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

Факультет Інформатики та обчислювальної техніки

Обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій Стіренко

«___»_____2021р

.

**Дипломний проєкт**

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Комп'ютерні системи та мережі»**

**спеціальності 123 «Комп'ютерна інженерія»**

**на тему: «Блок обчислення функцій арктангенса і модуля вектора»**

Виконав: студент 4 курсу, групи ІО-72
Липай Владислав Сергийович _____ (підпис)

Керівник:
д.т.н. Сергієнко Анатолій Михайлович _____ (підпис)

Консультант з нормо-контролю:
проф. д.т.н. Сімоненко Валерій Павлович _____ (підпис)

Рецензент: _____ (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____ (підпис)

Київ – 2021 рок

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

Факультет Інформатики та обчислювальної техніки

Обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Сергій Стіренко
«___»_____2021 р.

**ЗАВДАННЯ**

**на дипломний проєкт студенту**

**Липая Владислава Сергийовича**

1. Тема проєкту «Блок обчислення функцій арктангенса і модуля вектора», керівник проєкту Сергієнко Анатолій Михайлович, д.т.н, затверджені наказом по університету від «__» травня 2021 р. № _____

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту

4. Консультанти розділів проєкту

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормо-контроль | Сімоненко В. П. проф. | | |

7. Дата видачі завдання 10.12.2021 року_____

## Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконанная етапів проєкту(роботи) | Примітка |
|---|---|---|---|
| 1. | Затвердження теми роботи | 10.12.2021-15.12.2021 | |
| 2. | Вивчення та аналіз завдання | 15.12.2021-15.03.2021 | |
| 3. | Розробка архітектури та загальної структури систем | 15.03.2021-25.03.2021 | |
| 4. | Розробка структур окремих підсистем | 25.03.2021-5.04.2021 | |
| 5. | Програмна реалізація системи | 5.04.2021-15.04.2021 | |
| 6. | Оформлення пояснювальної записки | 15.04.2021-20.05.2021 | |
| 7. | Захист програмного продукту | 25.04.2021 | |
| 8. | Передзахист | 23.05.2021 | |
| 9. | Захист | 14.06.2021 | |

Студент Липай Влаислав

Керівник Сергієнко Анатолій

## Анотація

Робота присвячена огляду властивостей алгоритму CORDIC та зроблено висновок про те, що алгоритм CORDIC може бути використаний в двох режимах – «поворот» і «вектор». Досліджено головні області використання досліджуваного алгоритму та перелічено основні підходи до його класифікації.

Проведено аналіз можливостей мови VHDL та названо її функціональні можливості. Коротко описано її походження та проаналізовано способи використання на практиці. Досліджено та проаналізовано архітектурно-структурну організацію ПЛІС. Розглянуті сучасні напрямки розвитку ПЛІС-технологій.

Зроблено розробку модулю обчислення функції arctg y / x та $\sqrt{x^2 + y^2}$ з плаваючою комою та приведено програмні коди на мові VHDL для досліджуваних функцій.



## Abstract

The paper reviews the properties of the CORDIC algorithm and concludes that the CORDIC algorithm can be used in two modes – «rotation» and «vector». The main areas of application of the algorithm under study are investigated and the main approaches to its classification are described.

An analysis of the capabilities of the VHDL language and named its functionality. Its use is briefly described and ways of using it in practice are analyzed. The architectural and structural organization of FPGA is studied and analyzed. Modern trends in the development of FPGA technologies are considered.

The development of the module for calculating the function arctg y / x, $\sqrt{x^2 + y^2}$ with a floating point is made, and program codes in the VHDL language for the studied functions are given.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

| № з/п | форм | Позначення | Найменування | Кількість | Примітка |
|---|---|---|---|---|---|
| 1 | А4 | | Завдання на дипломний проект | 2 | |
| 2 | А4 | | Відомість проекту | 1 | |
| 3 | А4 | ІАЛЦ.467449.001 ПЗ | Пояснювальна записка | 60 | |
| 4 | А4 | ІАЛЦ.467449.002 Д1 | Принципова схема | 1 | |
| 5 | А4 | ІАЛЦ.467449.003 Д2 | Структурна схема | 1 | |
| 6 | А4 | ІАЛЦ.467449.004 Д3 | Код програми | 6 | |
| 7 | А4 | ІАЛЦ.467449.005 Д4 | Функціональна схема алгоритму обчислення функцій sin ϕ, cos ϕ | 1 | |
| 8 | А4 | ІАЛЦ.467449.006 Д5 | Схематичне зображення архітектури CORDIC | 1 | |
| 8 | А4 | ІАЛЦ.467449.006 Д5 | Приклад процесу проектування програмованих схем | 1 | |

# Пояснювальна записка

## до дипломного проєкту

## на тему: «Блок обчислення функцій арктангенса і модуля вектора»

**ЗМІСТ**

ВСТУП ..... 9

РОЗДІЛ 1 ЗАГАЛЬНА ХАРАКТЕРИСТИКА CORDIC-АЛГОРИТМІВ ..... 11

1.1. Огляд та властивості алгоритму CORDIC ..... 11

1.2. Основні підходи до класифікації CORDIC-алгоритмів ..... 25

1.3. Процесорні елементи з набором великих операцій ..... 27

РОЗДІЛ 2 АНАЛІЗ МОЖЛИВОСТЕЙ МОВИ VHDL ..... 31

2.1. Способи HDL-опису простих вузлів ..... 31

2.2. Функціональні можливості мови VHDL ..... 34

РОЗДІЛ 3 РОЗРОБКА МОДУЛЮ ОБЧИСЛЕННЯ ФУНКЦІЇ arctg y / x ТА sqrt (x2+y2) З ПЛАВАЮЧОЮ КОМОЮ ..... 41

3.1. Реалізація об’єкту дослідження в схемах ПЛІС ..... 41

3.2. Програмні коди на мові VHDL для досліджуваних функцій ..... 57

ВИСНОВКИ ..... 62

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 63

ДОДАТКИ ..... **Ошибка! Закладка не определена.**

# ВСТУП

**Актуальність теми.** Програмовані логічні інтегральні схеми (ПЛІС) – це сучасна елементна база, яка призначена для високопродуктивного виконання спеціалізованих алгоритмів з числами, які представлені з фіксованою комою. Дуже часто в таких алгоритмах зустрічається обчислення елементарних функцій. Але постачальники САПР ПЛІС не забезпечують розробників готовими високопродуктивними віртуальними модулями обчислення елементарних функцій, а фірми-розробники таких модулів поширюють їх за велику ціну (близько тисячі доларів США). Крім того, серед них не зустрічаються модулі, які здатні обчислювати кілька різних функцій. Отже, існує нестача в проектах пристроїв для обчислення елементарних функцій в ПЛІС і вони потребують удосконалення [9, с. 112].

**Ступінь розробки теми.** Комплексному дослідженню сутності алгоритму CORDIC та можливостей його застосування в мові VHDL для обчислення функції arctg y / x та $\sqrt{x^2 + y^2}$ з плаваючою комою присвячені роботи таких вчених, як Н. А. Петровский, М. Парфенюк [17], М. В. Рудницький, І. А. Клименко [19], Я. М. Николайчук [14], С. Р. Бікташева, Л. В. Мороз, М. Ю. [2], В. В. Яцків [24].

**Метою проекту** в роботі є аналіз використання алгоритму CORDIC в мові VHDL для обчислення функції arctg y / x та $\sqrt{x^2 + y^2}$ з плаваючою комою.

**Об’єктом нашого проекту** є сукупність необхідних умов, що забезпечують найкращий підхід для розуміння алгоритму CORDIC в мові VHDL для обчислення функції arctg y / x та $\sqrt{x^2 + y^2}$ з плаваючою комою.

**Предметом проекту** є алгоритм CORDIC.

**Методи проекту:** теоретичний аналіз наукової літератури; аналіз та узагальнення. Статистичні дані та порівняння. Класифікація теоретичного матеріалу та розробка рекомендацій.

Вирішення поставлених у роботі завдань здійснювалося з використанням системного підходу в доборі матеріалу, методів індуктивного і логічного аналізу, статистичні методи аналізу літературних даних.

У процесі роботи, залежно від поставлених цілей і завдань, використовувалися відповідні методи аналізу: структурного і системного, порівняльного і факторного аналізу, які ґрунтуються на застосуванні основних принципів логічних та статистичних методів оцінки первинного матеріалу.

**Завдання проекту** полягає в:

- провести загальну характеристику CORDIC-алгоритмів;
- зробити огляд та властивості алгоритму CORDIC;
- описати основні підходи до класифікації CORDIC-алгоритмів;
- перелічити процесорні елементи з набором великих операцій;
- дослідити способи HDL-опису простих вузлів та оцінити функціональні можливості мови VHDL;
- розробити модуль обчислення функції arctg y / x та $\sqrt{x^2 + y^2}$з плаваючою комою та проаналізувати їх в схемах ПЛІС;
- розробити програмні коди на мові VHDL для досліджуваних функцій.

**Новизна проекту.** Зроблено широкий літературний пошук з детальним аналізом наукової інформації. Проведено систематизацію та адаптацію отриманих літературних результатів. Зроблено рекомендації для покращення існуючої системи розрахунку функції arctg y / x та $\sqrt{x^2 + y^2}$ за допомогою алгоритму CORDIC і подальшої їх реалізацією в мові VHDL.

**Структура роботи.** Відповідно до мети і завдань дослідження структура дипломної роботи складається зі вступу, трьох розділів, висновків та списку використаної літератури. За час роботи опрацьовано 37 літературних джерел. Зміст дипломної роботи викладено на 67 сторінках машинописного тексту.

**Джерелами інформації** для вирішення перерахованих вище завдань є збірники наукових праць, монографії, періодична література, підручники та довідники, періодичні фахові журнали.

# РОЗДІЛ 1
# ЗАГАЛЬНА ХАРАКТЕРИСТИКА CORDIC-АЛГОРИТМІВ

## 1.1. Огляд та властивості алгоритму CORDIC

Алгоритм CORDIC був введений у вжиток в 1959 році Вольдером. У версії Вольдера CORDIC дозволяє виконувати обертання і множити або ділити числа, використовуючи тільки елементарні кроки зсуву і додавання. В результаті таких операцій утворюються функції синуса, косинуса і арктангенса.

Алгоритм CORDIC використовується для швидкого обчислення елементарних функцій, таких як множення, ділення тригонометричних та логарифмічних функцій, такі як перетворення прямокутної координати в полярні координати і навпаки. Хоча CORDIC не може бути найшвидшим методом для виконання цих операцій, але він привабливий через простоту своєї апаратної реалізації, оскільки один і той же ітераційний алгоритм може бути використаний для всіх цих додатків, що використовують основні операції shift-add. Алгоритм CORDIC може бути застосований на практиці у двох режимах (напр. обертання і векторизація) і трьох типах (напр. лінійний, круговий і гіперболічний режим). Цей алгоритм дуже привабливий для апаратної реалізації, оскільки він використовує тільки елементарні операції зсуву і складання для виконання обертання вектора. Він потребує тільки використання 2-х перемикаючих і 3-х суматорних модулів, тому його розсіювана потужність дуже мала, і він також дуже компактний. Він часто використовується в масиві обробних елементів на мікросхемах ПЛІС [3, с. 59].

Алгоритми цифрової обробки сигналів (DSP) демонструють зростаючу потребу в ефективній реалізації складних арифметичних операцій. Обчислення тригонометричних функцій, перетворення координат або обертання комплексних значень пов’язане з сучасними алгоритмами DSP. Популярні приклади застосування – алгоритми, які використовуються в цифровій техніці зв’язку і в адаптивній обробці сигналів. У той час як в цифровому зв’язку

важливою є пряма оцінка зазначених функцій, численні алгоритми адаптивної обробки сигналів на основі матриць, що вимагають вирішення систем лінійних рівнянь, QR-факторизації або обчислення власних значень, власних векторів або сингулярних значень [6, с. 75]. Всі ці завдання можуть бути ефективно реалізовані за допомогою елементів обробки, де виконують векторне обертання. Алгоритм координатного обертання цифрової обчислювальної машини (CORDIC) дає можливість обчислити всі потрібні функції досить простим методом.

Алгоритм CORDIC може бути закодований як в мікропрограмному забезпеченні, так і в невеликих мікроконтролерах. При незначних змінах початкових умов і таблиць даних основний алгоритм може множитися, розподілятися і обчислювати квадратні корені, гіперболічні функції, експоненти і логарифми. Саме універсальність і простота роблять алгоритм CORDIC кращою реалізацією математичних функцій на невеликих ручних калькуляторах. Він обчислює тригонометричні функції синуса, косинуса, величини і фази (арктангенс) з довільною точністю. Він також може обчислювати гіперболічні функції. CORDIC був реалізований в кишенькових калькуляторах, таких як HP 35 компанії Hewlett Packard, і в арифметичних співпроцесорах, таких як Intel 8087. Деякі автори пропонували використовувати CORDIC процесори для обробки сигналів таких додатків, як фільтрація, обробка зображень або вирішення лінійних систем. Алгоритм CORDIC знайшов своє широке застосування при обчисленні перетворення Фур'є. Сьогодні алгоритм CORDIC використовується в проектуванні нейронних мереж, високопродуктивних додатках DSP з векторним обертанням, розширеному схемотехнічному проектуванні, оптимізованому малопотужному дизайні.

Спочатку алгоритм CORDIC являв собою ітераційну процедуру для виконання операції «поворот» векторів на площині на довільний кут, використовуючи тільки операції зсуву і складання. Як відомо, дане перетворення визначається наступними співвідношеннями:

$$x = x_0 \cos\phi - y_0 \sin\phi$$
$$y = x_0 \sin\phi + y_0 \cos\phi \quad (1)$$

Переписуючи вищенаведені рівності у вигляді:

$$x = \cos\phi \cdot (x_0 - y_0 tg\phi)$$
$$y = \cos\phi \cdot (y_0 + x_0 tg\phi) \quad (2)$$

і вибираючи такий кут повороту, що / , легко помітити, що множення на тангенс в формулах (2) являє собою всього лише операцію зсуву на і розрядів в двійковій системі числення. Отже, якщо ми представимо деякий довільний кут у вигляді суми кутів / , i = 0,1,2... n, то операція повороту може бути розкладена в серію послідовно виконаних елементарних поворотів. Відзначимо, що величина множника $\cos(\alpha_i) = \cos(-\alpha_i)$ не залежить від напрямку повороту і є константою на кожному кроці. Таким чином, приходимо до наступних ітераційних формул для i-го кроку (i=0,1... n):

$$x_{i+1} = K_i \cdot (x_i - \sigma_i y_i 2^{-i})$$
$$y_{i+1} = K_i \cdot (y_i + \sigma_i x_i 2^{-i}) \quad (3)$$

Тут $K_i = \cos(arctg(2^{-i})) = \frac{1}{\sqrt{1+\frac{1}{tg^2(arctg(2^{-i}))}}} = \frac{1}{\sqrt{1+2^{-2i}}}$ коефіцієнт деформації вектора на і-му кроці $\sigma_i \in \{-1, +1\}$ - оператор, що визначає напрямок повороту (за годинниковою стрілкою або проти годинникової стрілки). Виключаючи величину Ki з (3), отримаємо ітераційний алгоритм для операції «поворот», що використовує тільки операції додавання і зсуву [4, с. 37].

$$x_{i+1} = x_i - \sigma_i y_i 2^{-i}$$
$$y_{i+1} = y_i + \sigma_i x_i 2^{-i}$$

Добуток коефіцієнтів деформації для всіх кроків є сходним і дає сумарне розтягнення вихідного вектора в $K_n = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} \sqrt{1+2^{-2i}} \approx 1.647$ раз

Отже, для того щоб провести поворот вектора на деякий кут, необхідно представити даний кут у вигляді послідовності керуючих операторів $a_i$. Для знаходження даної послідовності може бути використана як таблична реалізація, так і спосіб, коли оператори визначаються, виходячи з рекурентних

співвідношень. Метод визначення залежить від необхідного результату і наведено нижче [15, с. 92].

Алгоритм CORDIC може бути використаний в двох режимах – «поворот» і «вектор». У першому здійснюється довільний поворот на площині, у другому вектор стягується до однієї з координатних осей, що використовується для знаходження довжини вектора. У режимі операції «поворот» керуючі оператори вибираються за формулами:

$$\sigma_i = sign(z_i) \qquad i = \overline{0, n-1}$$
$$z_{i+1} = z_i - \sigma_i \alpha_i \qquad z_0 = \phi, \quad \alpha_i = arctg(2^{-i})$$

Таким чином, величина / . Другий варіант-вибір керуючої послідовності таким чином, що / . В результаті вектор стягується до осі x. Ця операція називається операцією «вектор» і визначається наступними співвідношеннями:

$$\sigma_i = -sign(y_i)$$
$$z_{i+1} = z_i - \sigma_i \alpha_i$$
$$i = \overline{0, n-1}, \quad z_0 = 0, \quad \alpha_i = arctg(2^{-i})$$

Після виконання n ітерацій отримаємо:

$$x_n \approx K_n \sqrt{x_0^2 + y_0^2}, \quad y_n \approx 0, \quad z_n \approx arctg(\frac{y_0}{x_0}).$$

Тобто одночасно можна обчислити як довжину вектора, так і його кут щодо осі x. Величина деформації $K_n$ – та ж сама, що і для операції «поворот». Таким чином, варіюючи початкові значення і використовуючи різні правила для визначення значень керуючих операторів, можна обчислювати основні тригонометричні функції, а також здійснювати перетворення координат.

Елементарні функції унікальні тим, що вони не можуть бути точно обчислені в кінцевому числі арифметичних операцій – їх точне уявлення вимагає використання нескінченного ряду алгебраїчних членів. Однак вони можуть бути апроксимовані з необхідною точністю за допомогою кінцевого числа операцій. Вони були використані в таких різноманітних додатках, як робототехніка, тривимірна комп'ютерна графіка, декомпозиція, цифрова обробка сигналів. Обчислення тригонометричної функції – це трудомістка операція. Насправді з

усіх арифметичних операцій, які повинен виконувати чіп, тригонометричні функції мають найгіршу затримку. Вони вимагають багато циклів для оцінки, так що інструкції, що залежать від їх оцінки, повинні зупинятися до тих пір, поки результат не стане доступним. Крім того, такі ресурси, як суматори, перемикачі та множники, пов'язані та недоступні для використання навіть іншими незалежними інструкціями, що призводить до зупинок через структурні небезпеки [22, с. 47].

Тому вкрай важливо, щоб ці елементарні функції були обчислені якомога швидше, щоб уникнути погіршення продуктивності. Результати повинні бути отримані з високою точністю для будь-якого з кутів в межах принципової області елементарної функції. Такі методи, як зменшення діапазону, корисні при зіставленні аргументу з його основною областю, але навіть в цьому випадку алгоритм повинен бути досить гнучким, щоб забезпечити точну відповідь з будь-якою вхідною точкою з його області.

Споживання енергії стало важливим показником в сучасному електронному дизайні, особливо коли гаджети і обчислювальні пристрої зменшуються в розмірах. Тепло, яке розсіюється при споживанні енергії в обчислювальному чіпі, ускладнює охолодження чіпа. Цей чіп повинен або працювати на зниженому рівні продуктивності, щоб запобігти його згорянню, або використовувати дорогі і громіздкі охолоджуючі механізми, такі як радіатори або повітряні потоки, щоб підтримувати температуру на прийнятному рівні. Коли енергоємні обчислювальні елементи використовуються в споживчих пристроях, таких як цифрові камери або MP3-плеєри, вони швидко розряджають батарею, що призводить до поганого досвіду для користувача. У будь-якому випадку, надмірне споживання енергії обмежує продуктивність арифметичної мікросхеми.

Існує п'ять основних способів, за допомогою яких елементарна функція може бути обчислена в апаратному забезпеченні-за допомогою пошуку таблиць, поліноміальної апроксимації, раціональної апроксимації і квадратичних методів

збіжності. Алгоритм CORDIC є найбільш універсальним з усіх алгоритмів, які можуть бути використані для оцінки елементарних функцій. Те ж саме обладнання може бути використано для обчислення тригонометричних співвідношень (sin, cos, tan і т.д.), гіперболічні відносини (sin h, cos h, tgh), множення, ділення, зворотні тригонометричні (arcsin, arccos) і зворотні гіперболічні відносини (arcsin h, arccos h). З невеликою модифікацією він також може обчислювати логарифми, експоненти тощо.

Алгоритм CORDIC використовується для оцінки обчислення елементарних функцій в реальному часі з використанням ітераційного обертання вхідного вектору. Обертання заданого вектора здійснюється за допомогою послідовності обертань з фіксованими кутами, які призводять до загального обертання через заданий кут або призводять до кінцевого кутового аргументу, що дорівнює нулю. Однак основним недоліком алгоритму CORDIC є його низька обчислювальна швидкість. Для ітеративної структури CORDIC, продуктивність швидкості роботи CORDIC обмежується великою кількістю ітерацій N, яка, як правило, дорівнює довжині внутрішнього складу. На алгоритмічному рівні одним з тривіальних рішень для подолання такої проблеми є пряме зменшення кількості ітерацій [29, с. 218].

Для цього вченими було введено ряд методів для зменшення кількості ітерацій CORDIC з метою підвищення його продуктивності. Он-лайновий CORDIC був розроблений науковцями Ercegovac і Lang для додатків, в яких вхідні біти стали доступні серійно. Он-лайновий метод CORDIC замінює змінні більш ефективними за площею затримками. Їх метод також може компенсувати значення K в режимі он-лайн. Дюпра і Мюллер досліджуючи алгоритм CORDIC вибирають напрямок подальших досліджень на скорочення часу циклу CORDIC ітерації за допомогою швидких суматорів, заснованих на використанні надлишкової арифметики для вираження операндів.

Для додатків, що вимагають тільки прямого обертання (або векторного обертання), метод перекодування кута забезпечує алгоритм CORDIC. Цифровий

обчислювальний алгоритм обертання координат дає можливість розрахувати всі необхідні функції простим способом. Враховуючи вимоги та обмеження різних прикладних середовищ, розробка алгоритму та архітектури CORDIC була проведена для досягнення високої пропускної здатності та зниження апаратної складності, а також затримки реалізації. Затримка реалізації є недоліком звичайного алгоритму CORDIC. Для реалізації зниженої латентності були розроблені схеми кутового перекодування, змішаного обертання і більш високого радіуса корду. Для високопродуктивних обчислень були запропоновані паралельні та конвеєрні CORDIC.

У звичайному алгоритмі CORDIC хід розрахунку та таблиця пошуку необхідні для досягнення обчислення кількох трансцендентних функцій, що призведе до ускладнення апаратної схеми і зниження швидкості роботи. Щоб подолати ці недоліки традиційного алгоритму CORDIC, Xin запропонував модифікований алгоритм CORDIC. Цей метод не потребує модуля поправочного коефіцієнту і таблиці підстановки, а просто потребує простого зсуву і додавання-віднімання для досягнення обчислення множинної трансцендентальної функції. Таким чином, він може знизити витрати на апаратне забезпечення і підвищити експлуатаційну продуктивність [37, с. 109].

Для архітектури апаратної обробки сигналів існує безліч апаратно ефективних алгоритмів. Серед цих алгоритмів є набір алгоритмів зсуву-додавання, спільно відомих як CORDIC (Cordinate Rotation for Digital Computers) для обчислення широкого спектру функцій, включаючи певні тригонометричні, гіперболічні, лінійні та логарифмічні функції. Синіт порівняв різні архітектури CORDIC з точки зору їх площі, швидкості і продуктивності пропускної здатності даних, особливо в трьох різних основних стилях ітераційних, паралельних і конвеєрних структурах.

Харе представив ефективний за площею і часом CORDIC алгоритм, який повністю виключає масштабний фактор. При відповідному виборі порядку апроксимації ряду Тейлора запропонована кордна схема задовольняє вимогам

точності і досягає бажаного діапазону збіжності. Крім того, він запропонував алгоритм перевизначення елементарних кутів для зменшення кількості кордових ітерацій. Пропонований алгоритм CORDIC забезпечує гнучкість управління кількістю ітерацій в залежності від вимог до точності, площі і затримки. Компенсація масштабного фактора, що є властивий алгоритму CORDIC, стає важливим недоліком при спробі покращити його застосовність, хоча деякі автори придумали нову версію без масштабування, яка була успішно реалізована в бездротових додатках.

Хоча апаратне забезпечення CORDIC дуже просте, що складається тільки з 2-х перемикачів і 3-х суматорів, воно здатне оцінювати широкий спектр елементарних функцій і, отже, знайшло своє застосування в багатьох різних інженерних додатках. Попередня робота дослідників в цій області була зосереджена на вдосконаленні різних аспектів алгоритму в залежності від характеристик програми, для якого він призначався.

Деякі методи були зосереджені на зменшенні кількості апаратного забезпечення, необхідного для CORDIC. Одним із способів зниження апаратної складності є об’єднання ітерацій сполучення, що призводить до необхідності використання менших значень. Гібридний алгоритм CORDIC зменшує обсяг простору, необхідного для апроксимації постійного кута або найбільш постійного кута для останніх двох третин кордових ітерацій.

Кілька методів були зосереджені на проблемі ефективної компенсації масштабного фактору. Масштабний коефіцієнт можна компенсувати паралельно, поки виконуються CORDIC ітерації. Інший метод полягає у виконанні додаткових ітерацій масштабування, які призводять загальний коефіцієнт масштабування до одиниці. Інший метод полягає в повторенні деяких кордових ітерацій, щоб зробити K потужністю машинного радіусу, вимагаючи лише простої операції зсуву в кінці, щоб отримати масштабовані результати. Деякі з них намагалися використовувати систему числення з високим радіусом для виконання обчислень. У цьому випадку потрібно менше ітерацій для

досягнення заданої точності за рахунок більш складної функції вибору, а також витрат на перетворення [33, с. 237].

Control CORDIC використовує методи демпфування з теорії управління, щоб зменшити кількість ітерацій приблизно на 11% в динамічних ситуаціях. Метод перекодування кута може досягти 50% або більше зменшення кількості ітерацій, але він обмежений статичними додатками, такими як перетворення chirp-Z, де кут повороту статичний і відомий апріорі. У таких випадках кутові константи, які можна пропустити, можуть бути заздалегідь обчислені в автономному режимі.

Метод CORDIC використовує підхід «один біт за один раз» для виконання обчислень з довільною точністю. Як правило, ці таблиці містять тільки один-два записи на Біт точності. Алгоритм CORDIC також використовує тільки правильні зрушення і доповнення, мінімізуючи час обчислень. Це апаратно ефективний алгоритм, оскільки в CORDIC немає мультиплікатора, щоб зберегти гейт, необхідний для реалізації на ПЛІС. Якщо мультиплікатор присутній, то тривалість розрахунку збільшується.

Алгоритм CORDIC став широко використовуваним підходом до оцінки елементарних функцій, де існує область обмежень. Апаратний мультиплікатор в CORDIC недоступний, оскільки сильною стороною CORDIC-алгоритму є його здатність вирішувати операції з обертанням вектора без використання мультиплікатора і прискорювати CORDIC-алгоритм. Єдина операція – це додавання, віднімання, зсув бітів і таблиця підстановки. Повернений вектор також масштабується, що робить необхідним масштабний коефіцієнт. Завдяки високій швидкості, низькій вартості і більшій гнучкості, що забезпечується ПЛІС в порівнянні з DSP-процесорами, обчислення на основі ПЛІС стають основними для всіх цифрових систем обробки сигналів сучасної епохи.

Phatak запропонував виконувати дві ітерації в одному циклі, використовуючи подвійні кордові блоки. В іншому методі арифметики, що використовує алгоритм CORDIC дозволяється використовувати швидкі

надлишкові суматори, щоб скоротити час циклу. Існує значний обсяг робіт, присвячених вирішенню таким завданням, як виявлення знаків, пов'язаних з використанням надлишкової арифметики. На відміну від цього, було проведено дуже мало досліджень з вивчення зниження латентності шляхом пропуску деяких ітерацій. Це відбувається головним чином через супутні незручності змінного коефіцієнту масштабування і необхідності зберігати їх в ПЗУ. Однак, оскільки в сучасних чіпах немає дефіциту в доступності місця, простір ROM набагато доступніший для використання, ніж раніше. Відповідно, має сенс досліджувати методи, які можуть мати змінну K, перестрибуючи через ітерації, якщо при цьому вони значно зменшують кількість ітерацій [28, с. 95].

2009 рік ознаменував собою завершення 50-річчя винаходу алгоритму CORDIC. Завдяки швидкому прогресу в області ПЛІС технології проклали шлях до абсолютно іншого підходу до комп'ютерного проектування і їх застосування в реальному часі, використовуючи архітектуру спеціального призначення з призначеними для користувача чіпами. Сьогодні весь світ обміну інформацією обертається навколо передачі і перегляду зображень в реальному часі. Багато додатків DSP намагаються тісно стимулювати реальні способи життя. Швидкість, чіткість і схожість з об'єктом реального часу – ось деякі з питань, які необхідно вирішити для досягнення мети goel.

Метод перекодування кута, що був запропонований Наганатаном використовував алгоритм для пропуску деяких кутів повороту з метою зменшення кількості необхідних ітерацій. Максимальна кількість ітерацій, що необхідна для цього методу, дорівнює N / 2, при цьому середнє значення становить приблизно N / 3 ітерації.

Родрігес представлена простіша реалізація схеми вибору кута, яка не вимагає збільшення часу циклу, що дозволяє динамічно використовувати метод перекодування кута для довільних кутів. Цей метод також має ту перевагу, що всі кутові константи знаходяться паралельно, на одному кроці, перевіряючи тільки початковий кут повороту, без необхідності виконувати послідовні

кордичні ітерації. Цей метод динамічного перекодування кутів може бути сформульований для використання «секції», щоб обмежити кількість необхідних компараторів дальності до розумного значення. Існує збільшення кількості необхідних адаптивних кордичних ітерацій, але ця проблема може бути пом'якшена за допомогою буфера в поєднанні з методом перетину.

Архітектура і реалізація складного процесора швидкого перетворення Фур'є (CFFT) з використанням технології 0,6 мкм арсеніду галію (GaAs) були представлені компанією Sarmiento. Цей процесор обчислює 1024-точковий БПФ з 16-бітних складних даних менш ніж за 8 мкс, працюючи на частоті понад 700 МГ. Архітектура процесора заснована на алгоритмі цифрової обчислювальної машини обертання координат (CORDIC), який уникає використання звичайних одиниць множення і накопичення (MAC), але оцінює тригонометричні функції, використовуючи тільки операції додавання і зсуву. Покращення в базовій архітектурі CORDIC вводяться для того, щоб зменшити площу і потужність процесора. Це разом з використанням конвеєрних і переносних суматорів дозволяє отримати дуже звичайний і швидкий процесор.

Гаррідо запропонував новий алгоритм без пам'яті CORDIC для обчислення БПФ. Цей підхід обчислює напрямок мікрооборотів від керуючого лічильника БПФ, тому площа ротатора практично не залежить від кількості оборотів, що особливо підходить для обчислення БПФ з великою кількістю точок. Крім того, новий CORDIC має й інші переваги, такі як спрощення базового процесора CORDIC, що використовується для розрахунку мікрооборотів, або простий спосіб компенсації внутрішнього посилення алгоритму CORDIC [10, с. 134].

Режими роботи алгоритму CORDIC. Даний алгоритм працює в двох режимах: обертання і векторизація (рис. 1.1.).

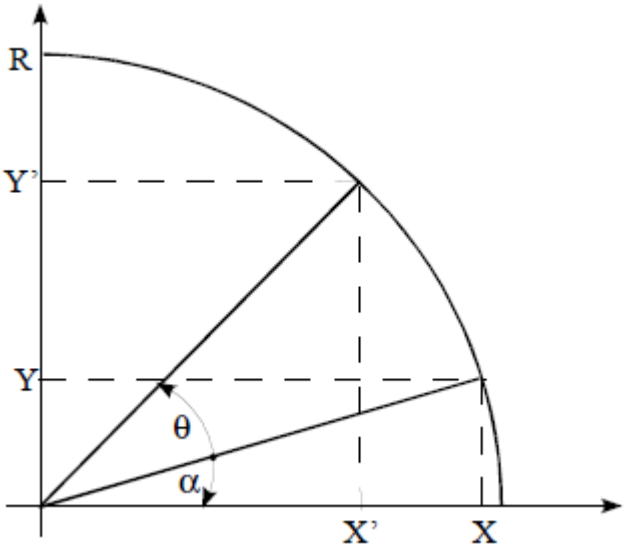



*Рис. 1.1. Обертання (X', Y') і векторизація (X, Y) режими алгоритму CORDIC*

У першому режимі він повертає вхідний вектор на заданий кут. Поворот кута ініціалізується на певний заданий кут. Для цього режиму існує таке CORDIC рівняння:

$$\begin{cases} x_{i+1} = x_i - y_i * d_i * 2^{-i} \\ y_{i+1} = y_i + x_i * d_i * 2^{-i} \\ z_{i+1} = z_i - d_i * \tan^{-1}(2^{-i}), \end{cases} \text{де} \quad d_i = \begin{cases} -1, \\ 1, \ z_i < 0 \quad z_i < 0 \end{cases}$$

Це дає наступний результат:

$$x_n = A_n \cdot [x_0 \cos z_0 - y_0 \sin z_0]$$
$$y_n = A_n \cdot [y_0 \cos z_0 - x_0 \sin z_0]$$
$$z_n = 0, \qquad \text{де} \quad A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

У режимі векторизації вхідний вектор повертається до осі x при записі кута, необхідного для цього повороту. Результатом операції векторизації є кут повороту і масштабована величина вихідного вектора. У CORDIC рівнянь в цьому режимі:

$$\begin{cases} x_{i+1} = x_i - y_i * d_i * 2^{-i} \\ y_{i+1} = y_i + x_i * d_i * 2^{-i} \\ z_{i+1} = z_i - d_i * \tan^{-1}(2^{-i}), \end{cases} \text{де} \quad d_i = \begin{cases} -1, \\ 1, \ y_i < 0 \end{cases}$$

Потім:

$$x_n = A_n\sqrt{x_0^2 + y_0^2} \qquad z_n = z_0 + \tan^{-1}\left(y_0/x_0\right)$$
$$y_n = 0$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

Алгоритм CORDIC в кожному режимі обмежений від $-\pi/2$ і / . Це обмеження викликано першим кутом повороту / .

На рис. 1.2. представлена структурна схема звичайного алгоритму CORDIC [12, с. 168], заснованого на суматорах перенесення пульсацій.

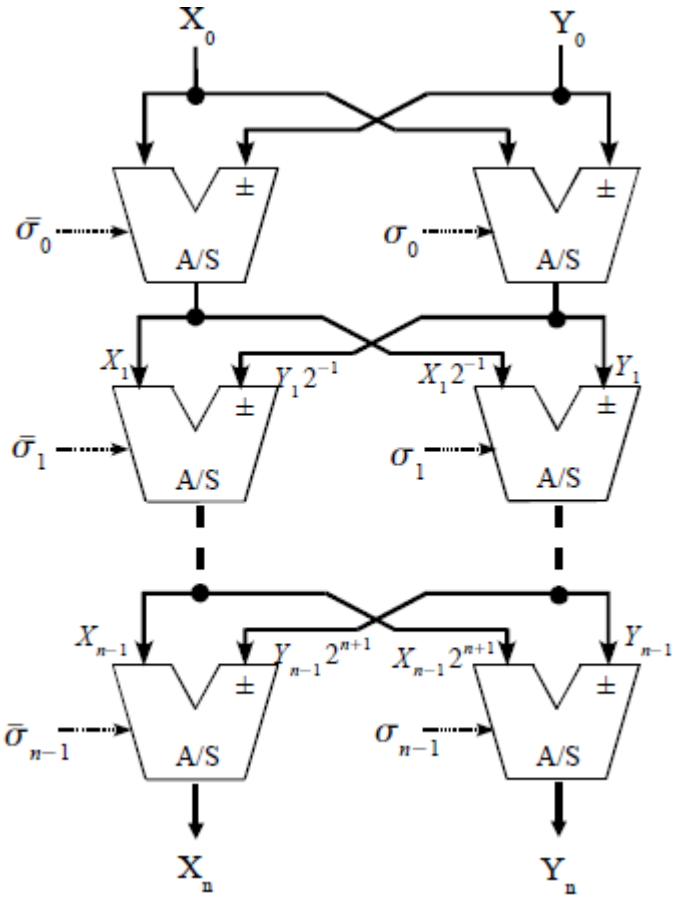



*Рис. 1.2. Структурна схема алгоритму CORDIC*

Суматор (A / S), в залежності від вхідного сигналу вибору, виконує додавання або віднімання. Цей вхідний сигнал вказує, чи є операнд негативним. Базова комірка A / S декомпозується двома функціями з 4 бітами введення кожна. Один з них призначений для розрахунку вихідного сигналу, а інший-для передачі перенесення. Відповідно до цього N-бітний A / S може помістится в (2N+1) / 2 CLBs (настроюється логічний блок). Додаткова половина CLB потрібна для

введення найменш значущого біта (LSB) у разі віднімання. Критичний шлях тут позначений поширенням перенесення пульсації і затримкою маршрутизації проводу A / S. У цьому випадку ця мережа має розгалуження 2N. це знижує продуктивність схеми і є головним недоліком звичайних кордових реалізацій. Як вирішення цієї проблеми можна було б використовувати надлишкову арифметику для збільшення швидкості CORDIC. Реалізація дозволяє уникнути поширення перенесення від LSB до найбільш значущого біту (MSB) через його властивості carry-free [11, с. 16].

Надлишкова арифметика хороша для прискорення тих операцій, які мають велику затримку поширення. З іншого боку, надмірна арифметика також має деякі недоліки. Наприклад, неможливо виявити знак надлишкового числа без перевірки всіх цифр, які очікують поширення від MSB до LSB. Інша проблема полягає в тому, що надлишкова арифметика використовує набір цифр {-1,0,1} і вимагає більше апаратних ресурсів для виконання простих завдань, ніж звичайна арифметика, що використовує набір цифр {-1,1}.

Згідно з результатами дослідження, надлишкова арифметика є більш точною, але вона вимагає набагато більшого апаратного забезпечення, ніж звичайна арифметика, і тому в даному дослідженні будек використана саме вона.

## 1.2. Основні підходи до класифікації CORDIC-алгоритмів

Для того щоб охопити модифікації CORDIC алгоритмів, нижче запропоновані дві класифікаційні схеми – за особливостями побудови та областям застосування із зазначенням відповідних робіт. З точки зору реалізації модифікації CORDIC-алгоритмів можуть бути класифіковані наступним чином:

– на підставі числення операндів:
– алгоритми з основою 4;
– алгоритми з великою основою (до 1024) [7, с. 202];
– за використовуваними системами представлення чисел:
– надлишкова, у вигляді цифр зі знаком (signed digit);
– надлишкова, зі збереженням переносу (carry save);
– за методом корекції результату:
– множення на величину, зворотну коефіцієнту деформації, після закінчення ітерацій;
– додавання спеціальних масштабуючих ітерацій;
– паралельне обчислення коригованого результату;
– за технікою прискорення обчислень:
– конвеєризація обчислень;
– використання попередньої декомпозиції кута повороту;
– алгоритми з передбаченням значень операторів;
– використання оцінки знака по малому числу старших бітів;
– алгоритм з розгалуженням.

Можна відзначити, що незважаючи на багатство існуючих методик та технік, запропонованих для прискорення алгоритму, майже всі вони зберігають його ітераційний характер. Прискорення досягається за рахунок зменшення загальної кількості ітерацій або необхідної кількості апаратури. Винятком є, мабуть, тільки техніка з передбаченням значень операторів, однак, отриманий алгоритм все одно не є повністю паралельним [1, с. 37].

Можна виділити такі основні області застосування CORDIC-алгоритмів:

– різні форми перетворення Фур’є: БПФ, ДПФ, дискретні синус – і косинус-перетворення;
– алгоритми лінійної алгебри: перетворення Хаусхолдера (Householder transformations), алгоритм розкладання матриці за сингулярними значеннями (SVD), узагальнення повороту вектора на багатовимірний випадок;
– алгоритми цифрової обробки сигналів: алгоритми цифрової фільтрації і алгоритми обробки зображень – перетворення Хоу (Hough transform).

### 1.3. Процесорні елементи з набором великих операцій

У науковій літературі існує дослідження, в якому наведена схема обчислювача, заснованого на застосуванні CORDIC IP-ядра, яке налаштовується і підтримує кілька функцій, в тому числі «поворот», обчислення cos^), arctan^) та інші. У пристрої підтримується паралельний режим, в якому вихідні дані обробляються в одному циклі тактової частоти, і послідовний, в якому вихідні дані обчислюються за кілька тактів. IP-ядро підтримує змінну точність і декількох варіантів алгоритмів округлення. Структурна схема пристрою представлена на рис. 1.3.

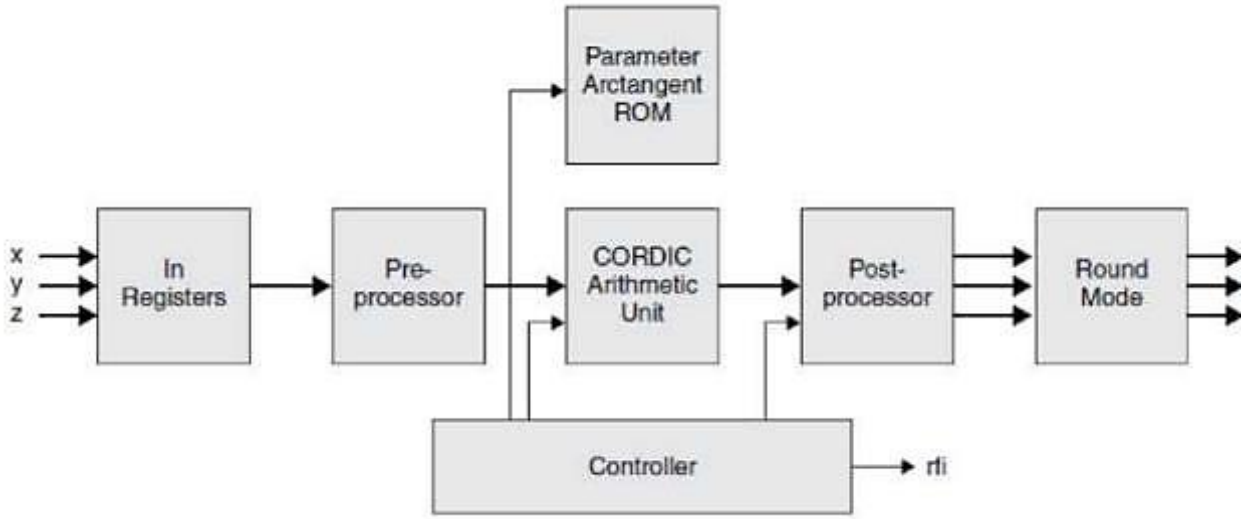



*Рис. 1.3. Блок-схема ядра розрахунку (CORDIC IP Core)*

У роботі [5, с. 53] пропонується конвеєрна архітектура елементу, призначеного для обчислення тригонометричних і гіперболічних функцій алгоритмами CORDIC (рис. 1.4.).

Елемент характеризується підтримкою:

– генерації описів елементів з конвеєрною і паралельною архітектурою;
– застосування в складі формованих елементів блоку «грубого» повороту вектора і розширення діапазону зміни аргументу обчислюваних тригонометричних функцій;
– корекції результату при виконанні операцій повороту і перетворення координат;

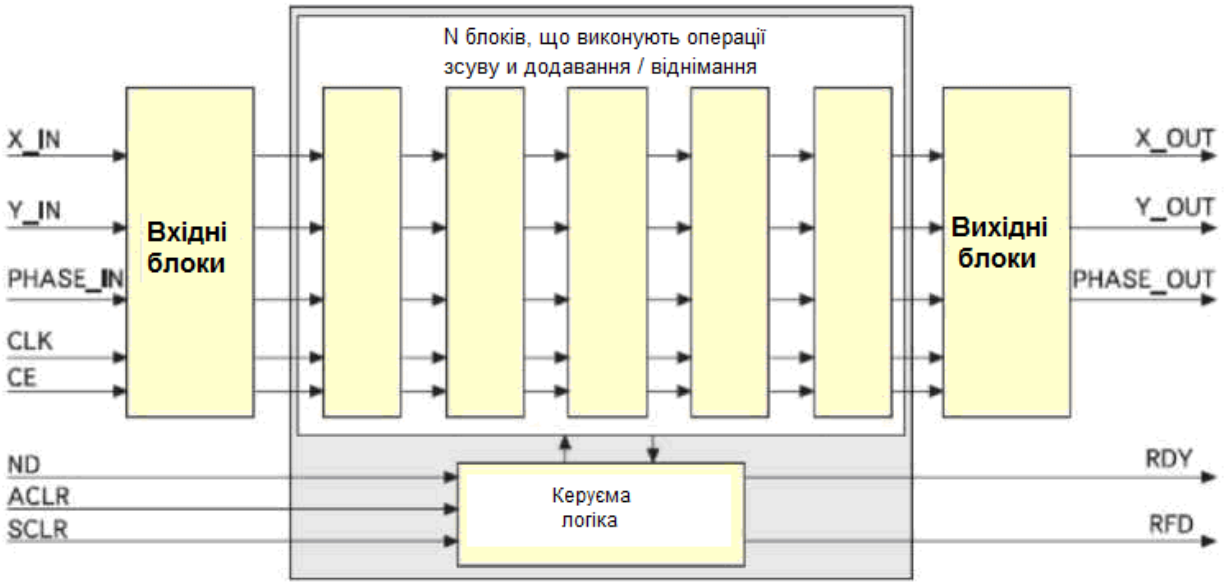



*Рис. 1.4. Структура конвеєрного елемента для обчислення функцій CORDIC*

– технології Xilinx Smart-IP, що забезпечує найкращу реалізацію генерованого елемента в обраному кристалі.

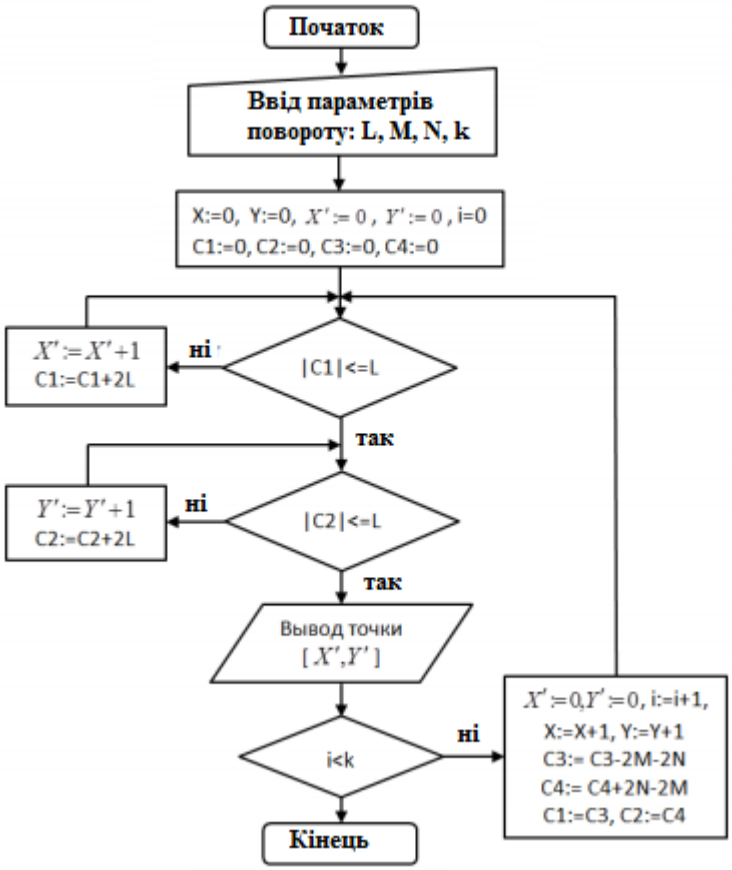



*Рис. 1.5. Алгоритм повороту вектору CORDIC*

На основі розроблених розрядно-паралельних схем запропонована структура спеціалізованого геометричного процесорного елементу (ГПЕ), операційна частина якого представлена на рис. 1.6.

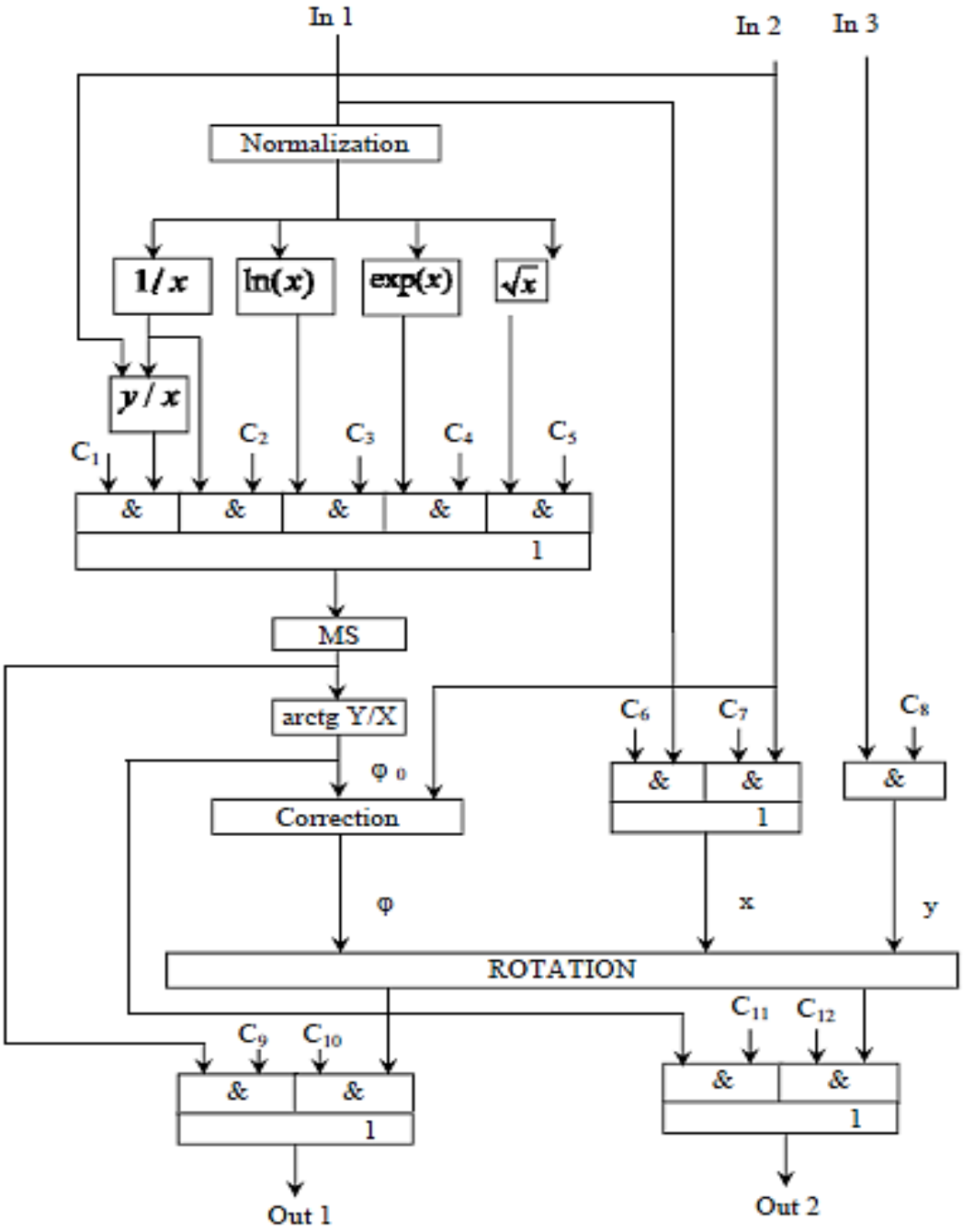



*Рис. 1.6. Операційна частина*

Основний упор зроблений на таблично-алгоритмічних обчисленнях, причому значна роль відводиться ПЗУ, в яких зберігається вся попередньо підготовлена інформація. Алгоритми досліджуваних типологій виглядають більш ефективними і компактними для операцій типу 1 / x і-Jx, в той час як

алгоритми CORDIC добре застосовні для виконання складових геометричних операцій, тому доцільно об’єднати їх кращі сторони є єдиний алгоритм. Кожна операція реалізується в пристрої шляхом відповідного налаштування його структури [8, с. 50]. Програмування структури здійснюється відповідною комутацією (призначенням) окремих складових ГПЕ за допомогою керуючих сигналів Q, C2,..., C12, які формуються після розшифровки коду команди пристроєм управління. Використання розрядно-паралельних алгоритмів дозволяє будувати ГПЕ на єдиній методологічній основі із застосуванням простого і зручного математичного апарату групового підсумовування і однорідної елементної бази FPGA [36, с. 102].

Обмежена система команд ГПЕ представлена в табл. 1.1.

*Таблиця 1.1*

**Система команд геометричного процесорного елементу**

| Операціїї | Код | Вх. 1 | Вх. 2 | Вх. 3 | Вих. 1 | Вих. 2 |
|---|---|---|---|---|---|---|
| «rotation» | 0000 | x | У | P | x' = x cos (p) – У sin (p) | У' = y cos (p) – x sin (p) |
| «vectoring» | 0001 | x | У | | 1 2 2 / x + У | p = arctg (y / x) |
| У / x | 0010 | x | У | | У / x | p = arctg (y / x) |
| (У / x) b | 0011 | x | У | b | (У / x) b | p = arctg (y / x) |
| sin (p), cos (p) | 0100 | 1 | 0 | P | cos (p) | sin (p) |
| ln (x) | 0101 | x | | | ln (x) (порядок) | ln (x) (мантіиса) |
| 4x | 0110 | x | | | 4x (порядок) | 4x (мантиса) |
| exp (x) | 0111 | x | | | exp (x) (порядок) | exp (x) (мантиса) |

Таблиця 1.1. передбачає подальше розширення за рахунок включення нових команд. ГПЕ апаратно підтримує структурне програмування алгоритмів машинної графіки. Вхідними і вихідними параметрами команд ГПЕ служать координати, які, наприклад, задають положення точок зображення на екрані дисплея, в робочому полі графобудівника або в інших системах відображення.

# РОЗДІЛ 2
# АНАЛІЗ МОЖЛИВОСТЕЙ МОВИ VHDL

## 2.1. Способи HDL-опису простих вузлів

HDL використовується не тільки для представлення проектованих схем, але і для опису тестуючих програм (testbench) і тестів. Маючи в своєму розпорядженні виконані з урахуванням вимог багаторазового використання HDL-опис раніше спроектованих пристроїв, за допомогою САПР нескладно включити ці описи до складу нових проектів, повторно реалізувати їх на більш сучасній технології і т.д.

Знаючи HDL-опис пристрою або тестуючої програми можна здійснити модернізацію схем та використати HDL-моделі при пошуку несправностей в схемі. За допомогою HDL-додатків більш ефективно вирішуються завдання тренінгу в сфері проектування та експлуатації радіоелектронної апаратури. Стандартизація вхідних мов і внутрішніх інтерфейсів підсистем САПР, в тому числі і на базі HDL, створює загальне комунікаційне середовище для САПР, дозволяє спростити стикування продуктів різних фірм, обмін бібліотеками моделей компонент і проектів, модернізацію окремих підсистем САПР [13, с. 17].

У HDL-описі, як і в будь-якій моделі, відображаються тільки деякі аспекти (характеристики) реальної системи. Цифрову апаратуру характеризують, наприклад, такі аспекти, як функціональний (реалізована функція, алгоритм); тимчасовий (затримки, продуктивність, час відгуку); структурний (типи і зв’язку компонент); ресурсний (число вентилів, площа кристала); надійнісний (час напрацювання на відмову); конструктивний (ВАГА, Габарити); вартісний і т.д. [35, с. 74]

HDL містить засоби, що дозволяють відобразити в основному перші три аспекти: функціональний, часовий і структурний. Як уже зазначалося, функція (поведінка) апаратури може деталізуватися від рівня системи команд і алгоритмів пристроїв до булівських функцій; структура – від рівня пристроїв

типу процесор – пам’ять до рівня вентилів і перемикаючих елементів; час – від затримок фронтів сигналів (нано – ns і фемтосекунди – fs) до тактів і затримок електромеханічних пристроїв (секунди і години).

Ступінь деталізації аспектів, що відображаються в описах апаратури, визначається конкретними завданнями. Наприклад, опис деякої мікропроцесорної системи може будуватися як описі структури, що складається з мікросхем БІС і СІС, а опис самих мікросхем будується поведінково, якщо їх опис на вентильному рівні або відсутній, або занадто громіздкий.

В HDL вбудовано ряд понять, які зазвичай використовуються проектувальниками апаратури. Засоби HDL для відображення структур цифрових систем базуються на уявленні про те, що описуваний об’єкт проекту (entity, module) являє собою структуру з більш простих об’єктів-компонентів (component), що з’єднуються один з одним лініями зв’язку (проводами – wire (VERILOG) або сигналами – signal (VHDL). Кожен компонент, у свою чергу, є об’єктом і може складатися з компонентів нижчого рівня (ієрархія об’єктів). Взаємодіють об’єкти шляхом передачі сигналів (signal) по лініях зв’язку. Лінії зв’язку (wire – в VERILOG) або ототожнюються з ними сигнали (signal – в VHDL) підключаються до вхідних (in, input) і вихідних (out, output) портів (port) зв’язуваних компонентів.

Коли ядро має вихід на стандартний інтерфейс, при його тестуванні за схемою на рис. 2.1 виникає задача опису на мові VHDL послідовності обміну даними цією шиною з врахуванням усіх її сигналів і часових параметрів. Одним з варіантів рішення цієї задачі є введення до програмного стенду моделі мікропроцесора (мікроконтролера), який керує шиною (рис. 2.2).

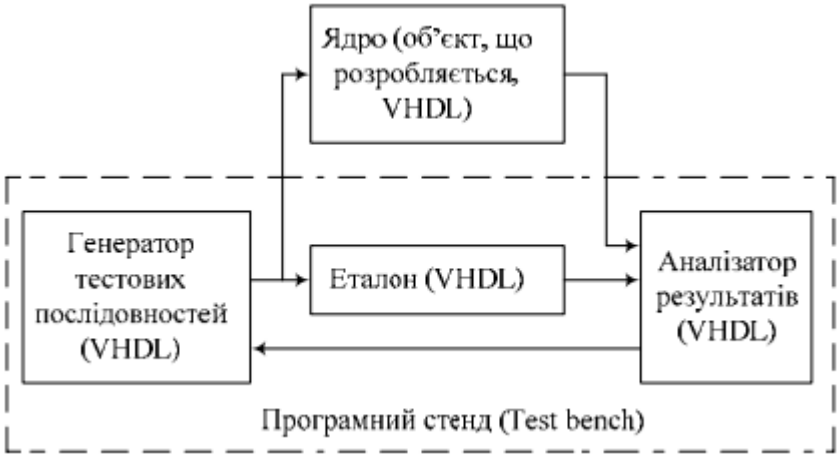



*Рис. 2.1. Схематичне зображення процесу тестування ядра із стандартним інтерфейсом*

За такою схемою мікроконтролер бере на себе функції генерації тестових послідовностей і аналізу результатів роботи тестованого ядра. При цьому не треба описувати послідовності обміну даними шиною, але виникає задача створення моделі (ядра) мікроконтролера (яку, у свою чергу, спочатку також треба буде протестувати) [17, с. 197].

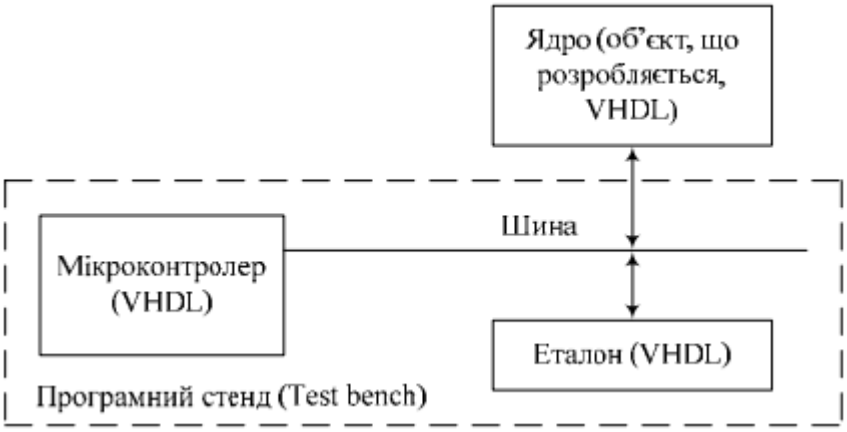



*Рис. 2.2. Програмний стенд з мікроконтролером*

Також приймається до уваги існування великої кількості тестових послідовностей, зафіксованих у табличному вигляді, створених і відпрацьованих ще до впровадження VHDL-описів, а також існування готових VHDL-описів різноманітних циклів шин мікропроцесорів.

## 2.2. Функціональні можливості мови VHDL

Для опису цифрових схем, що знаходяться всередині ПЛІС (ядер), використовується мова VHDL [3]. Ядра ПЛІС мають стандартні паралельні інтерфейси для під'єднання їх до мікропроцесорів або мікроконтролерів, у тому числі побудованих на основі процесорів ARM з 16-бітними і 32-бітними шинами, і стандартні послідовні інтерфейси, наприклад, такі як CAN та SciWay для під'єднання до віддалених периферійних пристроїв та побудови спеціалізованих комп'ютерних мереж. Для перевіряння правильності створених VHDL-описів ядер здійснюється їхнє моделювання за допомогою додатково створюваного (у вигляді опису на мові VHDL) програмного стенду (testbench) [18, с. 40]. Розробник об'єкта не має можливості приділяти велику увагу і велику кількість часу розробленню складного програмного стенда, а може викласти у вигляді простої таблиці послідовність потрібних йому тестових впливів.

Мова VHDL (Very high Speed integrated circuit Hardware Description Language) була розроблена міжнародною групою за завданням Міністерства оборони США на початку 80-х років. Стандарт IEEE 1076 на версію цієї мови був затверджений в 1987 р. Роботу над удосконаленням стандарту веде група VASG (VHDL Analysis and Standartisation Group). Термін регулярного перегляду стандарту-п'ять років, і черговий варіант мови з'явився в 1993 р. Ведуться також роботи з розширення VHDL в області опису аналогової апаратури – VHDL-A, стандартизації внутрішньої форми подання VHDL-описів в ЕОМ (група VI-FASG), форми завдання тестів для VHDL-моделей (група WAVES), завдання параметрів затримок компонент (VITAL), алфавіту представлення значень сигналів в моделях і операції в цьому алфавіті (стандарт IEEE stdJogic_l 164) і т.д. [20, с. 42].

До основних переваг мов VHDL слід віднести наступне:

1. Стандартність. Краще мати поганий стандарт, ніж не мати ніякого. Це підтверджує біблійний досвід створення Вавілонської вежі, різномовність будівельників якої, за переказами, призвело до сумного результату,

VHDL і VERILOG офіційно визнані стандартом опису цифрової апаратури, який підтримується військово-промисловим комплексом і радіоелектронної промисловістю західних країн. Цей стандарт полегшує обмін документацією між окремими групами розробників і експлуатаційників апаратури, різними системами автоматизації проектування (САПР).

2. Багатоаспектність і багаторівневість. Універсальний засіб замінює кілька спеціалізованих. Мови VHDL і VERILOG придатні для опису як схем апаратури, так і функціональних тестів і алгоритмів функціонування. Вони покривають широкий діапазон рівнів структурної деталізації описів ЦА: від описів архітектури ЕОМ на рівні пристроїв типу процесор-пам'ять до описів вузлів типу тригер на рівні вентилів і МОП-ключів, від описів алгоритмів ЕОМ на рівні команд до описів алгоритмів пристроїв на рівні міжреєстрових передач і булевських функцій, від описів функціональних тестів до тестів перевірки схем. На вищих рівнях Абстракції VHDL- і VERILOG-описи можна розглядати як засіб специфікації вимог до проекту.
3. Людино-машинність. Документація пишеться один раз, а читається багаторазово. Творці VHDL і VERILOG знайшли досить вдалий компроміс між вимогами до мови як до засобу документування, зручного для сприйняття людиною, і як формального засобу, зручного для введення і обробки описів систем в ЕОМ. VHDL – і VERILOG-описи апаратури придатні для обробки такими компонентами САПР, як підсистеми моделювання, підсистеми формальної верифікації, reuse-checker'bi (програми перевірки стилю HDL-описів на синтезабельність і придатність до повторного використання в різних проектах), підсистеми логічного синтезу, підсистеми синтезу з урахуванням тестопридатності, системи синтезу та аналізу контрольних тестів, тимчасові аналізатори, кремнієві компілятори, підсистеми автоматизації конструкторського проектування і т. п. [21, с. 4].

Існує дві реалізації програми VHDL, де одна її частина використовує чергування блоків обробки, а інша-конвеєризацію.

Основний блок однаковий для обох програм, і він показаний на рис. 2.3. Він вводить складний вектор. Початок сигналу вказує на те, що значення на вході готові, і програма повинна вивести значення кута.

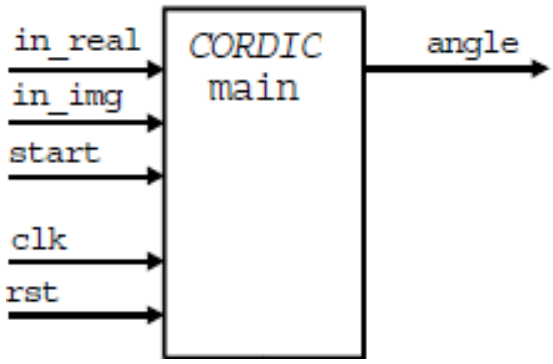



*Рис. 2.3. Основний блок програми VHDL.*

Цей блок складається з двох блоків управління і двох арифметичних блоків. Розглянемо кожну програму окремо. Структурна схема переміжної програми наведена на рис. 2.4.

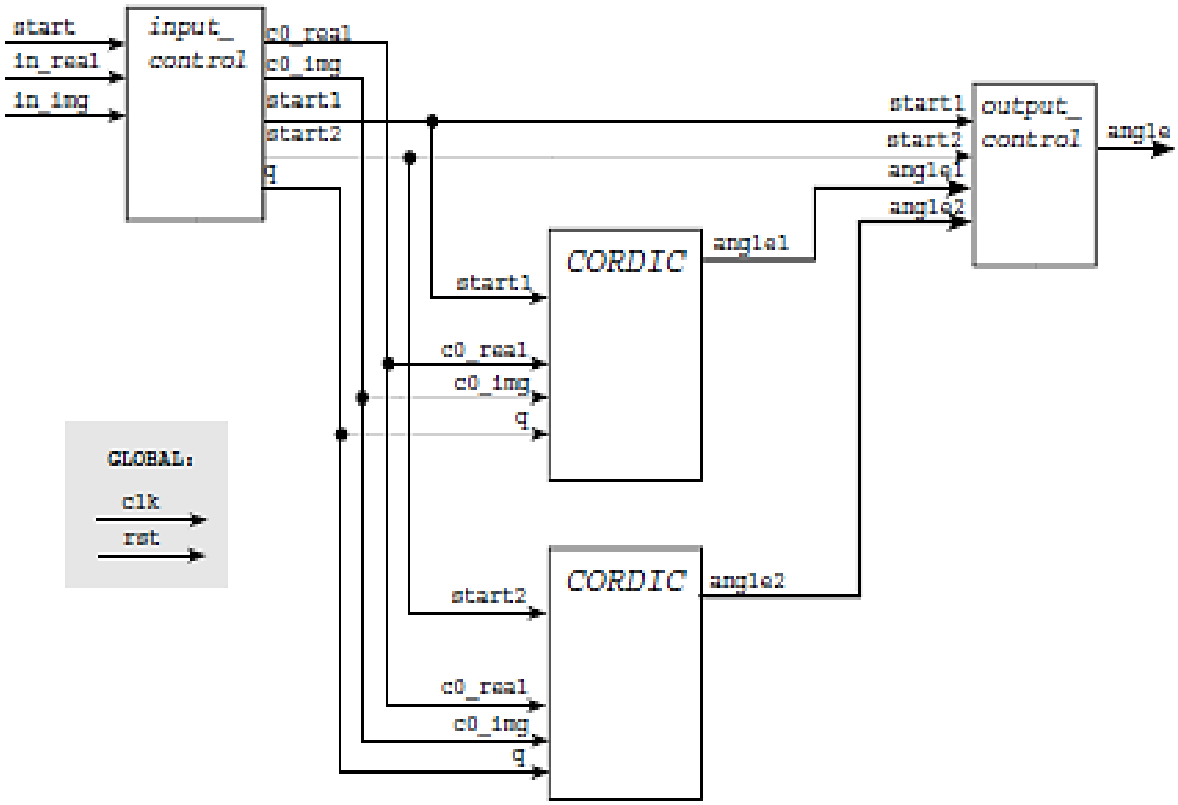



*Рис. 2.4. Структурна схема чергування блоку VHDL.*

Цей блок відповідає за перетворення вектора у внутрішній формат, оскільки алгоритм CORDIC обертає вектор у правій половині комплексної площини, але в бездротовій локальній мережі вектор направляється в будь-яку сторону створеної схеми.

Отже, вхідний вектор перетворюється відповідно до цієї вимоги. Коли вхідний вектор має негативну дійсну частину, він інвертується і початкове значення кута дорівнює плюс або мінус π (див. таблицю 2.1). В іншому випадку вхідний вектор не змінюється, а початкове значення кута дорівнює 0.

Сигнал quarter вказує на чверть вихідного вектора для внутрішніх блоків програми.

*Таблиця 2.1*

**Розрахунок кута нахилу**

| Чверть сигналу, Qi | Діапазон кутів | Початкове значення кута |
|---|---|---|
| «00», «10» | [-π /2; π /2] | Кут0 = 0 |
| «01» | [π /2; π ] | Кут0 = π |
| «11» | [-π; – π /2] | Кут0 = – π |

Вхідний блок управління зчитує вхідні сигнали і виводить на кожен кордовий блок сигнали початку періоду з двома вибірками і внутрішні векторні значення. Потім кожен блок виконує обертання кордового вектора і виводить кут на другий блок управління, який фактично є мультиплексором.

Всі сигнали представлені у форматі фіксованої точки. Їх структура наведена в таблиці 2.2:

*Таблиця 2.2*

**Структура сигналу**

| Ім’я сигналу | Тип | Інтерпритація | Діапазон |
|---|---|---|---|
| In real, in_img | std_logic_vector (11 до 0) | S###.######## | [-8; 7.99609] |
| x, y (внутрішній сигнал) | std_logic_vector (15 до 0) | S#####.##...## | [-32; 31.9990234] |
| кут1, кут2 | std_logic_vector (11 до 0) | S##.######### | [-4; 3.998046] |

Сигнали x і y використовуються всередині арифметичних блоків. У них достатньо захисних бітів, щоб уникнути переповнення, і круглих бітів, щоб зберегти необхідну точність обертання вектора. Розглянемо кожен блок.

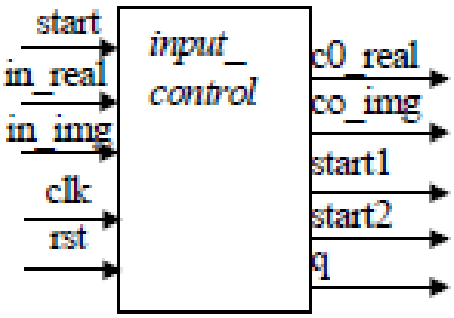



*Рис. 2.5. Вхідний блок управління*

На рис. 2.5. показана структура вхідного блоку управління. Часова діаграма для програми наведена в додатку В. при високому рівні сигналу запуску вона дивиться на свій внутрішній прапор, інвертує його і виводить startl, якщо він низький, і start2 в іншому випадку. Ці керуючі сигнали вказують на те, що 1-й або 2-й арифметичний блок, відповідно, повинен почати свій розрахунок.

Крім того, він перетворює вхідні векторні значення у внутрішнє 16-бітове представлення. Якщо дійсна частина негативна, то обидві частини беруться з негативним знаком (c0_real, c0_img) = (- in_real, – in_img). Сигнал Q (квартал) складається з двох бітів, які є знаком в img і в реальних сигналах [23, с. 85].

Арифметичні блоки, CORDIC

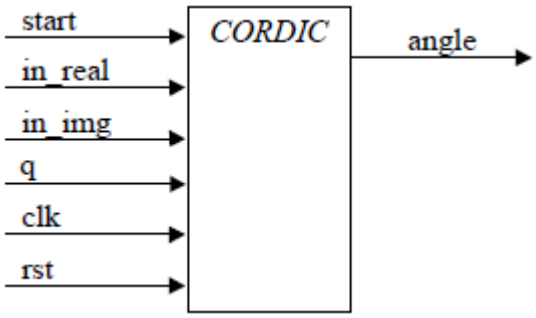



*Рис. 2.6. Блок CORDIC*

При високому рівні стартового сигналу кожен блок CORDIC скидає значення сигналу внутрішнього лічильника (cnt) на 1 і обчислює 1-й крок CORDIC. Обчислені значення зберігаються в сигналах кут, x і y для кута, дійсної і уявної частин відповідно. Потім блок починає виконувати Кордові обертання, в той час як cnt менше Nsteps, що задається як загальна константа. Після цього, коли cnt дорівнює n крокам, сигнал кута готовий, і блок припиняє робити

CORDIC обертання і чекає наступного стартового сигналу або, якщо замість цього надходять наступні стартові сигнали, він правильно виводить значення кута і запускає свій наступний цикл.

Блок управління виходом.

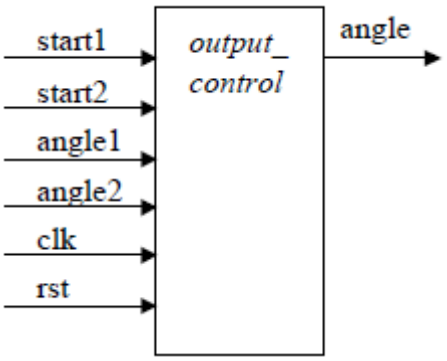



*Рис. 2.7. Блок управління виходом*

Цей блок відіграє роль мультиплексора, він виводить сигнал кут1 кут2 на начало1, коли приходить або з start2 сигналу.

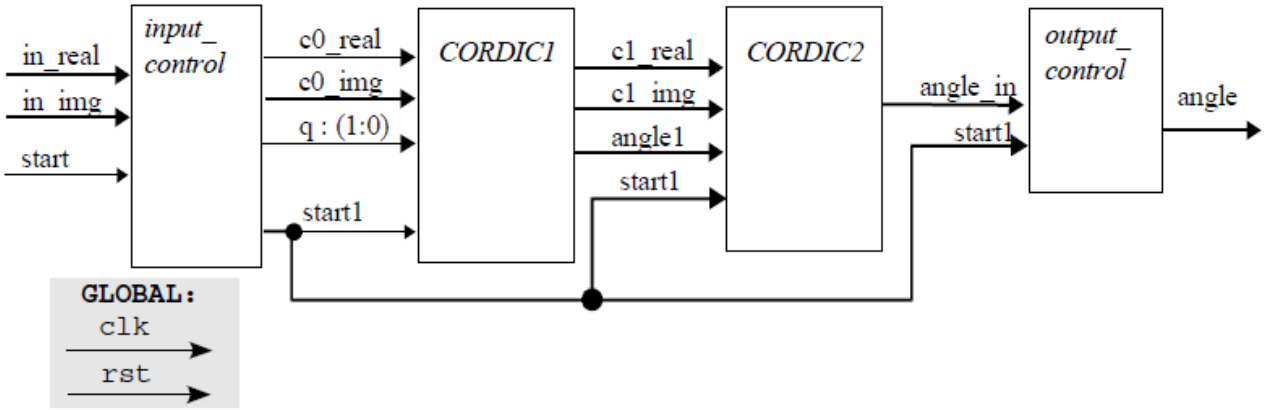



*Рис. 2.8. Структурна схема конвеєрної архітектури VHDL*

Ця програма схожа на програму, описану вище, але вона ділить блок CORDIC на дві частини, CORDIC1 і CORDIC2, з 5-тактовим циклом на період вибірки кожен. Структурна схема конвеєрної архітектури VHDL показана на рис. 2.8.

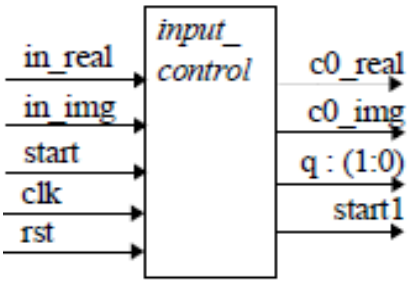



*Рис. 2.9. Вхідний блок управління.*

Цей блок аналогічний вхідному блоку управління для версії чергування. Він перетворює сигнали inreal і in_img у внутрішній формат і виводить q-сигнал таким же чином. Він також виводить конвеєрний сигнал start1 для всіх блоків, який вказує на те, що почався новий період вибірки.

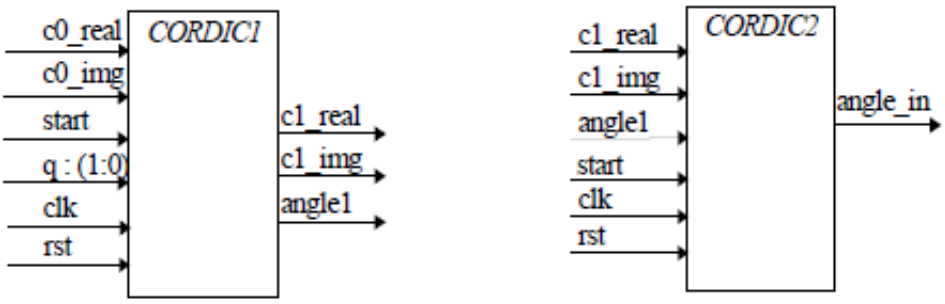



*Рис. 2.10. Арифметичні блоки для конвеєрної версії.*

Ці блоки показані на рис. 2.10. Вони мають однакову структуру, але 1-й блок виконує перші 5 CORDIC кроків, а 2-й-останні 5 кроків. Як і в чергується версії, кожен блок використовує свій внутрішній зустрічний сигнал для виконання обертань. Основна відмінність між цими двома блоками полягає в тому, що другий блок не потребує сигналу кутової чверті (q), і він також не виводить векторне значення [25, с. 218].

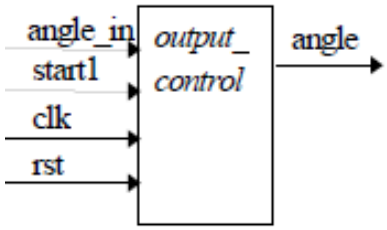



*Рис. 2.11. Блок управління виходом*

Для конвеєрної програми він діє як тригер кута в сигналі і виводить його, коли він готовий, тобто на високому рівні конвеєрного сигналу start1.

# РОЗДІЛ 3
# РОЗРОБКА МОДУЛЮ ОБЧИСЛЕННЯ ФУНКЦІЇ arctg y / x ТА sqrt (x2+y2) З ПЛАВАЮЧОЮ КОМОЮ

## 3.1. Реалізація об’єкту дослідження в схемах ПЛІС

Програмовані логічні інтегральні схеми (ПЛІС) – це сучасна елементна база, що призначена для високопродуктивного виконання спеціалізованих алгоритмів з числами, які представлені з фіксованою комою. Дуже часто в таких алгоритмах зустрічається обчислення елементарних функцій. Але постачальники САПР ПЛІС не забезпечують розробників готовими високопродуктивними віртуальними модулями обчислення елементарних функцій, а фірми-розробники таких модулів поширюють їх за велику ціну (близько тисячі доларів США). Сучасні ПЛІС характеризуються властивістю багаторазового перепрограмування, низькою вартістю виготовлення, досить високою швидкодією, малою енергією споживання тощо. Використання ПЛІС для побудови реконфігурованих комп’ютерних систем є актуальним та доречним рішенням для підвищення ефективності паралельної обробки даних. Така концепція базується на програмноапаратній реалізації поставленої задачі обробки даних, яка ґрунтується на пришвидшенні певних частин алгоритму, оскільки їх виконання є недоречним за допомогою простих процесорів. Крім того, серед них не зустрічаються модулі, які здатні обчислювати кілька різних функцій. Отже, існує нестача в проектах пристроїв для обчислення елементарних функцій в ПЛІС і вони потребують удосконалення.

За способом зберігання конфігурації ПЛІС [26, с. 322] поділяються на:

SRAM based. Найбільш розповсюдженій тип ПЛІС. Конфігурація зберігається в статичній пам’яті, типу CMOS. До переваг можна віднести – можливість багаторазової реконфігурації. Через особливість використаної пам’яті, мікросхему необхідно повторно конфігурувати при кожному вмиканні. Це призводить до збільшення апаратної складності, оскільки кінцевий пристрій

повинен мати додаткові компоненти, такі як мікросхема FLASH та завантажувач конфігурації;

Flash based. В даному типі, конфігурація мікросхеми зберігається у внутрішній FLASH чи EPROM пам'яті. Цей тип мікросхем не має недоліків ПЛІС типу SRAM, але за рахунок використання в них FLASH та EPROM пам'яті є дорожчими у виробництві. Як правило, мікросхема має обмежену кількість циклів перезапису [30, с. 197];

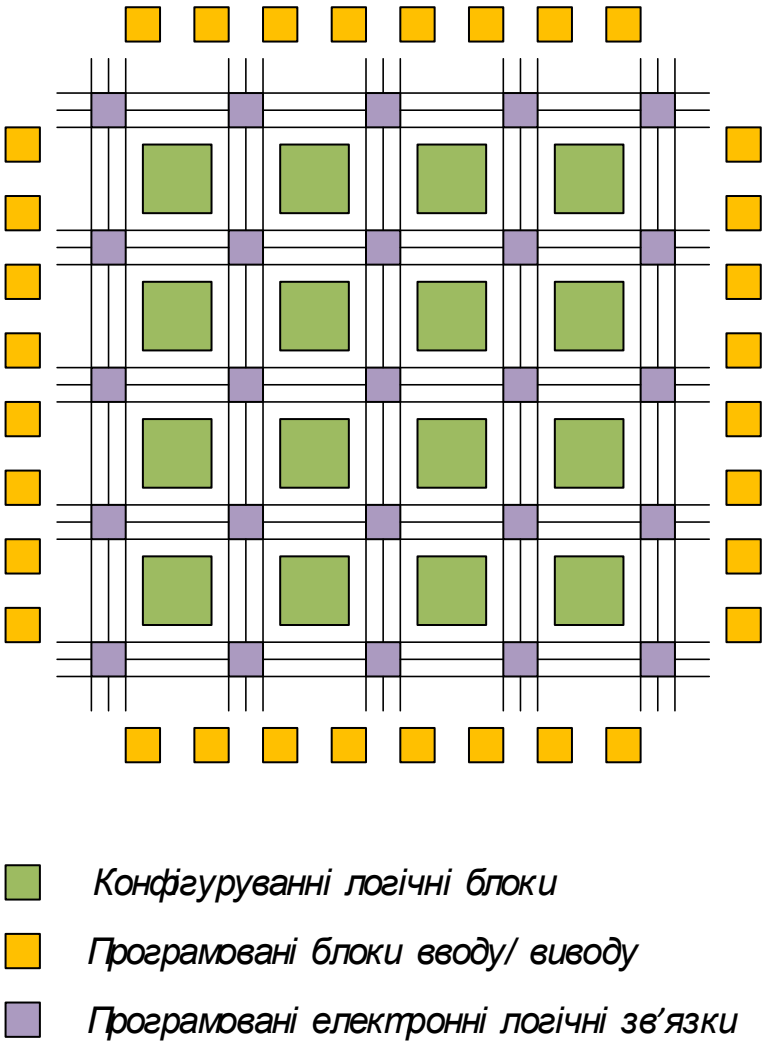



*Рис. 3.1. Спрощена структура програмованої логічної інтегральної схеми*

Anti-fuse based. Технологія за якою конфігурація програмованої логічної інтегральної схеми виконується одноразово. Цей процес побудовано на плавленні спеціальних перемичок у потрібному місці на чипі. До переваг даного типу відносять високу швидкодію та надійність [34, с. 18].

Адаптивна реконфігурація мікросхеми ПЛІС економить час за рахунок того, що коли приходить нова задача, мікросхема вже сконфігурована і може зразу починати виконання операції без необхідності завантаження.

На рис. 3.2. показано спрощену структуру обчислювальної системи на базі ПЛІС.

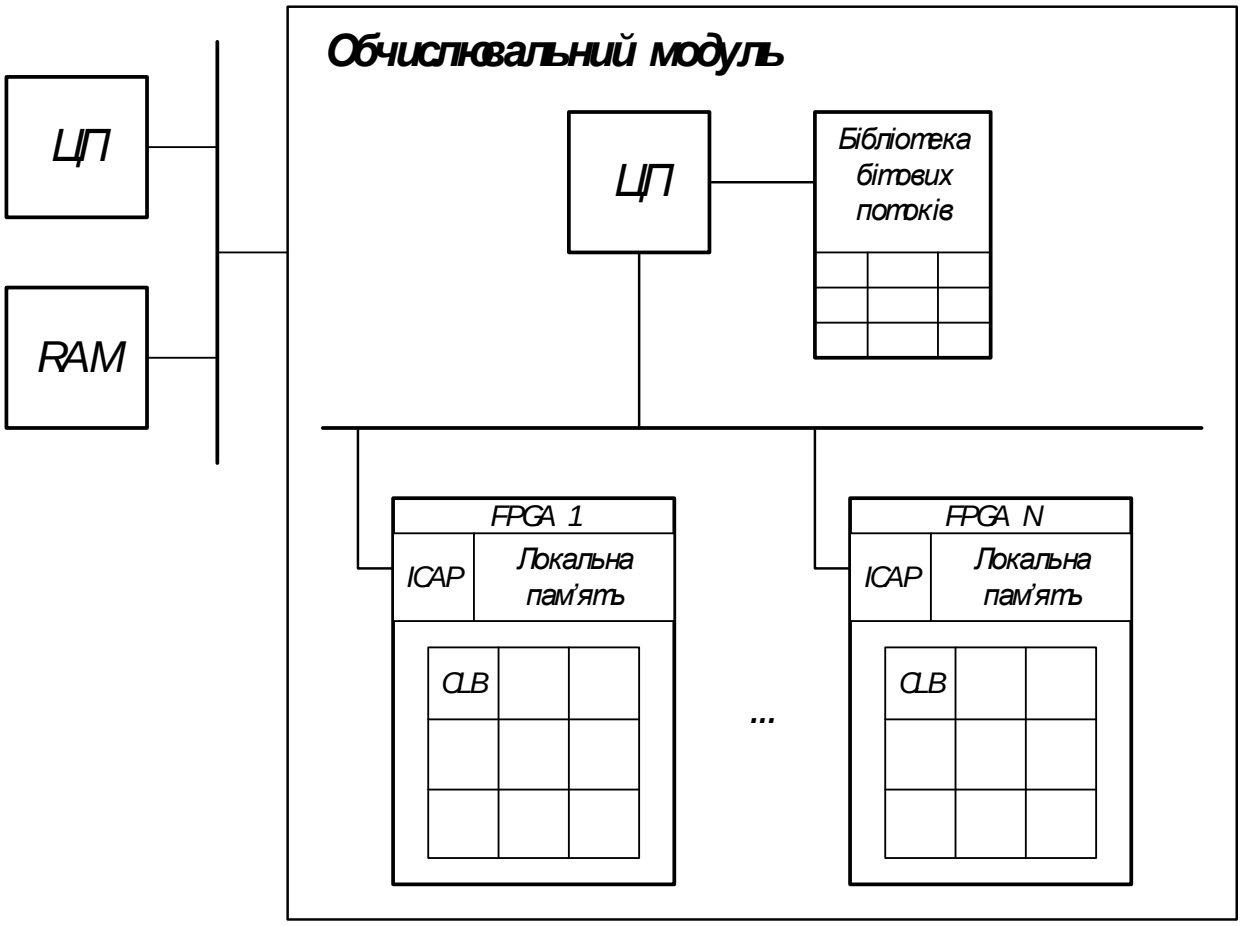



*Рис. 3.2. Спрощену структуру обчислювальної системи на базі ПЛІС*

ICAP-Internal Configuration Access Port

CLB-Configurable Logic Block

Робота обчислювальної системи має наступний вид. Сформований користувачем, граф задачі передається до обчислювального модуля. В ньому ЦП обчислювального модуля аналізує граф і розбиває його на яруси. Потім виконується побудова черги реконфігурації модулей ПЛІС бітовими потоками реконфігурації, що завантажуються із бібліотеки бітових потоків. В результаті, до ПЛІС передаються бітові потоки операцій, що треба сконфігурувати та виконати.

Граф задачі представляється у ярусно-паралельній формі рис. 3.3. На кожному ярусі проводиться реконфігурація задач наступного ярусу. В першу чергу реконфігуруються задачі, які мають найбільший час завантаження [31, с. 164].

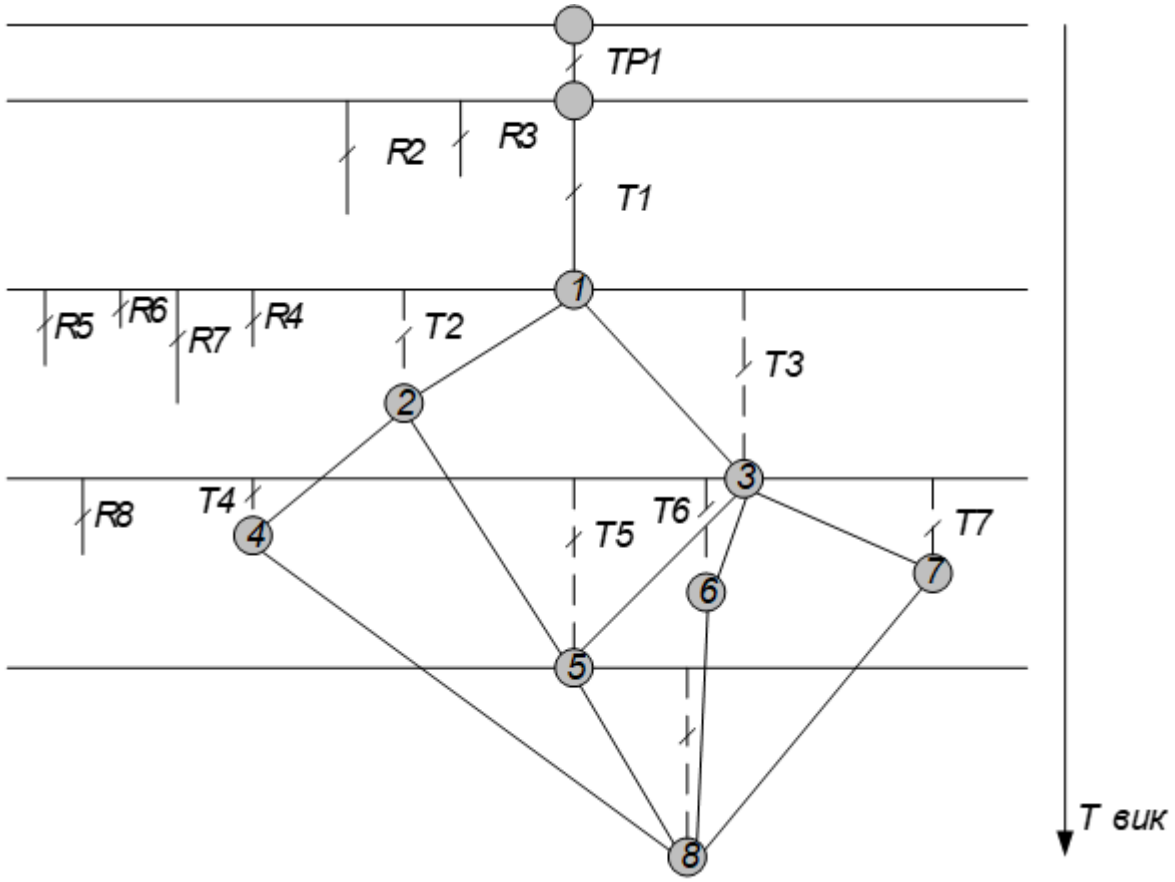



*Рис. 3.3. Ярусно-паралельна форма*

В нашій системі повинен бути один вхід – тактовий (наприклад, 96 МГц), два виходи даних для обох ЦАП (розрядністю 12 біт) і тактові частоти для обох ЦАП.

Щоб отримати потрібну частоту синуса на виході, необхідно на вході конвеєра CORDIC кожен такт вхідної частоти clk змінювати фазу з певним алгоритмом. Це буде виконувати модуль step_control. V – модуль управління кроком. Щоб кожен новий такт clk на виході системи з'являвся новий звіт синуса, реалізуємо CORDIC конвеєром (послідовний ланцюжок повертаючих модулів rotator. v). Необхідно сказати, що CORDIC обчислює тільки першу

чверть періоду синусоїди. Іншу частину сигналу допоможе «домалювати» модуль select_quarter. v.

Розглянемо ієрархію модулів проекту, який формує синус і косинус для двох 12-розрядних ЦАП (рис. 3.4.).

*. Структурна схема*

Sinus10kHz. V – головний модуль, або, як кажуть FPGA-дизайнери, top-level модуль. Він містить в собі всі інші модулі, пов’язує їх між собою і має наступні вхідні і вихідні сигнали:

– input clk – тактова частота від PLL обраної ПЛІС. У нашому проекті – 96 МГц.
– output led1 – вихід на світлодіод. Показує наявність живлення на ПЛІС.
– output idclk, qdclk – тактові виходи для кожної ЦАП.
– output [11: 0] I, Q – відліки для двох ЦАП (синус і косинус відповідно), для формування квадратурного сигналу.

Step_control. v – модуль, що формує кут (крок фази) для обчислення синуса і косинуса, а також чверть, в якій знаходиться вихідний сигнал. Нагадаємо, що частота вихідних синуса і косинуса повинна становити 10 кГц. Список вхідних і вихідних сигналів:

– input clk – тактова частота.
– output Angle – крок фази.
– output quarterjn – чверть.

Cordic. v – модуль, що формує конвеєр з блоків, що обчислюють проміжні значення синуса і косинуса (містить всі ітерації обчислень). Вхідні та вихідні сигнали модуля:

– input clk – тактова частота.
– input rst – апаратне скидання при включенні живлення.
– input theta_i – фаза, для якої необхідно обчислити синус і косинус.
– input quarter_in – вхідна чверть сигналу.

– output quarter_out – вихідна четзерть сигналу.
– input x_i – початкове значення синуса (дорівнює коефіцієнту деформації).
– input y_i – початкове значення косинуса (дорівнює нулю).
– output x_o – обчислене значення синуса.
– output y_o – обчислене значення косинуса.
– output theta_o – залишкове значення кута (похибка фази).

select_quarter. v – модуль, що підводить обчислене значення синуса і косинуса під весь діапазон ЦАП. Входи і виходи аналогічні розглянутим вище.

reset_block. v – організовує апаратне скидання і установку всіх регістрів в початкове значення.

rotator. v – модуль, що повертає вектор на заданий кут (реалізує кожну з ітерацій).

Розробка алгоритмів програми.

Дана програма повинна виконувати наступні функції:

1. Виконувати розбиття графу задачі на яруси;
2. Моделювати роботу ПЛІС, зокрема функції конфігурації бітових потоків операції та виконання сконфігурованої операції.

На рис. 3.5. було представлено розроблений алгоритм розбиття графу на яруси. Оскільки використовується об’єктно-орієнтована мова програмування Java, то для спрощення роботи із графом було створено примітив вершини графа, який має списки вхідних та вихідних вершин. Оскільки кожна вершина має інформацію про вершини, що входять і виходять із неї, то це дає можливість полегшати виконання розбиття графа на яруси. Також був створений примітив ярусу графа, що містить список вершини графа, які можуть виконуватися на даному ярусі. Це дозволить в подальшому спростити виконання планування реконфігурації ПЛІС [27, с. 4].

Для початку розбиття на яруси створюється список вершин, які були оброблені та список ярусів. В подальшому список вершин буде

використовуватися для визначення можливості виконання певної вершини на ярусі.

Далі виконується циклічний прохід по вершинах графу. На кожній ітерації вибирається вершина і перевіряється чи була вона оброблена. Якщо ні, то в даної вершини отримується список вхідних вершин, тобто вершин від яких вона залежить і виконується перевірка кожної вхідної на те, чи була вона виконана на попередньому ярусі. Якщо хоча б одна вхідна вершина не була виконана, то вершина пропускається, інакше вона додається на поточний ярус.

Після завершення заповнення поточного ярусу, він додається до списку ярусів, а його вершини – до списку оброблених вершин.

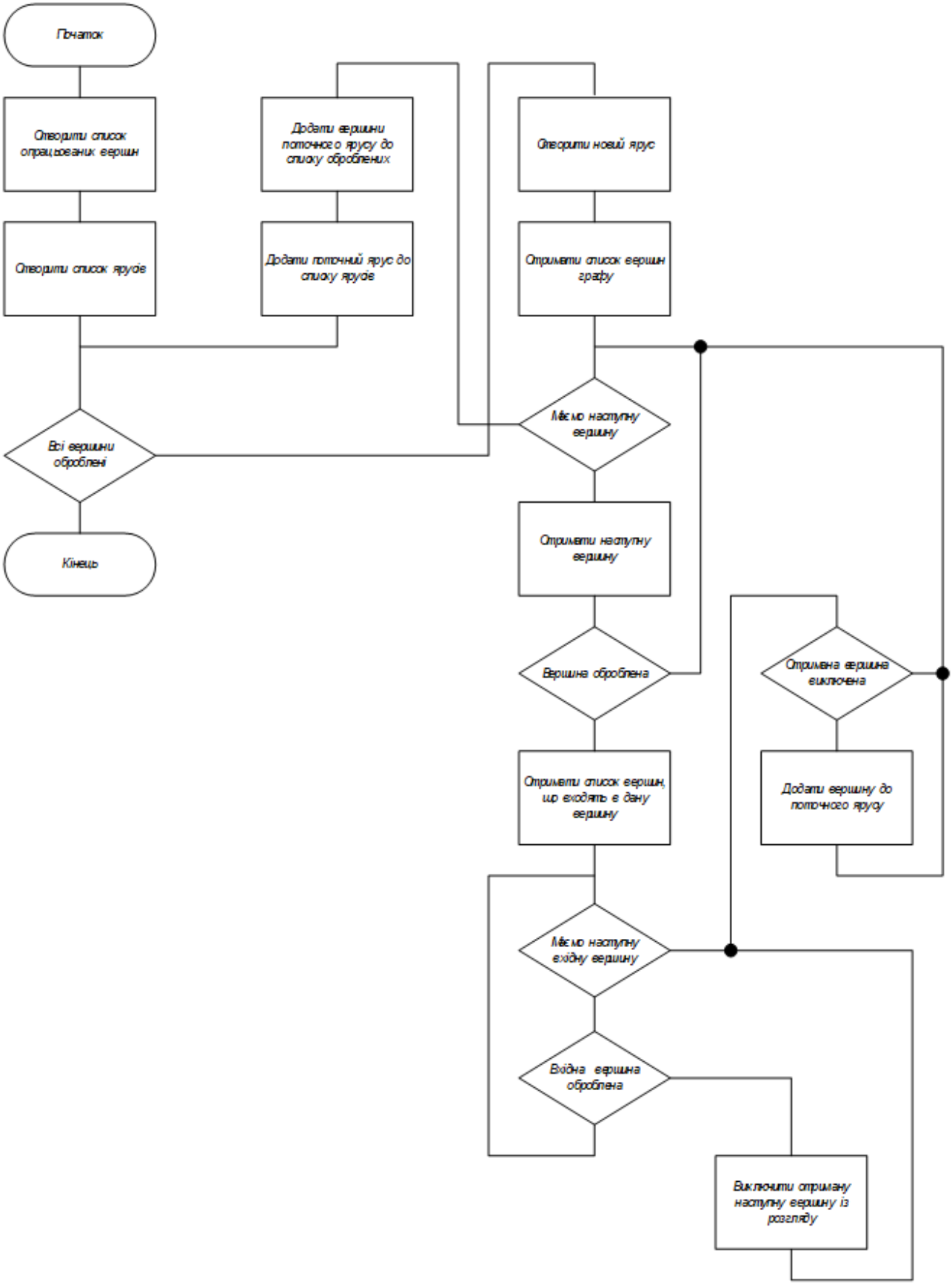



*Рис. 3.5. Алгоритм розбиття графу на яруси*

Далі після виконання етапу розбиття вершин графу на яруси, необхідно виконати моделювання роботи ПЛІС, щоб отримати часові характеристики виконання кожної задачі. На рис. 3.6. показано алгоритм моделювання роботи ПЛІС.

*Принципова схема алгоритму моделювання роботи ПЛІС*

Для виконання моделювання граф, розбитий на яруси, перетворюється на яруси симуляції. Щоб полегшати опрацювання результатів моделювання, було створено примітив «SimulationStage» – ярус симуляції, що містить у собі списки задач конфігурації та виконання операцій в ПЛІС [32, с. 37].

Під час моделювання, почергово вибираються яруси графа. Із них отримуються списки вершин, що необхідно виконати на даному ярусі. За назвою вершини отримується операція, яку вона виконує. Потім виконується конфігурація даної операції на ПЛІС, а потім її виконання. Під час моделювання роботи, час завершення задачі конфігурації або задачі виконання операції фіксується, надалі він буде використаний для побудови часової діаграми роботи ПЛІС. Кожна задача потім додається до ярусу симуляції.

У програмі адаптивної реконфігурації обчислювальної системи на базі ПЛІС, користувач буде мати змогу створити граф задачі у редакторі графу та промоделювати створену задачу, для отримання часової діаграми виконання операцій. Функціонал програми представлений діаграмі прецедентів (рис. 3.7.).

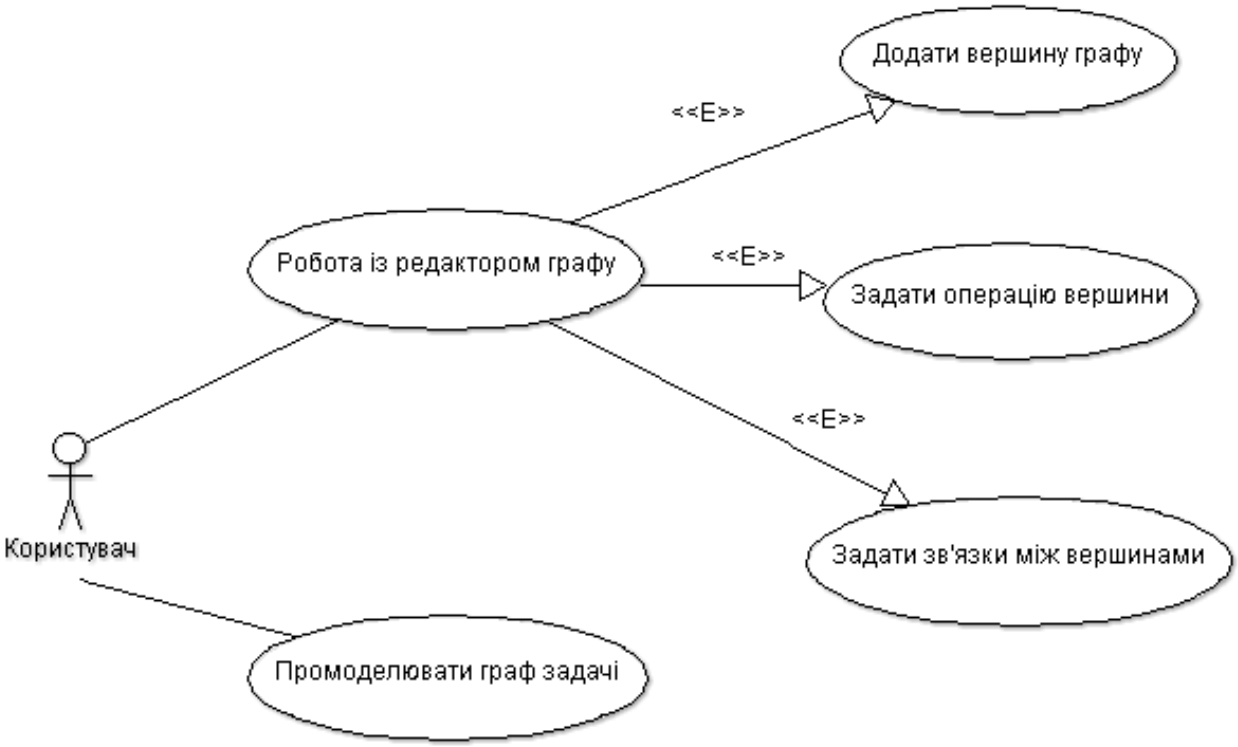



*Рис. 3.7. UML діаграма прецедентів*

Користувач у програмі буде мати можливість виконувати наступні дії:

1 Сценарій

Назва: Додати вершину графу.

Учасники: Користувач, Система.

Результат: Вершина графу додається на поле редактору.

Основний сценарій:

1. Користувач натискає на кнопку «*Add hill*»;
2. Система додає нову вершину на поле;
3. Користувач, за допомогою миші, обирає бажану позицію вершини на полі.

2 Сценарій

Назва: Задати операцію вершини.

Учасники: Користувач, Система.

Передумови: Вершина додана на поле редактору.

Результат: Задання операції вершини графу.

Основний сценарій:

1. Користувач натискає правою кнопкою миші на потрібній вершині;
2. Користувач обирає пункт меню «*Edit*»;
3. Система видає вікно «*Enter hill operation*»;
4. Користувач вводить необхідну операцію;
5. Користувач обирає дію «*OK*»;
6. Система задає операцію вершині.

3 Сценарій

Назва: Задати зв’язки між вершинами.

Учасники: Користувач, Система.

Передумови: Дві вершини присутні на полі редактору.

Результат: Створення зв’язку між двома вершинами.

Основний сценарій:

1. Користувач натискає правою кнопкою миші на вершині з якої проводить зв’язок;
2. Користувач обирає пункт меню «*Link (Start)*»;
3. Користувач натискає правою кнопкою миші на вершині до якої проводить зв’язок;
4. Користувач обирає пункт меню «*Link (Finish)*»;
5. Система створює зв’язок між двома зазначеними вершинами.

Виключні ситуації: -

4 Сценарій

Назва: Промоделювати граф задачі.

Учасники: Користувач, Система.

Результат: Система видає часову діаграму виконання задачі.

Основний сценарій:

1. Користувач натискає на кнопку «*Simulation*»;
2. Система видає вікно «*Result*» із часовою діаграмою.

Виключні ситуації: -

У програмі виділяються декілька основних компонентів: «*Model*», «*Controller*» та декілька «*View*». Основна логіка роботи із графом задачі зосереджена у класі «*EditorModel*». Логіка, що моделює роботу мікросхеми ПЛІС зосереджена у класі «*FPGAModel*». Програмна реалізація роботи ПЛІС моделює основні функції мікросхеми, такі як завантаження бінарних потоків операцій до локальної пам’яті моделя, конфігурація логічних блоків ПЛІС та виконання сконфігурованих операцій.

На основі необхідних функцій програми була розроблена діаграма класів програми (рис. 3.8.). Контролер у цій системі відіграє роль диспетчера, який передає запити від представлень до моделі та навпаки. У разі оновлення свого стану, модель повідомляю про це контроллер, який у свою чергу оновлює усі необхідні представлення.

Маніпуляції користувача на представленнями обробляється за допомогою команд, які передаються на виконання до контролера. Команда в цьому випадку має усю необхідну інформацію про дії, які треба виконати над моделлю.

Анотації у системі виконують роль міток, які містять інформацію про таблиці, з яких треба брати дані; відмічають спеціальні поля; містять інформацію про дані, які необхідні для відображення представлень.

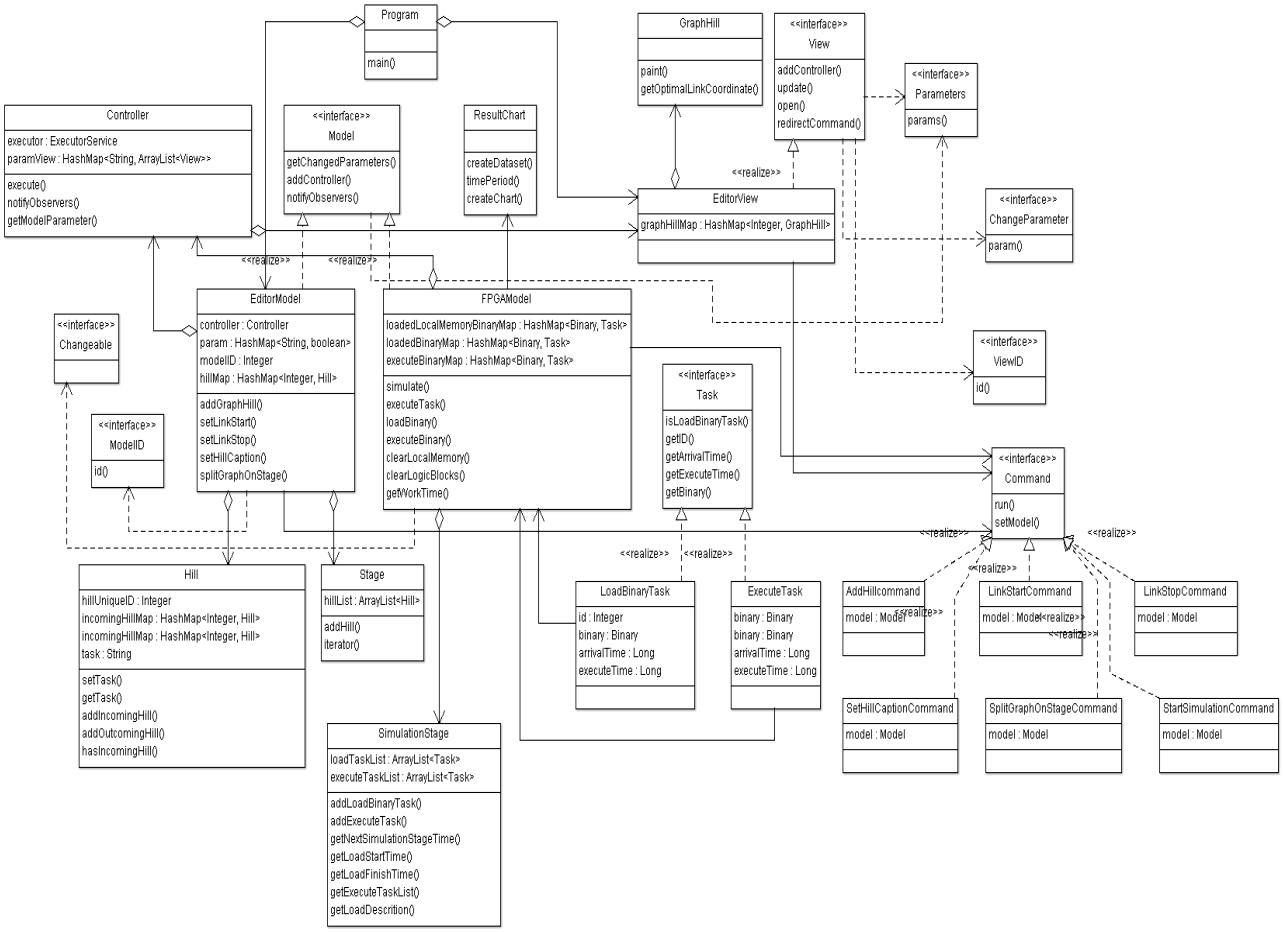



*Рис. 3.8. UML діаграма класів програми*

За наведеною діаграмою класів (рис. 3.8.) було розроблено програму адаптивної реконфігурації обчислювальної системи на базі ПЛІС. Було реалізовано зазначені функції, такі як: створення графу задачі у редакторі, розбиття графу задачі на яруси, моделювання роботи ПЛІС та відображення результатів моделювання у формі часової діаграми.

Функції, що моделюють роботу ПЛІС зосереджені у наступний класах:

– FPGAModel – клас, що реалізує програмну модель мікросхеми ПЛІС;
– Binary – програмна модель, що імітує властивості скомпільованих бітових потоків певних операцій;
– BinaryDatabase – виконує роль сховища бітових потоків;
– Task – імітація задач, що виконуються у мікросхемі;
– ExecuteTask – описує задачу виконання певної сконфігурованої операції мікросхемою;

– LoadBinaryTask – описує задачу завантаження бітових потоків до мікросхеми.

**Основні методи класу FPGAModel:**

– executeTask – виконання певної задачі мікросхемою ПЛІС;

– loadBinary – метод, що моделює завантаження певного бітового потоку до мікросхеми. Якщо на ПЛІС не вистачає вільних логічних блоків для конфігурації, виконується завантаження бітового потоку до локальної пам’яті мікросхеми. У разі недостатньої кількості вільної пам’яті, завантаження бітового потоку відхиляються;

– executeBinary – метод, що моделює виконання мікросхемою певної операції, що була попередньо сконфігурована;

– clearLocalMemory – моделює видалення бітового потоку із локальної пам’яті ПЛІС;

– clearLogicBlocks – моделює видалення бітового потоку із логічних блоків ПЛІС;

– getWorkTime – повертає сумарний час роботи мікросхеми.

Функції, для роботи із редактором та розбиттям його на яруси реалізовані в наступних класах:

– EditorModel – клас, що реалізує функції редактору графа та виконання розбиття графу на яруси;

– Hill – програмний примітив вершини графу, що містить тип операції, що виконується у вершині, списки вхідних та вихідних вершин;

– Stage – програмний примітив ярусу графа, що містить список вершин графа, які можуть виконуватися незалежно одна від одної;

– SimulationStage – програмний примітив ярусу симуляції, що використовується для отримання даних для побудови часової діаграми, та містить списки задач конфігурації та задач виконання певної операції на ПЛІС.

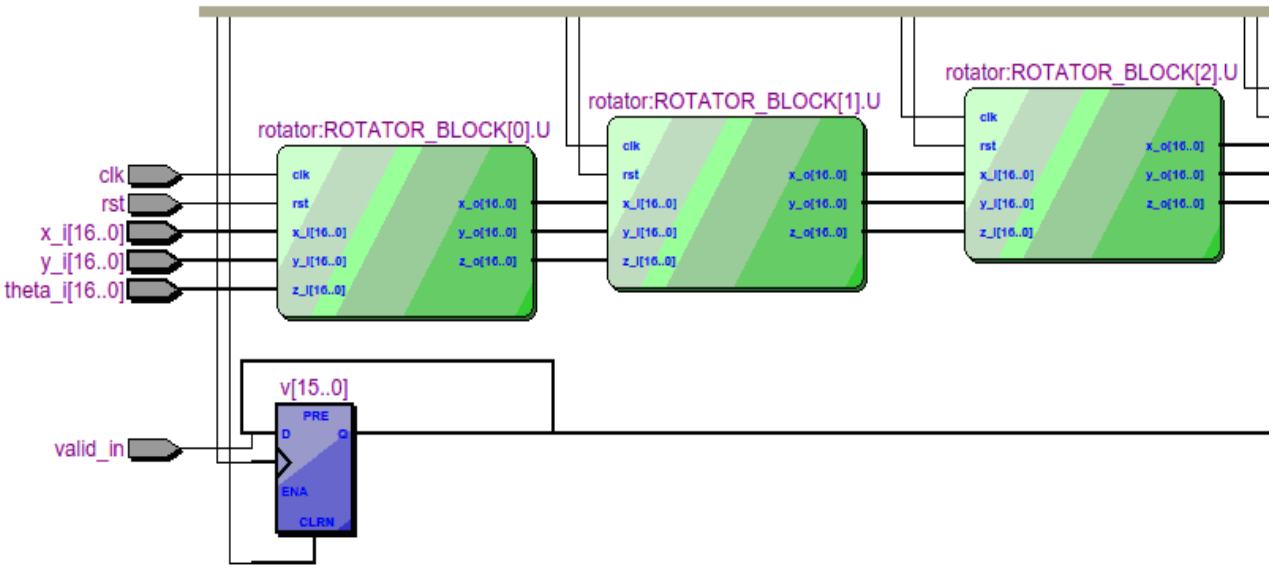



*Рис. 3.9. Відображення модулю CORDIC*

Основні методи класу EditorModel:

– addGraphHill – метод, що створює та додає вершину до списку вершин графа та дає команду на оновлення графічного представлення графу;
– setLinkStart – отримує та зберігає індекс вершини із якої проводиться зв’язок;
– setLinkStop – отримує індекс вершини до якої проводиться зв’язок та створює зв’язок між двома вершинами;
– setHillCaption – встановлює назву операції, що виконується в певній вершині графу;
– splitGraphOnStage – метод, що виконує пошук вершин, які можуть виконуватися незалежно одна від одної, та розміщує їх на ярусах графа задачі.

Опис досліджуваного об’єкту згідно алгоритму CORDIC на мові VHDL відноситься до алгоритмів «цифра за цифрою» і таким чином, за одну ітерацію знаходиться одна точна двійкова цифра результату. Обчислення базуються на простих операціях зсуву і додавання.

CORDIC sincos функція обчислює синус і косинус вхідних кутів в області значень [-π, π), використання алгоритму CORDIC. Ця функція бере кут (радіани)

і кількість ітерацій як вхідні параметри. Функція повертає наближення синуса і косинуса.

Обчислення CORDIC вихідні параметри масштабується посиленням крутного пристрою. Це посилення складається шляхом попереднього масштабування початкового постійного значення.

Розумний вибір початкових значень дозволяє алгоритму безпосередньо обчислювати і синус і косинус одночасно. Після ітерацій ці початкові значення призводять до наступних вихідних параметрів як підходи:

$$y_n = \sin \phi, x_n = \cos \phi$$

Модуль CORDIC відображається з 16 однотипних блоків (рис. 3.9):

Кожен з яких бере на вхід дані з попереднього і виходами підключений до входів наступного. Виглядає він так:

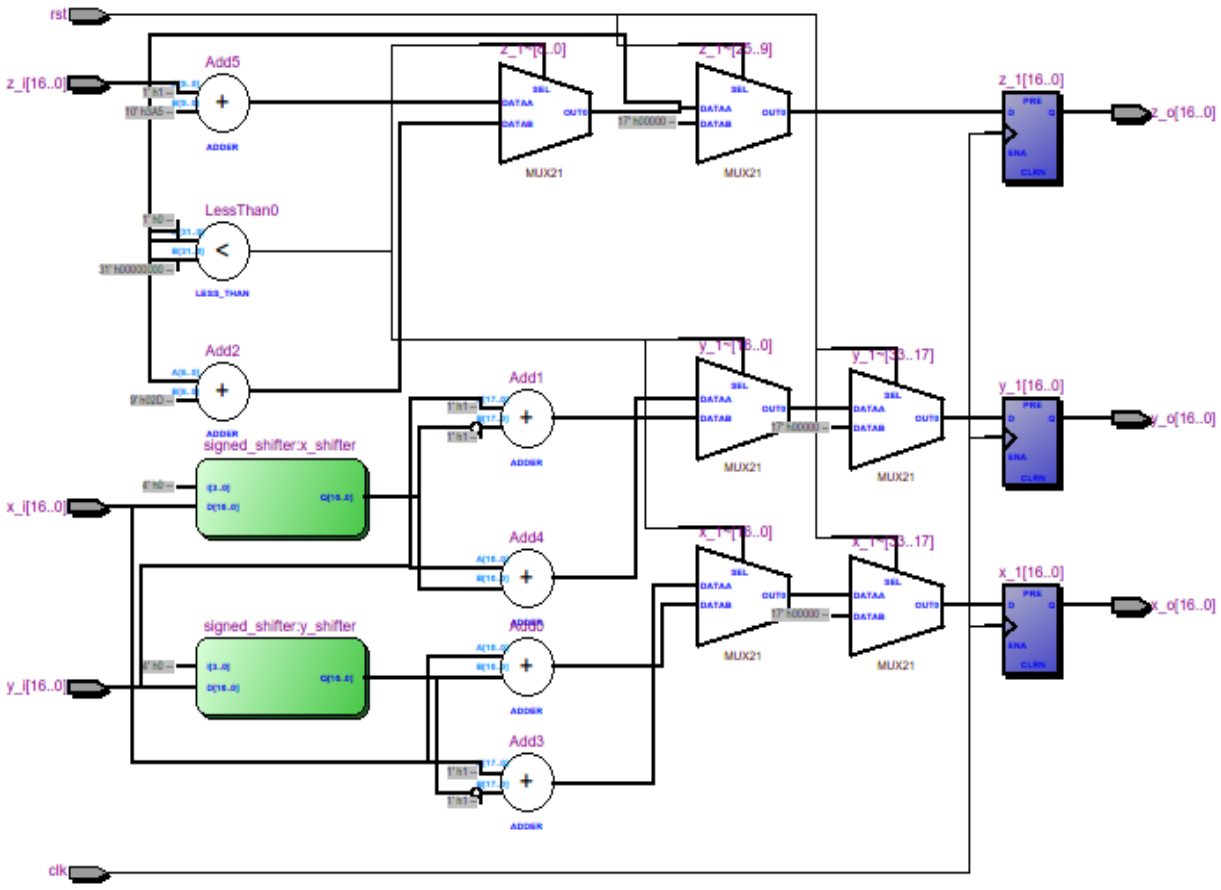



*Рис. 3.10. Функціональна будова досліджуваного об'єкту згідно алгоритму CORDIC*

## 3.2. Програмні коди на мові VHDL для досліджуваних функцій

**arctg y / x**

```
library ieee;
use ieee. std_logic_1164. all;
use ieee. std_logic_textio. all;
use ieee. std_logic_arith. all;
use ieee. numeric_bit. all;
use ieee. numeric_std. all;
use ieee. std_logic_signed. all;
use ieee. math_real. all;
use ieee. math_complex. all;

library STD;
use STD. textio;

entity arctg_y_x is
	port(
		CLK: in STD_LOGIC;
		RST: in STD_LOGIC;
		RED: in STD_LOGIC_VECTOR (23 downto 0);
		IMD: in STD_LOGIC_VECTOR (23 downto 0);
		EI: out STD_LOGIC;
		DO: out STD_LOGIC_VECTOR (23 downto 0);
		PH: out STD_LOGIC_VECTOR (15 downto 0));
end arctg_y_x;

architecture arctg_y_x of arctg_y_x is
	type arr is array (0 to 15) of std_logic_vector (15 downto 0);
	constant TableOfAtans: arr := («0111111111111111», «0100101110010000», «0010011111101101», «0001010001000100», «0000101000101100», «0000010100010111», «0000001010001100», «0000000101000110», «0000000010100011», «0000000001010001», «0000000000101001», «0000000000010100», «0000000000001010», «0000000000000101», «0000000000000011», «0000000000000001»);
	signal i: natural range 0 to 15;
	signal A: std_logic_vector (17 downto 0);
	signal S: std_logic_vector (26 downto 0);
	signal ans_RED: std_logic_vector (23 downto 0):= «000000000000000000000000»;
	signal ans_IMD: std_logic_vector (23 downto 0):= «000000000000000000000000»;
	signal ans_EI: std_logic;
	signal ans_DO: std_logic_vector (26 downto 0);
	signal ans_PH: std_logic_vector (15 downto 0);
begin
	process
	begin
		ans_EI <= '0';
		if RST = '1' then i <= 0;
		elsif rising_edge (CLK) then
			i <= (i + 1) mod 16;
			if i = 15 then ans_EI <= '1';
			end if;
```

```
                end if;
            wait on CLK, RST;
            end process;
            process
            begin
                if RST = '1' then
                    S <= «00000000000000000000000000»; ans_DO <=
«00000000000000000000000000»;
                    ans_IMD <= «00000000000000000000000»; ans_RED <=
«00000000000000000000000»;
                    A <= «00000000000000000»;
                    ans_PH <= «0000000000000000»;
                elsif rising_edge (CLK) then
                    if i = 0 and ans_IMD (23) = '1' then
                        A <= «00000000000000000» – TableOfAtans (i);
                        S <= (ans_IMD (23) & ans_IMD & «00») + (ans_RED & «00»);
                        ans_DO <= (ans_RED (23) & ans_RED & «00») – (ans_IMD
& «00»);
                    elsif i = 0 then
                        A <= «00000000000000000» + TableOfAtans (i);
                        ans_DO <= (ans_RED (23) & ans_RED & «00») + (ans_IMD
& «00»);
                        S <= (ans_IMD (23) & ans_IMD & «00») – (ans_RED & «00»);
                    elsif i < 15 and S (26) = '1' then
                        A <= A-TableOfAtans (i);
                        S <= S + shr (ans_DO, conv_std_logic_vector (i, 4));
                        ans_DO <= ans_DO – shr (S, conv_std_logic_vector (i, 4));
                    elsif i < 15 then
                        A <= A + TableOfAtans (i);
                        ans_DO <= ans_DO + shr (S, conv_std_logic_vector (i, 4));
                        S <= S – shr (ans_DO, conv_std_logic_vector (i, 4));
                    elsif i = 15 then
                        DO <= ans_DO (26 downto 3);
                        ans_IMD <= IMD;
                        ans_RED <= abs (RED);
                    end if;
                    if i = 15 and RED (23) = '0' then
                        ans_PH <= A (17 downto 2);
                    elsif i = 15 and IMD (23) = '0' then
                        ans_PH <= 32767 – A (17 downto 2);
                    elsif i = 15 then
                        ans_PH <= -32767 – A (17 downto 2);
                    end if;
                end if;
            wait on CLK, RST;
            end process;
            EI <= ans_EI;
            PH <= ans_PH;
        end arctg_y_x;
```

**sqrt (x^2+y^2)**

```
library ieee;
use ieee. std_logic_1164. all;
use ieee. std_logic_textio. all;
use ieee. std_logic_arith. all;
use ieee. std_logic_signed. all;
use ieee. math_real. all;
use ieee. math_complex. all;

library STD;
use STD. textio;

entity sqrt_xx_yy is
	port(
		CLK: in std_logic;
		RST: in std_logic;
		RERSP: in std_logic_vector (23 downto 0);
		IMRSP: in std_logic_vector (23 downto 0);
		REO: out std_logic_vector (23 downto 0);
		IMO: out std_logic_vector (23 downto 0);
		FREQ: out INTEGER;
		MAGN: out INTEGER;
		LOGMAGN: out REAL;
		PHASE: out REAL;
		ENA: inout std_logic);
end sqrt_xx_yy;

architecture sqrt_xx_yy of sqrt_xx_yy is
	signal ans_REO: integer;
	signal ans_IMO: integer;
	signal ans_FREQ: integer;
	signal N, D: std_logic;
	constant PI: real :=
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706 79;
begin
	process
		variable P: real := 0.0;
		variable i: integer := 0;
		variable was: boolean := false;
	begin
		if RST = '1' then
		i := 0; P := 0.0;
			ans_IMO <= 0; ans_REO <= 0;
			N <= '0';
		elsif rising_edge (CLK) and ENA = '1' then
		ans_IMO <= integer (1000.0 · sin (2.0 · P · PI));
			ans_REO <= integer (1000.0 · cos (2.0 · P · PI));
		i := i + 1;
			P := P + real (ans_FREQ) / 5000.0;
```

```
                was := true;
            end if;
            if was = true and i = 99 then D <= '1';
            elsif was = true then D <= '0';
            end if;
            if was = true and i = 100 then
                i := 0; N <= '1'; P := 0.0;
            elsif was = true then N <= '0';
            end if;
            wait on CLK, RST;
        end process;
        IMO <= conv_std_logic_vector (conv_signed (ans_IMO, 24), 24);
        REO <= conv_std_logic_vector (conv_signed (ans_REO, 24), 24);
        process
            variable i: integer := 0;
            variable was: boolean := false;
        begin
            if RST='1' then
                i := 0; ENA <= '0';
            elsif rising_edge (CLK) then
                i := i + 1; was := true;
            end if;
            if rising_edge (CLK) and i = 3 and was = true then
                i := 0; ENA <= '1';
            elsif rising_edge (CLK) and was = true then ENA <= '0';
            end if;
        wait on CLK, RST;
        end process;
        process
        begin
            if RST = '1' then ans_FREQ <= 0;
            elsif rising_edge (CLK) and ENA = '1' and N = '1' then ans_FREQ <=
ans_FREQ + 20;
            end if;
            wait on CLK, RST;
        end process;
        FREQ <= ans_FREQ;
        process
            variable ans_MAGN: real := 0.0;
            variable FirstPHASE, SecondPHASE: real := 0.0;
        begin
            if RST = '1' then LOGMAGN <= 0.0; MAGN <= 0; PHASE <= 0.0;
            elsif rising_edge (CLK) and D = '1' then
                ans_MAGN := sqrt (real (conv_integer (signed (RERSP))) · real
(conv_integer (signed (RERSP))) + real (conv_integer (signed (IMRSP))) · real (conv_integer
(signed (IMRSP))));
                if ans_MAGN = 0.0 then ans_MAGN := 0.01;
                end if;
            FirstPHASE := arctan (real (ans_IMO), real (ans_REO));
                SecondPHASE := arctan (real (conv_integer (signed (IMRSP))), real
(conv_integer (signed (RERSP))));
```

```
                        PHASE <= SecondPHASE-FirstPHASE;
                        if SecondPHASE-FirstPHASE > PI then PHASE <= SecondPHASE-
FirstPHASE–2.0 · PI;
                        else PHASE <= SecondPHASE-FirstPHASE;
                        end if;
                        if SecondPHASE-FirstPHASE < (-PI) then PHASE <= SecondPHASE-
FirstPHASE + 2.0 · PI;
                        end if;
                    MAGN <= integer (ans_MAGN);
                        LOGMAGN <= log10 (ans_MAGN / 1000.0) · 20.0;
                    end if;
                    wait on CLK, RST;
                end process;
            end sqrt_xx_yy;
```

# ВИСНОВКИ

Отже, на початку дослідження дано загальну характеристику CORDIC-алгоритмів. Зроблено огляд властивостей алгоритму CORDIC та наголошено, що даний алгоритм використовується для швидкого обчислення елементарних функцій, таких як множення, ділення тригонометричних та логарифмічних функцій, такі як перетворення прямокутної координати в полярні координати і навпаки. Приведемо опис математичної сутності даного алгоритму та зроблено висновок про те, що алгоритм CORDIC може бути використаний в двох режимах – «поворот» і «вектор».

Окремо досліджено головні області використання досліджуваного алгоритму та перелічено основні підходи до його класифікації.

В практичній частині роботи зроблено аналіз можливостей мови VHDL та названо її функціональні можливості. Коротко описано її походження та проаналізовано способи використання на практиці.

Досліджено та проаналізовано архітектурно-структурну організацію ПЛІС. Розглянуті сучасні напрямки розвитку ПЛІС-технологій. Окреслено актуальність та проблеми створення обчислювальних пристроїв на їхній основі. Проаналізовано засоби та методи, архітектурно-структурні рішення для вирішення проблеми моніторингу об’єктів критичного застосування на основі використання ПЛІС для реалізації таких систем.

Зроблено розробку модулю обчислення функції arctg y / x та sqrt ($x^2+y^2$) з плаваючою комою. Приведено програмні коди на мові VHDL для досліджуваних функцій.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Андреев А. Е. Прогнозирование производительности при реализации алгоритмов на гибридных архитектурах с сопроцессорами // Современные проблемы науки и образования. – 2012. № 3. – С. 36–43.
2. Бікташева С. Р. CORDIC-метод обчислення квадратного кореня / С. Р. Бікташева, Л. В. Мороз, М. Ю. Стахів // Вісн. Нац. Ун-ту» Львівська політехніка». Сер.: Електроніка: [зб. наук. пр.] Львів: Вид-во Нац. ун-ту» Львів. політехніка», – 2006. – С. 152–155.
3. Булкин В. И. Разработка метода построения нечеткой базы знаний с использованием ассоциативно-логических преобразователей / В. И. Булкин // Проблеми інформаційних технологій. – Херсон: ХНТУ, – 2016. №1 (019). – С. 57–64.
4. Вашкевич М. И. Проектирование вычислительных средств с динамически реконфигурируемой архитектурой. Лабораторный практикум: пособие / Минск: БГУИР, М. И. Вашкевич. 2019. – 64 с.
5. Герасименко Ю. А. Разработка криптографического модуля для защиты встроенного программного обеспечения программируемых логических интегральных схем (ПЛИС) / Ю. А. Герасименко. С. – Пет. – 2015. – С. 50–63.
6. Джамал Х. М. Алгоритм і структура модуля для обчислення квадратного кореня у ПЛІС / А. Сергієнко, П. Сергієнко // Праці міжнародної конференції «Безпека, Відмовостійкість, Інтелект», 2018. – С. 74–77.
7. Дунець Р. Б. Проблеми побудови частково реконфігурованих систем на ПЛІС / Р. Б. Дунець, Д. Я. Тиханський // Радіоелектронні і комп’ютерні системи. – 2010. № 7 (48). – С. 200–204.
8. Зотов В. Разработка базовых компонентов цифровых устройств, реализуемых на базе ПЛИС FPGA фирмы Xilinx, с помощью генератора параметризированных модулей CORE Generator // Компоненты и технологии, №2, – 2008, – С. 49–56.

9. Казимир В. В. Проектування комп’ютерних систем на основі мікросхем програмованої логіки: монографія / С. А. Іванець, Ю. О. Зубань, В. В. Казимир. – Суми: Сумський державний університет, 2013. – 313 с.
10. Каршенбойм И. SystemVerilog и улучшение отладки проектов / И. Каршенбойм // Компоненты и технологии. № 1. – 2010. – С. 132–136.
11. Ключеня В. В. Быстрое прототипирование встраиваемых программируемых систем на ПЛИС для мультимедийных приложений / В. В. Ключеня, Н. А. Петровский // Информатика. – 2015. № 47. – С. 13–28.
12. Краснобаєв В. А. Метод підвищення достовірності контролю даних, представлених у системі залишкових класів / В. А. Краснобаєв // Кибернетика и системний аналіз. – 2014. – Т. 50, №6. – С. 167–175.
13. Методи обробки даних на ПЛІС. [Електронний ресурс]. – 2015. Режим доступу: http: // ela. kpi. ua / bitstream / 123456789 / 19096 / 1 / Klymenko_aref. pdf. – Дата доступу: 27.06.2020.
14. Николайчук Я. М. Проблеми реорганізації структури процесорів у різних теоретико-числових базисах / Я. М. Николайчук // Збірник матеріалів міжнародної наукової координаційної наради «Інформаційні проблеми комп’ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління» (ICSM–2014). – Тернопіль, 2014. – С. 110–114.
15. Патерак В. О. Детектування фронтів сигналу за допомогою ПЛІС / Патерак В. О. // Актуальні задачі сучасних технологій». – Т.: ТНТУ, 2016. – Т. 2. – С. 90–91.
16. Петровский Н. А. Многомерный CORDIC алгоритм кватернионов с «разреженными» итерациями / Н. А. Петровский, М. Парфенюк // Материалы 15-й международной конференции «Цифровая обработка сигналов и её применение» (DSPA'2013). – Т. 2. – М.: 2013. – С. 206–210.
17. Петровский Н. А. Рекурсивный процессор умножителя кватернионов со структурной CORDIC-лестничной параметризацией / Н. А. Петровский, А. В.

Станкевич // Тезисы докладов международной научной конференции ИТС’2013 / БГУИР. – Минск, Беларусь: 2013. – С. 196–197.

18. Ратушний П. М. ПЛІС та їх програмування: лабораторний практикум / Ратушний П. М., Жагловська О. М., Огородник К. В. – Вінниця: ВНТУ, 2018. – 57 с.
19. Рудницький М. В. Класифікація реконфігурованих обчислювальних систем / М. В. Рудницький, І. А. Клименко // Вісник Вінницького політехнічного інституту. – 2014. – № 5. – С. 1–4.
20. Сергієнко А. М. Реалізація функції квадратного кореня у ПЛІС / П. А. Сергієнко // Вісник НТУУ «КПІ»: Інформатика, управління та обчислювальна техніка: [зб. наук. пр.] Київ. – 2014. Т. 60, – С. 40–45.
21. Токарев В. А., Хлуденев А. В. CORDIC-алгоритм. Оценка эффективности алгоритмов цифровой обработки сигналов при конвейерной реализации. Огарев on-line. Электронное периодическое издание для студентов и аспирантов, – 2015. № 10. – С. 1–4.
22. Хачумов М. В. Реализация алгоритмов навигации и управления в бортовых вычислительных комплексах летательных аппаратов // Программные системы: теория и приложения, – 2016, № 7:2 (29). – С. 35–59.
23. Цифрова схемотехніка. Частина 2. Електронні пристрої і системи: навчальний посібник / Й. Й. Білинський. – Вінниця: ВНТУ, 2016. – 171 с.
24. Яцків В. В. Контроль виконання арифметичних операцій на основі модулярних кодів / В. В. Яцків // Вимірювальна та обчислювальна техніка в технологічних процесах. – 2015. № 4 (53). – С. 135–138.
25. Яцків В. В. Концепція побудови безпровідних сенсорних мереж на основі колективного інтелекту / В. В. Яцків, Н. Г. Яцків // Вимірювальна та обчислювальна техніка в технологічних процесах. – 2015. № 2. – С. 217–221.
26. Bergeron, J. Verification Methodology Manual for SystemVerilog / J. Bergeron, E. Cerny, A. Hunter, A. Nightingale. Springer, – 2005. – 510 p.

27. Gadekar S. R. CORDIC Algorithm for WLAN. Int. Journal of Engineering Research and Applications, Vol. 5, Issue 8, (Part 2) August 2015, – Pp. 1–4. URL: http: // www. ijera. com / papers / Vol5_issue8 / Part% 20-%202 / A58020104. pdf. Дата звернення – 22.05.20.
28. Godbole B. B., Nikam R. H. FPGA implementation of CORDIC algorithm used in DDS based modulators. – International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 1, January 2015, – pp. 94–97. URL: http: // www. ijarcce. com / upload / 2015 / january / IJARCCE2D. pdf. Дата звернення – 27.06.20.
29. Meher P. CORDIC designs for fixed angle of rotation / P. Meher, S. Park // IEEE Trans. VLSI Syst. – 2013. – Vol. 21, №2. – P. 217–228.
30. Petrovsky N. Pipelined embedded processor of quaternionic m-band wavelets for image multiresolution analysis / N. Petrovsky, M. Parfieniuk, A. Petrovsky // 2013 2nd Mediterranean Conference on Embedded Computing (MECO). – Budva, Monternegro. – 2013. – P. 196–199.
31. Sergiyenko A. M. Square root calculations in FPGA / H. M. Jamal., P. A. Sergiyenko // System analysis and information technology: 20-th International conference SAIT 2018, Kyiv, Ukraine, May 21–24, 2018. Proceedings. ESC «IASA» NTUU «Igor Sikorsky Kyiv Polytechnic Institute», – 2018. – P. 163–164.
32. Szyzaki M. FPGA computation of magnitude of complex numbers using modified CORDIC algorithm / Smyk R. // Zeszyty Naukowe Wydzialu Elektrotechniki i Automatyki Politechniki Gdanskiej. – 2015. № 47. – P. 35–38.
33. Parhami B. Computer Arithmetic: Algorithms and Hardware Designs. Oxford University Press, 2010. – 641 p.
34. Yoshikawaa K. Development of Fixed-point Square Root Operation for High- level Synthesis / N. Iwanagaa, A. Yamawaki // Proc. 2nd Int. Conf. on Industrial Application Engineering. – 2014. – P. 16–20.
35. VHDL Test Bench Tutorial. University of Pennsylvania, Department of Electrical and Systems Engineering. – 2012. – 210 p.

36. Volder J. E. The birth of CORDIC // Journal of VLSI Signal Processing Systems, Vol. 25, № 2. – 2015. – P. 101–105.
37. Walther J. S. The story of unified CORDIC // Journal of VLSI Signal Processing Systems, Vol. 25, № 2. – 2016. – P. 107–112.

**ДОДАТОК 1**

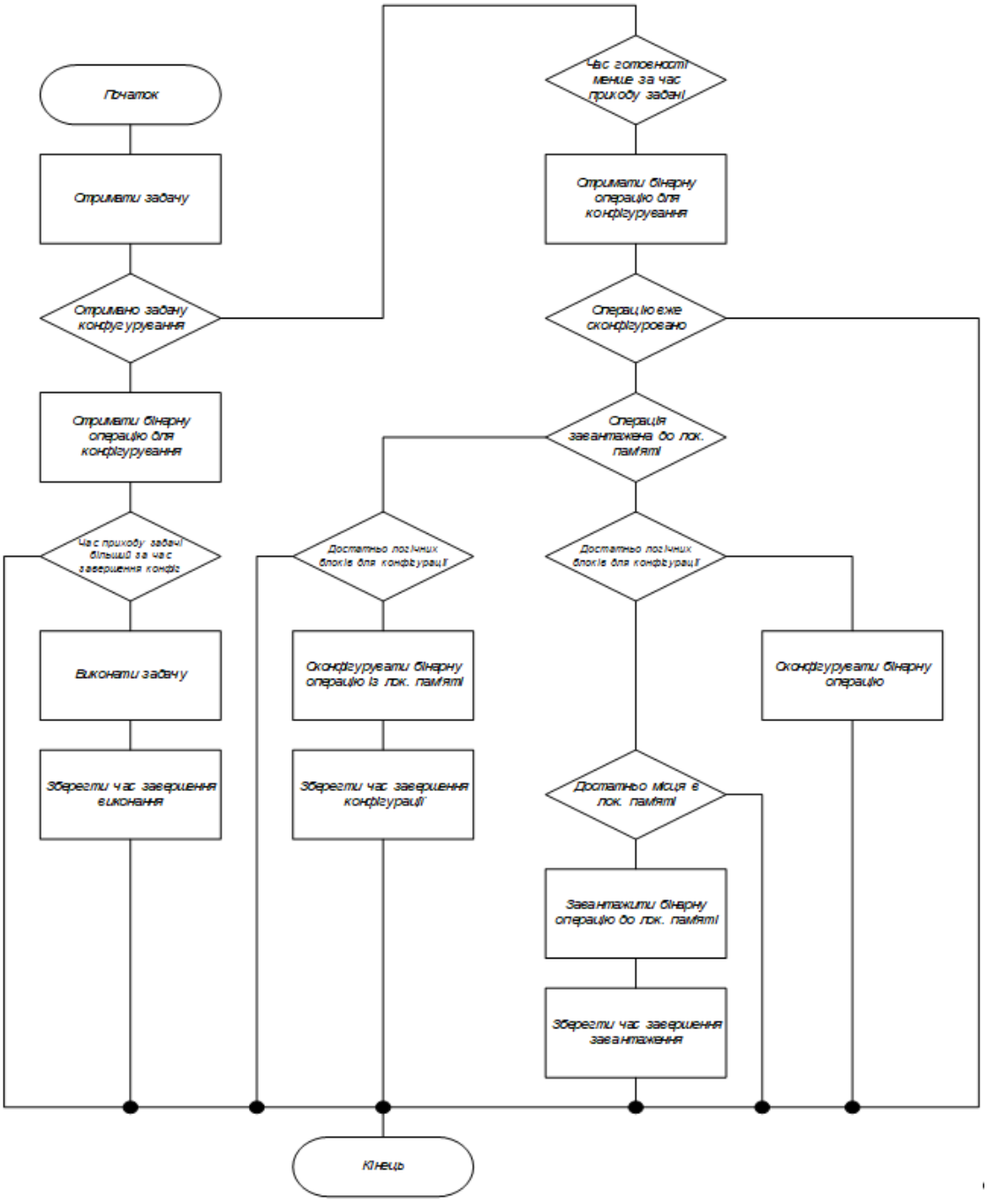
# Принципова схема алгоритму моделювання роботи ПЛІС

ІАЛЦ.467449.002 Д1

Аркушів 1

Київ 2021 р.

Принципова схема алгоритму моделювання роботи ПЛІС




| Зм. | Арк | № документа | Підп. | Дата | ІАЛЦ.467449.002Д1 | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Липай В.С. | | | Принципова схема алгоритму моделювання роботи ПЛІС | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Сергієнко А.М. | | | | | |
| Н.контр. | | Сімоненко В. П. | | | | НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-72 | |
| Затверд. | | Стіренко С. Г. | | | | | |

**ДОДАТОК 2**

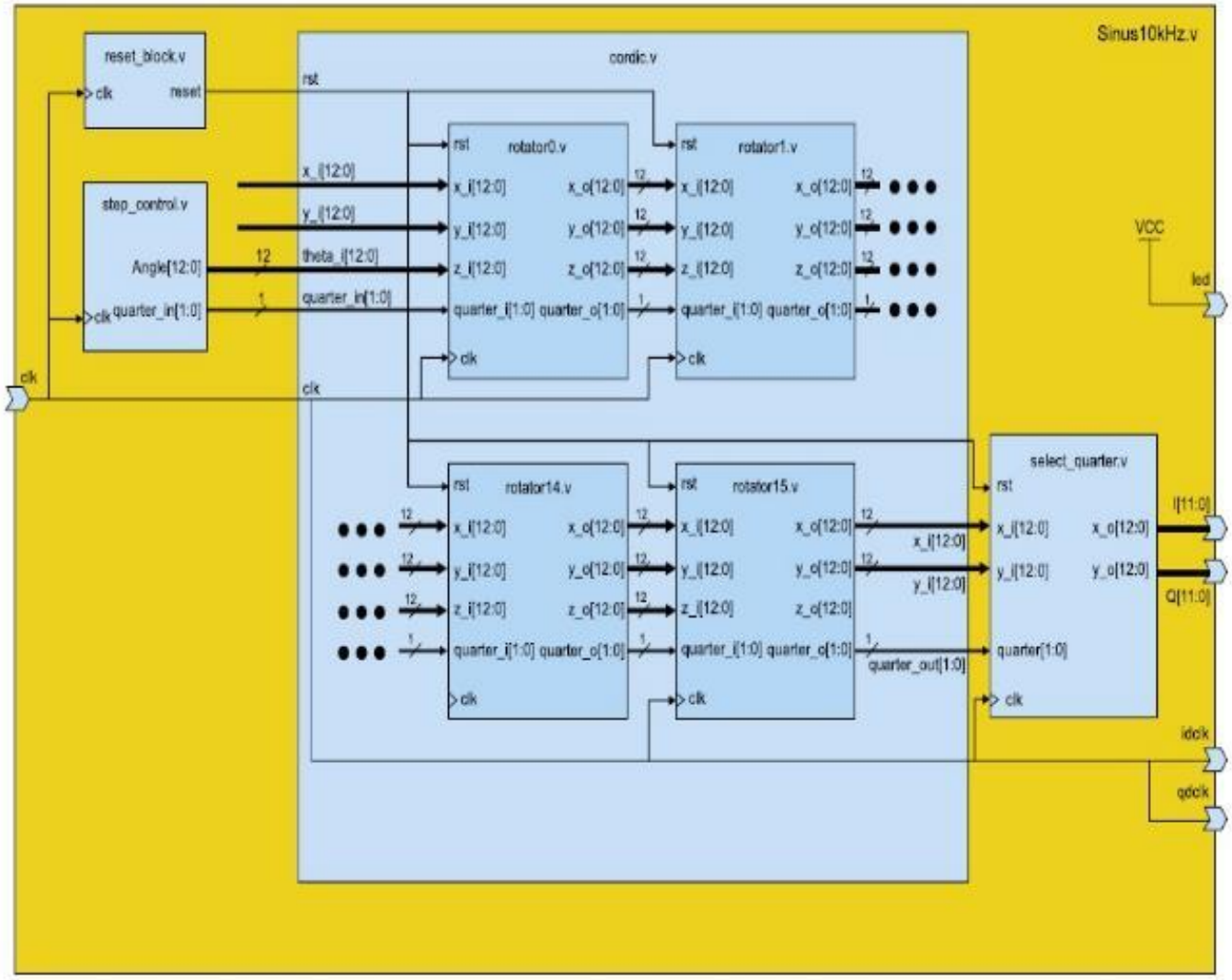
# Структурна схема

# ІАЛЦ.467449.003 Д2

Аркушів 1

Київ 2021 р.

Структурна схема




| Зм. | Арк | № документа | Підп. | Дата | ІАЛЦ.467449.003 Д2 | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Липай В.С.. | | | Структурна схема | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Сергієнко А.М. | | | | | |
| Н.контр. | | Сімоненко В. П. | | | | НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-72 | |
| Затверд. | | Стіренко С. Г. | | | | | |

**ДОДАТОК З**

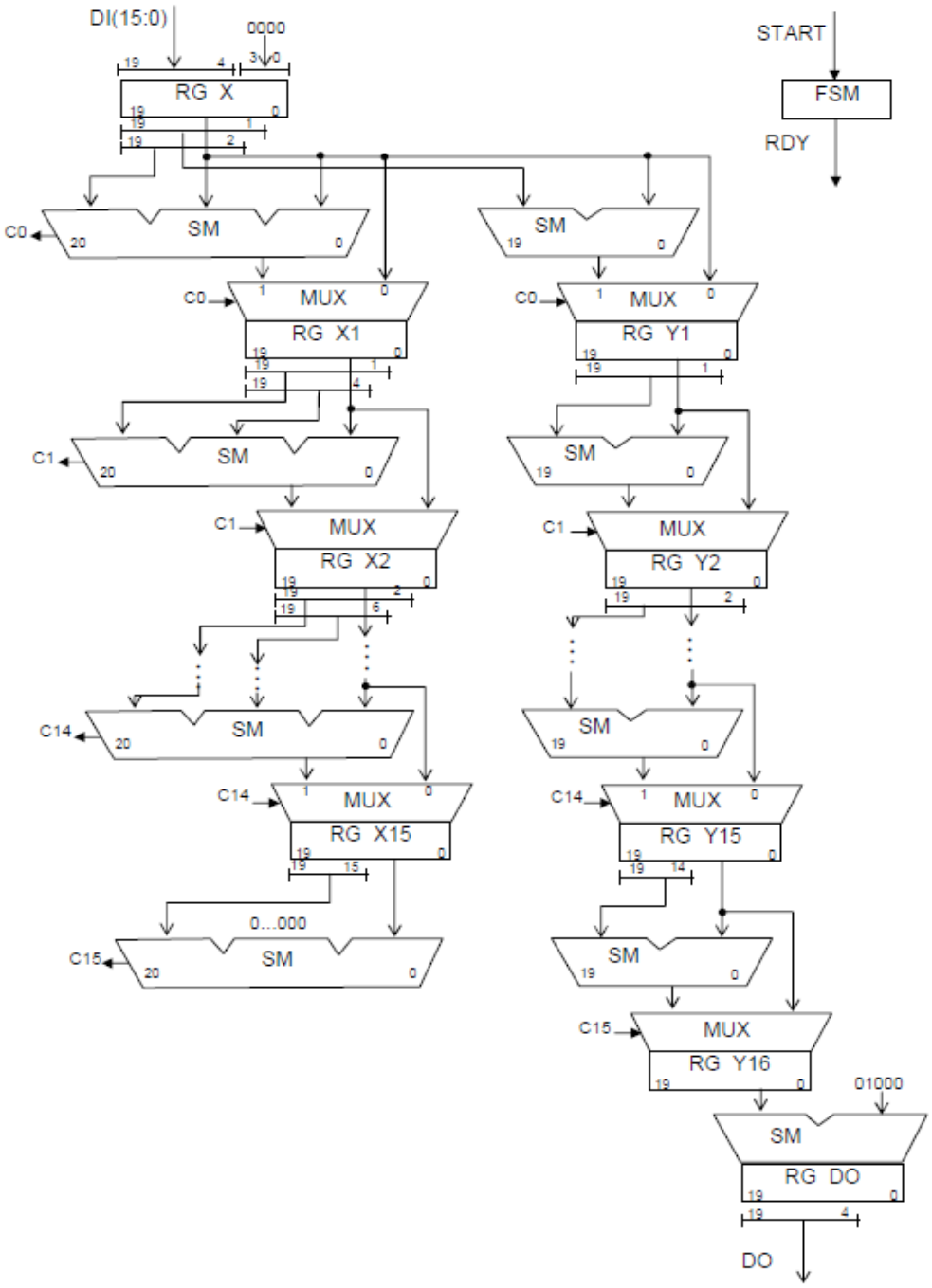
# Функціональна схема алгоритму обчислення функцій sin ϕ, cos ϕ

ІАЛЦ.467449.005 Д4

Аркушів 1

Київ 2021 р.

Функціональна схема алгоритму обчислення функцій sin ϕ, cos ϕ




| Зм. | Арк. | № документа | Підп. | Дата | ІАЛЦ.467449.005 Д4 | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Липай В.С. | | | Функціональна схема алгоритму обчислення функцій sin ϕ, cos ϕ | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Сергієнко А.М. | | | | НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-72 | |
| Н.контр. | | Сімоненко В. П. | | | | | |
| Затверд. | | Стіренко С. Г. | | | | | |

**ДОДАТОК 4**

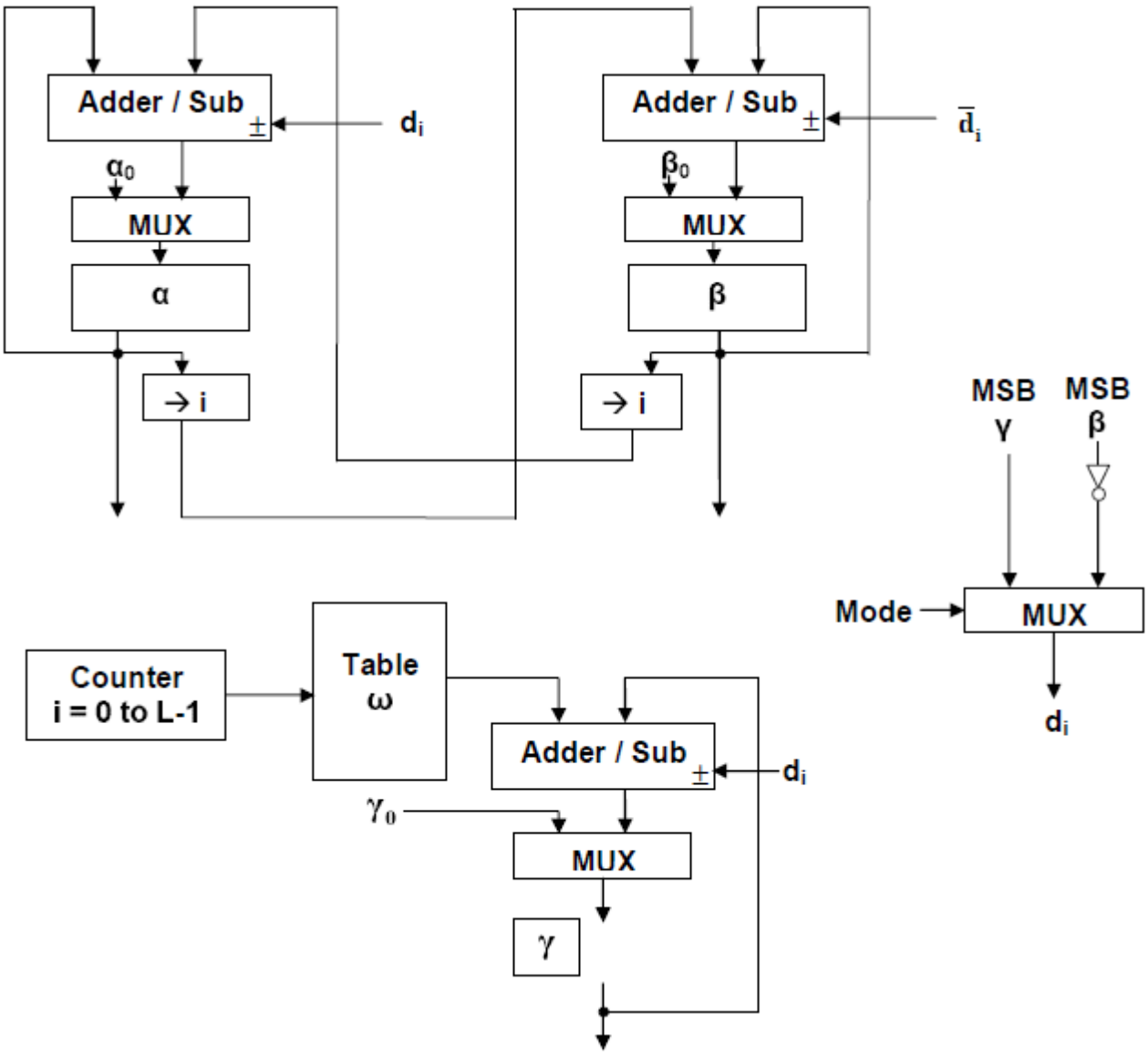
# Схематичне зображення архітектури CORDIC

ІАЛЦ.467449.005 Д4

Аркушів 1

Київ 2021 р.

**Схематичне зображення архітектури CORDIC**




| | | | | | ІАЛЦ.467449.006 Д5 | | |
|---|---|---|---|---|---|---|---|
| Зм. | Арк. | № документа | Підп. | Дата | | | |
| Розробив | | Липай В.С. | | | Схематичне зображення архітектури CORDIC | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Сергієнко А.М. | | | | | |
| Н.контр. | | Сімоненко В. П. | | | | НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-72 | |
| Затверд. | | Стіренко С. Г. | | | | | |

**ДОДАТОК 5**

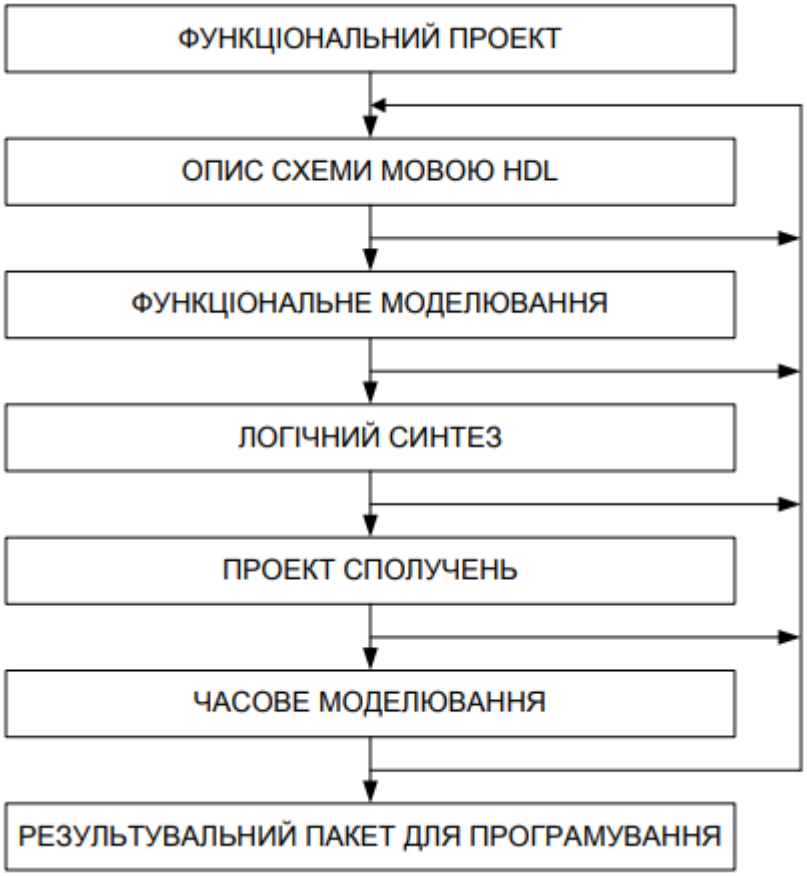
# Приклад процесу проектування програмованих схем

ІАЛЦ.467449.005 Д4

Аркушів 1

Київ 2021 р.

**Приклад процесу проектування програмованих схем**




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467449.006 Д5 | | |
| Зм. | Арк. | № документа | Підп. | Дата | | | |
| Розробив | | Липай В.С. | | | **Приклад процесу проектування програмованих схем** | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Сергієнко А.М.. | | | | | |
| Н.контр. | | Сімоненко В. П. | | | | НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-72 | |
| Затверд. | | Стіренко С. Г. | | | | | |