

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки
Факультет комп'ютерних наук та технологій
(повне найменування інституту, факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОДУЛЯ ОБЧИСЛЕНЬ З РУХОМОЮ КОМОЮ НА FPGA

Виконав: студент(ка) 4 курсу, групи КНТ-527

Спеціальності 123 «Комп'ютерна інженерія»
(код і найменування спеціальності)

Освітня програма (спеціалізація)
«Комп'ютерна інженерія»

Діденко Артем Євгенович

(прізвище та ініціали)

Керівник Зеленьова І.Я.

(прізвище та ініціали)

Рецензент Куляба-Харитоновна Т.І.

(прізвище та ініціали)

м. Запоріжжя
2021 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування вищого навчального закладу)

Інститут, факультет інформатики та радіоелектроніки, комп'ютерних наук і технологій
 Кафедра «Комп'ютерні системи та мережі»
 Ступінь вищої освіти (освітній ступінь) бакалаврський
 Спеціальність 123 Комп'ютерна інженерія
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерна інженерія
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри Кудерметов Р.К.

“ ” 2021 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА

Діденка Артема Євгеновича
 (прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка модуля обчислень з рухомою комою на FPGA

керівник проекту (роботи) Зеленьова Ірина Яківна, к. т. н., доцент
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “17” березня 2021 року № 81

2. Строк подання студентом проекту (роботи) 06 травня 2021 року

3. Вихідні дані до проекту (роботи) наявні алгоритми арифметичних операцій з рухомою комою у двійковому коді

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз предметної області ;

2) Розробка функціональних вузлів модуля обчислень з рухомою комою;

3) Тестування модуля обчислень з рухомою комою та аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Плакат 1 – Схема модуля обчислень з рухомою комою;

Плакат 2 – Схема блоку складання і віднімання;

Плакат 3 – Схема блоку множення;

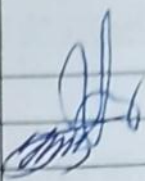
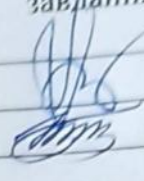
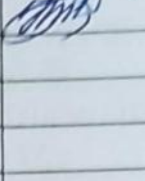
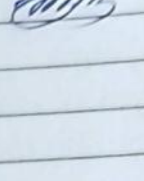
Плакат 4 – Результати моделювання і тестування модуля обчислень з рухомою комою;

Креслення 1 – Блок схема алгоритму складання і віднімання;

Креслення 2 – Блок схема алгоритму множення ;

Креслення 3 – Блок-схема алгоритму ділення.

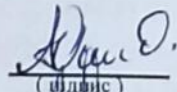
6. Консультанти розділів проекту (роботи)

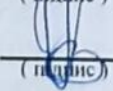
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	Зеленьова І.Я., к. т. н., доцент		
Нормоконтроль	Щербак Н.В., ст. викл.		

7. Дата видачі завдання 01.03.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз наявних алгоритмів арифметичних операцій, огляд аналогів	01.04.2021 р.	
2	Розробка функціональних вузлів модуля обчислень з рухомою комою	05.04.2021 р.	
3	Розробка опису модуля обчислень з рухомою комою на мові VHDL	15.04.2021 р.	
4	Моделювання модуля обчислень з рухомою комою	20.04.2021 р.	
5	Розробка випробувального стенду	25.04.2021 р.	
6	Тестування модуля обчислень з рухомою комою на FPGA	30.04.2021 р.	
7	Оформлення отриманих результатів у ПЗ	01.05.2021 р.	
8	Оформлення графічного матеріалу	10.05.2021 р.	
9	Оформлення допоміжного матеріалу	17.05.2021 р.	

Студент  А.Є. Діденко
(підпис) (ініціали та прізвище)

Керівник проєкту (роботи)  І.Я. Зеленьова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

ПЗ: 80 с., 61 рис., 4 табл., 35 джерел.

КОНВЕЕРИЗАЦІЯ, МОДУЛЬ ОБЧИСЛЕНЬ, ПОМНОЖУВАЧ, СУМАТОР, ЧИСЛО З РУХОМОЮ КОМОЮ, FPGA

Об'єкт розробки – модуль обчислення з рухомою комою.

Метою роботи є розробка модуля обчислення з рухомою, проведення моделювання в системі Modelsim, реалізація та тестування на мікросхемі FPGA, аналіз отриманих результатів.

Проект складається з трьох розділів.

У першому розділі описані основні поняття предметної області. Проведений детальний опис методів та алгоритмів, які використовуються в розробці модуля обчислень, виявлено їх переваги та недоліки.

У другому розділі описано основні алгоритми роботи функціональних блоків модуля обчислень. Розглянута розроблена структура блоків модуля обчислень та пояснено принципи роботи їх складових частин.

У третьому розділі описано процес проведення моделювання і тестування модуля обчислень та досліджено отримані результати.

Результатом розробки є реалізований на FPGA Cyclone IV модуль обчислень з рухомою комою, що виконує операції складання, віднімання та множення та представляє результат згідно зі стандартом IEEE-754.

ЗМІСТ

Вступ.....	6
1 Опис предметної області	7
1.1 Основні характеристики і використання чисел з рухомою комою.....	7
1.2 Структура, основні характеристики і застосування FPGA.....	12
1.3 Огляд видів суматорів.....	17
1.4 Огляд видів помножувачів	24
1.5 Огляд видів дільників	28
1.6 Огляд існуючих аналогів	30
1.7 Постановка задачі розробки.....	33
2 Розробка модуля обчислень з рухомою комою	33
2.1 Розробка структури модуля обчислень з рухомою комою	33
2.2 Розробка блоку складання та віднімання.....	35
2.2.1 Алгоритм і структура блоку складання та віднімання.....	35
2.2.2 Розробка функціональних складових блоку складання та віднімання.....	40
2.2.3 Опис блоку складання та віднімання на мові VHDL	51
2.3 Розробка блоку множення.....	53
2.3.1 Алгоритм і структура блоку множення чисел з рухомою комою.....	53
2.3.2 Розробка функціональних складових блоку множення	58
2.3.3 Опис блоку множення на мові VHDL	63
2.4 Розробка блоку ділення	65
3 Моделювання і тестування модуля обчислень з рухомою комою.....	68
3.1 Моделювання модуля обчислень з рухомою комою.....	68
3.2 Тестування модуля обчислень з рухомою комою	71
3.3 Аналіз результатів моделювання і тестування модуля обчислень	74
Висновки	76
Перелік джерел посилання	77

ВСТУП

Обчислювальна техніка відіграє важливу роль у вирішенні ряду математичних, фізичних, економічних задач. З ростом складності задач зростають і такі вимоги до обчислювальної техніки, як швидкість роботи, продуктивність, точність розрахунків. На даний момент неможливо представити аналіз великих об'ємів даних та обробку масивів інформації без залучення комп'ютерів або інших подібних обчислювальних пристроїв.

Окрім процесорів загального призначення, широкої популярності набувають і такі види цифрових пристроїв, як FPGA. На відміну від звичайних процесорів, які виконують команди послідовно, FPGA може бути використана як засіб для паралельної обробки даних. Більш того, FPGA дозволяє реалізовувати цифрові пристрої різноманітної структури, а їх поєднання з процесорами загального призначення розширює можливості останніх, що робить розробку більш гнучкою.

Числа з рухомою комою широко застосовуються в наукових дослідженнях. Вони поєднують у собі достатню точність результату та великий діапазон можливих значень. Їх використання відрізняється швидкістю та простотою проведення обчислень.

Застосування FPGA в таких сферах, як цифрова обробка сигналів, нейронні мережі та аналіз даних потребує наявності засобів обчислення, що виконують операції з достатньою швидкістю і точністю. Саме цим пояснюється актуальність розробки модуля обчислень з рухомою комою.

На даний момент досліджено багато алгоритмів проведення арифметичних дій над числами з рухомою комою, реалізовано безліч цифрових пристроїв, що в основі роботи яких лежать ці алгоритми. Розробка модуля обчислень з рухомою комою покаже основні принципи реалізації подібних пристроїв та виявить можливі шляхи для подальших досліджень і покращень.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні характеристики і використання чисел з рухомою комою

Число з рухомою комою, яке також називають експоненціальним записом числа, є представленням дійсного числа у вигляді мантиси та порядку (експоненти). Назва такого представлення числа відображає той факт, що кома, яка відділяє цілу частину від дробової, може бути розміщена де завгодно у числі, на відміну від числа з фіксованою комою, де кома може знаходитися тільки у попередньо визначеному місці.

Стандартний вигляд числа з рухомою комою представляється наступною формулою [1]:

$$y = (-1)^S \times M \times b^E, \quad (1.1)$$

де S – знак числа, $S \in \{0,1\}$;

M – мантиса числа;

b – основа системи числення;

E – експонента числа.

Формула (1.1) визначає запис числа з рухомою комою у так званому «sign-and-magnitude» форматі.

Проблема запису чисел з рухомою комою полягає в тому, що одне й те саме число можна записати багатьма способами. Так, наприклад числа 123.4, $12.34 \cdot 10$, $1.234 \cdot 10^2$ та $0.1234 \cdot 10^3$ позначають одну й ту саму величину, але записуються по-різному. Така властивість чисел з рухомою комою утворює надлишковість, що може супроводжуватися складністю обчислень. Для уникнення даної проблеми користуються нормалізованими числами з рухомою комою.

Нормалізоване число з рухомою комою має властивість, що виражається наступною формулою:

$$1 \leq M < b \quad (1.2)$$

З формули видно, що нормалізоване число має мантису, яка не перевищує значення основи системи числення. Представляючи число у нормалізованому вигляді, спрощується його реалізація та з'являється можливість зберегти один біт при двійковому представленні числа [2].

У випадках, що не задовольняють формулу (1.2), числа називаються денормалізованими. Денормалізовані числа, як буде видно згодом, дозволяють розширити діапазон чисел з рухомою крапкою, але в даній роботі не розглядаються через складність в реалізації.

Діапазони $[-max, -min]$ та $[min, max]$ чисел з рухомою комою задаються наступними рівняннями:

$$\min = M_{\min} \cdot b^{E_{\min}}, \max = M_{\max} \cdot b^{E_{\max}} \quad (1.3)$$

На рисунку 1.1 зображено діапазони чисел з рухомою комою на числовій прямій. З рисунку видно, що числа з рухомою комою займають не всю числову пряму, не доходячи до значень, що знаходяться біля нуля, та не перевищуючи максимального значення.

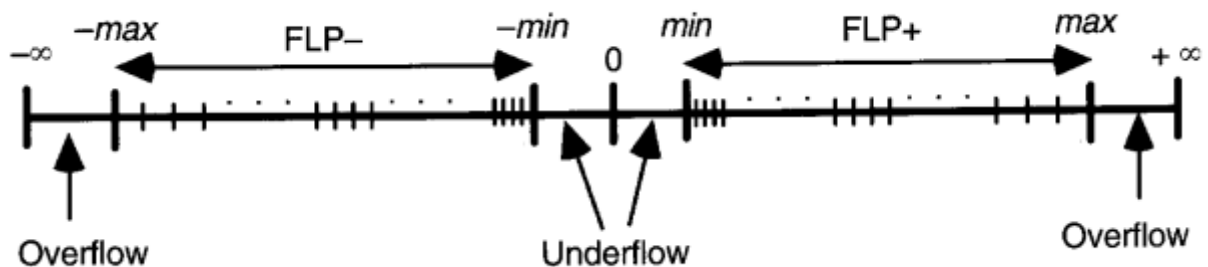


Рисунок 1.1 – Представлення чисел з рухомою комою на числовій прямій [3].

У випадках, коли число потрапляє в один з недозволених діапазонів, потрібно виявити, яке саме значення прийняло число та коректно його

відобразити. Виняткові ситуації будуть розглянуті в подальшому у контексті двійкових чисел з рухомою комою у стандарті IEEE-754.

З рисунку 1.1 видно також, що відстань між сусідніми нормалізованими числами зростає при віддаленні від нуля. Ця відстань збільшується на коефіцієнт, що дорівнює значенню основи системи числення кожного разу, як збільшується експонента і залишається незмінною до наступного значення експоненти [4].

Після кожної операції результат округлюється за певним алгоритмом з метою отримання найбільш коректного значення. Методи округлення поділяються на ті, що відкидають зайву частину числа без зміни основної та ті, що модифікують останню. До перших відносяться наступні методи: звичайне відкидання (truncation), округлення до позитивної або негативної нескінченності, округлення до або від нуля. До других відноситься метод округлення до найближчого парного або непарного числа. При виборі методу округлення звертають увагу на його точність та складність реалізації.

Найпростішим методом округлення є метод відкидання, при якому нехтують частиною числа, що непотрібна для подальших обчислень або не може бути записана певною кількістю біт. При округленні до позитивної або негативної нескінченності обирають найбільше число, що не перевищує задане або найменше число, що не менше заданого відповідно. Метод округлення до нуля представляє собою округлення до позитивної нескінченності від'ємних чисел та округлення до негативної нескінченності додатних чисел. Округлення найближчого парного числа відбувається наступним чином: якщо значення, що відкидається, дорівнює половині наступного розряду, то цей розряд збільшується у тому випадку, якщо число непарне. Метод округлення до найближчого непарного числа такий самий.

Кожен метод округлення характеризується двома величинами: середньою помилкою округлення та середнім зміщенням. Згідно з [5] середні відносні помилки представлення *ARRE* для методів округлення з відкиданням та методів округлення з модифікацією основної частини визначаються відповідно формулами:

$$ARRE(t, \beta) = \frac{\beta^{-t}(\beta - 1)}{2 \ln \beta} \quad (1.4)$$

$$ARRE(t, \beta) = \frac{\beta^{-t}(\beta - 1)}{2 \ln \beta}, \quad (1.5)$$

де t – кількість розрядів числа;
 β – основа системи числення.

З формул (1.4) та (1.5) видно, що метод округлення до найближчого парного або непарного має меншу помилку ніж метод округлення з відкиданням та подібні.

В той час, коли середня відносна помилка представлення визначає величину похибки округлення, середнє зміщення визначає тенденцію похибки приймати той чи інший знак. Для округлення до найближчого парного або непарного середнє зміщення дорівнює нулю [5]. Таким чином, використовуючи ці методи округлення можна позбутися зміщення вздовж числової прямої. Метод округлення до найближчого парного (rounding to nearest, ties to even) є стандартним методом округлення у стандарті IEEE-754.

Стандарт IEEE-754 – стандарт, що визначає формати та методи арифметики з рухомою комою в стандартних та розширених функціях комп'ютерних систем з одинарною, подвійною, розширеною та розширюваною точністю, а також формати для обміну даними. В цьому стандарті визначено умови виникнення виняткових ситуацій та визначено правила обробку цих умов [6].

Стандарт IEEE-754 визначає також запис числа з рухомою комою в двійковому вигляді. На рисунку 1.2 зображено вигляд числа з рухомою комою з одинарною точністю. В даній роботі розглядається модуль обчислення чисел з одинарною точністю.

S	E (експонента) 8 біт	M (мантиса) 23 біт
---	----------------------	--------------------

Рисунок 1.2 – Формат числа з рухомою крапкою у стандарті IEEE-754

Число одинарної точності має довжину 32 біт та складається з полів знаку, експоненти та мантиси, що мають довжину в один, вісім та 23 біт відповідно. Числа одинарної точності згідно зі стандартом можна представити формулою:

$$y = (-1)^S \times 1.M \times 2^{E-127} \quad (1.6)$$

Порівнюючи формули (1.1) та (1.6) можна помітити деякі відмінності. По-перше, мантиса числа з рухомою комою у стандарті IEEE-754 має на початку біт, що завжди дорівнює одиниці. Таким чином досягається нормалізація числа. По-друге, експонента представляється у вигляді $E-127$, що дозволяє позбутися від'ємних значень у полі експоненти. Таким чином полегшується операції порівняння експонент.

Як вже було зазначено, стандарт IEEE-754 визначає умови виникнення виняткових ситуацій та правила їх обробки. Даними винятковими ситуаціями є так звані overflow та underflow, нуль, додатна та від'ємна нескінченості, NaN (not a number).

Для чисел з одинарною точністю виняткові ситуації визначені при комбінаціях значень експоненти та мантиси, що представлені у таблиці 1.1

Якщо в результаті якоїсь операції експонента числа перевищила максимальне значення або стала менше за мінімальне значення, то число переводиться у нескінченність або денормалізований вигляд. Якщо денормалізовані числа не підтримуються, то число переводиться у нуль.

Таблиця 1.1 – Виняткові значення у форматі IEEE-754

Значення числа	Експонента	Мантиса
Нормалізоване число	1-254	Будь-яка
Денормалізоване число	0	Ненульова
Нуль	0	0
Нескінченність	255	0
NaN	255	Ненульова

Арифметика за рухомою комою невід’ємно пов’язана з цифровою обробкою сигналів. Для того, щоб використовувати перетворення Фур’є та інші операції цифрової обробки сигналів, числа у арифметиці з фіксованою комою повинні мати 24, 32 або 64 біт. Достатньо очевидний вплив такої точності на ціну та складність апаратного та програмного забезпечення. Реалізація чисел з рухомою комою довжиною, наприклад 22 біт має більш економічне використання обладнання та достатню точність для більшості застосувань [7].

Різного роду математичні розрахунки при моделюванні потребують використання чисел з рухомою комою. Так, наприклад при моделюванні системи космічних тіл використовується обчислення на основі чисел з рухомою комою, оскільки астрофізичні змінні можуть досягати великих значень [8].

Операції з рухомою комою широко застосовуються в мультимедії через те, що збільшується кількість програм, які використовують алгоритми з рухомою комою. На даний момент стандарти Міжнародного союзу електрозв’язку використовують тільки числа з рухомою крапкою. Сьогодні майже неможливо знайти жодне програмне забезпечення для розпізнавання голосу, яке б використовувало числа з фіксованою комою [9].

1.2 Структура, основні характеристики і застосування FPGA

FPGA (field programmable gate array) – вентильна матриця, що програмується користувачем. Даний тип інтегральних мікросхем програмується мовою опису апаратури, наприклад VHDL або Verilog.

На рисунку 1.3 представлено типову структуру FPGA.



Рисунок 1.3 – Структура FPGA [10]

З рисунку 1.3 видно, що FPGA представляє собою масив програмованих блоків різного призначення, такого як звичайна логіка, пам'ять та помножувачі, що оточені маршрутними лініями, які також можуть бути запрограмовані. Таким чином можна поєднувати різні блоки один між одним [10].

Для зв'язку мікросхеми з навколишнім середовищем передбачені порти введення виведення, більшість з яких програміст може обирати при створенні того чи іншого пристрою.

В залежності від призначення, виробника або цінового діапазону, мікросхеми FPGA містять різну кількість логічних елементів, блоків пам'яті та помножувачів.

На рисунку 1.4 представлено структуру логічного елементу FPGA фірми Altera сімейства Cyclone 3.

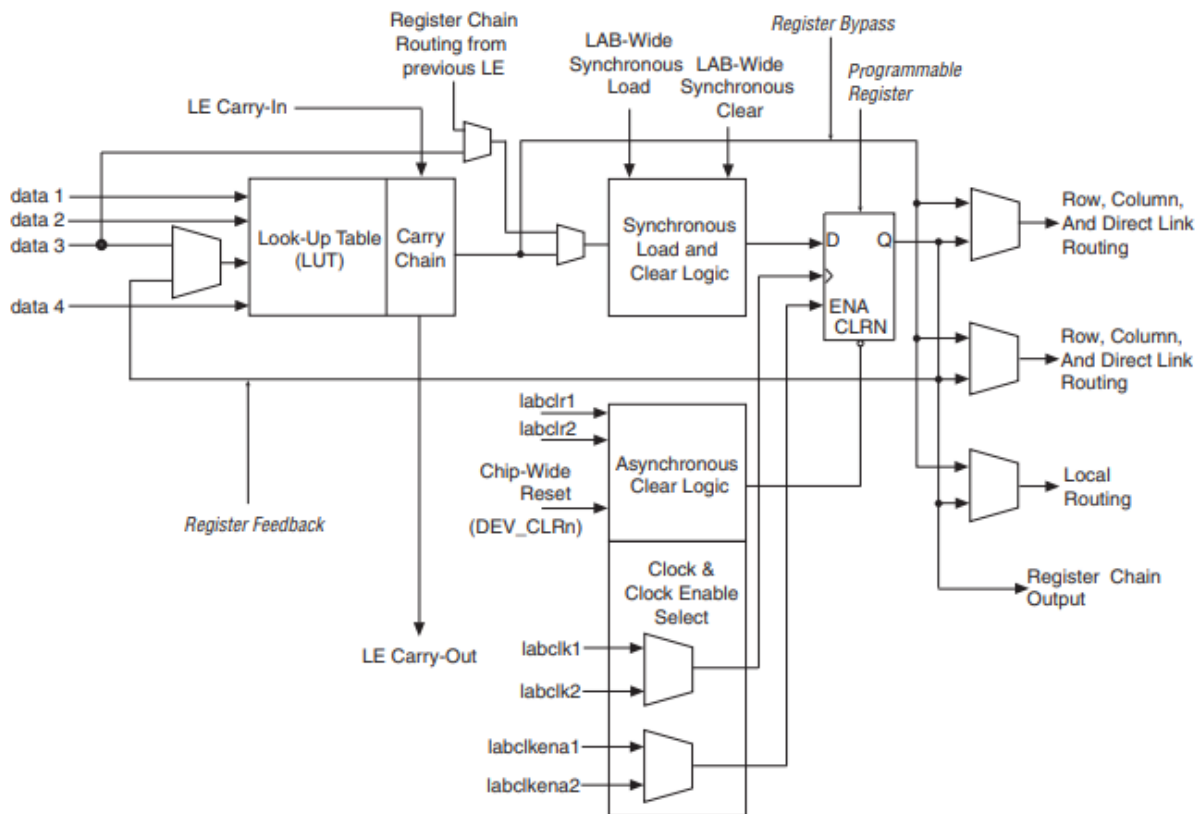


Рисунок 1.4 – Структура логічного елементу FPGA Cyclone 3 [11]

Логічний елемент складається з таких основних компонентів як LUT(look-up table), схема мультиплексування, схема переносу та тригер.

LUT представляє собою таблицю істинності для комбінаційної функції. Значення функції зберігаються у комірках SRAM пам'яті та обираються в залежності від значень вхідних сигналів. На рисунку 1.5 зображено структуру LUT. Видно, що вибір вихідного значення відбувається за допомогою схеми мультиплексування.

Блок пам'яті, що зберігає значення функції представляє собою регістр зсуву. Таким чином, звичайну LUT можна використовувати не тільки для комбінаційної логіки, а й в якості звичайної RAM пам'яті.

Кожний логічний елемент має схему переносу для реалізації арифметичних функцій суматорів, лічильників, акумуляторів та компараторів. При поєднанні з вихідним тригером можна створювати більш складні арифметичні блоки.

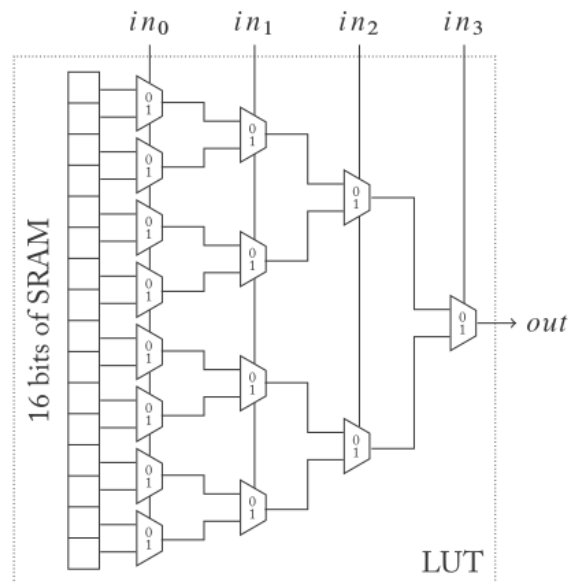


Рисунок 1.5 – Структура LUT [12]

Іншим головним компонентом FPGA є схема маршрутів, які поєднують логічні блоки між собою. Структура маршрутів поділяється на ієрархічну (hierarchical) та так звану острівну (island).

На рисунку 1.6 представлено ієрархічну структуру FPGA Altera Apex 2. З рисунку видно, що основні маршрути поєднуються з так званими MegaLAB, що представляють собою поєднання логічних елементів.

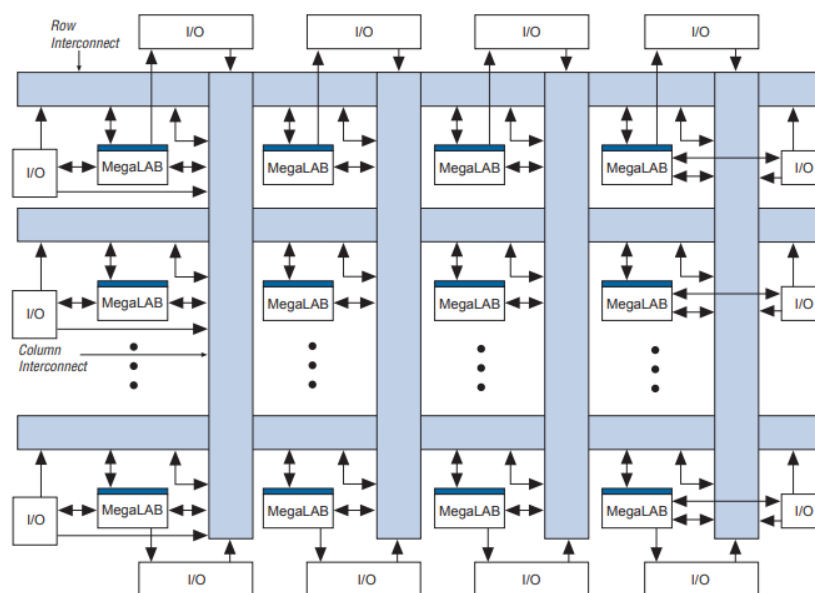


Рисунок 1.6 – Структура маршрутів Altera Apex 2 [13]

Острівна структура зв'язків представлена на рисунку 1.7. При такій структурі логічні блоки оточені лініями зв'язку та поєднуються з ними двома або чотирма виводами.

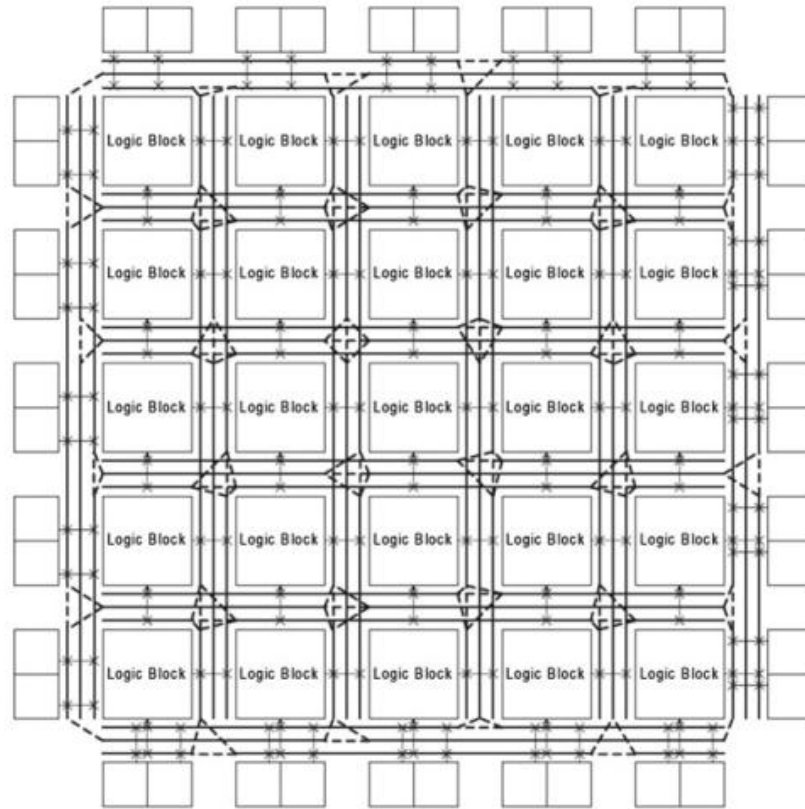


Рисунок 1.7 – Острівна структура маршрутів FPGA [14]

Головними характеристиками FPGA є кількість апаратних ресурсів, швидкість роботи та споживання енергії.

На практиці зазвичай не використовують повністю всі апаратні ресурси. Якщо схема того чи іншого пристрою, що реалізується на FPGA, має багато з'єднань між логічними елементами порівняно з кількістю самих логічних елементів, то частина останніх залишиться невикористаною. Така сама ситуація виникає, якщо логічних елементів в схемі більше за зв'язки між ними. Таким чином, розробнику потрібно обирати таку мікросхему FPGA, щоб використання як логічних елементів так і зв'язків між ними було якомога однаковим, щоб реалізована схема була збалансованою [15].

Споживання енергії в FPGA складається з динамічного та статичного споживання. Статичне споживання пов'язане зі струмом витоку в транзисторах. Динамічне споживання пов'язане з процесами вмикання/вимикання транзисторів та заряджанням/розряджанням конденсаторів. Динамічне споживання енергії залежить як від внутрішньої ємності мікросхеми, так і від напруги живлення та частоти роботи [16].

FPGA знаходять своє використання в надзвичайно різноманітних задачах, таких як сортування та пошук даних, обробка аудіо- та відеосигналів, обробка зображень, криптографія, обробка пакетів даних у мережах, генерація випадкових чисел та емуляція логіки. Серед важливих нових ринків використання можуть бути фінанси, біоінформатика, апаратне забезпечення для радіозв'язку, автомобільні та мобільні базові станції [17].

Висока продуктивність FPGA пов'язана з її гнучкістю, що робить можливим реалізовувати схеми, які повністю оптимізовані під специфічну задачу, та з великою кількістю пам'яті в середині мікросхеми, що підтримують високий паралелізм [18].

1.3 Огляд видів суматорів

Основними структурними елементами всіх суматорів в обчислювальних техніці є напівсуматор та повний суматор. На входи напівсуматору та повного суматору подаються два відповідних біти двох доданків, а на виході отримуються біт суми та біт переносу в інший розряд. Різниця між напівсуматором та повним суматором полягає у наявності в останнього вхідного сигналу переносу з попереднього розряду.

При мінімізації таблиці дійсності напівсуматору одержуються наступні формули суми s_i та переносу c_{out} :

$$s_i = x_i \oplus y_i, \quad (1.7)$$

$$c_{out} = x_i y_i \quad (1.8)$$

де x_i, y_i – відповідні біти двох доданків.

Як було сказано раніше, повний суматор враховує біт переносу з попереднього розряду, тому його формули дещо відрізняються від формул напівсуматору та мають наступний вигляд:

$$s_i = x_i \oplus y_i \oplus c_{in}, \quad (1.9)$$

$$c_{out} = x_i y_i + x_i c_{in} + y_i c_{in} \quad (1.10)$$

де c_{in} – біт переносу з попереднього розряду.

Для додавання двох n -бітних чисел одного напівсуматору або повного суматору недостатньо. У такому випадку створюється ланцюг повних суматорів, що з'єднанні між собою. Така структура зображена на рисунку 1.8, з якого видно, що вихідний біт переносу c_{out} одного з повних суматорів з'єднаний з вхідним бітом переносу c_{in} наступного.

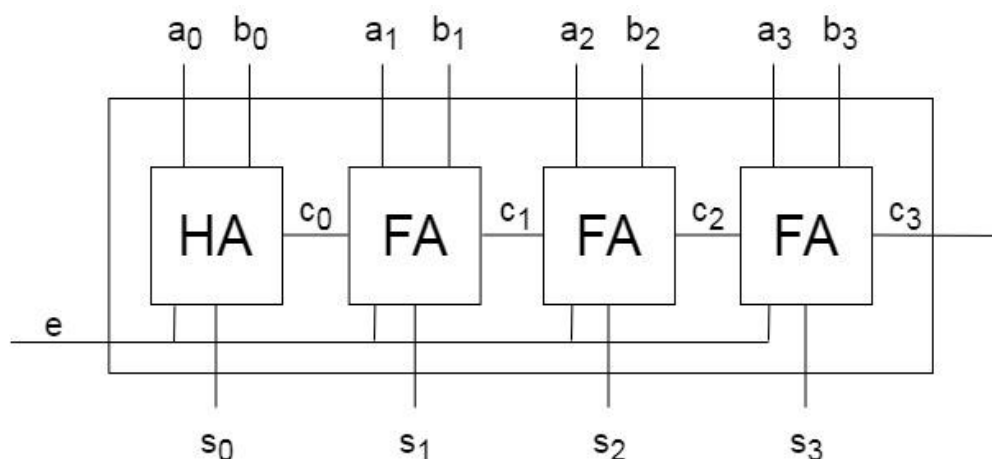


Рисунок 1.8 – Ланцюг з трьох повних суматорів і одного неповного суматора

Суматор зі структурою, представленою на рисунку 1.8 називається суматором з послідовним переносом (ripple-carry adder). Дана назва відображає

основний недолік даного типу суматору – послідовний порозрядне перенесення, що призводить до часових затримок.

В загальному випадку найгіршу затримку перенесення T_{RCA} від першого біта до останнього для n -бітного суматору можна визначити за формулою[3]:

$$T_{RCA} = nT_{FA}, \quad (1.11)$$

де T_{FA} – затримка на одному повному суматорі.

Затримка T_{FA} залежить від технології реалізації повного суматору. Таким чином, затримка переносу зростає лінійно зі збільшенням розрядності суматору. Відомо також, що середня затримка на суматорі з послідовним перенесенням не перевищує $\log_2 n$ [19].

Головною перевагою суматору з послідовним перенесенням є простота його реалізації.

Очевидно, що при розробці багаторозрядних суматорів необхідно шукати шляхи підвищення швидкості порозрядного переносу. Серед суматорів, що мають більш ефективні методи порозрядного переносу у порівнянні із суматором з послідовним переносом, можна виділити наступні: суматор з прискореним переносом (carry-look ahead adder, CLA), суматор із пропуском перенесення (carry-skip adder), суматор з вибором перенесення (carry-select adder), суматор зі збереженням перенесення (carry-save adder).

При розгляданні суматору з прискореним перенесенням вводяться наступні поняття: генерація (generation), розповсюдження (propagation) та поглинання (absorption) перенесення. Дані випадки описуються наступними формулами для генерації g_i , розповсюдження p_i та поглинання a_i перенесення в i -му розряді:

$$g_i = x_i y_i \quad (1.12)$$

$$p_i = x_i \oplus y_i \quad (1.13)$$

$$a_i = \overline{x_i + y_i} \quad (1.14)$$

Таким чином, формула перенесення у наступний $i+1$ -й розряд виражається формулою:

$$c_{i+1} = g_i + c_i p_i \quad (1.15)$$

Виконавши елементарні перетворення, отримаємо наступну формулу, яку легше реалізувати [1]:

$$c_{i+1} = g_i + c_i p_i = g_i + (g_i + p_i)c_i = g_i + (x_i + y_i)c_i \quad (1.16)$$

Розкриваючи формулу (1.16) для кожного i -го розряду отримаємо n формул, що описують перенесення у n -бітному суматорі. Кожен біт переносу у такому випадку не буде залежати від попереднього і буде вираховуватися паралельно з іншими. Очевидно, що чим більша розрядність суматору, тим громіздкими становляться формули для кожного перенесення з i -го розряду.

Перевагою суматору з прискореним перенесенням у порівнянні з суматором з послідовним перенесенням є зменшена часова затримка, яка виражається формулою [20]:

$$T_{CLA} = \alpha + \beta \log_2 n, \quad (1.17)$$

де α, β – часові затримки суматору та схеми переносу відповідно;

m – кількість біт в одному блоці суматору з прискореним переносом;

n – довжина доданків в бітах.

Таким чином, часова складність суматору з прискореним перенесенням виражається як $O(\log n)$, коли складність суматору з послідовним перенесенням – як $O(n)$.

Пояснення потребує поняття «блок» в формулі (1.17). Оскільки зі збільшенням розрядів суматора збільшується кількість логічних елементів у схемі переносу, то суматор з прискореним переносом розбивають на блоки однакової довжини.

Як і у випадку з суматором з прискореним переносом, суматор з пропуском перенесення зменшує найгірший час затримки. Даний тип суматора є модифікацією суматора з послідовним перенесенням.

Принцип дії такого суматора наступний. Суматор з послідовним перенесенням розбивається на декілька блоків, у кожного з яких є схема пропуску перенесення. Кожен блок або пропускає сигнал перенесення через себе, або поглинає його. Умова пропуску перенесення P даним блоком до наступного виражається формулою:

$$P = \prod_{i=0}^{m-1} p_i, \quad (1.18)$$

де m – кількість розрядів у блоці;

p_i – умова перенесення з i -го розряду.

Таким чином, сигнал перенесення пропустить блок, якщо $P=1$, тобто у тому випадку, коли кожен розряд у блоці розповсюджує перенесення. В іншому випадку сигнал перенесення поглинеться відповідним блоком.

На рисунку 1.9 зображена схема простого суматора з пропуском перенесення.

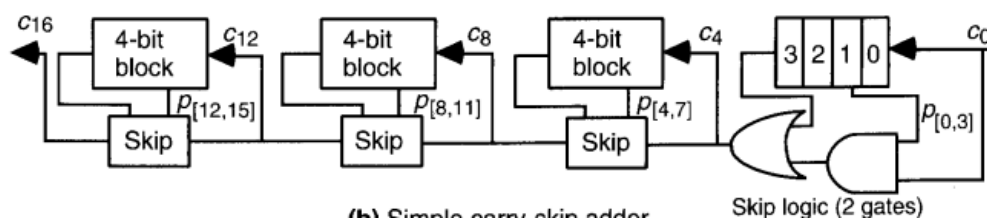


Рисунок 1.9 – Схема суматора з пропуском перенесення

Час найгіршої затримки виникає тоді, коли сигнал перенесення генерується в наймолодшому біті суматора та розповсюджується до найстаршого. У такому випадку час затримки T_{CSkA} виражається наступною формулою [21]:

$$T_{CSkA} = (2m + \frac{n}{m} - 3)t_u, \quad (1.19)$$

де m – кількість розрядів у блоці;

n – кількість розрядів суматора;

t_u – затримка розповсюдження на комбінація елементів AND-OR.

Оптимальний розмір окремого блоку знаходиться як мінімум функції $T_{CSkA}(m)$ і визначається за формулою:

$$m = \sqrt{\frac{n}{2}} \quad (1.20)$$

Принцип роботи суматора з вибором перенесення наступний. Суматор поділяється на декілька секцій. Кожна з секцій подвоєна, тому відбувається дві одночасних операцій додавання: одна для випадку з наявним переносом, інша – з відсутнім [22].

На рисунку 1.10 представлено простий суматор з вибором перенесення.

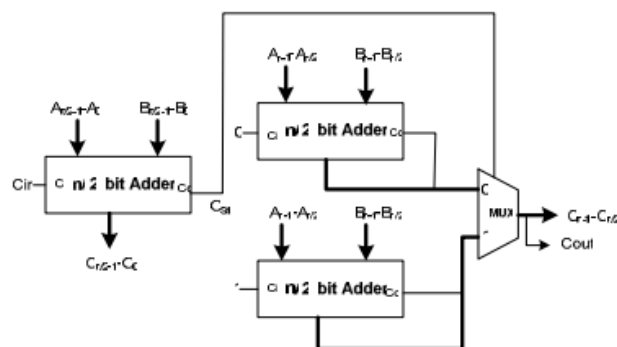


Рисунок 1.10 – Структура суматора з вибором перенесення

З рисунка видно, що друга половина сума вираховується двічі і відповідний результат подається на вихід в залежності від значення перенесення з попереднього розряду.

На часову затримку впливають затримки на k мультиплексорах та затримка на одному m -бітному блоці суматору, оскільки всі суми вираховуються одночасно. Таким чином часову затримку можна виразити формулою:

$$T_{CSIA} = t_{sum} + kt_{mux} = t_{sum} + \left(\frac{n}{m} - 1\right)t_{mux}, \quad (1.21)$$

де n – кількість розрядів суматору;

t_{mux} – затримка на мультиплексорі;

t_{sum} – затримка на одному блоці суматору.

При мінімальній кількості розрядів в одному блоці часова складність суматору з вибором перенесення виражається як $O=(n^{1/2})$ [23].

Таким чином, розглянувши чотири різні за функціональністю суматори і розрахувавши їх часові затримки, можна прийти до наступного висновку. Суматори з послідовним перенесенням мають найбільш компактну структуру, складність якої виражається як $O(n)$, але найгіршу часову затримку, складність якої виражається як $O(n)$. Суматори з прискореним перенесенням мають найменшу часову затримку $O(\log(n))$, але найгіршу структуру, складність якої $O(n\log(n))$. Суматори з вибором перенесення розглядаються як компроміс між двома попередніми типами суматорів, маючи часову складність $O(n^{1/2})$ та складність структури $O(2n)$ [24]. Суматори з пропуском перенесення мають часову складність $O(n^{1/2})$ та складність структури $O(n)$.

1.4 Огляд видів помножувачів

Найпростішим типом помножувача в двійковій арифметиці є помножувач, що працює за алгоритмом «shift-and-add». Даний алгоритм представляє собою реалізацію звичайного методу множення, який може бути виконаний на папері. Даний алгоритм обчислення добутку p двох n -бітних x та y чисел описується наступною формулою [1]:

$$p_{i+1} = (p_i + x \cdot 2^n \cdot y_i) \cdot 2^{-1}, i = \overline{0..n-1}, \quad (1.22)$$

де i – індекс часткової суми.

В формулі (1.22) 2^n порівнює множник зі старшою половиною добутку перед додаванням.

На рисунку 1.11 представлено типову структуру помножувача, що працює за алгоритмом «shift-and-add».

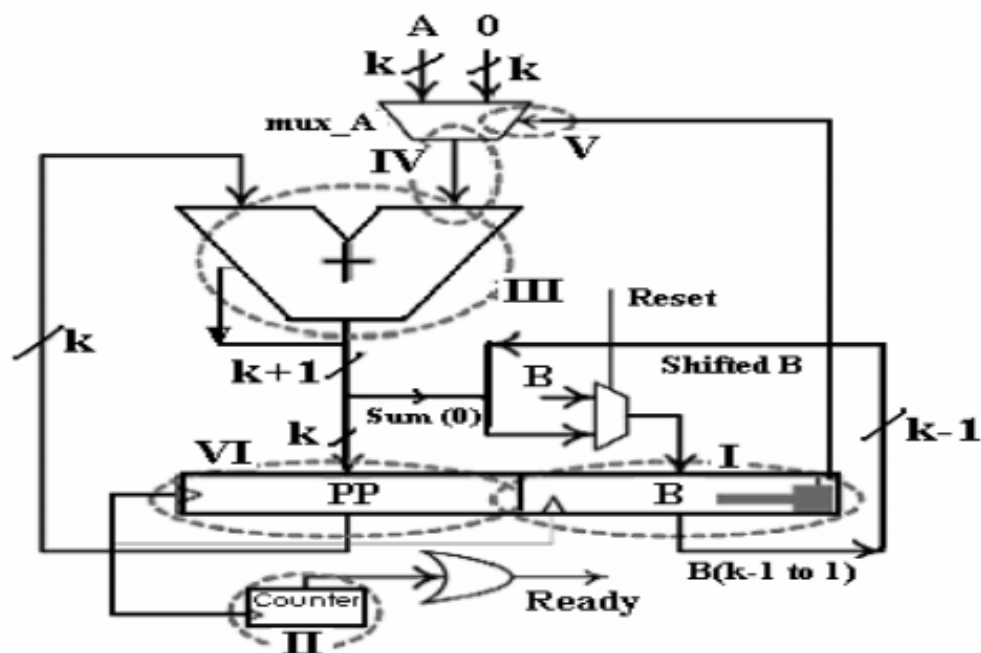


Рисунок 1.11 – Структура помножувача за алгоритмом «shift-and-add» [25]

З рисунку 1.11 видно, що схема складається з одного регістру зсуву, котрий зберігає одночасно часткову суму PP (partial product) та множник. Такий спосіб зберігання дозволяє зекономити на використанні додаткового регістру для множника. В схем наявний суматор, що складає часткову суму з множенням. В залежності від того, яке значення біту множника використовується в даний момент при множенні, на вхід суматору надходить нуль або значення множеного. При довжині k множеного та множника, результат множення представляється вектором довжини $2k$. Якщо при складанні множеного з частковою сумою виникає сигнал переносу, то сума все одно буде збережена коректно, оскільки вона кожний раз зсувається вправо на один біт.

Помножувач, що працює за алгоритмом «shift-and-add», є найпростішим у реалізації, але у той час потребує великої кількості часу для виконання обчислення.

Алгоритм множення двійкових чисел представляє собою процес деякої кількості часткових сум, кожна з яких обирається з кінцевого набору добутоків множеного та певної кількості розрядів множника. Прискорення процесу множення може бути досягнуто виконанням одного або декількох з наступних методів [26]:

- зменшення числа часткових сум;
- прискорення формування часткових сум;
- прискорення складання часткових сум.

Для зменшення числа часткових сум можна використовувати кількість розрядів множника більшу за один. Таким чином замість часткових сум типу 0 або A , де A представляє собою значення множеного, використовуються часткові суми $2A$, $3A$ та більше в залежності від кількості розрядів множника, що використовуються при формуванні часткових сум. Очевидно, що складність формування таких часткових сум складніша і тому потребує методу для спрощення процесу їх обчислення. Основним таким методом є так званий алгоритм Бута.

Алгоритм Бута заснований на тому факті, що процес множення значно спрощується при наявності в числі послідовності нулів або одиниць. Для групи з послідовних нулів немає необхідності послідовно виконувати операції зсуву та додавання. Достатньо зсунути вже обчислену на попередньому етапі суму на один біт вправо для кожного нуля. Для групи з послідовних одиниць можна так само зменшити кількість операцій зсуву та додавання. Так, послідовність з одиниць, яку можна записати як $\dots 0\{11\dots 11\}0\dots$, можна представити таким чином [27]:

$$\dots 0\{11\dots 11\}0\dots = \dots 1\{00\dots 00\}0\dots - \dots 0\{00\dots 01\}0\dots \quad (1.23)$$

Таким чином, послідовність операцій зсуву та додавання перетворюється на одну операцію додавання та віднімання.

Спрощення обчислення часткових сум при використанні алгоритму Бута можна побачити наочно. Якщо, наприклад множення відбувається за участі двох розрядів множника, тобто за основою чотири, то замість часткових сум $0, A, 2A$ та $3A$ використовуються часткові суми $0, A, -A, 2A$ та $-2A$. Таким чином не виконується обчислення часткової суми $3A$, яка є найскладнішою поміж інших. Спрощення обчислення часткових сум стає більш важливим, коли множення виконується за основами 8, 16 та навіть більше.

Для прискорення складання часткових сум використовуються суматори зі збереженням переносу. Їх використання дозволяє також зменшити кількість циклів складання, оскільки декілька часткових сум можна додавати одночасно.

На рисунку 1.12 зображено структуру помножувача, що побудований на основі так званого дерева Воллеса (Wallace tree). Дерево Воллеса представляє собою послідовність суматорів зі збереженням переносу, що виконують функцію зменшення кількості часткових сум. В кінці дерева знаходиться звичайний суматор з послідовним переносом, але можна також використовувати й інші суматори, що зменшують затримку розповсюдження сигналу перенесення.

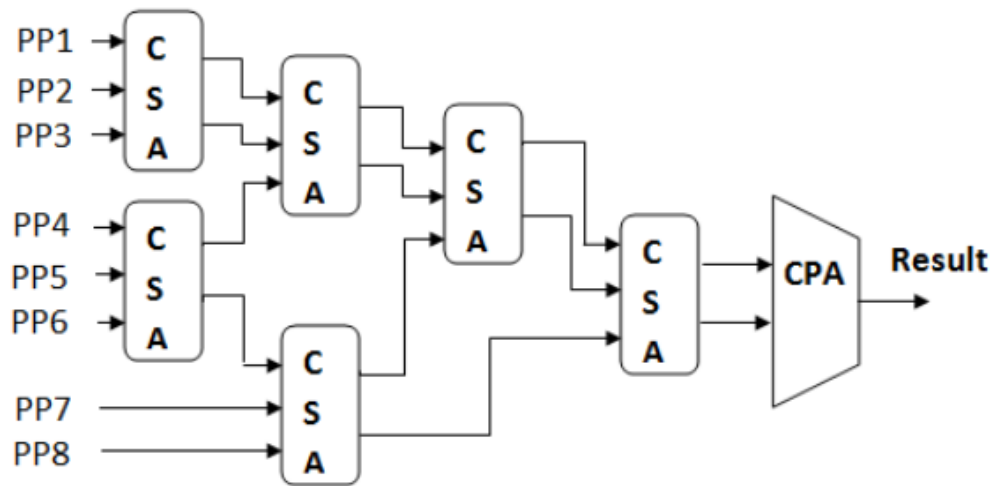


Рисунок 1.12 – Помножувач на основі дерева Воллеса [28]

Як вже було сказано, основною складовою помножувача на основі дерева Воллеса є суматор зі збереженням перенесення. Як відомо, суматор зі збереженням перенесення може складати одночасно три числа. Структура на рисунку 1.13 називається «4:2 компресор» та дозволяє складати чотири числа одночасно. Використання «4:2 компресору» при зображенні структури дерев Воллеса дозволяє спростити читання схеми.

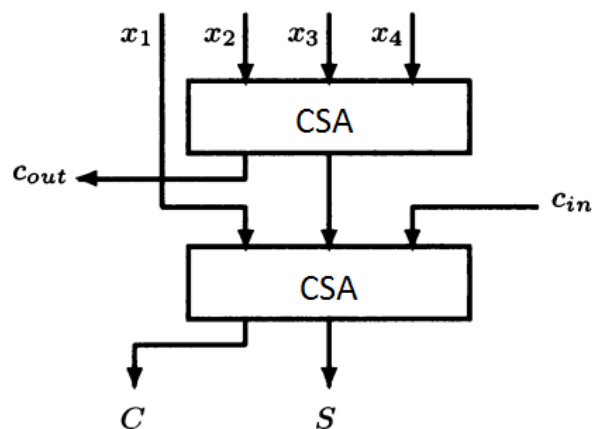


Рисунок 1.13 – Структура «4:2 компресору» [27]

На рисунку 1.14 представлено структуру типового матричного помножувача. Видно, що кожен рівень цього помножувача представляє собою

суматор зі збереженням переносу, до якого надходять сума та перенесення з попереднього суматора та нова часткова сума.

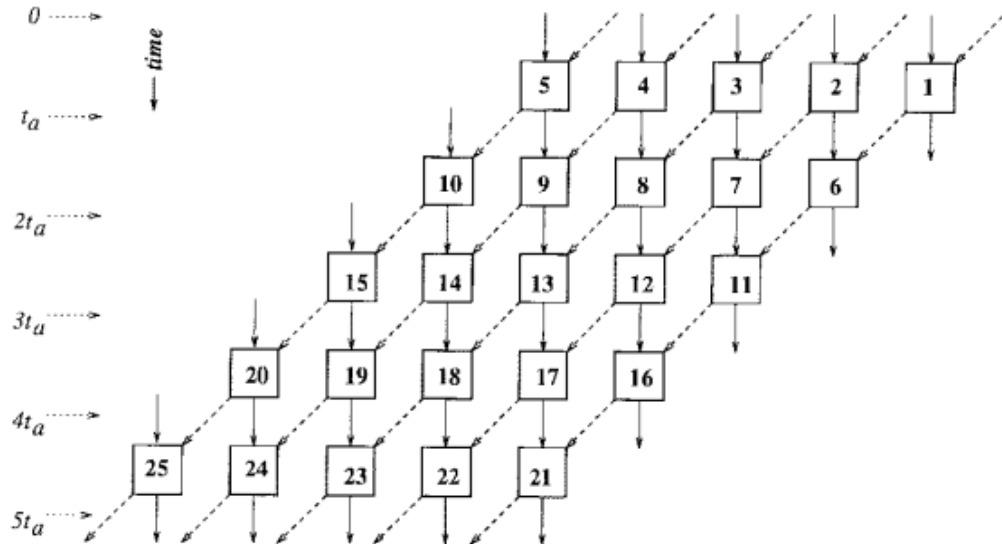


Рисунок 1.14 – Структура матричного помножувача [29]

Матричний помножувач має найбільшу затримку з поміж всіх помножувачів, що мають структуру дерева. Використання матричного помножувача головним чином обумовлене простотою його структури та можливістю його подальшої конвеєризації.

1.5 Огляд видів дільників

В двійковій арифметиці ділення є однією з найскладніших операцій, як з погляду на алгоритм так і на реалізацію, але на відміну від множення та додавання використовується не часто.

Як відомо, будь-яке число x при діленні на d можна представити у наступному вигляді:

$$x = q \cdot d + r, \quad (1.24)$$

де q – частка;

r – залишок.

Базовий алгоритм ділення двійкових чисел заснований на формулі (1.24) і в процесі його виконання відбувається пошук значень розрядів вектора частки q . Базовий алгоритм ділення представляється наступною формулою[3]:

$$r^{(i)} = 2r^{(i-1)} - q_{k-i}(2^k d), \quad (1.25)$$

де i – номер ітерації;

k – довжина дільника в бітах.

З формули (1.25) видно, що i -й залишок дорівнює різниці подвоєного попереднього залишку і добутку дільника та розряду частки. Множник 2^k в формулі (1.25) означає зсув дільника на k біт вліво для порівняння його з діленим. Множник $2r^{(i-1)}$ в формулі (1.25) означає зсув діленого на один біт вліво, для спрощення процесу віднімання від часткового залишку.

Послідовний алгоритм ділення, що представлений формулою (1.25), використовується у двох варіантах: алгоритм з відновленням та алгоритм без відновлення. Основна задача послідовного алгоритму полягає в знаходженні такого q_{k-i} , при підстановці якого у формулу (1.25) різниця $r^{(i)}$ стала додатною. Алгоритм з відновленням при від'ємній різниці відновлює залишок перед продовженням обчислень, на відміну від алгоритму без відновлення. Таким чином, алгоритм без відновлення залишку виконується швидше.

Основною проблемою звичайних дільників є той факт, що для того, щоб перейти до $i+1$ -го кроку, необхідно завершити i -й крок. Множення може бути легко пришвидшене генеруванням декількох часткових сум одночасно, так як вони незалежні. При діленні навпаки, часткові залишки залежать один від одного і не можна почати новий крок обчислень до того як стає відомим залишок і біт

частки на даному кроці. Кожен крок ділення складається з двох частин: вибір значення біту дільника та обчислення нового часткового залишку [27].

Як і у випадку з множенням одним із методів прискорення є використання методів ділення за основою більшою за 2. Так при діленні за основою 4 або 8 використовується в два та в три рази менше часткових залишків відповідно.

Зі збільшенням основи ділення збільшується і складність обчислення бітів частки, що може збільшити тактовий період. Більш того, для великих основ обчислення часткових залишків стає непрактичним. Метод SRT-ділення позбавляє від такої проблеми. Основою методу STR-ділення є використання надлишкового представлення частки. За допомогою надлишкового представлення кінцева частка може бути представлена декількома способами. Кожне дійсне надлишкове представлення може бути перетворено в звичайне. З надлишковістю вибір розрядів частки потребує лише апроксимації дільника та часткового залишку, тому що невеликі похибки можуть бути скореговані меншою кількістю значущих бітів з протилежним знаком [30].

1.6 Огляд існуючих аналогів

У роботі [31] було запропоновано модуль обчислення з рухомою комою з реалізацією на FPGA. На рисунку 1.15 представлено його структуру.

Модуль обчислення представлений на рисунку 1.15 дозволяє виконувати операції складання, віднімання, множення та ділення у форматі чисел з одинарною точністю.

Особливістю даного модуля обчислень є наявність конвеєру, який автори використали для підвищення пропускної здатності.

Іншим аналогом є математичний співпроцесор, який був розроблений для одноплатних комп'ютерів [32]. На рисунку 1.16 зображено структуру даного співпроцесору.

Даний співпроцесор працює за методом CORDIC, що дозволяє вираховувати складні функції за допомогою простих операцій зсуву та складання. За допомогою даного методу у представленому на рисунку 1.4 співпроцесорі вираховуються як звичайні арифметичні функції, так і тригонометричні та логарифмічні.

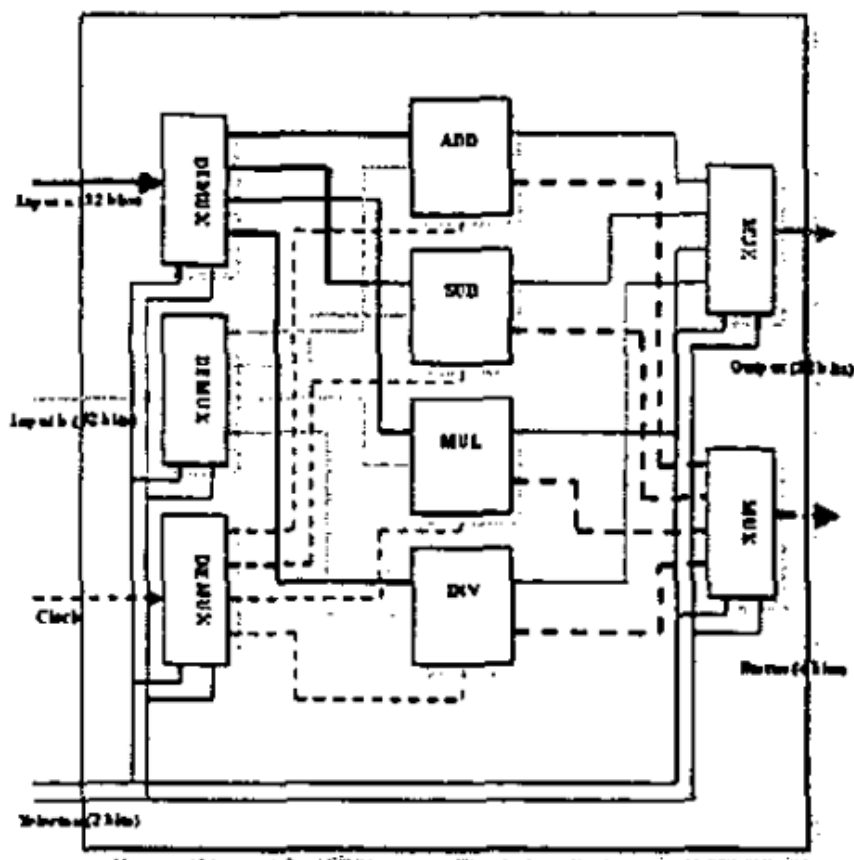


Рисунок 1.15 – Структура модуля обчислення з рухомою комою [31]

Всередині модуля наявний кінцевий автомат, який керує процесом обчислення функцій та перевіряє отриманий результат.

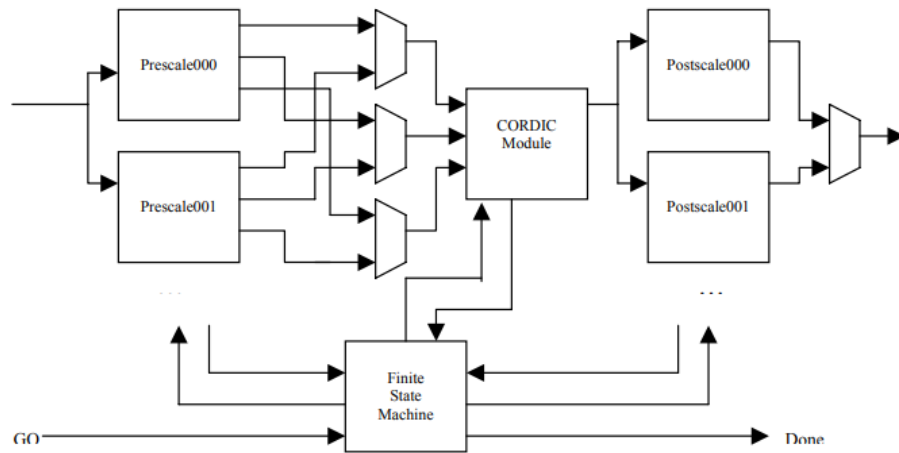


Рисунок 1.16 – Структура математичного співпроцесору [32]

Як і попередній модуль обчислення з рухомою комою, даний співпроцесор виконує операції над числами одинарної точності у форматі IEEE-754.

Покращенням модуля обчислення з рухомою комою, що був представлений у [31] є модуль обчислення [33]. Його структура зображена на рисунку 1.17.

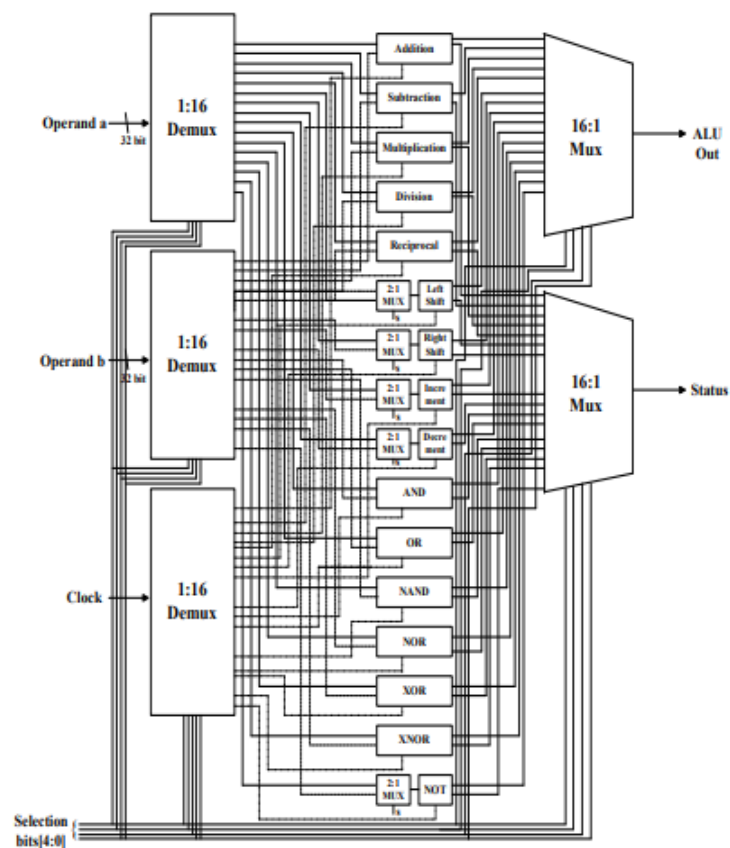


Рисунок 1.17 – Структура модуля обчислень з рухомою комою[33]

З рисунку 1.17 видно, що модуль може працювати з числами довжини 64 біт та має додаткові логічні і арифметичні операції, що в свою чергу збільшує кількість виводів у мультиплексорів.

1.7 Постановка задачі розробки

Згідно з завданням необхідно розробити модуль обчислень з рухомою комою, що оперує двома числами одинарної точності у форматі IEEE-754 та має можливість їх складати, віднімати, множити та ділити.

На основі проведеного аналізу визначено, яку структуру повинні мати модуль обчислення з рухомою комою і його внутрішні функціональні блоки. Виявлено, які методи виконання арифметичних операцій володіють найменшою часовою затримкою і найменшим використанням ресурсів. Розглянуто структуру аналогів та виявлено загальні принципи побудови модулів обчислень.

Модуль обчислень повинен коректно реагувати на виняткові події, котрі виникають при виконанні арифметичних операцій та сигналізувати про них.

Наприкінці необхідно провести моделювання і тестування модуля обчислень для підтвердження коректності його роботи.

2 РОЗРОБКА МОДУЛЯ ОБЧИСЛЕНЬ З РУХОМОЮ КОМОЮ

2.1 Розробка структури модуля обчислень з рухомою комою

Блоки складання та віднімання, множення і ділення необхідно об'єднати в одну систему, що буде виконувати ту чи іншу операцію над числами в залежності від коду операції.

На рисунку 2.1 зображено структуру модуля обчислення з рухомою комою.

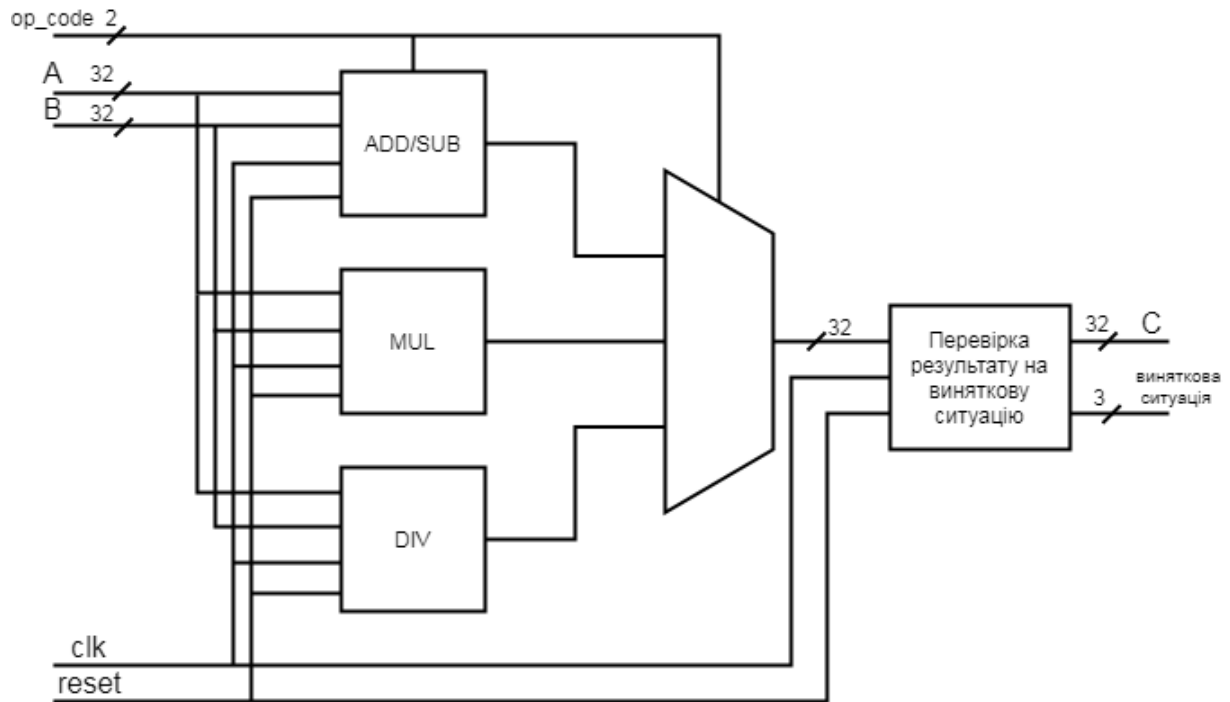


Рисунок 2.1 – Структура модуля обчислень з рухомою комою

З рисунку 2.1 видно, що блоки складання та віднімання, множення і ділення мають спільний тактовий сигнал, сигнал скидання та шини операндів. Результат з цих блоків потрапляє на мультиплексор, який подає на вихід необхідне число в залежності від значення сигналу «op_code», що є кодом команди. Код команди додатково надходить на блок складання та віднімання, оскільки він містить в собі дві різні операції.

Вихідний результат надходить на блок перевірки на виняткову ситуацію, де перевіряється на коректність. В залежності від значення результату формується код виняткової операції.

2.2 Розробка блоку складання та віднімання

2.2.1 Алгоритм і структура блоку складання та віднімання

Блок складання та віднімання був розроблений згідно зі стандартним алгоритмом складання двох чисел з рухомою комою. Алгоритм складання чисел з рухомою комою представлений на рисунку 2.2.

Перед початком складання відбувається перевірка чисел на виняткову ситуацію. Типові виняткові ситуації та їх вигляд у стандарті IEEE-754 представлені у таблиці 1.1. Перевірка на виняткову ситуацію перед початком операції складання або віднімання обумовлена тим, що комбінації звичайних чисел та чисел, що позначають нуль, нескінченність або NaN може давати різний результат. Більш того, інформація про виняткову ситуацію може втратитися в процесі обчислень, а саме при зміні експоненти, мантиси і знаку.

Для складання або віднімання двох чисел з рухомою комою необхідно виконати порівняння експонент. Число, яке має меншу експоненту, зсувається вправо на кількість розрядів, що дорівнює різниці експонент. Якщо навпаки, зсувається число, що має більшу експоненту вліво, то мантиса стане більшою за одиницю і в такому випадку необхідно буде використовувати більший суматор.

Після порівняння експонент та зсуву мантиси меншого числа виконується безпосередньо додавання мантис двох чисел з рухомою комою. Варто також враховувати знак чисел та саму операцію. Якщо число від'ємне, то в процесі складання воно представляється за допомогою доповняльного коду.

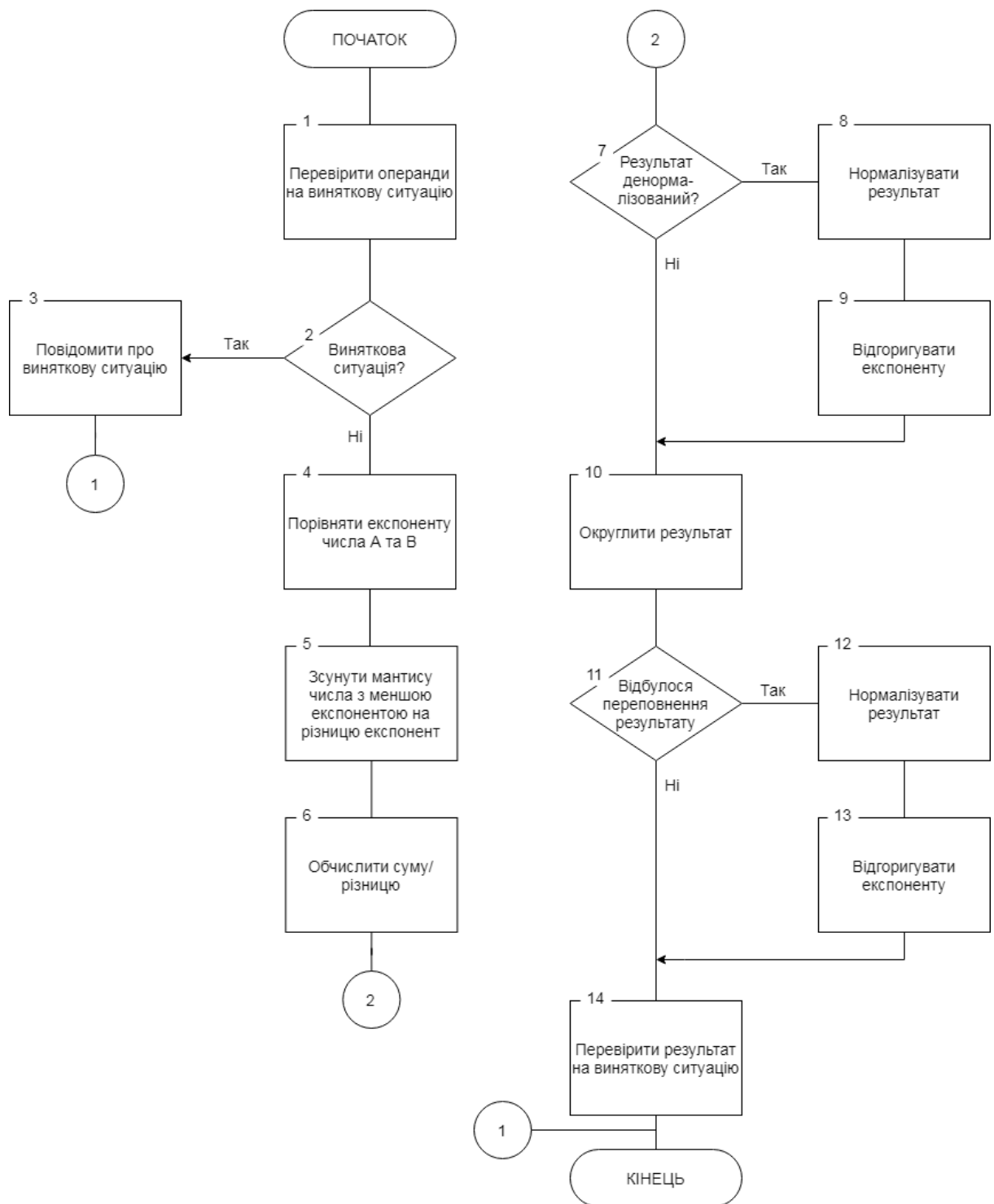


Рисунок 2.2 – Блок-схема алгоритму складання двох чисел з рухомою комою

Після складання або віднімання число нормалізується. Якщо результат вийшов за розрядну сітку, то відбувається зсув вправо на один біт. Якщо результат має в старший бітах нулі, то відбувається зсув вліво до тих пір, поки старший біт не стане дорівнювати одиниці. Таким чином зберігається нормалізована форма числа. Кожен зсув вліво або вправо враховується при

корекції експоненти. При зсуві вліво експонента зменшується, а при зсуві вправо збільшується.

Для більш точного обчислення результат округлюється. При округленні може також виникнути ситуація з виходом результату за розрядну сітку. В такому випадку число зсувається на один біт вправо, а експонента збільшується на одиницю. Отриманий результат повторно перевіряється на виняткову ситуацію.

Згідно з розглянутим алгоритмом складання та віднімання чисел з рухомою комою було розроблено структуру блоку складання та віднімання. Запропонована структура блоку складання та віднімання зображена на рисунку 2.3.

За допомогою структури зображеної на рисунку 2.3 можна виділити зв'язки між функціональними блоками. Так, наприклад видно, що корекція експоненти, на відміну від алгоритму, виконується один раз після нормалізації та округлення, а порівняння експонент та перевірку операндів на виняткову ситуацію можна проводити одночасно.

В схемі присутній блок LOD (leading one detector). Даний блок виконує пошук позиції першої одиниці в результаті. Таким чином вираховується кількість біт, на яку необхідно зсунути результат обчислення для представлення його в нормалізованому вигляді.

Окремим блоком представлено блок визначення знаку. Оскільки, окрім додатних чисел, на вхід блоку складання та віднімання можуть надходити числа з від'ємним знаком, то необхідно після операції складання або віднімання обчислити знак результату.

Конвеєризація обчислювальних пристроїв дозволяє збільшити їх пропускну здатність та реалізувати можливість роботи в синхронних системах, що керуються тактовим сигналом. Більш того, при балансуванні конвеєризації, тобто при зменшенні кількості функціональних блоків у кожній стадії конвеєру, можна підвищити частоту роботи пристрою.

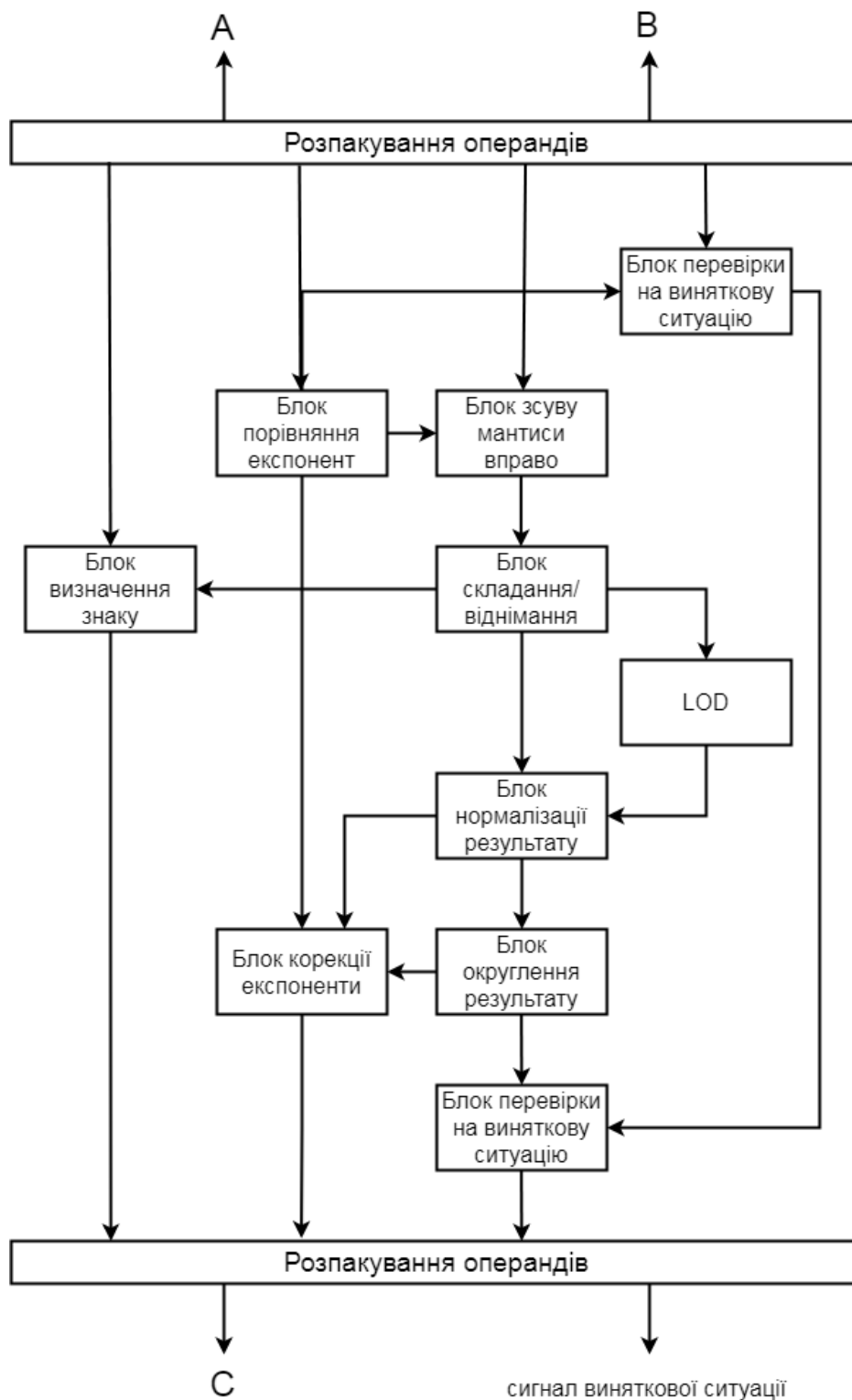


Рисунок 2.3 – Структура блоку складання та віднімання чисел з рухомою комою

Враховуючи дані принципи побудови конвеєрних пристроїв на основі роботи [34] було побудовано конвеєрний блок складання та віднімання чисел з рухомою комою. Відповідна структура зображена на рисунку 2.4.

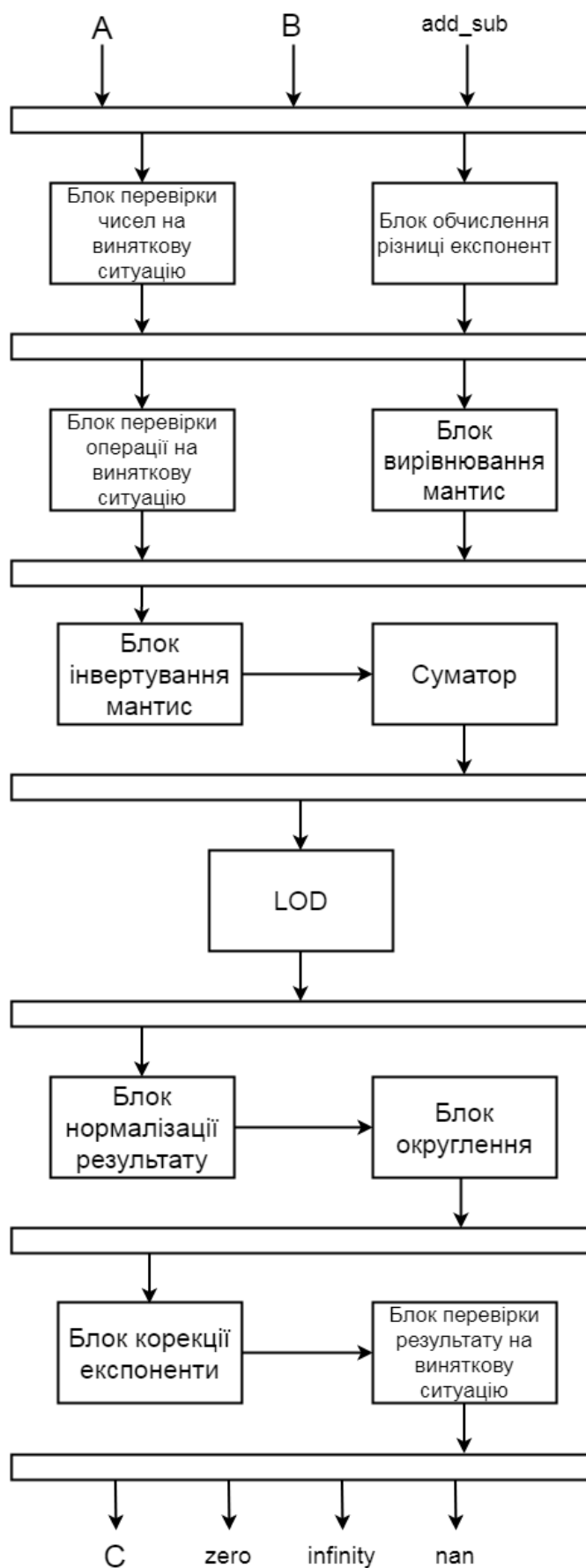


Рисунок 2.4 – Конверсна структура блоку складання та віднімання чисел з рухомою комою

З рисунку 2.4 видно, що в одному рівні конвеєру розміщено ті функціональні блоки, які незалежні один від одного і можуть виконувати обчислення паралельно. Так, наприклад, в першому рівні конвеєру розміщено блоки перевірки на виняткову ситуацію та обчислення різниці експонент. Оскільки результати обчислень кожного з цих блоків не впливають один на одного, то операції на них можуть виконуватися паралельно. Блок інвертування мантис та суматор також знаходяться в одному рівні конвеєру, але з'єднані послідовно, оскільки перед складанням мантис може виникнути необхідність інвертувати одну з них.

Швидкість роботи конвеєру визначається найповільнішим його рівнем. В розробленій структурі конвеєру найповільнішим рівнем є той, що містить блок LOD, оскільки останній має найбільшу затримку розповсюдження сигналу. Блок LOD є вузьким місцем конвеєру і розподіл інших блоків по рівнях проводився так, щоб не перевищити затримку на блоці LOD.

2.2.2 Розробка функціональних складових блоку складання та віднімання

Блок перевірки на виняткову ситуацію аналізує число та на основі таблиці 1.1 видає код виняткової ситуації. В даній роботі було використано наступні коди виняткових ситуацій: «00», «10», «11» та «01». Перші три сигналізують про нуль, нескінченність та NaN. Остання повідомляє, що операнд представляє собою звичайне число.

На рисунку 2.5 зображено розроблену структуру блоку обчислення різниці експонент.

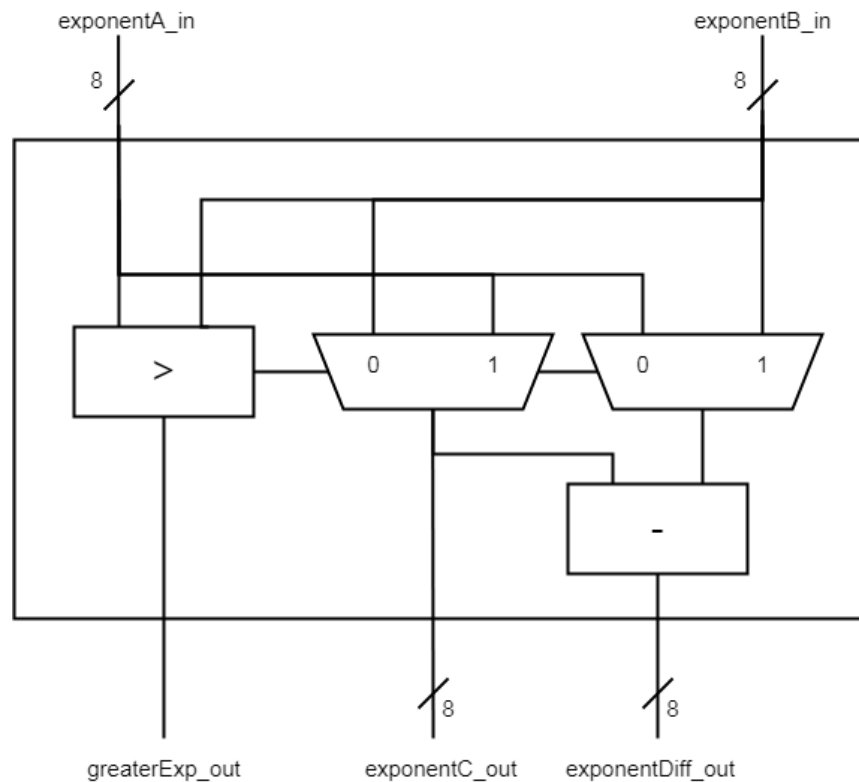


Рисунок 2.5 – Структура блоку обчислення різниці експонент

На вхід блоку обчислення експонент надходять сигнали «exponentA_in» та «exponentB_in», що позначають експоненти чисел А та В відповідно. Значення експоненти надходять на компаратор, що виявляє більшу з експонент. В залежності від того, яка експонента виявилася більшою, відбувається перестановка їх значень для подальшого обчислення різниці. Таким чином, переставляючи значення експонент, можна завжди отримувати додатній результат, не використовуючи додаткові блоки для представлення від’ємних чисел. Вихідний сигнал «greaterExp_out» приймає значення «0» та «1», що означає експоненту А та В відповідно. Сигнали «exponentC_out» «exponentDiff_out» містять більшу експоненту та різницю відповідно. В подальший обчисленнях буде приймати участь саме більша з експонент.

Блок перевірки операції на виняткову ситуацію виявляє заздалегідь, яка виняткова ситуація буде згенерована в результаті операції складання або віднімання.

В таблиці 2.1 та 2.2 представлено можливі варіанти виняткової події від тих чи інших значень двох операндів, які позначаються як «ОП1» та «ОП2» відповідно.

Таблиця 2.1 – Таблиця виняткових ситуацій для операції додавання

ОП1 ОП2	NaN	+Infinity	-Infinity	+0	-0	+Num	-Num
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
+Infinity	NaN	+Infinity	NaN	+Infinity	+Infinity	+Infinity	+Infinity
-Infinity	NaN	NaN	-Infinity	-Infinity	-Infinity	-Infinity	-Infinity
+0	NaN	+Infinity	-Infinity	+0	+0	+Num	-Num
-0	NaN	+Infinity	-Infinity	+0	-0	+Num	-Num
+Num	NaN	+Infinity	-Infinity	+Num	+Num	+Num	+Num
-Num	NaN	+Infinity	-Infinity	-Num	-Num	+Num	-Num

Таблиця 2.2 – Таблиця виняткових ситуацій для операції віднімання

ОП1 ОП2	NaN	+Infinity	-Infinity	+0	-0	+Num	-Num
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
+Infinity	NaN	NaN	-Infinity	-Infinity	-Infinity	-Infinity	-Infinity
-Infinity	NaN	+Infinity	NaN	+Infinity	+Infinity	+Infinity	+Infinity
+0	NaN	+Infinity	-Infinity	+0	-0	+Num	-Num
-0	NaN	+Infinity	-Infinity	+0	+0	+Num	-Num
+Num	NaN	+Infinity	-Infinity	+Num	-Num	+Num	-Num
-Num	NaN	+Infinity	-Infinity	+Num	+Num	+Num	-Num

На рисунку 2.6 представлено запропоновану структуру блоку вирівнювання мантис. З рисунку видно, що операнди, які надходять до блоку у вигляді сигналів «signifA_in» та «signifB_in» розширюються додаванням трьох біт в кінці, що необхідні для округлення, та одного біту в началі, що зберігає неявну одиницю, яку при звичайному записі числа з рухомою комою не пишуть.

В залежності від сигналу «greaterExp_in» відбувається направлення мантиси числа, яке має меншу експоненту, на блок зсуву вправо, а іншого – на вихід.

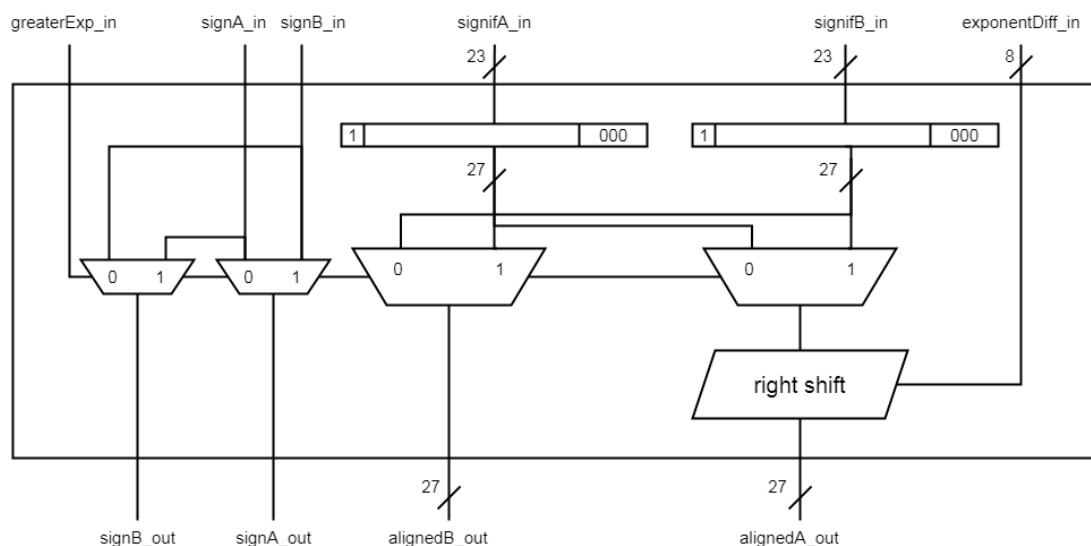


Рисунок 2.6 – Структура блоку вирівнювання мантис

Блок зсуву мантиси вправо приймає на вхід сигнал «exponentDiff_in», що позначає різницю експонент та кількість, на які необхідно зсунути мантису. Приклад внутрішньої структури блоку зсуву вправо зображена на рисунку 2.7.

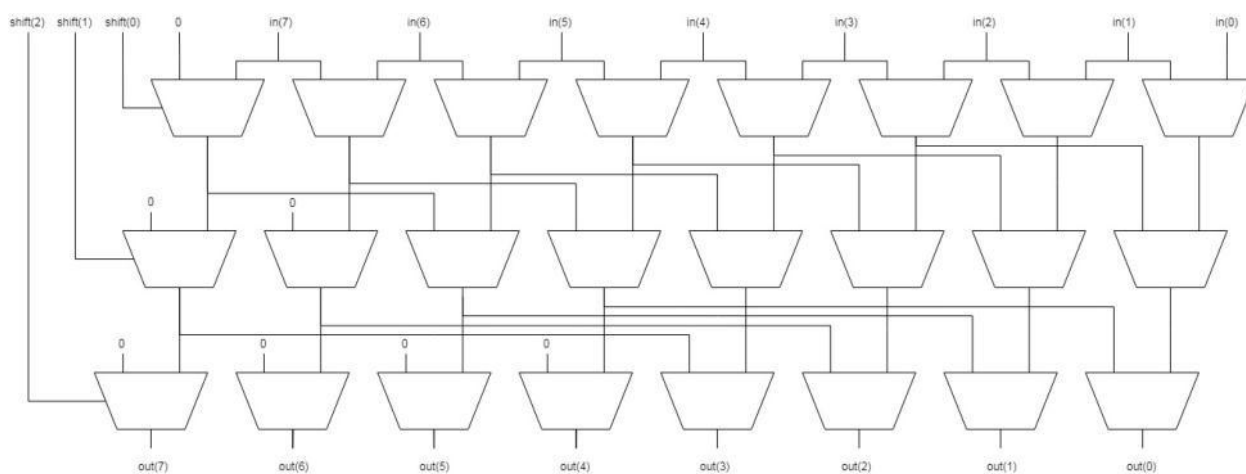


Рисунок 2.7 – Barrel shifter

Як видно з рисунку 2.7, внутрішньо блок зсуву складається з декількох рівнів мультиплексорів. Кількість рівнів розраховується як значення логарифму за основою 2 від максимального значення різниці експонент. Іншими словами, кількість рівнів дорівнює кількості біт в записі різниці експонент. В даному випадку, для запису різниці експонент використовується 8 біт, за допомогою яких можна зсунути мантису на 256 позицій. Оскільки при зсуненні мантису вже на 24

позиції вона дорівнює нулю, то використовуються лише останні 5 бітів значення різниці експонент. Якщо запис різниці експонент більший за 5 біт, то мантиса переводиться в нуль.

Сигнал «greaterExp_in» також керує мультиплексорами, на входи яких подаються знаки двох чисел «signA_in» та «signB_in». Знаки чисел повинні точно відповідати самим числам, оскільки в подальшому буде відбуватися інвертування мантис.

Розширені мантиси та знаки двох операндів подаються на визідні сигнали «alignedA_out», «alignedB_out», «signA_out» та «signB_out» відповідно.

На рисунку 2.8 зображено структуру розробленого блоку інвертування мантис. З рисунку видно, що в блоці інвертування мантис присутній лише один інвертор, який виконує перетворення прямого коду числа в доповняльний. Дане рішення обумовлене тим, що при додаванні може бути лише один від’ємний операнд. Якщо два операнди від’ємні, то відбувається зміна їх знаків на додатні.

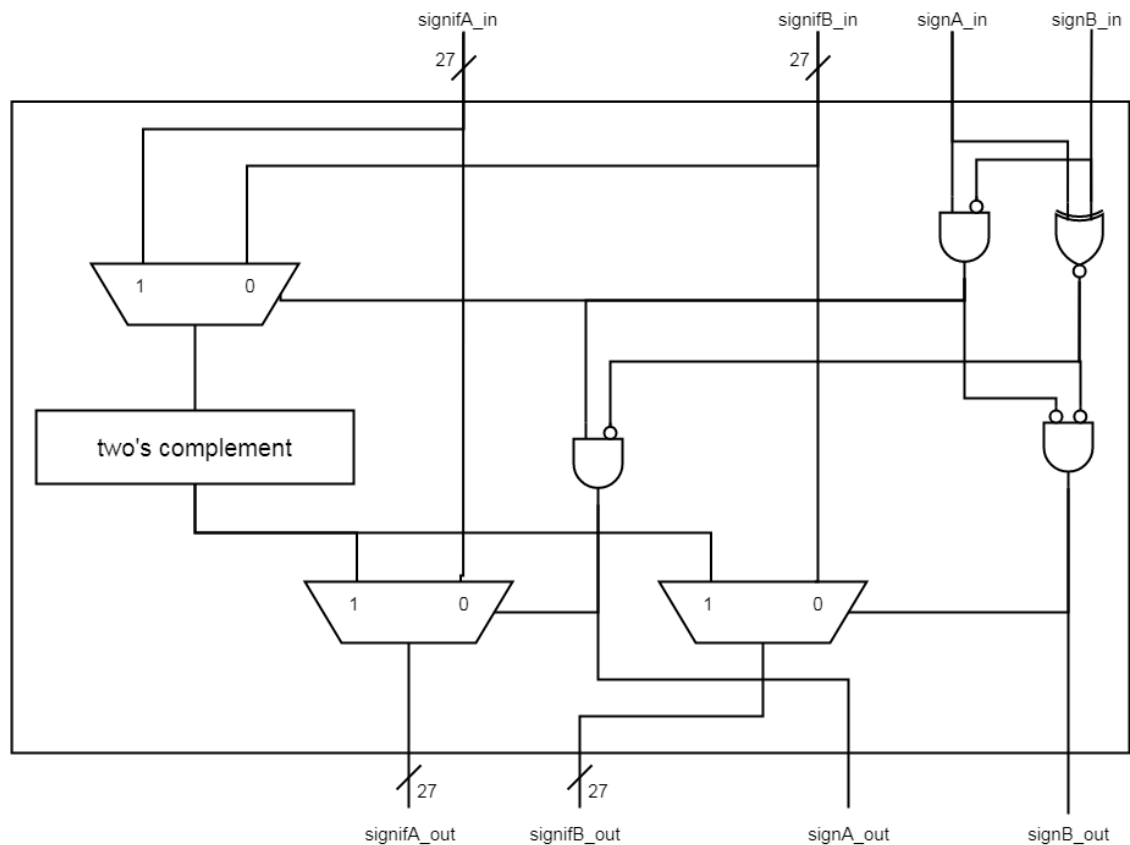


Рисунок 2.8 – Структура блоку інвертування мантис

Інвертування одного з чисел «signifA_in» або «signifB_in» відбувається направленням одного з них на інвертор за допомогою мультиплексору, керуючим сигналом якого є комбінаційна функція, що приймає на вхід знаки двох чисел «signA_in» та «signB_in». Одиничне значення функції «signA_in xnor signB_in» означає, що два числа мають рівні знаки, а одиничне значення функції «signA_in and not(signB_in)» відповідає ситуації, коли число А від’ємне, а В додатне. Якщо функція приймає значення нуль, то навпаки: А додатне, а В від’ємне. На основі цього робиться висновок, яке число необхідно представити у доповняльному коді.

Розроблена структура суматора мантис зображена на рисунку 2.9. Для складання двох чисел було обрано звичайний суматор з послідовним перенесенням, оскільки така структура найбільш підходить для реалізації на мікросхемах FPGA, через наявність спеціальних схем перенесення всередині кожного логічного елементу.

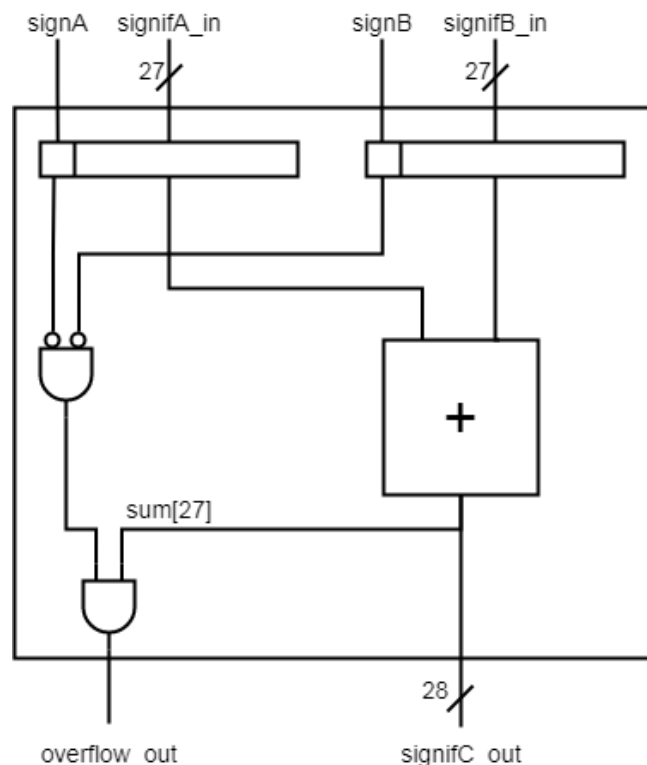


Рисунок 2.9 – Структура суматора

З рисунку 2.9 видно, що в додаванні приймає участь не тільки мантиси signifA_in та signifB_in, а й знак чисел signA_in та signB_in. Таким чином, можна

визначати ситуацію переповнення, коли відбувається зміна знаку результату, або сам результат виходить за розрядну сітку. Сигнал переповнення `overflow_out` приймає значення одиниці в тому випадку, коли знаки двох чисел дорівнюють нулю, і старший біт результату дорівнює одиниці. У такому випадку результат додатний, але його представлення на один біт більше, ніж представлення операндів. Виявлення ситуації переносу дозволяє в подальшому коректно виконати нормалізацію результату, що включає в себе інвертування і зсув числа.

На рисунку 2.10 представлено запропоновану структуру блоку попередньої нормалізації.

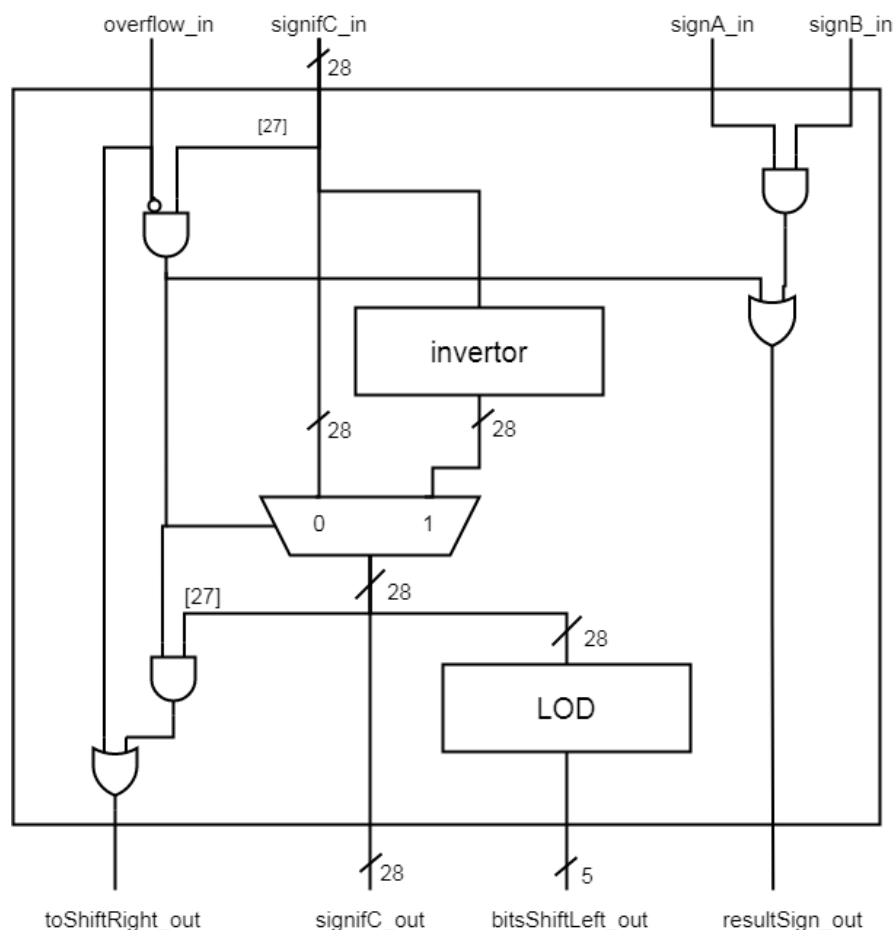


Рисунок 2.10 – Структура блоку попередньої нормалізації

Блок `LOD` є складовою даної структури і виконує операції підрахування кількості нулів на початку результату після інвертування результату.

Інвертування результату виконується у тому випадку, якщо старший біт результату дорівнює одиниці і не було переповнення.

Знак результату «resultSign_out» визначається за допомогою початкових знаків «signA_in» «signB_in» та сигналу про інвертування. Результат буде від'ємним тоді, коли обидва числа були від'ємними, або коли сигнал інвертування дорівнював одиниці.

Блок попередньої нормалізації результату визначає також, чи потрібно зсунути результат на один біт вправо. Сигнал «toShiftRight_out» приймає значення одиниці в двох випадках. Перший випадок відповідає ситуації наявності переповнення «overflow_in». Другий випадок відповідає ситуації наявності в інвертованого числа одиничного старшого біту.

На рисунку 2.11 зображено розроблену структуру блоку нормалізації результату.

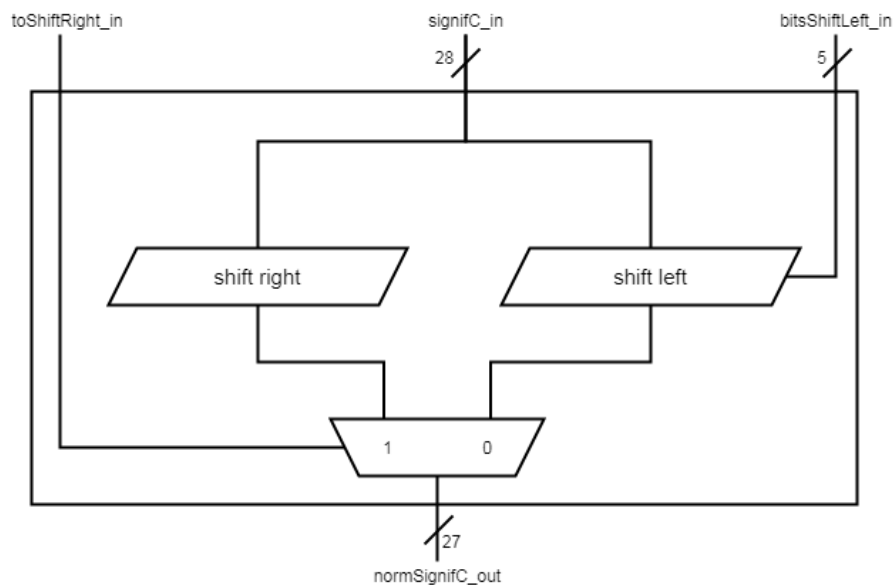


Рисунок 2.11 – Структура блоку нормалізації результату

Блок нормалізації результату складається з двох блоків зсуву, перший з яких зсуває число на один біт вправо, а другий на декілька бітів вліво. Блок, який зсуває число вліво виконано таким же чином, як і той, що представлений на рисунку 2.7.

На рисунку 2.12 зображено розроблену структуру блоку округлення.

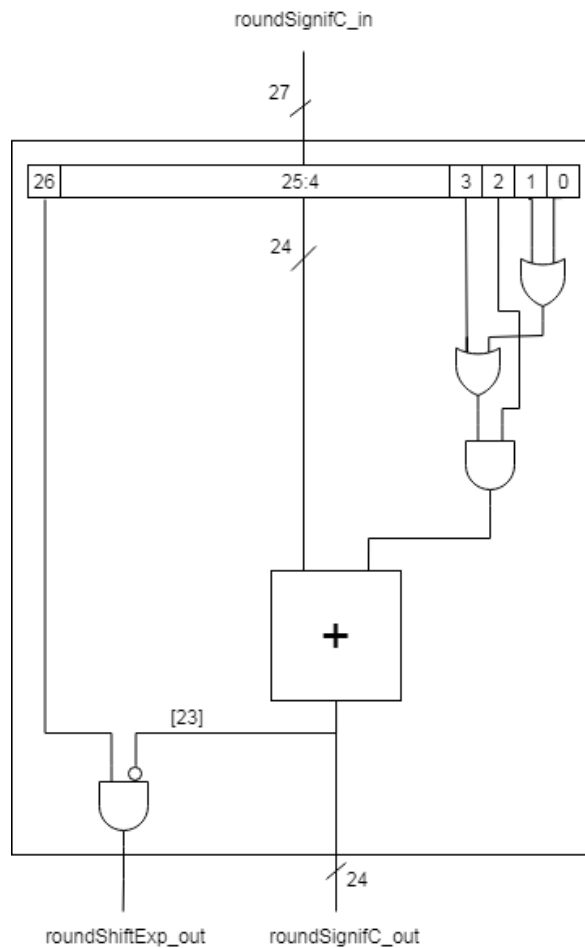


Рисунок 2.12 – Структура блоку округлення

Блок округлення складається з суматора, що додає до третього біту результату одиницю в тому випадку, якщо виконана умова округлення. В округленні беруть участь три останні біти результату. Якщо їх значення в двійковій формі «011» та менше, то округлення не відбувається. Якщо їх значення «101» більше, то округлення відбувається. При значенні в «100» округлення відбувається лише в тому випадку, якщо число непарне. Такий спосіб округлення називається округленням до найближчого парного, який є способом округлення за замовчуванням в стандарті чисел з рухомою комою IEEE-754.

На рисунку 2.13 зображено запропоновану структуру блоку корекції експоненти.

Блок корекції експоненти збільшує або зменшує експоненту, котра представлена сигналом «exponent_in». Збільшення експоненти відбувається в тому випадку, якщо мантиса результату була зсунута на один біт вправо. В іншому випадку експонента зменшується.

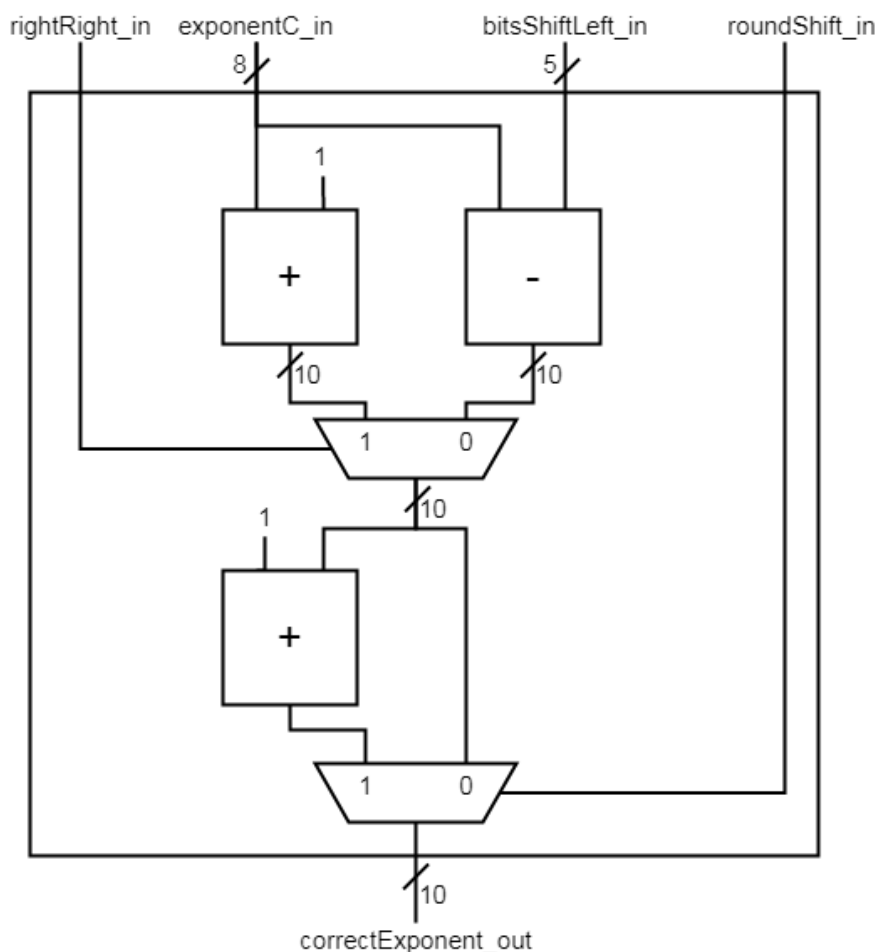


Рисунок 2.13 – Структура блоку корекції експоненти

Також перевіряється, чи була зсунута мантиса після округлення. В такому випадку експонента також збільшується на один.

На рисунку 2.14 зображено розроблену структуру блоку перевірки результату на виняткову ситуацію.

Блок перевірки на виняткову ситуацію приймає в якості вхідних сигналів експоненту «exponentC_in» та мантису «signifC_in» результату.

Якщо експонента числа більша за 255, то результат представляє нескінченність. Якщо експонента менша за нуль, то результат є денормалізованим

числом і представляється у вигляді нуля. Для перевірки кожного з цих випадків експонента має два додаткові старші біти.

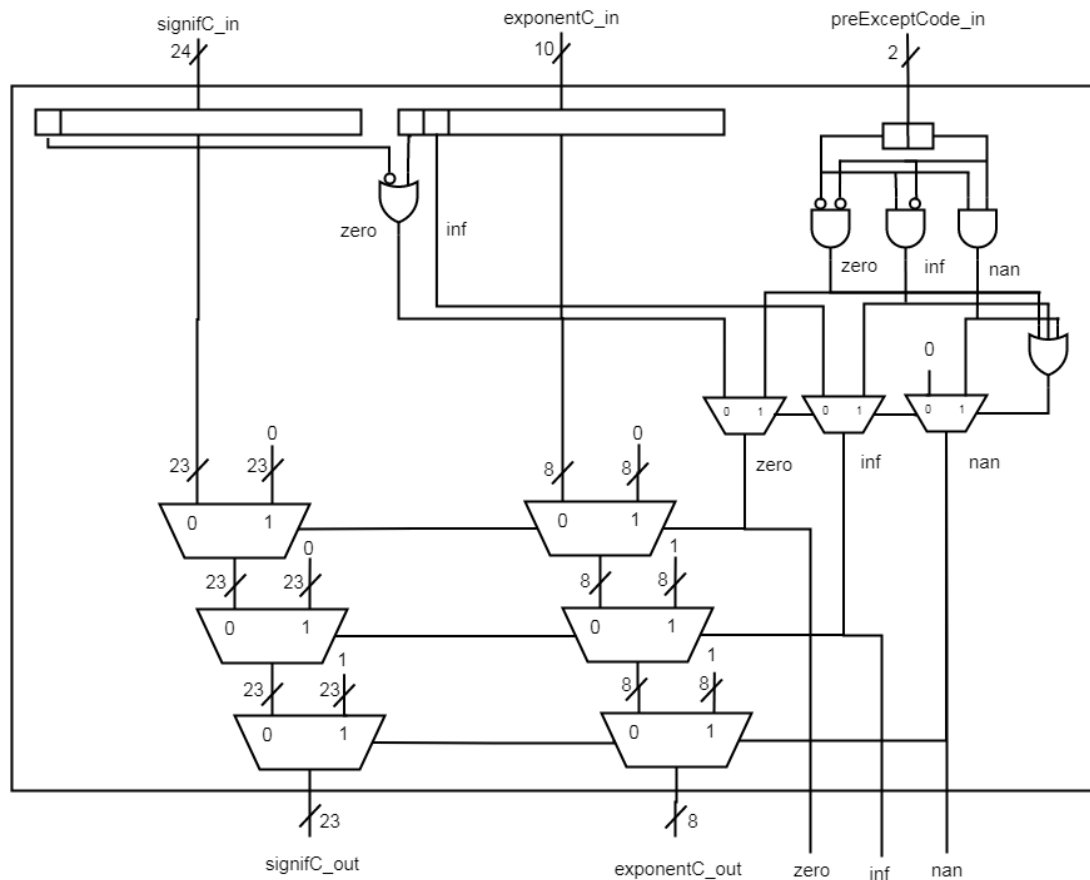


Рисунок 2.14 – Структура блоку перевірки на виняткову ситуацію

Якщо експонента числа більша за 255, то результат представляє нескінченність. Якщо експонента менша за нуль, то результат є денормалізованим числом і представляється у вигляді нуля. Для перевірки кожного з цих випадків експонента має два додаткові старші біти.

Про нуль також свідчить ситуація з нульовим старшим бітом мантиси результату. Така ситуація може виникнути у разі віднімання двох однакових чисел.

В блоці перевірки на виняткову ситуацію проводиться також перевірка коду виняткової ситуації, який був згенерований перед початком обчислення. Якщо код позначає одну з виняткових ситуацій, то перевага надається йому.

За допомогою мультиплексорів відбувається зміна значення мантиси та експоненти у випадку виникнення виняткової ситуації згідно з таблицею 1.1.

2.2.3 Опис блоку складання та віднімання на мові VHDL

Оскільки основною задачею даного проекту є розробка модуля обчислення з рухомою комою на FPGA, то після проектування структури необхідно описати кожний блок на мові VHDL з подальшою синтезацією схеми та пограмуванням мікросхеми.

На рисунках 2.15 та 2.16 зображено дві частини конвеєризованого блоку складання та віднімання чисел з рухомою комою. На рисунках зображено взаємне розміщення функціональних складових блоків блоку складання та віднімання, сигнали, якими блоки обмінюються даними та регістри. Зображено також розмір шин даних та їх назви.

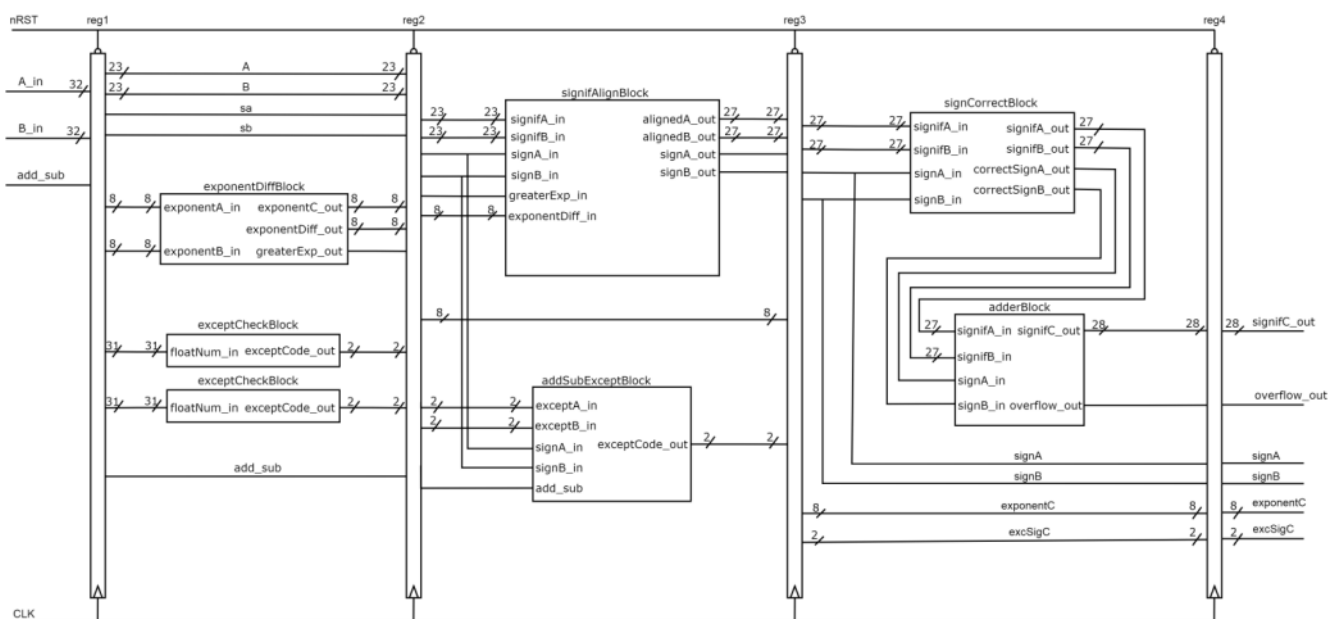


Рисунок 2.15 – Конвеєризований блок складання та віднімання

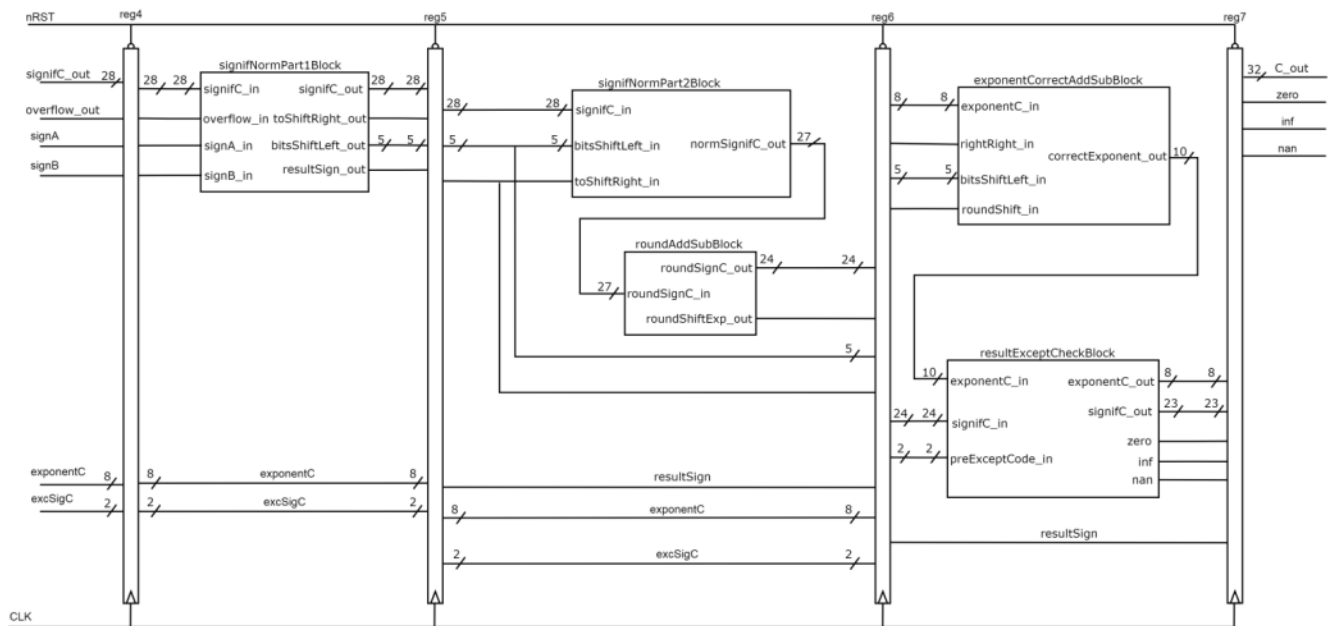


Рисунок 2.16 – Конверсізований блок складання та віднімання

На рисунку 2.17 представлено результати синтезування схеми на FPGA Altera Cyclone 3.

Family	Cyclone III
Device	EP3C5F256C6
Timing Models	Final
Total logic elements	834 / 5,136 (16 %)
Total combinational functions	725 / 5,136 (14 %)
Dedicated logic registers	361 / 5,136 (7 %)
Total registers	361
Total pins	102 / 183 (56 %)
Total virtual pins	0
Total memory bits	0 / 423,936 (0 %)
Embedded Multiplier 9-bit elements	0 / 46 (0 %)
Total PLLs	0 / 2 (0 %)

Рисунок 2.17 – Результати синтезування схеми на FPGA

З рисунку 2.17 видно, що синтезована схема займає 834 логічних елементів та 361 регістр, не використовуючи при цьому додаткові біти пам'яті та інші вбудовані ресурси.

2.3 Розробка блоку множення

2.3.1 Алгоритм і структура блоку множення чисел з рухомою комою

Блок множення був розроблений згідно зі стандартним алгоритмом множення двох чисел з рухомою комою. Алгоритм множення чисел з рухомою комою представлений на рисунку 2.18.

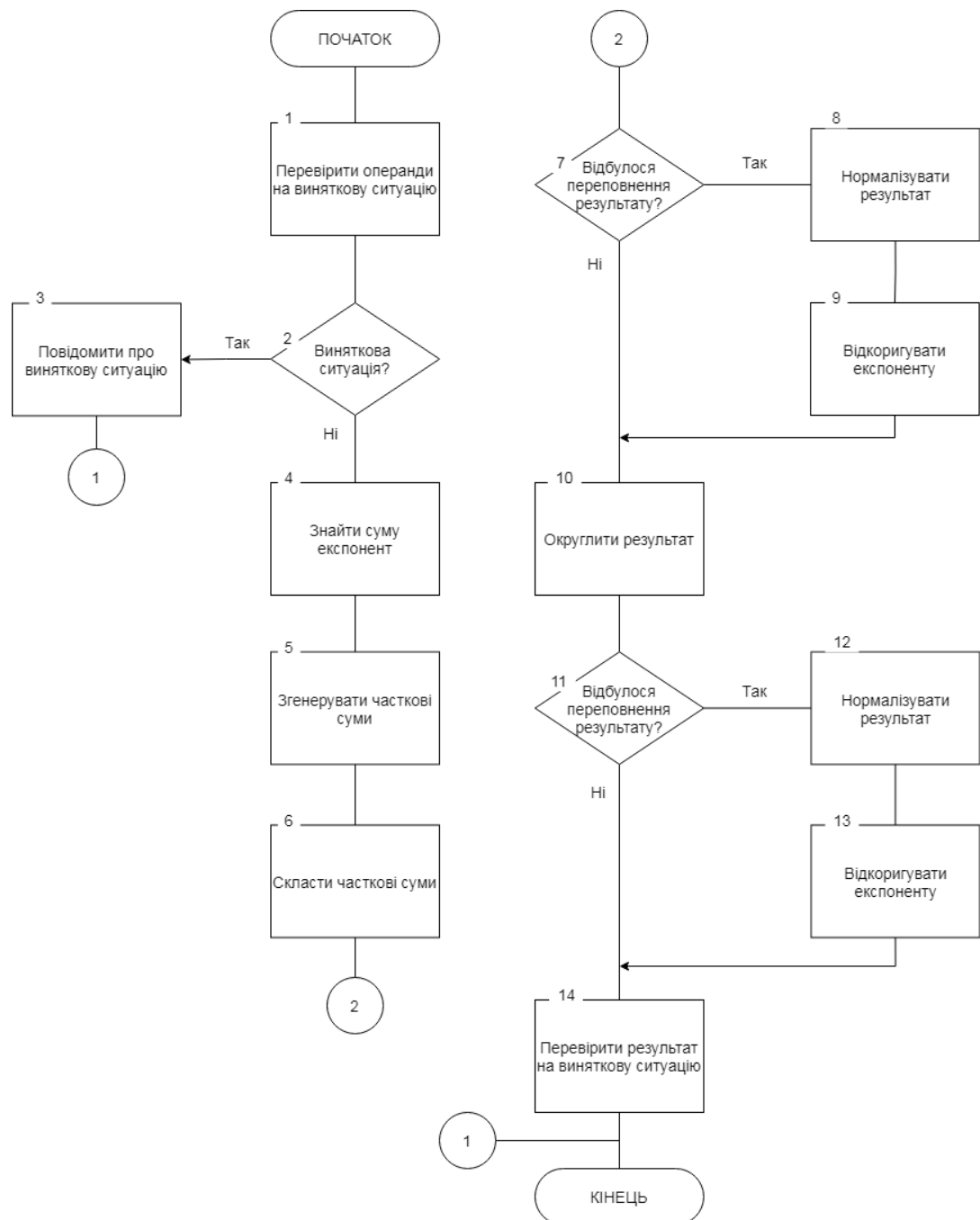


Рисунок 2.18 – Блок-схема алгоритму множення двох чисел з рухомою комою

Перед початком множення, як і у випадку зі складанням, відбувається перевірка чисел на виняткову ситуацію. Типові виняткові ситуації та їх вигляд у стандарті IEEE-754 представлені у таблиці 1.1.

На відміну від складання, для множення необхідно скласти дві експоненти. Числа з різними експонентами не зсувають, оскільки за правилами множення порядки чисел, які представлені експонентами, складаються, а мантиси перемножуються.

Після складання експонент генеруються часткові суми. Кожна часткова сума представляє собою добуток множеного та одного чи декількох біт множника.

Після генерації часткових сум відбувається їх складання. Даний процес є найскладнішим, оскільки потребує багато обчислювальних ресурсів. Досліджено багато способів складання часткових сум, деякі з них описано в розділі 1.4.

Після складання часткових сум результат нормалізується. Нормалізація завжди виконується зсувом вправо на один біт, оскільки мантиса в результаті не може бути менша за один і більша за чотири. Після нормалізації мантиси виконується корекція експоненти.

Для більш точного обчислення результат округлюється. При округленні може також виникнути ситуація з виходом результату за розрядну сітку. В такому випадку число зсувається на один біт вправо, а експонента збільшується на одиницю.

Отриманий результат повторно перевіряється на виняткову ситуацію.

Згідно з розглянутим алгоритмом множення чисел з рухомою комою було розроблено структуру блоку множення. Запропонована структура блоку множення зображена на рисунку 2.19. У порівнянні з блоком складання і віднімання, блок множення володіє більш простою структурою, що пов'язана з простотою алгоритму множення.



Рисунок 2.19 – Структура блоку множення чисел з рухомою комою

Як і у випадку з блоком складання та віднімання в блоці множення операнди спочатку надходять до блоку перевірки результату на виняткову ситуацію.

Помножувач мантис, представлений у схемі на рисунку 2.19 виконує функцію генерації часткових сум та їх складання.

Після множення мантис результат надходить на блоки нормалізації та округлення. Нормалізація після множення мантис відбувається тільки одним зсувом вправо, тому що величина результуючої мантиси не більше чотирьох. Кожен з цих блоків генерує сигнал збільшення експоненти, який надходить до блоку корекції експоненти.

Блок обчислення знаку результату представляє собою звичайну операцію «виключного або».

Враховуючи розроблену структуру блоку множення було спроектовано його конвеєризовану схему, яка представлена на рисунку 2.20. Конвеєризація проводилася таким чином, щоб максимально збалансувати кожен рівень конвеєру. Блок множення був розділений на два блоки: блок генерації часткових сум та блок складання часткових сум. Таким чином зменшено часову затримку розповсюдження сигналу, оскільки блок множення є вузьким місцем схеми, зображеної на рисунку 2.20.

З рисунку 2.20 видно, що в одному рівні конвеєру розміщено ті функціональні блоки, які незалежні один від одного і можуть виконувати обчислення паралельно. Так, наприклад, в першому рівні конвеєру розміщено одночасно чотири блоки: блок перевірки на виняткову ситуацію, блок обчислення знаку, блок складання експонент та блок генерації часткових сум. Оскільки результати обчислень кожного з цих блоків не впливають один на одного, то операції на них можуть виконуватися паралельно. Блок нормалізації результату та блок округлення також знаходяться в одному рівні конвеєру, але з'єднані послідовно, оскільки перед округленням необхідно нормалізувати результуючу мантису.

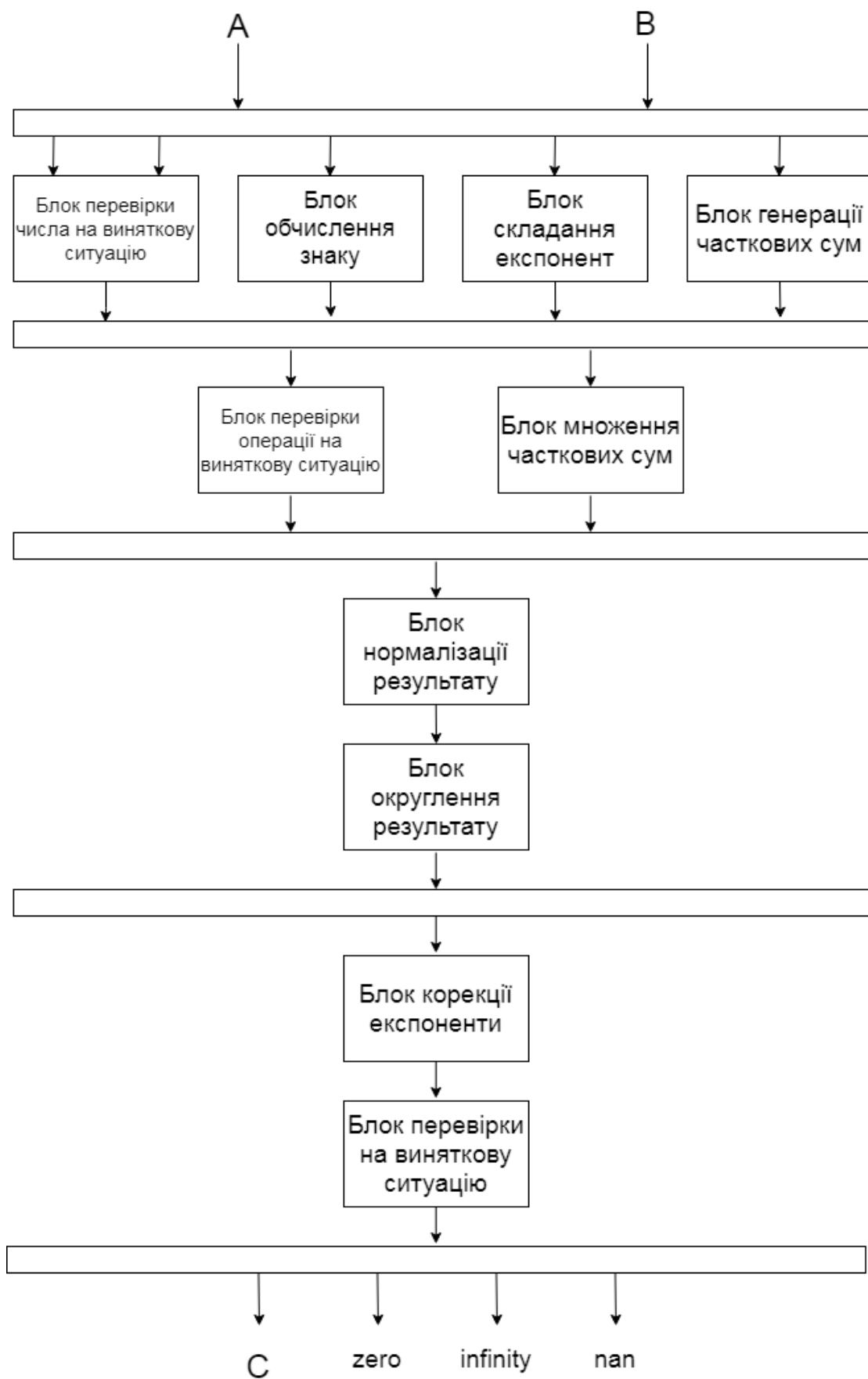


Рисунок 2.20 – Конвеєризована схема блоку множення чисел з рухомою комою

Конвеєризований блок множення містить меншу кількість рівнів, порівняно з блоком складання і віднімання і тому виконує обчислення з меншою кількістю тактових сигналів.

2.3.2 Розробка функціональних складових блоку множення

Блок перевірки на виняткову ситуацію схожий за принципом роботи з відповідним блоком в блоці складання та віднімання. Формат представлення виняткових ситуацій представлено у таблиці 1.1.

Блок обчислення знаку представляє собою операцію «виключне або», аргументами якої є знаки двох чисел. Знак в процесі множення не змінюється.

На рисунку 2.21 зображено розроблену структуру блоку складання експонент.

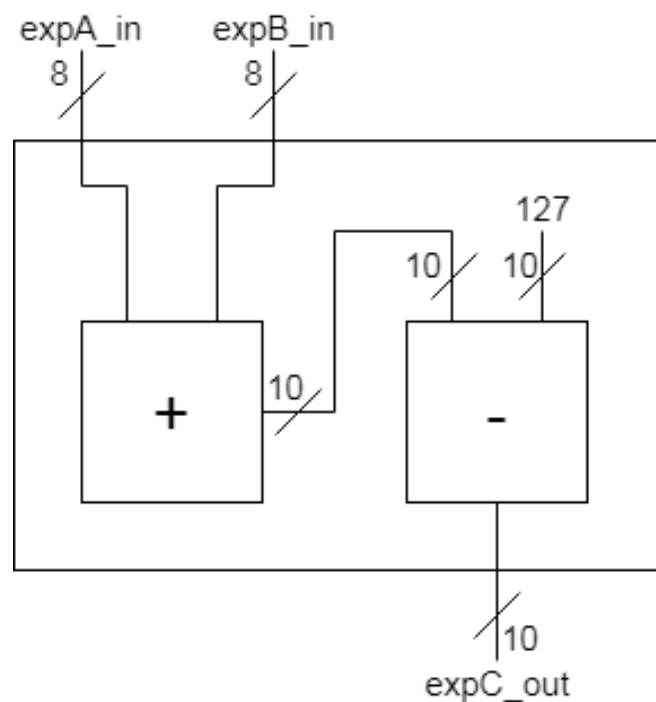


Рисунок 2.21 – Структура блоку складання експонент

Блок складається з суматора, який складає дві експоненти, та з вирахувача, котрий проводить корекцію результату. Необхідність вирахування значення 127 з результату складання викликано тим, що у форматі IEEE-754 експонента подається у вигляді суми власне значення експоненти та

зміщення, що дорівнює 127. В іншому випадку, значення результуючої експоненти було б неправильним і було б більшим на 127.

Результат «exp_out» записується за допомогою десяти біт, оскільки при додаванні може виникнути ситуація переповнення або результат може бути менше нуля.

Блок перевірки операції на виняткову ситуацію виявляє заздалегідь, яка виняткова ситуація буде згенерована в результаті операції множення.

В таблиці 2.3 представлено можливі варіанти виняткової події від тих чи інших значень двох операндів, які позначаються як «ОП1» та «ОП2» відповідно.

Таблиця 2.3 – Таблиця виняткових ситуацій для операції множення

ОП1 ОП2	NaN	+Infinity	-Infinity	+0	-0	+Num	-Num
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
+Infinity	NaN	+Infinity	-Infinity	NaN	NaN	+Infinity	-Infinity
-Infinity	NaN	-Infinity	+Infinity	NaN	NaN	-Infinity	+Infinity
+0	NaN	NaN	NaN	+0	-0	+0	-0
-0	NaN	NaN	NaN	-0	+0	-0	+0
+Num	NaN	+Infinity	-Infinity	+0	-0	+Num	-Num
-Num	NaN	+Infinity	+Infinity	-0	+0	-Num	+Num

Блок генерації часткових сум утворює часткові суми в залежності від значення двох бітів множника. Таким чином реалізується множення за основою 4. Виконуючи множення за основою 4 зменшується блок складання часткових сум, оскільки в такому випадку кількість часткових сум менша, ніж в звичайному множенні.

На рисунку 2.22 зображено загальну структуру помножувача. З рисунку видно, що помножувач складається з блоку генерації часткових сум та з дерева суматорів зі збереженням переносу. Часткові суми, потрапляючи до суматорів зі збереженням переносу, складаються один з одним, а отриманий вектор переносу і суми потрапляє до звичайного суматору з послідовним переносом. Вибір суматору з послідовним переносом обумовлений тим, що в мікросхемах FPGA наявні оптимізовані схеми переносу.

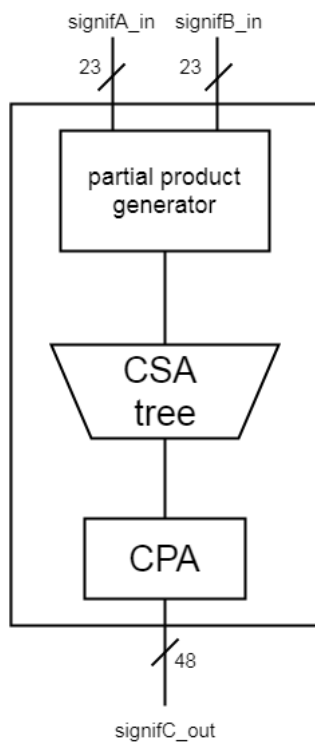


Рисунок 2.22 – Структура помножувача

Запропонована структура блоку складання часткових сум представлений на рисунку 2.23.

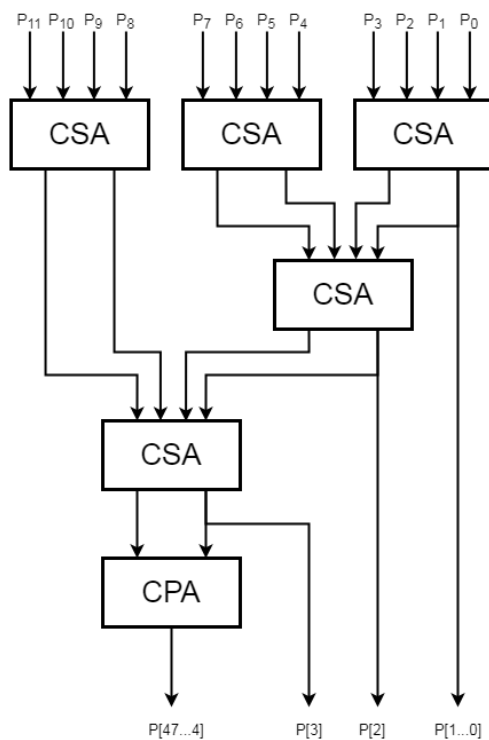


Рисунок 2.23 – Структура блоку складання часткових сум

Блок складання часткових сум представляє собою дерево Воллеса, основні характеристики якого було описано в розділі 1.4. З рисунку видно, що дерево складається з чотирьох рівнів 4-входових суматорів зі збереженням переносу та одного суматору з послідовним переносом, який складає вектор суми та переносу.

З рисунку також видно, що чотири біти добутку вираховуються до кінця обчислень.

В даній роботі для кінцевої операції додавання обрано суматор з послідовним перенесенням, оскільки для нього у внутрішній структурі FPGA вже розроблено і оптимізовано шляхи для розповсюдження сигналу перенесення.

Запропоновану структуру блоку нормалізації результату представлено на рисунку 2.24.

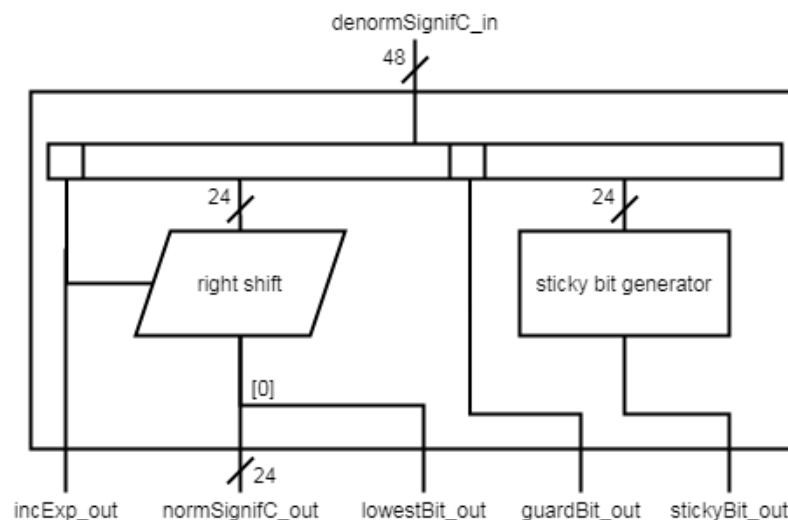


Рисунок 2.24 – Блок нормалізації результату

З рисунку видно, що денормалізована мантиса «denormSignifC_in» розбивається на декілька частин, над кожною з яких проводяться певні обчислення. Так, з молодшої половини мантиси генерується один з бітів округлення «stickyBit_out», старша половина зсувається на один біт вправо в залежності від значення найстаршого біта. Даний біт також сигналізує про необхідність корекції експоненти.

Розроблену структуру блоку округлення зображено на рисунку 2.25.

На відміну від блоку складання та віднімання, у блоці множення округлення відбувається за участі трьох бітів. Визвано це відсутністю зсувів вліво, для чого знадобився б додатковий біт округлення. Правила округлення такі ж самі, як і у випадку з блоком складання та віднімання.

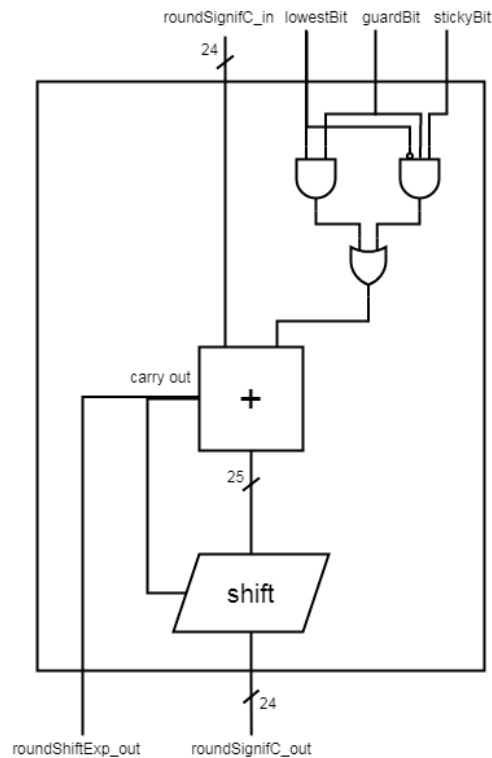


Рисунок 2.25 – Структура блоку округлення результату

Якщо при округленні результат виходить за розрядну сітку, то генерується сигнал «roundShiftExp_out» про необхідність збільшення експоненти.

На рисунку 2.26 зображено розроблену структуру блоку корекції експоненти.

З рисунку видно, що при появі одного з сигналів «incExpNorm_in» або «incExpRound_in» на вході, котрі позначають відповідно переповнення при нормалізації та округлені, відбувається збільшення експоненти на одиницю.

Якщо на вхід подано одночасно два сигнали, то експонента збільшується на два.

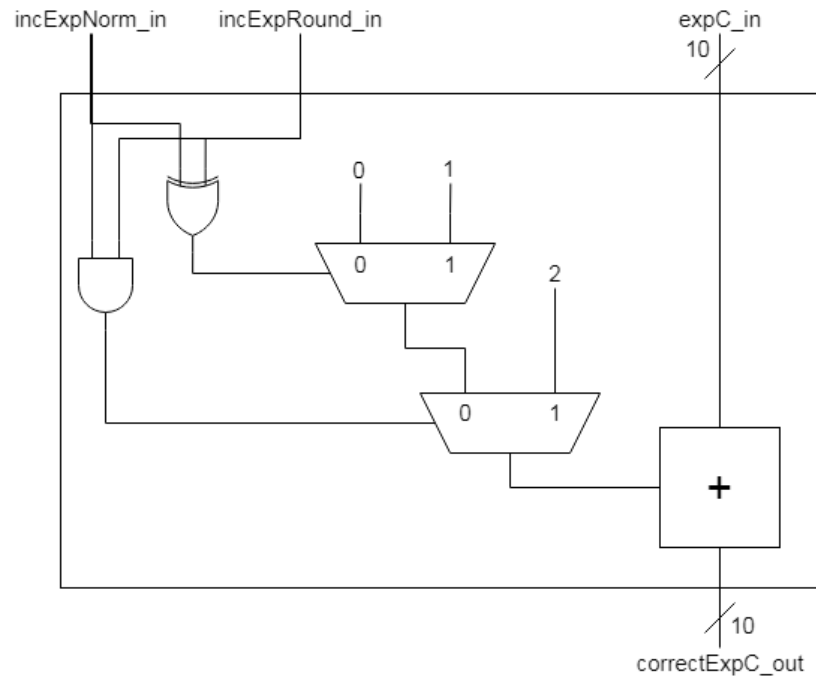


Рисунок 2.26 – Структура блоку корекції експоненти

Блок перевірки на виняткову ситуацію за структурою такий самий, як і у випадку з блоком складання та віднімання, оскільки основні правила стандарту IEEE-754 розповсюджуються однаково на всі математичні операції.

2.3.3 Опис блоку множення на мові VHDL

Як і у випадку з блоком складання та віднімання, блок множення був описаний на мові VHDL та синтезований засобами розробки Quartus II з подальшим програмуванням мікросхеми.

На рисунку 2.27 зображено конвеєризований блок множення чисел з рухомою комою. Так само, як і для блоку складання та віднімання, на рисунку зображено взаємне розміщення функціональних складових блоків блоку складання та віднімання, сигнали, якими блоки обмінюються даними та регістри. Зображено також розмір шин даних та їх назви.

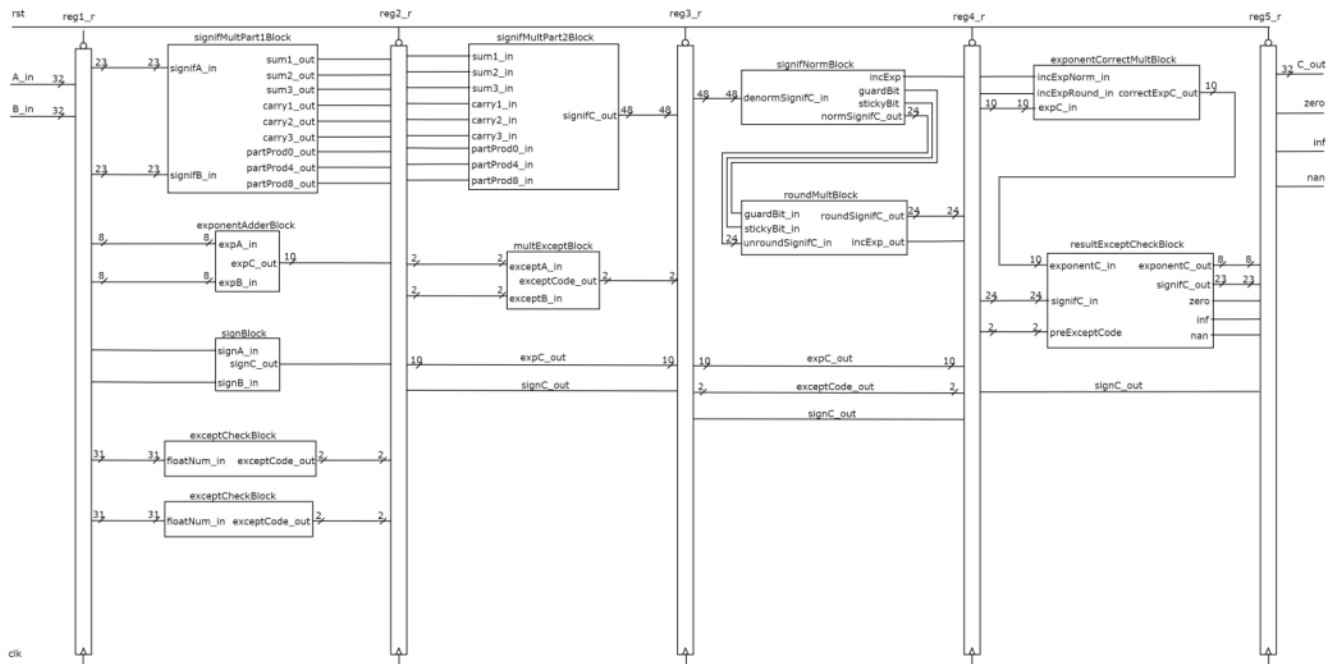


Рисунок 2.27 – Конверсізований блок множення

На рисунку 2.28 представлено результати синтезування схеми на FPGA Altera Cyclone 3.

Total logic elements	1,406 / 5,136 (27 %)
Total combinational functions	1,373 / 5,136 (27 %)
Dedicated logic registers	391 / 5,136 (8 %)
Total registers	391
Total pins	101 / 183 (55 %)
Total virtual pins	0
Total memory bits	0 / 423,936 (0 %)
Embedded Multiplier 9-bit elements	0 / 46 (0 %)
Total PLLs	0 / 2 (0 %)

Рисунок 2.28 – Результати синтезування схеми на FPGA

З рисунку видно, що синтезована схема займає 1,406 логічних елементів та 391 регістр, не використовуючи при цьому додаткові ресурси, такі як біти пам'яті та вбудовані помножувачі.

2.4 Розробка блоку ділення

Блок ділення функціонує згідно зі стандартним алгоритмом ділення двох чисел з рухомою комою. Алгоритм ділення чисел з рухомою комою представлений на рисунку 2.29.

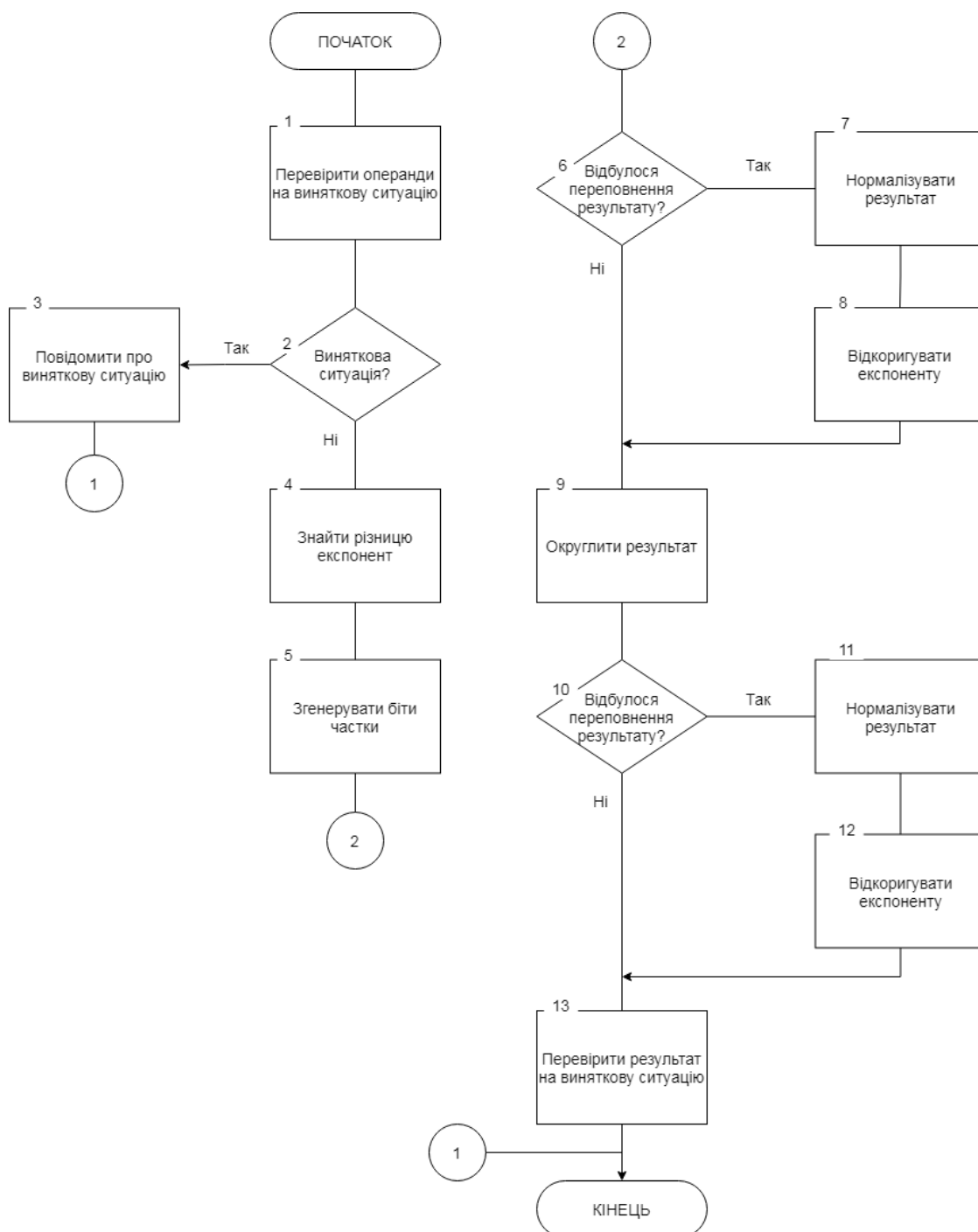


Рисунок 2.29 – Блок-схема алгоритму ділення чисел з рухомою комою

Як видно з рисунку 2.29 алгоритм ділення двох чисел з рухомою комою схожий на алгоритм множення двох чисел з рухомою комою. Так само і структура, що представлена на рисунку 2.30, схожа на структуру блоку множення.



Рисунок 2.30 – Структура блоку ділення чисел з рухомою комою

На рисунку 2.31 зображено конвеєризований блок ділення чисел з рухомою комою.

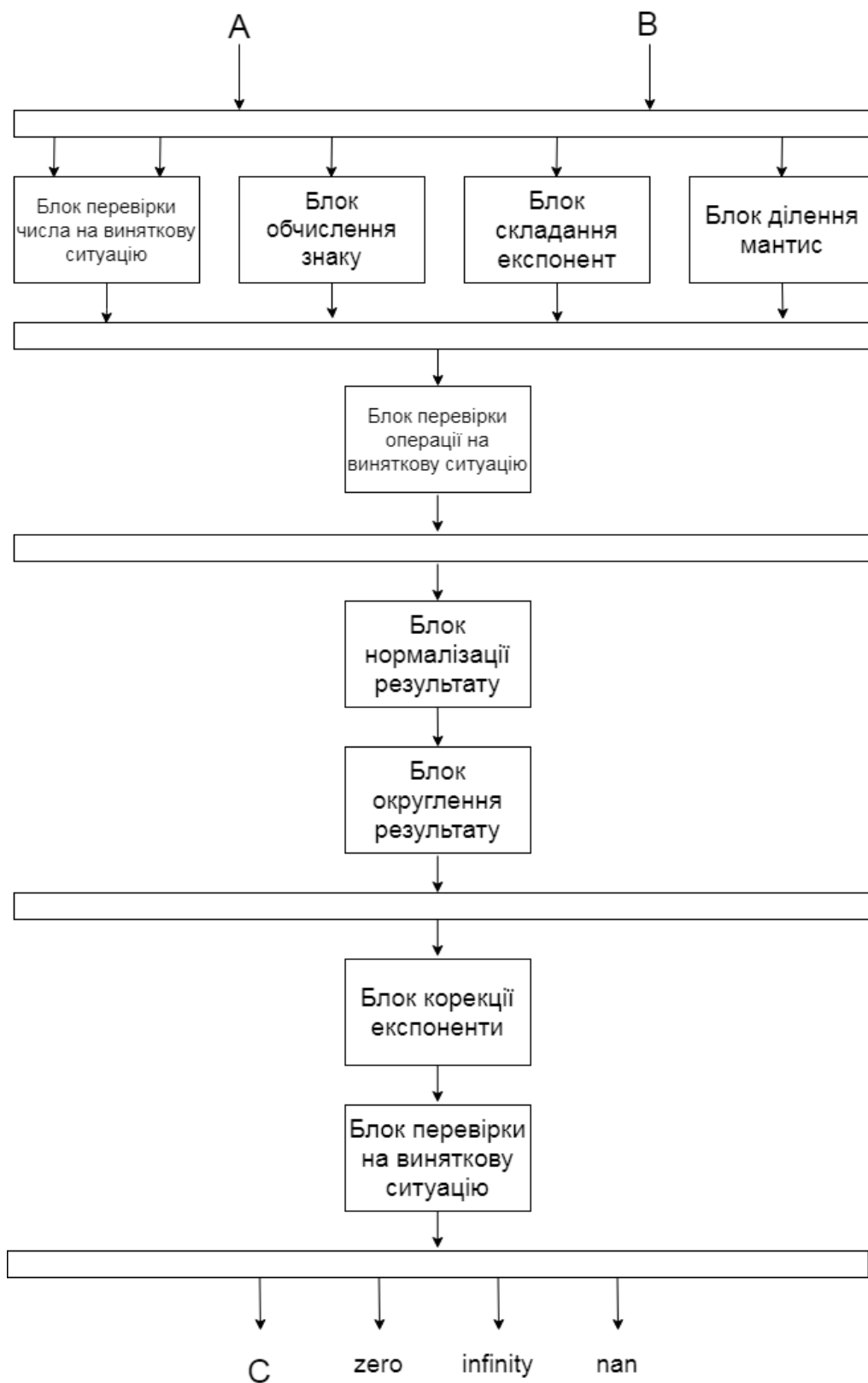


Рисунок 2.31 – Конвеєризований блок ділення чисел з рухомою комою

Блок ділення мантис, що є найскладнішим елементом, є вузьким місцем конвеєру, тому максимальна частота конвеєру визначається затримкою на цьому блоці.

3 МОДЕЛЮВАННЯ І ТЕСТУВАННЯ МОДУЛЯ ОБЧИСЛЕНЬ З РУХОМОЮ КОМОЮ

3.1 Моделювання модуля обчислень з рухомою комою

Для моделювання роботи модуля обчислень було розроблено тестове середовище, структура якого зображена на рисунку 3.1.

Тестове середовище складається з блоку DUT, що проходить тестування, з функцій читання вхідних даних та запису результатів до файлів. Користуючись тестовим середовищем, зображеним на рисунку 3.1 можна моделювати поведінку пристрою на великих об'ємах даних, позбавляючи себе від ручної роботи з введення вхідних даних.

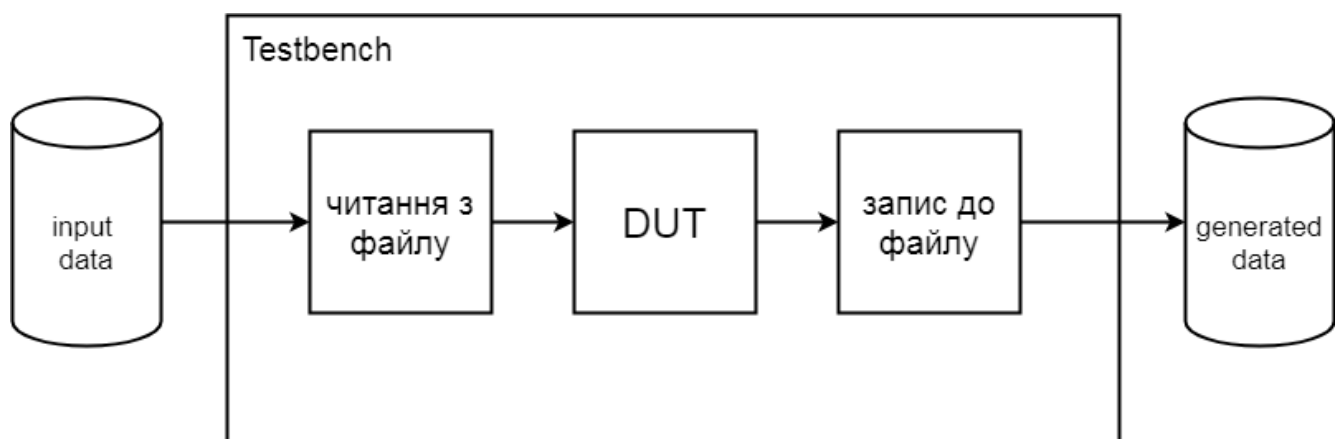


Рисунок 3.1 – Тестове середовище

Для створення масивів вхідних даних та перевірки згенерованих модулем обчислень файлів роботи було розроблено програму на мові Java. Програма

генерує тестовий масив даних, обчислює результат відповідно до вказаної операції та вираховує різницю між ним та результатом від модуля обчислень.

На рисунку 3.2 зображено UML-діаграму класів програми тестування.

Як видно з діаграми, програма складається з двох класів, що поєднані відношенням «асоціація». Таким чином, клас `TestDataProcessor` може викликати об'єкт класу `TextProcessor` та робити операції від його ім'я.

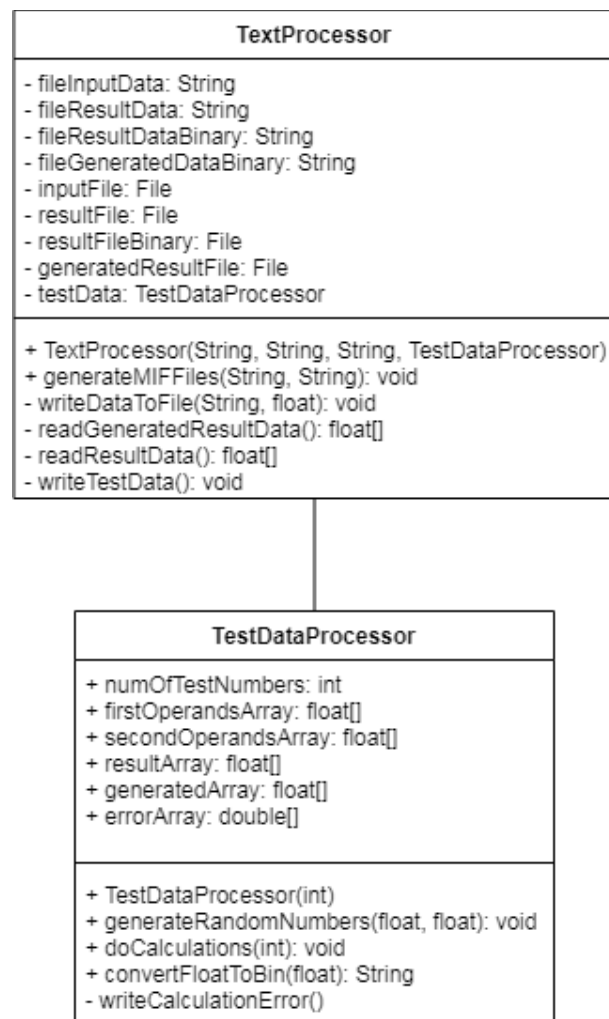


Рисунок 3.2 – Діаграма класів програми тестування

На рисунку 3.3 зображено приклад генерації вхідних даних.

```

C:\PROG\NetBeans\TestFPGA\dist>java -jar TestFPGA.jar
Enter test numbers quantity: 10000

1-Generate test data file
2-Test data
Enter operation mode: 1

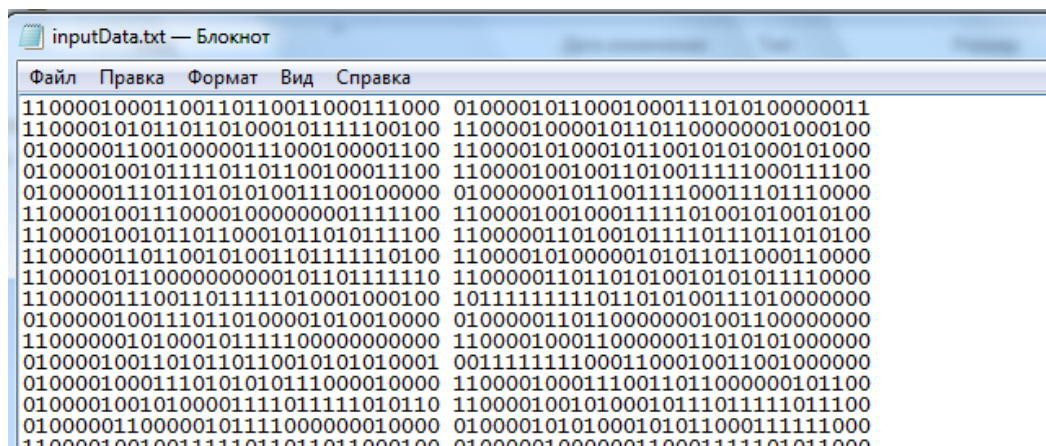
0-Addition
1-Subtraction
2-Multiplication
Enter operation code: 0

Enter range -100.0 100.0

```

Рисунок 3.3 – Приклад генерації вхідних даних

Рисунок 3.3 демонструє генерування масиву даних довжиною 10000 для операції додавання з числами з експонентою в діапазоні від -100 до 100. На рисунку 3.3 зображено приклад вхідного файлу з даними. Дані в файлі представлені у двійковому вигляді.



The screenshot shows a text editor window with the title 'inputData.txt — Блокнот'. The menu bar includes 'Файл', 'Правка', 'Формат', 'Вид', and 'Справка'. The main text area contains a large block of binary data, consisting of two columns of 0s and 1s, representing test data for an FPGA testbench.

Рисунок 3.4 – Вхідний файл з даними

Файл, що зображений на рисунку 3.4 надходить до тестового середовища, написаного на VHDL. Згодом файл перенаправляється до модуля обчислень з рухомою комою, який робить відповідні до коду операції обчислення. Функція запису результатів отримує дані та записує їх у відповідний файл.

На рисунку 3.5 зображено часову діаграму роботи модуля обчислень у середовищі Modelsim.

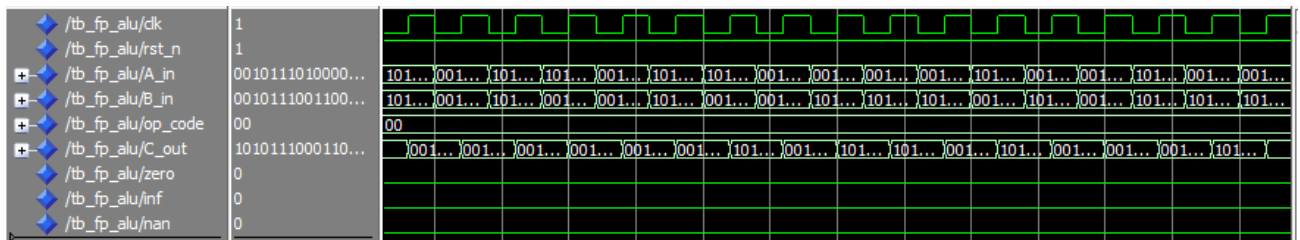


Рисунок 3.5 – Часова діаграма роботи модуля обчислень в Modelsim

На рисунку 3.6 зображено результати перевірки отриманих від модуля обчислень даних.

```
C:\PROG\NetBeans\TestFPGA\dist>java -jar TestFPGA.jar
Enter test numbers quantity: 100000
1-Generate test data file
2-Test data
Enter operation mode: 2
Average error = 0.0
```

Рисунок 3.6 – Результати перевірки отриманих даних

З рисунку 3.6 видно, що програма перевірки згенерованих модулем обчислень значень, аналізує отримані дані і виводить середню помилку обчислень.

3.2 Тестування модуля обчислень з рухомою комою

На рисунку 3.7 зображено відлагоджувальну плату, що була використана для реалізації модуля обчислень з рухомою комою. В основі плати знаходиться мікросхема FPGA Cyclone IV EP4CE6E22C8N.



Рисунок 3.7 – Відлагоджувальна плата

Для тестування модуля обчислень було використано утиліту SignalTap II Logic Analyzer, що складу середовища розробки Quartus II.

Логічний аналізатор SignalTap II – це засіб відлагодження, який фіксує та відображає поведінку сигналів в режимі реального часу в системі на програмованому кристалі (SOPC) або будь-якій FPGA. На рисунку 3.8 зображено структуру SignalTap II.

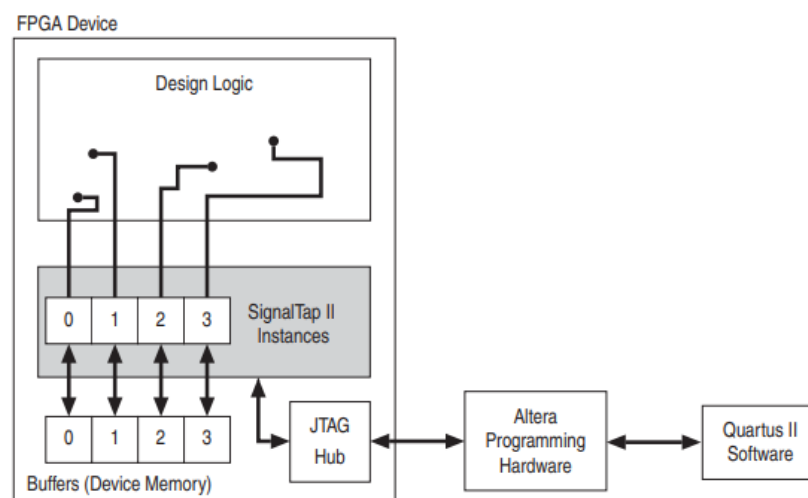


Рисунок 3.8 – Структура SignalTap II [35]

З рисунку 3.8 видно, що SignalTap II створює свій об'єкт, що програмується у мікросхемі поруч з проектом, що належить відлагодженню. Об'єкт SignalTap II

відстежує зовнішні та внутрішні сигнали, записуючи їх до своєї пам'яті. Зв'язок з об'єктом здійснюється за допомогою інтерфейсу JTAG.

Оскільки кількість робочих виводів в мікросхемі не дозволяє реалізувати на ній модуль обчислення, то було розроблено модуль тестування, який зберігає вхідні дані в RAM-пам'яті, виводи якої під'єднанні до входів модуля обчислень.

На рисунку 3.9 зображено структуру модуля тестування.

Завдяки тому, що вхідні дані зберігаються в пам'яті FPGA, полегшується процес тестування модуля обчислень, оскільки результати вводяться автоматично за тактовим сигналом.

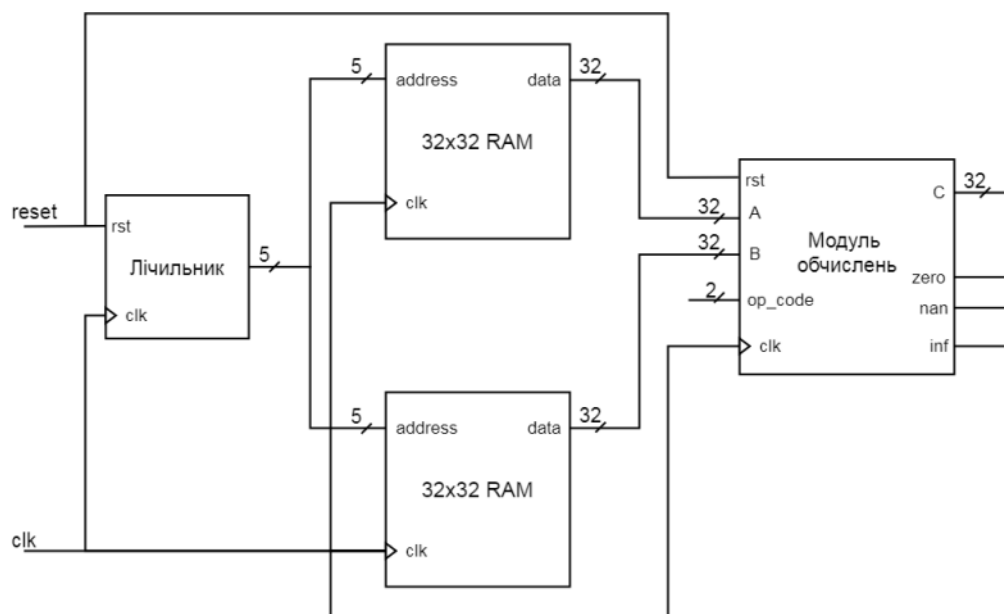


Рисунок 3.9 – Структура модуля тестування

На рисунку 3.10 зображено часову діаграму, отриману в SignalTap II Logic Analyzer.

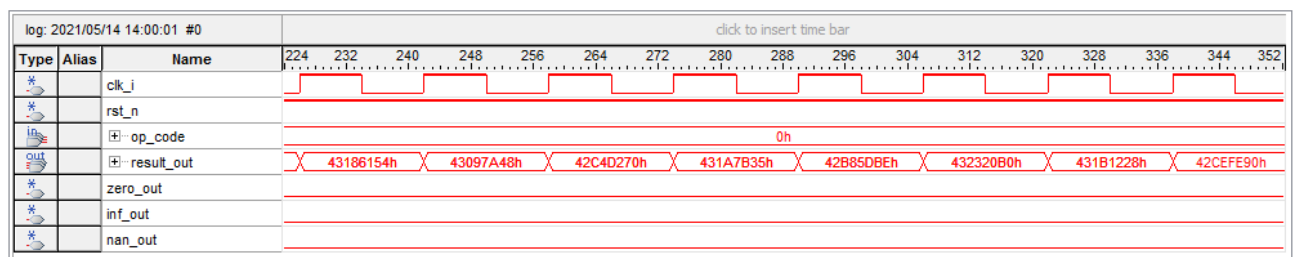


Рисунок 3.10 – Часова діаграма модуля обчислень на FPGA Cyclone IV

Як і очікувалось, результати тестування, представлені на рисунку 3.10 збігаються з результатами моделювання, представленими на рисунку 3.6.

3.3 Аналіз результатів моделювання і тестування модуля обчислень

На рисунках 3.11 – 3.13 зображено результати моделювання модуля обчислень у середовищі Modelsim на трьох операціях з масивом даних в 10^5 та діапазоном експонент від -100 до 100.

```
Enter test numbers quantity: 100000

1-Generate test data file
2-Test data
Enter operation mode: 2
Average error = 1.166337174903934E-5
```

Рисунок 3.11 – Результати моделювання операції додавання

```
Enter test numbers quantity: 100000

1-Generate test data file
2-Test data
Enter operation mode: 2
Average error = 1.1659266621993285E-5
```

Рисунок 3.12 – Результати моделювання операції віднімання

```
Enter test numbers quantity: 100000

1-Generate test data file
2-Test data
Enter operation mode: 2
Average error = 0.0
```

Рисунок 3.13 – Результати моделювання операції множення

З даних результатів моделювання можна прийти висновку, що модуль обчислень працює правильно, представляючи точний результат у стандарті IEEE-754. Уважно проаналізувавши згенерований модулем обчислень файл з результатами та файл, згенерований програмою перевірки, можна побачити, що помилку в обчисленнях вносить останній біт деяких чисел. Дана проблема може бути викликана не самим модулем обчислень, а неточністю представлення числа з рухомою комою у комп'ютері.

Тестування на реальному обладнанні, а саме на мікросхемі FPGA Cyclone IV EP4CE6E22C8N показало, що реалізований модуль обчислень працює коректно та відповідає очікуваним результатам, отриманим в процесі моделювання в середовищі Modelsim.

В подальшому планується збільшити масив даних для тестування та дослідити статистичні залежності між значеннями вхідних даних та отриманою похибкою обчислень.

ВИСНОВКИ

У результаті роботи досліджено основні методи побудови суматорів, помножувачів та дільників чисел з рухомою комою. Досліджено переваги та недоліки різних алгоритмів виконання арифметичних дій над числами у двійковій арифметиці.

Розроблено структуру блоків складання, віднімання, множення і ділення чисел з рухомою комою, описано їх складові та принципи роботи. Розроблені блоки складання, віднімання, множення і ділення конвеєризовано для збільшення пропускної здатності.

Проведено моделювання роботи розробленого модуля обчислень з рухомою комою у середовищі Modelsim. Моделювання на великому об'ємі даних, згенерованому засобами спеціальної програми написаної на мові програмування Java, показало коректність виконання арифметичних операцій з рухомою комою згідно зі стандартом IEEE-754.

Розроблену структуру модуля обчислень описано засобами мови VHDL та реалізовано на FPGA Cyclone IV. Проведено тестування модуля обчислень на реальному обладнанні за допомогою засобу тестування SignalTap II Logic Analyzer. Аналіз отриманих результатів показав, що модуль обчислення коректно виконує арифметичні операції на мікросхемі FPGA, відповідаючи результатам, отриманим під час моделювання. Таким чином, можна використовувати даний модуль обчислення у реальному обладнанні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Milos D. Ercegovac. Digital Arithmetic / M. D. Ercegovac, T. Lang. – San Francisco: Morgan Kaufman Publisher, 2004. – 709 p. – ISBN 1-55860-798-6.
2. Handbook of Floating-Point Arithmetic / J. Muller, N. Brisebarre, F. Dinechin. – Baser : Birkhäuser, 2010. – 572 p. – ISBN 978-0-8176-4705-6.
3. Parhami, B. Computer Arithmetic: Algorithms and Hardware Designs, 2nd edition / B. Parhami. – New York : Oxford University Press, 2010. – 672 p. – ISBN 978-0-19-532848-6.
4. Clenshaw, C. Beyond Floating Point / C.W. Clenshaw, F.W.J. Olver // Journal of the Association of Computing Machinery. – 1984. – Vol. 31, Issue 2. – P.319–328.
5. Kuck, D. Analysis of Rounding Methods in Floating-Point Arithmetic / D. Kuck, D.S. Parker, A. Sameh // IEEE Transactions on Computers. – 1977. – Vol. 26, Issue 7. – P. 643–650.
6. Standard for Floating-Point Arithmetic : IEEE Std 754-2008. – Los Alamitos: IEEE, 2008. – P. 1–70.
7. Eldon, J. A floating point format for signal processing / J. Eldon, C. Robertson//IEEE International Conference on Acoustics, Speech, and Signal Processing, Paris, 3-5 May 1982 : proceedings. – Los Alamitos: IEEE, 2003. – P. 717–720.
8. Lienhart, G. Using Floating-Point Arithmetic on FPGAs to Accelerate Scientific N-Body Simulations / G. Lienhart, A. Kugel, R. Manner // Annual IEEE Symposium on Field-Programmable Custom Computing Machines : Tenth annual conference, Napa, 24-24 April 2002 : proceedings. – Los Alamitos: IEEE, 2003. – P. 182–194.
9. Alvarez, C. Fuzzy Memoization for Floating-Point Multimedia Applications / C. Alvarez, J. Corbal, M. Valero // IEEE Transactions on Computers. – 2005. – Vol. 54, Issue 7. – P. 922–927.

10. Kuon, I. FPGA Architecture Survey and Challenges / I. Kuon, R. Tessier, J. Rose. – Hanover : Now Foundations and Trends, 2008. – 124p. – ISBN 9781601981264.
11. Cyclone III Device Handbook [Электронный ресурс]. – Режим доступа: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyc3/cyclone3_handbook.pdf.
12. Teubner, J. Data processing on FPGA / J. Teubner, L. Woods. – San Rafael : Morgan & Claypool publisher, 2013. – 118p. – ISBN 9781627050616.
13. APEX II Programmable Logic Device Family Data Sheet [Электронный ресурс]. – Режим доступа: https://www.ece.iastate.edu/~zambreno/classes/cpre583/documents/altera/ds_ap2.pdf.
14. Amano, H. Principles and structures of FPGA / H. Amano. – Berlin : Springer, 2018. – 240p. – ISBN 978-9811308239.
15. Tessier, R. Balancing Logic Utilization and Area Efficiency in FPGAs / R. Tessier, H. Giza // Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing. – Berlin : Springer, 2002. – P.535–544. – ISBN 978-3-540-44614-9.
16. P. Jamieson An Energy and Power Consumption Analysis of FPGA Routing Architectures / P. Jamieson, W. Luk, S.J.E. Wayne // 2009 International Conference on Field-Programmable Technology, Sydney, 9-11 December 2009 : proceedings. – Los Alamitos: IEEE, 2010. – P.324–327.
17. Leong, P. Recent Trends in FPGA Architectures and Applications / P.H.W. Leong // 4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008), Hong Kong, 23-25 January 2008 : proceedings. – Los Alamitos: IEEE, 2008. – P.137–141.
18. Asano, S. Performance comparison of FPGA, GPU and CPU in image processing / S. Asano, T. Maruyama, Y. Yamaguchi // 2009 International Conference on Field Programmable Logic and Applications, Prague, 31 August - 2 September 2009 : proceedings. – Los Alamitos: IEEE, 2009. – P.126 –131.

19. Burks, A. Preliminary Discussion of the Logical Design of an Electronic Computing Instrument / A.W. Burks, H.H. Goldstine, J. Neumann // *The Origins of Digital Computers: Selected Papers*. – Berlin : Springer, 1982. – P.399–413. – ISBN 978-3-642-61812-3.

20. Pai, Y. The fastest carry lookahead adder / Y. Pai, Y. Chen // *Proceedings. DELTA 2004. Second IEEE International Workshop on Electronic Design, Test and Applications*, Perth, 28-30 January 2004 : proceedings. – Los Alamitos: IEEE, 2005. – P.434–436.

21. Lehman, M. Skip Techniques for High-Speed Carry-Propagation in Binary Arithmetic Units / M. Lehman, N. Burla // *IRE Transactions on Electronic Computers*. – 1961. – Vol. 10, Issue 4. – P.691–698.

22. Bedrij, O. Carry-Select Adder / O. Bedrij // *IRE Transactions on Electronic Computers*. – 1962. – Vol. 11, Issue 3. – P.340–346.

23. Tyagi, A. A reduced-area scheme for carry-select adders / A. Tyagi // *IEEE Transactions on Computers*. – 1993. – Vol. 42, Issue 10. – P.1163–1170.

24. Amelifard, B. Closing the gap between carry select adder and ripple carry adder: a new class of low-power high-performance adders / B. Amelifard, F. Fallah, M. Pedram // *Sixth international symposium on quality electronic design (isqed'05)*, San Jose, 21-23 March 2005 : proceedings. – Los Alamitos: IEEE, 2005. – P.148–152.

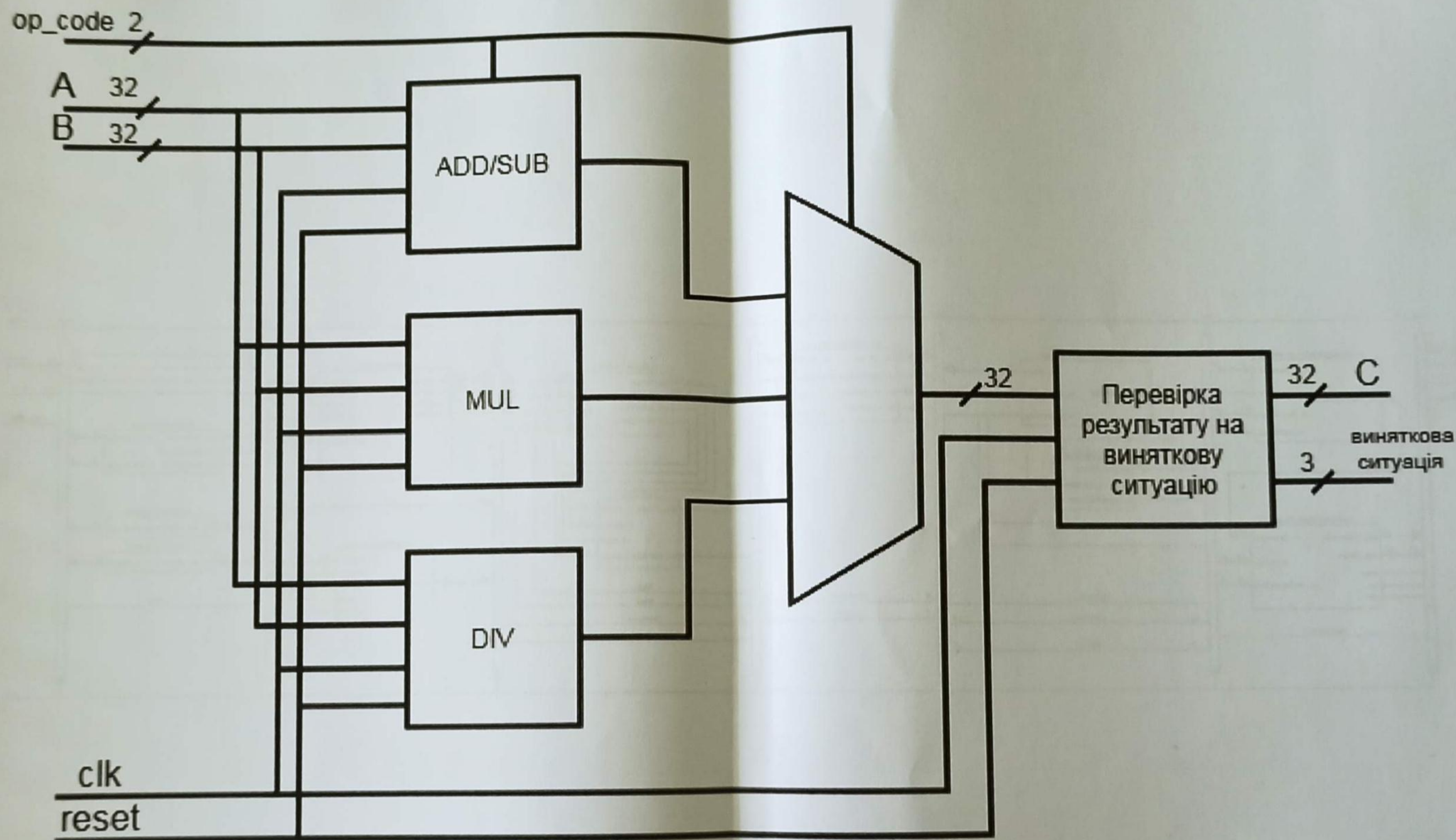
25. Marimuthu, C. Low Power Shift and Add Multiplier Design [Электронный ресурс] / C.N. Marimuthu, P. Thangaraj, A. Ramesan. – Режим доступа: <https://arxiv.org/ftp/arxiv/papers/1006/1006.1179.pdf>.

26. Wallace, C. A Suggestion for a Fast Multiplier / C.S. Wallace // *IEEE Transactions on Electronic Computers*. – 1964. – Vol. 13, Issue 1. – P. 14–17.

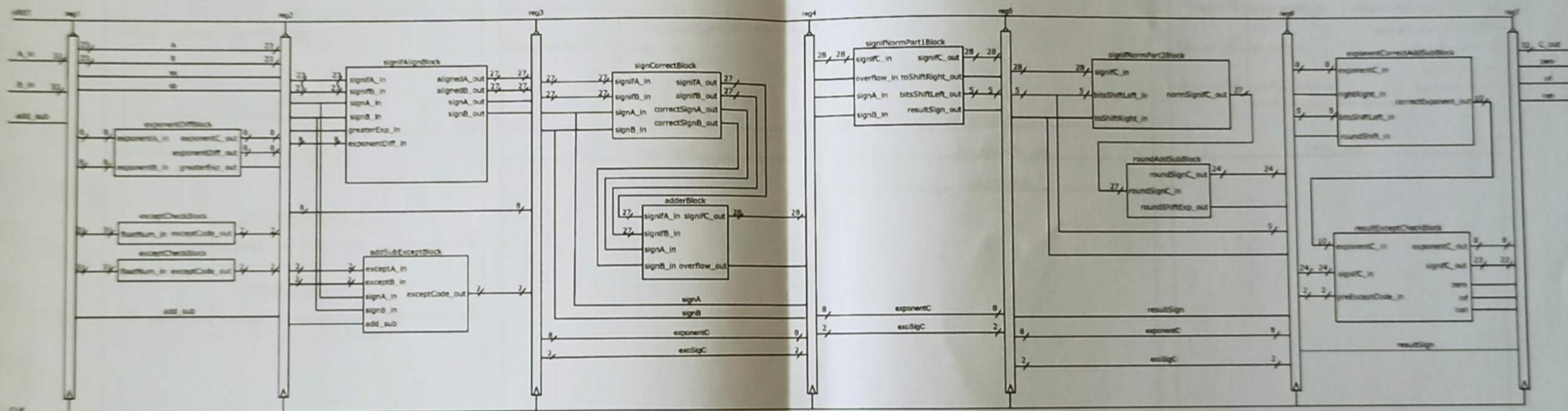
27. Koren, I. *Computer Arithmetic Algorithms 2nd Edition* / I. Koren. – Natick : A K Peters/CRC Press, 2001. – 296p. – ISBN 978-1568811604.

28. Gandhi, D. Comparative analysis for hardware circuit architecture of Wallace tree multiplier / D.R. Gandhi, N.N. Shah // *2013 International Conference on Intelligent Systems and Signal Processing (ISSP)*, Vallabh Vidyanagar, 1-2 March 2013 : proceedings. – Los Alamitos: IEEE, 2013. – P.1–6.

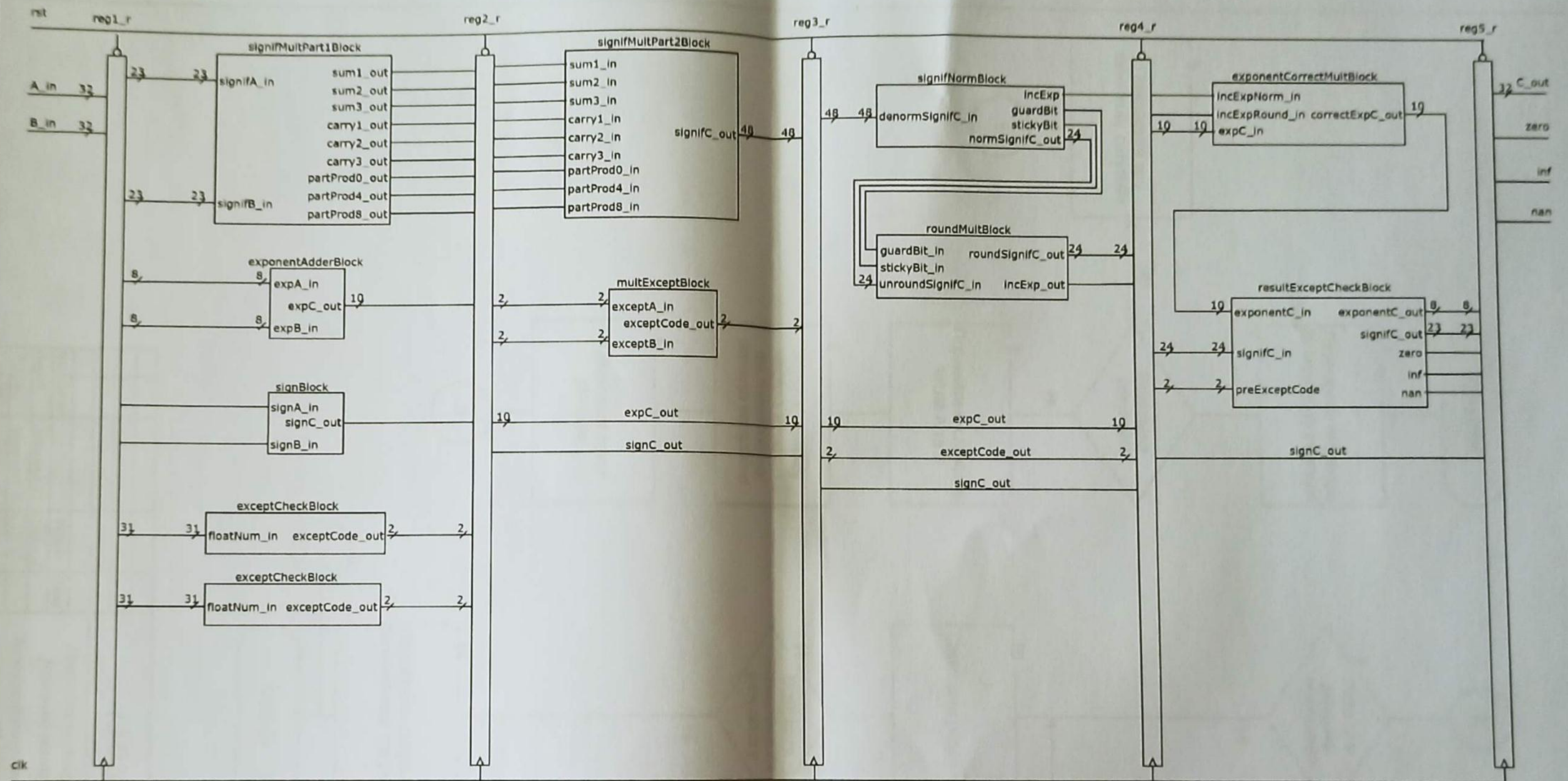
29. Mahant-Shetti, S. High performance low power array multiplier using temporal tiling / S.S. Mahant-Shetti, P.T. Balsara, C. Lemonds // IEEE Transactions on Very Large Scale Integration (VLSI) Systems. – 1999. – Vol. 7, Issue 1. – P.121–124.
30. Williams T. E. SRT Division Diagrams and Their Usage in Designing Custom Integrated Circuits for Division / Ted E. Williams, Mark Horowitzю – Stanford, CA: Stanford University, 1986. – 20 p.
31. Reaz, M. Pipeline Floating Point ALU Design using VHDL / M.B.I. Reaz, M.S. Islam, M.S. Sulaiman // IEEE International Conference on Semiconductor Electronics, Penang, 19-21 December 2002 : proceedings. – Los Alamitos: IEEE, 2003. – P.204–208.
32. Vladimirova, T. Floating-Point Mathematical Co-Processor for a Single-Chip On-Board Computer [Електронний ресурс] / T. Vladimirova, D. Eamey, S. Keller. – Режим доступу : [https://www.researchgate.net/publication/228761241_Floating - Point_Mathematical_Co-Processor_for_a_Single-Chip_On-Board_Computer](https://www.researchgate.net/publication/228761241_Floating_Point_Mathematical_Co-Processor_for_a_Single-Chip_On-Board_Computer).
33. Omkar, D. ASIC Implementation of 32 and 64 bit Floating Point ALU using Pipelining / D.R. Omkar, M. Aarthy // International Journal of Computer Applications. – 2014. – Vol. 94, Issue 17. – P.27–35.
34. Діденко А. Є. Конвеєрна модель і реалізація суматора дійсних чисел на FPGA / І. Я. Зеленьова, Т. В. Голуб, Т. С. Дьячук, А. Є. Діденко // Електротехнічні та комп'ютерні системи. – 2020. – №30 (109). С.21–31. ISSN 2221-3805.
35. Design Debugging Using the SignalTap II Logic Analyzer [Електронний ресурс].– Режим доступу: https://intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/qts/qts_qii53009.pdf.



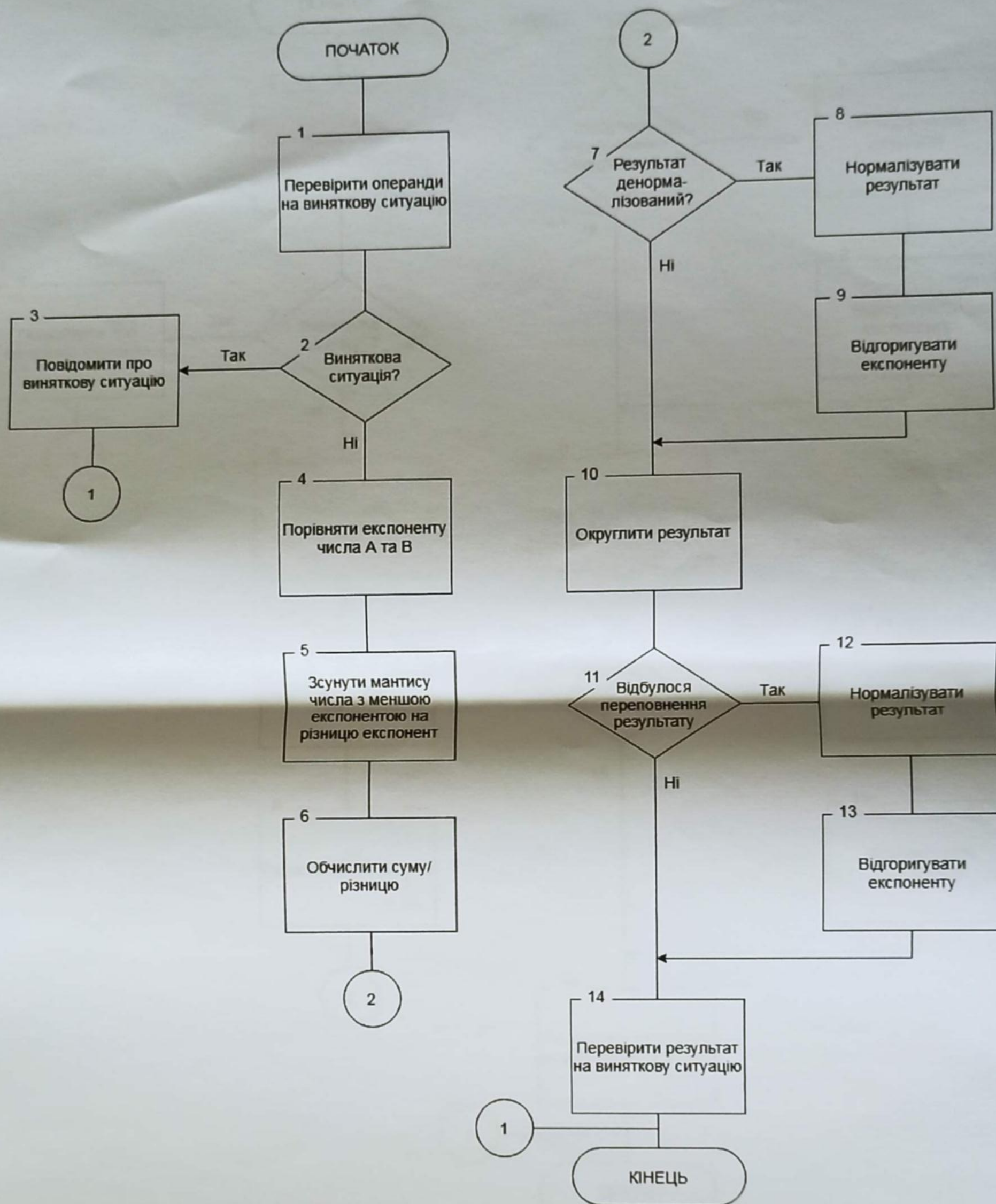
13.02070849.00029 ПЛ1				
Розробка модуля обчислень з рухомою комою на FPGA				
Схема модуля обчислень з рухомою комою				
Зам.	Лист	Підпис	Дата	
Розроб.	Діденко А. С.			
Перевір.	Зеленьова І. Я.			
Т. контр.				
Н. контр.	Щербак Н. В.			
Затверд.жс.	Кудерматов Р. К.			
				Лист 1
				Листів 1
НУ "Запорізька політехніка" КНТ-527				



13.02070849.00029 ПЛ2					
				Лист	Маса
Зам.	Лист	Підпис	Дата	Масштаб	
Розроб.	Діденко А. Є.				
Перевір.	Зеленьова І. Я.				
Т. контр.					
Н. контр.	Щербак Н. В.				
Затверд.	Кудерманов Р. К.				
Розробка модуля обчислень з рухомою комою на FPGA Схема блоку складання і віднімання				Лист 1	Листів 1
				НУ "Запорізька політехніка" КНТ-527	



13.02070849.00029 ПЛЗ					
Розробка модуля обчислень з рухомою комою на FPGA					
Схема блоку множення					
Зам.	Лист	Підпис	Дата	Лист	Маса
Розроб.	Діденко А. С.			Лист 1	Масштаб
Перевір.	Зеленьова І. Я.				
Т. контр.					
Н. контр.	Щербак Н. В.				
Інтерпрет.	Кудерметов Р. К.				
ІНУ "Запорізька політехніка" КНТ-527					



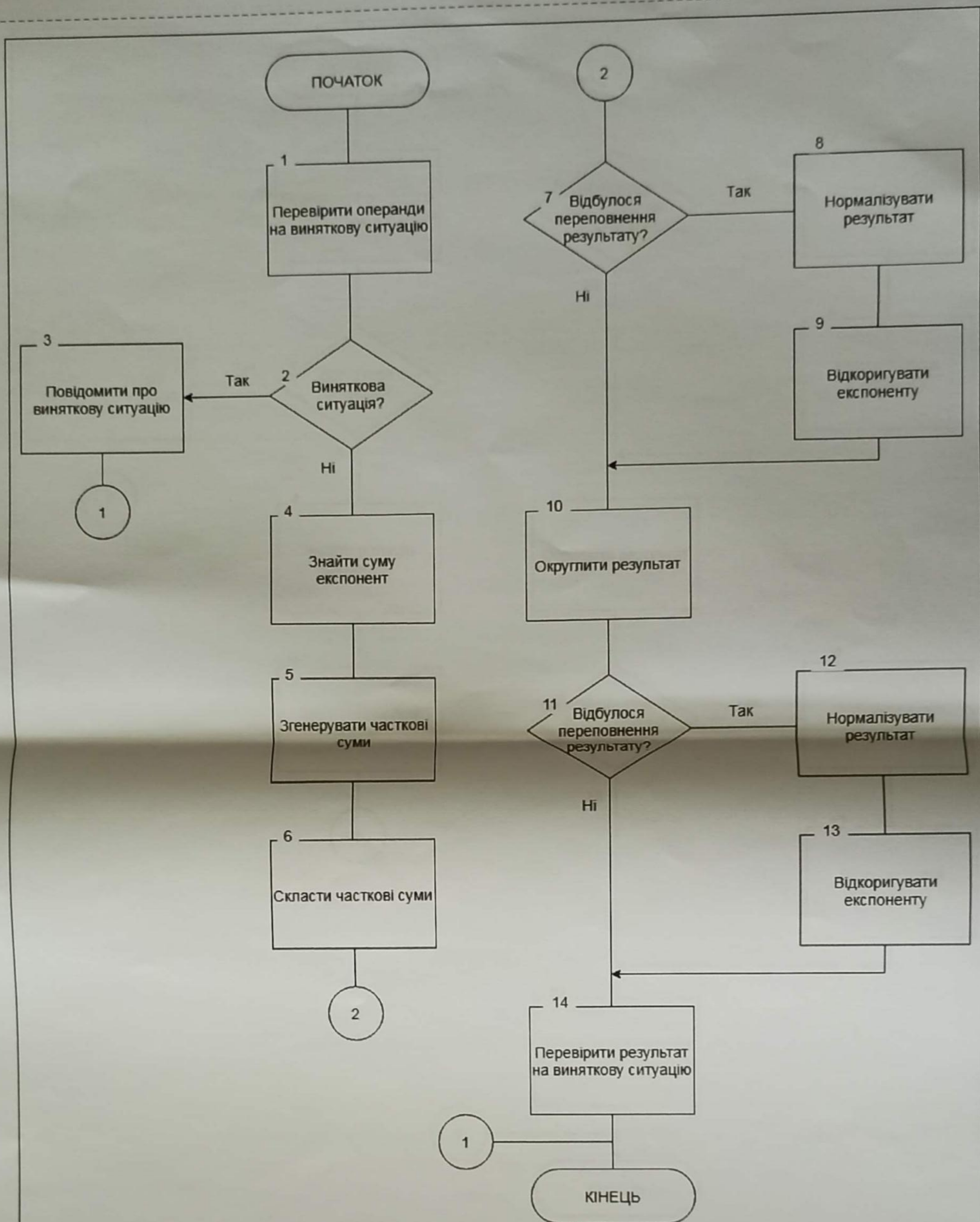
Зам.	Лист	Підпис	Дата
Розроб.	Діденко А. С.	<i>[Signature]</i>	
Перевір.	Зеленьова І. Я.	<i>[Signature]</i>	
Т. контр.			
Н. контр.	Щербак Н. В.	<i>[Signature]</i>	
Затвердж.	Кудерметов Р. К.	<i>[Signature]</i>	

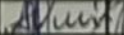
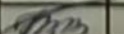
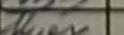
13.02070849.00029 A1

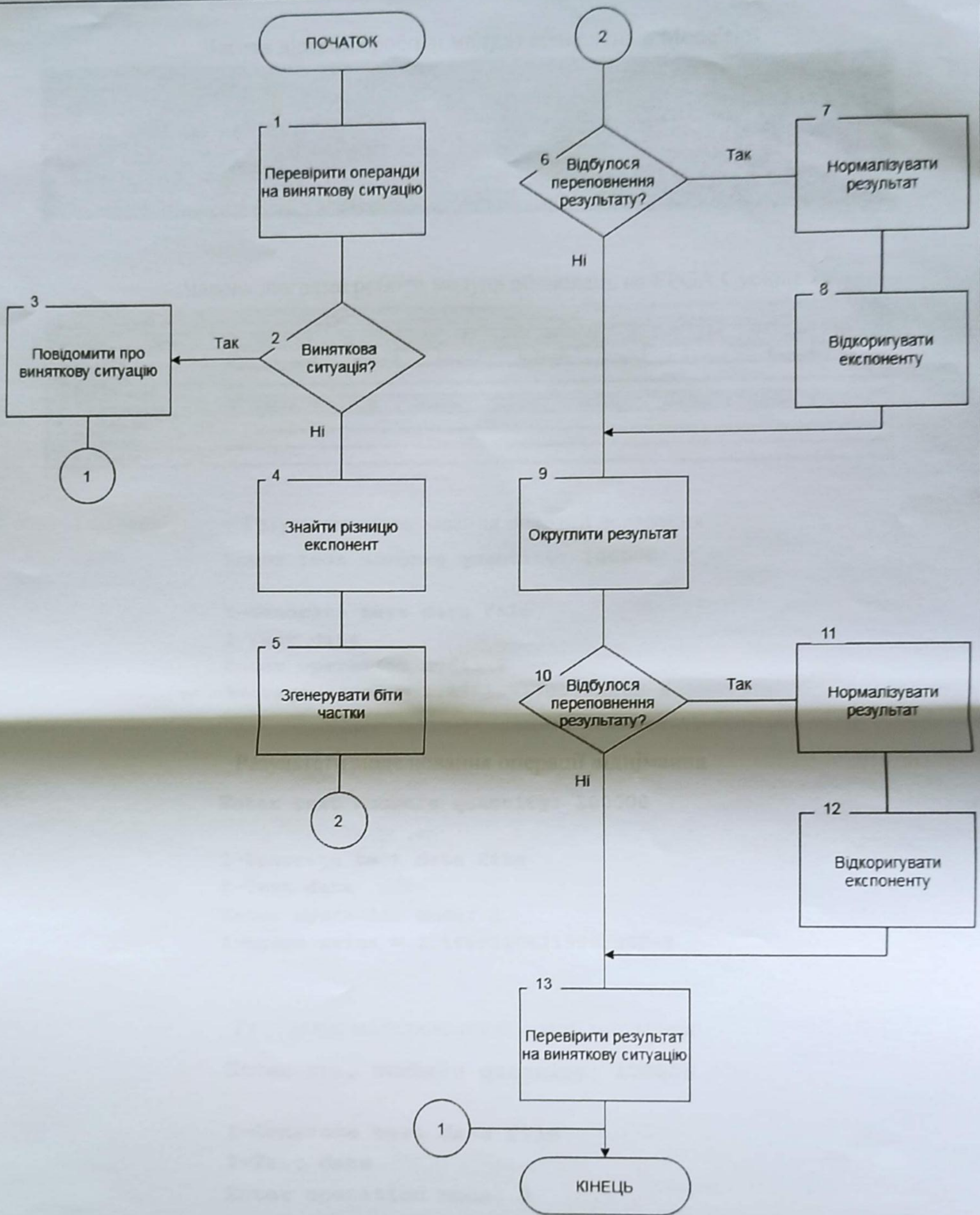
Розробка модуля обчислень з
рухомою камою на FPGA
Блок-схема алгоритму складання і
віднімання

Лист	Маса	Масштаб
Лист 1		Листів 1

НУ "Запорізька політехніка"
КНТ-527

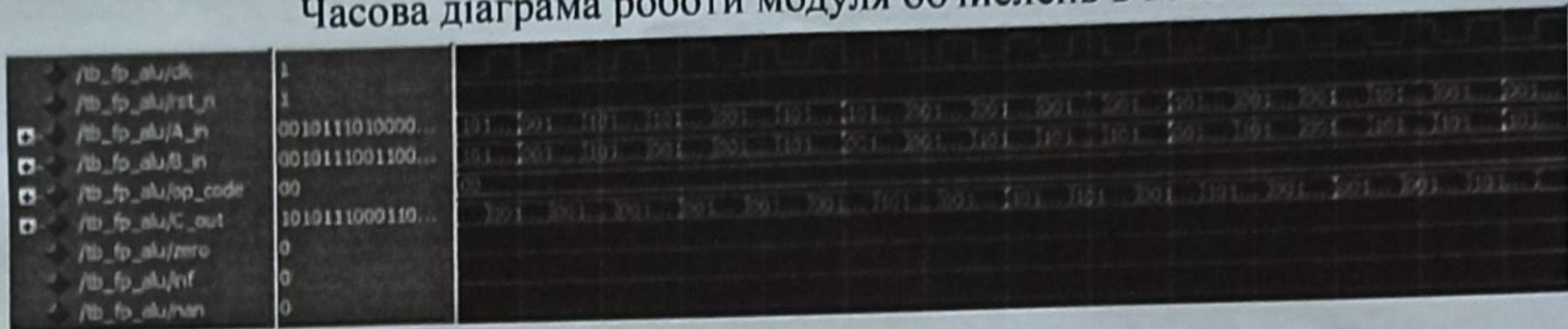


					13.02070849.00029 A1					
					Розробка модуля обчислень з рухомою комою на FPGA Блок-схема алгоритму множення			Лист	Маса	Масштаб
Зам.	Лист		Підпис	Дата						
Розроб.	Діденко А. С.									
Перевір.	Зеленьова І. Я.									
Т. контр.								Лист 1	Листів 1	
Н. контр.	Щербак Н. В.							ПУ "Запорізька політехніка"		
Затвердж.	Кудерметов Р. К.							КНТ-527		

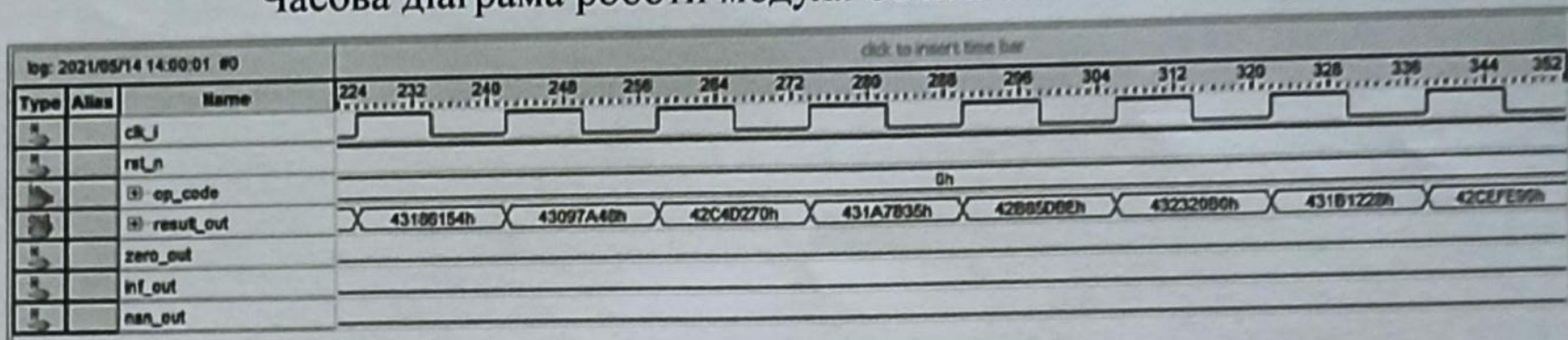


					13.02070849.00029 A1			
Зам.	Лист	Штук	Дати		Розробка модуля обчислень з рухомою комою на FPGA Блок-схема алгоритму ділення	Лист	Маса	Масштаб
Розроб.	Діденко А. С.							
Перевір.	Зеленьова І. Я.							
Т. контр.						Лист 1		Листів 1
Н. контр.	Щербак Н. В.					НУ "Київська політехніка" КНТ-527		
Затвердж.	Бударметов Р. К.							

Часова діаграма роботи модуля обчислень в Modelsim



Часова діаграма роботи модуля обчислень на FPGA Cyclone IV



Результати моделювання операції додавання

Enter test numbers quantity: 100000

1-Generate test data file

2-Test data

Enter operation mode: 2

Average error = 1.166337174903934E-5

Результати моделювання операції віднімання

Enter test numbers quantity: 100000

1-Generate test data file

2-Test data

Enter operation mode: 2

Average error = 1.1659266621993285E-5

Результати моделювання операції множення

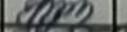
Enter test numbers quantity: 100000

1-Generate test data file

2-Test data

Enter operation mode: 2

Average error = 0.0

						13.02070849.00029 ПЛ4				
Зам.	Лист		Підпис	Дата		Розробка модуля обчислень з рухомою камою на FPGA		Лист.	Маса	Масштаб
Розроб.	Діденко А. С.					Блок-схема алгоритму складання і віднімання				
Перевір.	Зеленьова І. Я.									
Т. контр.								Лист 1	Листів 1	
Н. контр.	Шербак Н. В.							НУ "Запорізька політехніка"		
Затвердж.	Кудерметова Р. К.							КНТ-527		