

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут телекомунікаційних систем  
Кафедра телекомунікацій**

До захисту допущено:

В.о.завідувача кафедри

\_\_\_\_\_ Сергій КРАВЧУК

«\_\_\_» \_\_\_\_\_ 2026 р.

**Дипломна робота**

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інженерія та програмування  
інфокомунікацій»**

**спеціальності 172 «Телекомунікації та радіотехніка»**

**на тему: «Дослідження особливостей застосування SDR трансиверів AD9361  
для реалізації MIMO системи»**

Виконав:

Студент IV курсу, групи ТЗ-21

Оришич Михайло Петрович \_\_\_\_\_

Керівник:

Старший викладач кафедри ТК НН ІТС

Кайденко Микола Миколайович \_\_\_\_\_

Рецензент:

Доцент кафедри ІТТ НН ІТС, к.т.н., доцент

Правило Валерій Володимирович \_\_\_\_\_

Засвідчую, що у цій дипломній роботі немає  
запозичень з праць інших авторів без  
відповідних посилань.

Студент \_\_\_\_\_

Київ – 2026 року

**Національний технічний університет України**  
**«Київський політехнічний інститут імені Ігоря Сікорського»**  
**Навчально-науковий інститут телекомунікаційних систем**  
**Кафедра телекомунікацій**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інженерія та програмування інфокомунікацій»

ЗАТВЕРДЖУЮ

В.о.завідувача кафедри

\_\_\_\_\_ Сергій КРАВЧУК

«\_\_\_» \_\_\_\_\_ 2026 р.

**ЗАВДАННЯ**

**на дипломну роботу студенту**

**Оришичу Михайлу Петровичу**

1. Тема роботи «Дослідження особливостей застосування SDR трансиверів AD9361 для реалізації MIMO системи», керівник роботи Кайденко Микола Миколайович, старший викладач кафедри ТК НН ІТС, затверджені наказом по університету від «26» травня 2026 р. № 1912-с.
2. Термін подання студентом роботи 10 червня 2026 р.
3. Вихідні дані до роботи: Плата розробки Terasic DE10-Standard з підключеним SDR трансивером Analog Devices AD9361, будова системи MIMO з використанням цифрового формування променів (beamforming).
4. Зміст роботи: Дослідження особливостей при реалізації систем MIMO з використанням цифрового формування променів (beamforming) на основі SDR трансивера Analog Devices AD9361 з фокусом на проблему синхронізації фаз каналів прийому з порівняльним аналізом різних методів вирішення даної проблеми та практичною реалізацією цих рішень на SDR трансивері.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)
  - Слайд 1: Титульний лист
  - Слайд 2: Мета роботи
  - Слайд 3: Актуальність роботи

- Слайд 4: Структура роботи
- Слайд 5: Метод крос-кореляції
- Слайд 6: Метод внутрішнього добутку
- Слайд 7: Метод частотного домену (перетворення Фур'є)
- Слайд 8: Опис проведених досліджень
- Слайд 9: Дослідження точності методу крос-кореляції
- Слайд 10: Дослідження точності методів при дії шуму
- Слайд 11: Практична реалізація
- Слайд 12: Дослідження швидкості методів
- Слайд 13: Практична цінність
- Слайд 14: Загальні висновки роботи

6. Дата видачі завдання 27 жовтня 2025 року

#### Календарний план

| № з/п | Назва етапів виконання дипломної роботи                      | Термін виконання етапів роботи | Примітка |
|-------|--------------------------------------------------------------|--------------------------------|----------|
| 1     | Отримання завдання на виконання роботи                       | 27.10.2025                     | Виконано |
| 2     | Ознайомлення з технічною літературою. Аналіз існуючих рішень | 27.10.2025 — 17.03.2026        | Виконано |
| 3     | Формулювання структури дослідження                           | 18.03 — 29.04.2026             | Виконано |
| 4     | Ознайомлення з обладнанням для виконання роботи              | 20.04 — 26.04.2026             | Виконано |
| 5     | Моделювання методів визначення різниці фази в MATLAB         | 26.04 — 24.05.2026             | Виконано |
| 6     | Реалізація практичної частини роботи                         | 25.05 — 27.05.2026             | Виконано |
| 7     | Оформлення пояснювальної записки                             | 28.05 — 31.05.2026             | Виконано |
| 8     | Підготовка презентації та матеріалів до захисту              | 01.06 — 07.06.2026             | Виконано |

Студент

Михайло ОРИШИЧ

Керівник

Микола КАЙДЕНКО

## РЕФЕРАТ

Дипломна робота викладена на 42 сторінках з 8 рисунками, 6 таблицями, 11 посиланнями на сторонні джерела та 3 додатками.

Метою роботи є визначити основні проблем реалізації радіосистем з використанням технологій MIMO на SDR платформах з трансивером Analog Devices AD9361 та проаналізувати рішення, які можна використати для вирішення даних проблем, з особливим фокусом на проблему синхронізації фаз між каналами приймача AD9361.

В даній роботі надано детальні описи принципу роботи, особливості реалізації та дослідження характеристик точності та швидкості для трьох методів визначення різниці фази між сигналами: на основі крос-кореляції, на основі внутрішнього добутку та на основі частотного домену (перетворення Фур'є).

Також в даній роботі описано та надано практичну реалізацію даних методів та методу корекції фази сигналу для SDR трансиверів на базі Analog Devices AD9361. Практична реалізація написана на мові програмування C з використанням бібліотек `libiio` та `FFTW` та поділена на три частини: бібліотеку з реалізаціями методів визначення різниці фази між сигналами, застосунок для коригування фази та застосунок, що практично визначає швидкість кожного з методів.

Наостанок в даній роботі підведено підсумки з основними перевагами та недоліками кожного методу на основі даних, отриманих під час роботи, з висновком, який з методів найкраще підходить для вирішення проблеми синхронізації фаз.

Ключові слова: програмно-визначуване радіо, AD9361, MIMO, цифрове формування променів, синхронізація фаз, телекомунікації.

## **ABSTRACT**

The body of this thesis consists of 42 pages with 8 figures, 6 tables, 11 references to other sources and 3 appendices

The purpose of this thesis is to identify the most common issues arising while implementing radiosystems using MIMO technologies on SDR platforms using the Analog Devices AD9361 transceiver and analyse solutions that could be used to solve these problems, focusing on the problem of synchronizing phase between receive channels of the AD9361 transceiver.

This thesis contains a detailed theory of operation, implementation details and research of precision and performance characteristics for each of the three phase difference determination methods described in this thesis: the cross-correlation method, the inner product method and the frequency domain (or Fourier transform) method.

Additionally, the thesis contains a practical implementation of phase difference determination methods and a phase correction method for SDR transceivers based on the Analog Devices AD9361 transceiver. The practical implementation is written in C using the libiio and FFTW libraries and is split into three parts: a library containing implementations of the phase difference determination methods, an application for correcting phase difference and an application for practically measuring the performance for every method.

As the last part of the thesis, a summary of the main advantages and disadvantages of each method is presented, using the data obtained during research for this thesis, with a conclusion that specifies which method is best for solving the phase synchronization problem.

Keywords: software-defined radio, AD9361, MIMO, digital beamforming, phase synchronization, telecommunications.

## ЗМІСТ

|                                                                                                    |    |
|----------------------------------------------------------------------------------------------------|----|
| ВСТУП.....                                                                                         | 8  |
| РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ПО МЕТОДАМ КОРЕКЦІЇ ФАЗИ В<br>МІМО СИСТЕМАХ.....                    | 10 |
| Висновки.....                                                                                      | 12 |
| РОЗДІЛ 2 МЕТОДИ ВИЗНАЧЕННЯ РІЗНИЦІ ФАЗ.....                                                        | 13 |
| 2.1. Огляд принципу роботи методу крос-кореляції.....                                              | 13 |
| 2.2. Огляд принципу роботи методу внутрішнього добутку.....                                        | 14 |
| 2.3. Огляд принципу роботи методу частотного домену.....                                           | 16 |
| Висновки.....                                                                                      | 18 |
| РОЗДІЛ 3 ДОСЛІДЖЕННЯ ТОЧНОСТІ МЕТОДІВ ВИЗНАЧЕННЯ РІЗНИЦІ<br>ФАЗИ.....                              | 19 |
| 3.1. Загальний опис процесу дослідження точності методів.....                                      | 19 |
| 3.2. Обширне моделювання точності методів.....                                                     | 19 |
| 3.3. Точність методу крос-кореляції.....                                                           | 21 |
| 3.4. Точність методів визначення різниці фази при дії шуму на сигнали.....                         | 22 |
| Висновки.....                                                                                      | 30 |
| РОЗДІЛ 4 ПРАКТИЧНА РЕАЛІЗАЦІЯ КОРЕКТОРА РІЗНИЦІ ФАЗИ НА SDR<br>ТРАНСИВЕРІ AD9361.....              | 31 |
| 4.1. Опис практичної реалізації.....                                                               | 31 |
| 4.2. Дослідження швидкості методів визначення різниці фази на основі<br>практичної реалізації..... | 34 |
| Висновки.....                                                                                      | 44 |
| ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ.....                                                                   | 45 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....                                                                    | 46 |
| ДОДАТОК А. Опис експериментального стенду для досліджень.....                                      | 47 |
| ДОДАТОК Б. Код для моделювання методів у MATLAB.....                                               | 48 |
| ДОДАТОК В. Код практичної реалізації на C.....                                                     | 56 |

**ПЕРЕЛІК СКОРОЧЕНЬ**

|      |                                |
|------|--------------------------------|
| MIMO | Multiple Input Multiple Output |
| SDR  | Software Defined Radio         |
| FFT  | Fast Fourier Transform         |
| LO   | Local Oscillator               |

## ВСТУП

Для підвищення продуктивності та надійності радіосистем часто застосовується підхід MIMO (Multiple Input Multiple Output, або «кілька входів, кілька виходів»), суть якого полягає у використанні кількох антен для прийому чи передачі радіосигналів. За цим підходом слідує низка технологій, які по різному використовують множину антен для прийому/передачі даних.

Однією з таких технологій є цифрове формування променів (digital beamforming). Принцип формування променів полягає у використанні множини антен в певній конфігурації (антенної решітки) та маніпуляції амплітуди та фази сигналу кожного елементу антенної решітки таким чином, щоб сформувати необхідну діаграму спрямованості антенної решітки. Цифрове формування променів полягає у використанні методів цифрової обробки сигналів для маніпуляції сигналами.

Оскільки при формування променів маніпулюється фази та амплітуди сигналів, формування променів вимагає синхронізації фаз та амплітуд сигналів з антенних решіток. Будь-яка похибка може спотворити діаграму спрямованості антенної решітки та, як наслідок, подавити головну пелюстку, внести похибку в напрямок променю або підсилити побічні пелюстки, тим самим зменшивши відношення сигнал-шум променю. Отже, при реалізації систем MIMO з цифровим формуванням променів важливо забезпечити синхронізацію амплітуд та фаз кожного антенного елементу.

Дана робота зосереджена саме на радіосистемах, що використовують SDR (Software Defined Radio, або програмно визначуване радіо) трансивери. Оскільки SDR трансивери здійснюють обробки сигналу здебільшого в програмному середовищі, де процес обробки сигналу можна кардинально змінити або переконфігурувати з малою затримкою, вони є дуже гнучкими та чудово підходять для реалізації систем MIMO з цифровим формуванням променів.

Але, SDR трансивери, як і всі компоненти радіосистеми, вносять похибки в сигнал. Як описано вище, для систем MIMO з цифровим формування променів це є проблемою.

Дана робота зорієнтована саме на системи, побудовані на основі SDR трансивера Analog Devices AD9361, через його гнучкість, поширеність та доступність. Сам по собі, трансивер має лише два канали на прийом та передачу, що робить його непридатним для побудови систем MIMO з цифровим формуванням променів через низьку можливу кількість елементів антенної решітки, але можна об'єднати кілька трансиверів, що збільшить кількість каналів на прийом та передачу і, відповідно, можливу кількість елементів антенної решітки.

У випадку використання кількох трансиверів проблема синхронізації фаз між каналами є особливо важливою, оскільки цей SDR трансивер не має вбудованого способу синхронізації фаз радіочастотних сигналів між кількома трансиверами.

Отже, основним об'єктом дослідження в даній роботі є проблема синхронізації фаз між двома каналами. Метою даної роботи є дослідити дану проблему та можливі методи її вирішення.

## РОЗДІЛ 1

### ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ПО МЕТОДАМ КОРЕКЦІЇ ФАЗИ В МІМО СИСТЕМАХ

Проблема синхронізації фаз між каналами достатньо ґрунтовно описана в [1], в якій висвітлюється проблема синхронізації фаз між каналами. В даній статті наведено аргументування важливості даної проблеми, розгляд методів визначення різниці фаз між каналами, критерії вибору методу, формулювання різниці фаз як затримку в часі та детальний опис методу зсуву фази сигналу.

Загалом, ця стаття наводить 6 методів визначення різниці фаз між каналами:

- за допомогою перетинання нулів (zero crossings) — в статті аргументується недоцільність даного методу через необхідність обробки кожного з двох різних I/Q сигналів окремо та неточність при наявності зміщення постійного струму або різниці в амплітудах між сигналами;

- за допомогою крос-кореляції (cross correlation) — в статті вказується на приклад реалізації даного методу в застосунку ІО Oscilloscope, описується принцип роботи даного методу як вимірювання затримки сигналу (дійсного чи комплексного) через схожість сигналів та наводиться недолік даного методу: точність залежить від періоду дискретизації сигналу;

- за допомогою одноточкового аналізу (single point analysis) — в статті описується принцип роботи даного методу як використання закону косинусів для знаходження фази (кута) між двома точками та аргументується недоцільність даного методу через аналіз лише одного відліку часу;

- за допомогою внутрішнього добутку (inner product) — в статті наводиться формула (1.1) для обчислення різниці фаз за даним методом (заснована на одноточковому аналізі) та аргументується більша відповідність даного методу для синусоїдних сигналів.

$$\theta = \arccos \left( \frac{\sum_{n=0}^{N-1} (x[n] \bar{y}[n])}{\sqrt{\sum_{n=0}^{N-1} |x[n]|^2 \sum_{n=0}^{N-1} |y[n]|^2}} \right) \quad (1.1)$$

- за допомогою частотного домену (frequency domain) — в статті описується принцип дії даного методу: сигнали перетворюються в частотний домен, шукається відлік з необхідною частотою та обчислюється різниця кутів між ними;

- за допомогою перетворення Гільберта — в статті написано, що цей метод можливо реалізувати, але автор статті його не реалізував на момент написання статті.

Також в статті описано метод корекції фази, використовуючи окремі I/Q канали, та різні примітки для нього. Суть даного методу полягає в знаходженні значень синуса та косинуса кута зсуву фази та множенні і додаванні цих значень до I та Q каналів. Цей процес можна описати наступними формулами:

$$\begin{bmatrix} I'(t) \\ Q'(t) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} I(t) \\ Q(t) \end{bmatrix} \quad (1.2)$$

де  $\theta$  - зсув фази.

Попри високу деталізацію статті [1] в ній відсутній детальний опис принципу роботи та аналіз характеристик, переваг та недоліків для кожного методу. Отже, основним напрямком досліджень в даній роботі є детальне дослідження вказаних в [1] методів.

Для дослідження в даній роботі було вибрано 3 методи визначення різниці фаз між каналами: за допомогою крос-кореляції, внутрішнього добутку та частотного домену.

Для реалізації методу крос-кореляції було частково використано код застосунку ІО Oscilloscope від Analog Devices, а саме код втулки (plugin) «AD936X Advanced» [2]. Дана втулка має функцію синхронізації фаз між каналами для пристроїв з кількома трансиверами AD9361. Як вказано в [1], для визначення різниці фази між каналами використовується крос-кореляція. Процес

визначення відбувається наступним чином: обчислюється крос-кореляція двох сигналів, знаходиться затримка між фазами, обчислюється різниця фаз з використанням затримки, частоти сигналу, частоти дискретизації та амплітуди сигналу. Формула (3.2) є узагальненою та видозміненою версією формули з [2].

Для обчислення крос-кореляції в практичній реалізації був використаний алгоритм обчислення крос-кореляції двох сигналів з використанням FFT через бібліотеку FFTW 3 з [3].

### **Висновки**

В даному розділі було розглянуто наявні напрацювання по темі даної роботи та на основі них було визначено основний напрямок досліджень в роботі: детальне дослідження вказаних в [1] методів визначення різниці фази. Також було перераховано додаткові напрацювання, які було використано в даній роботі.

Для дослідження в роботі було вибрано наступні три методи визначення різниці фази: метод крос-кореляції, метод внутрішнього добутку та метод частотного домену.

Також з [1] було взято метод корекції фази комплексного сигналу шляхом домноження коефіцієнтів до його I/Q компонентів. Це корисно для середовищ, в яких немає підтримки математичних операцій з комплексними числами.

## РОЗДІЛ 2

### МЕТОДИ ВИЗНАЧЕННЯ РІЗНИЦІ ФАЗ

#### 2.1. Огляд принципу роботи методу крос-кореляції

Суть методу крос-кореляції полягає у знаходженні затримки сигналу через крос-кореляцію двох сигналів та перетворенні цієї затримки на різницю фаз.

Крос-кореляція — це міра схожості двох сигналів в залежності від затримки  $\tau$  одного сигналу щодо іншого. Крос-кореляцію можна описати наступною формулою [4]:

$$[a \star b](\tau) = \int_{-\infty}^{\infty} \bar{a}(t) b(t+\tau) dt \quad (2.1)$$

де  $a, b$  - сигнали,  $\tau$  - затримка,  $t$  - час,  $\bar{\phantom{x}}$  - оператор спряженого числа.

Затримку між сигналами можна визначити як аргумент  $\tau$ , що максимізує значення функції (2.1) [5]. Варто відмітити, що при роботі з дискретними сигналами затримка визначена як відлік.

Перетворення затримки сигналу на різницю фаз відбувається наступним чином: якщо розглядати фазу сигналу як затримку сигналу в межах одного періоду сигналу, можна перевести затримку сигналу зі шкали затримки (від 0 до періоду сигналу) на шкалу фази (від 0 до  $2\pi$  радіан):

$$\theta = 2\pi \cdot \frac{\tau}{T} \quad (2.2)$$

де  $\theta$  - різниця фаз,  $T$  - період сигналу.

З формули (2.2) випливає, що для використання даного методу має бути відомим період сигналу.

При роботі з дискретними сигналами виникає проблема: різниця фази, обчислена формулою (2.2), є неточною. Це виникає через те, що шкала затримок визначена як послідовність рівновіддалених відліків, але шкала фаз, в яку затримка переводиться, визначена неперервно. Отже, при переведенні фази через

формулу (2.2) фаза округлюється до тієї, яка відповідає цілочисельному значенню затримки.

Цю проблему можна вирішити, інтерполювавши значення затримки. Для цього можна використати параболічну інтерполяцію, що апроксимує аргумент максимізації неперервної функції, використовуючи відлік з максимальним значенням та по відліку до та після нього [7]:

$$\beta = y[0], \alpha = y[-1], \beta = y[1] \quad (2.3)$$

де  $y$  - дискретна функція, для якої необхідно апроксимувати аргумент максимізації.

Для початку знаходиться аргумент максимізації параболі [7]:

$$p = \frac{1}{2} \frac{\alpha - \gamma}{\alpha - 2\beta + \gamma} \quad (2.4)$$

Після обчислення (2.4) можна знайти апроксимацію аргументу максимізації:

$$k^* \approx k_\beta + p \quad (2.5)$$

де  $k^*$  - апроксимація аргументу максимізації,  $k_\beta$  - номер відліку  $\beta$ .

## 2.2. Огляд принципу роботи методу внутрішнього добутку

Для визначення різниці фаз між двома сигналами можна використати метод добутку векторів. Суть методу внутрішнього добутку полягає у розгляді двох сигналів як векторів і знаходженню кута між ними за допомогою закону косинусів.

Нижче наведена узагальнена формула для знаходження кута між двома векторами згідно закону косинусів:

$$\theta(\vec{a}, \vec{b}) = \arccos\left(\frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|}\right) \quad (2.6)$$

де  $\vec{a}, \vec{b}$  - вектори,  $\vec{a} \cdot \vec{b}$  - внутрішній добуток векторів (для дійсних векторів це скалярний добуток),  $|\vec{a}|$  - норма вектору.

При використанні дійсних векторів формулу (2.6) можна використовувати як є, але для використання комплексних векторів цю формулу необхідно модифікувати, оскільки внутрішній добуток комплексних векторів (або Ермітового внутрішнього добутку) є комплексним і значення кута, отримана з (2.6), буде комплексним.

В даному випадку можна наближено отримати Евклідов внутрішній добуток (або скалярний добуток векторів) з Ермітового внутрішнього добутку, взявши дійсну частину з нього [6]:

$$\theta(\vec{a}, \vec{b}) = \arccos \left( \frac{\Re[\vec{a} \cdot \vec{b}]}{|\vec{a}| |\vec{b}|} \right) \quad (2.7)$$

де  $\Re[z]$  - дійсна частина комплексного числа  $z$ .

Після цього в формулу (2.7) можна підставити формули Ермітового внутрішнього добутку та норми комплексного вектору, і вийде наступна формула для отримання різниці фаз між двома комплексними сигналами:

$$\theta(a, b) = \arccos \left( \frac{\Re \left[ \sum_{n=0}^N a[n] \bar{b}[n] \right]}{\sqrt{\sum_{n=0}^N |a[n]|^2} \sqrt{\sum_{n=0}^N |b[n]|^2}} \right) \quad (2.8)$$

де  $|z|$  - модуль комплексного числа  $z$ .

Формула (2.8) є схожою на формулу для методу внутрішнього добутку в [1], але має одну важливу відмінність — з Ермітового внутрішнього добутку добувається Евклідов внутрішній добуток.

Основною теоретичною перевагою даного методу є те, що оскільки всі суми в формулі (2.8) можна обчислити незалежно одна від одної, обчислення даної формули можна реалізувати алгоритмом зі складністю  $O(n)$ . Для порівняння, складність методів на основі алгоритму FFT з використанням бібліотеки FFTW 3 становить  $O(n \log n)$  [8].

Оскільки в формулі (2.8) використовується функція арккосинуса, то діапазон можливих кутів становить  $0 \leq \theta(a, b) \leq \pi$  радіан. Отже, цей метод не підходить для визначення різниці фаз в більше ніж  $\pi$  радіан (180).

### 2.3. Огляд принципу роботи методу частотного домену

Для визначення різниці фаз між двома сигналами можна використати метод частотного домену (перетворення Фур'є). Суть цього методу полягає у використанні комплексного перетворення Фур'є для перетворення обох сигналів в частотний домен та отримання різниці сигналів з відліків частотного домену.

Перетворення Фур'є можна описати наступною формулою [9]:

$$A(k) = \int_{-\infty}^{\infty} a(x) e^{(-i2\pi kx)} dx \quad (2.9)$$

де  $a(x)$  - сигнал в часовому домені,  $A(k)$  - сигнал в частотному домені,  $x$  - аргумент часу,  $k$  - аргумент частоти.

Для дискретних сигналів перетворення Фур'є (також відоме як дискретне перетворення Фур'є) виглядає наступним чином:

$$A[k] = \sum_{n=0}^{N-1} a[n] e^{(-i2\pi \frac{kn}{N})} \quad (2.10)$$

де  $a[n]$  - дискретний сигнал в часовому домені,  $A[k]$  - дискретний сигнал в частотному домені,  $N$  - кількість відліків, що враховуються при перетворенні,  $n$  - відлік часу,  $k$  - відлік частоти.

Інакше кажучи, дискретне перетворення Фур'є перетворює дискретний сигнал з набору відліків, що відповідають певним відміткам часу, у набір відліків, що відповідають певним частотним компонентам сигналу.

При застосуванні перетворення Фур'є сигнал стає набором комплексних відліків, що помітно з експоненту  $e^{-i}$  в формулах (2.9) та (2.10). Ці комплексні відліки містять амплітуду та фазу сигналу для певної частоти в модулі та аргументі певного відліку. Таким чином, якщо є два сигнали, що мають частотні компоненти, пов'язані між собою зсувом фази, за допомогою них можна знайти цей зсув фаз з сигналу в частотному домені:

$$\theta = \text{Arg } A[k_1] - \text{Arg } B[k_2] \quad (2.11)$$

де  $\text{Arg}$  - аргумент (фаза) комплексного числа,  $k_1$  - перший пов'язаний частотний компонент,  $k_2$  - другий пов'язаний частотний компонент.

Отже, основною проблемою реалізації даного методу є реалізація алгоритму вибору необхідних для визначення різниці фази частотних компонентів. Вибір даного алгоритму залежить від вимог до точності даного методу при наявності спотворень (наприклад шуму чи похибок частоти). В ході даної роботи було реалізовано наступний алгоритм: вибирається частотний компонент з найбільшою амплітудою в першому сигналі та відношенням між амплітудами компонентів менш ніж 50%.

Основними перевагами даного алгоритму є його низька складність та можливість реалізації зі складністю  $O(n)$ , оскільки суть цього алгоритму — пошук найбільшого значення (що можна реалізувати зі складністю  $O(n)$ ), але лише того, що відповідає певному критерію. Основним недоліком даного алгоритму є відсутність будь-якої адаптації до похибок частоти.

Для збільшення точності даного методу можна проаналізувати знаки відліків  $A[k_1]$  та  $B[k_2]$ , і переставити місцями компоненти формули (2.11):

$$\theta = \text{Arg } B[k_2] - \text{Arg } A[k_1] \quad (2.12)$$

## Висновки

В даному розділі було описано та розглянуто три методи визначення різниці фаз між сигналами, що було досліджено та реалізовано в ході даної роботи.

Метод крос-кореляції обчислює крос-кореляцію двох сигналів, знаходить з неї затримку та переводить її в зсув фази, що є можливим, оскільки фазу можна інтерпретувати як затримку в менш ніж один період.

Метод внутрішнього добутку обчислює різницю фази між сигналами, як кут між двома векторами. Основною перевагою даного методу є те, що його можна обчислити алгоритмом зі складністю  $O(n)$ . Основним недоліком даного методу є те, що різниця фази для даного методу обмежена: від 0 до 180 градусів.

Метод частотного домену обчислює різницю фази між сигналами, переводячи їх в частотний домен (за допомогою перетворення Фур'є) та визначаючи фазу, використовуючи відповідні частотні компоненти. Для вибору частотних компонентів було запропоновано базовий алгоритм, що має складність  $O(n)$ , але не коригує похибку частоти.

## **РОЗДІЛ 3**

### **ДОСЛІДЖЕННЯ ТОЧНОСТІ МЕТОДІВ ВИЗНАЧЕННЯ РІЗНИЦІ ФАЗИ**

#### **3.1. Загальний опис процесу дослідження точності методів**

В даній роботі дослідження точності методів визначення різниці фаз проведено за допомогою моделювання даних методів. Для моделювання було використано середовище MATLAB, в якому було реалізовано всі три методи визначення різниці фаз. Дане середовище було вибране, оскільки воно має багато інструментів, які дозволяють реалізовувати різні алгоритми, досліджувати їх, маніпулюючи багатьма можливими параметрами та візуалізовувати результати досліджень за допомогою різних видів графіків. Код MATLAB для досліджень в даному розділі наданий в додатку Б.

#### **3.2. Обширне моделювання точності методів**

Першим етапом дослідження точності методів визначення різниці фаз стало обширне моделювання точності методів. Мета обширного моделювання полягає у виявленні ключових закономірностей та визначення подальших напрямків досліджень шляхом моделювання певного сценарію, використовуючи обмежені за розміром вибірки всіх основних параметрів сценарію, та поєднанні результатів моделювання з використаними в ході моделювання параметрами.

Для обширного моделювання відбираються параметри та значення, що будуть використовуватися в ході моделювання, і з них формуються конфігурації — різні комбінації значень параметрів. Після формування конфігурацій проводяться моделювання певного сценарію з параметрами кожної конфігурації і результати цих моделювань пов'язуються з конфігураціями, що були використані при моделюванні. Останній крок дозволяє аналізувати результати моделювання за конкретним параметром чи кількома параметрами.

В даній роботі для обширного моделювання точності методів визначення різниці фаз були використані наступні параметри:

- частота сигналів (від 1 кГц до 2,5 МГц)

- частота дискретизації (від 2 кГц до 5 МГц)
- кількість циклів сигналів (1, 2, 4, 8, 16, 32, 64, 128)
- зсув фази одного сигналу щодо іншого (-180°, -90°, -45°, -30°, 0, 30°, 45°, 90°, 180°)
- відношення сигнал-шум в дБ (-10, -8, -4, -2, -1, 0, 10, 20, 40,  $\infty$  або сигнал без шуму)

Процес моделювання для кожної конфігурації наступний:

- згідно параметрів конфігурації формуються два комплексні косинусоїдні сигнали, один з них має зсув фази
  - якщо в сигналі має бути шум, накладається білий адитивний гаусівський шум з відповідною потужністю для досягнення відношення сигнал-шум в конфігурації
- на згенерованих двох сигналах застосовується певний метод визначення різниці фази (теж частина конфігурації)
  - записується визначена різниця фази
  - обчислюється та записується абсолютна похибка між реальним зсувом фази та визначеною різницею фази

За результатами даного етапу дослідження було виявлено наступні закономірності:

- метод крос-кореляції чутливий до кількості циклів сигналів, частоти дискретизації, частоти сигналів та зсуву фази навіть за відсутності шуму
- методи частотного домену та внутрішнього добутку абсолютно точні (похибки більше, ніж  $10^{-6}$  градусів відсутні) за відсутності дії шуму в сигналі
- метод частотного домену чутливий до частоти дискретизації, кількості циклів сигналів, частоти сигналів та зсуву фази при дії шуму
- метод внутрішнього добутку чутливий до зсуву фази, кількості циклів сигналів, частоти дискретизації та частоти сигналів при дії шуму

З цих закономірностей визначено наступні напрямки досліджень точності:

- дослідження точності методу крос-кореляції в залежності від частоти дискретизації

- дослідження точності методів визначення різниці фаз в залежності від рівня шуму в сигналах

### 3.3. Точність методу крос-кореляції

Для дослідження точності методу крос-кореляції в залежності від кількості циклів сигналів, частоти дискретизації та частоти сигналів даний метод було змодельовано, використовуючи наступні параметри:

- частота сигналів від 1 кГц до 250 кГц
- частота дискретизації від 2 кГц до 5 МГц
- кількість циклів сигналів: 1, 2, 4, 8, 16, 32, 64, 128
- зсув фази одного сигналу щодо іншого:  $30^\circ$

Процес моделювання в даному випадку ідентичний процесу моделювання при обширному моделюванні точності, але етап внесення шуму в сигнал пропущений. Результати моделювання оформлені в набір графіків залежності

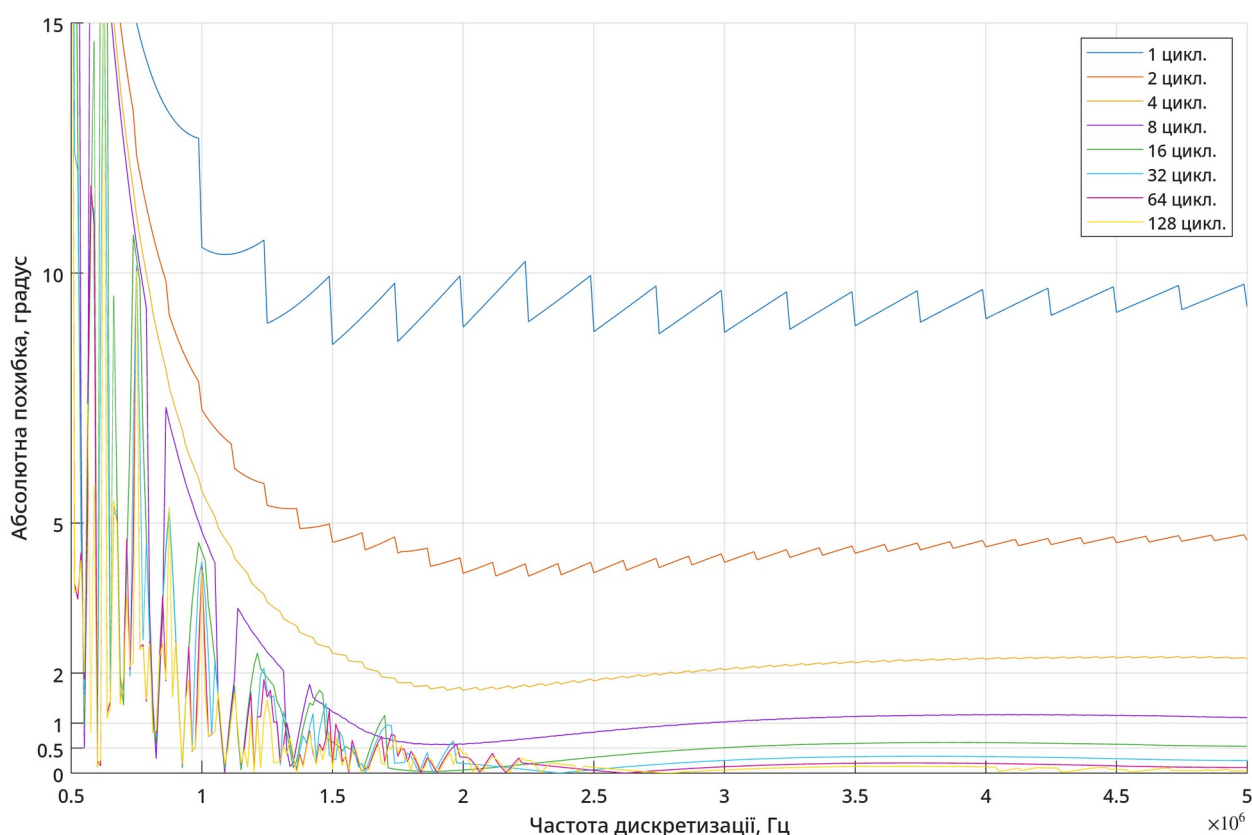


Рис. 3.1. Графік залежності абсолютної похибки від частоти дискретизації та кількості циклів для пари сигналів з частотою 250 кГц

абсолютної похибки в градусах від частоти дискретизації, з кількома лініями для різних кількостей циклів сигналів, по графіку на частоту сигналу. Один з таких графіків, зміст якого буде розглянуто в даному розділі, зображений на рисунку 3.1 для частоти 250 кГц.

На рисунку 3.1 помітно особливість даного методу: абсолютна похибка досить велика та нестабільна до частоти дискретизації в 10 раз вищу за частоту сигналу; після даної частоти похибка стабілізується. Це явище можна пояснити похибкою, внесеною неточною інтерполяцією затримки сигналів, що обчислюється з їх крос-кореляції. Також помітно, що точність цього методу дуже чутлива до точної частоти дискретизації і залежить від кількості циклів сигналу.

### **3.4. Точність методів визначення різниці фази при дії шуму на сигнали**

Для дослідження точності методів визначення різниці фази при дії шуму на сигнали було змодельовано всі три методи визначення різниці фази на сигналах з накладеним адитивним Гаусівським білим шумом. Для кожного методу було проведено окреме дослідження з різними ключовими параметрами.

Для вибору ключових параметрів було використано результати обширного моделювання: для кожного методу було проаналізовано вплив кожного параметру на абсолютну похибку в комбінації з іншими параметрами.

Для методу крос-кореляції були використані наступні параметри:

- частота сигналу 1 кГц
- зсув фази одного сигналу щодо іншого: 30 градусів
- частота дискретизації: від 10 кГц до 5 МГц
- кількість повторів: від 1 до 128
- відношення сигнал-шум: від -10 дБ до 40 дБ

Для методу внутрішнього добутку були використані наступні параметри:

- частота сигналу 1 кГц
- зсув фази одного сигналу щодо іншого: 30 градусів
- частота дискретизації: від 2 кГц до 5 МГц
- кількість повторів: від 1 до 128

- відношення сигнал-шум: від -10 дБ до 40 дБ

Для методу внутрішнього добутку були використані параметри для методу внутрішнього добутку, але з відношенням сигнал-шум в проміжку від -60 дБ до 40 дБ.

Методи будуть порівнюватися за мінімальним порогом для досягнення похибки виміряної різниці фази в менш ніж  $1^\circ$  та максимальною абсолютною похибкою. Варто відмітити, що відношенням сигнал-шум працює наступним чином: якщо це відношення більше, ніж нуль, то рівень сигналу переважає рівень шуму; якщо це відношення дорівнює нулю, то рівні сигналу та шуму однакові; якщо це відношення менше, ніж нуль; то рівень шуму переважає рівень сигналу.

В результаті даних досліджень було отримано набір графіків, що показують залежність абсолютної похибки від відношення сигнал-шум та кількості циклів сигналів або частоти дискретизації (залежить від того, який з параметрів найбільше впливає на точність методу).

Аналіз результатів буде розпочато з методу крос-кореляції. Спочатку буде розглянуто випадок з кількістю циклів сигналів 32. На рисунку 3.2 зображено графік залежності абсолютної похибки для даного випадку.

Згідно рисунку 3.2, абсолютна похибка стабілізується лише після відношення сигнал-шум 9 дБ. Загалом, частоти дискретизації мають досить різні мінімальні пороги відношення сигнал-шум для досягнення похибки різниці фази в менш ніж  $1^\circ$ . Проте, частоти, що діляться на 5 мають нижчий мінімальний поріг відношення сигнал-шум, ніж інші частоти. Також, зі збільшенням частоти дискретизації зменшується загальна похибка різниці фази.

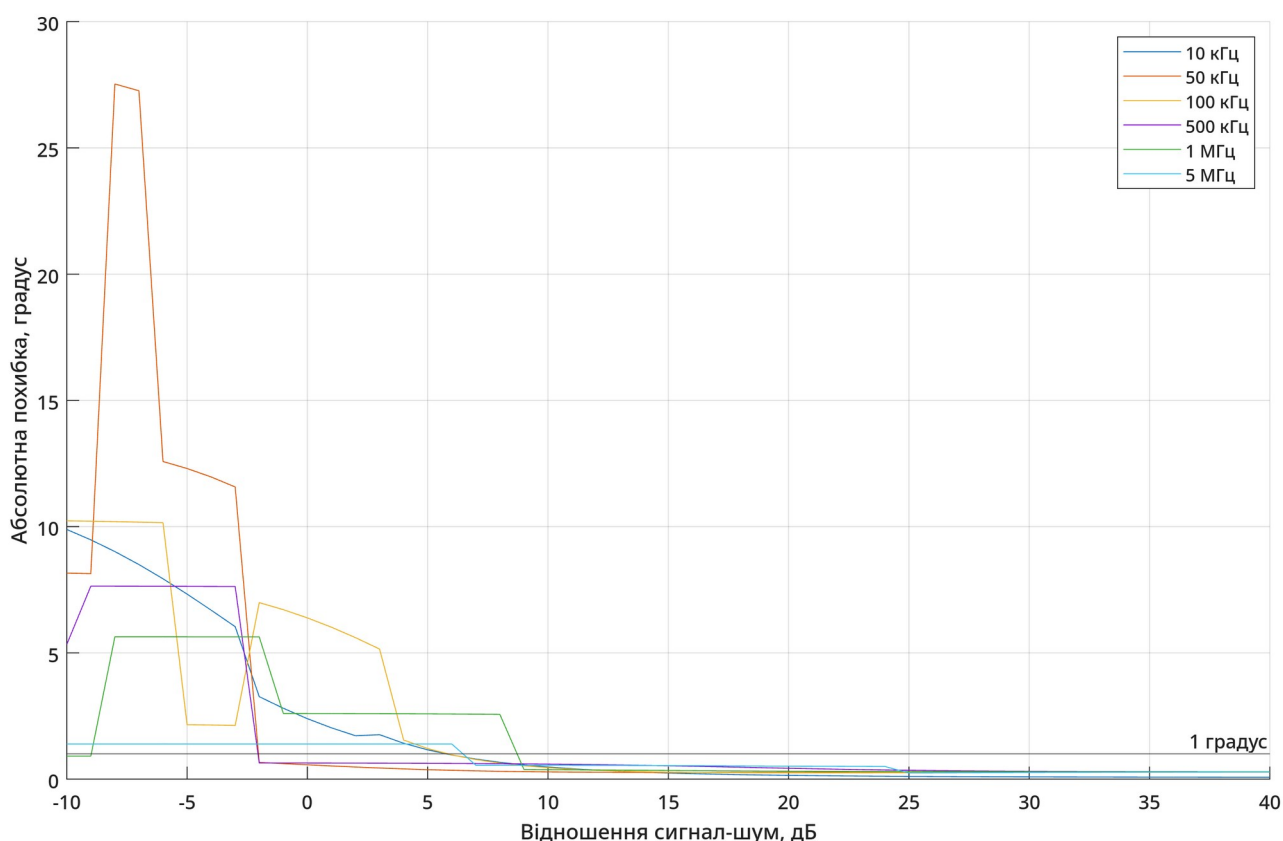


Рис. 3.2. Графік залежності абсолютної похибки від відношення сигнал-шум та частоти дискретизації для методу крос-кореляції при 32 циклах сигналів

Таку залежність абсолютної похибки для методу крос-кореляції можна пояснити неточною інтерполяцією затримки, як і при дослідженні точності даного методу взагалі.

Тепер буде розглянуто випадок з кількістю циклів сигналу 128. На рисунку 3.3 зображено графік залежності абсолютної похибки для даного випадку.

Графік на рисунку 3.3 схожий на той, що на рисунку 3.2, але з кількома відмінностями: абсолютна похибка не перевищує  $16^\circ$  та мінімальний поріг відношення сигнал-шум для досягнення похибки в менш ніж  $1^\circ$  в даному випадку становить приблизно 5 дБ для частот дискретизації 50, 100 кГц, 1, 5 МГц, приблизно 7 дБ для частоти дискретизації 10 кГц та приблизно -3 дБ для частоти дискретизації 500 кГц.

З цих результатів можна зробити висновок, що завадостійкість даного методу є помірною, але нестабільною: зі збільшенням кількості циклів сигналу точність даного методу стає краще, але вплив частоти дискретизації є досить нестабільним.

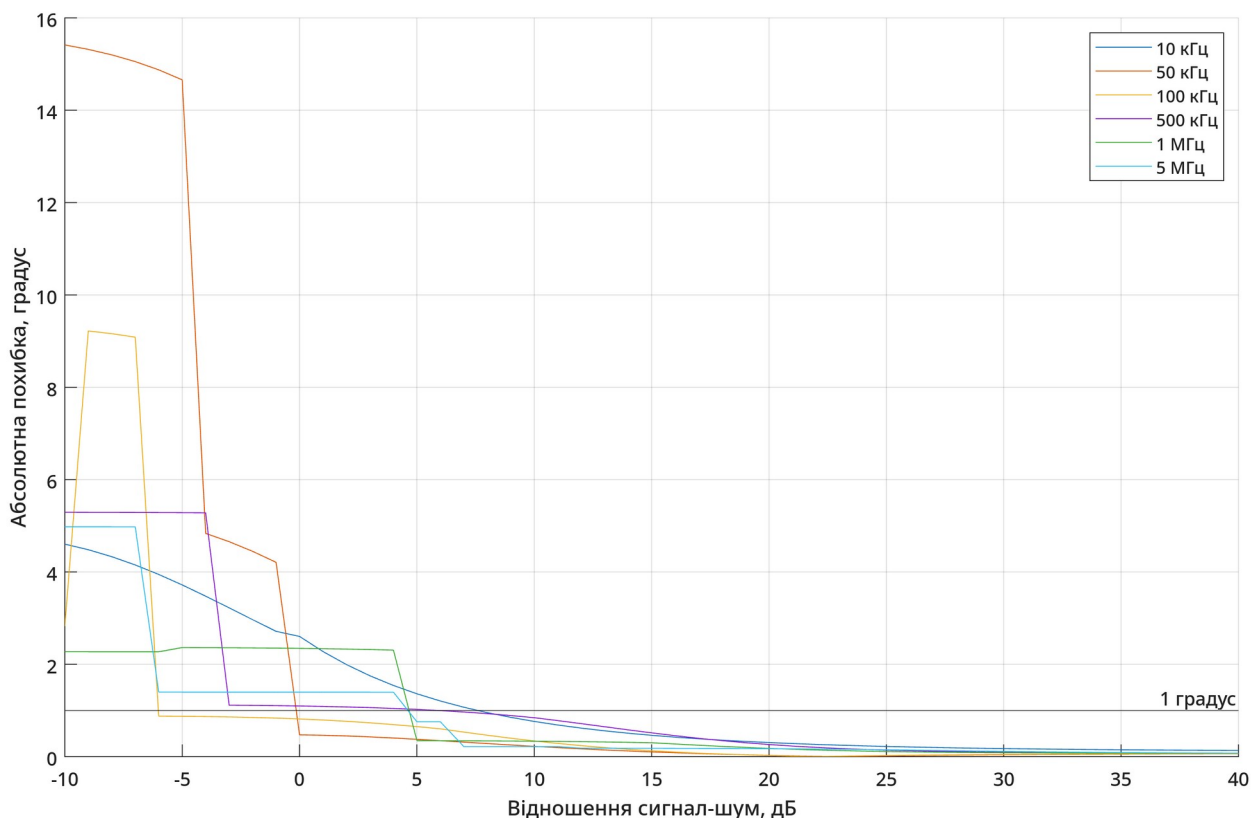


Рис. 3.3. Графік залежності абсолютної похибки від відношення сигнал-шум та частоти дискретизації для методу крос-кореляції при 128 циклах сигналів

Після аналізу методу крос-кореляції буде проаналізовано метод внутрішнього добутку. На рисунку 3.4 зображено графік залежності абсолютної похибки від відношення сигнал-шум та кількості циклів сигналів для методу внутрішнього добутку з частотою дискретизації 2 кГц.

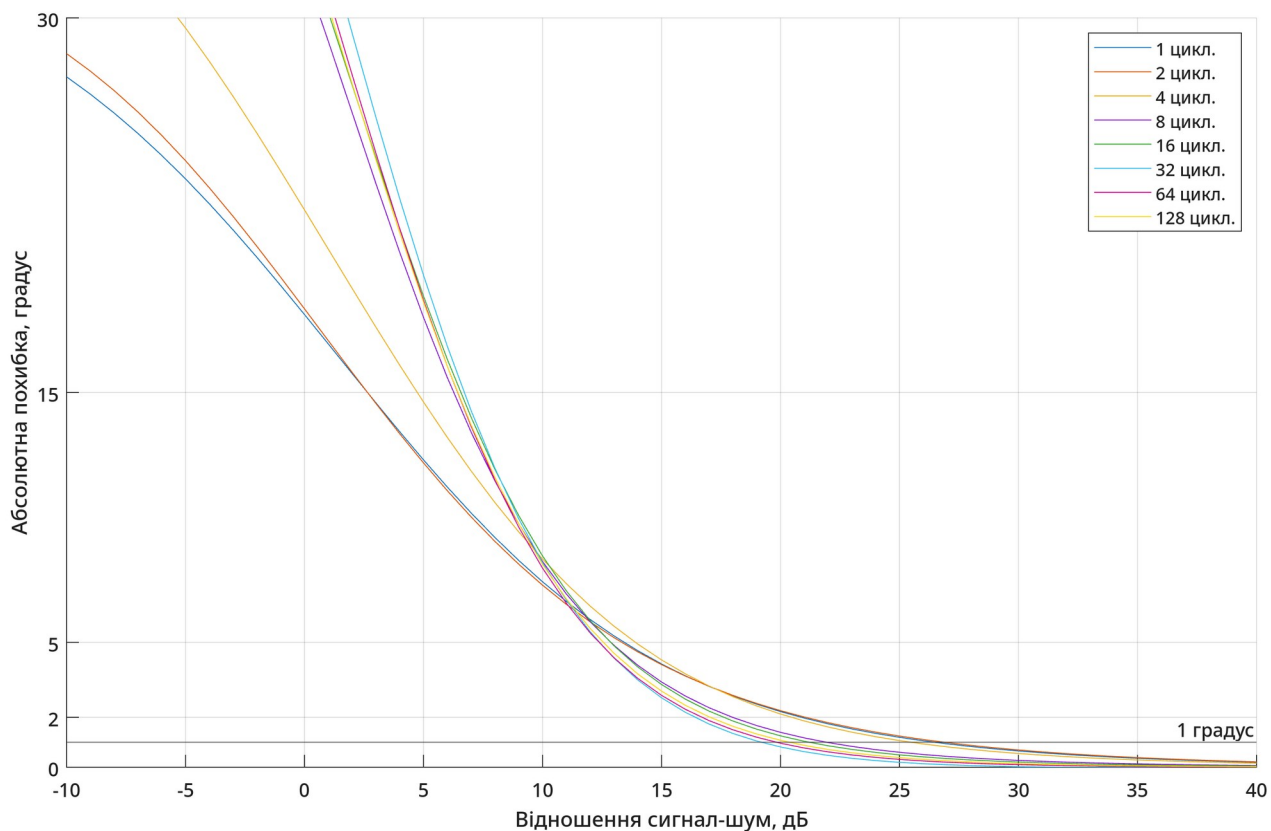


Рис. 3.4. Графік залежності абсолютної похибки від SNR та циклів сигналів для методу внутрішнього добутку з частотою дискретизації 2 кГц

Згідно рисунку 3.4, мінімальний поріг відношення сигнал-шум для досягнення похибки в менш ніж  $1^\circ$  в даному випадку становить приблизно 25 дБ для 8-128 циклів та 30 дБ для 1-4 циклів. В порівнянні з методом крос-кореляції цей результат є гіршим, але метод крос-кореляції використовує як мінімум в 5 разів більшу частоту дискретизації, тому варто розглянути точність даного методу при частоті дискретизації 500 кГц.

На рисунку 3.5 зображено відповідний частоті дискретизації 500 кГц графік залежності абсолютної похибки.

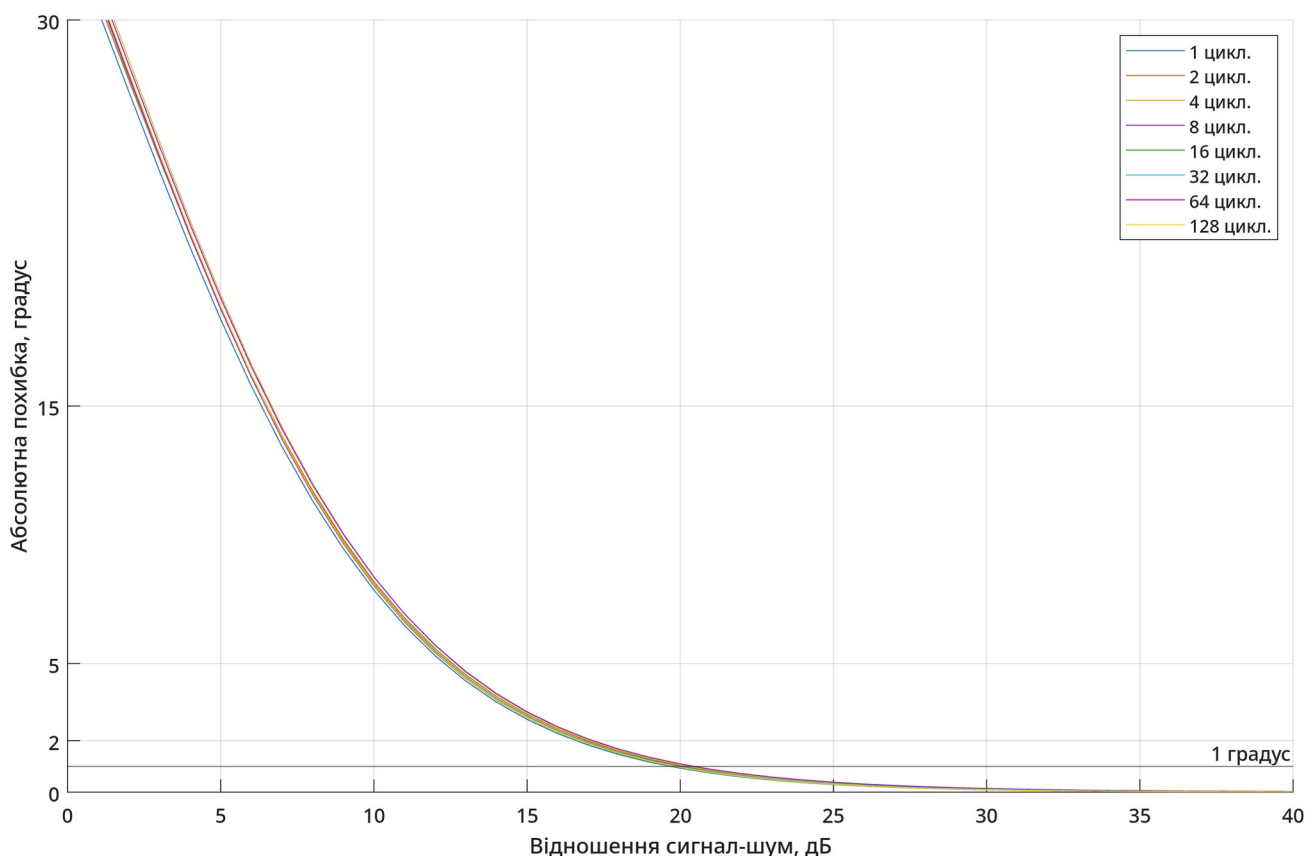


Рис 3.5. Графік залежності абсолютної похибки від SNR та циклів сигналів для методу внутрішнього добутку з частотою дискретизації 500 кГц

Графік на рисунку 3.5 є схожим на графік на рисунку 3.4, окрім відсутності впливу кількості циклів на абсолютну похибку та фіксованого мінімального порогу відношення сигнал-шум для досягнення похибки в менш ніж  $1^\circ$ : в даному випадку воно становить приблизно 21 дБ.

Варто розглянути абсолютну похибку даного методу при максимальній частоті дискретизації: 5 МГц. На рисунку 3.6 зображено відповідний даній частоті дискретизації графік залежності абсолютної похибки.

Даний графік є майже ідентичним графіку на рисунку 3.5, разом з фіксованим мінімальним порогом відношення сигнал-шум для досягнення похибки в менш ніж  $1^\circ$ .

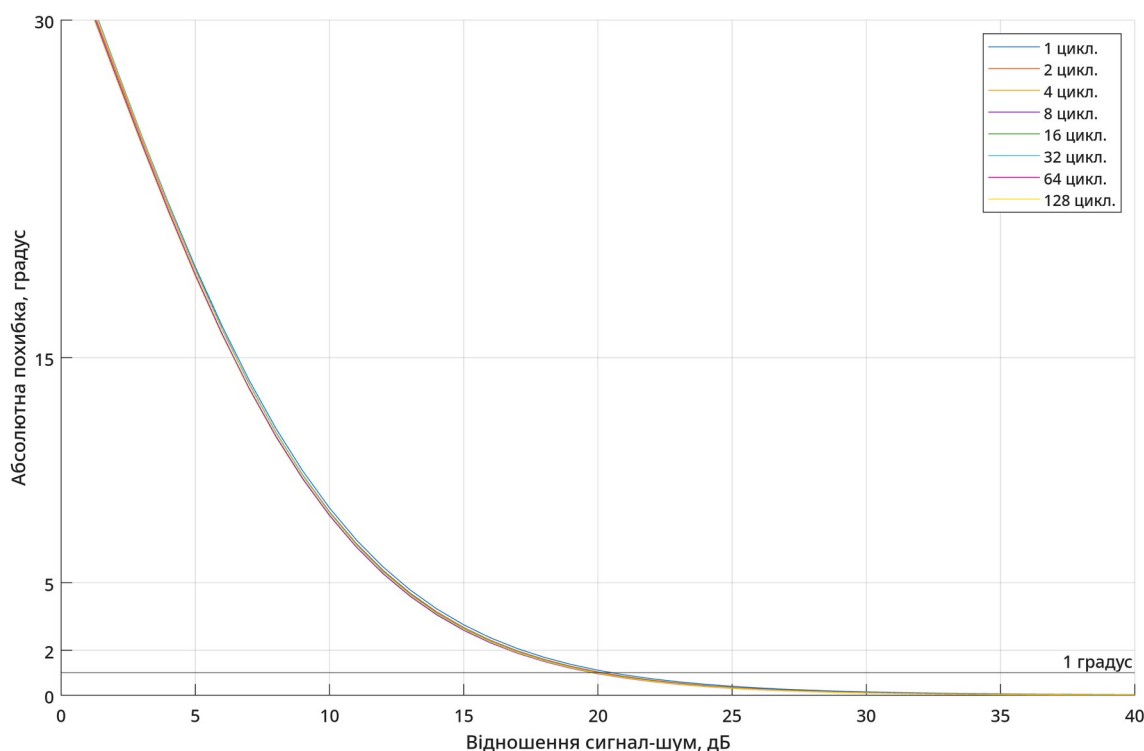


Рис. 3.6. Графік залежності абсолютної похибки від SNR та циклів для методу внутрішнього добутку з частотою дискретизації 5 МГц

З цих результатів можна зробити висновок, що завадостійкість методу внутрішнього добутку дуже низька, але точність даного методу не залежить від частоти дискретизації чи від кількості циклів сигналів.

Після цього буде розглянуто метод частотного домену (перетворення Фур'є). Спочатку буде розглянуто випадок з 32 циклами сигналів. Загалом якоїсь чіткої

На рисунку 3.7 зображено графік залежності абсолютної похибки для методу частотного домену при наявності 32 циклів сигналів. З даних цього графіку можна сказати, що точність даного методу сильно залежить від частоти дискретизації, оскільки розкид мінімальних порогів є досить великим: від -15 дБ для 5 МГц до 10 дБ для 2 кГц.

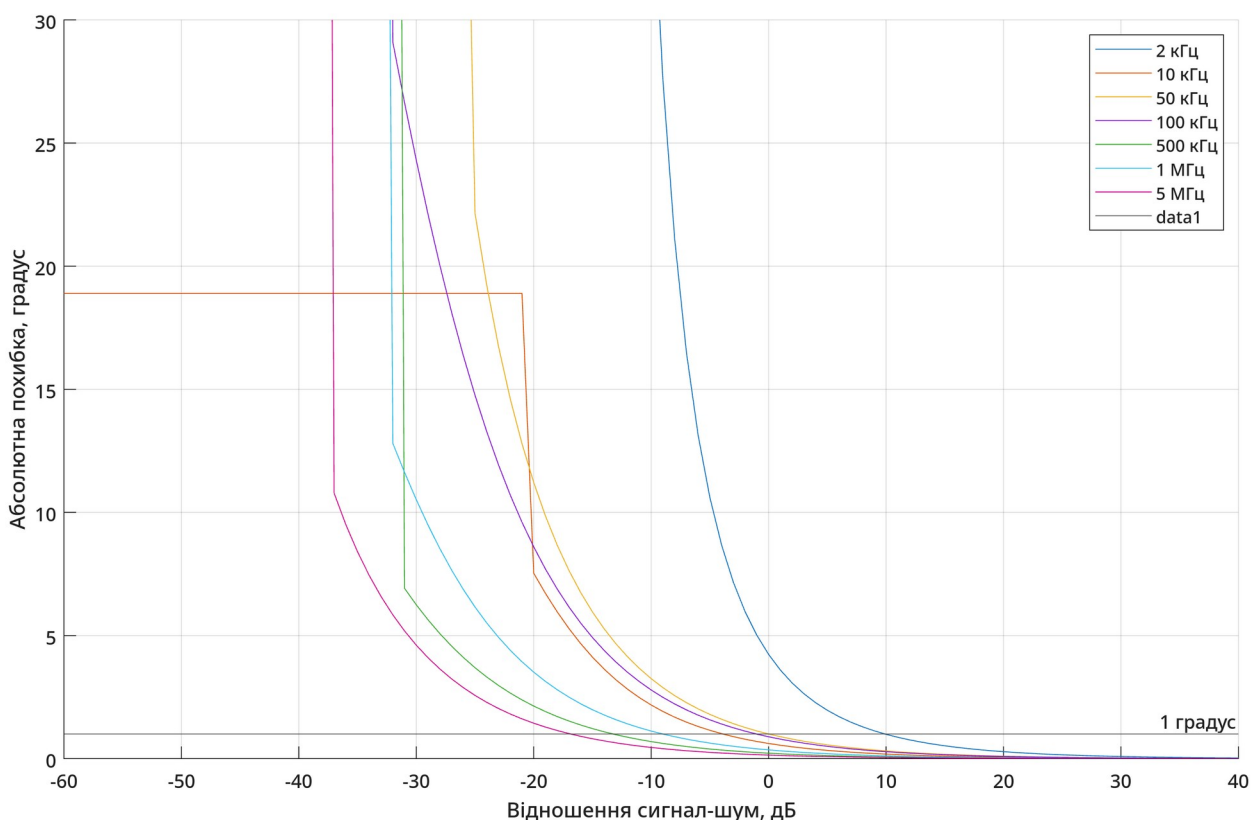


Рис 3.7. Графік залежності абсолютної похибки від SNR та циклів сигналів для методу частотного домену з кількістю циклів сигналів 32

Тепер буде розглянуто випадок з кількістю циклів сигналів 128. На рисунку 3.8 зображено графік залежності абсолютної похибки для даної частоти дискретизації. В порівнянні з графіком для кількості циклів сигналів 32 присутнє загальне зменшення мінімальних порогів: тепер вони розкидані від -37 дБ для частоти дискретизації 1 МГц до 4-5 дБ для частот 2 та 10 кГц. Але також змінився розподіл частот дискретизації за порядком мінімальних порогів: наприклад, в порівнянні з рисунком 3.7 мінімальний поріг для частоти 1 МГц йде раніше за мінімальний поріг для частоти 100 кГц.

З цього можна зробити висновок, що метод частотного домену має помірну, але нестабільну завадостійкість, яка краща за завадостійкість методу крос-кореляції. Скоріше за все, це пов'язано з тим, що білий гаусівський шум (той шум, що застосовувався при моделюванні) спотворює увесь спектр сигналу, а цей метод використовує частотний спектр для знаходження різниці фаз.

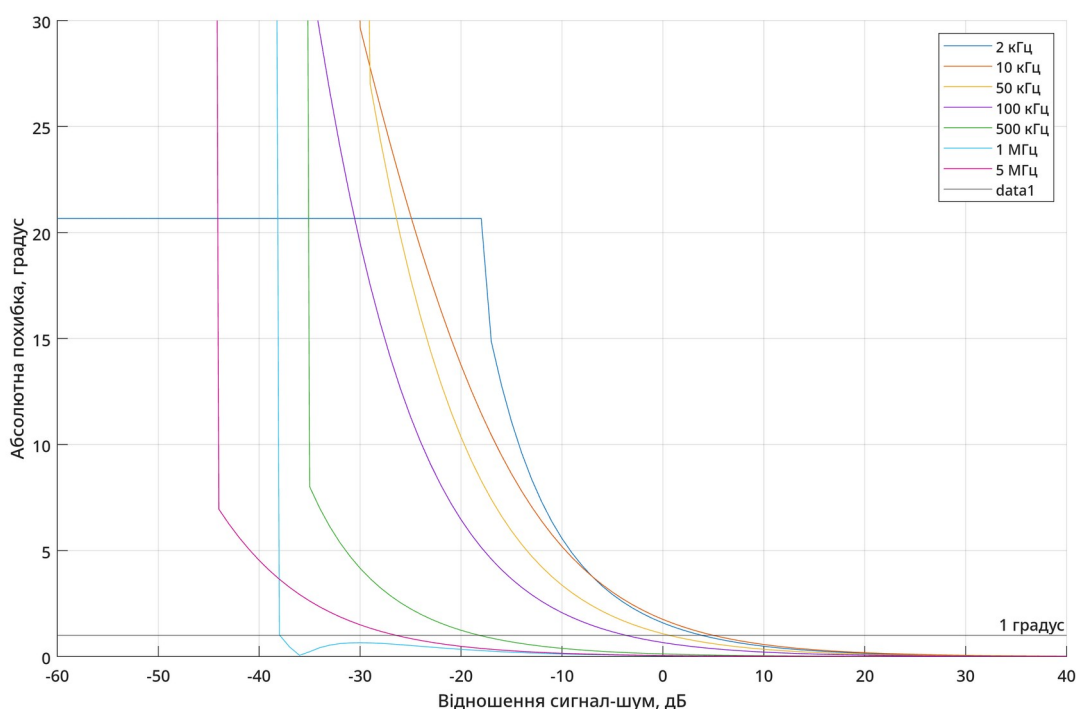


Рис. 3.8. Графік залежності абсолютної похибки від SNR та циклів сигналів для методу частотного домену з 128 циклами сигналів

## Висновки

В даному розділі було досліджено точність всіх трьох методів визначення різниці фази між двома сигналами за різних умов шляхом моделювання даних методів з використанням різних параметрів. З даних обширного моделювання, що було виконано на початку дослідження, було визначено два основні напрямки досліджень точності: дослідження точності методу крос-кореляції та дослідження точності методів визначення різниці фази при дії шуму на сигнал.

Дослідження точності методу крос-кореляції показало, що для даний метод є дуже чутливим до частоти дискретизації та оптимально працює лише за частот дискретизації, які принаймні в 10 разів більші за частоту сигналу.

Дослідження точності методів визначення різниці фази при дії шуму на сигнал показало завадостійкість кожного методу: метод крос-кореляції має помірну, але нестабільну завадостійкість; метод внутрішнього добутку має погану завадостійкість; метод частотного домену має помірну, але кращу завадостійкість в порівнянні з методом крос-кореляції.

## РОЗДІЛ 4

### ПРАКТИЧНА РЕАЛІЗАЦІЯ КОРЕКТОРА РІЗНИЦІ ФАЗИ НА SDR ТРАНСИВЕРІ AD9361

#### 4.1. Опис практичної реалізації

Мета розробки практичної реалізації — продемонструвати дані методи в роботі на SDR трансивері з AD9361 та провести практичне дослідження швидкості кожного методу визначення різниці фази. Код практичної реалізації викладений в додатку В.

Практична реалізація виконана на мові програмування C та складається з трьох частин:

- бібліотеки phaselib, в якій реалізовані методи визначення різниці фаз
- застосунку phasescorrect, що є прикладом коректора фази, який працює з SDR трансивером на базі AD9361 через бібліотеку libiio
- застосунку phasebench, що проводить практичне вимірювання швидкості методів визначення різниці фаз

В бібліотеці phaselib реалізовано всі три методи визначення різниці фази, розглянуті в даній роботі: крос-кореляція, внутрішній добуток та частотний домен. Також присутні допоміжні функції для генерування тестового комплексного косинусоїдного сигналу та обчислення періоду сигналу в відліках з частоти сигналу та частоти дискретизації.

Для реалізації алгоритму FFT бібліотека phaselib використовує сторонню бібліотеку FFTW 3, що є реалізацією дискретного перетворення Фур'є, яка адаптується під апаратне забезпечення, на якому працює, для максимізації продуктивності [8].

Для реалізації крос-кореляції бібліотека використовує код для обчислення крос-кореляції перетворенням Фур'є з бібліотекою FFTW 3 [3]. Оскільки цей код використовує алгоритм FFT, що має складність  $O(n \log n)$  [8], то цей код швидше обчислює крос-кореляцію, ніж пряма реалізація її формули.

Перед використанням методів крос-кореляції та частотного домену в phaselib необхідно створити так званий план для їх використання. Це вимагається принципом роботи бібліотеки FFTW 3: для оптимізації, перед перетворенням з багатьох стратегій перетворення (планів) шукається найефективніша [8]. Власне, в об'єктах планів для методів у бібліотеці phaselib містяться вказівники на плани FFTW, що використовуються даним методом.

Планувальник FFTW може працювати в двох основних режимах, від яких залежить процес визначення ефективності плану:

- режим вимірювання (FFTW\_MEASURE), в якому FFTW точно заміряє час виконання кожного плану
- режим апроксимації (FFTW\_ESTIMATE), в якому FFTW приблизно обчислює ефективність кожного плану

Різниця між цими двома методами в тому, що на планування в режимі вимірювання йде більше часу, ніж на режим апроксимації, але в результаті даний режим дає ефективніші плани.

Для реалізації методу внутрішнього добутку в бібліотеці phaselib використовується алгоритм зі складністю  $O(n)$ , що спочатку обчислює значення знаменника та чисельника в формулі (2.8), проходячи через всі елементи масиву, а вже потім обчислює фінальне значення. Алгоритму вибору частотного компоненту в реалізації методу частотного домену також має складність  $O(n)$ .

Застосунок phasescorrect дозволяє користувачу визначити та автоматично відкоригувати різницю фаз на SDR трансивері на базі AD9361. Він налаштовує трансивер на певну конфігурацію з прийманням двох сигналів на обох каналах та генерації косинусного сигналу, використовуючи цифровий генератор сигналів, вбудований у FPGA-інтерфейс для трансивера, на другому каналі трансивера. Після того він приймає буфери з двох каналів розміром з двома періодами сигналу у відліках та визначає між ними різницю фази. Також застосунок може автоматично скоригувати фазу другого каналу прийому, використовуючи визначену різницю фази.

Даний застосунок працює в режимі командного рядка та приймає команди наступного виду:

phasebench <режим> <адреса> <метод> <конфігурація трансивера>, де:

- <режим> - режим роботи програми (show чи set)
- <адреса> - адреса URI, за якою бібліотека libiiio підключатиметься до трансиверу
- <метод> - метод визначення різниці фази (xcorr, innerproduct чи fft)
- <конфігурація трансивера> - рядок значень, відокремлених двокрапкою, в якому містяться параметри для налаштування трансивера, в наступному порядку
  - частота LO, або гетеродина — радіочастота, на якій працюватиме трансивер (в Гц)
  - частота дискретизації — частота, з якою трансивер захоплює чи конвертує відліки (в Гц)
  - радіочастотна смуга пропускання — смуга пропускання, на яку налаштовуються аналогові фільтри трансивера
  - підсилення каналу приймання 1 (в дБ)
  - підсилення каналу приймання 2 (в дБ)
  - придушення каналу передачі 2 (в дБ, пишеться з мінусом)
  - затримка після налаштування передачі сигналу (в секундах, можна вказати дріб якщо необхідні мілісекунди)

В даного застосунку є два режими роботи:

- Режим визначення різниці фази (show) — лише визначає різницю фази
- Режим корекції різниці фази (set) — визначає та коригує різницю фази

Даний застосунок розрахований на роботу в конфігурації loopback, тобто другий канал передачі та канали прийому мають бути з'єднані між собою.

Для корекції фази застосунок використовує блок корекції I/Q, вбудований в приймальну частину FPGA-інтерфейсу для AD9361, змінюючи параметри `calibscale` та `calibphase`, що дозволяють задати коефіцієнти, на які будуть домножено відліки певного компонента сигналу каналу.

Тут стає потрібною формула (1.2), яка показує, як визначити коефіцієнти множення, необхідні для зсуву фази за компонентами сигналу. В даному випадку рядки матриці відповідають компоненту, для якого необхідно застосувати коефіцієнти, перша колонка матриці відповідає властивості `calibscale`, а друга колонка — властивості `calibphase`.

Застосунок `phasebench` проводить практичне вимірювання швидкості кожного методу визначення різниці фаз для різних комбінацій частоти дискретизації. Даний застосунок буде описаний детальніше в наступному підрозділі.

#### **4.2. Дослідження швидкості методів визначення різниці фази на основі практичної реалізації**

Для дослідження швидкості методів визначення різниці фази було розроблено практичну реалізацію даних методів та застосунок для практичного вимірювання їх швидкості під назвою `phasebench`.

Даний застосунок спочатку генерує різні комбінації частот дискретизації та кількості циклів сигналів, генерує для них сигнали (разом з додатковою інформацією для методів крос-кореляції та частотного домену) та плани (час створення планів заміряється) і потім кілька разів проводить наступне дослідження: для кожної конфігурації визначає різницю фази, використовуючи всі три методи та заміряючи час, витрачений на виконання методу.

Всі заміряні результати виводяться у стандартний вихід процесу в форматі, схожому на CSV (значення, відокремлені комою):

- ідентифікатор `bench`, наявний для ідентифікації результату, оскільки в купі з результатами програма також виводить інші рядки
- номер дослідження

- частота дискретизації (в Гц)
- кількість циклів сигналу
- ідентифікатор методу
  - для методу крос-кореляції: `PHASE_BENCH_XCORR`
  - для методу внутр. добутку: `PHASE_BENCH_INNER_PRODUCT`
  - для методу частотного домену: `PHASE_BENCH_FFT`
- час виконання (в мікросекундах)

Якщо результат для створення плану, то номер дослідження дорівнює -1 та ідентифікатори методу вказують на тип плану:

- для плану методу крос-кореляції: `WARMUP_PLAN_XCORR`
- для плану методу частотного домену: `WARMUP_PLAN_FFT`

Застосунок працює в режимі командного рядка та приймає команди наступного виду:

`phasebench <дослідження> <режим планування>`, де:

- `<дослідження>` - кількість досліджень, які необхідно провести
- `<режим планування>` - режим, в якому буде йти побудова планів FFTW (`measure` (режим вимірювання), `estimate` (режим апроксимації) чи `only-inner-product` (вимірювати лише метод внутрішнього добутку))

Останній режим був доданий, щоб унеможливити вплив виконання інших методів на тривалість виконання методу внутрішнього добутку, оскільки це було проблемою в ході виконання роботи.

Для тестування застосунок генерує косинусоїдні сигнали з частотою 1 кГц та з кількістю відліків, що є частотою дискретизації помноженою на кількість циклів сигналу. Кількість проведених досліджень становить 16 для методів крос-кореляції та частотного домену і 32 для методу внутрішнього добутку.

Тривалості виконання методу кожного дослідження не наводяться прямо в роботі, замість цього з наборів даних значень було обчислено три статистичні показники — мінімум, максимум та арифметичне середнє значення.

Варто відмітити, що всі дослідження були проведені на платі Terasic DE10-Standard з SoCFPGA-процесором Altera Cyclone V SX та двома ядрами ARM Cortex-A9 [11] з тактовою частотою 800 МГц.

Оскільки бібліотека FFTW 3 не підтримує роботу з набором інструкцій NEON, що прискорюють роботу з кількома числами, для чисел з плаваючою комою підвищеної точності (`double`) на 32-бітних платформах ARM [10], то FFTW для даної платформи було побудовано без прискорення за рахунок даних інструкцій.

Це можна вирішити, переписавши бібліотеку `phaselib` так, щоб вона використовувала тип чисел з плаваючою комою одинарної точності (`float`), і тоді FFTW можна побудувати у варіанті одинарної точності з прискоренням за рахунок даного набору інструкцій.

Для початку варто розглянути тривалості планування в режимі вимірювання (таблиця 4.1), оскільки вони є досить суттєвими.

Таблиця 4.1

## Тривалість планування в режимі вимірювання

| Частота дискретизації | Цикли сигналів | Тривалість для плану крос-кореляції | Час для плану частотного домену |
|-----------------------|----------------|-------------------------------------|---------------------------------|
| 2 кГц                 | 1              | 1,179 мс                            | 53,28 мкс                       |
| 2 кГц                 | 2              | 115,54 мкс                          | 59,83 мкс                       |
| 2 кГц                 | 4              | 121,62 мкс                          | 59,71 мкс                       |
| 2 кГц                 | 8              | 4,935 с                             | 124,35 мкс                      |
| 2 кГц                 | 16             | 1,386 с                             | 333,549 мс                      |
| 2 кГц                 | 32             | 6,968 с                             | 1,086 с                         |
| 2 кГц                 | 64             | 775,253 мс                          | 2,051 с                         |
| 2 кГц                 | 128            | 26,918 с                            | 3,23 с                          |
| 250 кГц               | 1              | 14,199 с                            | 2,074 с                         |
| 250 кГц               | 2              | 18,058 с                            | 4,806 с                         |
| 250 кГц               | 4              | 35,522 с                            | 7,915 с                         |
| 250 кГц               | 8              | 22,871 с                            | 11,968 с                        |
| 250 кГц               | 16             | 33,987 с                            | 17,044 с                        |
| 250 кГц               | 32             | 248,05 с                            | 23,595 с                        |
| 250 кГц               | 64             | 70,78 с                             | 30,75 с                         |

| Частота дискретизації | Цикли сигналів | Тривалість для плану крос-кореляції | Час для плану частотного домену |
|-----------------------|----------------|-------------------------------------|---------------------------------|
| 250 кГц               | 128            | 44,296 с                            | 40,602 с                        |
| 5 МГц                 | 1              | 1,356 мс                            | 215,82 мкс                      |
| 5 МГц                 | 2              | 9,552 мс                            | 247,73 мкс                      |
| 5 МГц                 | 4              | 4,342 мс                            | 258,93 мкс                      |
| 5 МГц                 | 8              | 283,972 мс                          | 323,14 мкс                      |
| 5 МГц                 | 16             | 9,722 мс                            | 252,46 мкс                      |
| 5 МГц                 | 32             | 63,316 мс                           | 381,78 мкс                      |
| 5 МГц                 | 64             | 32,841 мс                           | 1,557 мс                        |
| 5 МГц                 | 128            | 315,073 с                           | 95,307 с                        |

Наприклад, для випадку частоти дискретизації 5 МГц з кількістю циклів сигналу 128 час, необхідний для планування крос-кореляції, становить 315 секунд, або приблизно 5 хвилин. В деяких випадках це може бути неприпустимо.

FFTW має функцію, що пришвидшує даний процес, під назвою «Wisdom» (англ. мудрість). Після створення планів можна викликати функцію `fftw_export_wisdom_to_filename`, і FFTW збереже всі створені плани у файл, наданий як аргумент даної функції. Потім, перед створенням будь-яких планів можна викликати функцію `fftw_import_wisdom_from_filename`, і FFTW зчитає всі плани з вказаного файлу, що пришвидшить процес їх створення. Ця функціональність використовується в застосунку `phasebench` для пришвидшення процесу ініціалізації, але за необхідності її можна деактивувати, видаливши файл під назвою «`phasebench.wisdom`».

Тепер варто розглянути тривалість виконання методу крос-кореляції з плануванням в режимі вимірювання (таблиця 4.2).

Таблиця 4.2

Тривалість виконання методу крос-кореляції з плануванням в режимі вимірювання

| Частота дискретизації | Цикли сигналів | Мінімальна тривалість | Максимальна тривалість | Середня тривалість |
|-----------------------|----------------|-----------------------|------------------------|--------------------|
| 2 кГц                 | 1              | 11,46 мкс             | 15,16 мкс              | 12,964 мкс         |
| 2 кГц                 | 2              | 13,38 мкс             | 16,02 мкс              | 14,773 мкс         |

| Частота дискретизації | Цикли сигналів | Мінімальна тривалість | Максимальна тривалість | Середня тривалість |
|-----------------------|----------------|-----------------------|------------------------|--------------------|
| 2 кГц                 | 4              | 16,45 мкс             | 20,03 мкс              | 18,146 мкс         |
| 2 кГц                 | 8              | 40,8 мкс              | 45,47 мкс              | 42,156 мкс         |
| 2 кГц                 | 16             | 35,36 мкс             | 48,31 мкс              | 41,711 мкс         |
| 2 кГц                 | 32             | 256,48 мкс            | 344,67 мкс             | 286,053 мкс        |
| 2 кГц                 | 64             | 170,31 мкс            | 333,59 мкс             | 192,871 мкс        |
| 2 кГц                 | 128            | 829,4 мкс             | 983,94 мкс             | 869,273 мкс        |
| 250 кГц               | 1              | 1,303 мс              | 1,55 мс                | 1,376 мс           |
| 250 кГц               | 2              | 1,032 мс              | 1,247 мс               | 1,065 мс           |
| 250 кГц               | 4              | 6,899 мс              | 7,717 мс               | 7,11 мс            |
| 250 кГц               | 8              | 6,133 мс              | 6,727 мс               | 6,251 мс           |
| 250 кГц               | 16             | 23,558 мс             | 25,299 мс              | 23,78 мс           |
| 250 кГц               | 32             | 72,225 мс             | 76,656 мс              | 73,747 мс          |
| 250 кГц               | 64             | 146,925 мс            | 155,548 мс             | 150,125 мс         |
| 250 кГц               | 128            | 234,264 мс            | 246,498 мс             | 238,029 мс         |
| 5 МГц                 | 1              | 19,685 мс             | 21,824 мс              | 20,106 мс          |
| 5 МГц                 | 2              | 80,417 мс             | 85,463 мс              | 81,901 мс          |
| 5 МГц                 | 4              | 139,776 мс            | 144,211 мс             | 141,003 мс         |
| 5 МГц                 | 8              | 527,643 мс            | 568,757 мс             | 541,559 мс         |
| 5 МГц                 | 16             | 452,033 мс            | 472,406 мс             | 457,564 мс         |
| 5 МГц                 | 32             | 1,961 с               | 2,072 с                | 2,005 с            |
| 5 МГц                 | 64             | 2,452 с               | 2,589 с                | 2,494 с            |
| 5 МГц                 | 128            | 12,251 с              | 13,004 с               | 12,587 с           |

В цій таблиці простежується великий вплив кількості циклів сигналу та частоти дискретизації на час, необхідний для виконання даного методу. Він присутній через те, що частота дискретизації та кількості циклів сигналу прямо керують кількістю даних, які має опрацювати метод.

Тепер варто порівняти таблицю 4.2 з таблицею часу виконання методів крос-кореляції з плануванням в режимі апроксимації (таблиця 4.3).

Таблиця 4.3

Тривалість виконання методу крос-кореляції з плануванням в режимі апроксимації

| Частота дискретизації | Цикли сигналів | Мінімальна тривалість | Максимальна тривалість | Середня тривалість |
|-----------------------|----------------|-----------------------|------------------------|--------------------|
| 2 кГц                 | 1              | 10,68 мкс             | 19,74 мкс              | 13,235 мкс         |
| 2 кГц                 | 2              | 13,23 мкс             | 23,01 мкс              | 15,026 мкс         |
| 2 кГц                 | 4              | 15,81 мкс             | 20,91 мкс              | 18,302 мкс         |
| 2 кГц                 | 8              | 36,99 мкс             | 56,99 мкс              | 43,951 мкс         |
| 2 кГц                 | 16             | 45,75 мкс             | 49,04 мкс              | 47,289 мкс         |
| 2 кГц                 | 32             | 267,05 мкс            | 338,98 мкс             | 289,956 мкс        |
| 2 кГц                 | 64             | 179,83 мкс            | 402,33 мкс             | 202,231 мкс        |
| 2 кГц                 | 128            | 794,92 мкс            | 868,11 мкс             | 811,146 мкс        |
| 250 кГц               | 1              | 1,552 мс              | 1,987 мс               | 1,644 мс           |
| 250 кГц               | 2              | 1,447 мс              | 1,557 мс               | 1,487 мс           |
| 250 кГц               | 4              | 7,52 мс               | 9,257 мс               | 7,774 мс           |
| 250 кГц               | 8              | 6,463 мс              | 7,661 мс               | 6,743 мс           |
| 250 кГц               | 16             | 25,332 мс             | 26,577 мс              | 25,731 мс          |
| 250 кГц               | 32             | 81,188 мс             | 85,801 мс              | 83,017 мс          |
| 250 кГц               | 64             | 157,261 мс            | 168,155 мс             | 161,507 мс         |
| 250 кГц               | 128            | 275,179 мс            | 282,693 мс             | 278,589 мс         |
| 5 МГц                 | 1              | 21,243 мс             | 24,215 мс              | 21,691 мс          |
| 5 МГц                 | 2              | 86,698 мс             | 93,908 мс              | 89,347 мс          |
| 5 МГц                 | 4              | 153,822 мс            | 161,66 мс              | 156,118 мс         |
| 5 МГц                 | 8              | 677,624 мс            | 744,704 мс             | 693,646 мс         |
| 5 МГц                 | 16             | 523,326 мс            | 542,137 мс             | 530,299 мс         |
| 5 МГц                 | 32             | 2,247 с               | 2,354 с                | 2,288 с            |
| 5 МГц                 | 64             | 2,729 с               | 2,874 с                | 2,769 с            |
| 5 МГц                 | 128            | 12,771 с              | 13,38 с                | 13,058 с           |

Порівнявши ці дві таблиці, можна дійти до висновку, що різниця між часом виконання даного методу для обох режимів планування є дуже малою, але час створення планів для планування в режимі вимірювань є в рази більше. Отже, цей режим недоцільно використовувати для даного методу з вказаними в таблицях конфігураціями.

Тепер варто розглянути тривалість виконання методу частотного домену з плануванням в режимі вимірювання (таблиця 4.4).

Таблиця 4.4

Тривалість виконання методу частотного домену з плануванням в режимі вимірювання

| Частота дискретизації | Цикли сигналів | Мінімальна тривалість | Максимальна тривалість | Середня тривалість |
|-----------------------|----------------|-----------------------|------------------------|--------------------|
| 2 кГц                 | 1              | 16,27 мкс             | 30,95 мкс              | 19,317 мкс         |
| 2 кГц                 | 2              | 13,76 мкс             | 222,73 мкс             | 27,709 мкс         |
| 2 кГц                 | 4              | 17,64 мкс             | 22,82 мкс              | 18,876 мкс         |
| 2 кГц                 | 8              | 24,44 мкс             | 30,06 мкс              | 26,636 мкс         |
| 2 кГц                 | 16             | 56,65 мкс             | 111,36 мкс             | 69,147 мкс         |
| 2 кГц                 | 32             | 85,6 мкс              | 112,46 мкс             | 95,054 мкс         |
| 2 кГц                 | 64             | 119,18 мкс            | 162,17 мкс             | 130,901 мкс        |
| 2 кГц                 | 128            | 208,91 мкс            | 277,85 мкс             | 228,106 мкс        |
| 250 кГц               | 1              | 231,69 мкс            | 316,64 мкс             | 264,143 мкс        |
| 250 кГц               | 2              | 439,56 мкс            | 562,28 мкс             | 474,776 мкс        |
| 250 кГц               | 4              | 925,13 мкс            | 984,05 мкс             | 937,779 мкс        |
| 250 кГц               | 8              | 1,776 мс              | 2,052 мс               | 1,872 мс           |
| 250 кГц               | 16             | 3,849 мс              | 4,334 мс               | 3,924 мс           |
| 250 кГц               | 32             | 7,994 мс              | 8,689 мс               | 8,093 мс           |
| 250 кГц               | 64             | 16,253 мс             | 18,114 мс              | 16,581 мс          |
| 250 кГц               | 128            | 34,646 мс             | 39,084 мс              | 35,226 мс          |
| 5 МГц                 | 1              | 6,511 мс              | 7,034 мс               | 6,659 мс           |
| 5 МГц                 | 2              | 13,49 мс              | 14,765 мс              | 13,728 мс          |
| 5 МГц                 | 4              | 27,148 мс             | 29,515 мс              | 27,366 мс          |
| 5 МГц                 | 8              | 58,545 мс             | 62,543 мс              | 59,519 мс          |
| 5 МГц                 | 16             | 126,219 мс            | 131,751 мс             | 128,358 мс         |
| 5 МГц                 | 32             | 263,242 мс            | 273,937 мс             | 265,977 мс         |
| 5 МГц                 | 64             | 555,483 мс            | 570,966 мс             | 561,882 мс         |
| 5 МГц                 | 128            | 964,305 мс            | 1,001 с                | 975,862 мс         |

Порівнюючи таблицю 4.4 з таблицею 4.2, цей метод має приблизно той самий розподіл часу, що й метод крос-кореляції з плануванням в режимі вимірювання, але є набагато швидшим — визначення різниці фази сигналів на частоті дискретизації 5 МГц з кількістю повторів сигналу 128 в середньому займає 975,862 мілісекунд для даного методу, що є в 13,4 рази швидше, ніж 13,058 секунд, необхідних для методу крос-кореляції.

Тепер варто порівняти тривалості виконання даного методу при плануванні в режимі вимірювання (таблиця 4.4) та режимі апроксимації (таблиця 4.5).

Таблиця 4.5

Тривалість виконання методу частотного домену з плануванням в режимі апроксимації

| Частота дискретизації | Цикли сигналів | Мінімальна тривалість | Максимальна тривалість | Середня тривалість |
|-----------------------|----------------|-----------------------|------------------------|--------------------|
| 2 кГц                 | 1              | 13 мкс                | 35,67 мкс              | 17,148 мкс         |
| 2 кГц                 | 2              | 14,01 мкс             | 275,1 мкс              | 31,052 мкс         |
| 2 кГц                 | 4              | 17,4 мкс              | 22,3 мкс               | 18,615 мкс         |
| 2 кГц                 | 8              | 24,5 мкс              | 33,26 мкс              | 26,725 мкс         |
| 2 кГц                 | 16             | 45,02 мкс             | 48,45 мкс              | 46,206 мкс         |
| 2 кГц                 | 32             | 78,61 мкс             | 286,29 мкс             | 94,771 мкс         |
| 2 кГц                 | 64             | 142,79 мкс            | 198,61 мкс             | 148,973 мкс        |
| 2 кГц                 | 128            | 284,54 мкс            | 312,94 мкс             | 289,124 мкс        |
| 250 кГц               | 1              | 294,38 мкс            | 638,99 мкс             | 320,582 мкс        |
| 250 кГц               | 2              | 579,74 мкс            | 628,91 мкс             | 588,65 мкс         |
| 250 кГц               | 4              | 1,248 мс              | 1,32 мс                | 1,269 мс           |
| 250 кГц               | 8              | 2,489 мс              | 2,729 мс               | 2,563 мс           |
| 250 кГц               | 16             | 5,282 мс              | 5,608 мс               | 5,371 мс           |
| 250 кГц               | 32             | 10,821 мс             | 11,432 мс              | 10,943 мс          |
| 250 кГц               | 64             | 22,265 мс             | 22,448 мс              | 22,364 мс          |
| 250 кГц               | 128            | 46,961 мс             | 50,023 мс              | 47,613 мс          |
| 5 МГц                 | 1              | 6,558 мс              | 6,73 мс                | 6,66 мс            |
| 5 МГц                 | 2              | 13,652 мс             | 14,016 мс              | 13,725 мс          |
| 5 МГц                 | 4              | 28,761 мс             | 29,985 мс              | 28,958 мс          |
| 5 МГц                 | 8              | 60,755 мс             | 63,437 мс              | 61,338 мс          |

| Частота дискретизації | Цикли сигналів | Мінімальна тривалість | Максимальна тривалість | Середня тривалість |
|-----------------------|----------------|-----------------------|------------------------|--------------------|
| 5 МГц                 | 16             | 132,65 мс             | 136,919 мс             | 133,965 мс         |
| 5 МГц                 | 32             | 288,334 мс            | 299,722 мс             | 291,954 мс         |
| 5 МГц                 | 64             | 583,256 мс            | 598,223 мс             | 590,7 мс           |
| 5 МГц                 | 128            | 1,301 с               | 1,34 с                 | 1,319 с            |

Тривалості в таблицях 4.5 та 4.4 є дуже близькими. З цього можна зробити висновок, що в даному випадку планування в режимі вимірювання не робить жоден з методів швидшими. Це може бути через те, що планувальник в режимі апроксимації дуже добре апроксимує ефективний для даної платформи план. Тепер варто розглянути тривалість виконання методу внутрішнього добутку (таблиця 4.6).

Таблиця 4.6

Тривалість виконання методу внутрішнього добутку

| Частота дискретизації | Цикли сигналів | Мінімальна тривалість | Максимальна тривалість | Середня тривалість |
|-----------------------|----------------|-----------------------|------------------------|--------------------|
| 2 кГц                 | 1              | 12,39 мкс             | 145,97 мкс             | 19,268 мкс         |
| 2 кГц                 | 2              | 12,09 мкс             | 27 мкс                 | 14,313 мкс         |
| 2 кГц                 | 4              | 14,94 мкс             | 16,55 мкс              | 15,449 мкс         |
| 2 кГц                 | 8              | 19,15 мкс             | 33,88 мкс              | 21,873 мкс         |
| 2 кГц                 | 16             | 31,35 мкс             | 2,022 мс               | 100,67 мкс         |
| 2 кГц                 | 32             | 58,3 мкс              | 75,57 мкс              | 62,367 мкс         |
| 2 кГц                 | 64             | 104,85 мкс            | 113,32 мкс             | 109,523 мкс        |
| 2 кГц                 | 128            | 206,64 мкс            | 221,05 мкс             | 214,326 мкс        |
| 250 кГц               | 1              | 216,95 мкс            | 275,94 мкс             | 231,021 мкс        |
| 250 кГц               | 2              | 410,73 мкс            | 476,29 мкс             | 428,708 мкс        |
| 250 кГц               | 4              | 825,29 мкс            | 873,8 мкс              | 852,876 мкс        |
| 250 кГц               | 8              | 1,614 мс              | 1,745 мс               | 1,671 мс           |
| 250 кГц               | 16             | 3,268 мс              | 3,502 мс               | 3,387 мс           |
| 250 кГц               | 32             | 6,415 мс              | 6,81 мс                | 6,635 мс           |
| 250 кГц               | 64             | 12,862 мс             | 13,612 мс              | 13,279 мс          |
| 250 кГц               | 128            | 25,687 мс             | 27,177 мс              | 26,55 мс           |
| 5 МГц                 | 1              | 3,974 мс              | 4,259 мс               | 4,108 мс           |

| Частота дискретизації | Цикли сигналів | Мінімальна тривалість | Максимальна тривалість | Середня тривалість |
|-----------------------|----------------|-----------------------|------------------------|--------------------|
| 5 МГц                 | 2              | 7,941 мс              | 8,446 мс               | 8,198 мс           |
| 5 МГц                 | 4              | 15,878 мс             | 16,811 мс              | 16,366 мс          |
| 5 МГц                 | 8              | 31,741 мс             | 33,547 мс              | 32,717 мс          |
| 5 МГц                 | 16             | 63,416 мс             | 66,938 мс              | 65,364 мс          |
| 5 МГц                 | 32             | 126,856 мс            | 133,639 мс             | 130,678 мс         |
| 5 МГц                 | 64             | 253,477 мс            | 267,163 мс             | 261,42 мс          |
| 5 МГц                 | 128            | 507,5 мс              | 535,186 мс             | 523,036 мс         |

Порівнюючи з попередніми таблицями, для малих частот дискретизації (2 кГц) даний метод має приблизно однакову тривалість виконання, що й метод частотного домену, але для високих частот (наприклад, 250 кГц) він стає приблизно вдвічі швидше, ніж метод частотного домену. Отже, з усіх методів це — найшвидший.

## Висновки

В даному розділі було описано програмну реалізацію рішення проблеми синхронізації фаз двох каналів прийому на SDR трансивері AD9361, а також було досліджено швидкість кожного методу при практичному застосуванні, використовуючи програмну реалізацію.

Програмна реалізація складається з трьох частин: бібліотеки phaselib, що містить реалізації всіх трьох методів, описаних у роботі, та допоміжні функції; застосунку phasescorrect, що є демонстрацією коректора фази, що працює на SDR трансивері на базі AD9361; та застосунку phasebench, основною функцією якого це практичне вимірювання швидкості методів визначення різниці фази для різних конфігурацій частоти дискретизації та кількості циклів сигналів, використовуючи різні тестові сигнали.

Результати дослідження швидкості показали, що для даної програмної реалізації, запущеної на процесорі Altera Cyclone V SX, що має ядро процесора ARM Cortex-A9 з тактовою частотою 800 МГц:

- використання планування FFTW в режимі вимірювання не підвищує швидкість методів вимірювання різниці фаз
- метод крос-кореляції є найповільнішим, при високій кількості повторів сигналів він може виконуватися кілька секунд
- метод частотного домену має середню швидкість, в деяких сценаріях він може бути в 13,4 рази швидше
- метод внутрішнього добутку є найшвидшим, для високих частот він стає удвічі швидшим, ніж метод частотного домену

## ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

В даній роботі було досліджено основні проблеми при реалізації систем MIMO з цифровим формуванням променів (beamforming) на базі SDR трансивера Analog Devices AD9361 та рішення даних проблем з особливим акцентом на проблемі синхронізації фаз між каналами передавача, що є проблемою при цифровому формуванні променів через необхідність точної синхронізації фаз між каналами даної системи.

Було взято наявні методи визначення різниці фаз з [1], детально описано принцип їх роботи та особливості реалізації, досліджено дані методи на предмет точності і швидкості та розроблено практичну реалізацію даних методів, що працює безпосередньо на пристрої. Далі будуть описані особливості всіх методів.

Основною перевагою методу крос-кореляції є його помірна завадостійкість, але основними недоліками даного методу є нестабільність завадостійкості, високі вимоги і до частоти дискретизації, і до кількості циклів сигналу та низька швидкість. Проблеми точності можна вирішити, якщо знайти кращий метод інтерполяції затримки.

Основними перевагами методу внутрішнього добутку є мала складність та, відповідно, висока швидкість. Основними недоліками даного методу є низька завадостійкість та робота зі зсувами фази лише між 0 та 180 градусами.

Основною перевагою методу частотного домену (перетворення Фур'є) є хороше співвідношення точності-швидкості та помірна завадостійкість. Основним недоліком даного методу є нестабільність завадостійкості.

Отже, для вирішення проблеми синхронізації фаз двох каналів прийому найбільше підходить метод визначення різниці фаз на основі частотного домену через його завадостійкість.

Цією роботою можна користуватися при проєктуванні систем MIMO з цифровим формуванням променів на основі SDR трансиверів AD9361, а деякі частини цієї роботи, що не залежать від SDR трансивера, як-от методи визначення різниці фаз, можна використовувати і з іншими SDR трансиверами.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Article. I/Q rotation [Електронний ресурс] — Режим доступу до ресурсу: [https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/iq\\_rotation](https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/iq_rotation)
2. Code. iio-oscilloscope/plugins/fmcomms2\_adv.c (р. 517-525) [Електронний ресурс] — Режим доступу до ресурсу: [https://github.com/analogdevicesinc/iio-oscilloscope/blob/56c2ea5/plugins/fmcomms2\\_adv.c](https://github.com/analogdevicesinc/iio-oscilloscope/blob/56c2ea5/plugins/fmcomms2_adv.c)
3. Code. dMaggot/libxcorr [Електронний ресурс] — Режим доступу до ресурсу: <https://github.com/dMaggot/libxcorr>
4. Article. Cross-Correlation [Електронний ресурс] — Режим доступу до ресурсу: <https://mathworld.wolfram.com/Cross-Correlation.html>
5. Knapp C., Carter G. The generalized correlation method for estimation of time delay. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. 1976. Т. 24, № 4. С. 320–327. URL: <https://doi.org/10.1109/tassp.1976.1162830> (дата звернення: 17.06.2026).
6. Article. Hermitian Inner Product [Електронний ресурс] — Режим доступу до ресурсу: <https://mathworld.wolfram.com/HermitianInnerProduct.html>
7. Smith III J. O., Serra X. PARSHL: An Analysis/Synthesis Program for Non-Harmonic Sounds Based on a Sinusoidal Representation. *Proceedings of the International Computer Music Conference (ICMC-87, Tokyo)*. 1987. URL: [https://ccrma.stanford.edu/~jos/parshl/Peak\\_Detection\\_Steps\\_3.html](https://ccrma.stanford.edu/~jos/parshl/Peak_Detection_Steps_3.html).
8. Frigo M., Johnson S. G. The Design and Implementation of FFTW3. *Proceedings of the IEEE*. 2005. Т. 93, № 2. С. 216–231. URL: <https://doi.org/10.1109/jproc.2004.840301> (дата звернення: 17.06.2026).
9. Article. Fourier Transform [Електронний ресурс] — Режим доступу до ресурсу: <https://mathworld.wolfram.com/FourierTransform.html>
10. Article. Installation on UNIX (FFTW 3.3.11) [Електронний ресурс] — Режим доступу до ресурсу: [https://fftw.org/fftw3\\_doc/Installation-on-Unix.html](https://fftw.org/fftw3_doc/Installation-on-Unix.html)
11. Altera Corporation. Cyclone V Hard Processor System Technical Reference Manual. Версія «21.2». 2022. Розділ «2.2.2. Cortex-A9 MPCore»

## ДОДАТОК А. Опис експериментального стенду для досліджень

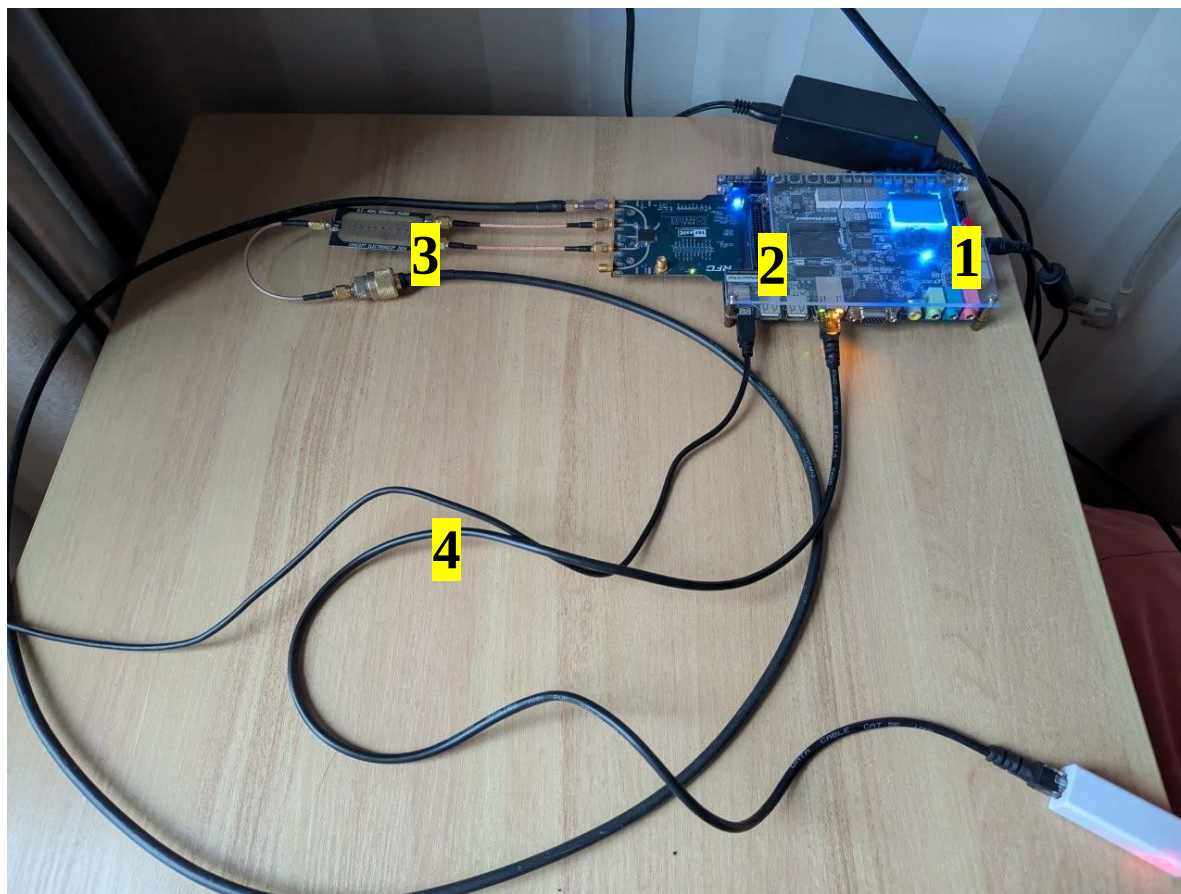


Фото експериментального стенду для досліджень

Для проведення практичних досліджень в роботі було використано плату розробки Terasic DE10-Standard (1 на фото) на базі SoCFPGA-процесора Altera Cyclone V SX, що має два ядра ARM Cortex-A9 з номінальною тактовою частотою 925 МГц. До даної платформи підключено HSMC-модуль Terasic Arradio з SDR трансивером Analog Devices AD9361 (2 на фото). Плата розробки працює з ОС на базі Ubuntu 16.04, модифікованої для роботи з SDR трансивером AD9361. Тактова частота процесора в системі становить 800 МГц.

Конфігурація з'єднань модуля Arradio наступна: конектор TX2A (другого каналу передачі) під'єднаний кабелем до входу розгалужувача Вілкінсона (3 на фото), виходи розгалужувача під'єднані кабелями до конекторів RX1A та RX2A.

Плата розробки під'єднана до робочої станції через USB-порт для консолі плати (кабель: 4 на фото) та через Ethernet-інтерфейс USB-адаптером (5 на фото).

## ДОДАТОК Б. Код для моделювання методів у MATLAB

### Файл phase\_diff\_xcorr.m

```
function phd = phase_diff_xcorr(a, b, period)
    [xc, lags] = xcorr(a, b);
    xc = real(xc);

    [~, max_lag] = max(xc);

    lag_idx = max_lag;

    % Copied from "Quadratic Interpolation Spectral Peaks"
    % by Julius O. Smith III

    % Get variables for interpolation
    % (replace some of them with zero)
    if (lag_idx - 1) > 0
        alpha = xc(lag_idx - 1);
    else
        alpha = 0;
    end

    if (lag_idx + 1) <= length(xc)
        gamma = xc(lag_idx + 1);
    else
        gamma = 0;
    end

    beta = xc(lag_idx);

    interp_lag = lags(lag_idx) + ((alpha - gamma) / (2 * (alpha - 2 * beta +
gamma)));

    % Copied from Analog Devices' IIO Oscilloscope application
    phd = scale_angle((2 * pi) * (interp_lag / period));
end
```

### Файл phase\_diff\_inner\_product.m

```
function phd = phase_diff_inner_product(a, b)
    % wrong and broken
    % cphd = acos(sum(a .* conj(b)) / sqrt(sum(abs(a) .^ 2) * sum(abs(b) .^ 2)));
    % phd = real(cphd) + imag(cphd); % I have no idea why but this works

    phd = acos(real(sum(a .* conj(b))) / sqrt(sum(abs(a) .^ 2) * sum(abs(b) .^
2)));
end
```

### Файл phase\_diff\_fft.m

```
function phd = phase_diff_fft(a, b, N)
    if size(a) ~= size(b)
        error("Sizes of both signals must be the same")
    end

    fft1 = fft(a, N);
    fft2 = fft(b, N);

    max_amp = -inf;
    max_idx = -1;
    for i = 1:N
        amp1 = abs(fft1(i));
        if (amp1 / abs(fft2(i))) > 0.5
            if (amp1 > max_amp)
                max_amp = amp1;
                max_idx = i;
            end
        end
    end

    if max_idx == -1
        phd = NaN;
        return
    end

    if real(fft1(max_idx)) < 0 && imag(fft1(max_idx)) > 0
        phd = angle(fft1(max_idx)) - angle(fft2(max_idx));
    else
        phd = angle(fft2(max_idx)) - angle(fft1(max_idx));
    end
end
```

### Файл **broad\_parameter\_analysis.m**

```
f = [1e3 2.5e3 5e3 10e3 25e3 50e3 100e3 250e3 500e3 1e6 2.5e6];
fs = [2e3 2.5e3 5e3 10e3 25e3 50e3 100e3 250e3 500e3 1e6 2.5e6 5e6];
repeat = 2.^(0:7);
pho = deg2rad([-180 -90 -45 -30 0 30 45 90 180]);
snr = [-10 -8 -4 -2 -1 0 10 20 40 Inf];
method = ["xcorr" "fft" "inner_product"];

cs = multi_property_analysis(f, fs, repeat, pho, snr, method);
save("bpa.mat", "cs")
```

### Файл **scale\_angle.m**

```
function scaled = scale_angle(unscaled)
    if length(unscaled) > 1
        scaled = zeros(length(unscaled), 1);
        for i = 1:length(unscaled)
            scaled(i) = scale_angle(unscaled(i));
        end
    else
        unscaled = mod(unscaled, 2*pi);
        if unscaled > pi
            scaled = unscaled - 2*pi;
        elseif unscaled < -pi
            scaled = unscaled + 2*pi;
        else
            scaled = unscaled;
        end
    end
end
```

### Файл **dds\_cosine.m**

```
function [s, t] = dds_cosine(f, f_s, repeat, pho)
    t = 0:(1/f_s):(repeat/f - 1/f_s);
    a = 2 * pi * f * t + pho;
    s = cos(a) + 1j * sin(a);
end
```

### Файл multi\_property\_analysis.m

```
function cs = multi_property_analysis(f, fs, repeat, pho, snr, method)
    cs = combinations(f, fs, repeat, pho, snr, method);
    cs = cs(cs.fs >= (2 * cs.f), :);
    cs = cs(cs.method ~= "inner_product" | cs.pho >= 0, :);
    cs.phm = zeros(1, height(cs));

    for ci = 1:height(cs)
        c = cs(ci, :);

        [a, ~] = dds_cosine(c.f, c.fs, c.repeat, 0);
        [b, ~] = dds_cosine(c.f, c.fs, c.repeat, c.pho);

        if c.snr ~= Inf
            a = awgn(a, c.snr, 0, 0);
            b = awgn(b, c.snr, 0, 1);
        end

        if c.method == "xcorr"
            cs.phm(ci) = phase_diff_xcorr(a, b, floor(c.fs / c.f));
        elseif c.method == "inner_product"
            cs.phm(ci) = phase_diff_inner_product(a, b);
        elseif c.method == "fft"
            cs.phm(ci) = phase_diff_fft(a, b, length(a));
        else
            error("Unknown method")
        end
    end

    cs.phe = abs(scale_angle(cs.pho - cs.phm));

    cs.phod = rad2deg(cs.pho);
    cs.phmd = rad2deg(cs.phm);
    cs.phed = rad2deg(cs.phe);
end
```

### Файл xcorr\_fs\_precision.m

```

farr = [1e3 2.5e3 5e3 10e3 25e3 100e3 250e3];
repeats = 2.^(0:7);
pho = deg2rad(30);

for i = 1:length(farr)
    f = farr(i);
    disp("Processing " + num2str(f))

    fstart = f*2;
    fend = min(f*20, 5e6);
    fsarr = fstart:(f/20):fend;
    errs = zeros(length(fsarr), length(repeats));

    for j = 1:length(fsarr)
        fs = fsarr(j);

        for k = 1:length(repeats)
            repeat = repeats(k);

            [a, ta] = dds_cosine(f, fs, repeat, 0);
            [b, tb] = dds_cosine(f, fs, repeat, pho);

            errs(j, k) = abs(scale_angle(pho - phase_diff_xcorr(a, b, fs / f)));
        end
    end

    clf
    ax = gca;
    if fstart >= 1e5
        ax.XAxis.Exponent = 6;
    else
        ax.XAxis.Exponent = 3;
    end
    colororder("gem12")
    xlabel('Частота дискретизації, Гц')
    ylabel('Абсолютна похибка, градус')
    hold on
    for k = 1:length(repeats)
        plot(fsarr, rad2deg(errs(:, k)));
    end
    hold off
    grid on
    ylim([0 rad2deg(pho/2)])
    xticks(linspace(fstart, fend, 10))
    yticks([0 0.5 1 2 5 10 rad2deg(pho/2)])
    legend(repeats + " цикл.")
    savefig("graphs/xcorr_fs_precision/" + num2str(farr(i)) + ".fig")
end

```

### Файл `xcorr_noise_precision.m`

```

f = 1e3;
fsarr = [10e3 50e3 100e3 500e3 1e6 5e6];
fsarrl = ["10 кГц" "50 кГц" "100 кГц" "500 кГц" "1 МГц" "5 МГц"];
pho = deg2rad(30);
repeats = 2.^(0:7);
snrs = -10:40;

for k = 1:length(repeats)
    repeat = repeats(k);
    errs = zeros(length(snrs), length(fsarr));
    for i = 1:length(snrs)
        snr = snrs(i);
        disp("Processing " + num2str(snr))

        for j = 1:length(fsarr)
            fs = fsarr(j);

            [a, ta] = dds_cosine(f, fs, repeat, 0);
            [b, tb] = dds_cosine(f, fs, repeat, pho);

            a_noisy = awgn(a, snr, 0, 0);
            b_noisy = awgn(b, snr, 0, 1);

            errs(i,j) = abs(scale_angle(pho - phase_diff_xcorr(a_noisy, b_noisy,
fs / f)));
        end
    end

    clf
    xlabel('Відношення сигнал-шум, дБ')
    ylabel('Абсолютна похибка, градус')
    hold on
    for i = 1:length(fsarr)
        plot(snrs, rad2deg(errs(:, i)));
    end
    hold off
    grid on
    colororder("gem12")
    maxy = max(max(rad2deg(errs)));
    disp(maxy)
    if maxy >= rad2deg(pho)
        ylim([0 rad2deg(pho)])
        yticks([0 2 5 rad2deg(pho)/2 rad2deg(pho)])
    end
    yline(1, '- ', '1 градус')
    legend(fsarrl)
    savefig("graphs/xcorr_noise_precision/" + repeat + ".fig")
end

```

### Файл inner\_product\_noise\_precision.m

```

f = 1e3;
fsarr = [2e3 10e3 50e3 100e3 500e3 1e6 5e6];
pho = deg2rad(30);

snrs = -10:40;
repeats = 2.^(0:7);
repeatsl = repeats + " цикл.";

errs = zeros(length(snrs), length(fsarr));

for k = 1:length(fsarr)
    fs = fsarr(k);

    for i = 1:length(snrs)
        snr = snrs(i);
        disp("Processing " + num2str(snr))

        for j = 1:length(repeats)
            repeat = repeats(j);

            [a, ta] = dds_cosine(f, fs, repeat, 0);
            [b, tb] = dds_cosine(f, fs, repeat, pho);

            a_noisy = awgn(a, snr, 0, 0);
            b_noisy = awgn(b, snr, 0, 1);

            errs(i,j) = abs(scale_angle(pho - phase_diff_inner_product(a_noisy,
b_noisy)));
        end
    end

    clf
    colororder("gem12")
    xlabel('Відношення сигнал-шум, дБ')
    ylabel('Абсолютна похибка, градус')
    hold on
    for i = 1:length(repeats)
        plot(snrs, rad2deg(errs(:, i)));
    end

    hold off
    grid on
    ylim([0 rad2deg(pho)])
    yticks([0 2 5 rad2deg(pho)/2 rad2deg(pho)])
    yline(1, '-', '1 градус')
    legend(repeatsl)
    savefig("graphs/inner_product_noise_precision/" + fs + ".fig")
end

```

### Файл `fft_noise_precision.m`

```
f = 1e3;
fsarr = [2e3 10e3 50e3 100e3 500e3 1e6 5e6];
fsarrl = ["2 кГц" "10 кГц" "50 кГц" "100 кГц" "500 кГц" "1 МГц" "5 МГц"];
pho = deg2rad(30);

snrs = -60:40;
repeats = 2.^(0:7);

errs = zeros(length(snrs), length(fsarr));

for k = 1:length(repeats)
    repeat = repeats(k);

    for i = 1:length(snrs)
        snr = snrs(i);
        disp("Processing " + num2str(snr))

        for j = 1:length(fsarr)
            fs = fsarr(j);

            [a, ta] = dds_cosine(f, fs, repeat, 0);
            [b, tb] = dds_cosine(f, fs, repeat, pho);

            a_noisy = awgn(a, snr, 0, 0);
            b_noisy = awgn(b, snr, 0, 1);

            errs(i,j) = abs(scale_angle(pho - phase_diff_fft(a_noisy, b_noisy,
length(a))));
        end
    end

    clf
    colororder("gem12")
    xlabel('Відношення сигнал-шум, дБ')
    ylabel('Абсолютна похибка, градус')
    hold on

    for i = 1:length(fsarr)
        plot(snrs, rad2deg(errs(:, i)));
    end

    hold off
    grid on
    ylim([0 rad2deg(pho)])
    legend(fsarrl)
    yline(1, '-', '1 градус')
    savefig("graphs/fft_noise_precision_2/" + repeat + ".fig")
end
```

## ДОДАТОК В. Код практичної реалізації на C

### Файл meson.build

```
project('phasebench-x', 'c', default_options: {
    'c_args': ['-D_GNU_SOURCE'],
    'c_std': 'c11'
})

cc = meson.get_compiler('c')
libm = cc.find_library('m', required: true)
libiio = dependency('libiio', version: '<1')
fftw = dependency('fftw3', static: true)

executable('phasescorrect', 'phasescorrect.c', 'phaseslib.c', 'utils.c', 'iio-
utils.c', 'proplist.c', dependencies: [libm, libiio, fftw])
executable('phasebench', 'phasebench.c', 'phaseslib.c', 'utils.c', dependencies:
[libm, fftw])
```

### Файл iio-utils.h

```
#pragma once

#include "utils.h"

#define print_iio_err(err, fmt, ...) fprintf(stderr, fmt ": %s\n", ##__VA_ARGS__,
utils_iio_strerror(err))
#define print_iio_eno(fmt, ...) print_iio_err(errno, fmt, ##__VA_ARGS__)
const char *utils_iio_strerror(int err);
```

### Файл iio-utils.c

```
#include "iio-utils.h"

#include <iio.h>

#define ERR_BUF_LENGTH 256

static char err_buf[ERR_BUF_LENGTH];

const char *utils_iio_strerror(int err)
{
    iio_strerror(err, err_buf, ERR_BUF_LENGTH);
    return err_buf;
}
```

### Файл utils.h

```
#pragma once

#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <stdint.h>
#include <complex.h>

#define print_err(fmt, ...) fprintf(stderr, fmt "\n", ##__VA_ARGS__)
#define print_eno(fmt, ...) fprintf(stderr, fmt ": %s\n", ##__VA_ARGS__,
strerror(errno))
```

```

#define length_of(x) (sizeof (x) / sizeof ((x)[0]))
#define min(a, b) ((a) < (b) ? (a) : (b))

#define STRINGIFY(x) #x

void bench_start();
double bench_end();

int dump_buffer_to_csv(double complex *buf, int n, const char *path);

static inline void unreachable(const char *id)
{
    print_err("Unreachable! at %s", id);
    exit(1);
}

static inline double deg2rad(double deg)
{
    return deg * (M_PI / 180.0);
}

static inline double rad2deg(double rad)
{
    return rad * (180.0 / M_PI);
}

```

### Файл proplist.h

```

#pragma once

#include <iio.h>

enum prop_target_type {
    PROP_TARGET_INVALID,
    PROP_TARGET_DEVICE,
    PROP_TARGET_CHANNEL,
};

enum prop_value_type {
    PROP_TYPE_STRING,
    PROP_TYPE_LONGLONG,
    PROP_TYPE_DOUBLE,
    PROP_TYPE_BOOL
};

struct prop_target {
    enum prop_target_type type;
    union {
        struct iio_device *device;
        struct iio_channel *channel;
    };
};

struct prop_value {
    enum prop_value_type type;
    union {
        const char *string;
        long long longlong;
        double fp_double;
        bool logic_bool;
    };
};

```

```

struct prop {
    struct prop_target target;
    const char *attr;
    struct prop_value value;
};

#define PROP_END { .target = { .type = PROP_TARGET_INVALID } }

#define TARGET_DEVICE(dev) { .type = PROP_TARGET_DEVICE, .device = dev }
#define TARGET_CHANNEL(chan) { .type = PROP_TARGET_CHANNEL, .channel = chan }

#define VALUE_STRING(v) { .type = PROP_TYPE_STRING, .string = v }
#define VALUE_LONGLONG(v) { .type = PROP_TYPE_LONGLONG, .longlong = v }
#define VALUE_DOUBLE(v) { .type = PROP_TYPE_DOUBLE, .fp_double = v }
#define VALUE_BOOL(v) { .type = PROP_TYPE_BOOL, .logic_bool = v }

#define PROP(tgt, att, val) { .target = tgt, .attr = att, .value = val }

int proplist_apply(struct prop *list);

```

### Файл utils.c

```

#include "utils.h"

#include <stdbool.h>
#include <stdint.h>
#include <time.h>
#include <inttypes.h>

struct bench {
    bool is_ongoing;
    struct timespec t1;
    struct timespec t2;
    struct timespec tdiff;
};

static struct bench bench_info = {0};

void bench_start(const char *id)
{
    if (bench_info.is_ongoing) {
        print_err("Another benchmark already ongoing");
        return;
    }

    bench_info.is_ongoing = true;

    if (clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &bench_info.t1) != 0) {
        print_eno("Failed to get bench timestamp 1");
        return;
    }
}

double bench_end()
{
    if (!bench_info.is_ongoing) {
        print_err("No bench is currently ongoing");
        return NAN;
    }

    if (clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &bench_info.t2) != 0) {

```

```

        print_eno("Failed to get bench timestamp 2");
        return NAN;
    }

    /* from https://stackoverflow.com/a/68804612 */
    struct timespec tdiff;

    tdiff.tv_sec = bench_info.t2.tv_sec - bench_info.t1.tv_sec;
    tdiff.tv_nsec = bench_info.t2.tv_nsec - bench_info.t1.tv_nsec;

    if (tdiff.tv_nsec < 0) {
        tdiff.tv_nsec += 1000000000;
        tdiff.tv_sec -= 1;
    }

    bench_info.is_ongoing = false;

    return (tdiff.tv_nsec / 1000.0) + (tdiff.tv_sec * 1000000.0);
}

int dump_buffer_to_csv(double complex *buf, int n, const char *path)
{
    FILE *file = fopen(path, "w");
    if (file == NULL) {
        print_eno("Failed to open file at '%s' for dumping", path);
        return -1;
    }

    for (int i = 0; i < n; i += 1) {
        fprintf(file, "%f%s%fj\n", creal(buf[i]), cimag(buf[i]) >= 0 ? "+" : "",
cimag(buf[i]));
    }

    fclose(file);
    return 0;
}

```

## Файл proplist.c

```

#include "proplist.h"

#include <stdio.h>

#include <iio.h>

#include "utils.h"
#include "iio-utils.h"

#define CONV_BUF_LENGTH 32
static char conv_buf[CONV_BUF_LENGTH];

static const char *resolve_dev_name_for_chan(struct iio_channel *chan)
{
    const struct iio_device *dev = iio_channel_get_device(chan);
    return iio_device_get_name(dev);
}

static const char *resolve_value(struct prop_value *val)
{
    if (val->type == PROP_TYPE_STRING) {
        return val->string;
    } else if (val->type == PROP_TYPE_BOOL) {

```

```

        return val->logic_bool ? "1" : "0";
    } else {
        switch (val->type) {
            case PROP_TYPE_LONGLONG:
                snprintf(conv_buf, CONV_BUF_LENGTH, "%lld", val->longlong);
                break;

            case PROP_TYPE_DOUBLE:
                snprintf(conv_buf, CONV_BUF_LENGTH, "%.06f", val->fp_double);
                break;

            default:
                unreachable("proplist resolve_value");
        }

        return conv_buf;
    }
}

static int write_dev_attr(struct iio_device *dev, const char *attr, const char
*val)
{
    int result;

    const char *name = iio_device_get_name(dev);

    printf("[DEBUG] Set attribute '%s' on device '%s' to '%s'\n", attr, name,
val);

    result = iio_device_attr_write(dev, attr, val);
    if (result < 0) {
        print_iio_err(result, "Failed to set attribute '%s' on device '%s'", attr,
name);
        return -1;
    }

    return 0;
}

static int write_chan_attr(struct iio_channel *chan, const char *attr, const char
*val)
{
    int result;

    const char *name = iio_channel_get_id(chan);
    const char *dev_name = resolve_dev_name_for_chan(chan);

    printf("[DEBUG] Set attribute '%s' on channel '%s' of device '%s' to '%s'\n",
attr, name, dev_name, val);

    result = iio_channel_attr_write(chan, attr, val);
    if (result < 0) {
        print_iio_err(result, "Failed to set attribute '%s' on channel '%s' of
device '%s'", attr, name, dev_name);
        return -1;
    }

    return 0;
}

int proplist_apply(struct prop *list)
{

```

```

    int result;

    for (struct prop *curr = list; curr->target.type != PROP_TARGET_INVALID; curr
    += 1) {
        const char *value = resolve_value(&curr->value);

        switch (curr->target.type) {
            case PROP_TARGET_DEVICE:
                result = write_dev_attr(curr->target.device, curr->attr, value);
                break;

            case PROP_TARGET_CHANNEL:
                result = write_chan_attr(curr->target.channel, curr->attr, value);
                break;

            default:
                unreachable("proplist_apply target type check");
        }

        if (result != 0) {
            print_err("Failed to apply proplist");
            return -1;
        }
    }

    return 0;
}

```

## Файл phaselib.h

```

#pragma once

#include <stdbool.h>
#include <complex.h>

void generate_dds_monotone(double complex *buf, int n, int f_t, int f_s, double
phase);

int calc_period_in_samples(int f_t, int f_s);

struct phase_diff_xcorr_plan;
struct phase_diff_xcorr_plan *phase_diff_xcorr_plan(double complex *a, double
complex *b, int n, bool do_estimate);
double phase_diff_xcorr(struct phase_diff_xcorr_plan *plan, int period_sps);

double phase_diff_inner_product(double complex *a, double complex *b, int n);

struct phase_diff_fft_plan;
struct phase_diff_fft_plan *phase_diff_fft_plan(double complex *a, double complex
*b, int n, bool do_estimate);
double phase_diff_fft(struct phase_diff_fft_plan *plan);

```

## Файл phaselib.c

```

#include "phaselib.h"

#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <stdbool.h>

#include <fftw3.h>

```

```

#include "utils.h"

void generate_dds_monotone(double complex *buf, int n, int f_t, int f_s, double
phase)
{
    double w = 2 * M_PI * f_t;

    for (int i = 0; i < n; i += 1) {
        double t = (double) i / f_s;
        double a = w * t + phase;
        buf[i] = cos(a) + I * sin(a);
    }
}

int calc_period_in_samples(int f_t, int f_s)
{
    return f_s / f_t;
}

struct phase_diff_xcorr_plan {
    double complex *a;
    double complex *b;

    double complex *a_ext;
    double complex *b_ext;

    double complex *a_fft;
    double complex *b_fft;

    double complex *xcorr_coeffs;
    double complex *xcorr;

    int n;
    int m;

    fftw_plan fft_plan_fwd;
    fftw_plan fft_plan_coeffs2corr;
};

struct phase_diff_xcorr_plan *phase_diff_xcorr_plan(double complex *a, double
complex *b, int n, bool do_estimate)
{
    struct phase_diff_xcorr_plan *plan = malloc(sizeof (struct
phase_diff_xcorr_plan));
    if (plan == NULL) {
        goto err_plan;
    }

    plan->a = a;
    plan->b = b;
    plan->n = n;

    int m = (2 * n) - 1;
    plan->m = m;

    plan->a_ext = fftw_malloc(sizeof (double complex) * m);
    plan->b_ext = fftw_malloc(sizeof (double complex) * m);
    if (plan->a_ext == NULL || plan->b_ext == NULL) {
        print_eno("Failed to allocate space for signal extension buffers");
        goto err_ext_bufs;
    }
}

```

```

/* we only need to do a single memset here
 * because the value of m is never changed afterwards
 * and it's always bigger than the buffers we process */
memset(plan->a_ext, 0, sizeof (double complex) * plan->m);
memset(plan->b_ext, 0, sizeof (double complex) * plan->m);

plan->a_fft = fftw_malloc(sizeof (double complex) * m);
plan->b_fft = fftw_malloc(sizeof (double complex) * m);
if (plan->a_fft == NULL || plan->b_fft == NULL) {
    print_eno("Failed to allocate space for FFT buffers");
    goto err_fft_bufs;
}

plan->xcorr_coefs = fftw_malloc(sizeof (double complex) * m);
if (plan->xcorr_coefs == NULL) {
    print_eno("Failed to allocate space for cross correlation coefficients");
    goto err_xcorr_coefs;
}

plan->xcorr = fftw_malloc(sizeof (double complex) * m);
if (plan->xcorr == NULL) {
    print_eno("Failed to allocate space for cross correlation");
    goto err_xcorr;
}

plan->fft_plan_fwd = fftw_plan_dft_1d(m, plan->a_ext, plan->a_fft,
FFTW_FORWARD, do_estimate ? FFTW_ESTIMATE : FFTW_MEASURE);
plan->fft_plan_coefs2corr = fftw_plan_dft_1d(m, plan->xcorr_coefs, plan-
>xcorr, FFTW_BACKWARD, do_estimate ? FFTW_ESTIMATE : FFTW_MEASURE);

return plan;

fftw_free(plan->xcorr);
err_xcorr:
fftw_free(plan->xcorr_coefs);
err_xcorr_coefs:
fftw_free(plan->a_fft);
fftw_free(plan->b_fft);
err_fft_bufs:
fftw_free(plan->a_ext);
fftw_free(plan->b_ext);
err_ext_bufs:
free(plan);
err_plan:
return NULL;
}

double phase_diff_xcorr(struct phase_diff_xcorr_plan *plan, int period_sps)
{
    /* perform the cross correlation */
    memcpy(plan->a_ext + (plan->n - 1), plan->a, sizeof (double complex) * plan-
>n);
    memcpy(plan->b_ext, plan->b, sizeof (double complex) * (plan->n - 1));

    fftw_execute_dft(plan->fft_plan_fwd, plan->a_ext, plan->a_fft);
    fftw_execute_dft(plan->fft_plan_fwd, plan->b_ext, plan->b_fft);

    for (int i = 0; i < plan->m; i += 1) {
        plan->xcorr_coefs[i] = plan->a_fft[i] * conj(plan->b_fft[i]);
    }

    fftw_execute(plan->fft_plan_coefs2corr);

```

```

/* find the peak */
double max_mag = -INFINITY;
int max_idx = -1;

for (int i = 0; i < plan->m; i += 1) {
    double mag = creal(plan->xcorr[i]);
    if (mag >= max_mag) {
        max_mag = mag;
        max_idx = i;
    }
}

/* compute the lag from it */
int max_lag = max_idx - plan->n;

/* interpolate */
double alpha = max_idx >= 1 ? creal(plan->xcorr[max_idx - 1]) : 0;
double beta = creal(plan->xcorr[max_idx]);
double gamma = max_idx < (plan->m - 1) ? creal(plan->xcorr[max_idx + 1]) : 0;

double interp_lag = max_lag + (alpha - gamma) / (2 * (alpha - 2 * beta +
gamma));

/* find the phase using interpolated lag */
double phase = (2 * M_PI) * (interp_lag / period_sps);

return phase;
}

double phase_diff_inner_product(double complex *a, double complex *b, int n)
{
    double complex num = 0;
    double denom_x = 0;
    double denom_y = 0;

    for (int i = 0; i < n; i += 1) {
        num += a[i] * conj(b[i]);
        denom_x += pow(cabs(a[i]), 2);
        denom_y += pow(cabs(b[i]), 2);
    }

    return acos(creal(num) / sqrt(denom_x * denom_y));
}

struct phase_diff_fft_plan {
    double complex *a;
    double complex *b;

    double complex *a_fft;
    double complex *b_fft;

    int n;
    fftw_plan fft_plan;
};

struct phase_diff_fft_plan *phase_diff_fft_plan(double complex *a, double complex
*b, int n, bool do_estimate)
{
    struct phase_diff_fft_plan *plan = malloc(sizeof (struct
phase_diff_fft_plan));
    if (plan == NULL) {

```

```

        goto err_plan;
    }

    plan->a = a;
    plan->b = b;
    plan->n = n;

    plan->a_fft = fftw_malloc(sizeof (double complex) * n);
    plan->b_fft = fftw_malloc(sizeof (double complex) * n);
    if (plan->a_fft == NULL || plan->b_fft == NULL) {
        print_eno("Failed to allocate space for FFT buffers");
        goto err_fft_bufs;
    }

    plan->fft_plan = fftw_plan_dft_1d(n, a, plan->a_fft, FFTW_FORWARD, do_estimate
? FFTW_ESTIMATE : FFTW_MEASURE);

    return plan;

    fftw_free(plan->a_fft);
    fftw_free(plan->b_fft);
err_fft_bufs:
    free(plan);
err_plan:
    return NULL;
}

double phase_diff_fft(struct phase_diff_fft_plan *plan)
{
    fftw_execute_dft(plan->fft_plan, plan->a, plan->a_fft);
    fftw_execute_dft(plan->fft_plan, plan->b, plan->b_fft);

    double max_abs = -INFINITY;
    int idx = -1;

    for (int i = 0; i < plan->n; i += 1) {
        double a_abs = cabs(plan->a_fft[i]);

        if ((a_abs / cabs(plan->b_fft[i])) > 0.5) {
            if (a_abs > max_abs) {
                max_abs = a_abs;
                idx = i;
            }
        }
    }

    if (idx == -1) {
        print_err("[FFT] No frequency components in common");
        return NAN;
    }

    if (creal(plan->a_fft[idx]) < 0 && cimag(plan->a_fft[idx]) > 0) {
        return carg(plan->a_fft[idx]) - carg(plan->b_fft[idx]);
    } else {
        return carg(plan->b_fft[idx]) - carg(plan->a_fft[idx]);
    }
}

```

### Файл phasecorrect.c

```

#include <stdio.h>
#include <errno.h>

```

```

#include <string.h>
#include <stdint.h>
#include <inttypes.h>
#include <complex.h>
#include <unistd.h>

#include <iio.h>
#include <fftw3.h>

#include "utils.h"
#include "iio-utils.h"
#include "proplist.h"
#include "phaselib.h"

#define KHZ ((long long) 1000)
#define MHZ (KHZ*1000)
#define GHZ (MHZ*1000)

#define F_T (1 * KHZ)
#define REPEAT 8

#define AD9361_PHY_DEV "ad9361-phy"
#define AD9361_RX_DEV  "cf-ad9361-lpc"
#define AD9361_TX_DEV  "cf-ad9361-dds-core-lpc"
#define AD9361_RX_LO_CHAN "RX_LO"
#define AD9361_TX_LO_CHAN "TX_LO"
#define AD9361_RX_CHANS 2
#define AD9361_TX_CHANS 2
#define AD9361_DDS_FREQS 2

static const char *ad9361_phy_rx_chans[AD9361_RX_CHANS] = {
    "voltage0",
    "voltage1"
};

static const char *ad9361_phy_tx_chans[AD9361_TX_CHANS] = {
    "voltage0",
    "voltage1"
};

static const char *ad9361_rx_chans[AD9361_RX_CHANS][2] = {
    { "voltage0", "voltage1" },
    { "voltage2", "voltage3" },
};

static struct iio_device *phy_dev;
static struct iio_device *rx_dev;
static struct iio_device *tx_dev;

static struct iio_channel *rx_lo;
static struct iio_channel *tx_lo;

static struct iio_channel *phy_rx_chans[AD9361_RX_CHANS];
static struct iio_channel *phy_tx_chans[AD9361_TX_CHANS];

static struct iio_channel *rx_chans[AD9361_RX_CHANS][2];
static struct iio_channel *dds_chans[AD9361_TX_CHANS][2][AD9361_DDS_FREQS];

static int setup_ad9361_devs(struct iio_context *ctx)
{
    phy_dev = iio_context_find_device(ctx, AD9361_PHY_DEV);
    if (phy_dev == NULL) {

```

```

    print_iiio_eno("Failed to find AD9361 PHY device");
    return -1;
}

rx_dev = iio_context_find_device(ctx, AD9361_RX_DEV);
if (rx_dev == NULL) {
    print_iiio_eno("Failed to find AD9361 RX device");
    return -1;
}

tx_dev = iio_context_find_device(ctx, AD9361_TX_DEV);
if (tx_dev == NULL) {
    print_iiio_eno("Failed to find AD9361 TX device");
    return -1;
}

rx_lo = iio_device_find_channel(phy_dev, AD9361_RX_LO_CHAN, true);
if (rx_lo == NULL) {
    print_iiio_eno("Failed to find AD9361 RX LO");
    return -1;
}

tx_lo = iio_device_find_channel(phy_dev, AD9361_TX_LO_CHAN, true);
if (tx_lo == NULL) {
    print_iiio_eno("Failed to find AD9361 TX LO");
    return -1;
}

for (int i = 0; i < AD9361_RX_CHANS; i += 1) {
    phy_rx_chans[i] = iio_device_find_channel(phy_dev, ad9361_phy_rx_chans[i],
false);
    if (phy_rx_chans[i] == NULL) {
        print_iiio_eno("Failed to find AD9361 PHY RX channel %d", i);
        return -1;
    }
}

for (int i = 0; i < AD9361_TX_CHANS; i += 1) {
    phy_tx_chans[i] = iio_device_find_channel(phy_dev, ad9361_phy_tx_chans[i],
true);
    if (phy_tx_chans[i] == NULL) {
        print_iiio_eno("Failed to find AD9361 PHY TX channel %d", i);
        return -1;
    }
}

#define DDS_CHAN_NAME_LEN 16
char dds_chan_name[DDS_CHAN_NAME_LEN];

for (int chan = 0; chan < AD9361_TX_CHANS; chan += 1) {
    for (int comp = 0; comp < 2; comp += 1) {
        for (int freq = 0; freq < AD9361_DDS_FREQS; freq += 1) {
            snprintf(dds_chan_name, DDS_CHAN_NAME_LEN, "TX%d_%c_F%d", chan +
1, comp == 0 ? 'I' : 'Q', freq + 1);

            dds_chans[chan][comp][freq] = iio_device_find_channel(tx_dev,
dds_chan_name, true);
            if (dds_chans[chan][comp][freq] == NULL) {
                print_iiio_eno("Failed to find AD9361 TX DDS channel '%s'",
dds_chan_name);
                return -1;
            }
        }
    }
}

```

```

    }
}

for (int chan = 0; chan < AD9361_RX_CHANS; chan += 1) {
    for (int comp = 0; comp < 2; comp += 1) {
        rx_chans[chan][comp] = iio_device_find_channel(rx_dev,
ad9361_rx_chans[chan][comp], false);
        if (rx_chans[chan][comp] == NULL) {
            print_iio_eno("Failed to find AD9361 RX channel %d for component
%c", chan, comp == 0 ? 'I' : 'Q');
            return -1;
        }
    }
}

return 0;
}

static int setup_ad9361_phy()
{
    struct prop phy_proplist[] = {
        PROP(TARGET_DEVICE(phy_dev), "ensm_mode", VALUE_STRING("fdd")),
        PROP_END,
    };

    return proplist_apply(phy_proplist);
}

static int setup_ad9361_tx(long long lo_freq, long long sample_freq, long long
rf_bw, double tx_gain)
{
    struct prop tx_proplist[] = {
        PROP(TARGET_CHANNEL(tx_lo), "frequency", VALUE_LONGLONG(lo_freq)),

        PROP(TARGET_CHANNEL(phy_tx_chans[0]), "rf_bandwidth",
VALUE_LONGLONG(rf_bw)),
        PROP(TARGET_CHANNEL(phy_tx_chans[0]), "sampling_frequency",
VALUE_LONGLONG(sample_freq)),

        PROP(TARGET_CHANNEL(phy_tx_chans[0]), "hardwaregain",
VALUE_DOUBLE(-89.75)),
        PROP(TARGET_CHANNEL(phy_tx_chans[1]), "hardwaregain",
VALUE_DOUBLE(tx_gain)),
        PROP_END,
    };

    return proplist_apply(tx_proplist);
}

static int setup_ad9361_tx_dds(long long freq)
{
    for (int comp = 0; comp < 2; comp += 1) {
        struct iio_channel *dds_chan = dds_chans[1][comp][0];

        struct prop dds_proplist[] = {
            PROP(TARGET_CHANNEL(dds_chan), "frequency", VALUE_LONGLONG(freq)),
            PROP(TARGET_CHANNEL(dds_chan), "phase", VALUE_LONGLONG(comp == 0 ? 0 :
90000)),
            PROP(TARGET_CHANNEL(dds_chan), "scale", VALUE_DOUBLE(1.0)),
            PROP(TARGET_CHANNEL(dds_chan), "raw", VALUE_BOOL(true)),
            PROP_END,
        };
    }
}

```

```

        };

        if (proplist_apply(dds_proplist) != 0) {
            print_err("Failed to set up AD9361 TX DDS component %c", comp == 0 ?
'I' : 'Q');
            return -1;
        }
    }

    return 0;
}

static int setup_ad9361_rx(long long lo_freq, long long sample_freq, long long
rf_bw, double rx1_gain, double rx2_gain)
{
    struct prop rx_proplist[] = {
        PROP(TARGET_CHANNEL(rx_lo), "frequency", VALUE_LONGLONG(lo_freq)),

        PROP(TARGET_CHANNEL(phy_rx_chans[0]), "rf_bandwidth",
VALUE_LONGLONG(rf_bw)),
        PROP(TARGET_CHANNEL(phy_rx_chans[0]), "sampling_frequency",
VALUE_LONGLONG(sample_freq)),
        PROP(TARGET_CHANNEL(phy_rx_chans[0]), "quadrature_tracking_en",
VALUE_BOOL(false)),
        PROP(TARGET_CHANNEL(phy_rx_chans[0]), "rf_dc_offset_tracking_en",
VALUE_BOOL(false)),
        PROP(TARGET_CHANNEL(phy_rx_chans[0]), "bb_dc_offset_tracking_en",
VALUE_BOOL(false)),

        PROP(TARGET_CHANNEL(phy_rx_chans[0]), "gain_control_mode",
VALUE_STRING("manual")),
        PROP(TARGET_CHANNEL(phy_rx_chans[1]), "gain_control_mode",
VALUE_STRING("manual")),
        PROP(TARGET_CHANNEL(phy_rx_chans[0]), "hardwaregain",
VALUE_DOUBLE(rx1_gain)),
        PROP(TARGET_CHANNEL(phy_rx_chans[1]), "hardwaregain",
VALUE_DOUBLE(rx2_gain)),
        PROP_END,
    };

    return proplist_apply(rx_proplist);
}

static void teardown_ad9361_tx_dds()
{
    for (int comp = 0; comp < 2; comp += 1) {
        struct prop dds_disable_proplist[] = {
            PROP(TARGET_CHANNEL(dds_chans[1][comp][0]), "raw", VALUE_BOOL(false)),
            PROP_END,
        };

        proplist_apply(dds_disable_proplist);
    }
}

static int setup_ad9361_rx_channels(struct iio_channel **chans)
{
    for (struct iio_channel **curr = chans; *curr != NULL; curr += 1) {
        const struct iio_data_format *fmt = iio_channel_get_data_format(*curr);

        if (fmt->length != 16) {
            print_err("Data format of channel is not 16 bits");

```

```

        return -1;
    }

    iio_channel_enable(*curr);
}

return 0;
}

struct untangle_cb_state {
    double complex *a;
    double complex *b;

    int i;
    int16_t real_scratch;
};

static ssize_t untangle_cb(const struct iio_channel *chan, void *src, size_t
bytes, void *d)
{
    struct untangle_cb_state *state = d;

    if (bytes != 2) {
        print_err("What... Sample size is not 16 bit");
        return -1;
    }

    int16_t sample;

    iio_channel_convert(chan, &sample, (int16_t *) src);

    if (state->i % 2 == 0) {
        /* I, or real component */
        state->real_scratch = sample;
    } else {
        /* Q, or imaginary component */
        double complex new_sample = state->real_scratch + sample * I;

        if (state->i % 4 == 1) {
            /* channel 1 */
            state->a[state->i / 4] = new_sample;
        } else {
            /* channel 2 */
            state->b[state->i / 4] = new_sample;
        }
    }

    state->i += 1;
    return 0;
}

static void untangle_buffers(struct iio_buffer *buf, double complex *a, double
complex *b)
{
    struct untangle_cb_state state = {
        .a = a,
        .b = b,
        .i = 0,
    };

    iio_buffer_foreach_sample(buf, untangle_cb, &state);
}

```

```

static int ad9361_apply_rx_phase_correction(int chan_set, double pho_rad) {
    int result;

    if (chan_set > AD9361_RX_CHANS) {
        print_err("Invalid channel set ID");
        return -1;
    }

    double sin_pho = sin(pho_rad);
    double cos_pho = cos(pho_rad);

    /* I coeffs */
    result = iio_channel_attr_write_double(rx_chans[chan_set][0], "calibphase",
-sin_pho);
    if (result < 0) goto err;

    result = iio_channel_attr_write_double(rx_chans[chan_set][0], "calibscale",
cos_pho);
    if (result < 0) goto err;

    /* Q coeffs */
    result = iio_channel_attr_write_double(rx_chans[chan_set][1], "calibphase",
sin_pho);
    if (result < 0) goto err;

    result = iio_channel_attr_write_double(rx_chans[chan_set][1], "calibscale",
cos_pho);
    if (result < 0) goto err;

    return 0;

err:
    print_iio_err(-result, "Failed to apply phase correction to RX%d", chan_set +
1);
    return -1;
}

enum phase_detect_method {
    METHOD_XCORR,
    METHOD_INNER_PRODUCT,
    METHOD_FFT,
};

int usage()
{
    print_err("Usage: phasecorrect mode context-uri method lo-frequency:sampling-
frequency:rf-bandwidth:rx1-gain:rx2-gain:tx-gain:dds-setup-delay");
    return 1;
}

int main(int argc, char *argv[])
{
    int ret = 1;
    int result;

    if (argc < 5) {
        return usage();
    }

    const char *mode_raw = argv[1];

```

```

bool skip_correct;
if (strcmp(mode_raw, "set") == 0) {
    skip_correct = false;
} else if (strcmp(mode_raw, "show") == 0) {
    skip_correct = true;
} else {
    print_err("Unknown mode (should be one of \"set\", \"show\")");
    return usage();
}

const char *uri = argv[2];
const char *method_raw = argv[3];

enum phase_detect_method method;
bool do_estimate = true;
if (strcmp(method_raw, "xcorr") == 0) {
    method = METHOD_XCORR;
} else if (strcmp(method_raw, "innerproduct") == 0) {
    method = METHOD_INNER_PRODUCT;
} else if (strcmp(method_raw, "fft") == 0) {
    method = METHOD_FFT;
} else if (strcmp(method_raw, "xcorr_measure") == 0) {
    method = METHOD_XCORR;
    do_estimate = false;
} else if (strcmp(method_raw, "fft_measure") == 0) {
    method = METHOD_FFT;
    do_estimate = false;
} else {
    print_err("Unknown method (should be one of \"xcorr\", \"innerproduct\" or \"fft\")");
    return usage();
}

long long lo_freq, sample_freq, rf_bw;
double rx1_gain, rx2_gain, tx_gain, dds_setup_delay;

if (sscanf(argv[4], "%lld:%lld:%lld:%lf:%lf:%lf:%lf",
            &lo_freq, &sample_freq, &rf_bw,
            &rx1_gain, &rx2_gain, &tx_gain, &dds_setup_delay) != 7) {
    print_err("Config string error");
    return usage();
}

struct iio_context *ctx = iio_create_context_from_uri(uri);
if (ctx == NULL) {
    print_iio_eno("Failed to create IIO context");
    goto err_context;
}

if (setup_ad9361_devs(ctx) != 0) {
    print_err("Failed to find AD9361 devices");
    goto err_ad9361;
}

if (setup_ad9361_phy() != 0) {
    print_err("Failed to setup AD9361 PHY");
    goto err_ad9361_phy;
}

if (setup_ad9361_tx(lo_freq, sample_freq, rf_bw, tx_gain) != 0) {
    print_err("Failed to setup AD9361 TX");
    goto err_ad9361_tx;
}

```

```

}

if (setup_ad9361_tx_dds(F_T) != 0) {
    print_err("Failed to setup AD9361 TX DDS");
    goto err_ad9361_tx_dds;
}

if (setup_ad9361_rx(lo_freq, sample_freq, rf_bw, rx1_gain, rx2_gain) != 0) {
    print_err("Failed to setup AD9361 RX");
    goto err_ad9361_rx;
}

struct iio_channel *chanlist[] = {
    rx_chans[0][0],
    rx_chans[0][1],
    rx_chans[1][0],
    rx_chans[1][1],
    NULL,
};

if (setup_ad9361_rx_channels(chanlist) != 0) {
    print_err("Failed to setup AD9361 RX channels");
    goto err_ad9361_rx_channels;
}

int period = calc_period_in_samples(F_T, sample_freq);
int n = REPEAT * period;

struct iio_buffer *rxbuf = iio_device_create_buffer(rx_dev, n, false);
if (rxbuf == NULL) {
    print_iio_eno("Failed to create IIO buffer");
    goto err_buffer;
}

double complex *a = fftw_malloc(sizeof (double complex) * n);
double complex *b = fftw_malloc(sizeof (double complex) * n);
if (a == NULL || b == NULL) {
    print_eno("Failed to allocate RX buffers");
    goto err_alloc_bufs;
}

struct phase_diff_xcorr_plan *xcorr_plan = NULL;
struct phase_diff_fft_plan *fft_plan = NULL;

switch (method) {
    case METHOD_XCORR:
        xcorr_plan = phase_diff_xcorr_plan(a, b, n, do_estimate);
        if (xcorr_plan == NULL) {
            print_err("Failed to plan xcorr");
            goto err_plan;
        }
        break;

    case METHOD_FFT:
        fft_plan = phase_diff_fft_plan(a, b, n, do_estimate);
        if (fft_plan == NULL) {
            print_err("Failed to plan FFT");
            goto err_plan;
        }
        break;

    default:

```

```

        break;
    }

    printf("Giving the hardware some time to stabilize before capturing
anything\n");
    usleep(dds_setup_delay * 1000);

    printf("Refilling buffer (%d samples)\n", n);
    result = iio_buffer_refill(rxbuf);
    if (result < 0) {
        print_iio_err(-result, "Failed to refill RX buffer");
        goto err_refill;
    }

    printf("Untangling buffer\n");
    untangle_buffers(rxbuf, a, b);

    double pho;
    switch (method) {
        case METHOD_XCORR:
            pho = phase_diff_xcorr(xcorr_plan, period);
            break;

        case METHOD_INNER_PRODUCT:
            pho = phase_diff_inner_product(a, b, n);
            break;

        case METHOD_FFT:
            pho = phase_diff_fft(fft_plan);
            break;
    }

    if (pho != pho) {
        print_err("Phase offset is NaN");
        goto err_pho_nan;
    }

    double pho_degrees = rad2deg(pho);
    printf("Phase offset: %.06f degrees\n", pho_degrees);

    if (!skip_correct) {
        printf("Applying phase correction to RX2\n");
        ad9361_apply_rx_phase_correction(1, pho);
    }

    ret = 0;

err_pho_nan:
err_plan:
err_refill:
    iio_buffer_destroy(rxbuf);
err_alloc_bufs:
err_buffer:
err_ad9361_rx_channels:
err_ad9361_rx:
err_ad9361_tx_dds:
    teardown_ad9361_tx_dds();
err_ad9361_tx:
err_ad9361_phy:
err_ad9361:
    iio_context_destroy(ctx);
err_context:

```

```

    return ret;
}

```

### Файл phasebench.c

```

#include "phaselib.h"

#include <stdlib.h>
#include <stdbool.h>
#include <math.h>

#include <fftw3.h>

#include "utils.h"

#define WISDOM_FILE "phasebench.wisdom"

#define K (1000)
#define M (1000 * 1000)

#define F_T (1 * K)

struct bench_config {
    bool valid;

    int f_s;
    int repeat;

    double complex *a;
    double complex *b;
    int n;
    int period;

    struct phase_diff_xcorr_plan *xcorr_plan;
    struct phase_diff_fft_plan *fft_plan;
};

static int fs_arr[] = { 2 * K, 2500, 5 * K, 10 * K, 25 * K, 100 * K, 250 * K, 500
* K, 1 * M, 2500 * K, 5 * M };
static int repeats[] = { 1, 2, 4, 8, 16, 32, 64, 128 };

static void end_bench_and_output_csv(int trial, const char *id, struct
bench_config *cfg)
{
    double t;

    t = bench_end();
    printf("bench,%d,%d,%d,%s,%f\n", trial, cfg->f_s, cfg->repeat, id, t);
}

static int setup_configs(struct bench_config **configs_out, int
*configs_count_out, bool do_estimate, bool do_only_inner_product)
{
    printf("Warming up...\n");

    int configs_count = length_of(fs_arr) * length_of(repeats);

    struct bench_config *configs = calloc(configs_count, sizeof (struct
bench_config));
    if (configs == NULL) {
        print_eno("Failed to allocate bench config structure");
        goto err_alloc_configs;
    }

```

```

}

int i = 0;
for (int j = 0; j < length_of(fs_arr); j += 1) {
    for (int k = 0; k < length_of(repeats); k += 1) {
        printf("Warming up... [%d/%d]\n", i + 1, configs_count);
        fflush(stdout);

        struct bench_config *cfg = &configs[i];

        cfg->f_s = fs_arr[j];
        cfg->repeat = repeats[k];
        cfg->period = calc_period_in_samples(F_T, cfg->f_s);
        cfg->n = cfg->repeat * cfg->period;

        cfg->a = fftw_malloc(sizeof (double complex) * cfg->n);
        cfg->b = fftw_malloc(sizeof (double complex) * cfg->n);
        if (cfg->a == NULL || cfg->b == NULL) {
            print_eno("Failed to allocate space for one of the buffers");
            goto err_alloc_buf;
        }

        if (!do_only_inner_product) {
            bench_start();
            cfg->xcorr_plan = phase_diff_xcorr_plan(cfg->a, cfg->b, cfg->n,
do_estimate);
            end_bench_and_output_csv(-1, "WARMUP_XCORR_PLAN", cfg);

            bench_start();
            cfg->fft_plan = phase_diff_fft_plan(cfg->a, cfg->b, cfg->n,
do_estimate);
            end_bench_and_output_csv(-1, "WARMUP_FFT_PLAN", cfg);
        }

        generate_dds_monotone(cfg->a, cfg->n, F_T, cfg->f_s, deg2rad(0));
        generate_dds_monotone(cfg->b, cfg->n, F_T, cfg->f_s, deg2rad(30));

        i += 1;
    }
}

printf("Warming up done!\n");

if (fftw_export_wisdom_to_filename(WISDOM_FILE) == 0) {
    print_err("Failed to export wisdom :(");
}

*configs_out = configs;
*configs_count_out = configs_count;

return 0;

err_alloc_buf:
/* don't bother, there's no way to free the plans anyway */
/* free(configs); */
err_alloc_configs:
return -1;
}

int usage()
{
    print_err("Usage: phasebench trials measure-or-estimate");
}

```

```

    return 1;
}

int main(int argc, char *argv[])
{
    if (argc < 3) {
        return usage();
    }

    int trials;
    if (sscanf(argv[1], "%d", &trials) < 1) {
        print_err("Invalid trials count specified");
        return usage();
    }

    bool do_estimate = false;
    bool do_only_inner_product = false;
    if (strcmp(argv[2], "estimate") == 0) {
        do_estimate = true;
    } else if (strcmp(argv[2], "measure") == 0) {
        do_estimate = false;
    } else if (strcmp(argv[2], "only-inner-product") == 0) {
        do_only_inner_product = true;
    } else {
        print_err("Invalid planning strategy specified, use either \"measure\",
        \"estimate\" or \"only-inner-product\"");
        return usage();
    }

    double diff;

    if (fftw_import_wisdom_from_filename(WISDOM_FILE) != 0) {
        printf("Wisdom loaded, this should go faster\n");
    }

    struct bench_config *configs;
    int config_count;

    if (setup_configs(&configs, &config_count, do_estimate, do_only_inner_product)
    != 0) {
        print_err("Failed to set up configs");
        return 1;
    }

    for (int trial = 0; trial < trials; trial += 1) {
        for (int i = 0; i < config_count; i += 1) {
            struct bench_config *cfg = &configs[i];

            bench_start();
            diff = phase_diff_inner_product(cfg->a, cfg->b, cfg->n);
            printf("Phase difference (inner product): %f\n", rad2deg(diff));
            end_bench_and_output_csv(trial, "PHASE_DIFF_INNER_PRODUCT", cfg);

            if (!do_only_inner_product) {
                bench_start();
                diff = phase_diff_xcorr(cfg->xcorr_plan, cfg->period);
                printf("Phase difference (cross correlation): %f\n",
rad2deg(diff));
                end_bench_and_output_csv(trial, "PHASE_DIFF_XCORR", cfg);

                bench_start();
                diff = phase_diff_fft(cfg->fft_plan);

```

```

        printf("Phase difference (Fourier transform): %f\n",
rad2deg(diff));
        end_bench_and_output_csv(trial, "PHASE_DIFF_FFT", cfg);
    }
}
}

return 0;
}

```

### Файл pivot-bench.py (скрипт для обробки вихідних даних phasebench)

```
#!/usr/bin/env python3
```

```
import sys
import csv
```

```
wisdom_configs = {}
configs = {}
events_found = []
```

```

with open(sys.argv[1], 'r') as input_csv:
    csv_reader = csv.reader(input_csv, delimiter=',')
    for row in csv_reader:
        # ignore likely non CSV entries
        if len(row) != 6:
            continue

        tag, trial, fs, repeat, event_id, time_us = row

        if tag != 'bench':
            continue

        trial = int(trial)
        fs = int(fs)
        repeat = int(repeat)
        time_us = float(time_us)

        config = (fs, repeat)

        if trial == -1:
            # wisdom trial
            if config not in wisdom_configs:
                wisdom_configs[config] = {}

            if event_id == 'WARMUP_XCORR_PLAN':
                wisdom_configs[config]['xcorr'] = time_us
            elif event_id == 'WARMUP_FFT_PLAN':
                wisdom_configs[config]['fft'] = time_us
        else:
            if config not in configs:
                configs[config] = {}

            config_events = configs[config]
            if event_id not in config_events:
                config_events[event_id] = {}

            event_times = config_events[event_id]
            if trial not in config_events:
                event_times[trial] = {}

            if event_id not in events_found:

```

```

        events_found.append(event_id)

    event_times[trial] = time_us

for event_id in events_found:
    with open(f'{sys.argv[2]}-{event_id}.csv', 'w') as output_csv:
        csv_writer = csv.writer(output_csv, delimiter=',')

        for config, config_events in configs.items():
            event_times = config_events[event_id]

            times_list = [event_times[i] for i in sorted(event_times.keys())]
            csv_writer.writerow([config[0], config[1], *times_list])

    with open(f'{sys.argv[2]}-warmup.csv', 'w') as output_csv:
        csv_writer = csv.writer(output_csv, delimiter=',')

        for config, config_warmups in wisdom_configs.items():
            csv_writer.writerow([config[0], config[1],
config_warmups.get('xcorr'), config_warmups.get('fft')])

```