Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
(рівень вищої освіти)

Методи автоматизованого проектування цифрових фільтрів на платформі SoC
(тема)

Виконав:

здобувач 2 року навчання, групи СКСм-23-2

_____Карась Д.В._____
(прізвище, ініціали)

Спеціальність *123 _ Комп'ютерна інженерія*

Тип програми ____*освітньо-професійна*_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____

*Спеціалізовані комп'ютерні системи*_____
(повна назва освітньої прграми)

Керівник роботи ___доц.Шкіль О.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління _____

Кафедра_____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти____другий (магістерський)_____

Спеціальність_____123_ Комп'ютерна інженерія _____
(шифр і назва)

Тип програми _____Освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____Спеціалізовані комп'ютерні системи_____
(повна назва

ЗАТВЕРДЖУЮ:

Зав. кафедри ___  ___
(підпис)

«__02__»___09____ 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві_____Карасю Даніілу Васильовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту)_ Методи автоматизованого проектування цифрових фільтрів на платформі SoC_____
_____

затверджена наказом по університету від «__08_»__11__2024р. № _1189 Ст_____

2. Термін подання студентом роботи до екзаменаційної комісії _____10.01.2025_____

3. Вихідні дані до роботи (проекту)_____
Налагджувальна плата ZebBoard з кристалом ZYNQ-7000 фирмы Xilinx Inc._
Мова програмування С_____
Мови опису апаратури VHDL, Verilog_____
САПР XILINX ISE, Vivado, Vitis_____
_____

Перелік питань, що потрібно опрацювати у роботі _____
Методи проектування цифрових фільтрів_____
Кіберфізичні системи на платформі SoC _____
Взаємодія процесорного ядра та ПЛІС у складі SoC_____
Методи автоматизованого проектування на платі ZebBoard з SoC ZYNQ-7000

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 19 слайдів _____
_____
_____
_____

6. Консультанти розділів роботи (проекту)

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

7. Дата видачі завдання 02.09.2024_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	02.09.2024-08.09.2024	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	09.09. 2024-15.09. 2024	
3	Аналіз методів проектування цифрових фільтрів	16.09.2024-29.09.2024	
4	Вибір налагоджувальної плати та інструментарію для реалізації проекту	30.09. 2024-20.10.2024	
5	Фізична реалізація проекту в налагоджувальній платі	20.10. 2024-20.11. 2024	
6	Макетування спроектованих фільтрів	20.11. 2024-10.12. 2024	
7	Оформлення пояснювальної записки	10.12. 2024-30.12. 2024	
8	Захист проекту	02.01. 2025-25.01. 2025	

Здобувач _____
(підпис)

Керівник роботи (проекту) _____ ___доц.Шкіль О.С_____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 60 сторінок, 23 рисунки, та 9 джерел за переліком посилань.

АЛГОРИТМИ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ, ЦИФРОВІ ФІЛЬТРИ, СИСТЕМИ НА КРИСТАЛІ, АВТОМАТИЗОВАНЕ ПРОЄКТУВАННЯ, МОВА ПРОГРАМУВАННЯ С.

Метою роботи є розробка методів автоматизованого проектування цифрових фільтрів низьких частот на технологічній платформі систем на кристалі. Проаналізовано різні типи цифрових фільтрів, в тому числі однорідні та гребінчасті. Приведено їх структури, порівняльну характеристики. Описані переваги та недоліки. Наведено їх особливості застосування.

В якості технологічної платформи обрано програмовану систему на кристалі FPGA сімейства ZYNQ-7000 фірми Xilinx Inc. В якості налагоджувальної плати використана недорога загальнодоступна плата ZedBoard, яка оснащена пристроєм XC7Z020 Zynq. Сформульована загальна послідовність етапів проектування системи на кристалі на платформі ZYNQ-7000. Практична реалізація виконана на базі стеку інструментальних засобів САПР Vivado/Vitis/Vitis HLS.

ABSTRACT

The explanatory note contains: 60 pages, 23 figures, 9 sources according to the list of links.

DIGITAL SIGNAL PROCESSING ALGORITHMS, DIGITAL FILTERS, SYSTEMS ON CHIP, COMPUTER-AIDED DESIGN, C PROGRAMMING LANGUAGE.

The goal of the work is to develop methods for automated design of digital low-pass filters on the system-on-chip technology platform. Various types of digital filters, including uniform and comb filters, are analyzed. Their structures and comparative characteristics are given. The advantages and disadvantages are described. Their application features are given.

The programmable system-on-chip FPGA of the ZYNQ-7000 family from Xilinx Inc. was chosen as the technology platform. An inexpensive publicly available ZedBoard board equipped with an XC7Z020 Zynq device was used as a development board. A general sequence of design stages for a system-on-chip on the ZYNQ-7000 platform is formulated. The practical implementation is based on the Vivado/Vitis/Vitis HLS CAD tool stack.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АЦП – аналогово-цифровий перетворювач;

АЧХ – амплітудно-частотна характеристика;

НІХ – *фільтр* з Нескінченною Імпульсною Характеристикою;

СІХ – *фільтр* зі Скінченною Імпульсною Характеристикою;

ПЛІС – програмована логічна інтегральна схема;

САПР – система автоматизованого проектування;

ФНЧ – фільтр нижніх частот;

ЦАП – цифро-аналоговий перетворювач;

ЦСП (DSP) – цифровий сигнальний процесор;

ЦОС – цифрова обробка сигналів;

ЦФ – цифровий фільтр;

AXI – Advanced eXtensible Interface (розширений розширюваний інтерфейс);

EMA – Exponential Moving Average (*фільтр* експоненціального рухомого середнього);

EWMA – Exponentially-Weighted Moving Average (*фільтр* експоненціально зваженого рухомого середнього);

FPGA – field-programmable gate array (різновид ПЛІС);

HDL – Hardware Description Language (мова опису апаратури);

HLS – High-Level Synthesis (вісокорівневий синтез);

IDE – Integrated Development Environment (интегрироване середовище розробки);

PS – processing system (процесорний блок SoC);

PL – programmable logic (блок програмовованої логіки SoC);

SMA – Simple Moving Average (простий рухомий середній *фільтр*);

SoC – System on Chip (система на кристалі).

ВСТУП

Проектування сучасних систем, що вбудовуються, засноване на застосуванні високотехнологічних систем автоматизованого проектування (САПР) цифрових пристроїв, що вимагає від розробників глибоких знань не тільки цифрової схемотехніки та архітектур обчислювальних систем, але й знання методів синтезу спеціалізованих пристроїв з мікропрограм ним та логічним управлінням, знання високорівневих мов проектування та методів контролепридатного проектування. Процес проектування сучасних вбудованих цифрових пристроїв і систем – це процес створення власних та використання стандартних цифрових компонентів інтелектуальної власності, які є не лише схемотехнічними описами, але по суті є повноцінними проектними документаціями з функціонального та параметричного моделювання, верифікації та виготовлення із застосуванням конкретних технологій.

Система на кристалі (System on Chip, SoC) – система, побудована на єдиному кристалі, в якій інтегруються такі елементи, як процесор (процесори, у тому числі спеціалізовані), пам'ять, ряд периферійних пристроїв, спеціалізовані обчислювальні блоки та їх з'єднання. Все вищезгадане складає оптимальний набір для деякого заздалегідь відомого функціонала, наприклад, цифрової обробки сигналів.

Обробка сигналів може здійснюватись за допомогою різних технічних засобів. В останнє десятиліття лідируюче положення займає цифрова обробка сигналів (ЦОС), яка в порівнянні з аналоговою має такі переваги: малу чутливість до параметрів навколишнього середовища, простоту перепрограмування та переносимість алгоритмів. Однією з поширених задач у ЦОС є фільтрація. Вигляд імпульсної характеристики цифрового фільтра (ЦФ) визначає їх поділ на ЦФ з скінченною імпульсною характеристикою (СІХ-фільтри) та ЦФ з нескінченною імпульсною характеристикою (НІХ-фільтри). Широке застосування ЦФ викликано тим, що властивості добре

досліджені. Використання особливостей архітектури SoC дозволяє проектувати компактні та швидкі ЦФ з використанням так званої розподіленої арифметики.

Таким чином, розробка методів проектування різних типів цифрових фільтрів на технологічній платформі SoC на різних стадіях циклу проектування є актуальною та технічно значущою задачею.

Об’єкт дослідження у роботі – процедури автоматизованого проектування систем цифрової обробки сигналів.

Предмет дослідження – моделі, методи та процедури проектування цифрових фільтрів на технологічній платформі SoC.

Метою роботи є розробка процедур автоматизованого проектування різних типів цифрових фільтрів низьких частот на базі технологічної платформи ZYNQ 7000 та проведення порівняльного аналізу різних варіантів їх програмно-апаратної реалізації.

Для досягнення поставленої мети мають бути вирішені такі задачі :

– розглянути математичні моделі та апаратні реалізації різних типів цифрових фільтрів;

– визначити можливі способи реалізації цифрових фільтрів низьких частот на технологічній платформі ZYNQ 7000;

– розробити процедури реалізації створених моделей цифрових фільтрів низьких частот на базі стеку інструментальних засобів САПР Vivado/Vitis/Vitis HLS з використанням мови програмування C;

– виконати апаратну реалізацію різних типів фільтрів низьких частот для SoC Zynq 7000 на платі ZedBoard та провести порівняльний аналіз їх апаратних витрат та швидкодії.

1 ОСНОВИ ЦИФРОВОЇ ФІЛЬТРАЦІЇ СИГНАЛІВ

1.1 Проектування пристроїв цифрової обробки сигналів

З розвитком цифрових технологій все більше уваги приділяється реалізації систем цифрової обробки сигналів як програмними так і апаратними засобами. Для реалізації алгоритмів цифрової обробки сигналів можна використовувати як програмні, так і апаратні ресурси. Апаратні ресурси включають в себе такі як цифрові сигнальні процесори, програмовані логічні інтегральні схеми (ПЛІС), системи на кристалі (SoC), мікроконтролери тощо.

Цифрові сигнальні процесори завдяки внутрішній архітектурі заточені для реалізацію різноманітних алгоритмів. Але вони мають обмеження по разрядності та швидкодії. Також можлива програмна реалізація цифрового фільтра у мікроконтролерах з використанням високорівневих мов.

Для реалізації цифрових алгоритмів обробки сигналів можна використовувати програмовані вентильні матриці, які мають відповідні блоки в своїй архітектурі. ПЛІС, в свою чергу, набувають більш широкого поширення для реалізації швидкодіючої цифрової обробки сигналів завдяки паралельній архітектурі і апарату, що пере конфігурується. Використання ПЛІС для реалізації пристроїв цифрової обробки сигналів є важливим інструментом, що дозволяє досягти високої швидкодії, гнучкості та ефективності в розробці та впровадженні сучасних систем обробки сигналів.. На відміну від цифрових сигнальних процесорів з закріпленою архітектурою ПЛІС мають велику обчислювальну продуктивність і дозволяють розробляти налаштовуються структури для ідеальної реалізації алгоритмів ЦОС а саме:

– ПЛІС можуть бути перепрограмовані для виконання різних функцій, що дозволяє легко адаптувати систему до змінних вимог або виправляти помилки без необхідності фізичної заміни апаратури;

– використання ПЛІС може забезпечити високу швидкодію виконання завдань, що особливо важливо для реального часу і обробки сигналів;

– ПЛІС можуть бути програмовані для оптимізації використання доступних ресурсів, таких як логічні елементи, блоки пам'яті та інші ресурси на чіпі, що дозволяє зменшити займану площу і споживану енергію;

– ПЛІС мають компоненти, такі як логічні вентилі, блоки пам'яті, множинники, а також спеціалізовані периферійні інтерфейси, що спрощує розробку і зменшує кількість потрібних компонентів для задач цифрової обробки;

– ПЛІС дозволяють швидко створювати прототипи систем, що дозволяє проводити експерименти та виконувати випробування до виготовлення фінальної версії апаратури.

Ці переваги роблять ПЛІС популярним вибором для реалізації ЦОС в різних галузях, включаючи автомобільну промисловість, медичні пристрої, телекомунікаційні системи та багато інших.

1.2 Цифрові фільтри

Цифрова фільтрація є однією з ключових частин цифрової обробки сигналів. Вона включає застосування математичних алгоритмів і операцій до цифрових сигналів для зміни їх властивостей.

Цифрові фільтри використовуються для різних цілей, таких як видалення шуму, виділення сигналів, зміна частоти, покращення якості зображень та звуку, та багато чого іншого. Це потужний інструмент, який знаходить застосування у різних галузях, включаючи обробку звуку, зображень, відео, сигналів радіозв'язку та інфокомунікацій, медичної діагностики та інших.

При реалізації різноманітних цифрових пристроїв для обробки сигналів особлива увага приділяється цифровій фільтрації, яка в середньому займає до половини обсягу обчислень.

Класифікацію цифрових фільтрів можна проводити у великій кількості класифікаційних ознак. Цифрові фільтри можна класифікувати за такими параметрами як структура, методологія проектування, характеристики передавальної функції та інші. Ось деякі основні класифікації цифрових фільтрів:

За структурою цифрові фільтри можна поділити на:

– нерекурсивні фільтри (СІХ фільтри);

– рекурсивні фільтри (НІХ фільтри).

Рекурсивні фільтри відрізняються від нерекурсивних (конволюційних) фільтрів деякими важливими характеристиками. Рекурсивні фільтри на відміну від нерекурсивних мають зворотний зв'язок, що дозволяє використовувати попередні вихідні значення як вхідні для подальшого обчислення. У нерекурсивних фільтрах такого зворотного зв'язку немає, і кожне вихідне значення обчислюється безпосередньо з вхідних даних.

Рекурсивні фільтри можуть мати затримку у вихідному сигналі через використання попередніх вихідних значень для обчислення поточного. У нерекурсивних фільтрах затримки не відбуваються, і вихідне значення обчислюється миттєво на основі вхідних даних.

Рекурсивні фільтри можуть бути менш складними в реалізації, оскільки вони можуть мати менше коефіцієнтів і обчислень, що може зменшити обсяг обчислювальних витрат у порівнянні з нерекурсивними фільтрами.

Рекурсивні фільтри можуть бути чутливі до початкових умов через їхню залежність від попередніх вихідних значень. У нерекурсивних фільтрах такої чутливості не спостерігається.

В таких системах як системи розпізнавання мови або вимірювальні системи важливим є забезпечення лінійності фазових характеристик. Ця вимога виконується при обробці сигналів цифровими фільтрами з кінцевою імпульсною характеристикою (СІХ-фільтрами або нерекурсивними фільтрами). Крім лінійності фазових характеристик СІХ-фільтри є принципово стійкими системами, і процес обчислення їх коефіцієнтів не

викликає труднощів. Однак при підвищених вимогах до вибірковості таких фільтрів – круті спади АЧХ та характеристики загасання – необхідні значні апаратні та обчислювальні витрати, що накладає обмеження на практичну реалізацію СІХ-фільтрів на основі традиційних компонентів цифрової електроніки, зокрема мікропроцесорів та сигнальних процесорів. Ці обмеження можна виключити під час реалізації СІХ-фільтрів на основі сучасної компонентної бази, а саме ПЛІС.

На відміну від СІХ-фільтрів фільтри з нескінченною імпульсною характеристикою (НІХ-фільтри, або рекурсивні фільтри) не володіють строго лінійною фазовою характеристикою, за винятком окремого випадку, коли всі полюси знаходяться на одиничному колі. Однак вони більш ефективні, ніж СІХ-фільтри, і мають менший порядок за однакових параметрів. Особливістю НІХ-фільтрів є можливість самозбудження за недостатньої розрядності коефіцієнтів фільтра або неправильному розрахунку передавальної характеристики. Щоб зменшити вплив обмеженої розрядності коефіцієнтів, НІХ-фільтри синтезують послідовним каскадним з'єднанням ланок другого порядку.

Рекурсивні фільтри відрізняються від СІХ-фільтрів наявністю зворотного зв'язку. Вихідні відліки рекурсивних фільтрів залежать не тільки від попередніх та поточних вхідних відліків (як у СІХ-фільтрі), а й від попередніх вихідних відліків.

За характеристиками передавальної функції цифрові фільтри поділяються на низькочастотні, високочастотні, полосно-пропускні фільтри та полосно-загортаючі фільтри.

Також розділяють типи вибірковості фільтрів. Це елементи обробки сигналів, які використовуються для відокремлення чи підсилення певних частот або діапазонів частот у вхідних сигналах. Основні типи фільтрів вибірковості включають наступні.

1. Фільтри низьких частот (Low Pass Filters) пропускають сигнали з низькими частотами і блокують сигнали з високими частотами.

Найпростішим прикладом такого фільтра є RC фільтр, який складається з резистора і конденсатора. Резистор визначає обмеження струму, а конденсатор визначає частоту зрізу.

Фільтри низьких частот використовуються для видалення високочастотного шуму з аналогових сигналів, а також для згладжування сигналів.

2. Фільтри високих частот (High Pass Filters) пропускають сигнали з високими частотами і блокують сигнали з низькими частотами.

Активні високочастотні фільтри можуть бути побудовані на основі операційних підсилювачів або транзисторів. Використовуються для вилучення низькочастотного шуму та підсилення сигналів, що містять швидкі зміни, наприклад, у високочастотних даних.

3. Фільтри смугової пропускання (Band Pass Filters) пропускають сигнали лише в певному діапазоні частот, блокуючи сигнали з інших частотних діапазонів. Вони застосовуються для виділення певних сигналів зі змішаних сигналів або для виділення конкретних смуг частот.

4. Фільтри смугового затримання (Band Stop Filters або Notch Filters) блокують сигнали в певному діапазоні частот, пропускаючи сигнали з інших частотних діапазонів. Використовуються для приглушення або видалення певних шумів, інтерференцій або небажаних сигналів зі спектра.

5. Фільтри фазового затримання (Phase Shift Filters) зміщують фазу вхідного сигналу в залежності від його частоти. Застосовуються для корекції фазових зміщень у сигналах, що можуть виникати під час їх обробки або передачі. Ці типи фільтрів можуть бути комбіновані та налаштовані для досягнення бажаних характеристик фільтрації залежно від конкретних потреб і умов застосування.

Амплітудно-частотні характеристики (АЧХ) різних типів типи фільтрів представлені на рис. 1.1.

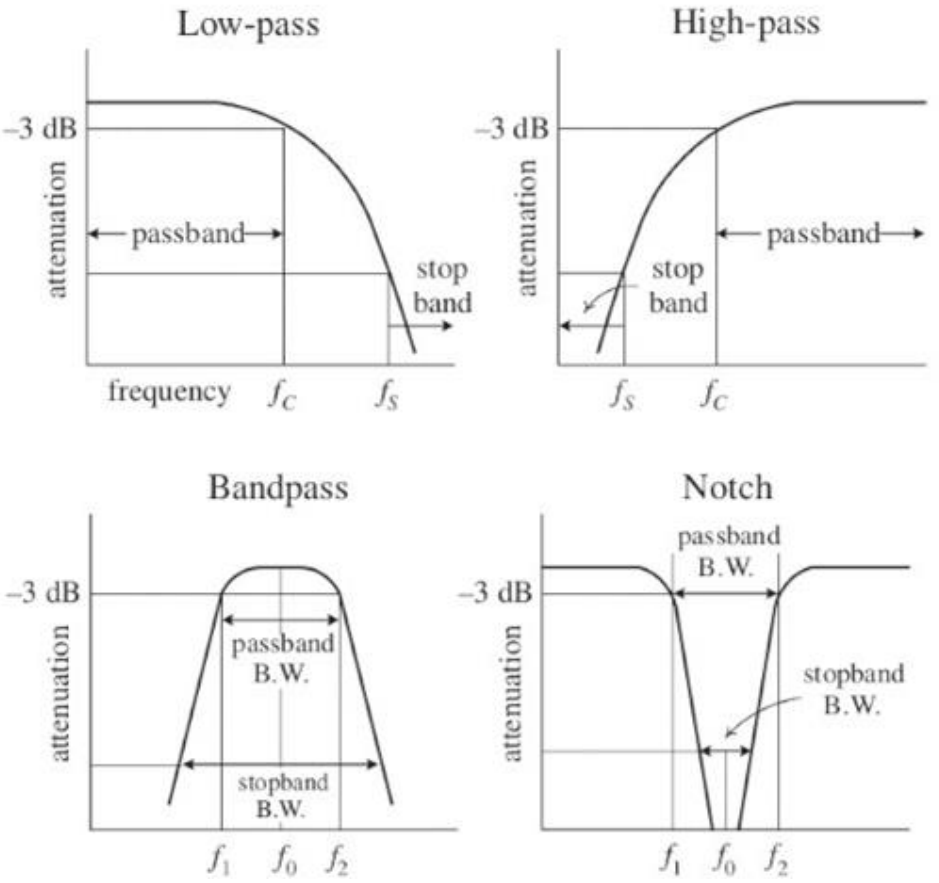


Рисунок 1.1 – Типи фільтрів

За типом структури цифрові фільтри можна поділити на фільтри із постійною структурою та фільтри із змінною структурою. Ці в свою чергу поділяються на адаптивні та неадаптивні фільтри.

За типом обробки сигналу можна виділити такі типи фільтрів:

– фільтри для обробки аудіо сигналів;

– фільтри для обробки відеосигналів;

– фільтри для обробки сигналів зображень;

– фільтри для обробки сигналів зв'язку;

– фільтри для обробки медичних сигналів.

Ця класифікація допомагає розуміти різноманітність і застосування цифрових фільтрів у різних областях сигнальної обробки та дозволяє вибрати відповідні рішення при побудуванні систем обробки сигналів.

1.3 Проектування цифрових фільтрів

Проектування цифрових фільтрів є важливою складовою сучасної цифрової сигнальної обробки. Проектування цифрових фільтрів вимагає ретельного аналізу властивостей сигналу та визначення специфікацій фільтрації.

Перший крок у проектуванні цифрових фільтрів – це визначення вимог щодо фільтрації. Це включає в себе частотний діапазон, який потрібно обробляти, а також тип фільтра (наприклад, низькочастотний, високочастотний, смугового пропускання тощо). Крім того, важливо враховувати допустимі відхилення від частотної характеристики та амплітуди.

Другий крок – вибір оптимальної структури фільтра. Це може бути реалізовано за допомогою різних методів, таких як прямий перетворювач Фур'є (FFT), кореляція, конволюція та інші. Кожен метод має свої переваги та обмеження, які варто враховувати при виборі.

Після вибору структури фільтра проводиться синтез коефіцієнтів фільтра. Цей процес включає в себе розрахунок коефіцієнтів фільтра на основі визначених вимог та структури фільтра. Для цього можна використовувати різні методи оптимізації, такі як метод мінімальних квадратів, метод вагових коефіцієнтів та інші.

Після синтезу коефіцієнтів фільтра проводиться аналіз та налагодження його характеристик. Це може включати в себе використання імпульсної відповіді, амплітудно-частотних та фазо-частотних характеристик для перевірки відповідності вимогам специфікацій.

Коли фільтр проектується та налагоджується, він може бути реалізований у вигляді програмного коду або апаратного обладнання в залежності від конкретних потреб застосування. Після цього фільтр може бути імплементований у відповідній системі та протестований для перевірки його ефективності.

1.3.1 Нерекурсивні фільтри

СІХ фільтри (англ. FIR – "finite impulse response") або фільтри з скінченною імпульсною характеристикою, є одним із ключових інструментів у обробці сигналів та керуванні системами реального часу. Ці фільтри є потужним математичним інструментом, який дозволяє ефективно фільтрувати шуми та невизначеності в даних, зберігаючи при цьому важливі сигнальні характеристики.

Імпульсна характеристика СІХ фільтра обмежена у часі, тобто має кількість коефіцієнтів, що можна перелічити. Починаючи з певного моменту часу, вона стає рівною нулю. Якщо на вхід СІХ-фільтра подати одиничний імпульс, на виході фільтра буде кінцева кількість відліків. Як правило, ФЧХ фільтра з кінцевою імпульсною характеристикою - лінійна

У випадку, СІХ-фільтри реалізуються без зворотних зв'язків, тобто вони нерекурсивні. Однак за допомогою математичних перетворень можна привести фільтр до рекурсивної форми.

Різностне рівняння СІХ-фільтра має наступний вигляд:

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k) \tag{1.1}$$

де y(n) – вихідний дискретний сигнал (сума зважених вхідних імпульсів),
x(n) – вхідний дискретний сигнал (послідовність відліків),
h(n) – коефіцієнти імпульсної характеристики фільтра,
N – довжина (порядок) фільтра.

Передатна характеристика СІХ-фільтра має наступний вигляд:

$$H(z) = \sum_{k=0}^{N-1} h(k)z^{-k} \tag{1.2}$$

Структурну схему цифрового фільтра можна подати у прямій та канонічній формі (рис.1.2). Цифрові фільтри можуть бути реалізовані за допомогою трьох цифрових елементів: помножувач, суматор і блок затримки. Очевидно, що помножувачі потрібні для множення відліків на коефіцієнти, а суматор поєднує отримані результати.

Кількість ліній затримки завжди на одиницю менше, ніж кількість коефіцієнтів чи порядок фільтра. Імпульсна характеристика СІХ фільтра збігається з набором коефіцієнтів цього фільтра.

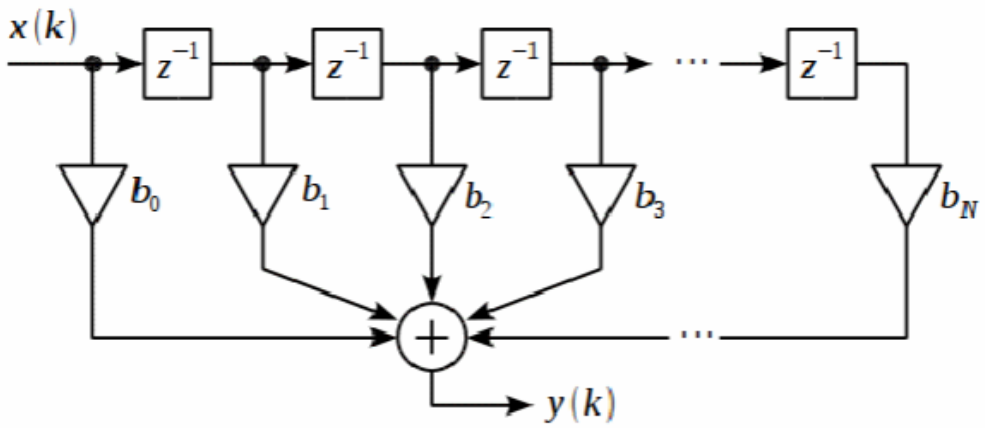


Рисунок 1.2 – Структурна схема нерекурсивного СІХ-фільтра

1.3.2 Однорідні фільтри

Одним з класів СІХ фільтрів є однородні фільтри. Однорідні фільтри, також відомі як фільтри середнього рухомого, стали широко поширеними в цифровій обробці сигналів (ЦОС) завдяки своїй простоті в реалізації. Вони також є оптимальним вибором для вирішення класичної задачі цифрової обробки – придушення адитивного шуму за збереження швидкого наростання перехідної характеристики. Простіше кажучи, однорідні фільтри перевершують інші цифрові фільтри під час обробки часових сигналів. Однак вони виявляють недостатню ефективність в обробці інформації в частотній області, оскільки не здатні розрізняти сигнали різних частот. Існують різні

модифікації однорідних фільтрів, такі як фільтр Гаусса, фільтр Блекмана та фільтр з багаторазовою обробкою, які покращують їхню здатність до вибіркової фільтрації за рахунок збільшення часу обробки.

Однорідні фільтри, які також називаються фільтрами рухомого середнього через свою основу, являють собою усереднення кількох відліків вхідного сигналу, що відбивається у відповідному різницевому рівнянні. Характеристика фільтра має наступний вигляд:

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i + j] \tag{1.3}$$

де x[ ] – вхідний сигнал; y[ ] – вихідний сигнал; M – число відліків сигналу. Цей вираз можна представити в наступному вигляді, наприклад, для фільтра 5-го порядку:

$$y[i] = (x[j-2] + x[j-1] + x[j] + x[j+1] + x[j+2])/5. \tag{1.4}$$

Однорідні фільтри, відомі також як фільтри з простою імпульсною характеристикою, мають просту структуру (рис.1.3). Наприклад, однорідний нерекурсивний фільтр п'ятого порядку має імпульсну характеристику, яка представлена наступним чином: {..., 0, 0, 1/5, 1/5, 1/5, 1/5, 1/5, 0, 0, ...}. Це означає, що в основі однорідних нерекурсивних фільтрів лежить алгоритм згортки вхідного сигналу з прямокутним імпульсом одиничної площі.

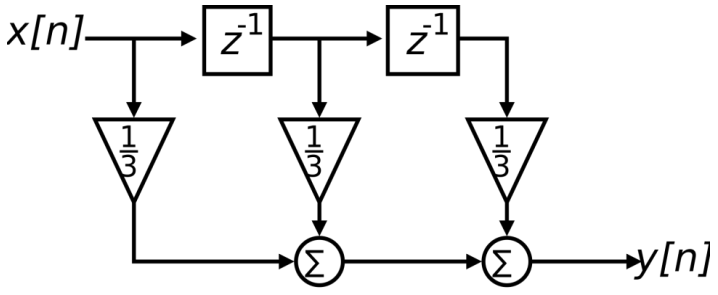


Рисунок 1.3 – Структура однорідного нерекурсивного фільтру

Однорідні фільтри не лише знаходять широке застосування в

різноманітних практичних сферах, але і є оптимальними при розв'язанні таких поширених задач, як приглушення білого шуму з одночасним збереженням бажаної швидкості наростання перехідної характеристики.

Після проходження через фільтр суттєво зменшується рівень потужності шуму, що є позитивним результатом, проте разом із цим спостерігається зниження крутості фронтів імпульсу, що є негативним наслідком.

Серед усіх лінійних фільтрів однорідні фільтри відзначаються особливо сильним придушенням шуму при збереженні заданої крутості фронтів імпульсу. Коефіцієнт придушення шуму пропорційний квадратному кореню з кількості відліків, які беруть участь у середньому. Так, наприклад, фільтр 100-го порядку може зменшити шум до 10 разів.

Чим більше відліків вхідного сигналу використовуються для усереднення, тим менше виявляється потужність шуму у вихідному сигналі, проте, водночас, стають меншими крутості фронтів імпульсу. Однорідний фільтр є оптимальним варіантом для досягнення компромісу у цій ситуації, забезпечуючи максимальне приглушення шуму при збереженні заданої крутості фронтів імпульсу.

Недоліки неоднорідних фільтрів включають наступні моменти.

Перше, це складність реалізації: В порівнянні з однорідними фільтрами, неоднорідні фільтри можуть вимагати більш складних обчислювальних операцій для їхньої реалізації та відповідно апаратурних ресурсів, або додаткового часу при програмній реалізації.

По-друге, можлива втрата інформації при великій кількості коефіцієнтів. Імпульсна характеристика неоднорідних фільтрів може призводити до втрати деякої інформації в сигналі через різницю відповідей на різні частоти.

Також, неоднорідні фільтри можуть мати більшу затримку, особливо у порівнянні з простішими однорідними фільтрами, що може бути критичним у деяких додатках реального часу.

Зазвичай неоднорідні фільтри можуть вимагати більшого обсягу пам'яті або обчислювальних ресурсів для їхнього виконання через складнішу структуру імпульсних характеристик.

Переваги неоднорідних фільтрів включають:

– гнучкість у налаштуванні, так як неоднорідні фільтри можуть бути налаштовані для оптимальної обробки сигналів з різною частотною характеристикою; все це робить їх більш гнучкими у порівнянні з однорідними фільтрами;

– вони мають кращу адаптацію до сигналів зі змінною частотною характеристикою, а саме неоднорідні фільтри можуть краще адаптуватися до сигналів зі змінною частотною характеристикою, так як їхня імпульсна характеристика може бути налаштована для відповідного сигналу;

– вони мають кращу роздільна здатність, а саме через можливість налаштування імпульсної характеристики, неоднорідні фільтри можуть забезпечити кращу роздільну здатність у порівнянні з однорідними фільтрами, особливо при обробці сигналів з широким діапазоном частот.

– вони ефективно приглушують шуму, а саме неоднорідні фільтри можуть забезпечувати ефективне приглушення шуму при одночасному збереженні корисного сигналу, що робить їх корисними для застосувань, де важливо відокремлювати сигнал від шуму.

Однорідні фільтри, як найпростіші в реалізації, завжди першими вибором для застосування. Однорідні фільтри не лише мають широке застосування в різних практичних областях, але також є оптимальними при вирішенні типової задачі – приглушення білого шуму з одночасним збереженням певної швидкості наростання перехідної характеристики.

Розглянемо операцію однорідного фільтра на вхід якого подається сигнал, який є прямокутним імпульсом та він засмучений адитивним шумом. Після фільтрації однорідним фільтром потужність шуму помітно зменшується. Це є позитивний ефект, проте разом із цим спостерігається зниження крутості фронтів імпульсу. А це є негативний ефект. Серед усіх

лінійних фільтрів однорідні фільтри відзначаються найбільш сильним пригніченням шуму при заданій крутості фронтів імпульсу. Коефіцієнт пригнічення шуму дорівнює квадратному кореню з кількості відліків, що беруть участь у усередненні. Так, наприклад, фільтр 100-го порядку може зменшити шум в 10 разів.

Щоб зрозуміти, чому однорідний фільтр найефективніше пригнічує шум, що присутній у вхідному сигналі, спробуємо розрахувати фільтр, задавши обмеження на швидкість наростання перехідного процесу. Припустимо, що тривалість перехідного процесу складає 13 відліків. Отже, фільтр має 13 вагових коефіцієнтів. Задача оптимізації полягає в тому, як правильно вибрати значення цих 13 вагових коефіцієнтів, щоб потужність шуму на виході фільтра була мінімальною.

Оскільки шумова перешкода – це випадковий процес, жоден з відліків не має будь-яких унікальних властивостей в порівнянні з іншими, тобто кожен відлік підданий впливу шуму у тій же мірі, що й його навколишні відліки. Тому не має значення. Віддавати перевагу будь-якому конкретному відліку вхідного сигналу, збільшуючи для цього один з коефіцієнтів фільтра, не має сенсу. Мінімальний шум досягається, коли всі вагові коефіцієнти вирівняні, що означає використання однорідного фільтра.

Типи однорідних фільтрів включають:

– простий рухомий середній фільтр (SMA), який обчислює середнє значення з деякого вікна даних, яке переміщується по вхідному сигналу;

– експоненційно зважений рухомий середній фільтр (EWMA), який обчислює середньозважене значення, де ваги експоненційно зменшуються з часом;

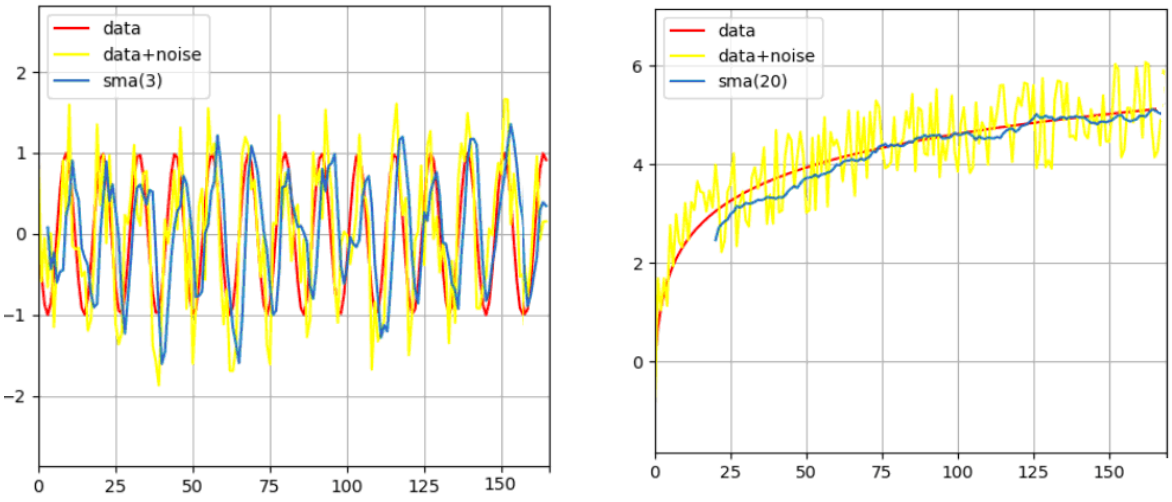
– фільтр Хольта, це метод, який розширює EWMA для моделювання трендів та сезонності у данних;

– цифровий фільтр Чебишева, він використовує коефіцієнти, що базуються на коефіцієнтах Чебишева, для досягнення заданих параметрів у частотній області;

– фільтр Кальмана, який можна назвати алгоритмом статистичної оцінки, який комбінує інформацію з вимірювань та передбачення моделі для отримання оптимальної оцінки стану системи.

Це лише не повний перелік однорідних фільтрів, які використовуються у різних областях, таких як обробка сигналів, аналіз часових рядів та автоматизація.

Простий рухомий середній фільтр (SMA) – це один з найпростіших і найбільш поширених типів фільтрів у сигнальній обробці. Він використовується для згладжування шуму та вирівнювання коливань у часових рядах даних (рис. 1.4).



а) б)

Рисунок 1.4 – Фільтрація сигналу за допомогою SMA фільтра

Формула для розрахунку SMA:

$$r_i = \frac{1}{s} \cdot \sum_{j=1}^{s} y_{(i-j)} \; ; \; i > s \tag{1.5}$$

де r_i – поточні значення фільтра;

s – кількість відліків, які беруться для обчислення;

y_i – відліки сигналу.

Принцип роботи простого рухомого середнього полягає в обчисленні середнього значення за певний часовий період, що переміщується по вхідному сигналу. Наприклад, якщо ми маємо часовий ряд з N значень, для кожного нового відліку береться середнє арифметичне значень з попередніх N відліків, щоб отримати нове вирівняне значення.

На рисунку 1.4 наведено фільтрацію сигналів при N = 3 та N = 20.

Цей фільтр є простим у реалізації та досить ефективним для згладжування високочастотного шуму та виділення трендів у даних. Однак він може бути не так ефективним у випадках, коли вхідний сигнал містить різні аномалії або великі зміни.

Переваги простого рухомого середнього фільтра (SMA):

– простота реалізації, що робить його доступним для використання навіть для початківців у сигнальній обробці;

– згладжування шуму, що допомагає виділити тренди та основні закономірності в даних.

Недоліки простого рухомого середнього фільтра (SMA):

– SMA має затримку, яка залежить від розміру вікна фільтра, а це може стати проблемою у додатках реального часу, де важлива низька затримка обробки сигналу;

– при великих розмірах вікна фільтра SMA може призводити до втрати деталей в даних або розмиття різких змін;

– неефективність для аномальних даних, тобто SMA не ефективний у випадках, коли дані містять аномалії або великі зміни, оскільки він не реагує на них адекватно.

Фільтр зваженого рухомого середнього (WMA) – це метод згладжування даних, який використовує зважену суму спостережень для обчислення середнього значення. На відміну від простого рухомого середнього, в якому всі точки даних мають однакову вагу, зважене рухоме середнє дає більшу вагу останнім спостереженням.

Принцип роботи WMA полягає в наступному:

– задається період згладжування, тобто кількість послідовних спостережень, що використовуються для розрахунку середнього;

– кожне спостереження множиться на відповідну йому вагу; Зазвичай ваги зменшуються при віддаленні від останнього спостереження;

– зважені значення підсумовуються і поділяються на суму ваги для отримання підсумкового згладженого значення.

Формула для розрахунку WMA має віигляд:

$$r_i = \frac{2}{s \cdot (s+1)} \cdot \sum_{j=1}^{s} j \cdot y_{(i-j)} \; ; \; i > s \tag{1.6}$$

Переваги використання WMA:

– фільтр зваженого рухомого середнього гнучкіший і адаптивніший до змін ринку метод порівняно з простим рухомим середнім;

– фільтр WMA дозволяє давати більшу вагу останнім даним, що може бути корисним для прогнозування змін у даних.

Однак використання WMA вимагає визначення відповідних ваг для кожного спостереження, що може вимагати деяких розрахунків або досвідченого підбору, а це викликає додаткові апаратні та програмні витрати.

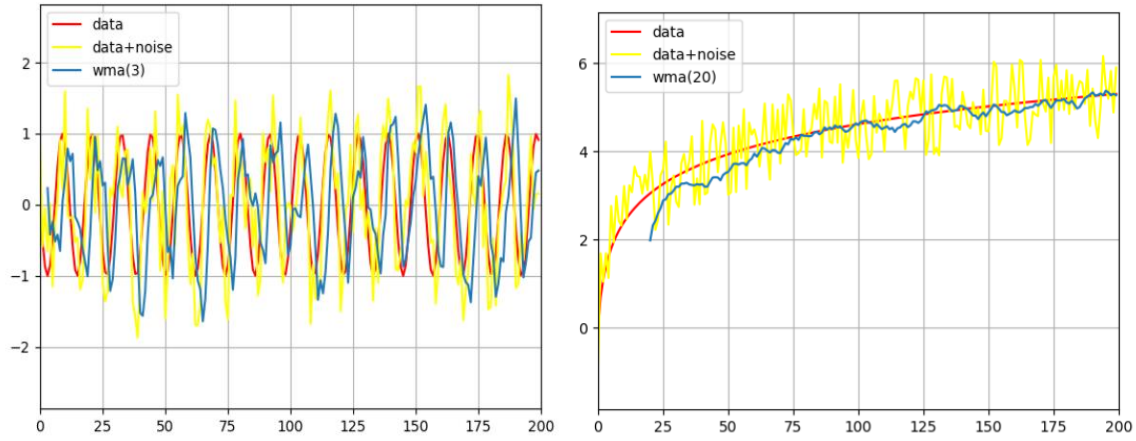
Експоненційно зважений рухомий середній фільтр (EWMA) є типом фільтра, який використовується для згладжування шуму та аналізу трендів у часових рядах даних. Він є розширенням простого рухомого середнього фільтра, де кожне значення у вихідному ряді має вагу, яка зменшується експоненційно з часом.

Принцип роботи EWMA полягає в обчисленні поточного значення як зваженого середнього між поточним значенням сигналу і попереднім значенням фільтра. Вага для кожного нового відліку вираховується експоненційно, що означає, що нові значення мають більшу вагу, а старі значення мають меншу вагу у вихідному ряді.

Одна з основних переваг EWMA полягає в тому, що він дозволяє швидше реагувати на нові тенденції та зміни у вхідних даних порівняно з

простим рухомим середнім фільтром. Крім того, він менше схильний до затримок порівняно з іншими видами фільтрів.

Проте, треба враховувати, що EWMA може бути більш чутливим до випадкових аномалій у вхідних даних через експоненційний характер ваг. Крім того, він має параметр згладжування, який потрібно правильно налаштувати для оптимальної роботи фільтра в конкретному випадку. На рисунку 1.5 наведено фільтрацію сигналів при N = 3 та N = 20.



а) б)

Рисунок 1.5 – Фільтрація сигналу за допомогою EWMA фільтра

Переваги експоненційно зваженого рухомого середнього фільтра (EWMA) наступні:

– швидка реакція на зміни, а саме EWMA здатний швидко реагувати на зміни у вхідних даних, оскільки нові значення мають більшу вагу;

– даний фільтр має меншу затримку порівняно з іншими методами згладжування, що робить його більш придатним для використання у додатках реального часу;

– відсутність необхідності у фіксованому вікні, а саме EWMA не потребує фіксованого розміру вікна, оскільки він враховує всі попередні значення з відповідними вагами.

Недоліки експоненційно зваженого рухомого середнього фільтра:

– EWMA може бути чутливим до випадкових аномалій у вхідних

даних, оскільки вони можуть спричинити великі зміни у вагах;

– важливість налаштування параметрів, а саме необхідно правильно налаштувати параметр згладжування для досягнення оптимальних результатів, що може бути складним завданням без достатнього розуміння характеристик вхідних даних;

– при використанні EWMA може відбуватися зменшення деталізації вихідних даних, оскільки він згладжує коливання та шум.

1.3.3 Гребінчасті фільтри

Гребінчастий фільтр – це тип фільтра, який має характерну частотну характеристику, що нагадує ряд зубців, звідси походить його назва "гребінчастий". Він застосовує затримку до оригінального сигналу і комбінує його з самим собою, створюючи специфічну частотну характеристику з піками і провалами, що нагадують зубці гребеня. Цей ефект досягається за рахунок інтерференції між оригінальним сигналом та його затриманими копіями (рис. 1.6, 1.7).

Гребінчасті фільтри (comb filters) широко використовуються в різних областях, наприклад, в аудіо обробці для створення ефекту реверберації, а також для придушення або посилення певних частотних компонентів сигналу. Також вони використовуються для придушення або підсилення певних частотних компонентів у сигналі.

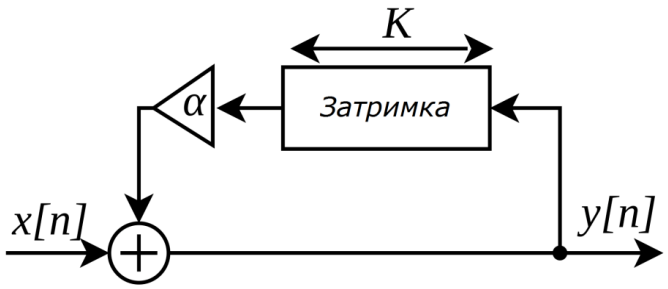


Рисунок 1.6 – Схема гребінчастого фільтра із зворотним зв'язком (з нескінченною імпульсною характеристикою)

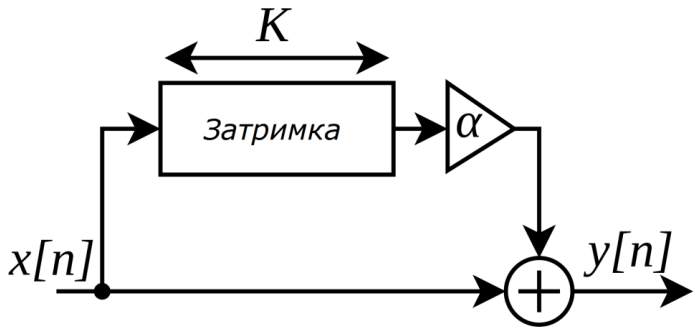


Рисунок 1.7 – Схема гребінчастого фільтра без зворотного зв'язку (нескінченною імпульсною характеристикою)

Головною особливістю гребінчастого фільтра є наявність піків (провалів) у його частотній характеристиці з рівним інтервалом між ними, схожими на зубці гребеня. Провали можуть бути створені шляхом поєднання оригінального сигналу з його затриманими копіями, що призводить до інтерференції між ними.

Одним із основних застосувань гребінчастих фільтрів є ефект звучання або звукового простору в аудіо обробці. Затримані копії сигналу створюють реверберацію та відображення, що надає звучанню обсягу та глибини. Гребінчасті фільтри також використовуються для придушення або посилення певних частотних компонентів у сигналі, таких як зворотні відлуння, шуми та резонанси.

Існує кілька типів гребінчастих фільтрів, включаючи фазові та частотні. Фазові гребінчасті фільтри створюють свої зубці шляхом додавання затримки до оригінального сигналу, тоді як частотні гребінчасті фільтри використовують комбінацію зубців та провалів у їх частотній характеристиці.

Гребінчасті фільтри можуть бути реалізовані як фільтри з кінцевою імпульсною характеристикою, так і фільтри з нескінченною імпульсною характеристикою.

Принцип роботи гребінчастого фільтра заснований на комбінуванні оригінального сигналу з його затриманими копіями з метою створення

специфічної частотної характеристики з піками та провалами, що нагадують зубці гребеня. Гребінчастий фільтр використовує затримку до оригінального сигналу. Ця затримка може бути фіксованою або змінною, вона визначає інтервал між зубцями гребінчастого фільтра.

Затриманий сигнал комбінується з оригінальним сигналом шляхом додавання або множення, залежно від конкретної реалізації фільтра. Зазвичай це робиться з використанням лінійних операцій, таких як додавання або множення. Комбінація оригінального та затриманого сигналу створює специфічну частотну характеристику фільтра з піками та провалами. Піки зазвичай виникають там, де сигнали збігаються або посилюють один одного, а провали – там, де вони зменшуються або віднімаються.

Отримана частотна характеристика може бути використана різних цілей, таких як створення ефекту реверберації, придушення певних частотних компонентів або посилення сигналу в певних частотних діапазонах. Формула гребінчастого фільтра, що створює ефект реверберації в аудіо обробці має вигляд:

$$y(n)=x(n)+ax(n-\tau)y(n)=x(n)+ax(n-\tau) \quad (1.7)$$

де $x(n)$ – вхідний сигнал;

$y(n)$ – вихідний сигнал гребінчастого фільтра;

a – коефіцієнт зворотного зв'язку, який визначає амплітуду затриманого сигналу та, отже, ефект реверберації, зазвичай він вибирається в діапазоні від 0 до 1 де 0 – відсутність реверберації, а 1 – максимальна реверберація;

τ – затримка сигналу, яка визначає інтервал між зубцями гребінчастого фільтра. Ця затримка зазвичай виражається у кількості відліків сигналу і може бути фіксованою чи змінною залежно від необхідного ефекту.

Реалізація цього фільтра додає до оригінального сигналу його затриману копію із заданим коефіцієнтом зворотного зв'язку. Це створює ефект реверберації, який імітує відбиття звуку від стін та поверхонь у

приміщенні.

Таким чином, принцип роботи гребінчастого фільтра полягає в комбінуванні оригінального сигналу з його затриманими копіями для створення специфічної частотної характеристики з піками та провалами, що робить його корисним інструментом у галузі обробки сигналів, аудіо та звукової інженерії.

Головна перевага гребінчастих фільтрів – це здатність створювати ефект реверберації, який надає звучанню об'єму та просторової глибини. Це особливо корисно в аудіо обробці та музичному виробництві для додавання натуральності звуку. Також до переваг гребінчастих фільтрів слід віднести те, що вони можуть бути відносно прості в реалізації як в аналоговій, так і цифровій формі. Їх структура зазвичай проста, що дозволяє легко впроваджувати їх у різні програми.

Параметри гребінчастих фільтрів, такі як коефіцієнт зворотного зв'язку та затримка, зазвичай можуть бути легко налаштовуються, що дозволяє користувачеві контролювати інтенсивність та характеристики реверберації. Однак при неправильному налаштуванні параметрів гребінчастого фільтра або зайвого використання реверберації можуть виникати артефакти та кольорування звуку, що може спотворювати звук або призвести до небажаних ефектів.

Використання гребінчастих фільтрів може призвести до незначної затримки сигналу через додану затримку затриманого сигналу. Це може бути критично в деяких програмах, таких як обробка звуку в реальному часі.

Це накладає певні обмежені можливості гребінчастих фільтрів. Так як вони призначені в основному для створення ефекту реверберації і їх можливості можуть бути обмежені для інших видів обробки сигналу, таких як шумопридушення або еквалізація.

Незважаючи на деякі недоліки, гребінчасті фільтри залишаються важливим інструментом в галузі аудіо обробки та музичного виробництва завдяки їх здатності створювати природні та привабливі звукові ефекти.

Вони також використовуються для ослаблення або посилення певних частотних складових сигналу. Це важливо, тому що дозволяє контролювати спектральний склад сигналу відповідно до конкретних потреб та вимог програми. Придушення або посилення певних частотних компонентів може бути необхідним для покращення якості сигналу, усунення небажаних шумів або артефактів, а також для створення певних звукових ефектів або емоційного забарвлення в аудіо сигналі. Це забезпечує більш точне та ефективне управління звуковим матеріалом, що є важливим аспектом у галузі аудіо обробки, звукової інженерії.

Виходячи з огляду робіт та існуючих задач у галузі реалізації цифрових фільтрів разом з зростаючими обчислювальними потужностями SoC є доцільним визначення оптимального розподілу обчислювальних витрат алгоритму фільтрації між PL та PS частинами SoC.

Постановка задачі – на основі проведеного аналізу існуючих досліджень актуальним є розгляд реалізація на платформі SoC гребінчастого фільтра та фільтра нижніх частот з кінцевою імпульсною характеристикою з метою аналізу витрат апаратних і програмних ресурсів та отриманої швидкодії кожного з блоків, що дозволить визначить доцільність розміщення складових частин обчислювального алгоритму фільтрації у SoC на PS або PL частині для максимально компромісного розподілу обчислень у SoC. Для реалізації використовується SoC на базі Xilinx ZYNQ 7020 з платою налагодження ZedBoard.

2 ТЕХНОЛОГІЧНА ПЛАТФОРМА РЕАЛІЗАЦІЇ SoC

2.1 Процесорні платформи сімейства Zynq-7000 AP SoC

Досягнення сучасної напівпровідникової промисловості дають змогу виготовляти інтегральні схеми цифрових пристроїв із надвеликим ступенем інтеграції. Виробники ПЛІС типу FPGA збільшують не тільки число і щільність компонування логічних блоків, але й здійснюють розміщення на програмованому кристалі таких функціональних блоків, як мікропроцесорні ядра, вбудовані пристрої, блоки арифметичних прискорювачів, інтерфейсні контролери стандартного обладнання тощо, перетворюючи цим ПЛІС в програмовані системи на кристалі.

Одним із прикладів програмованої системи на кристалі є FPGA сімейства ZYNQ-7000 фірми Xilinx Inc. Для побудови SoC різного призначення платформа Zynq-7000 EPP містить наступні функціональні блоки: процесорну підсистему, що включає процесорний модуль, інтерфейси пам'яті, периферійні інтерфейси, міжблокові інтерфейси та інтерфейси до програмованої логіки, а також програмовану логіку. При цьому процесор ARM Cortex A9 MPCore має вбудовану пам'ять, багатий набір периферійних пристроїв, інтерфейси до зовнішньої пам'яті. Взаємодія між двома процесорами може здійснюватися за допомогою міжпроцесорних переривань через область загальної пам'яті шляхом передачі повідомлень [6].

Інтегральна схема ZYNC-7000 EPP (Extensible Processing Platform) виконана за технологічним процесом SRAM 28 нм і є FPGA з впровадженими додатковими функціональними блоками. Крім 350 тисяч логічних блоків (для FPGA Z-7045) кристал включає багатоядерний блок обчислювачів, побудований на базі двох ядер процесора ARM Cortex-A9, 2 Мбіт вбудованої багатопортової пам'яті, контролери зовнішніх динамічних оперативних запам'ятовуючих пристроїв (DDR2 і DDR3), контролери пристроїв flash-

пам'яті (NAND і NOR), два вбудовані блоки високошвидкісних 12-розрядних аналого-цифрових перетворювачів, вбудований контролер інтерфейсу PCIe, вбудовані контролери спеціалізованих інтерфейсів таких, як I2C, USB 2 Ethernet, UART, CAN, SPI тощо.

Програмована логіка містить конфігуровані логічні блоки (CLB), конфігуровані двопортові блоки пам'яті (BRAM), комірки цифрової обробки сигналів з 25×18-бітним помножувачем, 48-бітним акумулятором і суматором (DSP48E1), АЦП XADC, керовані блоки формування, конфігурований блок шифрування та автентифікації, конфігурований блок вводу-виводу (Select IO). (рис. 2.1).

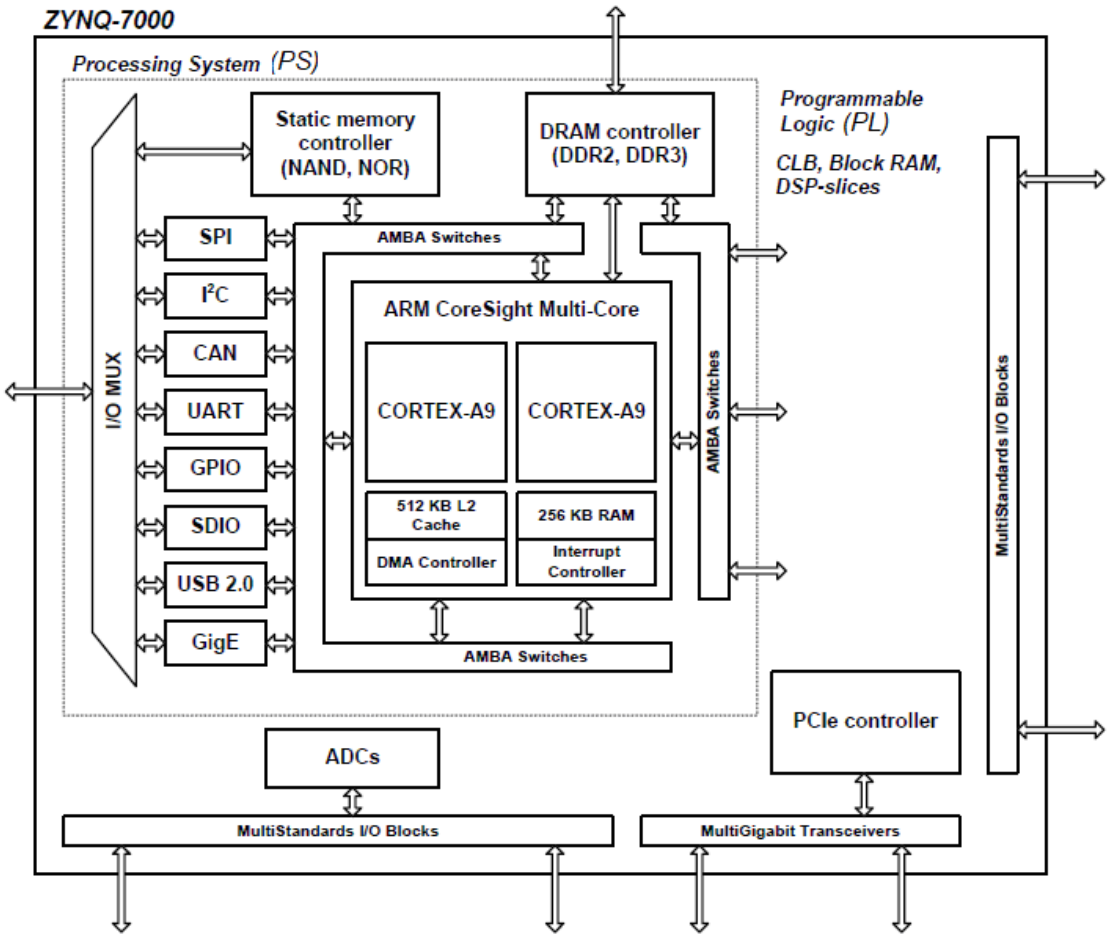


Рисунок 2.1 – Архітектура SoC Xilinx ZYNQ-7000

Xilinx ZYNQ-7000 платформа має архітектуру, де поєднано PS (processing system) – систему обробки та PL (programmable logic) -

програмовану логіку. Взаємодія двох систем використовується через інтерфейс AMBA на основі протоколу AXI.

Однією з популярних платформ розробки для роботи з Zynq є ZedBoard. Назва «Zed» означає Zynq Evaluation and Development. Загальний вигляд плати ZedBoard з елементами інтерфейсу представлений на рис. 2.2.

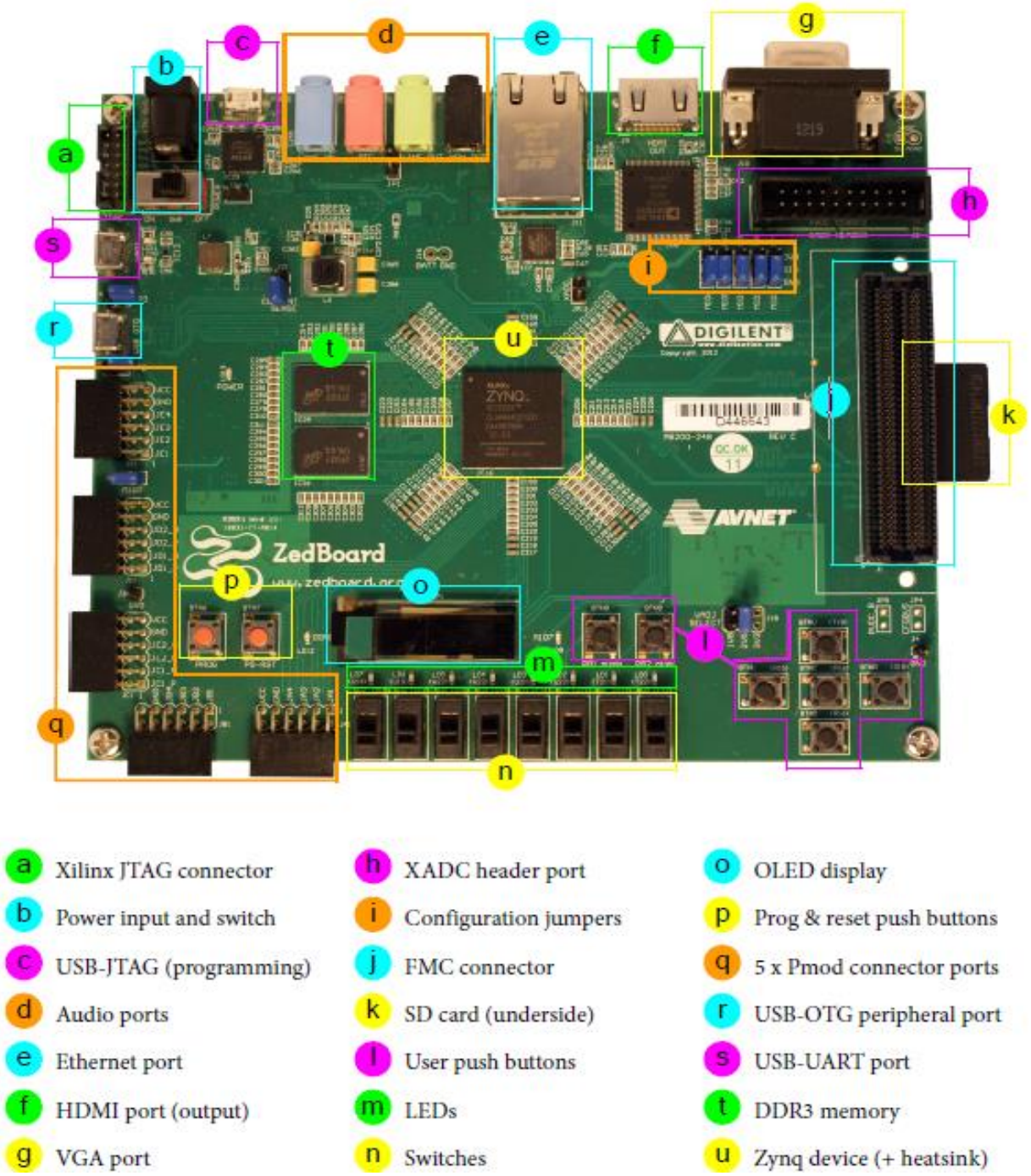


Рисунок 2.2 – Загальний вигляд плати ZedBoard

ZedBoard – це недорога загальнодоступна плата, оснащена пристроєм XC7Z020 Zynq. Це спільне підприємство Xilinx, Avnet (дистриб'ютор) і Digilent (виробник плат). Незважаючи на те, що ZedBoard є платформою розробки для промисловості, він також орієнтований на студентів, учених і любителів, пропонує спеціальні матеріали для нових користувачів Zynq і орієнтований на навчання новачків [7].

2.2 Цифровий інтерфейс AXI Bus

Привабливість Zynq-7000 полягає не тільки у властивостях його складових частин, PS та PL, але й у можливості використовувати їх у тандемі для формування закінчених інтегрованих систем. Ключовим фактором у цьому відношенні є набір ретельно визначених міжз'єднань та інтерфейсів AXI, що утворюють міст між двома частинами. AXI (Advanced eXtensible Interface), що означає розширений розширюваний інтерфейс, є протоколом інтерфейсу, визначений ARM як частина стандарту AMBA (Advanced Microcontroller Bus Architecture, розширена архітектура шини мікроконтролера). Багато пристроїв та IP-блоки, що випускаються сторонніми виробниками та розробниками, засновані на цьому стандарті [7].

Розширена архітектура шини мікроконтролера ARM (Advanced Microcontroller Bus Architecture, AMBA) – це відкрита стандартна специфікація міжз'єднань на кристалі для підключення та управління функціональними блоками в конструкціях система на кристалі (SoC). Це полегшує розробку багатопроцесорних конструкцій із великою кількістю контролерів та компонентів з архітектурою шини. З моменту свого створення AMBA, незважаючи на назву, вийшла далеко за межі мікроконтролерів. Сьогодні AMBA широко використовується в ряді частин ASIC та SoC, включаючи процесори, що використовуються у сучасних портативних мобільних пристроях, таких як смартфони. AMBA – зареєстрована торгова марка ARM Ltd..

AMBA була представлена ARM в 1996 році. Першими шинами AMBA були Advanced System Bus (ASB) та Advanced Peripheral Bus (APB). У другій версії AMBA 2 в 1999 році ARM додала високопродуктивну шину AMBA (AHB), яка є протоколом з одним фронтом тактової частоти. У 2003 році ARM представила третє покоління AMBA 3, включаючи AXI для досягнення ще більш високої продуктивності міжз'єднання та Advanced Trace Bus (ATB) як частина вбудованого рішення CoreSight для налагодження та трасування. У 2010 році були представлені специфікації AMBA 4, починаючи з AMBA 4 AXI4, а потім у 2011 році розширивши загальносистемну узгодженість за допомогою AMBA 4 AXI Coherency Extensions (ACE). У 2013 році було представлено специфікацію AMBA 5 з високошвидкісним транспортним рівнем та функціями, призначеними для зменшення навантаження.

Ці протоколи сьогодні є стандартом де-факто для архітектур із вбудованими процесорами, оскільки вони добре документовані та можуть використовуватись без ліцензійних відрахувань.

AXI - це двоточковий інтерфейс, розроблений для високопродуктивних та швидкодіючих мікроконтролерних систем. Протокол AXI заснований на каналі «точка-точка», що виключає спільне використання шини та дозволяє збільшити пропускну здатність та зменшити затримку. AXI є найпопулярнішим серед усіх міжз'єднань інтерфейсу AMBA. Він забезпечує основу у тому, як різні блоки всередині кожного чіпа взаємодіють друг з одним. Механізм квитування гарантує, що дані передаються чітко та надійно, що дозволяє різним компонентам взаємодіяти без будь-яких перешкод.

Передача даних між Master та Slave пристроями відбувається відповідно до специфікації протоколу AXI:

– ведучий (Master) та ведений (Slave) пристрої повинні встановити «міжз`єднання» (handshake) для автентифікації дійсних сигналів;

– передача сигналу, що управляє, здійснюється в окремі фази;

– є окремі канали передачі сигналу;

– безперервна передача досягається за допомогою пакетного зв'язку.

Протокол AXI визначає 5 каналів (рис. 2.3):

– 2 використовують для транзакцій читання:
  прочитати адресу, прочитати дані;
– 3 використовуються для транзакцій запису:
  записати адресу, записати дані, записати відповідь.

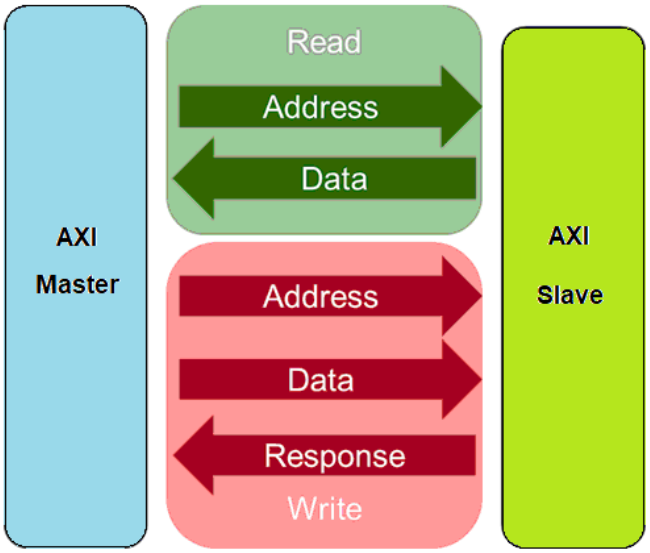


Рисунок 2.3 – Структура протоколу AXI

2.3 Інструментальні засоби розробки вбудованих систем на Zynq-7000

Об'єднання в одному кристалі апаратного двоядерного процесорного блоку з архітектурою ARM Cortex-A9 та програмованої логіки FPGA останнього покоління надає розробникам широкі можливості для проектування вбудованих систем різного призначення з динамічно конфігурованою структурою. Програмовані системи на кристалі All Programmable System-On-Chip (AP SoC) сімейства принципово відрізняються від ПЛІС з апаратними мікропроцесорними ядрами PowerPC сімейств Virtex-4 FX і Virtex-5 FXT, які виробляє компанія Xilinx, як в архітектурному відношенні, так і організацією взаємодії програмованої логіки та процесорного ядра. У ПЛІС цих сімейств мікропроцесорне ядро може

функціонувати лише після завершення процедури конфігурування основних логічних ресурсів. У кристалах обчислювальних платформ сімейства Zynq-7000 AP SoC, що розширюються, апаратний процесорний блок є основним компонентом, який доступний до конфігурування програмованої логіки і може використовуватися незалежно від логічних ресурсів.

Важливою перевагою кристалів процесорних платформ, порівняно з ПЛІС перелічених вище сімейств є істотно менша вартість, що дозволяє ефективно використовувати їх для серійної реалізації вбудованих систем різного рівня складності та функціонального призначення. Активному поширенню програмованих систем на кристалі сімейства Zynq-7000 AP SoC сприяє також надання фірмою Xilinx широкого спектру інструментальних налагоджувальних комплектів та комплексів засобів автоматизованого проектування, що дозволяють у короткі терміни виконувати всі етапи розробки вбудованих мікропроцесорних систем, включаючи налагодження апаратної частини та програмного забезпечення.

Процес проектування вбудованих систем, що реалізуються на базі кристалів програмованої логіки з архітектурою FPGA і розширюваних обчислювальних платформ сімейства Zynq-7000 AP SoC, в загальному випадку включає наступні етапи [7].

1. Розробка проекту мікропроцесорної системи. На цьому етапі визначається архітектура системи та розподіл функцій по виконанню обчислень між апаратною та програмною частиною, використовуючи обрані критерії та метрики (енергоспоживання, швидкість реакції системи, оптимальність виконання блоків). Також додається необхідність вибору цільового процесора або SoC для подальших етапів проектування.

2. Проектування апаратної платформи системи, що розробляється, до якого входить формування проекту апаратної платформи та визначення IP-ядер, необхідних для конкретної задачі. Наступними задачами тут є реалізація апаратної платформи на базі обраного чіпу та її верифікація.

3. Підготовка системних програмних засобів нижнього (апаратного)

рівня. До цього етапу входить розробка та налагодження первинного завантажувача системи (First Stage BootLoader, FSBL) та формування пакету підтримки плати (Board Support Package, BSP). Під час розробки FSBL можуть бути розгорнуті базові засоби для оновлення (Over-The-Air update, OTA), що може спростити подальші етапи тестування системи. Розробка BSP включає в себе формування необхідного пакету драйверів системи та їх тестування/інтеграцію/адаптацію для обраного набору периферії.

4. Формування основного програмного забезпечення системи, що розробляється. На цьому етапі формується архітектура ПЗ системи та пишеться код застосунків разом з виконанням типової ітераційної розробки ПЗ, орієнтованої на тестування (Test Driven Development, TDD).

5. Комплексне моделювання та налагодження. Під час цього етапу виконується інтеграція розроблених програмних та апаратних компонентів на цільовій платформі. Комплексне налагодження містить як перевірку інтеграції компонентів, так і налагодження граничних станів системи (сплячий режим, вихід зі сплячого режиму, перехід у режим енергозбереження, навантажувальне тестування та перевірки безпеки системи в цілому).

6. Генерація завантажувального образу та розгортання розробленої системи. Для обраного варіанту завантаження системи генерується образ, який може містити такі компоненти, як образ для прошивки енергонезалежної вбудованої пам’яті (Embedded Multimedia Memory Card, eMMC), образ для розгортання системи, використовуючи завантажувач NFS boot (Network File System) для розробників, генерацію та прошивання fuse-конфігурації для захисту системи.

Після вибору варіанта конфігурації процесорного блоку, що забезпечує необхідну продуктивність при виконанні функцій вбудованої системи, можна перейти безпосередньо до проектування її апаратної платформи.

Розробка вбудованого програмного забезпечення здійснюється з використанням Vitis IDE. Узагальнений процес проектування ПЗ на базі Vitis

IDE являє собою типовий цикл розробки для вбудованої платформи. Основна частина – це конфігурування платформи для цільової плати та генерація Software Development Kit (SDK).

У типовому процесі проектування для платформи ZYNQ є доцільним використовувати інструментарій як Vitis HLS для синтезу IP блоків, так і Vivado для проектування системи. Написання програмного забезпечення для ARM (PS частини) виконується у середовищі Vitis IDE. На рисунку 2.4 зображено типовий маршрут використання інструментів Xilinx. Цей процес являє собою реалізацію окремих IP блоків у середовищі Vitis HLS та написання відповідних testbench для них, після чого імпорт до середовища Vivado, де інтегруються блоки IP з PS частиною ZYNQ. Після синтезу та отримання бітстриму для конфігурації FPGA-частини ZYNQ виконується операція Export Hardware including bitstream для отримання xsa-файлу. Далі, згідно до створеної архітектури системи, є можливим або згенерувати образ Linux з використанням petalinux на базі Yocto Framework та xsa-файлу, або імпортувати отриманий xsa-файл до Vitis IDE та використовувати baremetal SDK від Xilinx. Також, є можливим проводити налагодження у середовищі Vitis IDE для кросс-компільованого додатку на базі PetaLinux.

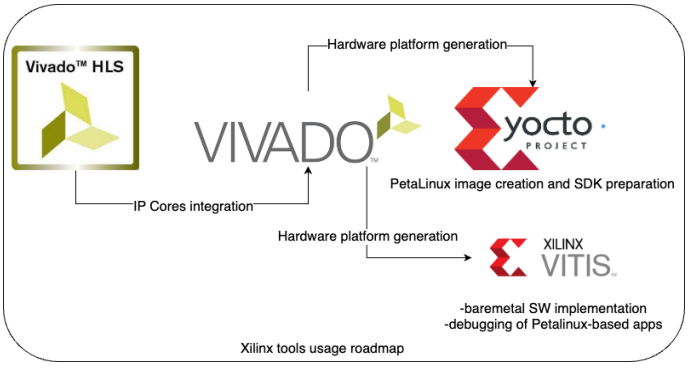


Рисунок 2.4 – Маршрут використання інструментів Xilinx для платформи ZYNQ

Розподіл функцій вбудованих мікропроцесорних систем, що проектуються, між апаратною та програмною частиною можна визначити таким чином. Для функцій, час виконання яких істотно впливає загальну продуктивність проектованих систем, можна успішно використовувати програмний спосіб реалізації операцій.

Вихідні описи таких функцій задаються мовами програмування С і С++. Потім ці описи перетворюються за допомогою відповідних програмних засобів розробки в код, що виконується процесорною системою кристала. При програмній реалізації функції проектованих систем представляються як сукупність операцій, послідовно виконуваних мікропроцесорним ядром. Інакше кажучи, у разі програмної реалізації функцій використовується метод послідовної організації обчислень.

Для функцій, що вимагають великої тривалості використання (завантаження) мікропроцесорного ядра для їх виконання і безпосередньо визначають продуктивність систем, що розробляються, рекомендується застосовувати апаратний спосіб реалізації операцій, який забезпечує можливість паралельної організації обчислень. Апаратна реалізація функцій здійснюється на базі стандартних і спеціалізованих ресурсів програмованої логіки кристалів ПЛІС і процесорних платформ сімейства Zynq-7000 AP SoC, що розширюються. При цьому для підвищення швидкості виконання операцій доцільно застосовувати апаратні секції цифрової обробки сигналів, які включають високошвидкісні помножувачі і суматори.

Подання апаратно реалізованих функцій вбудованих систем, що розробляються, як правило, виконується за допомогою мов опису апаратури, в першу чергу VHDL і Verilog. На основі сформованих HDL-описів засобами автоматизованого проектування генерується відповідна послідовність конфігурації для програмованої логіки кристалів.

На рис. 2.5 показано розподіл функцій вбудованих систем, що проектуються, за способом реалізації операцій між процесорною системою PS і програмованої логікою PL кристалів розширюваних обчислювальних

платформ сімейства Zynq-7000 AP SoC. Пошук оптимального рішення задачі розподілу функцій систем, що проектуються, між апаратною та програмною частиною можна ефективно здійснювати за рахунок нових засобів синтезу високого рівня Xilinx Vivado High-Level Synthesis.

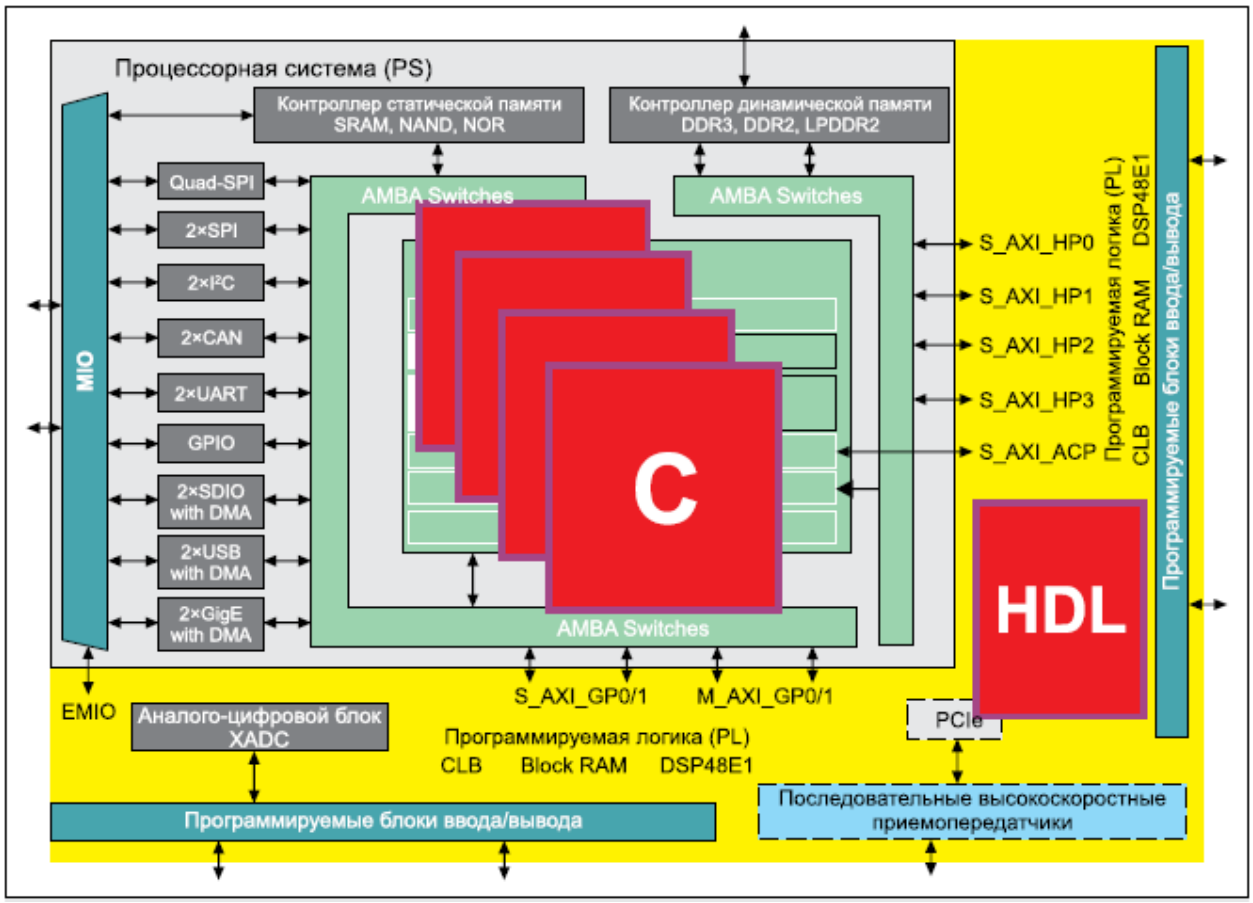


Рисунок 2.5 – Розподіл функцій проектованих систем між PS та PL для кристалів сімейства Zynq-7000 AP SoC

Вони дозволяють задавати вихідні описи всіх функцій систем, що проектуються, за допомогою мов програмування C, C++ і System C. Потім, за допомогою пакета Xilinx Vivado High-Level Synthesis описи окремих функцій, зазначених розробником, можуть бути автоматично перетворені на синтезований HDL-код, як показано на рис. 2.6.

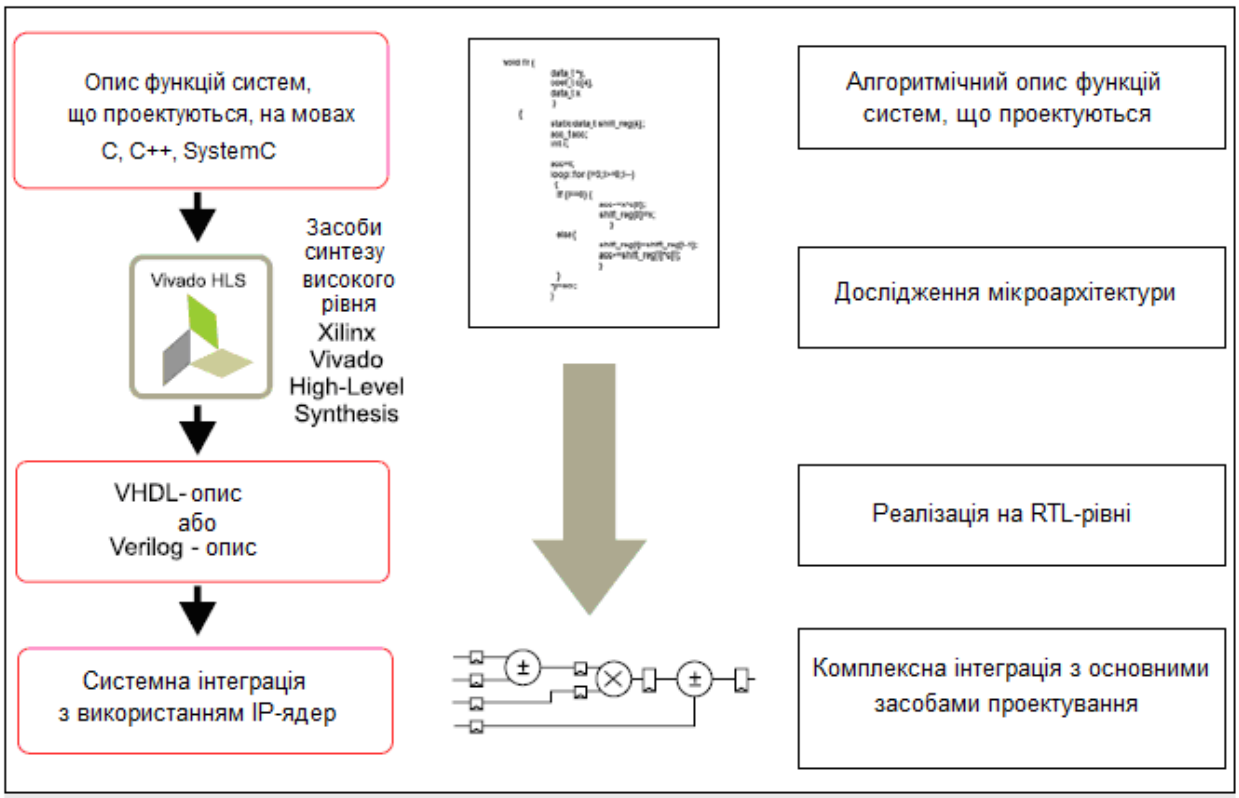


Рисунок 2.6 – Перетворення алгоритмічних описів функцій мовами C, C++ і System C на синтезований HDL-код

Після цього, виконавши етапи синтезу низького рівня, розміщення та трасування проекту системи, що розробляється в кристалі, і використовуючи засоби моделювання, часового аналізу та оцінки енергоспоживання застосовуваної САПР, можна оцінити продуктивність і споживану потужність обраного варіанту реалізації. Всі перелічені процеси з використанням засобів синтезу високого рівня Xilinx Vivado High-Level Synthesis і основних інструментів розробки вбудованих систем на базі кристалів програмованої логіки та обчислювальних платформ, що розширюються, багаторазово повторюються до отримання прийнятних значень параметрів, відповідних умов технічного завдання. Таким чином, проаналізувавши кілька різних варіантів розподілу функцій між процесорною системою та програмованою логікою кристалів, можна вибрати найбільш

ефективний спосіб реалізації операцій для кожної функції вбудованої системи, що проектується.

Технології застосування засобів синтезу високого рівня Xilinx Vivado High-Level Synthesis у процесі розподілу функцій систем, що розробляються, між апаратною та програмною частиною можна визначити наступним чином. Функції, що потребують значних часових ресурсів при програмної реалізації операцій, з допомогою засобів синтезу високого рівня реалізуються у вигляді відповідних апаратних прискорювачів, конфігурованих з урахуванням ресурсів програмованої логіки. Для вирішення завдання оптимального розподілу функцій мікропроцесорних систем, що розробляються, між апаратною і програмною частиною можна також застосовувати інструменти модельно-орієнтованого проектування MATLAB/Simulink компанії MathWorks.

В останніх версіях цих програмних інструментів передбачений спеціальний комплекс засобів, що забезпечує підтримку розробки вбудованих систем на базі кристалів обчислювальних платформ сімейства Zynq-7000 AP SoC, що розширюються. Цей комплекс включає, зокрема, програмні засоби Embedded Coder і HDL Coder. Засоби Embedded Coder надають можливість автоматичного формування компактного та ефективного програмного коду мовами C і C++ на основі моделей систем, що розробляються, підготовлених у середовищі MATLAB/Simulink. C/C++ код, генерований засобами Embedded Coder максимально адаптується до особливостей ПЛІС компоненти архітектури процесорного блоку PS програмованої системи на кристалі сімейства Zynq-7000 AP SoC.

Засоби HDL Coder призначені для автоматичного створення описів апаратної частини систем, що проектуються, мовами VHDL і Verilog. HDL-код, що формується засобами HDL Coder, оптимізується з урахуванням архітектурних особливостей програмованої логіки PL кристалів сімейства Zynq-7000 AP SoC. Для тестування згенерованих C/C++ та HDL-кодів у середовищі MATLAB/Simulink передбачені відповідні засоби верифікації.

Після отримання успішних результатів перевірки сформованих C/C++ та HDL-кодів вони можуть бути перетворені на завантажувальний код процесорної системи PS та конфігураційну послідовність програмованої логіки PL за допомогою САПР серії Xilinx ISE Design Suite або Xilinx Vivado Design Suite. Разом з рішенням задачі розподілу функцій між апаратною і програмною частиною проектованих систем необхідно вибрати варіант конфігурації процесорного блоку, який найбільш повно відповідає вимогам проекту.

3 РЕАЛІЗАЦІЯ ЦИФРОВИХ ФІЛЬТРІВ НА ПЛАТФОРМІ ZedBoard

3.1 Схемна реалізація проекту

Розробку та прототипування обраних цифрових фільтрів було виконано у середовищах Vivado IDE та Vitis HLS. Інтеграційна частина включає до себе вибір реалізації алгоритмів, розробку програмної частини та перевірку коректності реакції фільтрів на одиничний імпульс.

Архітектуру розробленої системи зображено на рис. 3.1

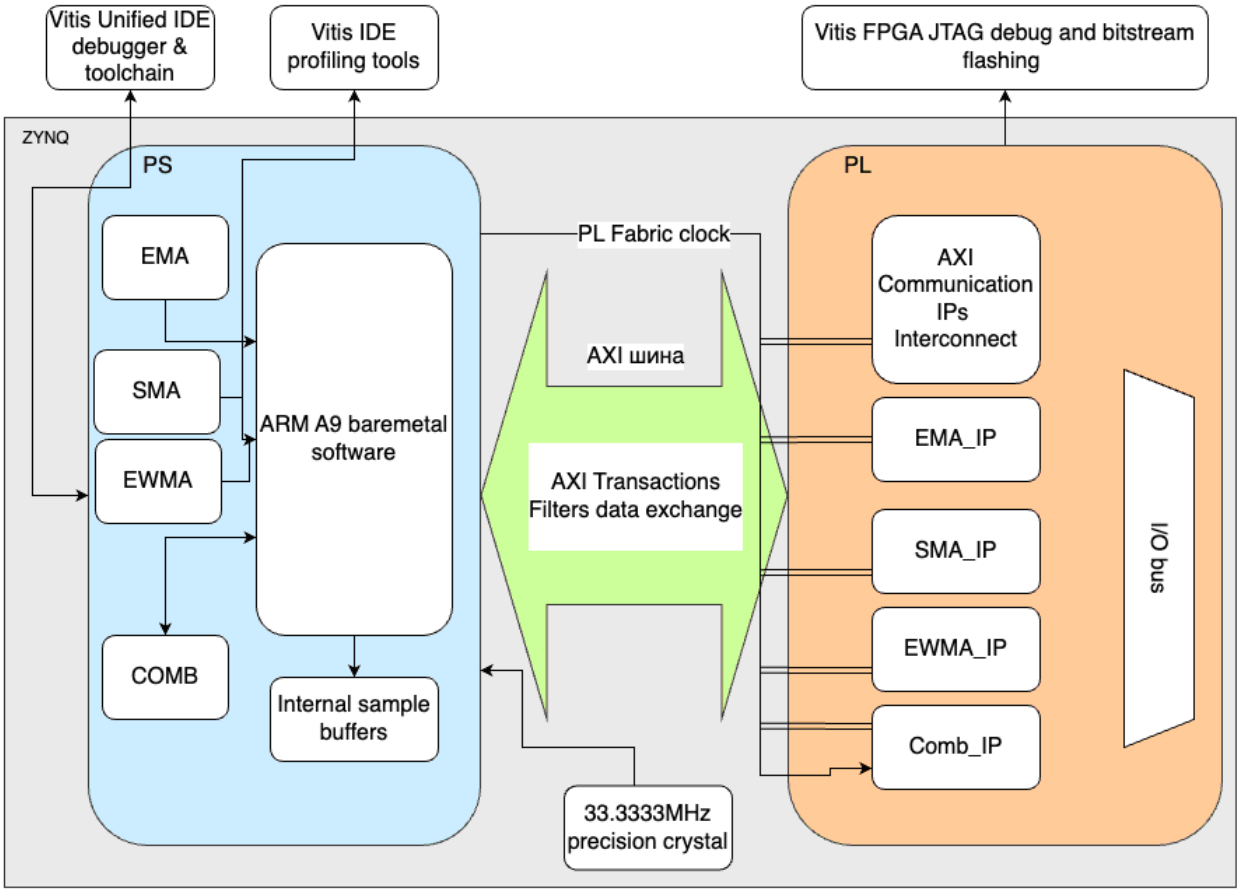


Рисунок 3.1 – Архітектура розробленої системи у середовищі Vivado IDE

Реалізацію фільтрів було виконано як на ARM частині платформи ZedBoard, так і на PL частині відповідно. Для зв’язку з PL частиною було

обрано використання AXI4-Lite інтерфейсу через невеликий обсяг даних, що пересилаються.

Діаграму реалізованої системи наведено на рис. 3.2

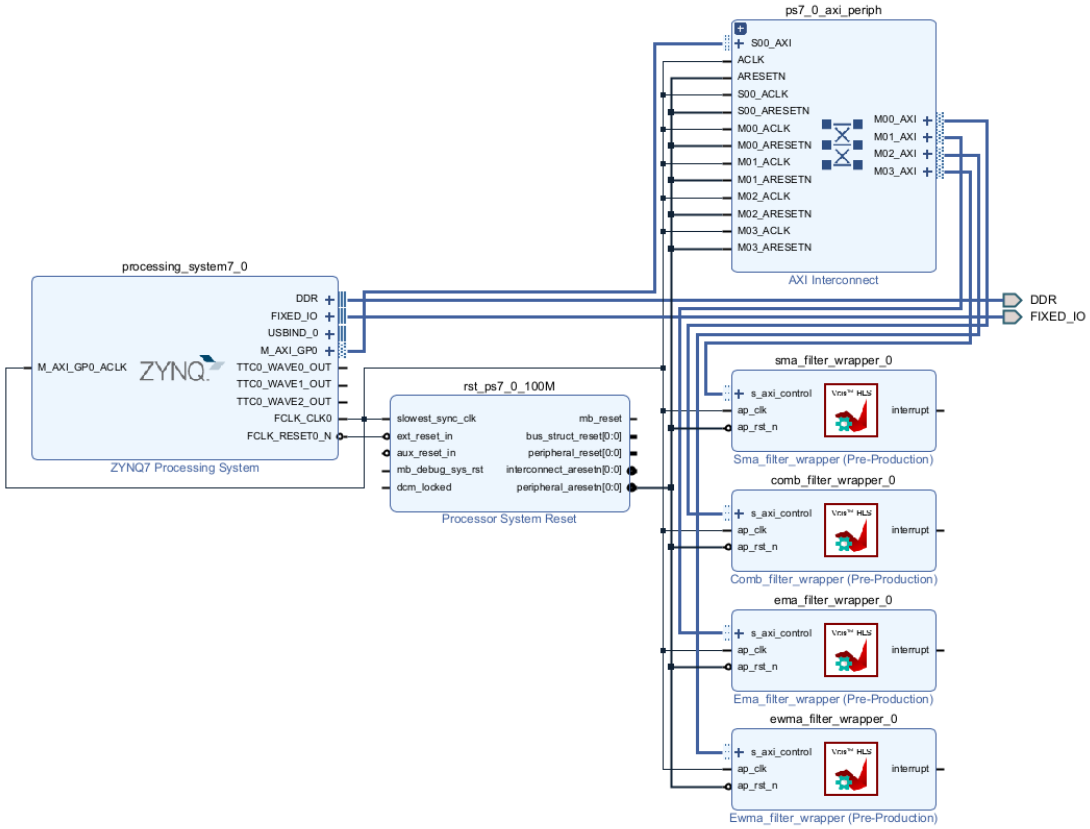


Рисунок 3.2 – Block Diagram розробленої архітектури у середовищі Vivado IDE

Кожен фільтр може бути синтезований як і окремий IP блок, так і бути складовою частиною блоку апаратно-прискореної фільтрації, де алгоритм може бути обраний через конфігурацію блоку за допомогою AXI4-Lite інтерфейсу. Через те, що ці фільтри є лінійними та інваріантні в часі - вихідні дані для лінійної комбінації такі ж, як і лінійна комбінація окремих вихідних даних на вхідні дані. Через цю властивість є можливою як окрема реалізація фільтрів у блоках і їх послідовний виклик, так і їх каскадування під час реалізації у IP ядрі для SoC.

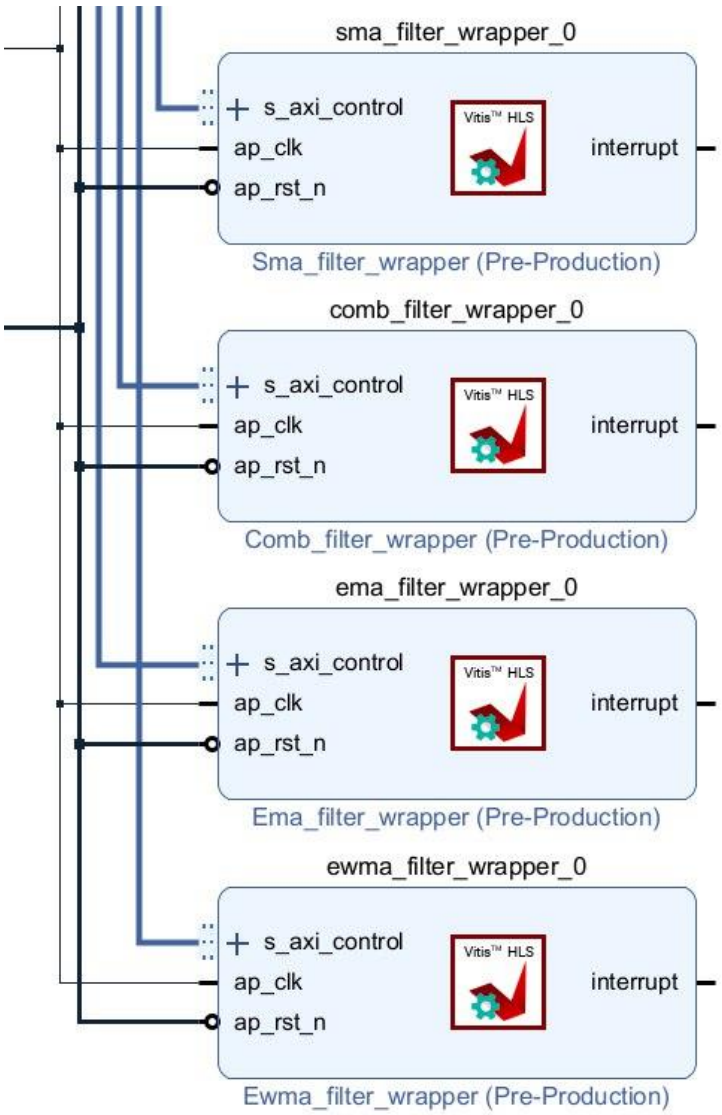


Рисунок 3.3 – Загальний вигляд розроблених IP блоків фільтрів середовищі Vivado

3.2 Програмна реалізація проекту

Основними етапами програмної реалізації проекту є реалізація прототипування фільтрів та отримання їх реакції на одиничний імпульс, реалізацію їх у засобах Vitis HLS для високорівневого синтезу та їх компіляцію для ARM частини.

Реалізація драйверів для взаємодії з PL частиною є частково автоматизованою за допомогою засобів Vitis IDE. Нашою задачею є

реалізація логіки верхнього рівня взаємодії з розробленими блоками.

Для кожного з розроблених IP блоків є необхідним провести процедуру ініціалізації та задачі вхідних параметрів. Фрагмент реалізації EWMA фільтру наведено у Лістингу 3.1.

Лістинг 3.1 – Фрагмент реалізації EWMA фільтру для високорівневого синтезу

```
EwmaFilter<T>::EwmaFilter(T alpha, unsigned int alpha_scale_) {
   init(alpha, alpha_scale_, 0);
   has_initial_ = false;
}

template <typename T>
EwmaFilter<T>::EwmaFilter(T alpha, unsigned int alpha_scale_, T initialOutput) {
   init(alpha, alpha_scale_, initialOutput);
}

template <typename T>
void EwmaFilter<T>::init(T alpha, unsigned int alpha_scale_, T initialOutput) {
   alpha = alpha;
   alpha_scale_ = alpha_scale_;
   output_scaled_ = initialOutput * alpha_scale_;
   has_initial_ = true;
}

template <typename T>
void EwmaFilter<T>::reset() {
   has_initial_ = false;
}

template <typename T>
T EwmaFilter<T>::ProcessSamples(T input) {
   if (has_initial_) {
     output_scaled_ = alpha*input+(alpha_scale_- alpha) * output_scaled_  / alpha_scale_;
   } else {
      output_scaled_ = input * alpha_scale_;
      has_initial_ = true;
   }
   return output();
}

template <typename T>
T EwmaFilter<T>::output() {
   return (output_scaled_ + alpha_scale_ / 2) / alpha_scale_;
}
```

Було реалізовано EWMA, EMA, SMA та гребінчастий фільтри і визначено апаратні витрати на кожен з них при синтезі у SoC.

Для реалізації гребінчастого фільтру була виконана реалізація структури кільцевого буфера для зберігання вхідних відліків сигналу та відповідно їх додавання з затримкою до вхідного потоку даних. Фрагмент реалізації гребінчастого фільтра наведено у Лістингу 3.2.

Лістинг 3.2 – Фрагмент реалізації гребінчастого фільтру

```
void dsp_comb(ModeSelection_t mode, float *sample, uint32_t extra_args)
{

  float delayed_sample = 0;
  ModeSelection_t mode_enum = (ModeSelection_t) (mode);
  switch (mode_enum) {

  case InitFilter:
          dsp_init_stereo_comb(&comb_filter_instance, extra_args);
          break;

  case BypassMode:
  dsp_delay_line_update(&comb_filter_instance.delay_line, *sample);
          break;

  case ProcessSamples:

          delayed_sample = dsp_delay_line_update(
                  &comb_filter_instance.delay_line, *sample);
          *sample -= delayed_sample;

          break;
  case Reset:
          dsp_stereo_comb_reset(&comb_filter_instance);
          break;
  }
```

Базовою структурою гребінчастого фільтру є кільцевий буфер, який реалізований за принципом FIFO. Фрагмент методу dsp_delay_line_update наведено у Лістингу 3.3.

Лістинг 3.3 – Фрагмент реалізації кільцевого буфера для гребінчастого фільтру

```
void dsp_init_delay_line(DelayLine_t* delay_line, uint32_t delay_line_length) {
  for (int i = 0; i < DELAYLINE_MAX_LEN; ++i) {
          #pragma HLS PIPELINE II=1
          delay_line->circular_buffer_memory[i] = 0;  }
```

```
  delay_line->cirular_buffer_idx = 0;
  delay_line->delay_line_length = delay_line_length;
}

float dsp_delay_line_update(DelayLine_t* delay_line, float new_sample) {
  float output_sample =
          delay_line->circular_buffer_memory[delay_line->cirular_buffer_idx];

  delay_line->circular_buffer_memory[delay_line->cirular_buffer_idx] = new_sample;
  ++delay_line->cirular_buffer_idx;

  if (delay_line->cirular_buffer_idx >= delay_line->delay_line_length) {
  delay_line->cirular_buffer_idx -= delay_line->delay_line_length;}
  return output_sample;
}
```

Загальний інтерфейс в них є однаковим-функція ініціалізації та обробки надходження нового відліку сигналу і отримання фільтрованого відповідно. За рахунок спільного інтерфейсу була реалізована можливість вибору синтезу фільтру у середовищі Vitis HLS. Фрагмент реалізації підготовлений для обробки високорівневим синтезатором наведено у Лістингу 3.4.

Лістинг 3.4 – Фрагмент реалізації IP блоку фільтрів для Vitis HLS

```
static EmaFilter<float> emaFilter{0.5};
static SmaFilter<float,4> smaFilter{};
static CombFilterWrapper combFilter(10);
static EwmaFilter<float> ewmaFilter(1,10);

void filter_wrapper(float input_value,float* output_value)
{
#pragma HLS INTERFACE s_axilite port =input_value bundle = control
#pragma HLS INTERFACE s_axilite port=output_value bundle = control
#pragma HLS INTERFACE s_axilite port = return bundle = control
   *output_value = emaFilter.ProcessSamples(input_value);
}
```

Прототипування фільтрів перед переносом у SoC було виконано на ПК з архітектурою x86_64. Для перевірки коректності реалізації кожного з алгоритмів була виконана подача одиничного імпульсу на вхід фільтру для отримання імпульсної характеристики. Результати подачі одиничного імпульсу та реакцію EMA фільтру наведено на рис. 3.5. На графіку синя лінія відповідає вхідному сигналу, тобто одиничному імпульсу, помаранчева відповідає виходу фільтру відповідно.

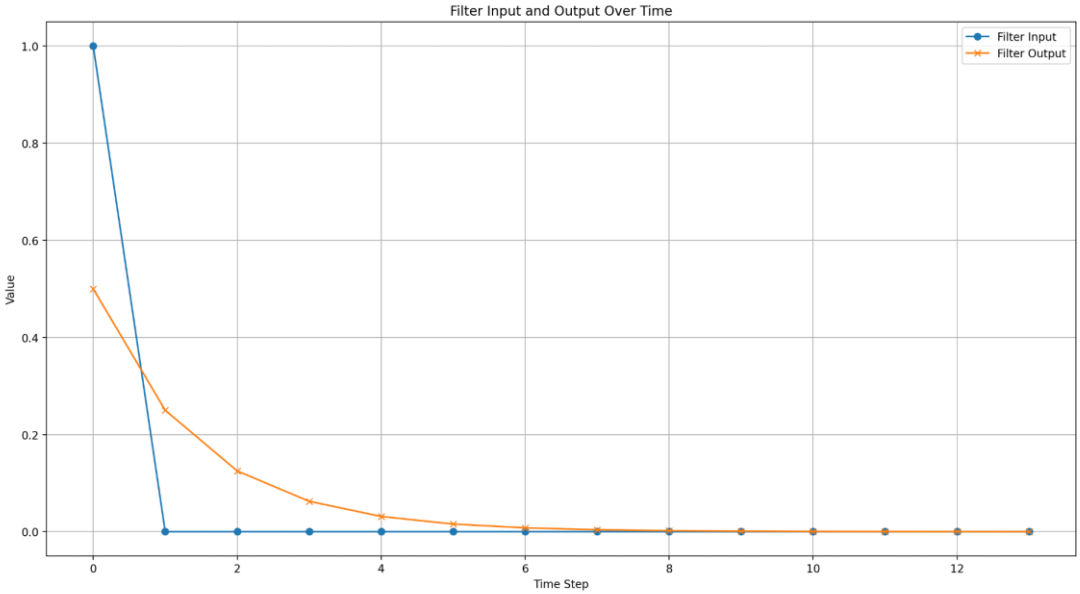


Рисунок 3.5 – Реакція EMA фільтру на подачу одиничного імпульсу

Результати подачі одиничного імпульсу та реакцію SMA фільтру наведено на рис. 3.6. Відповідно до вікна фільтра у чотири семпли на графіку видно, що після подачі одиничного імпульсу на виході формується коректна вихідна послідовність згідно до рівняння медіанного фільтру.

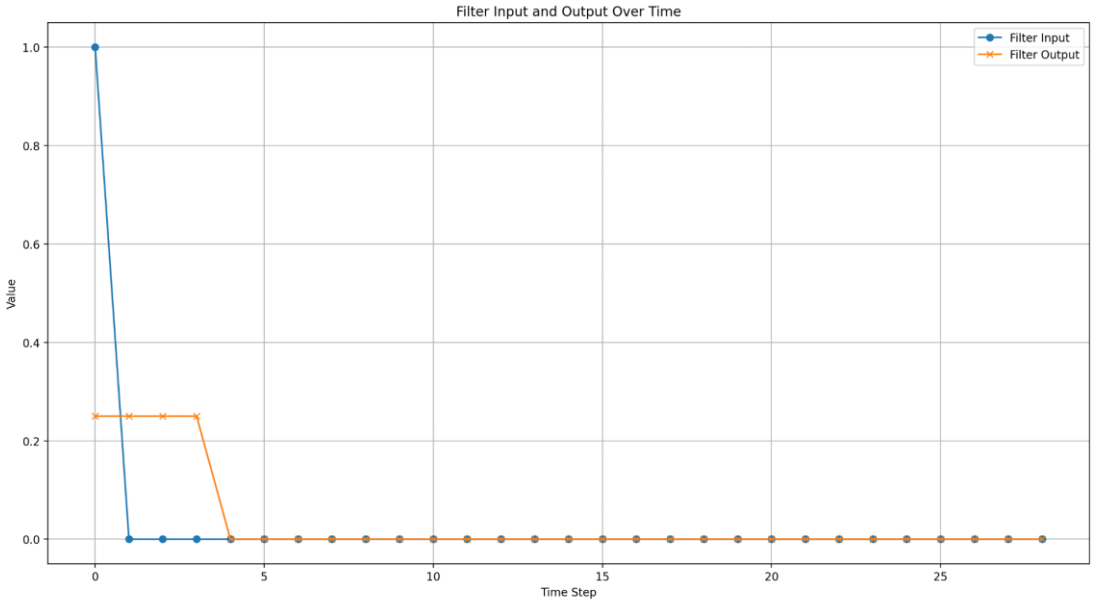


Рисунок 3.6 – Реакція SMA фільтру на подачу одиничного імпульсу

Результати подачі одиничного імпульсу та реакцію гребінчастого фільтру наведено на рис. 3.7. Відповідно до лінії затримки сигналу в 10

відліків на графіку видно, що 10-й відлік дорівнює інвертованому поданому одиничному імпульсу.

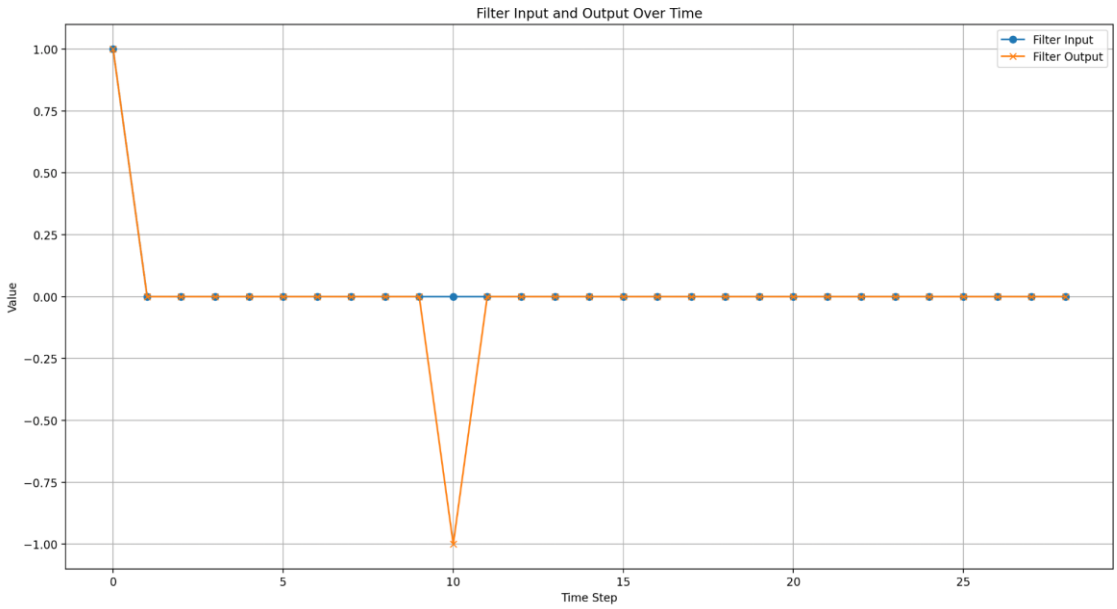


Рисунок 3.7 – Реакція гребінчастого фільтру на подачу одиничного імпульсу

Результати подачі одиничного імпульсу та реакцію EWMA фільтру наведено на рис. 3.8. На графіку помітно повільний час реакції фільтру з параметром alpha=0.1

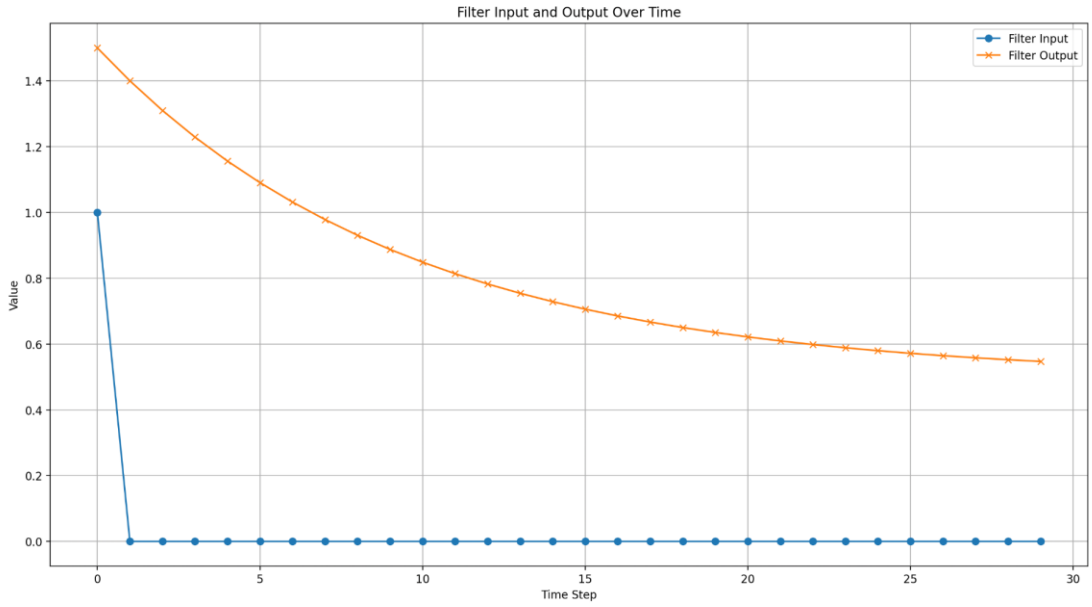


Рисунок 3.8 – Реакція EWMA фільтру на подачу одиничного імпульсу

3.3 Макетування фільтрів на платі ZedBoard

Макетування розроблених фільтрів було виконано на платі налагодження ZedBoard від Digilent. Для перевірки було виконано діагностичний експеримент з метою визначення реакції на одиничний імпульс кожного з розроблених фільтрів і порівнянням з прототипом. З метою аналізу отриманого результату було використано вбудований відладчик Vitis IDE. На рис. 3.9. зображено скриншот з середовища Vitis з отриманими результатами профілювання часу виконання фільтрів. На рис. 3.10. зображено процес макетування фільтрів у середовищі Vitis з платою налагодження ZedBoard.

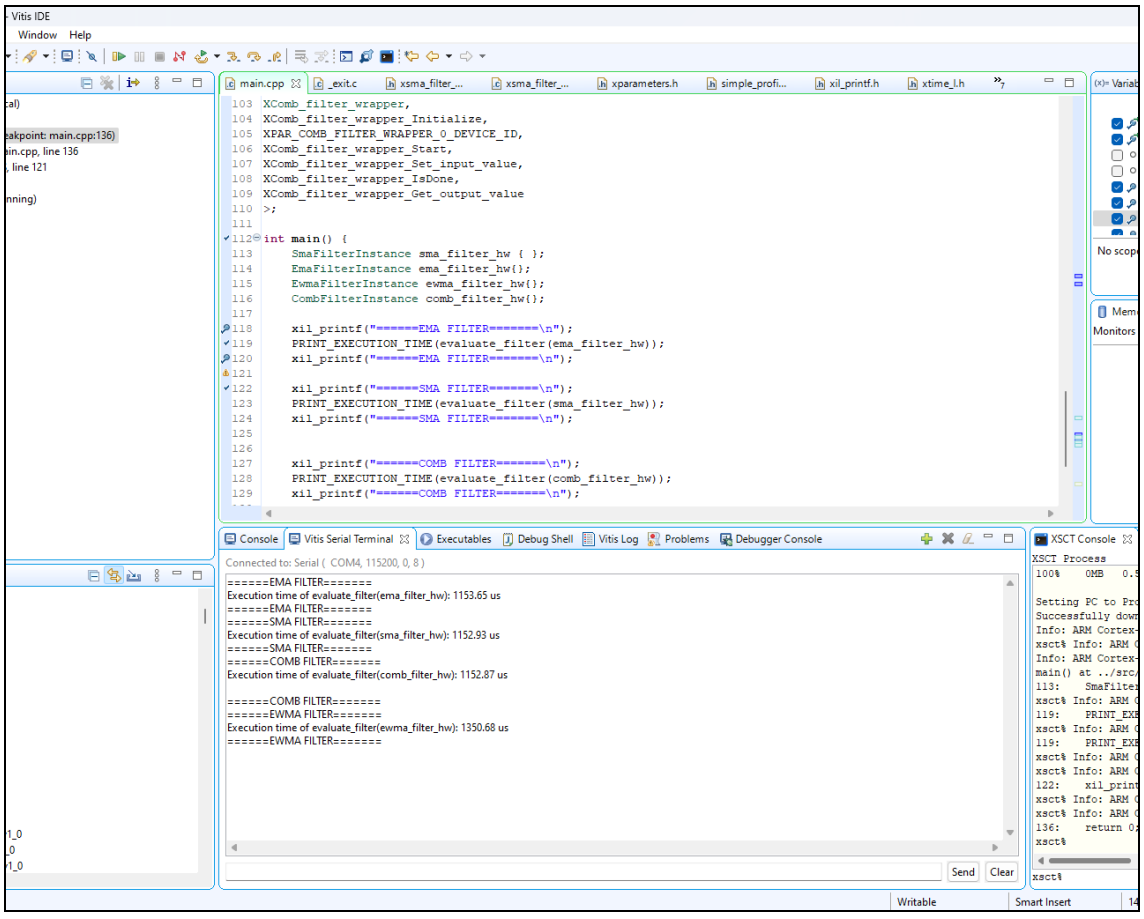


Рисунок 3.9 – Скриншот з Vitis IDE з фрагментом прошивки для визначення часу виконання фільтрів

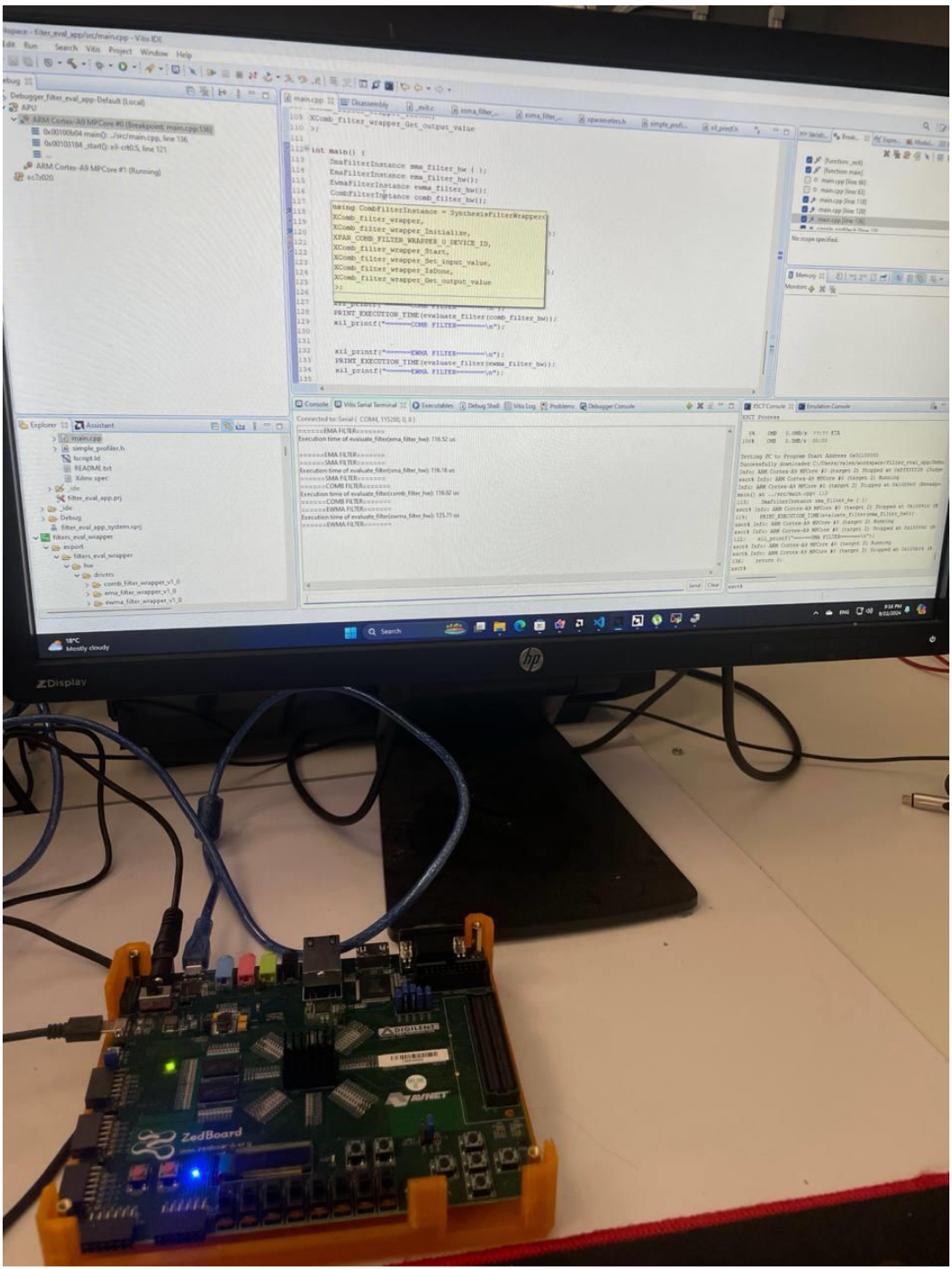


Рисунок 3.10 – Процес макетування фільтрів на платформі ZedBoard

У таблиці 3.1 наведено фрагмент перевірки коректності реалізації фільтрів шляхом виведення діагностичних повідомлень у вбудований термінал Vitis IDE. Результати використання апаратних витрат та швидкодії синтезованих блоків наведено на у таблиці 3.2.

Таблиця 3.1 – Результати перевірки коректності виконання фільтрів на платформі ZedBoard

```
======EMA FILTER=======
Filter Input: 1.000000 Output: 0.500000
Filter Input: 0.000000 Output: 0.250000
Filter Input: 0.000000 Output: 0.125000
Filter Input: 0.000000 Output: 0.062500
Filter Input: 0.000000 Output: 0.031250
Filter Input: 0.000000 Output: 0.015625
Filter Input: 0.000000 Output: 0.007812
Filter Input: 0.000000 Output: 0.003906
Filter Input: 0.000000 Output: 0.001953
Filter Input: 0.000000 Output: 0.000977
Filter Input: 0.000000 Output: 0.000488
Filter Input: 0.000000 Output: 0.000244
Filter Input: 0.000000 Output: 0.000122
Filter Input: 0.000000 Output: 0.000061
Filter Input: 0.000000 Output: 0.000031

======COMB FILTER======
Filter Input: 1.000000 Output: 1.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: -1.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000

======SMA FILTER=======
Filter Input: 1.000000 Output: 0.250000
Filter Input: 0.000000 Output: 0.250000
Filter Input: 0.000000 Output: 0.250000
Filter Input: 0.000000 Output: 0.250000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000
Filter Input: 0.000000 Output: 0.000000

======EWMA FILTER=======
Filter Input: 1.000000 Output: 1.500000
Filter Input: 0.000000 Output: 1.400000
Filter Input: 0.000000 Output: 1.310000
Filter Input: 0.000000 Output: 1.229000
Filter Input: 0.000000 Output: 1.156100
Filter Input: 0.000000 Output: 1.090490
Filter Input: 0.000000 Output: 1.031441
Filter Input: 0.000000 Output: 0.978297
Filter Input: 0.000000 Output: 0.930467
Filter Input: 0.000000 Output: 0.887421
Filter Input: 0.000000 Output: 0.848678
Filter Input: 0.000000 Output: 0.813811
Filter Input: 0.000000 Output: 0.782430
Filter Input: 0.000000 Output: 0.754187
Filter Input: 0.000000 Output: 0.728768
```

Таблиця 3.2 – Зведена таблиця параметрів реалізованих фільтрів

Тип фільтру	LUT	FF	DSP	BRAM	Час обробки 100 семплів(μs)
SMA	1021	755	5	0	116.19
EWMA	1171	936	5	0	135.67
Comb	373	432	2	4	116.04
EMA	421	585	8	0	116.53

Програмний код для аналізу швидкодії (затримки сигналу) наведений у Додатку Б.

Згідно до отриманих результатів можна зробити висновок, що гребінчастий фільтр є найменш затратним з боку використання LUT і DSP блоків SoC. У той самий час, EWMA фільтр використовує найбільшу кількість я LUT і FF, так і займає частину наявних DSP блоків. За швидкодією всі фільтри мають майже однакові характеристики по часу виконання у мікросекундах. Але найшвидшим є гребінчастий фільтр. найповільнішим є фільтр EWMA який був повільніше, ніж інші .

Таким чином, гребінчастий фільтр є найкращім за всіма характеристиками.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проаналізовано однорідні фільтри та гребінчасті. Приведено їх структури, порівняльну характеристики. Описані переваги та недоліки. Наведено їх особливості застосування.

В якості технологічної платформи обрано програмовану систему на кристалі FPGA сімейства ZYNQ-7000 фірми Xilinx Inc. Платформа Xilinx ZYNQ-7000 має архітектуру, де поєднано PS (processing system) – систему обробки та PL (programmable logic) - програмовану логіку. Взаємодія двох систем використовується через інтерфейс AMBA на основі протоколу AXI. В якості налагоджувальної плати використана недорога загальнодоступна плата ZedBoard, яка оснащена пристроєм XC7Z020 Zynq.

Сформульована загальна послідовність етапів проектування системи на кристалі на платформі ZYNQ-7000. Практична реалізація виконана на базі стеку інструментальних засобів САПР Vivado/Vitis/Vitis HLS. Це дозволяє виконати наскрізне проектування користувацького програмного забезпечення з боку Processing System (PS) на мові програмування С.

Для реалізації гребінчастого фільтру було реалізовано кільцевий буфер для збереження вхідних семплів та формування лінії затримки. Окремо було реалізовано методи для встановлення довжини лінії затримки і передачі вхідних семплів сигналу. Для реалізації профілювання часу виконання функцій було реалізовано макрос, що використовує вбудований таймер ARM A9 через виклик XTime_GetTime() з Vitis SDK. Порівняльний аналіз програмно-апаратних реалізацій різних типів фільтрів показав, що гребінчастий фільтр є найкращім за всіма характеристиками.

Наукова новизна роботи полягає у подальшому розвитку методів автоматизованого проектування цифрових фільтрів, реалізованих на технологічній платформі систем на кристалі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lee E. A., Seshia S. A. Introduction to embedded systems: A cyber-physical system approach. – UC Berkeley, 2011. – 499 p.

2. Greaves D. Modern system-on-chip design on ARM. – Arm Education Media, 2021. – 608 p.

3. Мандзій Б. А. Основи теорії сигналів: підручник для вищих навчальних закладів України / Б. А. Мандзій, Р. І. Желяк. Національний університет «Львівська політехніка». – Львів: Ініціатива, 2008. – 239 с.

4. Філатова Г.Є. Проектування цифрових фільтрів: навчальний посібник / Г.Є. Філатова. – Х. : НТУ «ХПІ», 2017. – 120 с.

5. The Zynq Book. Embedded Processing with the ARM® Cortex®-A9 on the Xilinx® Zynq®-7000 All Programmable SoC [Electronic resource] – Zynqbook.com – Mode of access: http://www.zynqbook.com/ – Date of access: 17.09.2023.

6. Зотов В.Ю. Средства автоматизированного проектирования и этапы разработки встраиваемых микропроцессорных систем на базе расширяемых процессорных платформ семейства ZYNQ-7000 AP SOC / В.Ю. Зотов // Компоненты и Технологии. – 2014. – №3 (152). – С. 79-88.

7. Шкіль О. Проєктування та самодіагностика кіберфізичних пристроїв керування на платформі SoC / О. Шкіль, Д. Рахліс, І. Філіпенко, В. Корнієнко // Сучасний стан наукових досліджень та технологій в промисловості. – 2023. – № 4 (26). – С. 122-134.

8. Шкіль О. Автоматизоване проєктування вбудованих систем цифрового оброблення сигналів на платформі SoC / О. Шкіль, Д. Рахліс, І. Філіпенко, В. Корнієнко, Т. Рожнова // Сучасний стан наукових досліджень та технологій в промисловості. – 2024. – № 1 (27). – С. 72-83.

9. Карась Д.В. Автоматизоване проектування цифрових фільтрів на платформі SoC / О.С. Шкіль, В.Р. Корнієнко, Д.В. Карась // Проблеми

інформатизації: тези доп. 12-ї міжнар. наук.-техн. конф., 21-22 листопада 2024 р., Баку – Харків – Бельсько-Бяла: . Т. 3 : секція 5 / Інститут систем управління МНО Азербайджанської республіки [та ін.]. – Харків : ХНУРЕ, 2024. – С. 11.