9.

Таблиця 4.9

Таблиця 4.9. Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Туристи	Готові	Високий	Середня	Середня
2	Дослідники	Готові	Середній	Низька	Середня

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (Таблиця 4.10).

Таблиця 4.10

Таблиця 4.10. Визначення базової стратегії розвитку

№ п/п	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Динамічний розвиток завдяки висвітленню	Дистанція можливого дзвінку.	Стратегія диференціації.

	унікальних характеристик продукту.		
2	Надання товару важливих з точки зору споживача відмітних властивостей, які роблять товар відмінним від товарів конкурентів.	Формування попиту якісною, надійною та простою роботою пристрою.	Стратегія диференціації.

В результаті аналізу була обрана стратегія диференціації. Наступним кроком був вибір стратегії конкурентної поведінки (Таблиця 4.11) і визначення стратегії позиціонування (Таблиця 4.12).

Таблиця 4.11

Таблиця 4.11. Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект "першопрохідцем" на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Ні.	І те і інше.	Ні.	Стратегія наслідування лідеру за для економії фінансових ресурсів.

Таблиця 4.12

Таблиця 4.12. Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап- проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Якість та простота роботи	Диференціації	Середня ціна, висока якість, надійність та простота у використанні	Швидко, просто, надійно

1.18 Розроблення маркетингової програми стартап-проекту

Маркетингова програма - це намічений для планомірного здійснення, об'єднаний єдиною метою та залежний від певних строків комплекс взаємопов'язаних завдань і адресних заходів соціального, економічного, науково-технічного, виробничого, організаційного характеру з визначенням ресурсів, що використовуються, а також джерел одержання цих ресурсів. Основну увагу слід приділяти вибору, значенню та формі інструментів маркетингу, їх об'єднанню в найбільш оптимальний з погляду визначеної мети комплекс, а також розподілу фінансових ресурсів у межах бюджетування маркетингу.

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару що представлене у таблиці 4.13.

Таблиця 4.13

Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує технологія	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Відео та аудіо зв’язок в умовах відсутності стільникового покриття	Надійність, ефективність, простота у роботі з пристроєм	Порівняно великі відстані що охоплюються

Надалі розробляють трирівневу маркетингову модель товару: уточнюють ідею продукту та/або послуги, його фізичні складові, особливості процесу його надання (Таблиця 4.14).

Таблиця 4.14

Опис трьох рівнів моделі товару

№ п/п	Рівні товару	Сутність та складові
1	Товар за задумом	Пристрій для відео та аудіо зв’язку на ??? відстанях
2	Товар у реальному виконанні	Властивості: доступність, зручність, надійність
		Товар представляє собою пристрій та програмне забезпечення реалізоване з використанням GNU Radio (прототип) та C (кінцевий продукт)
3	Товар із підкріпленням	Гарантія
		Настройка пристрою після покупки за необхідністю

1.19 Висновки

В даному розділі був проведений маркетинговий аналіз перспектив реалізації SoC SDR пристрою для відео та аудіо зв’язку на *???* дистанції в умовах відсутності стільникового зв’язку та проведене оцінювання можливостей її ринкового впровадження.

Проведено аналіз потенційних техніко-економічних переваг порівняно з пропозиціями конкурентів. Визначено, що технологія переважає найближчих конкурентів в економічних та технологічних характеристиках.

Досліджено ринкові загрози та ринкові можливості, які складають на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Розроблено ринкову стратегію проекту та маркетингову програму стартап-проекту.

Проведений аналіз підтверджує, що подальша імплементація проекту є доцільною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. J. Mitola, “Software Radios Survey, Critical Evaluation and Future Directions,” in Telesystems Conference, NTC. National, (Washington, DC, USA), pp. 13/15 –13/23, IEEE, 1992.
2. T. Ulversoy, “Software defined radio: Challenges and opportunities,” Communications Surveys & Tutorials, IEEE, vol. 12, no. 4, pp. 531–550, 2010.
3. M. Palkovic, P. Raghavan, M. Li, A. Dejonghe, L. Van der Perre, and F. Catthoor, “Future Software-Defined Radio Platforms and Mapping Flows,” Signal Processing Magazine, IEEE, vol. 27, no. 2, pp. 22–33, 2010.
4. https://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf
5. https://www.ni.com/ru-ru/innovations/white-papers/17/software-defined-radio--past--present--and-future.html
6. José Raúl Machado-Fernández “Software Defined Radio: Basic Principles and Applications”, 2014:
7. Software Communications Architecture Specification. FINAL / 15 May 2006 Version 2.2.2, - Режим доступу: http://www.public.navy.mil/jtnc/SCA/SCAv2_2_2/SCA_version_2_2_2.pdf.
8. Joint Tactical Radio Systems. - Режим доступу: http://jtrs.army.mil
9. T. Newman; G. J. Minden “A Software Defined Radio Architecture Model to Develop Radio Modem Component Classifications”, First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005. (IEEE Xplore Digital Library, DOI:10.1109/DYSPAN.2005.1542674), 2005. – P. 1-4
10. Kaidenko M.M., Roskoshnyi D.V. (2019) Software Defined Radio in Communications. In: Ilchenko M., Uryvsky L., Globa L. (eds) Advances in Information and Communication Technologies. UKRMICO 2018. Lecture Notes in Electrical Engineering, vol 560. Springer, Cham DOI

https://doi.org/10.1007/978-3-030-16770-7_11 Print ISBN 978-3-030-16769-1
11. M. Ilchenko, M. Kaidenko, S. Kravchuk, V. Khytrovskyy, “Compact troposcatter station for transhorizon communication”, Proceedings of the 2017 International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo) 11-15 Sept. 2017 Year, Odessa, Ukraine. - IEEE Conference Publications (IEEE Xplore Digital Library, DOI: 10.1109/UkrMiCo.2017.8095364), 2017. – P. 1-4
12. S. Kravchuk, M. Kaidenko, “Features of creation of modem equipment for the new generation compact troposcatter stations”, Proc. of the International Scientific Conference "RadioElectronics & InfoCommunications" (UkrMiCo’2016), 11-16 September 2016, Kyiv, Ukraine. – (IEEE Digital Library), 2016.– P. 365-368.
13. Ilchenko M., Kravchuk S., Kaydenko M. (2019) Combined Over-the-Horizon Communication Systems. In: Ilchenko M., Uryvsky L., Globa L. (eds) Advances in Information and Communication Technologies. UKRMICO 2018. Lecture Notes in Electrical Engineering, vol 560. Springer, Cham DOI https://doi.org/10.1007/978-3-030-16770-7_6 Print ISBN 978-3-030-16769-1
14. N. Kaydenko, “Adaptive modulation and coding in a broadband wireless access systems”, 23rd International Crimean Conference on “Microwave and Telecommunication Technology”, CriMico'2013. Sevastopol, on September 8-13, 2013. - P. 275-276.
15. https://www.altera.com/content/dam/altera/www/global/en_US/pdfs/literature/wp/wp-01198-fpga-software-debug-soc.pdf
16. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_soc_eds.pdf
17. Кайденко В.М., Кайденко М.М., Крылач О.Ф., Роскошний Д.В. «Организация взаимодействия между HPS и FPGA на базе SoC технологий при создании телекоммуникационных устройств» // X Міжнародна науково-технічна конференція "Проблеми телекомунікацій" ПТ-2016: Збірник матеріалів конференції. К.: НТУУ "КПІ", 2016. - с. 254-256.
18. http://www.eejournal.com/files/8713/9837/5380/WP-14033_-_Altera_SoC_Design_Advantages_1.pdf
19. https://microtoolsinc.com/papers/why-linux/
20. https://semiengineering.com/bare-metal-programming/

21. https://www.intel.com/content/www/us/en/programmable/documentation/lro1424280108409.html
22. https://www.intel.com/content/www/us/en/programmable/buy/design-software.html
23. https://www.forbes.com/sites/greatspeculations/2017/05/24/blackberrys-qnx-faces-significant-threats-in-the-auto-market/?sh=1f7ee5421313
24. https://www.mathworks.com/pricing-licensing.html?intendeduse=comm&s_tid=htb_learn_gtwy_cta1
25. https://ts.data61.csiro.au/publications/papers/Engel_HKPR_04.pdf
26. https://blackberry.qnx.com/en/rtos/what-is-real-time-operating-system/?utm_source=google&utm_medium=cpc&gclid=Cj0KCQjwqrb7BRDlARIsACwGad62DD2WWHKfP02KOOpUvj_d85vx6PqVYjxTnvO0if6N2GSEtOWV7oIaAtn7EALw_wcB
27. https://gcc.gnu.org/
28. https://stackoverflow.com/questions/25871579/what-is-the-difference-between-rtos-and-embedded-linux
29. https://www.kernel.org/doc/html/v4.12/driver-api/iio/index.html
30. https://wiki.analog.com/resources/tools-software/linux-software/libiio
31. https://github.com/analogdevicesinc/libiio
32. Кайденко М.М. «Використання модельно-орієнтованого проектування для програмно-визначуваних радіосистем» // XI Міжнародна науково-технічна конференція "Проблеми телекомунікацій" ПТ-2017: Збірник матеріалів конференції. К.: КПІ ім. Ігоря Сікорського, 2017 с. 181-183
33. IIO System Object. Analog Devices Wiki. - Режим доступу: https://wiki.analog.com/ resources/tools-software/linux-software/libiio/clients/matlab_simulink?s%5b%5d =libiio
34. http://www.mathworks.com/help/dsp/gs/what-are-system-objects.html
35. Бондаренко К.С., Кайденко М.М., Роскошний Д.В. Розробка програмно-визначуваних радіосистем на базі SoC в середовищі Matlab/Simulink // XIII Міжнародна науково-технічна конференція "Перспективи телекомунікацій" ПТ-2019: Збірник матеріалів конференції. К.: КПІ ім. Ігоря Сікорського, 2019. - с. 182-185
36. https://wiki.gnuradio.org/index.php/What_is_GNU_Radio%3F

ДОДАТОК А

А.1 Текст програми “Hello world”

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    puts(“Hello world!”);
    return EXIT_SUCCESS;
}
```

А.2. Вичитка даних за допомоги libiio

Main.c

```
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <assert.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <math.h>

#ifdef __APPLE__
#include <iio/iio.h>
#else
#include <iio.h>
#endif

/*********************************************
 * the "functions.c" document holds
 * all the functions that can be found in the example
code
```

```
*****************************************/

#include "functions.c"


/*****************************************
 * defines here were created as
 * shortcuts to change the code fast
*****************************************/


/*****************************************
 * STREAMING
 * streaming is useful in case you don't want to
overload the device with
 * significant amounts of data
 * the device does go slower over time,
 * since in about 3 iterations over while cycle
 * you'll get about 6 000 000 values
 * (3 000 000 for I channel, the same amount for Q
channel)
 * however, if you will stream to the terminal,
instead of streaming to
 * your computer for example,
 * the program will work MUCH MUCH SLOWER
 * which kind of contradicts the main purpose of
streaming idea
 * you can use NETCAT/NCAT for streaming data between
 * the device and the computer
```

```
 * there is an article in wiki on how to do that
*****************************************/
// #define STREAMING 1

/*****************************************
 * REFRESH_SETTINGS_EACH_LOOP
 * refreshing settings each while loop kind of makes
sure
 * you read the values properly
 * however, there are much more settings that
influence
 * what you can get, so this option does not
guarantee
 * the result
*****************************************/
#define REFRESH_SETTINGS_EACH_LOOP 1

/*****************************************
 * SAVE_WHAT_IM_READING_FROM_DEVICE
 * saving data to file might
 * get pretty slow, since you will be getting
 * a substantial amount of data
 * DO MAKE SURE to not run the program for
 * more that 10 iterations of while loop
 * (I prefer to count to 8 if there are no prints)
 * since the device's memory resources are limited
```

```
*****************************************/
#define SAVE_WHAT_IM_READING_FROM_DEVICE 1

//#define SAVE_ADDITIONAL_INFO 1


int main (int argc, char **argv)
{
    int buffer_size_in_samples = 1024*1024;

    #ifdef SAVE_WHAT_IM_READING_FROM_DEVICE
    FILE *f_from_device;
    f_from_device = fopen("saved_output.txt", "w");
    if (f_from_device == NULL)
    {
        printf("Error opening file!\n");
        shutdown();
    }
    #endif

    #ifdef SAVE_ADDITIONAL_INFO
    FILE *f_buf_size;
    f_buf_size = fopen("buf_size.txt", "w");
    if (f_buf_size == NULL)
    {
```

```
        printf("Error opening file!\n");
        shutdown();
    }
    #endif

    struct iio_device *rx;
    size_t nrx = 0;             // RX sample counter
    struct stream_cfg rxcfg;    // Stream
configurations

    // Listen to ctrl+c and assert
    signal(SIGINT, handle_sig);

    /*****************************************
     * Settings for an RX channel
     * it is IMPORTANT to make sure they match
     * the ones set for the TX channel!
    *****************************************/
    rxcfg.bw_hz = MHZ(18);          // 2 MHz rf
bandwidth
    rxcfg.fs_hz = MHZ(8);           // 2.5 MS/s rx
sample rate
    rxcfg.lo_hz = GHZ(2.5);         // 2.5 GHz rf
frequency
    rxcfg.rfport = "A_BALANCED";    // port A (select
for rf freq.)
```

```
    printf("* Acquiring IIO context\n");

    assert((ctx = iio_create_default_context()) &&
"No context");

    assert(iio_context_get_devices_count(ctx) > 0 &&
"No devices");


    assert(get_ad9361_stream_dev(ctx, RX, &rx) && "No
rx dev found"); // Acquiring AD9361 streaming devices

    assert(cfg_ad9361_streaming_ch(ctx, &rxcfg, RX,
0) && "RX port 0 not found"); // Configuring AD9361
for streaming


    assert(get_ad9361_stream_ch(ctx, RX, rx, 0,
&rx0_i) && "RX chan i not found"); // Initializing
AD9361 IIO streaming channels

    assert(get_ad9361_stream_ch(ctx, RX, rx, 1,
&rx0_q) && "RX chan q not found");


    iio_channel_enable(rx0_i); // Enabling IIO
streaming channels

    iio_channel_enable(rx0_q);


    rxbuf = iio_device_create_buffer(rx,
buffer_size_in_samples, false); // Creating non-
cyclic IIO buffers with 1 MiS

    if (!rxbuf) {

        perror("Could not create RX buffer");

        shutdown();
```

```
    }


    printf("* Starting IO streaming (press CTRL+C to
cancel)\n");


    while (!stop)
    {
        ssize_t nbytes_rx, nbytes_tx;
        void *p_dat, *p_end;
        ptrdiff_t p_inc;


        #ifdef REFRESH_SETTINGS_EACH_LOOP
        /*****************************************
         * Settings for an RX channel
         * make sure they match the ones you set
before
         * and the ones set to TX channel!
        *****************************************/
        rxcfg.bw_hz = MHZ(18);          // 18 MHz rf
bandwidth
        rxcfg.fs_hz = MHZ(8);           // 8 MS/s rx
sample rate
        rxcfg.lo_hz = GHZ(2.5);         // 2.5 GHz rf
frequency
        rxcfg.rfport = "A_BALANCED";    // port A
(select for rf freq.)
        #endif
```

```
        nbytes_rx = iio_buffer_refill(rxbuf); //
Refill RX buffer

        if (nbytes_rx < 0) { printf("Error refilling
buf %d\n",(int) nbytes_rx); shutdown(); }


        /*****************************************

         * Option for memcpy

         * here lies the option to just memcpy things
instead of putting them one by one

         * really doesn't matter how you do it, feel
free to experiment

         * memcpy the memory between p_dat and p_end

        *****************************************/

        // READ: Get pointers to RX buf and read IQ
from RX buf port 0

        p_dat = iio_buffer_first(rxbuf, rx0_i);

        p_inc = iio_buffer_step(rxbuf);

        p_end = iio_buffer_end(rxbuf);


        #ifdef SAVE_ADDITIONAL_INFO

        /*****************************************

         *
SAVE_ADDITIONAL_INFO_____________________________

         * test prints just to make sure that the
memory chunk has the same size
```

```
         * the addresses will differ, the DISTANCE
between them shall not
         * if the distance and/or inc differs -
things are not going well!
         * be super alerted, check channels and
everything related
        ******************************************/
        fprintf(f_buf_size, "rx buf start %p\n",
p_dat);
        fprintf(f_buf_size, "rx buf inc   %p\n",
p_inc);
        fprintf(f_buf_size, "rx buf end   %p\n",
p_end);
        fprintf(f_buf_size,
"_____________________________\n");
        #endif


        for (; p_dat < p_end; p_dat += p_inc)
        {

/*****************************************
             * Reading data
             * getting pointer to real data here, and
then just iterating over it

*****************************************/
            const int16_t i = ((int16_t*)p_dat)[0];
// Real (I)
```

```
            const int16_t q = ((int16_t*)p_dat)[1];
// Imag (Q)


            #ifdef STREAMING

/*****************************************
             * STREAMING
             * as you already read, do not stream to
terminal
             * catch the output with something

*****************************************/
            printf("%d,", i);
            printf("%d\n", q);
            #endif


            #ifdef SAVE_WHAT_IM_READING_FROM_DEVICE
            fprintf(f_from_device, "%d,", i);
            fprintf(f_from_device, "%d\n", q);
            #endif
        }
        #ifdef SAVE_WHAT_IM_READING_FROM_DEVICE
        fflush(f_from_device);
        #endif
    }
    shutdown();
```

```
    return 0;
}
```

А.3 Запис даних за допомогою libiio

Main.c

```
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <assert.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <math.h>
#include <iio.h>

#include "functions.c"

#define SINE_FILE_LEN 1024
```

```
// #define REFRESH_SETTINGS_EACH_LOOP 1



int main (int argc, char **argv)
{
	/*************************************
	 * Getting ready to read from file
	*************************************/
	FILE *f_generated_parsed_sine;
	f_generated_parsed_sine = fopen("parsed_sine", "r");
    if (f_generated_parsed_sine == NULL)
	{
		printf("Error opening file!\n");
		shutdown();
	}

	/*************************************
	 * Setting up TX channel
	*************************************/
	struct iio_device *tx;
	size_t ntx = 0;                                 // TX
sample counter
	struct stream_cfg txcfg;        // Stream configurations

	signal(SIGINT, handle_sig);             // Listen to
ctrl+c and assert

	// TX stream config
```

```
	txcfg.bw_hz = MHZ(18);                 // 1.5 MHz rf
bandwidth

	txcfg.fs_hz = MHZ(8);            // 2.5 MS/s tx sample
rate

	txcfg.lo_hz = GHZ(2.5);          // 2.5 GHz rf frequency

	txcfg.rfport = "A";                    // port A (select
for rf freq.)


	printf("* Acquiring IIO context\n");

	assert((ctx = iio_create_default_context()) && "No
context");

	assert(iio_context_get_devices_count(ctx) > 0 && "No
devices");


	assert(get_ad9361_stream_dev(ctx, TX, &tx) && "No tx dev
found"); // Acquiring AD9361 streaming devices

	assert(cfg_ad9361_streaming_ch(ctx, &txcfg, TX, 0) &&
"TX port 0 not found"); // Configuring AD9361 for streaming


	assert(get_ad9361_stream_ch(ctx, TX, tx, 0, &tx0_i) &&
"TX chan i not found"); // Initializing AD9361 IIO streaming
channels

	assert(get_ad9361_stream_ch(ctx, TX, tx, 1, &tx0_q) &&
"TX chan q not found");


	iio_channel_enable(tx0_i); // Enabling IIO streaming
channels

	iio_channel_enable(tx0_q);



	/************************************
```

```
        * Setting up TX buffer
        ***************************************/
        int buffer_size_in_samples = SINE_FILE_LEN;


        txbuf = iio_device_create_buffer(tx,
buffer_size_in_samples, false); // Creating non-cyclic IIO
buffers with 1 MiS
        if (!txbuf) {
                perror("Could not create TX buffer");
                shutdown();
        }


        /***************************************
        * Saving data from file to array
        ***************************************/
    int16_t sine_arr[SINE_FILE_LEN] = {0};
    int i = 0;
    char buf[20];           // 20 is a random value that's big
enough to contain "minus" + "char"
    int temp_int = 0;
        int sine_arr_size = (SINE_FILE_LEN * sizeof(int16_t));


        while (fgets(buf, sizeof buf, f_generated_parsed_sine)
!= NULL)
    {
          if (feof(f_generated_parsed_sine)) { printf("\nEnd of
file reached"); }


          temp_int = atoi(buf);
```

```
        sine_arr[i] = (int16_t)temp_int << 4;
        i++;
    }
    fclose(f_generated_parsed_sine);


    while (!stop)
    {
        ssize_t nbytes_tx;
        void *p_dat;

        #ifdef REFRESH_SETTINGS_EACH_LOOP
            // TX stream config
            txcfg.bw_hz = MHZ(18);                // 18
MHz rf bandwidth
            txcfg.fs_hz = MHZ(8);           // 8 MS/s
tx sample rate
            txcfg.lo_hz = GHZ(2.5);         // 2.5 GHz
rf frequency
            txcfg.rfport = "A";                   //
port A (select for rf freq.)
        #endif

        p_dat = iio_buffer_first(txbuf, tx0_i);

        /*************************************
        * Copying data to buffer and pushing it
        *************************************/
        memcpy(p_dat, sine_arr, sine_arr_size);
```

```
            /// NEEDS TO BE EXPLAINED

            memcpy(p_dat + sine_arr_size, sine_arr,
sine_arr_size);



            nbytes_tx = iio_buffer_push(txbuf); // Schedule
TX buffer

            if (nbytes_tx < 0) { printf("Error pushing buf
%d\n", (int) nbytes_tx); shutdown(); }

      }



      shutdown();

      return 0;

}
```

A.4 Допоміжні функції для запису та вичитки даних за використанням libiio

Functions.c

```
#include <stdbool.h>

#include <stdint.h>

#include <string.h>

#include <assert.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <unistd.h>

#include <math.h>

#include <iio.h>
```

```
/* helper macros */
#define MHZ(x) ((long long)(x*1000000.0 + .5))
#define GHZ(x) ((long long)(x*1000000000.0 + .5))

/* RX is input, TX is output */
enum iodev { RX, TX };

/* common RX and TX streaming params */
struct stream_cfg {
	long long bw_hz; // Analog banwidth in Hz
	long long fs_hz; // Baseband sample rate in Hz
	long long lo_hz; // Local oscillator frequency in Hz
	const char* rfport; // Port name
};

/* static scratch mem for strings */
char tmpstr[64];

/* IIO structs required for streaming */
struct iio_context *ctx   = NULL;
struct iio_channel *rx0_i = NULL;
struct iio_channel *rx0_q = NULL;
struct iio_channel *tx0_i = NULL;
struct iio_channel *tx0_q = NULL;
struct iio_buffer  *rxbuf = NULL;
```

```
struct iio_buffer  *txbuf = NULL;

bool stop;



/* cleanup and exit */
void shutdown()
{
	printf("* Destroying buffers\n");
	if (rxbuf) { iio_buffer_destroy(rxbuf); }
	if (txbuf) { iio_buffer_destroy(txbuf); }

	printf("* Disabling streaming channels\n");
	if (rx0_i) { iio_channel_disable(rx0_i); }
	if (rx0_q) { iio_channel_disable(rx0_q); }
	if (tx0_i) { iio_channel_disable(tx0_i); }
	if (tx0_q) { iio_channel_disable(tx0_q); }

	printf("* Destroying context\n");
	if (ctx) { iio_context_destroy(ctx); }
	exit(0);
}



void handle_sig(int sig)
{
	printf("Waiting for process to finish...\n");
```

```
	stop = true;
}

/* check return value of attr_write function */
void errchk(int v, const char* what) {
	 if (v < 0) { fprintf(stderr, "Error %d writing to
channel \"%s\"\nvalue may not be supported.\n", v, what);
shutdown(); }
}

/* write attribute: long long int */
void wr_ch_lli(struct iio_channel *chn, const char* what, long
long val)
{
	errchk(iio_channel_attr_write_longlong(chn, what, val),
what);
}

/* write attribute: string */
void wr_ch_str(struct iio_channel *chn, const char* what,
const char* str)
{
	errchk(iio_channel_attr_write(chn, what, str), what);
}

/* helper function generating channel names */
char* get_ch_name(const char* type, int id)
{
	snprintf(tmpstr, sizeof(tmpstr), "%s%d", type, id);
```

```
	return tmpstr;
}

/* returns ad9361 phy device */
struct iio_device* get_ad9361_phy(struct iio_context *ctx)
{
	struct iio_device *dev =  iio_context_find_device(ctx, "ad9361-phy");
	assert(dev && "No ad9361-phy found");
	return dev;
}

/* finds AD9361 streaming IIO devices */
bool get_ad9361_stream_dev(struct iio_context *ctx, enum iodev d, struct iio_device **dev)
{
	switch (d) {
	case TX: *dev = iio_context_find_device(ctx, "cf-ad9361-dds-core-lpc"); return *dev != NULL;
	case RX: *dev = iio_context_find_device(ctx, "cf-ad9361-lpc");  return *dev != NULL;
	default: assert(0); return false;
	}
}

/* finds AD9361 streaming IIO channels */
bool get_ad9361_stream_ch(struct iio_context *ctx, enum iodev d, struct iio_device *dev, int chid, struct iio_channel **chn)
{
```

```
	*chn = iio_device_find_channel(dev,
get_ch_name("voltage", chid), d == TX);

	if (!*chn)

		*chn = iio_device_find_channel(dev,
get_ch_name("altvoltage", chid), d == TX);

	return *chn != NULL;

}


/* finds AD9361 phy IIO configuration channel with id chid */

bool get_phy_chan(struct iio_context *ctx, enum iodev d, int
chid, struct iio_channel **chn)

{

	switch (d) {

	case RX: *chn =
iio_device_find_channel(get_ad9361_phy(ctx),
get_ch_name("voltage", chid), false); return *chn != NULL;

	case TX: *chn =
iio_device_find_channel(get_ad9361_phy(ctx),
get_ch_name("voltage", chid), true);  return *chn != NULL;

	default: assert(0); return false;

	}

}


/* finds AD9361 local oscillator IIO configuration channels */

bool get_lo_chan(struct iio_context *ctx, enum iodev d, struct
iio_channel **chn)

{

	switch (d) {

	 // LO chan is always output, i.e. true
```

```
	case RX: *chn =
iio_device_find_channel(get_ad9361_phy(ctx),
get_ch_name("altvoltage", 0), true); return *chn != NULL;

	case TX: *chn =
iio_device_find_channel(get_ad9361_phy(ctx),
get_ch_name("altvoltage", 1), true); return *chn != NULL;

	default: assert(0); return false;

	}

}


/* applies streaming configuration through IIO */

bool cfg_ad9361_streaming_ch(struct iio_context *ctx, struct
stream_cfg *cfg, enum iodev type, int chid)

{

	struct iio_channel *chn = NULL;


	// Configure phy and lo channels

	printf("* Acquiring AD9361 phy channel %d\n", chid);

	if (!get_phy_chan(ctx, type, chid, &chn)) {  return
false; }

	wr_ch_str(chn, "rf_port_select",     cfg->rfport);

	wr_ch_lli(chn, "rf_bandwidth",       cfg->bw_hz);

	wr_ch_lli(chn, "sampling_frequency", cfg->fs_hz);


	// Configure LO channel

	printf("* Acquiring AD9361 %s lo channel\n", type == TX
? "TX" : "RX");

	if (!get_lo_chan(ctx, type, &chn)) { return false; }

	wr_ch_lli(chn, "frequency", cfg->lo_hz);

	return true;
```

```
}
```

A.5 Візуалізація даних що були вичитані використовуючи libiio

main.py

```
#!/usr/bin/python3


import matplotlib.pyplot as plt

import numpy as np


def print_help():


print("=============================================
================================")

    print("Программа для визуализации данных
полученных от АД")

    print("\t У данной программы слабый
вычислительный ресурс. Она долго запускается, а при
слишком большом обьеме входных данных может не
запуститься совсем. Для правильной работы программы,
запись данных с АД должна вестись всего несколько
секунд")

    print("Закрыть приложение: закройте окно с
рисунком и терминал")

    print("Возможные настройки: ")

    print("\t 1) Название файла с данными")

    print("\t 2) Отрисовывать один или два канала")

    print("Изменить настройки: откройте main.py как
текстовый документ и инструкцию")
```

```
    print()

print("==========================================
==================================")
    return 0



print_help()

# ==========================================
# Первая настройка - название файла с данными

file_name = "what_im_sending.txt"
# ==========================================

input_file = open(file_name, "r")

i_channel = []
q_channel = []
x_axis = []
counter = 0;

for line in input_file:
    line_list = line.split(",")
    i_channel.append(int(line_list[0]))
```

```
    q_channel.append(int(line_list[1]))
    counter = counter + 1;
    x_axis.append(counter)

# ================================================
# Вторая настройка - количество каналов для отрисовки
# Важно: ОДНА из строк должна быть закомментирована
(# в начале строки), а ДРУГАЯ - нет

plt.plot(x_axis, i_channel)
#plt.plot(x_axis, i_channel, x_axis, q_channel)

# ================================================

plt.show()

input()
```