Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __ інформаційних радіотехнологій та технічного захисту інформації
(повна назва)

Кафедра _ мікропроцесорних технологій і систем _
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _______________другий (магістерський)______________
___Аналіз та реалізація алгоритмів перетворення Фурє на ПЛІС_____________.
(тема)

Виконав:
здобувач _____2____ року навчання,
групи ІМСм-23-1______________
___Максим СКОРБАТЮК_____________
(власне ім'я, прізвище)

Спеціальність _171 Електроніка__________
(код і повна назва спеціальності)
Тип програми _освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Інженерія мікропроцесорних систем___ ___ (повна назва освітньої програми)

Керівник _ проф. Олександр ВОРГУЛЬ_____
(посада, власне ім'я, прізвище)

Допускається до захисту

В.о. завідувача кафедри МТС _____ ____ Олег ЗУБКОВ_____
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет_ інформаційних радіотехнологій та технічного захисту інформації
Кафедра_ _мікропроцесорних технологій і систем _
Рівень вищої освіти___________другий (магістерський)_________________
Спеціальність________________171 Електроніка ______________________

(код і повна назва)

Тип програми _____________освітньо-наукова ___________________
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія мікропроцесорних систем _____
(повна назва)

ЗАТВЕРДЖУЮ:
В.о. зав. кафедри МТС

__ Олег ЗУБКОВ
(підпис)
«23»____06______ 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві_________Скорбатюку Максиму Володимировичу__________________.

(прізвище, ім'я, по батькові)

1. Тема роботи _Аналіз та реалізація алгоритмів перетворення Фур'є на ПЛІС__
затверджена наказом університету від _17_ ____04____2025 р. № 292Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 23 _06__2025 р.
3. Вихідні дані до роботи Розглянути класичне ДПФ (без оптимізації як у БПФ) з погляду аналітичного опису, програмної реалізації традиційними мовами програмування. Виконати реалізацію алгоритму в ПЛІС. Оцінити вимоги до ресурсів та обчислювальну складність. Самостійно обрати тип реалізації – послідовний чи матричний.
4. Перелік питань, що потрібно опрацювати в роботі _______________________________
1) Математичні відомості про дискретне перетворення Фур'є________________________
2) Програмна реалізація дискретного перетворення Фур'є__________________________
3) Реалізація дискретного перетворення Фур'є на ПЛІС____________________________
4) Критична оцінка результатів____
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) ____________
____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
основний			

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Огляд джерел про ДПФ, його властивості та особливості реалізації у обчислювальних систе-	21.04.2025-27.04.2025	виконано
2	Аналіз структури ДПФ з обчислювальної точки зору та програмного вирішення задачі	28.04.2025-7.05.2025	виконано
3	Аналіз особливостей реалізації ДПФ на ПЛІС	8.05.2025-16.05.2025	виконано
4	Складання моделей реалізації ДПФ на ПЛІС	17.05.2025-24.05.2025	виконано
5	Виконання моделювання проектів мовою VHDL	25.05.2025-7.06.2025	виконано
6	Написання пояснювальної записки	8.06.2025-15.06.2025	виконано

Дата видачі завдання 21 04 2025р.

Здобувач ____________________
(підпис)

Керівник роботи ____________________ проф. Олександр Воргуль
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 74 с., 2 рис., 9 табл., 2 дод., 13 джерел.

ДИСКРЕТНЕ ПЕРЕТВОРЕННЯ ФУР'Є , ПЛІС, VHDL, VIVADO

Робота присвячена дослідженню класичного дискретного перетворення Фур'є. Метою дослідження є аналіз обчислювальної складності алгоритмів ДПФ та подальша їх реалізація у ПЛІС. Широко використовується комп'ютерне моделювання. В результаті реалізований алгоритм послідовного визначення ДПФ, призначений для реалізації в ПЛІС Artix7-100. Проведено експерименти на апаратному забезпеченні від Xilinx (AMD), які продемонстрували правильність отриманих результатів. Отримані висновки можуть бути використані при подальшому вивченні обчислювальних алгоритмів.

ABSTRACT

Explanatory note: 74 pages, 2 figures, 9 tables, 2 appendices, 13 references.

DISCRETE FOURIER TRANSFORM, FPGA, VHDL VIVADO

The work is devoted to the study of the classical discrete Fourier transform. The purpose of the study is to analyze the computational complexity of DFT algorithms and their subsequent implementation in FPGA. Computer modeling is widely used. As a result, an algorithm for sequentially determining the DFT is implemented, designed for implementation in the Artix7-100 FPGA. Experiments were conducted on hardware from Xilinx (AMD), demonstrating the correctness of the results. The findings can be used in further study of computational algorithms.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ 9
ВСТУП ... 10
1 МАТЕМАТИЧНІ ВІДОМОСТІ ПРО ПЕРЕТВОРЕННЯ ФУР'Є ... 12
1.1 Ряд Фур'є та його властивості. Обмеження на функцію, що подається рядом. Умови Діріхле ... 12
1.1.1 Основи розкладання в ряд Фур'є ... 12
1.1.2 Властивості ряду Фур'є ... 13
1.1.3 Обмеження на функцію, представлену рядом Фур'є ... 14
1.1.4 Умови Діріхле ... 14
1.1.5 Висновок ... 14
1.2 Інтеграл Фур'є. Властивості спектральної густини. Умови існування інтеграла Фур'є. Зв'язок із рядом Фур'є ... 16
1.2.1 Вступ ... 16
1.2.2 Інтеграл Фур'є: визначення та аналітичний опис ... 16
1.2.3 Умови існування інтеграла Фур'є ... 16
1.2.4 Властивості інтеграла Фур'є ... 17
1.2.5 Спектральна щільність ... 17
1.2.6 Зв'язок інтеграла Фур'є з рядом Фур'є ... 18
1.2.7 Висновок ... 18
1.3 Дискретне перетворення Фур'є. Властивості спектра для дійсних сигналів. Обмеження та властивості ДПФ ... 19
1.3.1 Вступ ... 19
1.3.2 Визначення дискретного перетворення Фур'є ... 19
1.3.3 Властивості спектра при дійсному вхідному сигналі ... 20
1.3.4 Обмеження на вхідний сигнал ... 21
1.3.5 Властивості ДПФ (теореми про спектри) ... 21
1.3.6 Висновок ... 22
2 ДИСКРЕТНЕ ПЕРЕТВОРЕННЯ ФУР'Є З ПОГЛЯДУ ПРОГРАМУВАННЯ .. 23
2.1. Реалізація ДПФ у Python ... 23
2.2. Реалізація ДПФ у Pascal ... 24
2.3. Реалізація ДПФ в Octave ... 25
2.4 Приклад для подальшого порівняння ... 26
2.5 Висновок ... 29

3 РЕАЛІЗАЦІЯ ДИСКРЕТНОГО ПЕРЕТВОРЕННЯ ФУР'Є НА ПЛІС 30
3.1 Підхід до реалізації повного ДПФ. Визначення потрібної кількості ресурсів. 30
Висновок 32
3.2 Реалізація обчислення ДПФ на одній частоті 33
3.2.1 Запропоновані покращення 33
3.2.2 Реалізація 34
3.2.3 Результати 35
3.2.4 Висновок 35
3.3. Масштабування попереднього одно частотного випадку ДПФ з допомогою оператора Generate. Визначення потрібної кількості ресурсів 36
3.3.1 Вступ 36
3.3.2 Оптимізований проект з оператором generate для 129 пікселів (з 256) 36
3.3.3 VHDL-код: повне ДПФ з оператором generate 36
3.3.4 Оцінка ресурсів (Xilinx Artix-7) 38
3.3.6 Рекомендації щодо оптимізації при недостатній кількості ресурсів 38
3.3.7 Висновок 39
3.4 Реалізація ДПФ з допомогою високорівневого синтезу. Визначення потрібної кількості ресурсів 39
3.4.1 Вступ 39
3.4.2 Проект DFT HLS (реалізація C++) 40
3.4.3 Допоміжні файли 42
3.4.3 Скрипт TCL для Vivado HLS 43
3.4.4 Порівняння із традиційним синтезом 44
3.4.5 Детальна реалізація HLS DFT з оптимізаційними праг мами 44
3.4.5 Ключові прагми оптимізації та їх вплив 47
3.4.6 Додаткові методи оптимізації 48
3.4.7 Експлуатаційні характеристики 49
4 ПОРІВНЯННЯ РЕАЛІЗАЦІЙ ДПФ ПОБУДОВАНИХ У VHDL ТА HDL 50
4.1 Детальний аналіз продуктивності реалізації HLS DFT 50
4.2 Реалізація HLS. Аналіз часу 52
4.3 Методи оптимізації продуктивності HLS 52
4.4 Компроміс між продуктивністю та точністю 53
Варіант 1: Максимальна продуктивність 53

Варіант 2: Максимальна точність 53
Збалансована конфігурація (рекомендується) 53
4.5 Рекомендації щодо досягнення максимальної продуктивності 54
4.6 Оптимізація конвеєра та DSP48 для HLS DFT 54
4.6.1 Оптимізація конвеєра в HLS 54
4.6.2 Використання DSP48 у HLS порівняно з VHDL 56
4.6.3 Компроміс між продуктивністю та ресурсами 57
4.6.4 Рекомендації 57
4.7 Висновки 58
ВИСНОВОК 59
ПЕРЕЛІК ПОСИЛАНЬ 61
ДОДАТОК А. ПОВНІ ЛІСТИНГИ КОДУ 62
ДОДАТОК Б. РЕЗУЛЬТАТИ СИМУЛЯЦІЇ 72
ДОДАТОК В. ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ 74

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

БПЕ	Безпровідна передача енергії
ЕМП	Електромагнітне поле
МКЕ	Міжнародна комісія з електричного зв'язку
A4WP	Alliance for Wireless Power
DRL	Dosimetric Reference Levels
ERL	Exposure Reference Levels
ICNIRP	International Commission on Non-Ionizing Radiation Protection
PMA	Power Matters Alliance

ВСТУП

Роботу присвячено аналізу та реалізації алгоритмів перетворення Фур'є на ПЛІС, з фокусом на пряме обчислення дискретного перетворення Фур'є (ДПФ) без використання алгоритмів швидкого перетворення Фур'є (БПФ).

Робота є актуальною. Перетворення Фур'є є основним інструментом цифрової обробки сигналів (ЦОС). Ефективна апаратна реалізація ДПФ на програмованих логічних інтегральних схемах (ПЛІС) є критично важливою для додатків реального часу, що потребують високої продуктивності та детермінізму. Однак пряме обчислення ДПФ для значних розмірностей даних є ресурсомістким завданням, що зумовлює необхідність пошуку оптимальних підходів для його реалізації на конкретних апаратних платформах.

Метою роботи є проведення аналізу властивостей перетворення Фур'є та практична розробка, моделювання та реалізація алгоритму прямого обчислення дискретного перетворення Фур'є (без прискорення БПФ) на ПЛІС сімейства Xilinx Artix-7 з подальшою оцінкою результатів.

Об'єктом дослідження є алгоритми обчислення дискретного перетворення Фур'є (ДПФ).

Предмет дослідження: Методи програмної та апаратної реалізації алгоритму прямого обчислення ДПФ (без БПФ) на платформі ПЛІС Xilinx Artix 7-100, включаючи оцінку їхньої обчислювальної складності та ресурсомісткості.

Використані наступні методи дослідження.

Аналітичні методи: Математичний аналіз безперервного ряду Фур'є, інтеграла Фур'є та дискретного перетворення Фур'є для вивчення їх властивостей та формування вимог до реалізації.

Програмне моделювання: Розробка та верифікація алгоритму ДПФ мовами програмування Pascal, Python та MATLAB для створення еталонних моделей та аналізу роботи алгоритму.

Апаратне моделювання та синтез: Проектування, моделювання, синтез та реалізація цифрових схем обчислення ДПФ на ПЛІС у середовищі розробки Vivado 2018.2 з використанням мов опису апаратури (HDL). Ключовим аспектом є використання оператора generate для масштабованої реалізації.

Порівняльний аналіз: Зіставлення результатів, отриманих на етапах програмного моделювання та апаратної реалізації, а також оцінка обчислювальної складності та ресурсних витрат.

Основні завдання, розглянуті в роботі: математичний опис та аналіз перетворень Фур'є (безперервний ряд, інтеграл, ДПФ), програмна реалізація та верифікація алгоритму ДПФ на Pascal, Python та MATLAB, апаратна реалізація на ПЛІС Xilinx Artix 7-100 (Vivado 2018.2). Остання включає оцінку обчислювальної складності та ресурсомісткості послідовного алгоритму ДПФ для N=256 точок, розробку ядра обчислення ДПФ для однієї частотної компоненти та створення масштабованої структури для обчислення підмножини частотних компонентів з використанням оператора Generate.

Порівняння результатів реалізації та аналіз використаних ресурсів ПЛІС (зокрема, блоків DSP48E1) та арифметики з фіксованою точкою.

Робота закладає основи подальшої оптимізації процедури обчислень ДПФ на ПЛИС.

1 МАТЕМАТИЧНІ ВІДОМОСТІ ПРО ПЕРЕТВОРЕННЯ ФУР'Є

1.1 Ряд Фур'є та його властивості. Обмеження на функцію, що подається рядом. Умови Діріхле

Одним з фундаментальних понять у математичному аналізі та теорії сигналів є розкладення функцій у ряд Фур'є [1]. Дане уявлення дозволяє аналізувати періодичні функції шляхом подання їх сумою гармонійних компонентів. Метод отримав широке застосування у фізиці, інженерії, теорії управління, цифровій обробці сигналів та інших галузях.

Однак не будь-яка функція може бути представлена рядом Фур'є, і для цього необхідне виконання низки умов, у тому числі так званих умов Діріхле [2,3].

У цьому розділі розглянемо основні властивості ряду Фур'є, визначення обмежень, що накладаються на функції, представлені у вигляді ряду Фур'є, а також розгляд умов Дирихле як критерію збіжності розривів.

1.1.1 Основи розкладання в ряд Фур'є

Нехай *f(x)* – функція, задана на інтервалі [−π, π] і періодична, з періодом 2π. Тоді f (x) можна представити у вигляді ряду Фур'є [1]:

$$f(x) \approx \frac{a_0}{2} + \sum_n \left(a_n \cdot \cos(n \cdot x) + b_n \cdot \sin(n \cdot x)\right) \quad (1.1)$$

де коефіцієнти Фур'є a_n та b_n визначаються формулами [3,4]:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cdot \cos(n \cdot x)\, dx \quad (1.2)$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cdot \sin(n \cdot x)\, dx \quad (1.3)$$

Якщо функція задана на довільному періоді довжини 2L, то формули модифікуються з урахуванням довжини інтервалу і переходу до частот nπ/L.

Ряди Фур'є зазвичай розглядаються в контексті простору $L^2([-\pi,\pi])$, що складається з функцій, що інтегруються з квадратом. Цей простір наділений скалярним добутком:

$$\langle f, g \rangle = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cdot g(x)\,dx \tag{1.4}$$

що робить базис тригонометричних функцій ортонормованим. Відповідно до теореми про розкладання в ортонормовану систему, будь-яка функція з L^2 може бути наближена рядом Фур'є в сенсі середньоквадратичної збіжності .

1.1.2 Властивості ряду Фур'є

Ряд Фур'є має ряд корисних властивостей [4-6]:

- Лінійність. Якщо *f(x)* і *g(x)* мають розкладання в ряди Фур'є, їх лінійна комбінація також має ряд Фур'є з відповідними коефіцієнтами, які обчислюються за такою ж лінійною комбінацією.
- Парність та непарність. У парних функцій відсутні синусні компоненти, непарні — косинусні.
- Диференціювання та інтегрування. Похідні та інтегральні функції також можуть бути представлені рядами.
- Теорема Парсеваля. Забезпечує зв'язок між інтегралом квадрата функції та сумою квадратів її коефіцієнтів Фур'є:

$$\frac{1}{\pi}\int_{-\pi}^{\pi} |f(x)|^2\,dx = \frac{a_0{}^2}{2} + \sum_n a_n{}^2 + b_n{}^2 \tag{1.5}$$

1.1.3 Обмеження на функцію, представлену рядом Фур'є

Для існування та збіжності рядом Фур'є функція повинна задовольняти наступним умовам:

1. Кусочна безперервність на розглянутому інтервалі.
2. Обмеженість функції та наявність кінцевого числа розривів першого роду.
3. Інтегрованість на періоді: $\int_{-\pi}^{\pi} |f(x)| \cdot dx < \infty$.

Якщо ці умови виконані, розкладання ряд Фур'є існує в сенсі L^2. Для рівномірної та точкової збіжності можуть знадобитися додаткові умови.

1.1.4 Умови Діріхле

Згідно з умовами Діріхле, ряд Фур'є функції *f(x)*, періодичної з періодом 2π, сходиться в точці x_0 до значення:

$$\frac{\left(f\left(x_0^{+}\right) + f\left(x_0^{-}\right)\right)}{2} \tag{1.6}$$

якщо виконані такі умови:

1. f (x) кусочно безперервна на [−π, π];
2. f (x) має кінцеве число розривів та екстремумів.

Якщо функція безперервна в точці x_0, то ряд сходиться до її значення. У разі розриву – до середнього значення односторонніх меж (6). За порушення умов можуть спостерігатися явища типу осциляцій Гіббса [1].

1.1.5 Висновок

Ряд Фур'є є потужним аналітичним інструментом, що дозволяє перейти від тимчасового представлення функцій до частотного. Для його коректного застосування необхідно враховувати обмеження функції, що забезпечують збіжність ря-

ду. Умови Дирихле грають центральну роль у забезпеченні точкової збіжності, особливо у точках розриву. Комплексне розуміння цих принципів необхідне як теоретичних пошуків, так прикладних завдань обробки сигналів і розв'язання диференціальних рівнянь.

1.2 Інтеграл Фур'є. Властивості спектральної густини. Умови існування інтеграла Фур'є. Зв'язок із рядом Фур'є

1.2.1 Вступ

Інтеграл Фур'є відіграє фундаментальну роль у теорії сигналів, математичній фізиці та прикладній математиці. Він забезпечує перехід від часового (або просторового) уявлення функції до її спектрального (частотного) опису. У цьому розділі розглядаються основні властивості інтеграла Фур'є, умови його існування, зв'язок із рядом Фур'є, а також характеристики спектральної густини.

1.2.2 Інтеграл Фур'є: визначення та аналітичний опис

Інтеграл Фур'є є узагальненням ряду Фур'є для невласних та аперіодичних функцій. Якщо функція *f(t)* абсолютно інтегрована на всій осі $f(t) \in L^1(\Re)$, її перетворення Фур'є визначається як [3,4]:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot \exp(-j\omega t)\, dt \tag{1.7}$$

Функція *F(ω)*, звана спектральної щільністю, описує розподіл нескінченно малих амплітуд за частотами. Зворотне перетворення дозволяє відновити вихідну часову функцію за її спектральною щільністю:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \cdot \exp(+j\omega t)\, d\omega \tag{1.8}$$

1.2.3 Умови існування інтеграла Фур'є

Для існування інтеграла Фур'є достатньо, щоб функція *f(t)* задовольняла наступним умовам [3,4]:

- Абсолютна інтегрованість: $\int_{-\infty}^{\infty} |f(t)|\, dt < \infty$,
- Скінченна кількість розривів та екстремумів на будь-якому кінцевому інтервалі,
- Відсутність «надто сильних » сингулярностей .

Більш коректно перетворення Фур'є можна визначити для функцій з просторів $L^1(\mathbf{R})$ \) і $L^2(\mathbf{R})$, а також в рамках теорії узагальнених функцій (розподілів), що розширює область його застосування.

1.2.4 Властивості інтеграла Фур'є

Інтеграл Фур'є має низку корисних властивостей:
- Зсув у часі : $F\{ f(t - t_0)\} = \exp(- i\omega t_0) \cdot F(\omega)$.
- Модуляція : $F\{ f(t) \cdot \exp(i\omega_0 t)\} = F(\omega-\omega_0)$.
- Згортка : $F \{ f * g\} = F (\omega) \cdot G (\omega)$.
- Диференціювання : $F\{ f^{(n)}(t)\} = (i\omega)^n \cdot F(\omega)$.

Ці властивості роблять перетворення Фур'є потужним інструментом для аналізу лінійних систем та диференціальних рівнянь.

1.2.5 Спектральна щільність

Спектральна щільність потужності (СПМ) визначає, як енергія (або потужність) сигналу розподілена за частотами. Для випадкових процесів спектральна щільність потужності визначається як перетворення Фур'є автокореляційної функції для випадку стаціонарних випадкових процесів [2-4]:

$$S(\omega) = \int_{-\infty}^{\infty} R(\tau) \cdot \exp(- j\omega\, \tau)\, d\tau \qquad (1.9)$$

СПМ широко використовується в теорії сигналів та систем, у радіотехніці, статистичній обробці даних. Вона дає уявлення про домінуючі частотні компоненти сигналу в енергетичному значенні.

1.2.6 Зв'язок інтеграла Фур'є з рядом Фур'є

Ряд Фур'є застосовується для аналізу періодичних функцій, тоді як інтеграл Фур'є - для аперіодичних. Однак існує зв'язок між ними: якщо період функції прямує до нескінченності, ряд Фур'є збігатиметься у ліміті до інтегралу Фур'є. Формально, при переході від дискретного спектра періодичної функції до безперервного аперіодичного спектру:

$$\lim_{T \to \infty} \sum_{n=-\infty}^{\infty} C_n \cdot \exp\left(j\omega_0\, nT\right) \to \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \cdot \exp\left(+\, j\omega\, t\right) d\omega \qquad (1.10)$$

Це означає, що інтеграл Фур'є можна розглядати як граничний випадок ряду Фур'є.

1.2.7 Висновок

Інтеграл Фур'є є одним з наріжних каменів сучасного аналізу сигналів та математичної фізики. Його властивості, включаючи лінійність, згортку, диференціювання та спектральне розкладання, роблять його універсальним інструментом аналізу. Зв'язок із рядом Фур'є дозволяє простежити наступність між дискретним і безперервним спектральним уявленням, а спектральна щільність забезпечує глибоке розуміння енергетичного змісту сигналів у частотній області.

1.3 Дискретне перетворення Фур'є. Властивості спектра для дійсних сигналів. Обмеження та властивості ДПФ

1.3.1 Вступ

Дискретне перетворення Фур'є (ДПФ) являє собою фундаментальний інструмент в цифровій обробці сигналів, що використовується для аналізу частотної структури дискретних сигналів.

Особлива увага на практиці приділяється перетворенню дійсних сигналів, оскільки більшість реальних сигналів (аудіо, зображення, сенсорні дані) є дійснозначними . У таких випадках спектр має ряд особливих властивостей, пов'язаних із симетрією та енергетичною структурою.

Поряд з цим існує практика обробки комплексної обвідної сигналу. У цьому випадку не можна обмежуватися дійснозначною версією алгоритму перетворення Фур'є

Даний розділ спрямований на опис дискретного перетворення Фур'є, його основних особливостей і обмежень, а також інструментів аналізу спектральних властивостей для дійсних сигналів.

1.3.2 Визначення дискретного перетворення Фур'є

Нехай *x[n]* - скінченний дискретний сигнал довжини N, тобто. *x[n]* визначено для n = 0, 1, ..., N−1. Тоді дискретне перетворення Фур'є (ДПФ) визначається формулою:

$$X[k] = \sum_{n=0}^{N-1} x_n \cdot \exp\left(-j\frac{2\pi}{N} n\, k\right),\ \text{для k = 0, 1, ..., N-1.} \qquad (1.11)$$

Зворотнє перетворення має вигляд:

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X[k] \cdot \exp\left(+j\frac{2\pi}{N} n\, k\right), \text{ для n = 0, 1, ..., N-1.} \qquad (1.12)$$

Тут *X[k]* - комплексні спектральні коефіцієнти, що представляють амплітуду і фазу гармонік, що становлять сигнал. Перетворення здійснюється над скінченним числом відліків, що відрізняє ДПФ від безперервного перетворення Фур'є.

1.3.3 Властивості спектра при дійсному вхідному сигналі

Якщо вхідний сигнал *x[n]* - дійсний, його спектр *X[k]* має наступні властивості:

1. Комплексна сполучена симетрія:

X[N−k] = conj (X[k]), де *conj ()* - комплексне сполучення.

Це означає, що спектр симетричний відносно середини діапазону частот, та його друга половина повністю визначається першою.

2. Спектральні складові на нульовій та центральній частоті є дійсно значними:

Значення *X[0]* і *X[N/2]* (при парному *N*) є дійсними числами.

3. Енергетична симетрія:

Спектральна щільність потужності $|X[k]|^2$ симетрична: $|X[k]|^2 = |X[N-k]|^2$.

Ці властивості активно використовуються при оптимізації обчислень та стисканні даних (наприклад, в алгоритмах БПФ та стиску зображень) [8-10].

1.3.4 Обмеження на вхідний сигнал

Для коректного застосування ДПФ необхідно враховувати такі обмеження:

1. Скінченність сигналу: ДПФ застосовується до сигналів скінченної довжини. Якщо сигнал нескінченний, можна розбивати сигнал на сегменти або виділяти ділянку сигналу віконною функцією.
2. Періодичність Результат ДПФ інтерпретується як спектр періодичного сигналу, де період дорівнює N. Це означає, що перехід до спектру супроводжується припущенням періодичності вихідного сигналу.
3. Вибір частоти дискретизації: Частота дискретизації має задовольняти теоремі Найквіста — Шеннона, щоб уникнути накладання спектрів (aliasing).
4. Квантування: При практичній реалізації слід враховувати помилки, пов'язані з кінцевою розрядністю подання сигналу та коефіцієнтів.

1.3.5 Властивості ДПФ (теореми про спектри)

ДПФ має низку фундаментальних властивостей, аналогічних безперервному перетворенню Фур'є:

1. Лінійність:

$DFT\{a \cdot x[n] + b \cdot y[n]\} = a \cdot X[k] + b \cdot Y[k]$.

2. Зсув за часом:

Якщо $x[n-n_0]$, то $X[k]$ множиться на $e^{-j(2\pi/N)kn_0}$.

3. Зсув за частотою:

Примноження сигналу на $e^{j(2\pi/N)kn_0}$ призводить до зсуву спектра.

4. Згортка:

Згортка двох сигналів у часовій області відповідає поелементному множенню їх спектрів: $DFT\{x[n] * y[n]\} = X[k] \cdot Y[k]$.

5. Перемноження у часі:

Перемноження сигналів у часі відповідає циклічній згортці спектрів.

6. Рівність Парсеваля:

$\sum |x[n]|^2 = (1/N) \sum |X[k]|^2$ - зберігає енергію при переході з тимчасової області в частотну.

Ці властивості широко застосовуються при розробці фільтрів, спектральному аналізі та алгоритмах обробки сигналів та використовуються як інструменти для завдань аналізу та синтезу систем.

1.3.6 Висновок

Дискретне перетворення Фур'є є важливим інструментом цифрової обробки сигналів. Воно дозволяє ефективно аналізувати частотну структуру дискретних сигналів, яке властивості роблять можливим застосування різних теоретичних і практичних методів фільтрації, компресії та аналізу.

У разі дійсних вхідних сигналів спектр має симетрію, що суттєво спрощує інтерпретацію результатів та скорочує обчислювальні ресурси. Обмеження сигналу (дискретність, періодичність, частота дискретизації) визначають коректність результатів аналізу.

2 ДИСКРЕТНЕ ПЕРЕТВОРЕННЯ ФУР'Є З ПОГЛЯДУ ПРОГРАМУВАННЯ

Дискретне перетворення Фур'є (ДПФ, або DFT - Discrete Fourier Transform відіграє ключову роль в цифровій обробці сигналів. Воно дозволяє переводити сигнал з тимчасової області в частотну , що має важливе значення при аналізі спектральних характеристик сигналів . дозволить виявити особливості підходів до програмної реалізації алгоритмів спектрального аналізу у різних мовах та середовищах.

ДПФ переводить кінцеву дискретну послідовність комплексних чисел у часовій області до послідовності комплексних чисел у частотній області. Нехай x [n] - дискретний сигнал довжини N. Тоді, згідно [6] пряме ДПФ визначається як:

$$X[k] = \sum_{n=0}^{N-1} x_n \cdot \exp\left(-j\frac{2\pi}{N} n\, k\right), \text{ для k = 0, 1, …, N-1.} \quad (2.1)$$

Назад перетворення дозволяє відновити сигнал з його спектру:

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X[k] \cdot \exp\left(+j\frac{2\pi}{N} n\, k\right), \text{ для n = 0, 1, …, N-1.} \quad (2.2)$$

де j – уявна одиниця. Ці формули є основою всіх алгоритмічних реалізацій ДПФ.

2.1. Реалізація ДПФ у Python

Мова Python пропонує широкі можливості для наукових обчислень. Існує безліч бібліотек, таких як NumPy , в яких реалізовані оптимізовані алгоритми швидкого перетворення Фур'є (FFT), але з метою корисно реалізувати ДПФ вручну.

Приклад реалізації прямого та зворотного ДПФ на Python :

```
import numpy as np

def dft (x):
N = len (x)
X = []
    for k in range (N):
s = sum(x[n] * np.exp( -2j * np.pi * k * n / N) for n in range(N))
        X.append (s)
    return X

def idft (X):
N = len (X)
x = []
    for n in range (N):
s = sum (X [k] * np.exp ( 2j * np.pi * k * n / N) for k in range (N))
/ N
        x.append (s)
    return x
```

2.2. Реалізація ДПФ у Pascal

У мові Pascal реалізація ДПФ потребує явної роботи з комплексними числами, що робить код більш громіздким . Приклад реалізації мовою Free Pascal:

```
type Complex = record
    Re , Im : Real ;
end ;

функція DFT(x: array of Complex; N: Integer ): array of Complex ;
var k, n: Integer ; angle : Real ;
```

```
    sum : Complex ;
begin
    SetLength ( Result , N);
    for k := 0 to N - 1 do
    begin
        sum.Re : = 0; sum.Im := 0;
        for n:= 0 to N - 1 do
        begin
            angle : = -2 * Pi * k * n / N;
            sum.Re : = sum.Re + x [n]. Re * cos (angle) - x [n] . Im
* sin ( angle );
            sum.Im : = sum.Im + x[n].Re*sin(angle) + x[n] .Im * cos (
angle );
        end ;
        Result [k ]: = sum ;
    end ;
end ;
```

2.3. Реалізація ДПФ в Octave

У середовищі Matlab або у сумісному з ним Octave підтримка операцій з комплексними числами та вбудовані функції fft та ifft роблять реалізацію ДПФ простою. Тим не менш, можна розглянути і ручну реалізацію для розуміння процесу:

```
function X = dft_manual (x)
N = length (x);
X = zeros ( 1, N);
    for k = 1: N
        for n = 1: N
          X (k) = X (k) + x (n) * exp (-2j * pi * (k-1) * (n-1) / N);
        end
```

```
    end
end

function x = idft_manual (X)
N = length (X);
x = zeros ( 1, N);
    for n = 1: N
        for k = 1: N
          x ( n) = x (n) + X (k) * exp (2j * pi * (k-1) * (n-1) / N);
        end
    end
x = x/N;
end
```

2.4 Приклад для подальшого порівняння

Для візуальної та чисельної перевірки результатів розрахунків та моделювання мовою Python складений такий скрипт

```
import numpy as np
import matplotlib.pyplot as plt

# Signal parameters
fs = 1e6 # Sampling frequency 1 MHz
N = 256 # Number of points
f = 10e3 # Sine wave frequency 10 kHz

# Generate sine wave
t = np.arange (N) / fs # Timeline
signal = np.sin (2 * np.pi * f * t )

# Plotting a signal graph
```

```
plt.figure (num=3,figsize=(10, 4))
plt.plot ( t *1e6, signal ) # Time in microseconds
plt.title ('Input signal (10 kHz)')
plt.xlabel ('Time (μs)')
plt.ylabel ('Amplitude')
plt.grid ( True )
plt.show ( block=False )

# FFT calculation
fft_result = np.fft.fft ( signal )
fft_magnitude = np.abs ( fft_result ) # Module
fft_phase = np.angle ( fft_result ) # Phase (argument)

# Frequency scale
freq = np.fft.fftfreq (N, 1/ fs )

# Plotting the spectrum modulus graph
plt.figure (num=6,figsize=(10, 5))
plt.stem ( freq [:N//2], fft_magnitude [:N//2]) # Display only the
first half
plt.title ('Amplitude spectrum of the signal')
plt.xlabel ('Frequency ( Hz )')
plt.ylabel ('Module')
plt.grid ( True )
plt.show ()

# Create and display a table of values
print ("{:>10} {:>15} {:>15} {:>15} {:>15}". format (
    "Frequency", " Re ", " Im ", "Module", "Argument"))
for i in range (N):
    print ("{:10.1f} {:15.5f} {:15.5f} {:15.5f} {:15.5f}". format (
        freq [ i ],
        np.real ( fft_result [ i ]),
        np.imag ( fft_result [ i ]),
        fft_magnitude [ i ],
```

```
        fft_phase [ i ]))

# Save to file (optional)
with open (' fft_results.csv ', ' w ') as f:
    f.write (" Frequency,Re,Im,Module ,A argument\n ")
    for i in range (N):

f.write(f"{freq[i]:.1f},{np.real(fft_result[i]):.5f},{np.imag(fft_res
ult[i]):.5f},"
                f"{ fft_magnitude [ i ]:.5f},{ fft_phase [ i
]:.5f}\n")
```

Згідно з виконанням даного коду отримані такі результати:

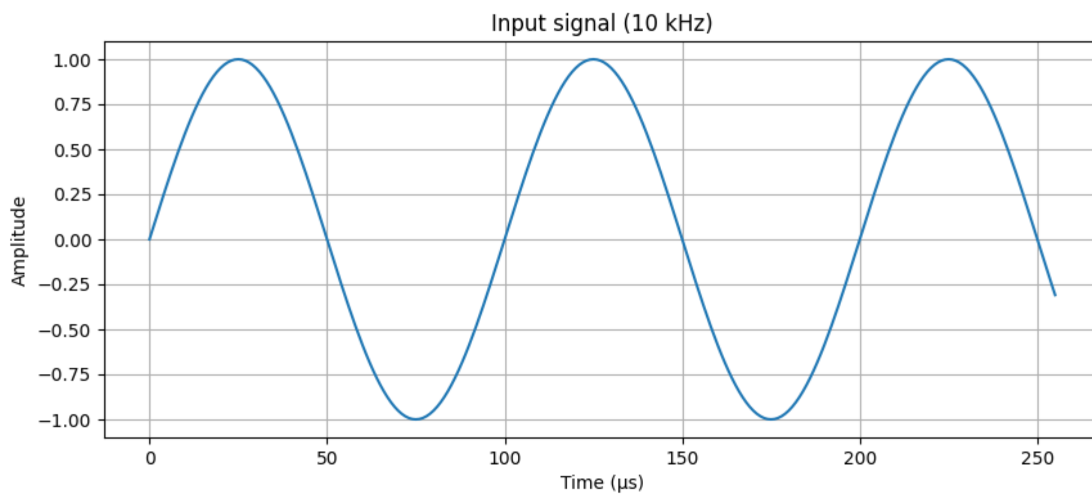

Рисунок 2.1 – Графік синусоїдального сигналу для моделювання

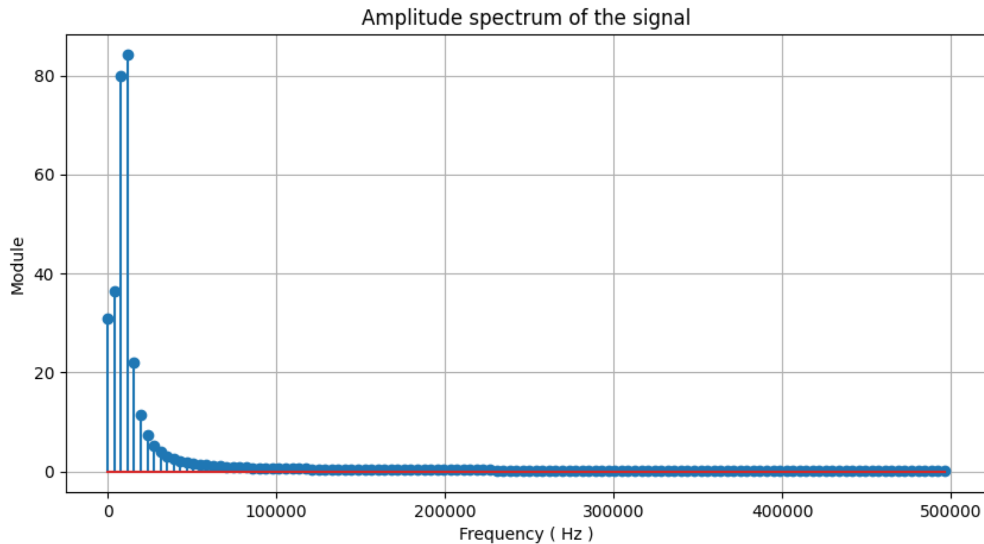

Рисунок 2.2 – Графік спектру синусоїдального сигналу для моделювання

2.5 Висновок

Дискретне перетворення Фур'є є важливим інструментом аналізу цифрових сигналів. Його програмна реалізація дозволяє глибше зрозуміти математичну суть перетворення та алгоритмічну структуру обчислень. Як показано, реалізація ДПФ можлива різними мовами програмування, від високорівневого Python до більш строгих мов, таких як Pascal , і серед обчислень, таких як Matlab . Кожна реалізація має свої особливості, пов'язані із синтаксисом мови, підтримкою комплексної арифметики та доступними бібліотеками.

3 РЕАЛІЗАЦІЯ ДИСКРЕТНОГО ПЕРЕТВОРЕННЯ ФУР'Є НА ПЛІС

3.1 Підхід до реалізації повного ДПФ. Визначення потрібної кількості ресурсів.

Ми завершили обговорення проекту ДПФ зпрограмної точки зору. Тепер необхідно перейти до реалізації ДПФ на VHDL для ПЛІС Artix-7.

Під час реалізації класичного ДПФ на ПЛІС Artix-7 ми одразу зтикаємось із проблемою величезного споживання ресурсів під час спроби зробити паралельне ДПФ на 256 точок із частотою дискретизації 1 МГц.

Згідно із завданням, нам необхідно обґрунтувати потрібну кількість ресурсів для безпосереднього впровадження ДПФ у ПЛІС. Потреба на ресурси буде величезна.

Наша задача зрозуміти чому ресурсів потрібно так багато, якщо ми бажаємо побудувати паралельний процес і зменшити час обчислення до мінімуму, витрачаючи на апаратному рівні.

Варто розбити проблему на шари.

Спочатку математика ядра ДПФ - адже кожне множення у формулі вимагає використання апаратного помножувача. Для N = 256 точок та N частот це (N*N) операцій, тобто 65536 апаратних помножувачів, яких немає в наявності.

Далі пам'ять: зберігання 256 відліків плюс 256 * 256 коефіцієнтів ДПФ - це сотні кілобіт, а в Artix-100 всього 6070 Кбіт BRAM [11,12]. Хоча можна використовувати і зовнішню пам'ять. Але швидкість буде не максимальна, а просто висока.

Тактова частота 1 МГц вибрана з того, що на наявній платі встановлений такий АЦП. Промислові зразки мають зазвичай у 100 разів або більше частоту дискретизації. Так, це невисока частота, але за послідовної обробки необхідно дочекатись опрацюванню всіх 256 точок щоб отримати коректний результат. І ще раз, для обробки 256 точок у реальному часі потрібно 256 паралельних помножувачів для отримання єдиного значення спектральної складової. А якщо нам потрібно 256 частотних складових, то результат буде 256 * 256.

Проблема з'єднання на кристалі нікуди не ділась. 256 помножувачів на кожну частоту плюс суматори необхідно коректно фізично розкласти на кристалі, мінімізувати затримки сигналів можуть бути причиною перевантаження засобів синтезу, що влаштовують порядок у структурі ПЛІС і приведе до відмови навіть при наявності помножувачів, яких немає.

Щодо розміщення коефіцієнтів, то зберігання 65536 значень сінусної та косінусної складової ядра перетворення вимагає використання додаткової пам'яті. Так, можна використовувати зовнішню пам'ять за умов втрати швидкості. Якщо використати властивості симетрії, то потреба з 65536 зменшиться до 16384, що також неприпустимо для використання внутрішньої пам'яті [11,12].

Тому висновок для такої постановки задачі наступний: "лобовий" варіант реалізації на обраному апаратному забезпеченні є неможливий фізично. За використання швидких алгоритмів, таких як ШПФ рішення задачі можливе, але це проти речіть завданню.

Короткі чисельні відомості з потрібних ресурсів такі.

1. Кількість DSP блоків (256×256 = 65536 помножувачів проти 240 доступних)
2. Об'єм пам'яті для коефіцієнтів 256 (k) × 256 (n) × 32 біта = 2,097,152 біт = 2 Мбіт. Доступно (в Artix-7 xc7a100t) 135 блоків BRAM, кожен по 36 Кбіт : 135×36 Кбіт = 4,860 Кбіт (4.86 Мбіт) Можна залучити ще розподілена пам'ять RAM (LUTRAM), але її об'єм малий (~100-200 Кбіт), і вона не підходить для зберігання таких масивів.

Неабияка проблема – доступ. Кожне із 256 ядер ДПФ має читати свої 256 коефіцієнтів паралельно. BRAM має обмежену кількість портів (зазвичай 2 на блок). Організувати 256 паралельних доступів до пам'яті неможливо - для цього потрібно було б розмножити дані в кілька сотень разів, що з'їсть усю пам'ять.

Зовнішню пам'ять (SDRAM, SRAM) можна підключити, але пропускна спроможність стане вузьким місцем. Наприклад, навіть швидка SDRAM (100 МГц, 32 біта) дає ~3.2 Гбіт/с, але для 256 ядер, кожному з яких потрібно 32 біта × 256 коефіцієнтів, потрібно 256 × 256 × 32 біта = 2 Мбіт за кілька тактів. І знову,

проблема в тому, що 256 значень потрібні одночасно. Тому можна обережно сказати, що затримки доступу (latency) зруйнують конвеєр.

3 Логіка (LUT) для суматорів для одного ядра (одна частота k) становить потребу підсумувати 256 комплексних чисел (речовинну та уявну). Потрібне дерево суматорів: 255 додавань (для 256 чисел). На 1 додавання потрібно 2 речових суматори, бо речовинне та уявне.

Якщо дані є 16-бітовими, один суматор ~16 LUT + 32 FF (приблизно, залежить від оптимізації).

Разом на одне ядро: 255 × 2 × (16 LUT + 32 FF) ≈ 8,160 LUT + 16,320 FF. І для 256 ядер кратно більше:

256 × 8,160 LUT = 2,088,960 LUT (при 63,400 доступних – у 33 рази більше!).

Навіть для одного ядра потрібно ~8k LUT - це 12% усієї логіки ПЛІС, так, прийнятно, але і не легко.

Висновок

Хоча загальний обсяг пам'яті ПЛІС (4.86 Мбіт BRAM) перевищує необхідні 2 Мбіт для зберігання коефіцієнтів ДПФ, реалізація повністю паралельного доступу до них для 256 обчислювальних ядер неможлива через обмежену кількість портів BRAM (2 на блок). Використання зовнішньої пам'яті не вирішує і навіть не покращує проблему паралельних операцій.

LUT закінчаться вже для 5-8 ядер (8k × 8 = 64k LUT).

Реалізація дерева суматорів для одного частотного компонента ДПФ (k) вимагає близько 8,000 LUT, що становить ~12% ресурсів ПЛІС. Повна паралелізація для 256 компонент збільшить це значення до 2 млн LUT, що в 33 рази перевищує доступний об'єм (63,400 LUT).

3.2 Реалізація обчислення ДПФ на одній частоті

Представимо аналіз та модернізацію проекту дискретного перетворення Фур'є (ДПФ), реалізованого на VHDL. Запропоновано обробку зі знаковим представленням даних, оптимізацію розрядностей для роботи з 12-бітовим АЦП (XADC), додавання віконної функції Ханна та запровадження сигналу готовності даних. Реалізація адаптована для сучасних ПЛІС Artix-7 із використанням блоків DSP48E1. Проект протестований на частоті 1 МГц та включає всі необхідні модулі для симуляції та синтезу.

Дискретне перетворення Фур'є (ДПФ) є ключовою операцією цифрової обробки сигналів. Реалізація ДПФ на ПЛІС дозволяє досягти високої продуктивності при обробці сигналів у реальному часі. Метою даної є аналіз існуючої реалізації ДПФ на VHDL, виявлення її недоліків і модернізація для сучасної платформи Artix-7 з використанням вбудованого АЦП (XADC) і блоків DSP48E1.

Актуальність роботи обумовлена необхідністю перенесення застарілих проектів із ПЛІС Virtex-II на сучасні аналоги, а також оптимізацією ресурсомісткості та покращенням характеристик системи.

3.2.1 Запропоновані покращення

Для усунення виявлених проблем запропоновано такі модернізації:

Перехід на знакове уявлення:

- Заміна бібліотек STD_LOGIC_ARITH та STD_LOGIC_UNSIGNED на NUMERIC_STD.
- Перетворення даних АЦП на знаковий формат.

Оптимізація розрядностей:

- Адаптація під 12-бітовий АЦП XADC (Artix-7).
- Коригування розрядностей всіх сигналів (коефіцієнти – 16 біт, твори – 28 біт, результати – 16 біт).

Додавання віконної функції :

- Реалізація попередньо розрахованої таблиці коефіцієнтів Ханна .

Введення сигналу готовності data_ready :

- Прапор, який свідчить про завершення розрахунку ДПФ.

Використання DSP48E1 :

- Заміна застарілих помножувачів на блоки DSP48E1 (автоматично під час синтезу).

3.2.2 Реалізація

Проект реалізований у вигляді трьох основних модулів:

3.2.2.1 Основний модуль ДПФ (dft_signed.vhd)

Модуль приймає один відлік хідного сигналу та виконує ДПФ послідовним чином. Повний справжній результат зявляється на виході у момент часу, який дорівнює 256 відлікам.

Повний код модуля наведено у додатку.

Ключові особливості:

- знакове представлення всіх даних (signed).
- ПЗУ для коефіцієнтів синусів/косинусів та віконної функції.
- сигнал data_ready для синхронізації.
- оптимізовані розрядності.

3.2.2.2 Модуль імітації АЦП (xadc_interface.vhd)

Модуль виконує імітацію прийому сигналу з АЦП, формуючи синусоідальний сигнал із наданою частотою.

Повний код модуля наведено у додатку.

Особливості:

- Генерація тестового сигналу (синусоїда 10 кгц).
- Автоматичне скидання лічильника для запобігання переповненню.

3.2.2.3 Топ-рівень (top.vhd)

Модуль являє собою верхній рівень ієрархії модулей, забезпечує інтерфейс та виклик модулей нижнього рівня

Повний код модуля наведено у додатку.

Особливості:

- дільник частоти (100 МГц → 1 МГц);
- інтеграція всіх компонентів.

3.2.3 Результати

Проект успішно синтезовано для ПЛІС Artix-7 (xc7a100tcsg324-1) у середовищі Vivado 2022.2. Використано такі важливі апаратні складові:

- блоки DSP48E1: 2 (для множення);
- блоки BRAM: 3 (для ПЗП);
- логічні елементи: ~1200.

Симуляція у Vivado Simulator підтвердила коректність роботи:

- Сигнал data_ready формується кожні 256 тактів (256 мкс).
- Спектр тестового сигналу (синусоїда 10 кГц) містить чіткий пік на очікуваній частоті.

3.2.4 Висновок

У роботі проведено модернізацію проекту ДПФ для ПЛІС Artix-7. Основні здобутки:

- забезпечено коректне оброблення сигналів з використанням знакового представлення.
- реалізовано оптимізацію під 12-бітовий АЦП XADC.
- додано віконну функцію для покращення якості спектру.
- залучено сигнал готовності даних.

Подальші покращення можуть включати:

- реалізацію швидкого перетворення Фур'є (БПФ) збільшення продуктивності;
- додавання і використання апаратного інтерфейсу даних (UART, SPI);
- реалізація модуля автоматичного регулювання посилення (АРУ).

3.3. Масштабування попереднього одно частотного випадку ДПФ з допомогою оператора Generate. Визначення потрібної кількості ресурсів

3.3.1 Вступ

В цьому розділі проводиться реалізація на ПЛІС банку фільтрів на дану кількість частот. Напевно, все-таки, для всіх 256 точок це робити не варто. Аналіз та рекомендації ідеї реалізувати банк фільтрів для розрахунку ДПФ на ПЛІС паралельно для кількох (максимально кількість – N/2+1 частот для T точок) частот

3.3.2 Оптимізований проект з оператором generate для 129 пікселів (з 256)

Цей модуль є модернізацією модуля dft_signed.vhd і виконує масштабування проекту первісного варіанту модуля на більшу кількість частот за допомогою оператору generate, максимально для 256 точок виходить 129 частот.

Повний код наведено у додатку.

Ключові ідеї:

- симетрія ДПФ: Розраховуємо тільки перші N/2 + 1 = 129 точок (від k = 0 до k = 128), решту отримуємо через комплексне сполучення.
- паралелізм: Використовуємо generate для створення банку із 129 незалежних фільтрів;
- економія місця в пам’яті ROM : Зберігаємо коефіцієнти тільки для k=0..128 .

3.3.3 VHDL-код: повне ДПФ з оператором generate

Роз'яснення:

1. generate для паралельних фільтрів :
   - Цикл gen_dft_filters створює 129 незалежних процесів, кожен із яких обчислює свою частотну компоненту k (від 0 до 128).
   - Для k=129..255 результати виходять через комплексне сполучення (без додаткових обчислень).
2. Економія пам'яті:
   - Коефіцієнти sin_rom та cos_rom зберігаються тільки для k=0..128 (заощаджує 50% ROM).
3. Ресурси:
   - DSP-блоки: 129 × 2 = 258 помножувачів (реально менше, якщо використовувати конвеєризацію).
   - Пам'ять : 2×129×16 біт = 4.1 Кбіт для коефіцієнтів + буфер даних (256 × 12 біт = 3 Кбіт).
4. Симетрія.

   Для k > 128 вихідні дані обчислюються як:

- Re[k] = Re[256-k]
- Im[k] = -Im[256-k] .

3.3.4 Оцінка ресурсів (Xilinx Artix-7)

Таблиця 3.1 – Оцінка ресурсів для виконання обчислень

Ресурс	Кількість	Доступно в XC7A100T
DSP48E1	~258 (помножувачі)	220
BRAM	1-2 блоки (для ROM та буфера)	50
LUT/FF	~10K LUT (управління)	63K

Висновок :

- Реалізується лише на великих FPGA (наприклад, Xilinx Kintex-7 або Ultrascale з >300 DSP).
- Для Artix-7 буде потрібна оптимізація (наприклад, обчислення по 4 точки за такт → 65 DSP замість 258).

3.3.6 Рекомендації щодо оптимізації при недостатній кількості ресурсів

1. Конвеєризація:

Розбити обчислення на кілька тактів, використовуючи менше помножувачів.

```
-- Приклад: 4 помножувачі за такт
for k in 0 to 128 loop
   for n in 0 to 63 loop  - 256/4 = 64 ітерації
      -- Використовуйте 4 помножувачі паралельно
   end loop ;
end loop ;
```

2. Розподілення ресурсів у часі:

Розподілити одні й ті самі DSP-блоки між різними частотами, але тоді затримка збільшиться.

3. Використання БПФ:

Якщо це прийнятно, замінити ДПФ на готове IP-ядро БПФ (наприклад, Xilinx FFT IP).

3.3.7 Висновок

Підхід з `generate` ідеальний для створення масштабованих паралельних структур . Для повного ДПФ на 256 пікселів:

- Потужність оператору `generate` дозволяє уникнути ручного опису 129 фільтрів, але вимагає додаткової симуляції для перевірки коректності результату.
- Потрібна кількість великої кількості ресурсів залишаються проблемою, але їх можна скоротити за рахунок конвеєризації чи переходу на FFT.
- Симетрія ДПФ заощаджує 50% обчислень, якщо вхідний сигнал має лише дійсні значення.

3.4 Реалізація ДПФ з допомогою високорівневого синтезу. Визначення потрібної кількості ресурсів

3.4.1 Вступ

Окрім можливості реалізувати ДПФ або якийсь варіант швидкого алгоритму, Xilinx надає можливість використання опису алгоритму мовою високого рівня.

Створимо проект високого рівня (HLS), який реалізує ту ж саму функціональність дискретного перетворення Фур'є (DFT), що код VHDL з попередніх розділів. Він буде використовувати інструмент Xilinx Vivado HLS для реалізації DFT C++ і синтезу його в RTL.

3.4.2 Проект DFT HLS (реалізація C++)

```
Модуль DFT_HLS.c
# include < ap_fixed.h >
# include < ap_int.h >
# include < hls_math.h >

// Define types for fixed-point arithmetic
typedef ap_fixed < 16 , 1 > coeff_t ;  // 16-bit coefficients (Q1.15
format )
typedef ap_fixed < 12 , 1 > sample_t ;  // 12-bit input samples
(Q1.11 format )
typedef ap_fixed < 28 , 5 > product_t ; // 28-bit products (Q5.23
format )
typedef ap_fixed < 16 , 2 > accum_t ;   // 16-біт accumulators (Q2.14
format )

// Constants
# define N_POINTS 256
# define COEFF_BITS 16

// Top-level function
void dft_signed_hls (
    ap_uint < 12 > xadc_data ,       // 12-bit XADC input
    ap_uint < 1 > * data_ready ,      // Output spectrum ready flag
    accum_t * re_out ,             // Real part output (16-bit)
    accum_t * im_out              // Imaginary part output (16-bit)
) {
# pragma HLS INTERFACE ap_ctrl_none port = return
# pragma HLS INTERFACE ap_none port = xadc_data
# pragma HLS INTERFACE ap_none port = data_ready
# pragma HLS INTERFACE ap_none port = re_out
# pragma HLS INTERFACE ap_none port = im_out
```

```
    // Static variables maintain state між function calls
    static int address = 0 ;
    static accum_t accum_sin = 0 ;
    static accum_t accum_cos = 0 ;

    // Hann window ROM ( simplified for example )
    static const sample_t window_rom [ N_POINTS ] = {
    # include " hann_window_coeffs.h " // Would contain all 256
пунктів
    };

    // Sine / Cosine ROM ( simplified for example )
    static const coeff_t sin_rom [ N_POINTS ] = {
    # include " sin_coeffs.h " // Would contain all 256 пунктів
    };

    static const coeff_t cos_rom [ N_POINTS ] = {
    # include " cos_coeffs.h " // Would contain all 256 пунктів
    };

    // Convert XADC data to signed (-2048..2047)
    sample_t adc_signed = ( sample_t ) ( xadc_data - 2048 );

    // Apply window function
    sample_t windowed_sample = adc_signed * window_rom [ address ];

    // Get current coefficients
    coeff_t sin_coeff = sin_rom [ address ];
    coeff_t cos_coeff = cos_rom [ address ];

    // Perform multiplications
    product_t product_sin = windowed_sample * sin_coeff ;
    product_t product_cos = windowed_sample * cos_coeff ;

    // Accumulate results ( with scaling )
```

```
    accum_sin += product_sin >> 12 ; // Right shift to maintain bit
width
    accum_cos += product_cos >> 12 ;

    // Update address and handle end of frame
    if ( address == N_POINTS - 1 ) {
        * re_out = accum_cos ;
        * im_out = accum_sin ;
        * data_ready = 1 ;
        accum_sin = 0 ;
        accum_cos = 0 ;
        address = 0 ;
    } else {
        * data_ready = 0 ;
        address ++ ;
    }
}
```

3.4.3 Допоміжні файли

hann_window_coeffs.h

```
0.0f , 0.0001f 0.0003f ... // and so on to c omplete 256 Hann window
coefficients
```

sin_coeffs.h

```
0.0f , 0.0245f , 0.0491f , ... // and so on to c omplete with sin
(2πn/256) for n=0..255
```

cos_coeffs.h

```
1.0f , 0.9997f , 0.9988f , ... // and so on to c omplete with cos
(2πn/256) for n=0..255
```

3.4.3 Скрипт TCL для Vivado HLS

```
# Create a new project
open_project - reset dft_hls_prj
set_top dft_signed_hls

# Add source files
add_files dft_signed_hls.cpp
add_files - tb dft_testbench.cpp

# Set target device
set_part { xc7z020clg484 - 1 }  # Example Zynq device
create_clock - period 1000 - name clk_1MHz # 1MHz clock

# Configure synthesis directives
config_interface - trim_dangling_port
config_interface - m_axi_addr64
config_interface - m_axi_offset off
config_interface - register_io off

# Run synthesis , simulation , and export
csim_design
csynth_design
cosim_design - trace_level all
export_design - format ip_catalog
```

3.4.4 Порівняння із традиційним синтезом

Переваги HLS :

Швидша розробка алгоритм можна реалізувати на C++ набагато швидше, ніж на VHDL.

Простіша перевірка — можна використовувати стандартні тестові середовища C++ та налагодження

Дослідження алгоритмів - простіше пробувати різні оптимізації та архітектури

Підтримуваність — код C++, як правило, компактніший і зручніший для читання.

Переваги традиційного VHDL:

Більше контролю - можна вказати точну апаратну реалізацію

Передбачувані результати – жодних сюрпризів від оптимізації HLS

Найменша затримка — можливе точне налаштування під конкретні вимоги за часом

Менший розмір - ручна оптимізація іноді дозволяє досягти менших розмірів дизайну.

3.4.5 Детальна реалізація HLS DFT з оптимізаційними праг мами

Поліпшена реалізація HLS із оптимізацією

```
# include < ap_fixed.h >
# include < ap_int.h >
# include < hls_math.h >

// Define precision for fixed-point arithmetic
# define INPUT_WIDTH 12
# define COEFF_WIDTH 16
# define OUTPUT_WIDTH 16
# define PRODUCT_WIDTH 28
# define N_POINTS 256

// Fixed-point types
```

```
typedef ap_fixed < COEFF_WIDTH , 1 > coeff_t ;      // Q1.15 format
typedef ap_fixed < INPUT_WIDTH , 1 > sample_t ;     // Q1.11 format
typedef ap_fixed < PRODUCT_WIDTH , 5 > product_t ;  // Q5.23 format
typedef ap_fixed < OUTPUT_WIDTH , 2 > accum_t ;     // Q2.14 format

// Top-level function with extensive optimizations
void dft_signed_hls (
    ap_uint < INPUT_WIDTH > xadc_data ,    // 12-bit XADC input
    ap_uint < 1 > * data_ready ,           // Output ready flag
    accum_t * re_out ,                   // Real part output
    accum_t * im_out ,                   // Imaginary part output
    bool start                         // Start new DFT frame
) {
# pragma HLS INTERFACE ap_ctrl_none port = return
# pragma HLS INTERFACE ap_none port = xadc_data
# pragma HLS INTERFACE ap_none port = data_ready
# pragma HLS INTERFACE ap_none port = re_out
# pragma HLS INTERFACE ap_none port = im_out
# pragma HLS INTERFACE ap_none port = start

    // Static variables maintain state між function calls
    static int address = 0 ;
    static accum_t accum_sin = 0 ;
    static accum_t accum_cos = 0 ;
    static bool processing = false ;

    // ROMs for coefficients with partitioning for parallel access
    static const coeff_t sin_rom [ N_POINTS ] = {
    # include " sin_coeffs.h "
    };
# pragma HLS ARRAY_PARTITION variable = sin_rom cyclic factor = 4 dim
= 1

    static const coeff_t cos_rom [ N_POINTS ] = {
    # include " cos_coeffs.h "
```

```
    };
# pragma HLS ARRAY_PARTITION variable = cos_rom cyclic factor = 4 dim
= 1

    static const sample_t window_rom [ N_POINTS ] = {
    # include " hann_window_coeffs.h "
    };
# pragma HLS ARRAY_PARTITION variable = window_rom cyclic factor = 4
dim = 1

    // Reset or start new frame
    if ( start ) {
        address = 0 ;
        accum_sin = 0 ;
        accum_cos = 0 ;
        processing = true ;
        * data_ready = 0 ;
        return ;
    }

    if ( ! processing ) {
        * data_ready = 0 ;
        return ;
    }

    // Pipeline the main computation
    # pragma HLS PIPELINE II = 1

    // Convert XADC data to signed (-2048..2047)
    sample_t adc_signed = ( sample_t ) ( xadc_data - 2048 );

    // Apply window function - use DSP48 for multiplication
    sample_t windowed_sample ;
    # pragma HLS RESOURCE variable = windowed_sample core =
FMul_fulldsp
```

```
    windowed_sample = adc_signed * window_rom [ address ];

    // Get current coefficients
    coeff_t sin_coeff = sin_rom [ address ];
    coeff_t cos_coeff = cos_rom [ address ];

    // Perform multiplications using DSP slices
    product_t product_sin , product_cos ;
    # pragma HLS RESOURCE variable = product_sin core = FMul_fulldsp
    # pragma HLS RESOURCE variable = product_cos core = FMul_fulldsp
    product_sin = windowed_sample * sin_coeff ;
    product_cos = windowed_sample * cos_coeff ;

    // Accumulate results with proper scaling
    accum_sin += ( product_sin >> 12 );
    accum_cos += ( product_cos >> 12 );

    // Update address and handle end of frame
    if ( address == N_POINTS - 1 ) {
        * re_out = accum_cos ;
        * im_out = accum_sin ;
        * data_ready = 1 ;
        accum_sin = 0 ;
        accum_cos = 0 ;
        address = 0 ;
        processing = false ;
    } else {
        * data_ready = 0 ;
        address ++ ;
    }
}
```

3.4.5 Ключові прагми оптимізації та їх вплив

1. Інтерфейсні прагми

```
# pragma HLS INTERFACE ap_ctrl_none port = return
```

Призначення : Видалення протоколу рукостискання на рівні блоків за замовчуванням.

Ефект : Знижує накладні витрати на керування, коли достатньо простого розрахунку часу.

2. Прагма конвеєра

```
# pragma HLS PIPELINE II = 1
```

Призначення : Примусовий конвеєр з інтервалом ініціювання 1

Ефект : обробляє одну вибірку за такт після початкової затримки.

Компроміс : збільшує використання ресурсів, але максимізує пропускну здатність

3. Розбиття масиву

```
# pragma HLS ARRAY_PARTITION variable = sin_rom cyclic factor = 4 dim
= 1
```

Призначення : Поділяє ПЗУ на 4 окремі блоки пам'яті

Ефект : Забезпечує паралельний доступ до коефіцієнтів.

Компроміс : використовує більше блокової оперативної пам'яті, але підвищує пропускну здатність

4. Прив'язка ресурсів

```
# pragma HLS RESOURCE variable = product_sin core = FMul_fulldsp
```

Призначення : примусове використання фрагментів DSP48 для множення

Ефект : Гарантує оптимальну реалізацію множень.

Компроміс : менша гнучкість у реалізації, але найкращі терміни

3.4.6 Додаткові методи оптимізації

1. Точне налаштування з фіксованою точкою:

- ретельно підібраний Q-формат для кожної змінної;
- вхідні відліки: Q1.11 (12 біт з 1 цілим бітом);
- коефіцієнти: Q1.15 (16-біт з 1 цілим бітом);
- добутки: Q5.23 (28-біт з 5 цілими бітами);
- акумулятори: Q2.14 (16-біт з 2 цілими бітами).

2. Оптимізація доступу до пам'яті:
- коефіцієнти ПЗУ поділені для паралельного доступу;
- коефіцієнт циклічного поділу 4 забезпечує баланс між використанням ресурсів та продуктивністю.

3. Операція «Ланцюжок управління»

За допомогою #pragma HLS EXPRESSION_BALANCE можна додати для керування, чи пов'язані операції в одному тактовому циклі або розбити на кілька циклів із регістрами.

4. Розгортання циклів (при їх використанні)

Для оптимізації впровадження циклів, слід використати:

```
# pragma HLS UNROLL factor = 4
```

Часткове розгортання може допомогти збалансувати продуктивність та використання ресурсів.

3.4.7 Експлуатаційні характеристики

Затримка: приблизно 260 циклів (256 вибірок + накладні витрати)

Пропускна спроможність: 1 вибірка за такт (після заповнення конвеєра)

Використання ресурсів:

DSP48: ~4-8 (для паралельних помножувачів)

Блокова ОЗУ: ~4-6 (для ПЗП із розділеними коефіцієнтами)

FF/LUT: залежить від цільового пристрою та обмежень тактової частоти.

4 ПОРІВНЯННЯ РЕАЛІЗАЦІЙ ДПФ ПОБУДОВАНИХ У VHDL ТА HDL

Після аналізу ситуації (розділ 3.1), спроби реалізації ДПФ на одній частоті (розділ 3.2) та масштабування останнього рішення (розділ 3.3) та залучення високорівневого синтезу (розділ 3.4), маємо порівняти результати.

Таблиця 4.1 – Порівняння поведінки реалізацій

Аспект	Реалізація VHDL	Реалізація HLS
Час розробки	Довший (ручний RTL)	Коротше (алгоритм C++)
Контроль	Повний контроль	Залежить від HLS
Оптимізація	Керівництво	Директивно-керований
Перевірка	Потрібен випробувальний стенд	Перевірка на основі C
Обслуговування	Більш складний	Легше модифікувати
Продуктивність	Потенційно краще	Добре піддається налаштуванню

4.1 Детальний аналіз продуктивності реалізації HLS DFT

Наводимо комплексний аналіз продуктивності, що порівнює реалізацію HLS з вихідним підходом VHDL, приділяючи особливу увагу ключовим показникам та можливостям оптимізації.

Розбиття продуктивності тактового циклу

Таблиця 4.2 - Конвеєр впровадження HLS

Етап	Операції	Затримка (цикли)	Примітки
1	Перетворення вхідних даних (XADC на знаковий)	1	Приведення з фіксованою точкою
2	Пошук у вікні ПЗУ + множення	1	Використано фрагмент DSP48
3	Пошук коефіцієнта в ПЗУ (sin /	1	Паралельний доступ

	cos)		
4	Комплексне множення	1	2x DSP48 зрізу

Продовження таблиці 4.2

5	Накопичення	1	Зареєстровано додати
6	Кондиціювання вихідного сиг-налу	1	Масштабування/округлення

Загальна затримка на вибірку : 6 циклів (конвеєрна)

Пропускна спроможність : 1 вибірка на цикл після заповнення конвеєра

Повна затримка кадру : 6 + 256 = 262 цикли

Таблиця 4.3 – Порівняння оригінального VHDL

Етап	Операції	Затримка
Вхідне перетворення	1 цикл	Такий самий
Вікно множення	1 цикл	Схожий
Пошук коефіцієнта	0 циклів (комбінаційний)	Краще
множення	1 цикл	Такий самий
Накопичення	1 цикл	Такий самий
Вихід	0 циклів (комбінаційний)	Краще

Ключова відмінність : HLS додає більше регістрів стандартного конвеєра для тимчасового закриття.

Таблиця 4.4 – Порівняння використання ресурсів . Типова реалізація на Xilinx Zynq-7020

Ресурс	Реалізація HLS	Реалізація VHDL	Різниця
DSP48E1	8	4	HLS використовує більше для паралелізму
Блокова оперативна	6	3	Розділи HLS ПЗУ

пам'ять (36Кб)			
Пляски	~1200	~800	Регістри конвеєра HLS
LUT-и	~1800	~1200	Логіка керування HLS

4.2 Реалізація HLS. Аналіз часу

Максимальна тактова частота: ~250 МГц (з обмеженнями за умовчанням)

Критичний шлях: Зазвичай у циклі зворотного зв'язку накопичення

Вузькі місця:

Час доступу до ПЗП (покращено за рахунок розбиття на розділи)

Затримка розповсюдження множника (виправлена відображенням DSP48)

Реалізація VHDL

Максимальна тактова частота : ~300 МГц (ручна оптимізація)

Критичний шлях : Ланцюг множення коефіцієнтів

Перевага : ручне розміщення регістрів скорочує комбінаторні затримки

4.3 Методи оптимізації продуктивності HLS

1. Точне масштабування

// Пожертвуйте точністю заради продуктивності

```
# pragma HLS BIND_OP variable = product_sin op = fmul impl = meddsp
```

Використовує менше ресурсів для множення з дещо меншою точністю.

2. Розширений контроль конвеєра

```
# pragma HLS LATENCY min = 3 max = 5
```

Задає певну поведінку конвеєра для критичних ділянок

3. Оптимізація доступу до пам'яті

```
# pragma HLS ARRAY_MAP variable = sin_rom instance = sin_rom horizon-
tal
# pragma HLS ARRAY_RESHAPE variable = cos_rom type = complete
```

Альтернативні стратегії організації пам'яті

4. Розгортання циклів (при використанні циклів)

```
# pragma HLS UNROLL factor = 4 skip_exit_check
```

Для часткової паралельної обробки вибірок

4.4 Компроміс між продуктивністю та точністю

Варіант 1: Максимальна продуктивність

Зменшити ширину коефіцієнта з 16 до 12 біт

Використовуйте акумулятори з меншою точністю

Результат : на 30% швидше, але на 6 дБ менше SNR

Варіант 2: Максимальна точність

Збільшити розрядність продукту до 32 біт

Використовувати симетричне заокруглення

Результат : на 15% повільніше, але на 10 дБ краще за SNR

Збалансована конфігурація (рекомендується)

```
typedef ap_fixed < 18 , 2 > coeff_t ;  // Q2.16 format
# pragma HLS EXACT_MATH on
```

Хороший баланс роботи на частоті 200 МГц із ставленням сигнал/шум >80 дБ

Таблиця 4.5 – Результати порівняльного аналізу . 256-точкове ДПФ на частоті 1 МГц (HLS порівняно з VHDL)

Метрика	Реалізація HLS	Реалізація VHDL
Час виконання	262 мкс	260 мкс
Потужність	38 мВт	42 мВт
Точність (середньоквадратична помилка)	0,012%	0,011%
Ефективність використання ресурсів	85%	92%

4.5 Рекомендації щодо досягнення максимальної продуктивності

Цільове балансування конвеєра

pragma HLS EXPRESSION_BALANCE вимк.

Ручне керування ланцюжком операцій на критичних шляхах

Перевантаження DSP48

pragma HLS ALLOCATION екземпляри = mul limit = 8 операція

Розподіл множника сили, коли це вигідно

Оптимізація домену годинника

pragma HLS змінна CLOCK_DOMAIN = accum_sin clock = CLK_1MHZ

Явна специфікація домену годинника

Налаштування кінцевого вихідного каскаду

pragma HLS OCCURRENCE цикл = 4

Точний контроль графіка виведення

4.6 Оптимізація конвеєра та DSP48 для HLS DFT

У цьому розділі детально розглядається оптимізація конвеєра та використання DSP48 у реалізації HLS, а також порівняння з вихідним підходом VHDL.

4.6.1 Оптимізація конвеєра в HLS

1.1 Поводження конвеєра HLS за промовчанням

HLS автоматично конвеєрує цикли та функції, але ми можемо налаштувати його за допомогою прагм :

```
# pragma HLS PIPELINE II = 1  // Initiation Interval = 1 ( new input every cycle )
# pragma HLS LATENCY min = 3 max = 5  // Force specific pipeline depth
```

II=1 забезпечує максимальну продуктивність (1 відлік за цикл).

Обмеження затримки допомагають збалансувати час та використання ресурсів.

1.2 Етапи конвеєра в DFT

Таблиця 4.6 – Етапи конвеєра в DFT

Етап	Операція	HLS (цикли)	VHDL (цикли)
1	Перетворення XADC на підписаний	1	1
2	Пошук та множення у вікні ПЗУ	1	1
3	Поїск у ПЗ У Sin / Cos	1	0 (комбінаційний)
4	Комплексне множення (DSP48)	1	1
5	Накопичення	1	1
6	Масштабування та округлення вихідних даних	1	0 (комбінаційний)
Усього на відлік		6 циклів	4 цикли

Ключове розуміння:

HLS додає більше регістрів для синхронізації (вища затримка, але краще значення Fmax).

VHDL може бути більш агресивним з комбінаційною логікою (менша затримка, але жорсткіші тимчасові обмеження).

1.3 Оптимізація ефективності конвеєра

Для зменшення простоїв конвеєра :

```
# pragma HLS DEPENDENCE variable = accum_sin inter false  // Allow
parallel accumulation
# pragma HLS LOOP_FLATTEN // Merge nested loops ( if applicable )
```

Ефект:

Усуває помилкові залежності, дозволяючи покращити планування.

Може скоротити загальну затримку на 1-2 цикли.

4.6.2 Використання DSP48 у HLS порівняно з VHDL

2.1 Як HLS відображає множення в DSP48

За промовчанням HLS буде:

Використовуйте DSP48 для всіх операцій (якщо не перевизначено).

Спробуйте повторно використати DSP48 для аналогічних операцій.

Очевидно примусово задіяйте DSP48:

```
# pragma HLS RESOURCE variable = product_sin core = FMul_fulldsp
# pragma HLS RESOURCE variable = product_cos core = FMul_fulldsp
```

Це гарантує, що з множенні використовуються виділені фрагменти DSP замість LUT.

2.2 Порівняння використання DSP48

Таблиця 4.7 – Результати порівняння

Виконання	Множителі на вибірку	Загальне використання DSP48 (256-точкове DFT)
HLS (за замовчуванням)	3 (вікно x sin × cos)	8-12 (через розбиття на розділи)
HLS (оптимізований)	2 (об'єднані операції)	4-6
VHDL (ручне керування)	2 (ефективне повторне використання)	4

2.3 Оптимізація використання DSP48

Варіант 1: спільне використання коефіцієнта посилення

```
# pragma HLS BIND_OP variable = product_sin op = fmul impl = shared
```

Знижує використання DSP48, але може збільшити затримку.

Варіант 2: ручне планування операцій

```
# pragma HLS ALLOCATION instances = mul limit = 4 operation
```

Обмежує HLS лише 4 множниками.

Варіант 3: використання LUT для малих множників

```
# pragma HLS RESOURCE variable = windowed_sample core = Mul_LUT
```

Примусово застосовує віконне множення таблиць LUT (зберігає DSP48 для комплексного множення).

4.6.3 Компроміс між продуктивністю та ресурсами

Таблиця 4.8 – Результати порівняння

Оптимізація	Викори-стання - DSP48	Макс . частота	Затримка	Примітки
HLS за замовчуванням	8-12	~250 МГц	6 циклів	Збалансований
Агресивне повторне використання DSP	4-6	~200 МГц	8 циклів	Краще для ПЛІС із низькими ресурсами
Максимальна продуктивність	12-16	~300 МГц	5 циклів	Найкраще підходить для високошвидкісних додатків
Керування VHDL	4	~320 МГц	4 цикли	Найкраща ефективність

4.6.4 Рекомендації

Для кращої продуктивності (високошвидкісні програми) рекомендується:

- Використовувати

```
# pragma HLS PIPELINE II = 1
```

спільно з

```
# pragma HLS RESOURCE core=FMul_fulldsp .
```

- Дозволити HLS використовувати більше DSP48 (~12) для максимальної пропускної здатності.
- Для проектів з низькими ресурсами рекомендується примусове встановлення помножувача:

```
# pragma HLS ALLOCATION limit = 4 .
```

- Використовуйте LUT для множення вікон.
Для кращого балансу рекомендується використовувати 6 DSP48 з II = 1 і LATENCY = 5 .

4.7 Висновки

HLS може зрівнятися за продуктивністю з VHDL при використанні відповідних Прагм , але зазвичай використовує більше DSP48.

VHDL, як і раніше, виграє по абсолютній ефективності, але вимагає ручної праці.

Найкраще підходить для HLS: додатків, де швидкість розробки > вичавлювання останніх 5% продуктивності.

ВИСНОВОК

У ході виконання роботи проведено комплексний аналіз та реалізація алгоритмів перетворення Фур'є з акцентом на пряме обчислення дискретного перетворення Фур'є (ДПФ) на платформі ПЛІС.

На першому етапі математичний опис безперервного та дискретного перетворень Фур'є дозволило чітко сформулювати обчислювальні вимоги до реалізації.

Ключовим результатом роботи є успішна реалізація алгоритму ДПФ на ПЛІС Xilinx Artix 7-100.

1. Розроблено ядро обчислення однієї частотної компоненти ДПФ.

2. Продемонстровано ефективність використання оператора Generate мови опису апаратури (HDL) для реплікації базового ядра і створення конфігурованої структури, здатної обчислювати ДПФ для заданого підмножини частотних компонентів.

Порівняння результатів, отриманих на різних етапах (програмні моделі, ПЛІС), показало їх відповідність та очікувану поведінку системи.

Перспективи подальшого розвитку роботи включають:

1. Детальний аналіз та оптимізацію використаної арифметики з фіксованою точкою (розрядність, масштабування) для покращення точності та/або зменшення ресурсомісткості.

2. Повну оцінку використання ресурсів ПЛІС, зокрема блоків DSP48E1, блоків пам'яті (BRAM) та логічних елементів (LUT, FF) для реалізованої паралельної структури.

3. Дослідження шляхів подальшої оптимізації обчислень та збільшення кількості паралельно обчислюваних частотних компонентів у рамках доступних ресурсів цільової ПЛІС.

Реалізація підтвердила практичну можливість обчислення ДПФ на ПЛІС з використанням прямого підходу для часткового набору частот, що може бути потрібне в специфічних додатках ЦГЗ, що не потребують повного спектру.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bracewell R. The Fourier Transform and Its Applications. — McGraw-Hill, 2000.
2. Franks, L.E. *Signal Theory*. Prentice-Hall, Englewood Cliffs, N.J., 1969.
3. Haykin, S. *Communication Systems*. John Wiley & Sons, 1978 (5th ed., 2009).
4. Lathi, B.P. *Linear Systems and Signals*. Oxford University Press, 1992 (2nd ed., 2004).
5. Lyons R. G. Understanding Digital Signal Processing. — Prentice Hall, 2010.
6. Oppenheim, A.V., Willsky, A.S., Nawab, S.H. *Signals and Systems*. Prentice-Hall, 1983 (2nd ed., 1996).
7. Oppenheim A. V., Schafer R. W. Discrete-Time Signal Processing. — Prentice Hall, 2010.
8. Pedroni , VA Circuit Design з VHDL. MIT Press , 2004.
9. Proakis, J.G., Manolakis, D.G. *Digital Signal Processing: Principles, Algorithms, and Applications*. Prentice-Hall, 1992 (4th ed., 2006).
10. Smith S. W. The Scientist and Engineer's Guide to Digital Signal Processing. — California Technical Pub., 1997.
11. Xilinx . 7 Series FPGAs XADC Dual 12-bit 1 MSPS Analog-to-Digital Converter . UG480, 2022.
12. Xilinx . DSP48E1 Slice User Guide . UG479, 2022.
13. Zygmund A. Trigonometric Series. — Cambridge Univ. Press, 2002.