НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ С. Г. СТІРЕНКО

«____» _____ 2020 р.

Дипломний проект

на здобуття ступеня бакалавра

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

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення»

Виконав:
студент IV курсу, групи ІО-64
Гаврилюк Олександр Володимирович _____

Керівник:
професор, доктор технічних наук
Жабін Валерій Іванович _____

Консультант з нормо контролю:
професор, доктор технічних наук
Симоненко Валерій Павлович _____

Рецензент:
професор, доктор технічних наук
Симоненко Валерій Павлович _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

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

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ С. Г. СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ

на бакалаврський дипломних проект студента

Гаврилюка Олександра Володимировича

1. Тема проекту (роботи) *Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення*

керівник проекту (роботи) *Жабін Валерій Іванович, доктор технічних наук, професор*, *затверджені наказом по університету від 7 травня 2020 року №1180-С*

1. Термін здачі студентом закінченого проекту (роботи)_____.
3. Вихідні дані до проекту (роботи) *технічна документація. теоретичні та статистичні дані,*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

*Огляд існуючих варіантів реалізації розробки, дослідження способів покращення даних прототипів, розробка пристрою для множення чисел з плаваючою комою*__

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема пристрою, блок-схема алгоритму обробки чисел, функціональна схема роботи системи

6. Консультанта проекту (робота), з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	*д.т.н., проф. Симоненко В. П.*		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітки
1.	*Затвердження теми роботи*	*01.09.2019*	
2.	*Вивчення та аналіз завдання*	*16.01.2020-18.02.2020*	
3.	*Розробка архітектури та загальної структури системи*	*18.02. 2020-04.04. 2020*	
4.	*Програмна реалізація системи*	*04.04. 2020-15.05. 2020*	
5.	*Оформлення пояснювальної записки*	*15.05. 2020-23.05. 2020*	
6.	*Захист програмного продукту*	*23.05. 2020*	
7.	*Передзахист*	*26.05. 2020*	
8.	*Захист*	*...*	

Студент-дипломник _____
(підпис)

Керівник роботи _____

(підпис)

АНОТАЦІЯ

Робота присвячена розробці пристрою для множення чисел з плаваючою комою, реалізованого на ПЛІС. Через зростання актуальності створення обчислювальних системи, головною структурною складовою яких є програмовані логічні інтегральні схеми – ПЛІС, виникає потреба в розробці методів прискорення виконання арифметичних операцій, що виконуються безпосередньо в цифрових обчислювальних машинах.

Запропонована структура пристрою дозволяє прискорити існуючі методи множення чисел з плаваючою комою в online режимі за допомогою використання надлишкової двійкової системи числення {-1, 0, 1} та порозрядного введення операндів, або так званої методики «цифра за цифрою» в обчислювальних машинах.

Для реалізації пристрою використовується мова опису апаратури VHDL та САПР Quartus II. Для візуалізації введення даних та демонстрації векторних сигналів під час множення двох чисел застосовується середовище опису і моделювання електронного обладнання ModelSIM.

ANNOTATION

The work is devoted to the development of a device for multiplying numbers with a moving comma, implemented on FPGA. Due to the growing relevance of creating computer systems, the main structural component of which are field programable gate array - FPGA, there is a need to develop methods to accelerate the execution of arithmetic operations performed directly by digital computers.

The proposed structure of the device allows to accelerate the existing methods of multiplication of floating point numbers by using redundant binary number system {-1, 0, 1} and bitwise input of operands, or the so-called method "digit by digit" in computers.

The device description language VHDL and CAD Quartus II is used to implement the device. ModelSIM electronic equipment description and modeling environment is used to visualize data entry and demonstrate vector signals when multiplying two numbers.

Опис альбому

до дипломного проекту
на тему: «Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення»

Київ – 2020 року

№ рядка	Формат	Позначення	Найменування	Кількість	Примітка
			Завдання на дипломний проект	2	
1	А4	ІАЛЦ.467100.001 ОА	Опис альбому	1	
2	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	4	
3	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	56	
4	А3	ІАЛЦ.467100.004 Д1	Структурна схема пристрою	1	
5	А3	ІАЛЦ.467100.005 Д2	Блок-схема алгоритму	1	
			обробки операндів		
6	А3	ІАЛЦ.467100.006 Д3	Функціональна схема	1	
			пристрою		
7	А4	ІАЛЦ.467100.007 Д4	Лістинінг програми	13	

Змн.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467100.001 ОА		
Розроб.		Гаврилюк О.В.			*Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення*	Літ.	Арк. 1 / Аркушів 1
Перев.		Жабін В.І.					
Н. Контр.		Симоненко В.П.			*Опис альбому*	*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІО-64*	
Затверд.		Жабін В.І.					

Технічне завдання
до дипломного проекту
на тему: «Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення»

Київ – 2020 року

Технічне завдання

Зміст

1. Тема і область застосування

Дана розробка представляє собою створення пристрою для множення чисел з плаваючою комою, який є альтернативою вже існуючим своїм варіантам і може бути використаний в комп'ютерних системах, які потребують швидких та точних обчислень.

2. Підстави для розробки

Підставою для розробки даного пристрою є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського», а також власна зацікавленість в даній темі.

3. Ціль та призначення розробки

Ціль даного проекту є розробка пристрою на ПЛІС для множення чисел з плаваючою комою в online режимі.

4. Джерела розробки

Джерелами розробки є патенти, науково-технічна література по архітектурі комп'ютера та комп'ютерним системам, публікації в періодичних виданнях, довідники з операціями, які виконуються в двійковій системі числення, публікації в інтернет-джерелах по даній темі.

5. Технічні вимоги

5.1. Вимоги до розробки

- Правильність виконання операції множення чисел з плаваючою комою
- Швидкодія та дешевизна створення
- При потребі може інтегруватись в різні пристрої

5.2. Вимоги до програмного забезпечення

- Операційна система Windows 7 / Windows 8 / Windows 8.1 / Windows 10
- Altera Quartus Prime Lite Edition 19.1
- Mentor Graphics ModelSIM Intel FPGA Edition 10.b

5.3. Вимоги до апаратної частини

- Процесор Intel Pentium 3 і кращі
- Оперативна пам'ять не менше 1Гб
- Вільне місце на диску не менше 10Гб

6. Етапи розробки

	Дата
Вивчення літератури	16.01.2020
Створення і узгодження технічного завдання	05.02.2020
Створення алгоритму операції, що виконується на пристрої	18.02.2020
Створення структурної схеми пристрою	25.02.2020
Розробка мікропрограми та реалізація на ПЛІС	04.04.2020
Тестування та виправлення помилок	15.05.2020
Оформлення документації дипломної роботи	27.05.2020

Пояснювальна записка
до дипломного проекту
на тему: «Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення»

Київ – 2020 року

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

БОМ	Блок Обробки Мантис
БОП	Блок Обробки Порядків
ГПД	Граф Потоку Даних
ЕОМ	Електронна Обчислювальна Машина
ОЗП	Оперативний Запам’ятовуючий Пристрій
ОМ	Обчислювальний Модуль
ПЛІС	Програмована Логічна Інтегральна Схема
ПСБЗ	Паралельна Система з Безпосередніми Зв’язками
САПР	Система Автоматичного ПРоектування
CPLD	(Complex Programmable Logic Device) Складні програмовані логічні пристрої
FPGA	(Field Programmable Gate Array) Програмована користувачем вентильна матриця
HDL	(Hardware Description Language) Мова опису апаратури
LUT	(LookUp Table) Таблиця пошуку
PLB	(Programmable Logic Block) Програмований логічний блок
VHDL	(Very high speed integrated circuits Hardware Description Language) Мова опису апаратури швидкісних інтегральних схем

ВСТУП

Основними технічними вимогами до обчислювальних та інформаційних систем є швидкодія, надійність та ціна. Важливість цих вимог зростає для сучасної елементної бази, а саме програмованих логічних інтегральних схем (ПЛІС). Швидкодія пристроїв на їх основі експоненційно зростає і як наслідок, майже за 30 років прогресу, перетворились з чогось неймовірного і нереального (кімнати величезного розміру з електронними обчислювальними машинами) в абсолютно звичайне і буденне (обчислювачі та пристрої збереження даних, багато тисяч яких можуть з легкістю поміститись в ПЛІС).

Надійність систем на ПЛІС залежить від складності системи внутрішньої комутації та кількості зовнішніх виходів.

Також, немало важливу роль відіграє і дешевизна – варто тільки вдуматись в кількість техніки та транспорту, які використовують комп’ютерні технології. Серійне створення таких технологій спрямоване на максимально зручне використання їх користувачем, а до їх складу мають входити пристрої, які поєднують швидкодію, оптимізацію своєї роботи, безпеку використання та дешевизну виготовлення (в більшості випадків – малі розміри). Як приклад – саме така концепція і стала основою в розробках компанії Space Exploration Technologies Corporation (SpaceX), що дало їй змогу здешевити вивід ракет на орбіту більш ніж на 50%. Саме тому, потреба у використанні пристроїв множення в теперішньому житті невпинно зростає.

Актуальність роботи полягає у необхідності постійно покращувати характеристики систем, побудованих на основі ПЛІС, таких як швидкодія, надійність, апаратурні витрати.

Мета роботи – створення пристрою множення чисел з плаваючою комою, який буде мати переваги в швидкодії, надійності та дешевизні складових з уже існуючими його варіантами та зможе використовуватись в

будь-яких обчислювальних технологіях, які потребують швидкого та точного обчислення даних.

Під час підготовки до даної роботи було поставлено наступні задачі:

1. Розглянути та вивчити літературу по даній темі
2. Проаналізувати існуючі варіанти рішення проблеми множення чисел з плаваючою комою.
3. Обрати найкращий варіант пристрою множення.
4. Розробити власний пристрій на основі обраного з розглядом всіх переваг та недоліків.
5. Обрати мову опису апаратури інтегральних схем
6. Обрати середовище опису на ПЛІС апаратного забезпечення, видимого редагування логічних схем і моделювання векторних сигналів.

Розробка пристрою множення чисел з плаваючою комою включає:

1. Створення алгоритму множення чисел з плаваючою комою
2. Створення структурної схеми пристрою
3. Розробка мікропрограми за допомогою однієї з мов опису апаратури інтегральних схем
4. Реалізація даної програми на ПЛІС
5. Моделювання векторних сигналів, тестуванням швидкодії і перевірка правильності множення

РОЗДІЛ 1.
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ПРОБЛЕМИ МНОЖЕННЯ ЧИСЕЛ З ПЛАВАЮЧОЮ КОМОЮ

Сучасний етап розвитку комп'ютерних технологій включає в себе одну з головних складових, а саме процес обробки інформації, що веде за собою потік виконання певної кількості різних арифметико-логічних операцій над числами різного типу. Аналіз цього процесу дав можливість на створення пристроїв, які безпосередньо виконують такі операції та можуть, в залежності від своєї структури, прискорювати обчислення або надавати результату будь-яку потрібну точність. Але, в міру складності такого пристрою, виникають певні проблеми такі як громіздкість його елементів та самого пристрою в цілому, ресурсозалежність (як на структурному так і на програмному рівнях), накопичення помилок під час обрахунків через використання чисел з великим діапазоном значень, низька швидкість виконання операцій для отримання бажаної точності обчислень тощо. Саме тому виникає проблема створення такого пристрою, який буде мати мінімум недоліків і буде вигравати в швидкодії та дешевизні складових з уже існуючими його варіантами, а також зможе використовуватись в будь-яких обчислювальних технологіях, які потребують безпомилкового обчислення даних.

Таким чином, дослідження та розробка пристроїв для виконання арифметичних операцій в комп'ютерних системах, являють собою надзвичайно важливу і актуальну задачу поставлену перед розробниками для покращення та розвитку теперішній технологій.

В даній роботі розроблено пристрій для множення чисел з плаваючою комою, оскільки на теперішній час росте потреба у використанні технологій розрахунків над числами, що подані в формі з плаваючою комою.

1.1. Пристрій для множення методом аналізу двох бітів множника

Огляд відомих технічних рішень будемо проводити з оцінкою їх швидкодії, надійності та витрат апаратури.

Відомий пристрій для множення реалізований за допомогою методу аналізу двох бітів множника[1].

До складу даного пристрою входять регістр першого множника (1), мультиплексор першого множника (2), мультиплексор другого множника (3), регістр другого множника (4), блок аналізу (5), блок зсуву (6), суматор (7), регістр зсуву результату (8), блок управління (9), регістри результату основного і допоміжного (10), шина мантиси (11).

Пристрій працює наступним чином: перший множник надходить на перший вхід мультиплексора першого множника з шини мантиси. Далі з виходу мультиплексора множник надходить на вхід регістра першого множника, вихід якого з'єднаний зі входом блоку аналізу і блоку зсуву. Блок аналізу видає код кількості зсувів першого множника на необхідну кількість розрядів згідно табл. 1.1. Якщо в першому розряді множника нуль, то проводиться аналіз кількості поспіль записаних нулів, якщо одиниця, то проводиться аналіз кількості поспіль поданих одиниць. Блоком зсуву перший множник зсувається вправо на необхідну кількість розрядів, згідно з отриманим кодом з блоку аналізу і надходить на другий вхід мультиплексора першого множника. Другий множник надходить на вхід регістра другого множника з шини мантиси, з виходу якого надходить на правий вхід суматора. На рис. 1.1. подано структурну схему даного пристрою.

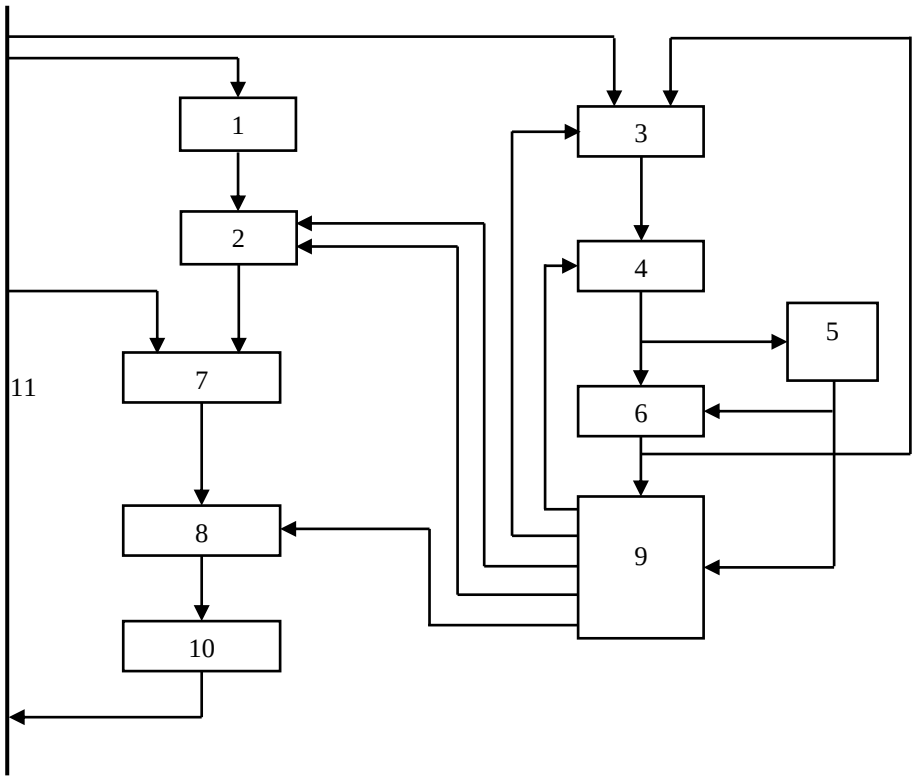


Рис. 1.1. Структурна схема пристрою множення чисел за допомогою зсувів

В основу функціонування пристрою закладено алгоритм множення заснований на аналізі всього множника. Суть алгоритму полягає в наступному: підраховується кількість поспіль записаних нулів або одиниць починаючи з молодшого розряду. На початку виконання операції початковий стан дорівнює нулю. Потім він міняється в залежності від кількості поспіль нулів або одиниць і початкового стану згідно табл. 1.1.

Перевагами даного пристрою є швидкодія за рахунок використання алгоритму аналізу всього множника, а також отримання на виході відразу нормалізованого результату.

Головний недолік це переповнення, яке може виникнути під час множення чисел зі знаками. Також до недоліків можна віднести нормалізацію першого множника поза пристроєм та потребу в використанні складного блоку управління, який сильно зменшує дешевизну виготовлення даного прототипу.

Таблиця 1.1

Таблиця зсувів блоку аналізу першого операнду

№	Кількість розрядів	Вміст розрядів	Попередній стан	Новий стан	Операції, що виконуються
1	n=1	0	0	0	Зсув першого множника на два розряди +2M, зсув результату на два розряди
2	n=1	0	1	1	Зсув першого множника на два розряди -M, зсув результату на два розряди
3	n=1	1	0	0	Зсув першого множника на два розряди +M, зсув результату на два розряди
4	n=1	1	1	0	Зсув першого множника на два розряди +2M, зсув результату на два розряди
5	n>1	0	0	0	Зсув першого множника на n розрядів +, зсув результату на n розрядів
6	n>1	0	1	0	Зсув першого множника на n розрядів +M, зсув результату на n розрядів
7	n>1	1	0	1	Зсув першого множника на n розрядів -M, зсув результату на n розрядів
8	n>1	1	1	1	Зсув першого множника на n розрядів +, зсув результату на n розрядів

1.2. Пристрій для множення методом порозрядного введення операндів

Існує пристрій для множення чисел з фіксованою комою, реалізований за допомогою методу порозрядного введення операндів[2].

Даний пристрій містить регістр першого множника (1), регістр другого множника (2), регістр результату (3), суматор результату (4), суматор множників (5), блок аналізу розрядів (6), регістр зсуву (7), елементи І (8-11) та АБО (12-15), входи пристрою (16-22). Такий пристрій дозволяє поєднувати в часі порозрядне введення операндів і їх обробку, тобто отримувати результати множення на виході пристрою в міру надходження операндів старшими розрядами спочатку на його входи. Структурну схему пристрою множення з фіксованою комою подано на рис. 1.2.

Робота пристрою полягає в наступному. У початковому стані в регістрах 1, 2 та 3 записані нулі, а в регістрі зсуву одиниця записана в крайньому лівому розряді. Будемо вважати, що до початку кожного і-го циклу (i = 1, 2, ... n) на вхідних кодових шинах з'являються чергові і-ті розряди множників, починаючи зі старших розрядів. Крім того, в кожному циклі на вхідних шинах 17, 16 і 18 з'являються послідовно сигнали відповідно Т1, Т2 і Т3. Множники подаються на вхідних шинах в надлишковій формі з цифрами «1», «-1» і «0». При цьому «1» в черговому розряді співмножника кодується одиничним сигналом на вхідних шинах 20 або 21, «-1» - одиничним сигналом на шинах 19 і 22, а «0» кодується відсутністю сигналів на обох шинах, що представляють даний співмножник. Добуток також формується в надлишковій формі послідовно зі старших розрядів. Сигнал Т, по шині 16 надходить до ланцюга видачі і прийому коду регістра другого , а через елемент 15 - до ланцюга прийому коду регістра формування результату, а також на керуючі входи елементів І 10 і 11. Крім того, одиничний сигнал з виходу елемента І 8 через елемент АБО 13 поступає до ланцюга подання оберненого коду регістра зсуву 7 і до входу ланцюга

переносу молодшого розряду суматора співмножників для перетворення, що надходить з регістра зсуву 7, зворотного коду в доповняльний, який підсумовується в суматорі з кодом з регістра першого множника, причому результат підсумовування запам'ятовується в регістрі 2. Цим забезпечується підсумовування «-1» до і-го розряду коду множника з регістра 2. При відсутності одиничних сигналів ва обох вхідних кодових шинах множника вміст регістрів 3 і 2 в цьому такті не змінюватиметься. Після закінчення дії сигналу Т2 на шині 18 з'являється керуючий сигнал Т3, який подається до ланцюга зсуву регістрів 7 і 3 і на керуючий вхід блоку аналізу розрядів. Код регістра 7 зсувається на один розряд вправо, а код регістра 3 - на один розряд вліво, причому при наявності одиничного сигналу між двома старшими розрядами регістра 3 передача цифри при зсуві здійснюється з інвертуванням, а при відсутності одиничного сигналу на цій шині - без інвертування. Блок аналізу розрядів здійснює формування цифр добутку, а також формування сигналу корекції. Цей блок аналізує три старших розряди регістра 3 (два знакових розряди і перший розряд після коми) .

Таким чином, введення елементів і нових конструктивних зв'язків дає можливість обробляти операнди в пристрої, починаючи із старших розрядів, з одночасним формуванням розрядів результату, що значно збільшує швидкодію пристрою в цілому. Однак цей пристрій не дозволяє виконувати множення чисел, представлених у формі з плаваючою комою, що є великим недоліком пристрою, оскільки форма представлення чисел з плаваючою комою значно розширює діапазон представлення чисел в машині і дозволяє автоматизувати процес стеження за положенням коми в числі. Неточності, які виникають під час множення з фіксованою комою можуть бути незначними, але в деяких випадках під час накопичення результату можуть нести згубний характер. Хоча, така структура пристрою для множення чисел з фіксованою комою значно виграє в ресурсозалежності, ніж її аналоги для плаваючої коми.

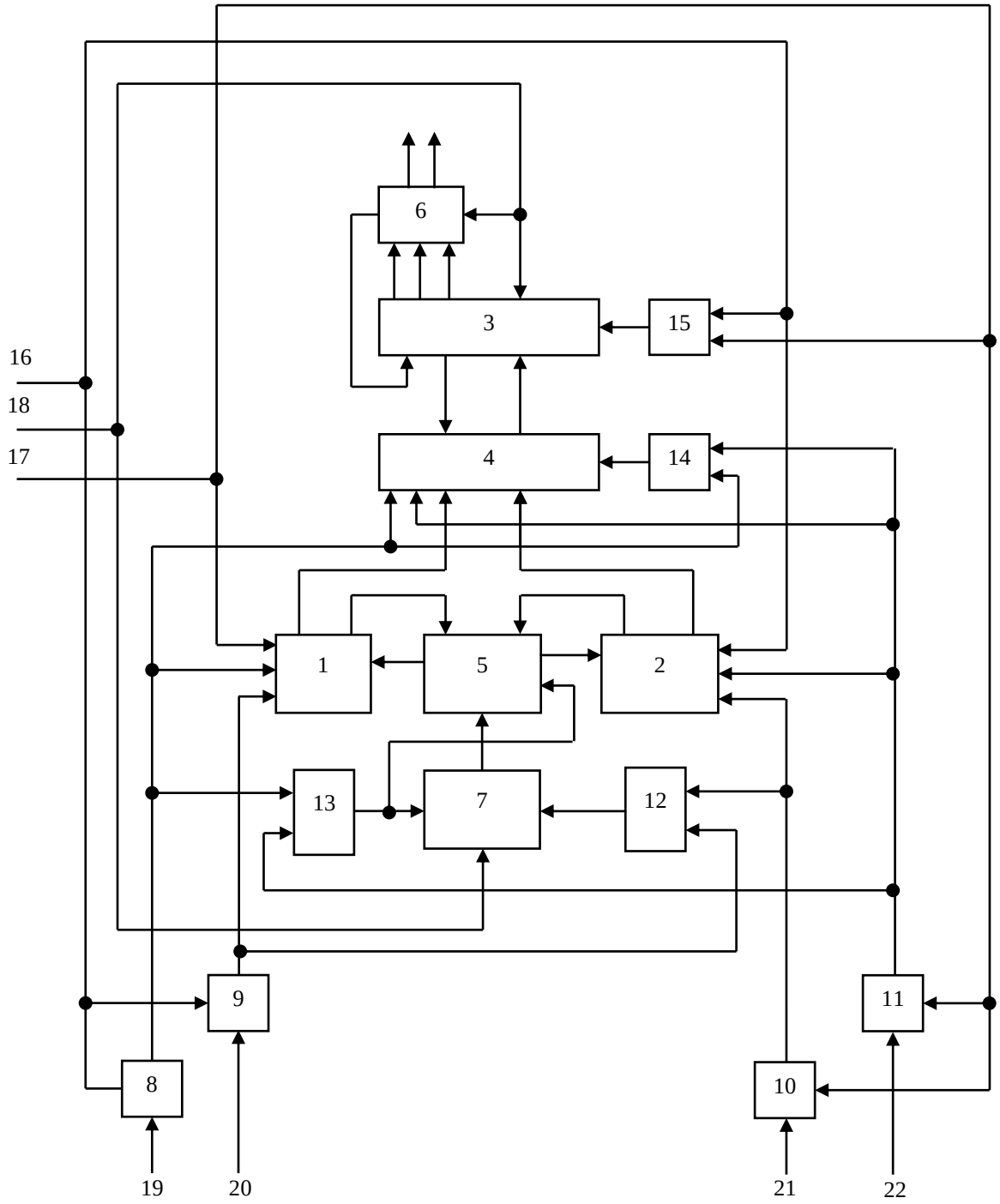


Рис. 1.2. Структурна схема множення чисел з фіксованою комою

1.3. Система обробки інформації для множення чисел з плаваючою комою

Якщо потрібно перемножити два числа з плаваючою комою, в більшості випадків для отримання добутку потрібно виконувати нормалізацію. Зазвичай це передбачає трудомісткий процес зсуву на ранніх етапах множення, таких як додавання і віднімання, і які ведуть за собою створення додаткової схеми обробки даних, що неабияк підвищує складність кінцевого пристрою. Відома система обробки інформації для множення чисел з плаваючою комою, яка безпосередньо включає блок нормалізації результату. [3]

Розглянута система обробки інформації виконує множення чисел з плаваючою комою та передбачає вдосконалену методику нормалізації добутку, ніж її прототипи. Вона містить пристрій обчислення мантиси добутку, до складу якого входить блок множення мантис двох чисел і пристрій зберігання добутку. Крім того, система включає пристрій підсумовування порядків операндів і пристрій, який аналізує мантису добутку. Структурну схему такої системи приведено на рис. 1.3.

Перевагами такої системи є рішення проблеми переповнення під час множення чисел з плаваючою комою та отримання нормалізованого добутку, навіть якщо числа були подані в довільній формі з плаваючою комою. Однак ця система не дозволяє поєднувати в часі процес порозрядного введення операндів, що формуються поза системою, і їх обробку, що веде за собою низьку швидкодію пристрою, оскільки протягом порозрядного надходження мантис операндів на входи такої системи вона буде простоювати і тільки після закінчення цього процесу вона починає виконувати множення введених операндів.

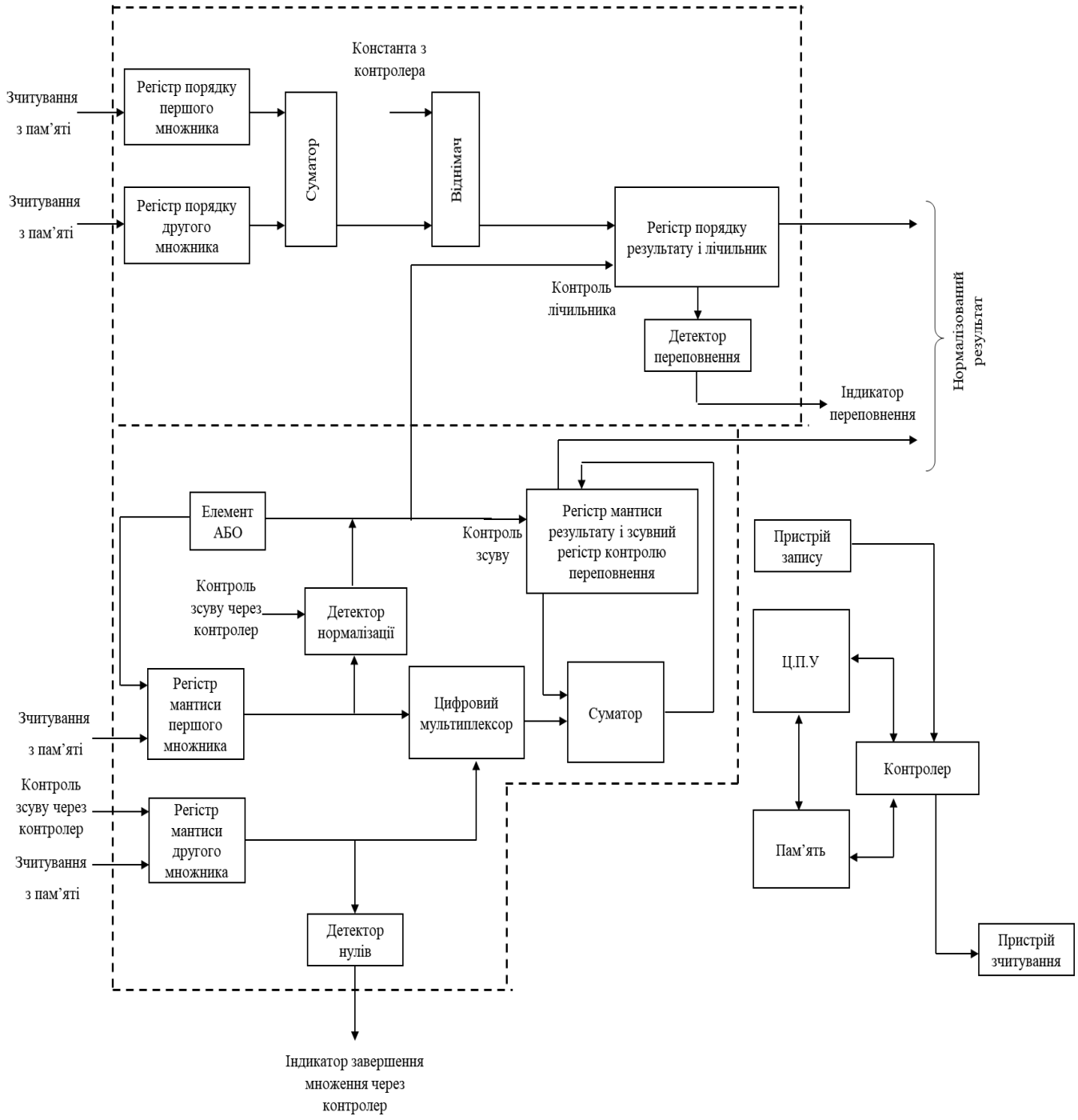


Рис. 1.3. Структурна схема пристрою множення чисел з плаваючою комою

ВИСНОВКИ ДО РОЗДІЛУ 1

В результаті огляду поточного розділу бакалаврської роботи, спрямованого на вивчення існуючих прототипів пристрою для множення чисел можна зробити наступні висновки.

У ЕОМ, які включають множення чисел з фіксованою комою, попри точність обчислень, існує метод порозрядного введення операндів, які подаються на пристрій в надлишковій системі числення, та обчислення добутку, що неабияк прискорює операцію множення.

В свою чергу пристрої, які використовують подання чисел у формі із плаваючою комою, виконують арифметичні операції і над мантисами чисел, і над їхніми порядками, що часто потребує виконання операції нормалізації результату, яка ускладнює та сповільнює систему загалом. Тому машини із плаваючою комою є більш складними в побудові і поступаються у швидкодії машинам на яких реалізовано операції з фіксованою комою.

Головними перевагами розглянутих в цьому розділі пристроїв стали:

1. Швидкодія, яка досягається за рахунок використання online режиму, а саме суміщення в часі введення обробки і виводу розрядів, що дозволяє виконувати операції паралельно на рівні обробки розрядів;
2. Надійність, яка заключається в порозрядній передачі даних для спрощення комутацій всередині ПЛІС і скорочення числа пінів;
3. Точність – використання чисел з рухомою комою, що збільшує діапазон допустимих значень як операндів, так і результату під час операцій.

Поєднання online режиму обробки операндів у надлишковій системі числення з методами множення чисел у формі з плаваючою комою та подальшою нормалізацією результату без суттєвих ресурсовитрат та втрати швидкодії формує мету даної роботи, а саме створення такого пристрою, який би поєднав в собі ці переваги.

РОЗДІЛ 2.
ОСОБЛИВОСТІ МНОЖЕННЯ ДВІЙКОВИХ ЧИСЕЛ ТА АНАЛІЗ ПЕРЕВАГ ВИКОРИСТАННЯ ONLINE РЕЖИМУ ОБЧИСЛЕНЬ

З плином час, все більш вимогливими комп’ютерні системи стають до ефективності виконання арифметичних операцій, зокрема множення, через швидке збільшення кількості одночасно вхідних та оброблюваних даних. Розглянуті в першому розділі приклади прототипів пристроїв рішення проблеми множення чисел було реалізовано за одним з можливих методів множення двох двійкових чисел, які безпосередньо подаються на пристрій і мають довільну форму. Суттєвим недоліком більшості пристроїв, які давали результат у вигляді добутку з фіксованою комою стало використання одного з базових методів множення чисел в прямому коді, оскільки таке рішення призводило до зменшення діапазону отриманого числа. Не менш важливим недоліком є і те, що задля покращення швидкодії пристрою було використано громіздкі структури та доводилось вводити масштабні коефіцієнти для того, щоб уникнути переповнення розрядної сітки під час виконання арифметичних операцій. При цьому від правильності їх вибору залежала і поява помилок під час подання.

В свою чергу цифрові обчислювальні системи з плаваючою комою конструктивно більш складні, оскільки необхідно додавати додаткове обладнання для виконання арифметичних операцій над порядками чисел, а також вводити операції нормалізації і вирівнювання порядків чисел. Також, час, за який виконуються операцій над числами з плаваючою комою більший, ніж в аналогічних пристроях з фіксованою комою, що зумовлено необхідністю роботи з порядками.

Виходячи з цього, росте потреба в створенні пристроїв, які б використовували метод множення, що поєднує в собі переваги уже відомих

методів як в структурі створення так і в реалізації, що б підвищило ефективність та прискорило робото ЕОМ в цілому.

2.1. Множення двійкових чисел у прямому коді

Існує 4 способи множення двійкових чисел[5]. Нехай Y – множене, а X - множник і це правильні двійкові дроби, при чому

$$Y = 0, y_1, y_2, \ldots, y_n, X = 0, x_1, x_2, \ldots, x_n, \text{ де } y_n, x_n \in \{0,1\}. \quad (2.1)$$

Тоді добуток Z від множення величин чисел Y та X дорівнює

$$Z = YX = Yx_1 2^{-1} + Yx_2 2^{-2} + \ldots + Yx_i 2^{-i} + \ldots + Yx_n 2^{-n}. \quad (2.2)$$

Добуток являє собою суму часткових добутків повного першого множника на і-ий елемент другого множника. Для формування і накопичення такого добутку можна використовувати комбінаційний суматор і регістр добутку, або їх композицію – накопичувальний суматор. Для множення кожним із способів використано умовні позначення: SM – суматор, RG1- регістр добутку, RG2 – регістр першого множника, RG3 – регістр другого множника, СТ – лічильник, q – розрядність лічильника.

- Перший спосіб множення

Під час використання першого способу множення в кожному першому такті і-го циклу проводиться аналіз значення RG2(n) – молодшого n-го розряду регістру першого множника. Вміст регістру другого множника додається до суми часткових добутків, що знаходяться в регістрі добутку (RG2(n) = 1), або не додається (RG2(n) = 0). В другому такті здійснюється зсув вправо у регістрах добутку і першого множника, що еквівалентно множенню їх на 2^{-1}. Після зсуву цифра молодшого розряду добутку переходить у старший розряд регістру першого множника. Після n циклів молодші розряди 2n-розрядного добутку будуть записані в регістр першого множника, а старші - у регістр добутку. Операційну схему даного способу показано на рис. 2.1.

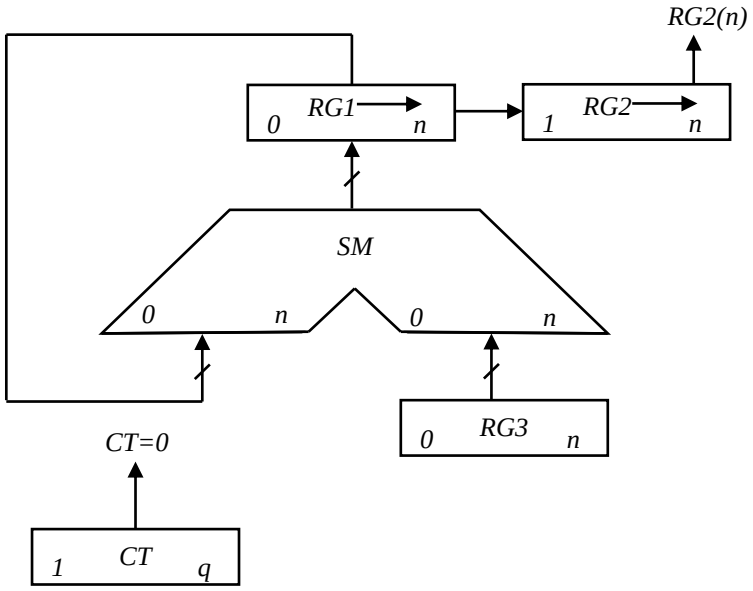


Рис.2.1. Операційна схема пристрою множення чисел першим способом

- Другий спосіб множення

Перед початком множення другим способом X записують в регістр першого множника, а множене Y – в молодші розряди регістру другого множника. В кожному і-му циклі множення додаванням кодів на регістрах добутку і другого множника управляє цифра RG2[1], і в регістрі другого множника здійснюється зсув вліво на один розряд, в результаті чого виконується $Y_i = 2Y_{i-1}$. Через те, що сума часткових добутків в процесі множення є нерухомою, зсув в регістрі другого множника можна виконувати суміщенням в часі з підсумовуванням. Завершення операції множення буде визначено через нульовий вміст регістру першого множника, що також веде за собою збільшення швидкодії, якщо другий множник ненормалізований. Операційну схему даного способу показано на Рис. 2.2.

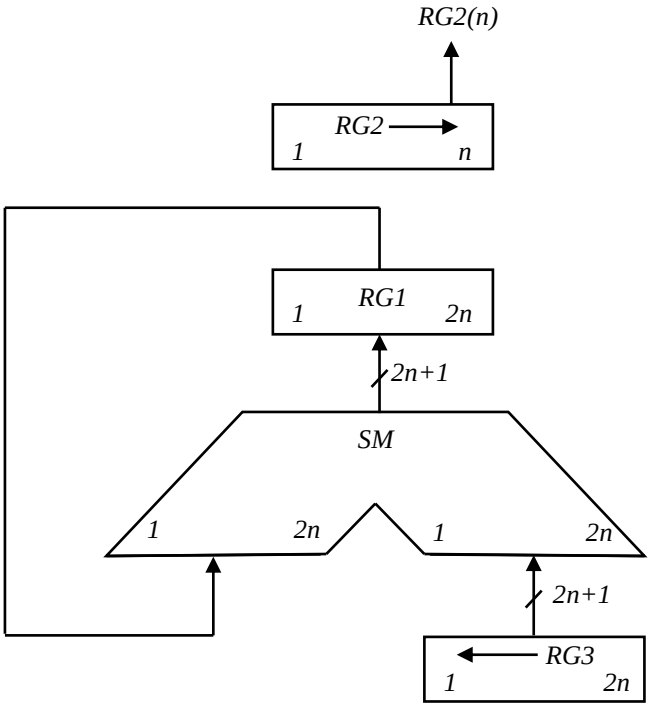


Рис.2.2. Операційна схема пристрою множення чисел другим способом

- Третій спосіб множення

При множенні третім способом X записується в старші розряди регістру першого множника (RG2[1]=0). Вага молодшого розряду регістра другого множника дорівнює 2^{-2n} , тому отриманий код в цьому регістрі представлений значенням $Y2^{-n}$. В кожному циклі множення підсумовування операндів здійснюється якщо RG2[n+1]=1. В регістрах добутку і першого множника виконується зсув вліво. В результаті підсумовування вмісту регістра другого множника і регістра добутку може виникнути перенос в молодший розряд регістру першого множника, що реалізується за допомогою суматора. Збільшення довжини регістру першого множника на один розряд усуває можливість поширення переносу в розряди другого множника. Після виконання n циклів молодші розряди добутку знаходитимуться в регістрі добутку, а старші – в регістрі першого множника. Операційну схему даного способу показано на Рис. 2.3.

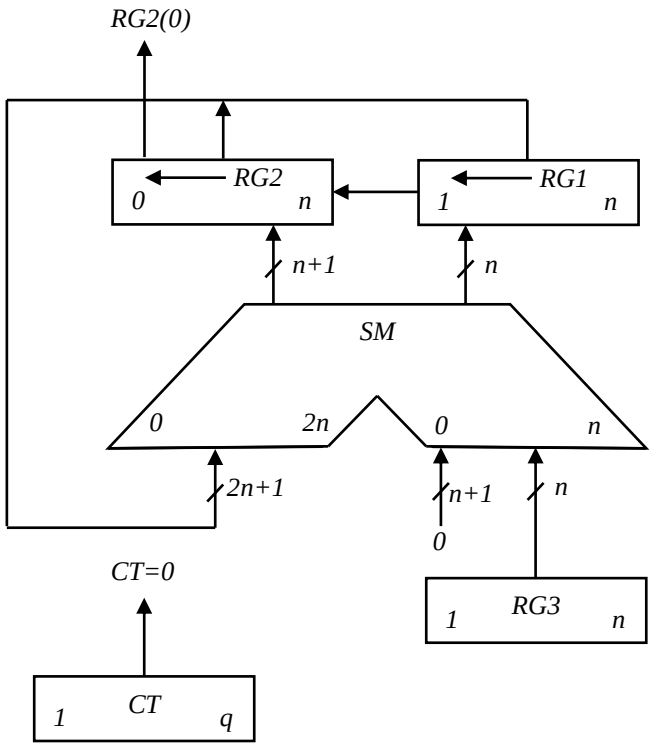


Рис.2.3. Операційна схема пристрою множення чисел третім способом

- Четвертий спосіб множення

Перед множенням четвертим способом перший множник записують в регістр першого множника, а другий – в старші розряди регістру другого множника, тобто в регістрі другого множника встановлюють Y_0=Y2–1. В кожнім циклі цифра RG2[n+1], що знаходиться в старшому розряді регістра першого множника, керує підсумовуванням, а в регістрі другого множника здійснюється зсув вправо на один розряд, що еквівалентно множенню всього вмісту цього регістра на 2^{-1} . Операційну схему даного способу показано на Рис. 2.4.

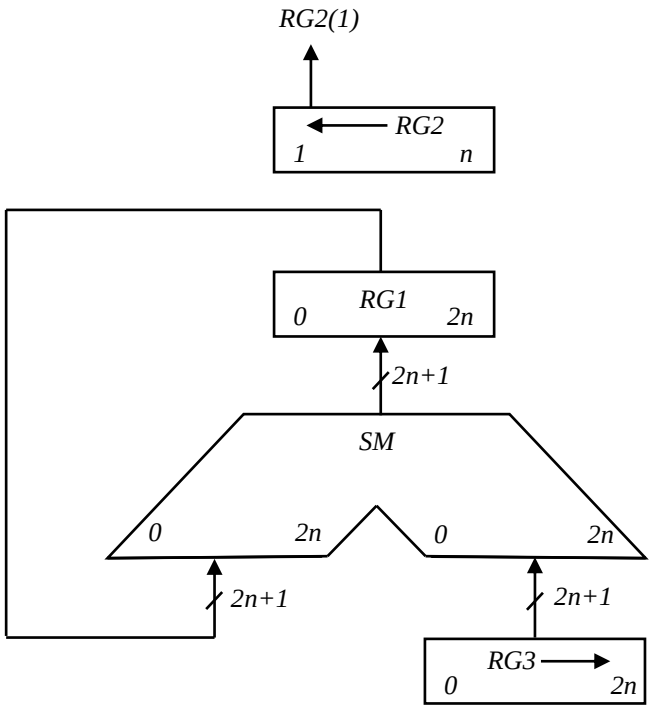


Рис.2.4. Операційна схема пристрою множення чисел четвертим способом

2.2. Множення двійкових чисел у формі з плаваючою комою

Для чисел, представлених у формі з плаваючою комою, загальним правилом є запис у вигляді мантиси і порядку. Множення двох чисел $X = 2^{P_X} M_X$ і $Y = 2^{P_Y} M_Y$, заданих у формі з плаваючою комою, можна подати у вигляді

$$Z = 2^{P_Z} M_Z = 2^{P_X} M_X * 2^{P_Y} M_Y = 2^{(P_X+P_Y)} * (M_X * M_Y) \qquad (2.3)$$

Під час операції множення дії, що проводяться над мантисами і порядками, різні: мантиси перемножуються, порядки додаються. Очевидно, що існує варіант, коли результат множення може вийти ненормалізованим, тоді буде потрібна нормалізація з відповідною корекцією порядку результату. [6]

Етапи множення чисел з плаваючою комою:

- Отримати порядок результату, що дорівнює сумі порядків множників

  $P_Z = P_X + P_Y$

- Розрахувати мантису результату:

  $M_Z = M_X * M_Y$

- Провести нормалізацію мантиси:

  $2^{-1} \leq M_Z \leq 1$

- Записати відповідь у вигляді

  $Z = 2^{Pz} M_Z$

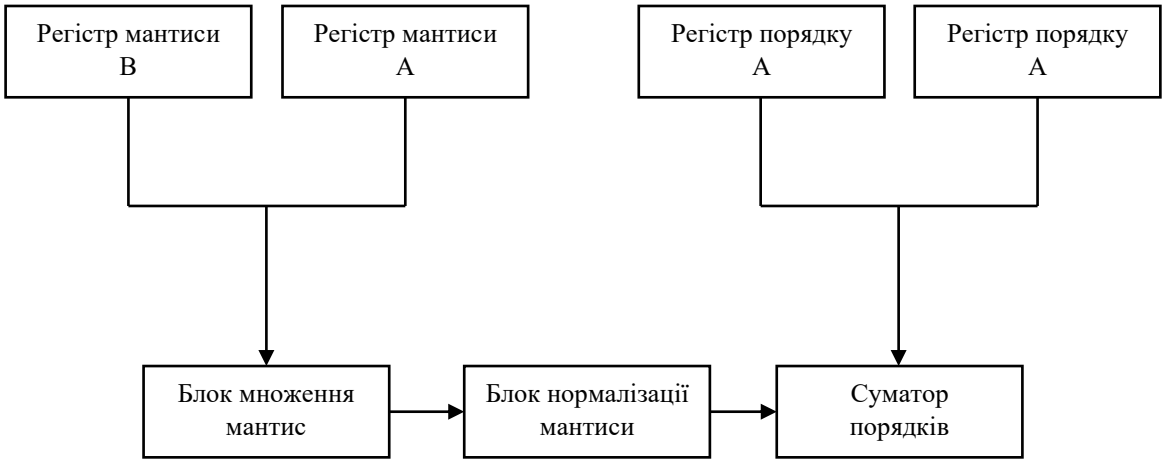


Рис 2.5. Узагальнена структурна схема множення чисел з плаваючою комою

Як наслідок, за рахунок використання чисел з рухомою комою, отримаємо збільшення діапазону допустимих значень під час множення чисел в пристрої, що неабияк збільшує точність обчислень.

2.3. Огляд надлишкової двійкової системи числення {-1, 0, 1}

Як відомо надлишкова система числення – це система числення з основою *p*, в якій кількість використовуваних символів більше ніж *p*.

Надлишкова двійкова система числення пов’язана із звичайною двійковою системою таким співвідношенням:

$$\underbrace{1\,1\,1\,\ldots 1}_{k} = \sum_{i=0}^{k-1} 2^i = 2^k - 2^0 = 1\,\underbrace{0\,0\,0\,\ldots 0\,\bar{1}}_{k} \qquad (2.4)$$

При цьому в надлишковій системі можна зменшити кількість одиниць в представленому числі. Симетричні надлишкові системи числення дозволяють в багатьох випадках спростити виконання різних арифметичних дій. Наприклад, надлишкову двійкову систему числення дуже часто використовують в деяких алгоритмах прискорення операції множення[21].

В двійковій системі числення з цифрами {-1, 0, 1} для позначення від'ємної величини до числа не потрібно приєднувати додатковий знак, отже, при виконанні арифметичних операцій над числами не потрібно проводити ще і правила операцій із знаками. Крім того, в такій системі при виконанні операції можна уникнути непотрібного поширення перенесення між розрядами далі, ніж на два сусідні розряди. [7]

Інакше кажучи, в поданій системі за рахунок введеної надлишковості, можна реалізувати методи додавання, в яких кожен розряд суми є функцією тільки суміжних справа розрядів складових величин. Це означає, що час виконання операції додавання не залежить від розрядності операндів і еквівалентний часу додавання трьох або навіть двох розрядів. Звідси виходить, що коди чисел при додаванні можуть подаватись в порядку від старших розрядів до молодших, тобто починаючи зі старших розрядів. [8]

2.4. Виконання операцій в online режимі

Швидкість обробки інформації залежить не тільки від часу виконання операцій, але і від витрат часу на обмін інформацією між елементами, тобто між обчислювальними модулями (ОМ) паралельної системи. При пересиланні інформації між ОМ на рівні окремих слів недоцільно використовувати процедури обміну інформацією через загальну пам'ять зі складними процедурами доступу. Суттєво зменшити витрати часу на обмін даними в такому випадку дозволяє використання потокових систем з

безпосередніми зв'язками (ПСБЗ) між ОМ. Одним з ефективних підходів до вирішення проблеми скорочення зв'язків між компонентами системи є застосування online методів виконання операцій, заснованих на порозрядній передачі інформації між ОМ. За допомогою такого режиму обробки даних, крім скорочення числа зв'язків, з'являється можливість виконувати залежні за даними операції в режимі часткового суміщення. За певних умовах такий режим обчислень створює передумови до скорочення часу виконання залежних за даними операцій.

У ПСБЗ виходи одних ОМ підключаються до входів інших ОМ відповідно до графу потоку даних (ГПД). ОМ працюють в online режимі. В процесі обчислень дані пересилаються безпосередньо від одних ОМ до інших, перетворюючись на кожному кроці відповідно до операцій, які виконуються в них. В такому випадку відсутні складні процедури пересилання даних між ОМ, тобто зменшуються витрати часу на обмін даними між ними.[9]

Саме тому, пристрій за рахунок використання online режиму, а саме суміщення в часі введення, обробки і виводу розрядів операндів, дозволяє неабияк збільшити швидкість виконання операцій завдяки їх паралельного виконання на рівні обробки розрядів та формуванні кінцевого результату.

ВИСНОВКИ ДО РОЗДІЛУ 2

В результаті досліджень, що складають поточний розділ роботи і які направлені на огляд машинного множення, було встановлено, що розробка пристрою множення чисел з плаваючою комою може бути реалізована у тому числі за допомогою використання схеми одного з методів множення чисел у прямому коді. Другим важливим і одним з головних факторів є безпосередньо використання чисел з рухомою комою, задля збільшення діапазону допустимих значень під час множення чисел в пристрої, що неабияк збільшує точність обчислень.

Використання надлишкової системи числення надає схемі можливість не зважати на потребу виконувати арифметичні дії над числами зі знаками, а час операцій над операндами не залежить від їх розрядності. Використовуючи ці переваги можна реалізувати метод online обчислень, тобто порозрядне введення співмножників та розрахунок результату, що неабияк збільшує швидкодію пристрою за рахунок скорочення необхідних зв’язків між ОМ.

Швидкодія, яка досягається за рахунок використання online режиму, а саме суміщення в часі введення обробки і виводу розрядів операндів, дозволяє виконувати операції паралельно на рівні обробки розрядів і є головною ціллю даної роботи.

РОЗДІЛ 3.
ОПИС РОЗРОБЛЮВАНОГО ПРИСТРОЮ НА СТРУКТУРНОМУ РІВНІ

Одним з підходів до вирішення проблеми зменшення кількості зв'язків між ОМ є використання методу поєднання процесів порозрядного введення в ОМ операндів і порозрядного формування результатів. Порозрядний обмін даними зі старших розрядів дозволяє не тільки зменшити число зв'язків між ОМ, а й забезпечити виконання залежних за даними операцій, що виконуються безпосередньо на ОМ.

Однак, швидкість обробки інформації залежить не тільки від часу виконання операцій в паралельних гілках, але і від витрат часу на обмін інформацією між гілками, тобто між обчислювальними модулями (ОМ) паралельної системи. При передачі даних між ОМ неефективно користуватись процедури обміну інформацією через загальну пам'ять зі складними процедурами доступу. Саме тому, зменшити витрати часу на обмін даними дозволяє введення потокових систем з безпосередніми зв'язками (ПСБЗ) між ОМ.

У ПСБЗ виходи одних ОМ підключаються до входів інших ОМ. ОМ працюють в online режимі, а саме, в процесі обчислень дані пересилаються безпосередньо від одних ОМ до інших, перетворюючись на кожному кроці відповідно до виконуваних арифметичних операцій. В такому випадку відсутні складні процедури пересилання інформації між ОМ, тобто в рази зменшуються витрати часу на обмін даними між ними.

3.1. Алгоритм обробки мантис і порядків співмножників в online режимі

Розглянемо узагальнену структуру алгоритму обробки співмножників для виконання операцій з плаваючою комою, продемонстровану за допомогою паралельних ОМ, кожен з яких обчислює значення виразу $Z = XY$, операнди подано в надлишковій системі числення.[10,22]

Будемо вважати, що мантиси є нормалізованими дробовими числами і представлені у формі:

$$X = \sum_{i=1}^{n} x_i 2^{-i}, Y = \sum_{i=1}^{n} y_i 2^{-i}, \quad (3.1)$$

де $x_i, y_i \in \{-1,0,1\}$ - цифри операндів, n - розрядність операндів.

Операнди, що містять тільки і розрядів праворуч від коми, позначимо відповідно через X_i, Y_i. Наприклад, $X_i = x_1 x_2 x_3 0 \ldots 0$.

Результат Z може формуватися порозрядно в системі з такими ж цифрами з запізненням на деяке число кроків.

Вимагатимемо, щоб похибка результату була знакозмінна і по абсолютній величині не перевищувала половини ваги n-го розряду після коми. Ця вимога буде виконуватися, якщо на i-му кроці буде мати місце співвідношення

$$Z_i - 2^{-i-1} \le 2^{-p} X_i Y_i < Z_i + 2^{-i-1}, \quad (3.2)$$

де p - число кроків запізнення формування цифр результату.

Використовуючи формули (3.1) і (3.2), можна отримати алгоритм обчислення Z в online режимі. Нехай $p=3$. Отже, для отримання n розрядів результату після коми необхідно виконати $n+3$ кроків обчислення.

Алгоритм обробки мантис має такий вигляд.

1. $X_0, Y_0, R_0 := 0$.

2. Для $i = \overline{1,n+3}$ виконувати пункти 3-7.

3. $H_i = 2R_{i-1} + 2^{-3} X_{i-1} y_i + 2^{-3} Y_{i-1} x_i + 2^{-3-i} x_i y_i$.

4. $X_i = X_{i-1} + x_i 2^{-i}$

5. $Y_i = Y_{i-1} + y_i 2^{-i}$

6. $z_i = \begin{cases} -1, & H_i < -2^{-1}; \\ 0, & -2^{-1} \le H_i < 2^{-1}; \\ 1, & 2^{-1} \le H_i. \end{cases}$

7. $R_i = H_i - z_i$

Тут H_i і R_i - допоміжні змінні.

Порядки є цілими числами, які надходять в ОМ разом з першим розрядом відповідного операнду і обробляються спільно з мантисами.

Обробка порядків проводиться таким чином:

1. Отримати P_X і P_Y , задати допоміжні змінні $q := 0, j := 0, s := 0$.

2. Отримати попередній порядок результату P_Z, виконавши перетворення: $P_W := P_X + P_Y$; якщо $P_W \ge 0$, то $q := 1$; $\Delta P := max(P_W, 0) - min(P_W, 0)$; $P_Z := max(P_W, 0) + p.$;

3. Якщо в Буф. M_X і Буф. M_Y є чергові цифри мантис операндів *x*, *y*, то перейти до п. 4, інакше виконати повторно п. 3.

4. Якщо $\Delta P = 0$, то прийняти в БОМ цифри мантис *x*, *y* і перейти до п. 7.

5. Якщо $q = 0$, то прийняти в БОМ цифри *x*, *y*, інакше - нулі замість *x*, *y*.

6. $\Delta P := \Delta P - 1$.

7. Сформувати цифру результату z.

8. Якщо $s \ne 0$, то перейти до п. 11.

9. Якщо $z = 0$, то $P_Z := P_Z - 1$ і перейти до п. 3.

10. $s := 1$ і видати з БОП остаточний порядок результату P_Z.

11. Видати з БОМ чергову цифру результату z і $j := j + 1$.

12. Якщо $j \ne n$, то перейти до п. 3, інакше кінець операції.

3.2. Загальний вигляд пристрою

Розібравши алгоритм поданий в п. 3.1. можна впевнено використати його основу для створення пристрою для множення чисел з плаваючою комою в online режимі обчислень за допомогою надлишкової двійкової системи числення з цифрами {-1, 0, 1}.

Для досягнення поставленої мети використано пристрій, що містить суматор порядків, регістр управління, блок порівняння з мінімальним порядком, регістр порядку результату, суматор і блок множення мантис, причому виходи суматора порядків підключені до першої групи входів суматора, виходи якого з'єднано з інформаційними входами регістра порядку результату, виходи якого підключені до вихідних шин порядку результату пристрою, виходи блоку множення мантис підключені до вихідних шин мантиси результату пристрою, входи блоку множення мантис з'єднані з вхідними шинами мантис двох множників пристрою, містить лічильник нормалізації, елемент АБО, тригер і блок аналізу розрядів, входи якого з'єднані з виходами блоку множення мантис, вхід обнулення з'єднаний з установочним входом лічильника нормалізації і виходом елемента АБО, перший вхід якого з'єднаний з виходом блоку порівняння з мінімальним порядком, другий вхід елемента АБО з'єднаний з нульовим входом тригера, входом обнулення регістра порядку результату і з виходом старшого розряду регістра управління, вхід установки якого з'єднаний з входом блокування блоку порівняння з мінімальним порядком, вхід дозволу запису регістра порядку результату і виходом тригера, одиничний вхід якого з'єднаний з виходом блоку аналізу розрядів, другий вихід якого з'єднаний з інкриментом лічильника нормалізації, виходи якого підключені до другої групи входів суматора, виходи якого підключені до інформаційних входів блоку порівняння з мінімальним порядком, входи суматора порядків підключені до вхідних шин порядків двох множників пристрою.

Крім того, блок аналізу розрядів містить елемент ВИКЛЮЧНЕ АБО і елемент НЕ, вхід якого з'єднаний з першим виходом блоку і виходом елемента ВИКЛЮЧНЕ АБО, входи якого з'єднані з входами блоку, вихід елемента НЕ підключений до другого виходу блоку.

На Рис.3.1. зображена структурна схема розроблюваного пристрою для множення чисел; на Рис.3.2. - структурна схема блоку аналізу розрядів.

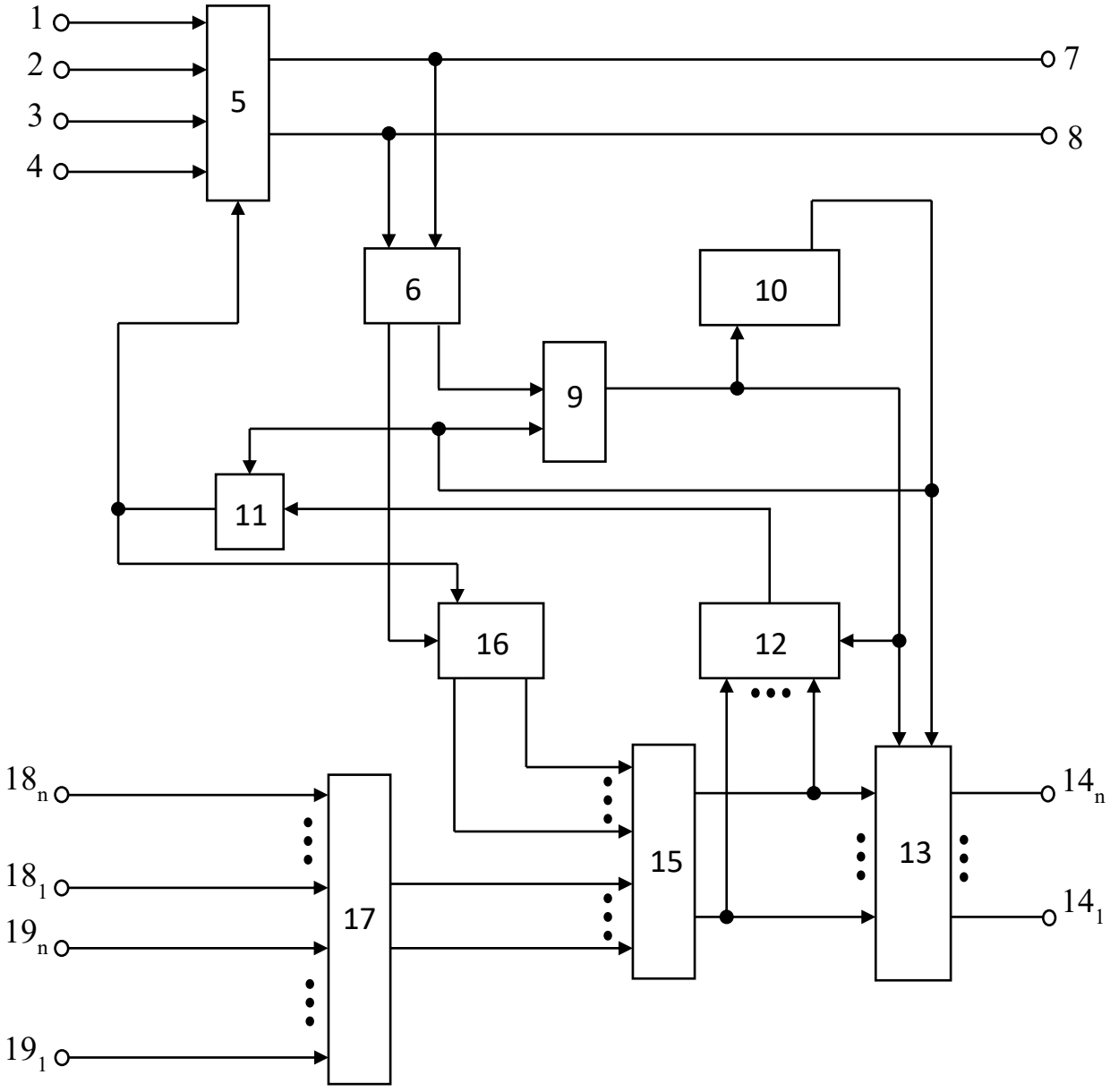


Рис. 3.1. Структурна схема пристрою для множення чисел

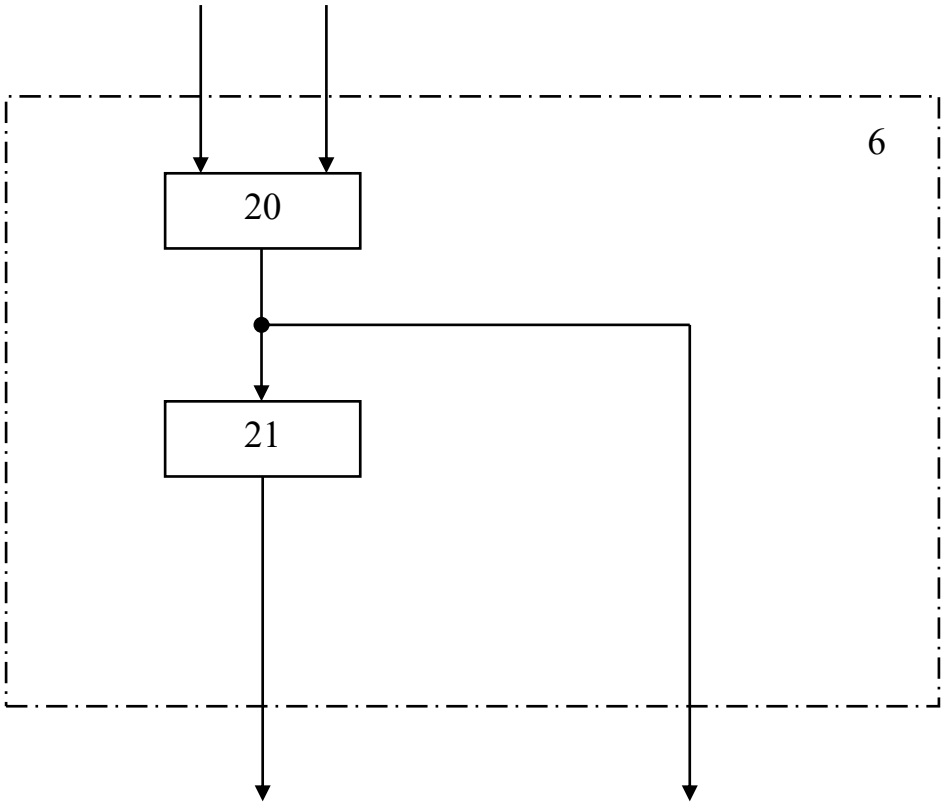


Рис.3.2. Структурна схема блоку аналізу розрядів

3.3. Структура пристрою для множення чисел з плаваючою комою

Пристрій містить вхідні шини мантиси першого множника (1, 2), вхідні шини мантиси другого множника (3, 4) і блок множення мантис (5), входи якого з'єднані з входами 1, 2 і 3, 4. До складу пристрою також входять блок аналізу розрядів (6), до входів якого підключені виходи блоку 6 і вихідні шини мантиси результату пристрою (7, 8), тригер (9), регістр управління (10), елемент АБО (11), блок порівняння з мінімальним порядком (12), регістр порядку результату(13).

До виходів регістра (13) підключені вихідні шини порядку результату пристрою (14_1-14_n). Крім того, до складу пристрою включені суматор (15), лічильник нормалізації (16) і суматор порядків (17), до входів якого підключені вхідні шини порядку першого множника (18_1-18_n) і вхідні шини порядку другого множника (19_1-19_n). Виходи суматора 17 з'єднані з першою групою входів суматора 15, друга група входів якого підключена до виходів

лічильника 16. Цей лічильник своїм установчим входом з'єднаний з виходом елемента АБО 11 і входом обнулення блоку 5. Перший вхід елемента АБО 11 сполучений з виходом блоку 12, а другий вхід цього елемента - з нульовим входом тригера, входом обнулення регістра 13 і з виходом старшого розряду регістра 10. Вхід установки цього регістра підключений до входу блокування блоку 12, входу дозволу запису регістра 13 і виходу тригера 9, одиничний вхід якого з'єднаний з першим виходом блоку 6. Другий вихід блоку 6 підключений до входу лічильника 16. Крім того виходи суматора 15 з'єднані з інформаційними входами регістра 13 і блоку 12.

Блок аналізу розрядів (див. Рис. 3.2) містить елемент ВИКЛЮЧНЕ АБО (20) і елемент НЕ (21), вхід якого з'єднаний з першим виходом блоку 6 і виходом елемента 20. Входи елемента ВИКЛЮЧНЕ АБО 20 підключені до входів блоку 6, а вихід елемента НЕ 21 з'єднаний з другим виходом того ж блоку.

Блок 6 використовується для розпізнавання бінарних кодів цифр мантиси результату, які в кожному циклі обчислень формує на своїх виходах блок 5. Це розпізнавання здійснюється блоком 6 по сигналам, які синхронізують надходження розрядів мантиси співмножників і видачу розрядів мантиси добутку.

Блок множення мантис 5 призначений для перемноження мантис співмножників, що надходять порозрядно, починаючи зі старших розрядів, на його входи.

Регістр управління 10 являє собою m-розрядний (m - розрядність мантиси) зсувний регістр.

Блок порівняння з мінімальним порядком 12 являє собою схему порівняння чисел, що виконує порівняння обчислюваного значення порядку результату з величиною мінімального порядку, при якому отриманий результат можна вважати рівним нулю. Величина мінімального порядку при

n-розрядах порядку дорівнює -2^{n-1} (один розряд порядку використовується для представлення його знаку).

Лічильник нормалізації 16 - це тригерний n-розрядний двійковий лічильник, що має установчий вхід. При подачі на цей вхід сигналу логічної “1” в тригери лічильника заноситься n-розрядний двійковий зворотній код числа “-2”, що є початковим станом цього лічильника.

3.4. Застосування online обчислень в роботі пристрою

Перед виконанням операції всі регістри, лічильник і тригери встановлюються в початковий стан. На шини 18_1-18_n пристрою надходить двійковий код порядку першого множника, а на шини 19_1-19_n двійковий код порядку другого множника. Суматор 17 формує суму надійшовших на його входи порядків співмножників з урахуванням їх знаків. З виходів суматора 17 ця сума порядків подається на суматор 15 для подальших перетворень у відповідності зі значенням цифр мантиси добутку, які в кожному і-му циклі обчислень видаються з блоку 5. (i = 1, 2, 3 ...)

Одночасно з надходженням порядків співмножників, на вхідні шини 1, 2 і 3, 4 надходять коди старших розрядів мантиси співмножників. По кожному і-му синхронізуючому сигналу блок 5 при надходженні на його входи розрядів мантиси співмножників з вагою 2, формує розряди мантиси добутку з вагою 2^{2-i}, тобто розряди мантиси добутку виходять з запізненням на два цикли обчислень по відношенню до вхідних розрядів. При цьому забезпечується суміщення в часі процесів порозрядного введення мантис співмножників і їх обробка. У кожному і-му циклі обчислень через вихідні шини 7 і 8 по синхронізуючим сигналам з пристрою видаються коди розрядів мантиси добутку, а на вхідні шини 1, 2 і 3, 4 пристрою надходять наступні коди мантис співмножників. Для суміщення в часі процесу введення і перемноження з процесом нормалізації мантиси добутку і обчислення

остаточного порядку результату блок 6 по кожному і-му синхронізуючому сигналу виконує декодування кодів розрядів мантиси добутку. Це декодування кодів розрядів полягає в наступному.

Для представлення мантис співмножників і добутку використовується надлишкова двійкова система числення з цифрами “-1”, “0” і “1”. Ці цифри в свою чергу можуть бути подані кодом цифр “1” і “0” канонічної двійковій системи числення. Так, наприклад цифрі "-1" відповідає наявність сигналу логічної “1” на вихідних шинах 1 або 3, а також на вихідній шині 7. Наявність сигналу логічної “1” на вхідних шинах 2 або 4, а також на вихідний шині 8 відповідає цифрі “1”. Для цифри “0” відповідає відсутність сигналів логічної “1” на вхідних шинах 1-4 або на вихідних шинах 7 і 8.
Якщо цифра мантиси добутку “0”, то блок 6 на своєму другому виході формує сигнал логічної “1”. У випадку коли ця цифра дорівнює “1” або “-1”, блок 6 видає сигнал логічної “1” на свій перший вихід.

Нормалізація мантиси добутку і одночасне формування його остаточного порядку відбувається наступним чином. Починаючи з першої старшої цифри мантиси добутку, що дорівнює нулю, блок 6 видає сигнал логічної “1” на свій другий вихід. Цей сигнал надходить на інкримент лічильника 16 і збільшує його вміст на одиницю. При цьому суматор 15 виконує віднімання вмісту лічильника 16 і числа на виході суматора 17, тобто зменшує суму порядків співмножників на одиницю. Такий процес відбувається до отримання на виходах блоку 5 першої значущої цифри мантиси добутку, що дорівнює “1” або “-1”. У цьому випадку на першому виході блоку 6 по синхронізуючому сигналу з'являється сигнал логічної “1”, який встановлює на виході тригера 9 сигнал логічної “1”. У свою чергу цей сигнал буде забороняти роботу блоку 12 (на його виході в цьому випадку буде сигнал логічного “0” протягом всього часу дії логічної "1" на вході блокування цього блоку) і зробить установку в одиницю перший молодший розряд регістра 10, а решта старші (m-1) розряди цього регістра будуть

встановлені в нуль. Крім того, логічна “1” від тригера 9 дозволить записати в регістр 13 значення обчисленого на суматорі 15 остаточного порядку результату. На вихідних шина x 14_1-14_n при цьому буде встановлено порядок ще не сформованої, але вже нормалізованої мантиси результату. Під дією наступних синхронізуючих сигналів в регістрі 10 відбувається зрушення одиниці від молодших розрядів до старших цього регістра, що дозволяє вести відлік кількості виданих з пристрою розрядів нормалізованої мантиси добутку. Обчислення розрядів мантиси добутку припиняються, коли одиниця в регістрі 10 перейде в його старший m-ий розряд. Через вихідні шини 7 і 8 при цьому буде видано m-розрядів нормалізованої мантиси добутку. Логічна “1” в m-ому розряді регістра 10 встановить в початковий стан тригер 9, регістр 13 і через елемент АБО 11 - блок 5 і лічильник 16. Наступним синхронізуючим сигналом m-ий розряд регістра 10 встановиться в нуль. Після цього пристрій готовий до виконання операції множення над наступною парою операндів.

Якщо ж в процесі одночасного формування мантиси добутку, її нормалізації та обчислення остаточного порядку результату число на виходах суматора 15 стане рівне значенню мінімального порядку, блок 12 визначить це, встановивши на своєму виході сигнал логічної "1". Такий сигнал через елемент АБО 11 приведе в початковий стан блок 5 і лічильник 16, підготувавши пристрій для виконання операції множення над наступною парою операндів.

Таким чином, пропонований пристрій дозволяє множити операнди, представлені в формі з плаваючою комою. При цьому підвищення швидкодії досягається за рахунок поєднання в часі процесу порозрядного введення операндів з процесом обчислення в пристрої. Спрощену блок-схему виконання операції множення подано на Рис. 3.3.

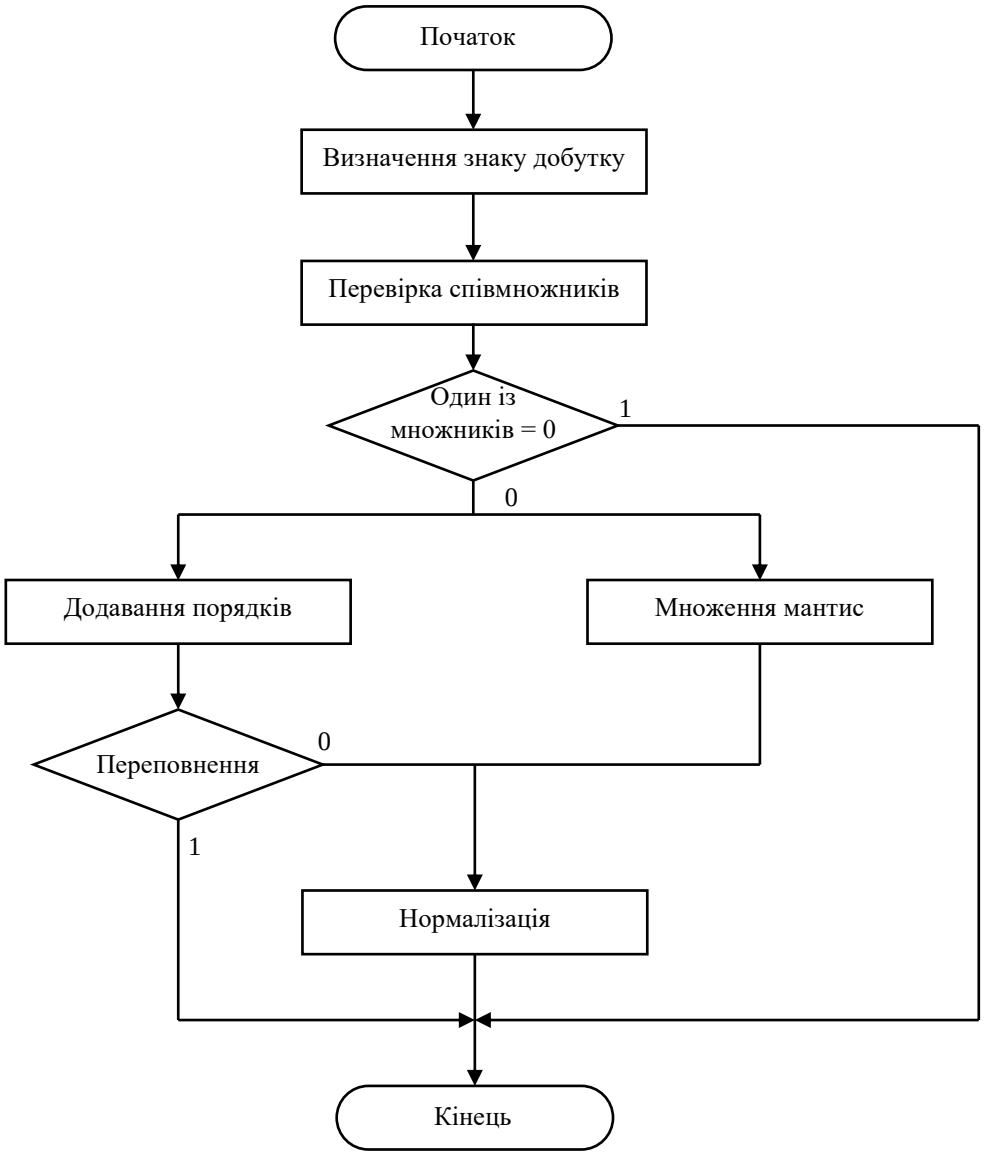


Рис. 3.3. Спрощена блок-схема операції множення чисел з плаваючою комою

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі показано можливість реалізації online обчислень з плаваючою комою в обчислювальних системах. Для вирішення проблеми громіздкості реалізації ПСБЗ використовується так звана система подання digit-by-digit, яка полягає в реалізації алгоритму обробки співмножників, або іншими словами порозрядного введення операндів, і яка дозволяє розширити діапазон представлення чисел, скорочуючи необхідний ресурс під час реалізації завдяки малому числу зв'язків між модулями, що безсумнівно збільшує швидкість обробки вхідних даних та розрахунок результату.

Розроблюваний пристрій може бути використаний в цифрових обчислювальних машинах для множення чисел у формі з плаваючою комою. Головною перевагою такого пристрою є використання схеми множення двох чисел у формі з плаваючою комою в online режимі обчислень, що безсумнівно має неймовірну перевагу над прототипами для множення чисел з фіксованою комою у точності розрахунків – навіть незначні погрішності з плином часу та накопиченням результату можуть призвести до суттєвих помилок. В свою чергу, надійність пристрою заснована на порозрядній передачі даних для спрощення обміну інформації між внутрішніми елементами інтегральної схеми.

Саме такий варіант реалізації даного пристрою дозволяє поєднати в часі процес порозрядного введення операндів, що формуються поза системою, і їх обробку, що неабияк збільшить швидкодію та надійність пристрою, та зменшить ресурсозалежність порівняно з його аналогами. Також даний пристрій буде позбавлений проблеми переповнення під час розрахунків, задля збільшення точності, що в інших аналогах призводило до скорочення діапазону обчислень та значно погіршувало ефективність машини в цілому.

РОЗДІЛ 4.
РОЗРОБКА ПРИСТРОЮ НА ПРОГРАМНОМУ РІВНІ ТА ЙОГО РЕАЛІЗАЦІЯ НА ПЛІС

4.1. ПЛІС як спосіб створення цифрових інтегральних схем

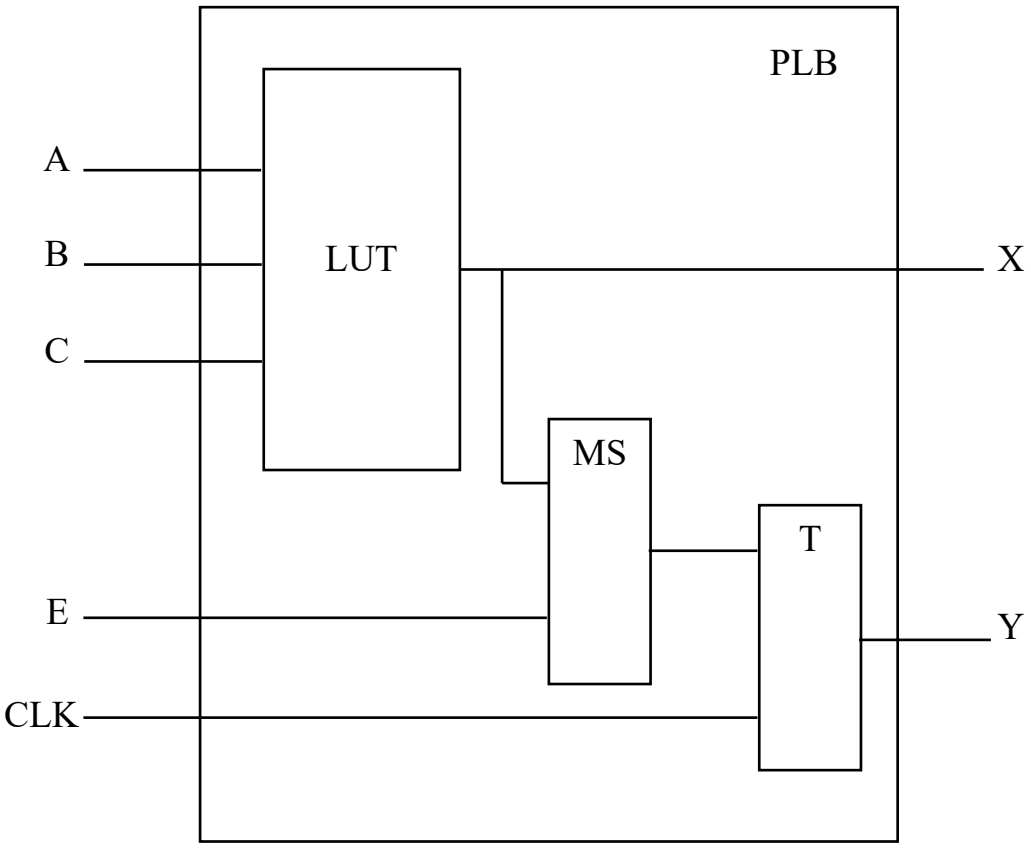
Забезпечити відносну гнучкість інтегральної мікросхеми можна, якщо керувати з'єднаннями не механічно, шляхом додавання або розриву провідників, а електрично, програмуючи заздалегідь передбачені на кристалі з'єднувачі. Програмована логічна інтегральна схема (ПЛІС) – схема, логіка якої задається під час програмування, в той час як на звичайних цифрових мікросхемах вона задається під час виготовлення[11].

У загальному випадку мікросхеми містять обчислювальні ядра, пам'ять і програмовану логіку, що дає потенційну можливість оптимально адаптувати апаратні засоби до конкретного застосування[9].

Мікросхеми ПЛІС протягом свого існування зазнали досить помітну еволюцію. Від простих програмованих пристроїв таких як Simple Programmable Logic Device (SPLD), які мали обмежені можливості модифікації, вони пройшли шлях до комплексних пристроїв з гнучкими можливостями побудови схеми – Complex Programmable Logic Devices (CPLD) і Field Programmable Gate Array (FPGA). FPGA в даний час виробляються із застосуванням найбільш сучасних технологічних процесів, що забезпечує високі технічні характеристики і великий обсяг в логічних елементах. [11,16]

ПЛІС виступає в ролі компоненту для створення цифрових схем. Досягнення в області інтегральної технології дозволяють створювати паралельні системи CSoC (Configurable System on Chip)[12-13], які реалізуються на ПЛІС з використанням інтерактивних засобів розробки.

Основу ПЛІС склала концепція використання так званих програмованих логічних блоків (PLB), до складу яких входила таблиця відповідності (LUT), яка по суті являлась таким собі статичним ОЗП, основою логіки якого стало «Ввімкнув – записав, вимкнув – видалив», регістр, що виконував роль одного з тригерів (T) та мультиплексор (MS), а



також деякі інші елементи відповідно до моделі та структури мікросхеми. На рис. 4.1. подано узагальнену структуру простого PLB.

Рис. 4.1. Простий програмований логічний блок

В свою чергу, PLB є найменшою складовою ПЛІС і, у міру розвитку технологій, схема може містити в собі десятки або навіть сотні тисяч таких блоків, які з’єднані між собою та блоками вводу\виводу.[14] На рис. 4.2. продемонстровано узагальнену спрощену архітектуру будь-якого ПЛІС.

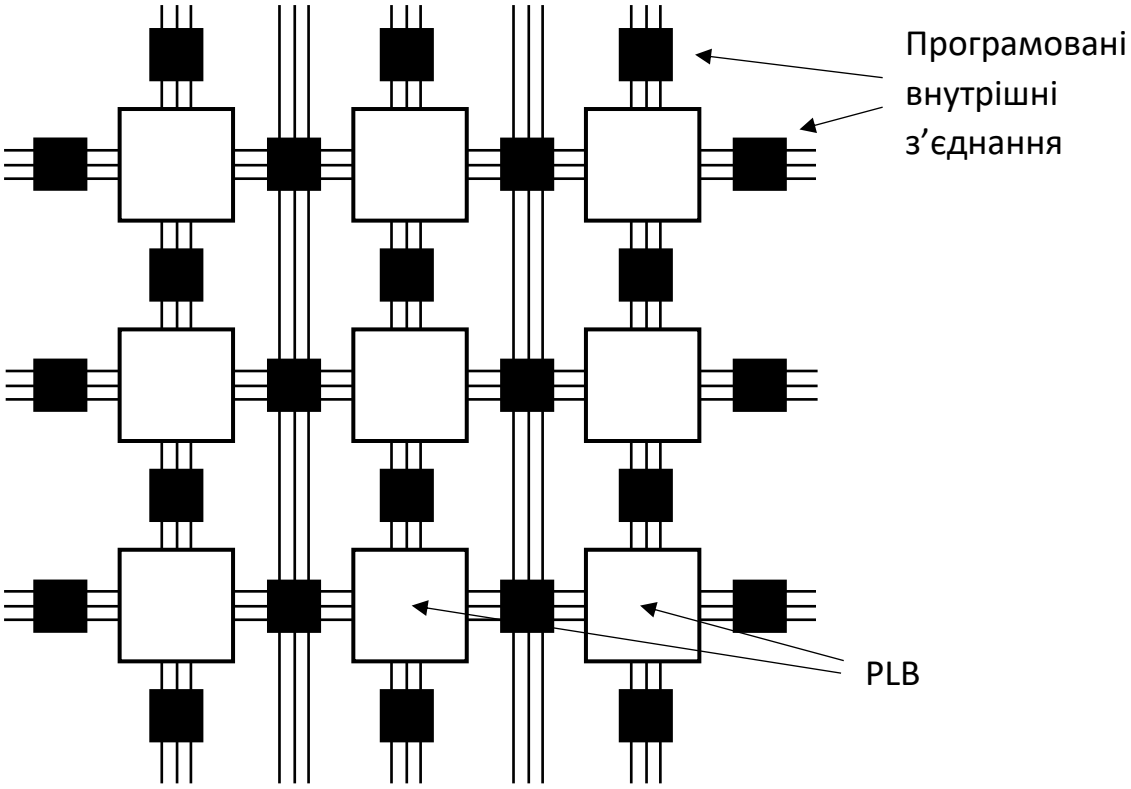


Рис. 4.2. Спрощена архітектура ПЛІС

4.2. Середовище програмування і розробки на ПЛІС

Для програмної реалізації структурної схеми (див. рис. 3.1), було використано середовище для програмування мікросхем САПР Quartus II, яке включає в себе опис апаратного забезпечення на VHDL або Verilog HDL та візуальне редагування логічних схем, а для демонстрації правильності роботи множення за допомогою векторних сигналів застосовано середовище опису і моделювання електронного обладнання ModelSIM.

Під час створення проекту в середовищі Quartus II слід розглянути момент зображений на рис. 4.3.

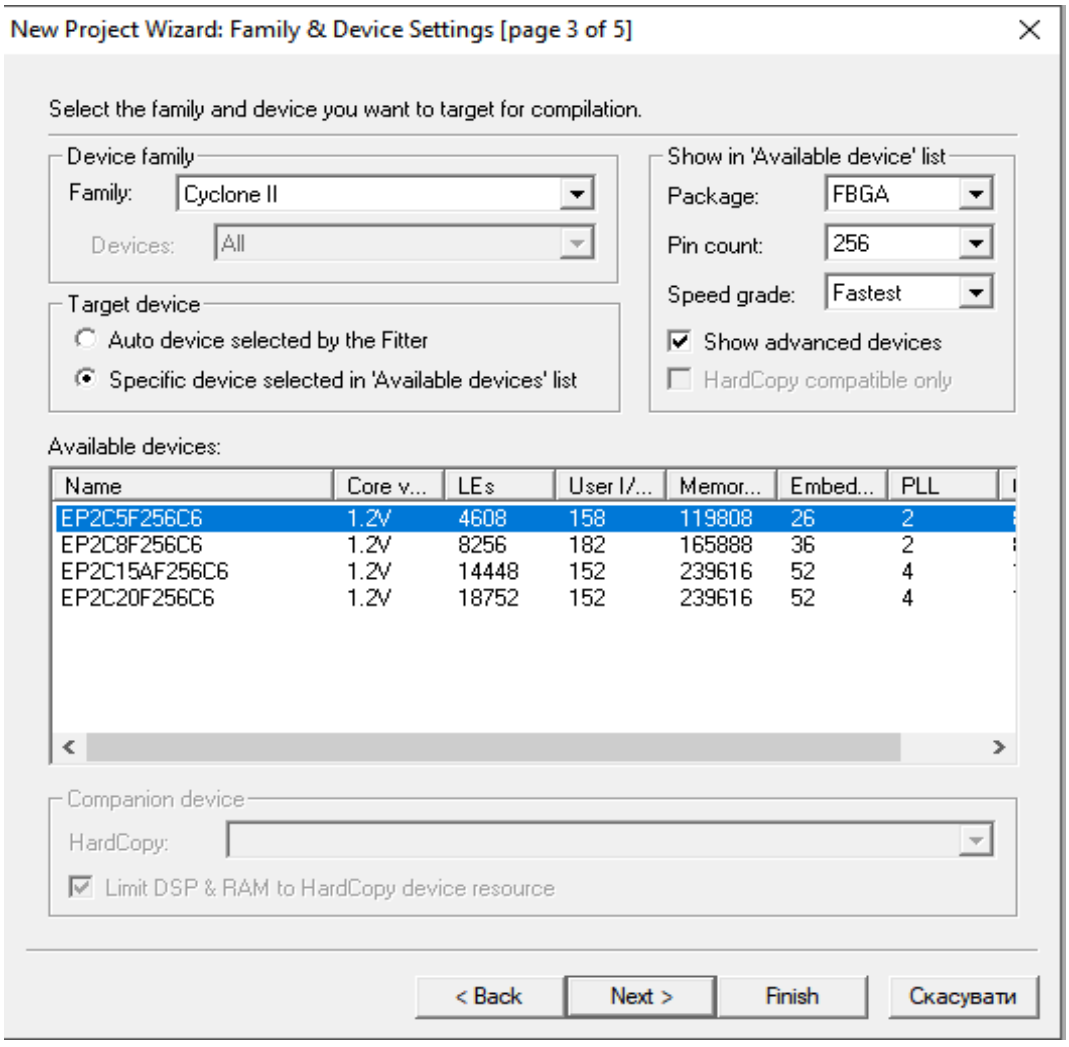


Рис. 4.3. Діалогове вікно створення проекту у САПР Quartus II

У цьому діалоговому вікні обираємо ПЛІС, яку ми безпосередньо будемо програмувати. Перш за все потрібно визначити сімейство та тип корпусу інтегральної мікросхеми. Потім в назві буде подана кількість використовуваних ніжок, як от наприклад EP2C5F**256**C6. І в останню чергу обираємо параметр, який характеризує час затримки проходження сигналу між внутрішніми зв'язками в ПЛІС і визначається останньою цифрою(ами) в назві мікросхеми. Слід зазначити, що проект, розроблений для ПЛІС з параметром Speed grade рівним 5 без проблем буде працювати і в ПЛІС з Speed grade 10, але якщо перша буде мати більший Speed grade, то мікросхема не те що не буде працювати, а більш за все навіть не запуститься. Саме тому слід це брати до уваги під час розробки задля передбачення можливих проблем з сумісністю. Наостанок, слід обрати з отриманого відфільтрованого списку бажану ПЛІС, посилаючись вже на її власні характеристики, такі як вольтаж, кількість входів та виходів, розмір пам’яті тощо. Для реалізації даного проекту було використано ПЛІС типу EP2C20F484C6 сімейства Cyclone II з корпусом виду FBGA, кількістю ніжок 484 і часом затримки внутрішніх сигналів 6. В налаштуваннях проекту за необхідності в будь-який час можна змінити використовувану мікросхему на іншу. Детальніше з даною ПЛІС можна ознайомотись в [15].

Наступним кроком розробки даного проекту буде створення файлу, який містить модуль програми написаний на одній з мов опису апаратури. Найкраще для цього зарекомендували себе VHDL і Verilog HDL (не слід їх плутати між собою).

В даній роботі було використано саме VHDL з ряду причин. Насамперед, VHDL являється мовою високого рівня, що говорить саме за себе. VHDL – це строго типізована і більш детальна мова ніж Verilog HDL. Проекти, які написані на цій мові потребують постійної необхідності перетворення даних з одного типу в інший, але компенсується це тим, що VHDL відразу відображає помилки, які б можливо пропустив Verilog

HDL[17]. Головною перевагою стало те, що VHDL має однозначно-визначену семантику, що дозволяє проектам, які розробляються за допомогою даної мови, набагато краще переноситись між різними САПР, а якщо детальніше – у декілька разів точніше передавати всі тонкості вихідного проекту. Інакше кажучи, у порівнянні цих двох мов, перевагу отримали ефективність та правильність створення розробки за допомогою VHDL, навіть попри швидкість опису моделі на Verilog HDL.

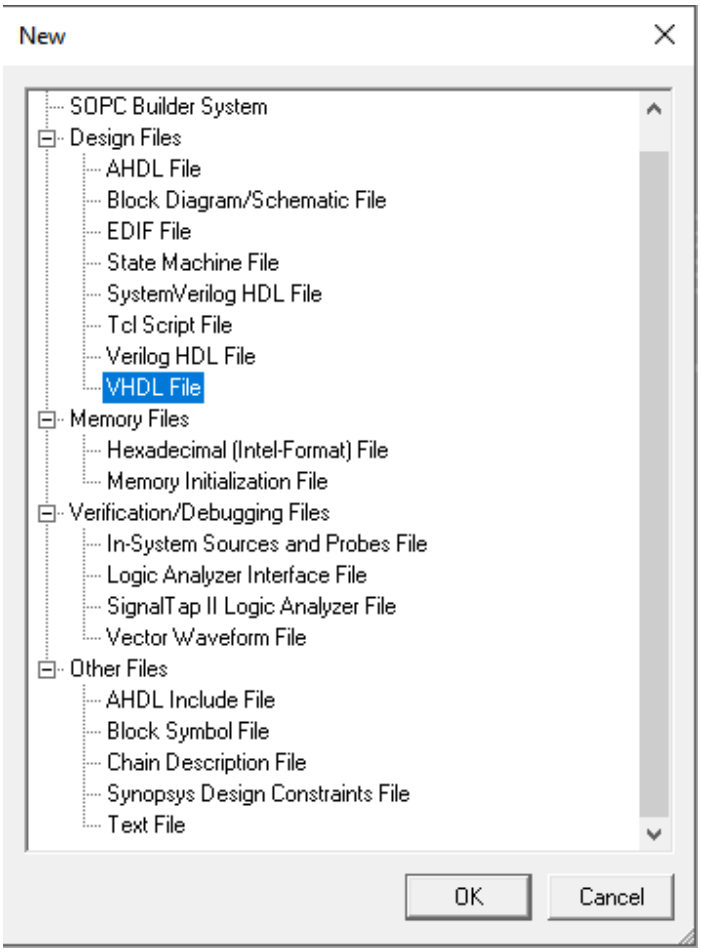


Рис. 4.4. Діалогове вікно вибору типу створюваного файлу

4.3. Симуляція пристрою в САПР Quartus II

Після створення проекту слід описати пристрій за допомогою мови опису апаратури (в даному випадку – VHDL). Звісно, опис повністю всіх методів і понять мови не має ніякого логічного сенсу, але огляд деяких складових опису розроблюваного пристрою допоможе більш детальніше зрозуміти мікропрограмний рівень обробки даних, що виконується над елементами ПЛІС під час online обчислень. [18-19]

Застосування визначених мовою стандартних бібліотек за допомогою library і use:

```
library ieee;
use ieee.std_logic_1164.all;
```

Даний код дозволяє присвоювати сигналам не тільки значення 0 і 1.

Опис точок входу і виходу пристрою за допомогою ключового слова entity. Іншими словами – опис інтерфейсу:

```
entity multiplier is
        port (clk   : in  std_logic;
                reset : in  std_logic;
                addrA : in  std_logic_vector(2  downto 0);   --Вхідні сигнали
                addrB : in  std_logic_vector(2  downto 0);
               showAB: in  std_logic;
               outAB : out std_logic_vector(31 downto 0); --Вихідні сигнали
               );
end multiplier;
```

Опис архітектури складових пристрою та архітектури пристрою в цілому здійснюється за допомогою ключового слова architecture і структури поданої нижче:

```
architecture synt of multiplier is              -- опис архітектури пристрою
    component device                            --опис архітектури компоненту
        port (a,b: in std_logic_vector(31 downto 0);
            clk,en,rst: in std_logic;
            c: out std_logic_vector(31 downto 0);
            done: out std_logic);
    end component;
    begin
        process(CurrentState) begin
            NextState<=initial;      -- опис залежності сигналів
            result_graph_s<=(others=>'0');
    end process;
end synt;
```

Слід зазначити, що якщо в систему входять декілька однотипних елементів, наприклад 4 однакові регістри, то в даній частині програми слід описувати лише один такий елемент[20]. Повну мікропрограму пристрою описаного мовою VHDL представлено в розділі додатків (див. додаток 2).

Після опису пристрою слідує наступний крок, а саме компіляція проекту, що дозволяє прослідкувати за цілісність коду, структуризацію, правильністю виконання на мікропрограмному рівні, а також ресурсозалежність і час виконання. Для цього скористаємось функцією САПР Quartus II – Start Compilation. Результати компіляції подано на рис. 4.5.

Flow Status	Successful - Fri May 22 16:09:19 2020
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	multiplier
Top-level Entity Name	multiplier
Family	Cyclone II
Device	EP2C20F484C6
Timing Models	Final
Met timing requirements	No
Total logic elements	820 / 18,752 (4 %)
Total combinational functions	819 / 18,752 (4 %)
Dedicated logic registers	101 / 18,752 (< 1 %)
Total registers	101
Total pins	82 / 315 (26 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

Рис. 4.5. Результати компіляції проекту в САПР Quartus II

Як можна прослідкувати, компіляція пройшла успішно, без помилок, а використання ресурсів даної ПЛІС у процентному співвідношенні вказує на доцільність використання даного методу, оскільки завдяки цьому досягається гнучкість майбутнього проекту, який буде включати даний пристрій і буде мати більш ніж достатньо ресурсів для реалізації інших пристроїв та систем.

4.4. Моделювання роботи пристрою для множення чисел

Оскільки детальну роботу пристрою можна показати лише з використанням стенду для налагодження роботи ПЛІС, то для демонстрації векторних сигналів використаємо середовище опису і моделювання електронного обладнання ModelSIM від компанії Mentor Graphics. Детальну інформацію щодо створення проекту за допомогою даного середовища, а також використання мови VHDL в ньому подано в [23-25].

Одною з складових пристрою є структура mult_int, яка виконує функцію порозрядного введення та обчислення операндів з використанням індексу попереднього розряду результату. Після створення та компіляції проекту слід розпочати симуляцію та додати за допомогою функції «Execute Macro» в симуляцію файл конфігурації сигналів test_mult.do. За допомогою деяких змін в структурі множення для того, щоб явно побачити принцип її дії, можна отримати результат роботи, продемонстрований на рис. 4.6.

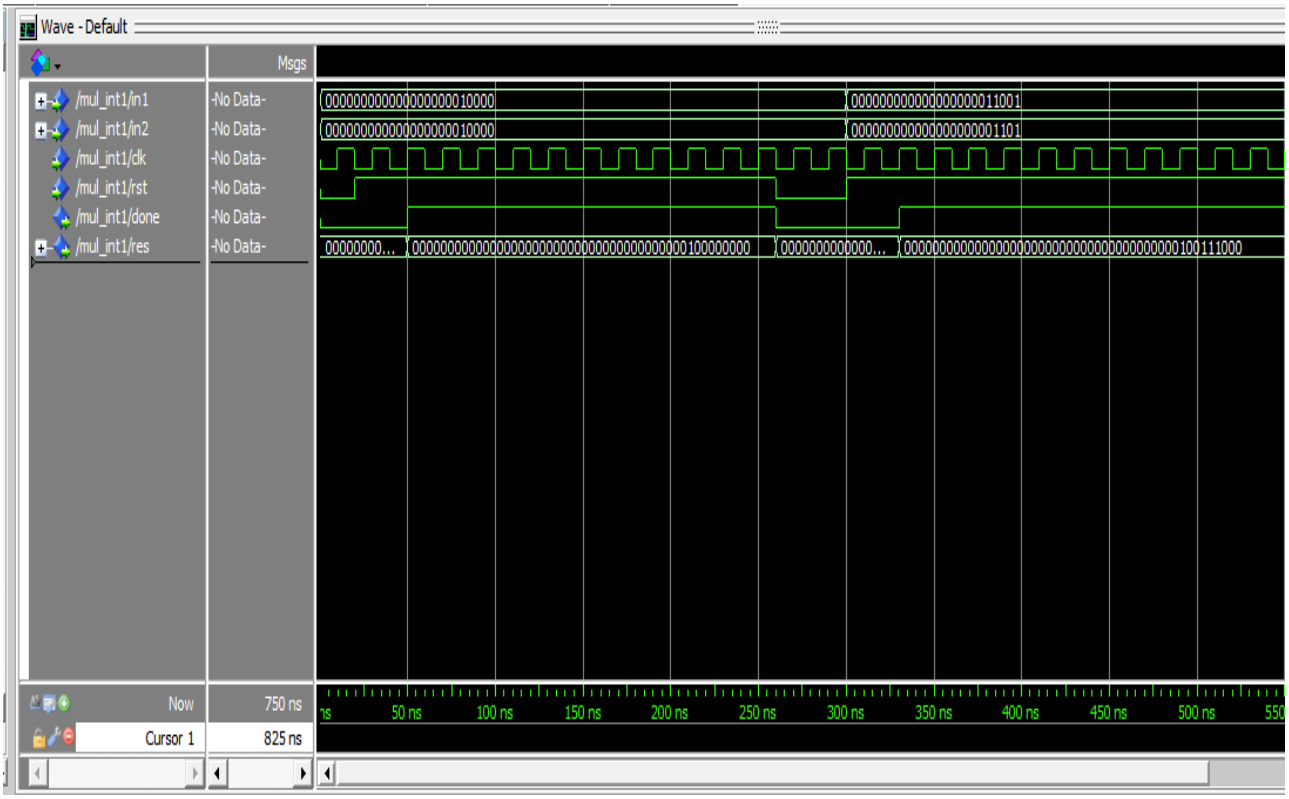


Рис. 4.6. Графічне вікно результатів моделювання mult_int

З рис. 4.6 видно, що розрахунки в даній структурі проводяться одночасно, пристрій правильно множить два різних числа, а дані, які отримуються в результаті, застраховані від створення переповнень.

Головною частиною проекту є структура datapath, яка і формує кінцевий результат. Нижче подано декілька прикладів множення.

Приклад 1

$A = 134.0625_{10} = 01000011000001100001000000000000_2$

$B = -2.25_{10} = 11000000000100000000000000000000_2$

$Res = -301.640625_{10} = 11000011100101101010010000000000_2$

Результат множення продемонстровано на рис. 4.7.

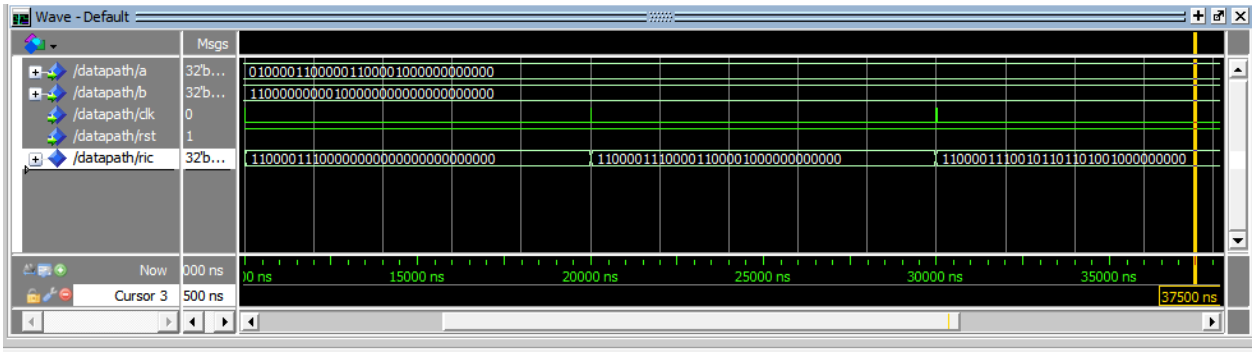


Рис. 4.7. Графічне вікно результатів моделювання «Приклад 1».

Приклад 2

$A = -14.5_{10} = 11000001011010000000000000000000_2$

$B = -0.375_{10} = 10111110110000000000000000000000_2$

$Res = 5.4375_{10} = 01000000101011100000000000000000_2$

Результат множення подано на рис. 4.8.

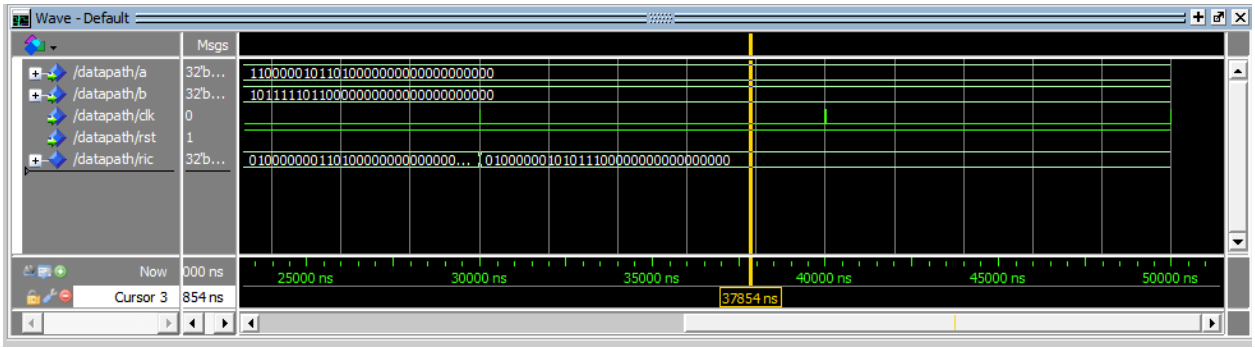


Рис. 4.8. Графічне вікно результатів моделювання «Приклад 2».

ВИСНОВКИ ДО РОЗДІЛУ 4

В даному розділі бакалаврської роботи було продемонстровано роботу даного пристрою за допомогою моделювання в САПР Quartus II, а саме виконання на базі ПЛІС типу EP2C20F484C6 сімейства Cyclone II компанії Altera. Використання порозрядного введення операндів або так званого online режиму обчислень дозволяє застосовувати набагато менше ресурсів однієї ПЛІС. В результаті компіляції було отримано список ресурсів схеми, які використовуються даним пристроєм і , як наслідок, можна зазначити, що порівняно з іншими аналогами даного проекту, під час online режиму обчислення та передачі даних використовується в декілька разів менша кількість логічних елементів, регістрів та ніжок, які належать ПЛІС, що безсумнівно надає можливість реалізації на одній і тій самій мікросхемі ряд інших пристроїв, які безпосередньо відносяться до даної системи.

З даних, отриманих за допомогою моделювання, можна зробити висновок, що пристрій працює належним чином і реалізує вирішення більшості недоліків його прототипів: множення відбувається над числами з плаваючою комою, що без сумнівів збільшує діапазон допустимих значень результату та точність обчислень; збільшення розрядності результату, що веде за собою унеможливлення виникнення переповнень; підвищення надійності пристрою за рахунок спрощення комутації всередині ПЛІС та зменшення кількості потрібних ресурсів; використання методу «digit by digit», який полягає в порозрядному введенні операндів та їх обчисленні і неабияк збільшує швидкість пристрою в цілому.

ВИСНОВКИ

Сучасний етап розвитку комп'ютерних технологій включає в себе одну з головних складових, а саме процес обробки інформації, що веде за собою потік виконання певної кількості різних арифметико-логічних операцій над числами різного типу. Таким чином, дослідження та розробка пристроїв для виконання арифметичних операцій в комп'ютерних системах, являють собою надзвичайно важливу і актуальну задачу поставлену перед розробниками для покращення та розвитку теперішній технологій.

Беручи до уваги дослідження, проведені в даній бакалаврській роботі, а також всі оглянуті та вивчені роботи авторів направлені на створення пристрою для множення в ЕОМ, зокрема для множення в online режимі в надлишковій системі числення, було отримано ряд висновків.

Подання чисел з фіксованою комою в обчислювальних пристроях призводить до ряду недоліків а саме:

- Істотні обмеження в діапазоні можливих значень;
- Накопичення похибки під час виконання арифметичних операцій за рахунок фіксованої розрядності результату;
- Необхідність масштабування операндів для забезпечення необхідного співвідношення їх величин під час певного етапу обчислень;
- Необхідність введення в систему елементів затримки під час вводу операндів з різних елементів.;

В свою чергу пристрої, які використовують подання чисел у формі із плаваючою комою, виконують арифметичні операції як над мантисами чисел, так і над їхніми порядками, що часто потребує виконання операції нормалізації результату, що веде за собою введення в схему нових конструкцій і як наслідок - ускладнює та сповільнює систему загалом. Тому машини із плаваючою комою є більш складними в побудові і поступаються у швидкодії машинам на яких реалізовано операції з фіксованою комою.

Підвищення продуктивності ЕОМ являється однією з основних задач структурного покращення обчислювальної техніки. Розробка даної роботи, а саме пристрій множення чисел з плаваючою комою може бути реалізований у тому числі за допомогою використання схеми одного з методів множення чисел у прямому коді. Другим важливим і одним з головних факторів є безпосередньо використання чисел з рухомою комою, задля збільшення діапазону допустимих значень під час множення чисел в пристрої, що неабияк збільшує точність обчислень.

Використання надлишкової системи числення надає схемі можливість не зважати на потребу виконувати арифметичні дії над числами зі знаками, а час операцій над операндами не залежить від їх розрядності. Використовуючи ці переваги можна реалізувати метод online обчислень, тобто порозрядне введення співмножників та розрахунок результату, що неабияк збільшує швидкодію пристрою за рахунок скорочення необхідних зв’язків між ОМ.

Саме скорочення таким чином необхідних зв’язків між ОМ дало можливість за необхідності реалізовувати на тій же самій мікросхемі ряд інших пристроїв. Побудова системи на одній ПЛІС забезпечує підвищення її надійності, зменшення енергоспоживання та габаритів, а також дає потенційну можливість підвищити частоту тактів, що в свою чергу, прискорює обробку інформації.

Швидкодія, яка досягається за рахунок використання online режиму, а саме суміщення в часі введення обробки і виводу розрядів операндів, дозволяє виконувати операції паралельно на рівні обробки розрядів і є головною ціллю даної роботи.

Важливим фактором стало використання мови опису апаратури VHDL, перевагами якої є гнучкість проекту описаного нею, що дозволяє легко настроювати його під конкретні задачі; універсальність мови – загальноприйнятий стандарт для всіх основних фірм-розробників мікросхем

ПЛІС; моделювання з урахуванням всіх затримок, так як САПР забезпечують більш детальну генерацію моделей результату описаного цією мовою; єдиний стандартний формат опису апаратури, що дозволяє швидко і надійно поєднувати ОМ між собою; виключення можливого утворення помилок в кінцевому проекті за рахунок постійного тестування елементів одними і тими ж самими тестами на всіх етапах розробки. Слід зазначити, що всі сучасні САПР засновані на основі трансляції опису апаратури. А використання VHDL – це гарантія того, що через десятки років матиме місце САПР, яка буде підтримувати теперішні розробки.

Розроблюваний пристрій може бути використаний в цифрових обчислювальних машинах для множення чисел у формі з плаваючою комою. Головною перевагою такого пристрою є використання схеми множення двох чисел у формі з плаваючою комою в online режимі обчислень, що безсумнівно має неймовірну перевагу над прототипами для множення чисел з фіксованою комою у точності розрахунків – навіть незначні погрішності з плином часу та накопиченням результату можуть призвести до суттєвих помилок. В свою чергу, надійність пристрою заснована на порозрядній передачі даних для спрощення обміну інформації між внутрішніми елементами інтегральної схеми.

Саме такий варіант реалізації даного пристрою дозволяє поєднати в часі процес порозрядного введення операндів, що формуються поза системою, і їх обробку, що неабияк збільшить швидкодію та надійність пристрою, та зменшить ресурсозалежність порівняно з його аналогами. Також даний пристрій буде позбавлений проблеми переповнення під час розрахунків, задля збільшення точності, що в інших аналогах призводило до скорочення діапазону обчислень та значно погіршувало ефективність машини в цілому.

Використання порозрядного введення операндів або так званого online режиму обчислень дозволяє застосовувати набагато менше ресурсів однієї

ПЛІС. Беручи до уваги дані, отримані за допомогою моделювання, можна зробити висновок, що пристрій працює належним чином і реалізує вирішення більшості недоліків його прототипів: множення відбувається над числами з плаваючою комою, що без сумнівів збільшує діапазон допустимих значень результату та точність обчислень; збільшення розрядності результату, що веде за собою унеможливлення виникнення переповнень; підвищення надійності пристрою за рахунок спрощення комутації всередині ПЛІС та зменшення кількості потрібних ресурсів; використання методу «digit by digit», який полягає в порозрядному введенні операндів та їх обчисленні і неабияк збільшує швидкість пристрою в цілому.

Таким чином, отримані результати підтверджують ефективність застосування методів порозрядної обробки інформації для створення пристрою множення чисел з плаваючою комою поданих в надлишковій системі числення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Багаторозрядний пристрій множення чисел з плаваючою комою // Патент UA №19536, 25.12.1997. // Нагорний Л. Я., Сидорчук О. Д., Радкевич О. І.

2. Устройство для умножения // Патент SU №603989, 25.04.1978. //Жабин В. И., Корнейчук В. И., Кобзар С.П., Тарасенко В. П.

3. Floating point number processor for a digital computer // Patent US №3725649, 03.04.1973 // Alan J. Deerfield

4. Data processing system for multiplying and intergerizing floating point numbers // Patent US №3871578, 18.03.1975 // Adrianus J. Van De Goor, Leonard B. Hughes

5. Прикладна теорія цифрових автоматів: Навчальний посібник з грифом МОН України. // Жабін В.І., Клименко І. А., Ткаченко В. В., Жуков І.А. // Київ: Книжкове видавництво НАУ, 2009. – 359с.

6. Архітектура комп’ютера // Мельник А.О. // Луцьк: Волинська обласна друкарня, 2008. – 470с.

7. Повышение эффективности параллельных вычислений в потоковых системах // В.И.Жабин, В.В.Жабина // Вісник Національного технічного університету України "КПІ". Інформатика, управління та обчислювальна техніка.- 2013. - Вип. 59. - С 122-128. // Режим доступу: http://nbuv.gov.ua/UJRN/Vkpi_iuot_2013_59_22

8. Арифметичні основи проектування мікропроцесорних систем: навчальний посібник // Р.Н. Квєтний, П.П. Повідайко, М.М. Компанець, В.В. Гармаш, Я.А. Кулик // Вінниця: Вінницький національний технічний університет, 2017. – 111 с.

9. Вычисление полиномов в системах,реализованных на ПЛИС // В.И.Жабин, В.В.Жабина, М.А.Безгинский // Вісник Національного технічного університету України "КПІ". Інформатика, управління та обчислювальна техніка: Зб. наук. праць. -- Київ: ВЕК+. 2012

10. Некоторые машинные методы вычисления рациональных функций многих аргументов // В.И. Жабин, В.И.Корнейчук, В.П.Тарасенко // Автоматика и телемеханика. – 1977. – №12. – С. 145-154.
11. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования // Тарасов И.Е. // Москва: 2019, - 538с.
12. Реализация неавтономных вычислений в избыточных системах счисления на ПЛИС // Жабин В.И., Жабина В.В., Скориченко А.В. // Вісник Національного технічного університету України "КПІ". Інформатика, управління та обчислювальна техніка: ВЕК+. 2016
13. Совмещение зависимых операций на уровне обработки разрядов операндов. // И.А.Дичка, В.В.Жабина // Інститут проблем штучного інтелекту МОН України та НАН України: 649-654, 2008.
14. Комп'ютеризація спеціалізованих середовищ : навчальний посібник // Н. М. Бондіна, А. І. Поворознюк, О. М. Шеін // Національний технічний університет "Харківський політехнічний інститут". – Харків : НТУ "ХПІ", 2013. – 378 с.
15.Altera Documentations // Section I. Cyclone II Device Family Data Sheet https://www.intel.com/content/www/us/en/programmable/products/fpga/cyclone-series/cyclone-ii/support.html
16. Огляд FPGA [Електронний ресурс]. – SlideShare, 2013. – Режим доступу: http://www.slideshare.net/abhilash128/lec-23.
17. Проектирование на ПЛИС. Архитектура, средства и методы // К. Максфилд. – Москва: Издательский дом «Додэка-XXI», 2007, - 408 с.
18. Проектування комп'ютерних систем на основі мікросхем програмованої логіки // С. А. Іванець, Ю. О. Зубань, В. В. Казимир, В. В. Литвинов // Суми: Сумський державний університет, 2013. – 313 с.
19. Основы языка VHDL Издательство 3-е, дополнение // Бибило П.Н. // Москва: Издательство ЛКИ, 2007. - 328 с.

20. Системы автоматизированного проектирования фирмы Altera MAX + plus II и Quartus II. Краткое описание и самоучитель // Комолов Д.А., Мяльк Р.А., Зобенко А.А, Филиппов А.С. // Москва: РадиоСофт, 2002. — 352 с.:
21. Вычисление элементарных функций методом «цифра за цифрой» в избыточных системах счисления // Луцкий Г.М., Пенчев О.И. // Препринт АН УССР, Институт кибернетики; 83-22, Киев, 1983.30 с.
22. Floating-point on-line arithmetic: algorithms // Watanuki 0., Ercegovac M.D. // Proc. 5th. Sjmp. Comput. Arith., Ann Arbor, Mich., 1981, p. 81-86.
23. Арифметика цифровых машин. Учебное пособие // Карцев М.А. // Москва: Издательство "Наука", 1969. - 576 с.
24. VHDL для проектирования вычислительных устройств // Сергиенко А.М. // Киев: ЧП "Корнейчук", ТИД ДС, 2003 — 208 с.
25. Системы проектирования интегральных схем на основе языка VHDL. StateCAD. ModelSim. LeonardoSpectrum // П.Н. Бибило // Москва: СОЛОН-Пресс, 2005. — 384 с.

Додаток 1
до дипломного проекту
на тему: «Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення»

Київ – 2020 року

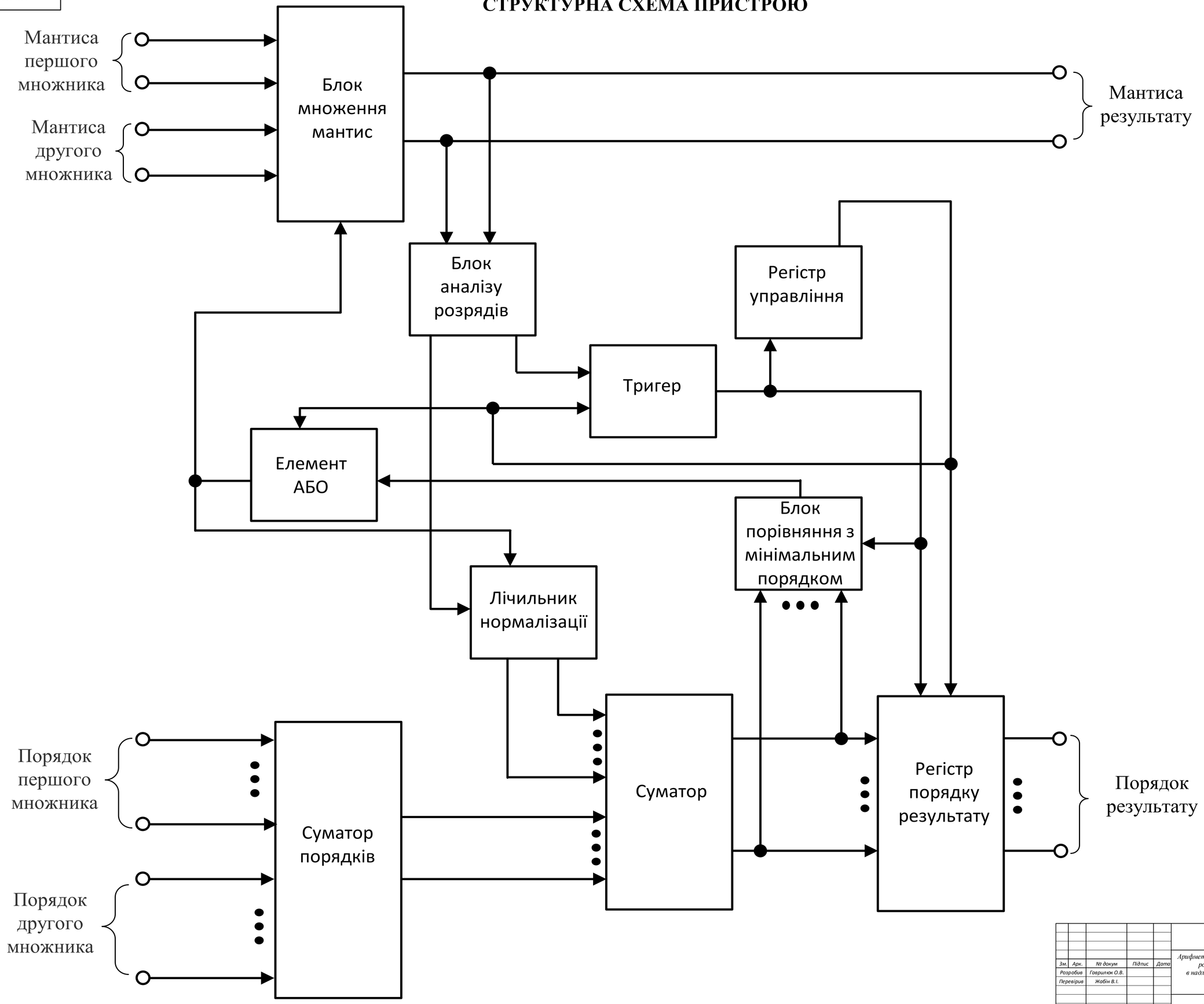
Додаток 2
до дипломного проекту
на тему: «Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення»

Київ – 2020 року

Додаток 3
до дипломного проекту
на тему: «Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення»

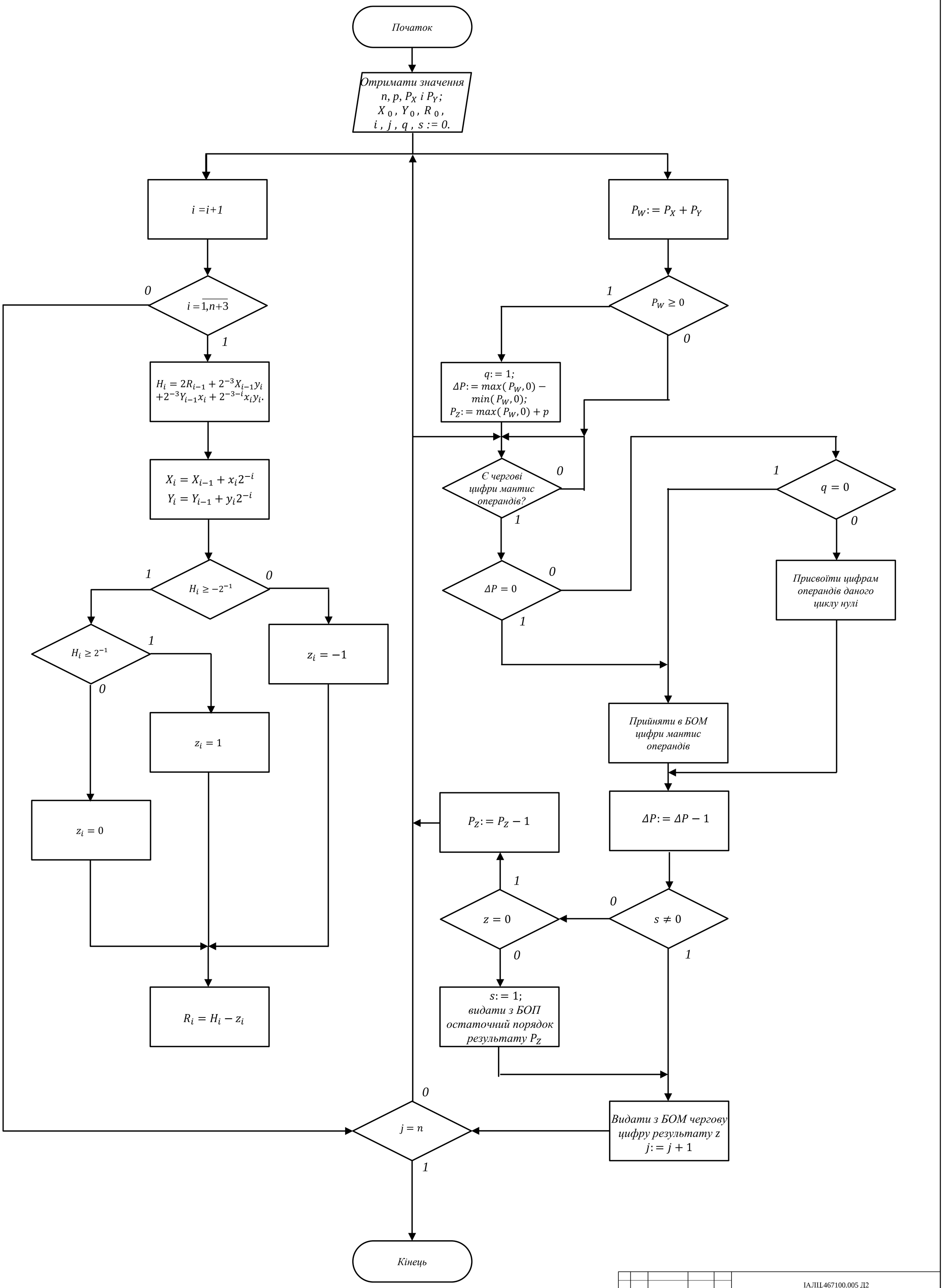
Київ – 2020 року

СТРУКТУРНА СХЕМА ПРИСТРОЮ



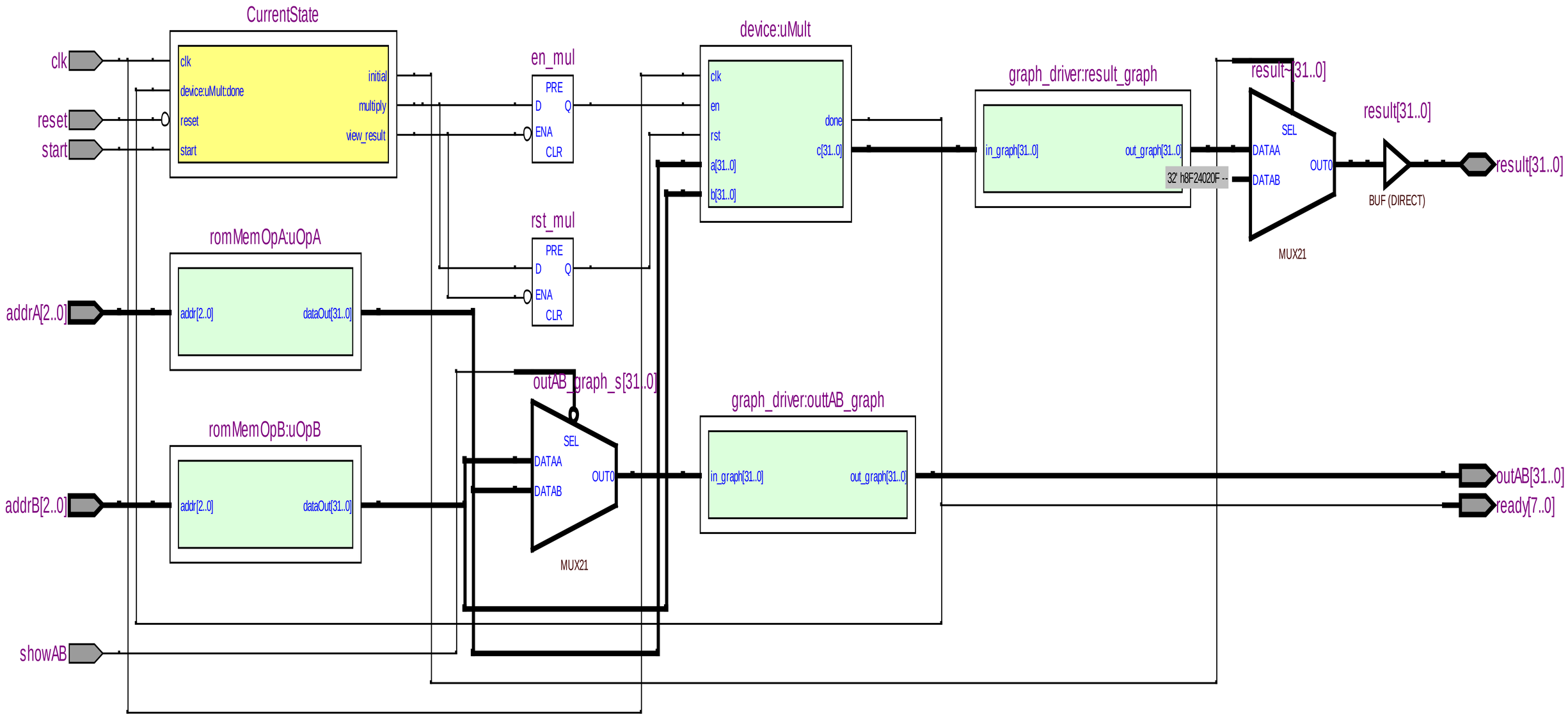
					ІАЛЦ.467100.004 Д1	
Зм.	Арк.	№ докум	Підпис	Дата	*Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення*	Літера / Маса / Масштаб
Розробив		Гаврилюк О.В.				
Перевірив		Жабін В.І.			*Додаток 1*	Аркуш 1 / Аркушів 1
Н. контр.		Симоненко В.П.			*Дипломний проект*	НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІО-64
Затверд.		Жабін В.І.				

БЛОК-СХЕМА АЛГОРИТМУ ОБРОБКИ ОПЕРАНДІВ



					ІАЛЦ.467100.005 Д2		
Зм.	Арк.	№ докум	Підпис	Дата	*Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення*	*Літера*	*Маса* / *Масштаб*
Розробив		*Гаврилюк О.В.*					
Перевірив		*Жабін В.І.*			*Додаток 2*	*Аркуш 1*	*Аркушів 1*
Н. контр.		*Симоненко В.П.*			*Дипломний проект*	*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІО-64*	
Затверд.		*Жабін В.І.*					

ФУНКЦІОНАЛЬНА СХЕМА ПРИСТРОЮ



						ІАЛЦ.467100.006 Д3	
Зм.	Арк.	№ докум	Підпис	Дата	Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення Додаток 3	Літера	Маса / Масштаб
Розробив		Гаврилюк О.В.					
Перевірив		Жабін В.І.				Аркуш 1	Аркушів 1
Н. контр.		Симоненко В.П.			Дипломний проект	НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІО-64	
Затверд.		Жабін В.І.					

Додаток 4
до дипломного проекту
на тему: «Арифметичний пристрій на ПЛІС для роботи в online режимі в надлишковій системі числення»

Київ – 2020 року

ЛІСТИНГ ПРОГРАМИ

multiplier.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity multiplier is
  port (clk   : in  std_logic;
        reset : in  std_logic;
        addrA : in  std_logic_vector(2  downto 0);
        addrB : in  std_logic_vector(2  downto 0);
        showAB: in  std_logic;
        start: in  std_logic;
        result: inout std_logic_vector(31 downto 0);
        outAB : out std_logic_vector(31 downto 0);
        ready : out std_logic_vector(7  downto 0)
  );
end multiplier;

architecture synt of multiplier is

  signal romAOut : std_logic_vector (31 downto 0);
  signal romBOut : std_logic_vector (31 downto 0);

  component graph_driver
     port (in_graph : in  std_logic_vector(31  downto 0);
               out_graph  : out  std_logic_vector(31  downto 0)
     );
     end component;
     signal outAB_graph_s: std_logic_vector(31 downto 0);
     signal result_graph_s: std_logic_vector(31 downto 0);

  component romMemOpA
    port(
      addr    : in  std_logic_vector (2  downto 0);
      dataOut : out std_logic_vector (31 downto 0)
    );
  end component;

  component romMemOpB
    port(
      addr    : in std_logic_vector  (2  downto 0);
      dataOut : out std_logic_vector (31 downto 0)
    );
  end component;
```

```
  component device
     port (a,b: in std_logic_vector(31 downto 0);
                   clk,en,rst: in std_logic;
                   c: out std_logic_vector(31 downto 0);
                   done: out std_logic
                   );
  end component;

  signal en_mul:std_logic;
  signal rst_mul:std_logic;
  signal result_graph_ss: std_logic_vector(31 downto 0);
   signal res: std_logic_vector(31 downto 0);
  signal done_mult: std_logic;
  signal graph: std_logic_vector(31 downto 0);
  type state is (initial, multiply, view_result);
  signal CurrentState, NextState : state;

begin
  process(CurrentState) begin
    NextState<=initial;
      result_graph_s<=(others=>'0');

    case CurrentState is
      when initial =>
        en_mul<='0';
        rst_mul<='0';
        if(start='0') then
          NextState<=multiply;
        else

          NextState<=initial;
        end if;

      when multiply =>
        en_mul<='1';
        rst_mul<='1';
        if(done_mult='1') then
          NextState<=view_result;
        else
          NextState<=multiply;
        end if;

      when view_result =>
        --en_mul<='1';
        --if(reset='0') then
          --NextState<=initial;
        --else
          --NextState<=view_result;
        --end if;
            NextState<=multiply;

    end case;
```

```
  end process;

  transitions:process (clk, reset) begin
    if reset='0'then
      CurrentState <= initial;
    elsif (clk'event and clk='1')then
      CurrentState <= NextState;
    end if;
  end process;

  uOpA: romMemOpA
    port map (
      addr    => addrA,
      dataOut => romAOut
    );

  uOpB: romMemOpB
    port map (
      addr    => addrB,
      dataOut => romBOut
    );

  uMult: device
    port map (
      a    => romAOut,
      b    => romBOut,
      clk=>clk ,
      en=> en_mul,
      rst=> rst_mul,
      c =>  result_graph_ss,
      done=>done_mult
    );

      outtAB_graph: graph_driver port
map(in_graph=>outAB_graph_s,out_graph=>outAB);
      result_graph: graph_driver port
map(in_graph=>result_graph_ss,out_graph=>res);
      with  currentState select
      result<="11110000010000000010010011110001" when initial,
    res when others ;


  -- When the button assoiciated to outAB is pushed romAOut is
displayed
   outAB_graph_s<= romAOut when showAB = '0'
                   else romBOut;

  ready(7 downto 0) <= (others => done_mult);

end synt;
```

mult_int1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mul_int1 is
     port (in1: in std_logic_vector(23 downto 0);
           in2: in std_logic_vector(23 downto 0);
                clk,rst: in std_logic;
                done:out std_logic;
                res: out std_logic_vector(47 downto
0):=(others=>'0')
                );
end mul_int1;

architecture arch_mul_int_1 of mul_int1 is

  component shifter
    port (in1: in std_logic_vector(23 downto 0);
          in2: in unsigned(4 downto 0);
          clk,rst: in std_logic;
          res: out std_logic_vector (47 downto 0)
          );
  end component;

    type RAM is array (23 downto 0) of std_logic_vector(47
downto 0);
    signal out_shifters: RAM;
    signal out_mux: RAM;

    signal r:std_logic_vector(47 downto 0 ):=(others=>'0');

      signal r1:unsigned(47 downto 0 ):=(others=>'0');
      signal r2:unsigned(47 downto 0 ):=(others=>'0');
      signal r3:unsigned(47 downto 0 ):=(others=>'0');
      signal r4:unsigned(47 downto 0 ):=(others=>'0');
      signal r5:unsigned(47 downto 0 ):=(others=>'0');
      signal r6:unsigned(47 downto 0 ):=(others=>'0');
      signal r7:unsigned(47 downto 0 ):=(others=>'0');
      signal r8:unsigned(47 downto 0 ):=(others=>'0');

      signal done1:std_logic:='0';

  begin

    res<=r;


r1<=unsigned(out_mux(0))+unsigned(out_mux(1))+unsigned(out_mux(2
))+unsigned(out_mux(3));
```

```
r2<=unsigned(out_mux(4))+unsigned(out_mux(5))+unsigned(out_mux(6
))+unsigned(out_mux(7));

r3<=unsigned(out_mux(8))+unsigned(out_mux(9))+unsigned(out_mux(1
0))+unsigned(out_mux(11));

r4<=unsigned(out_mux(12))+unsigned(out_mux(13))+unsigned(out_mux
(14))+unsigned(out_mux(15));

r5<=unsigned(out_mux(16))+unsigned(out_mux(17))+unsigned(out_mux
(18))+unsigned(out_mux(19));

r6<=unsigned(out_mux(20))+unsigned(out_mux(21))+unsigned(out_mux
(22))+unsigned(out_mux(23));
      r7<=r1+r2+r3;
      r8<=r4+r5+r6;

    SH1 : for N in 23 downto 0 generate
      shifter_array : shifter
        port map
(in1=>in2,in2=>to_unsigned(N,5),clk=>clk,rst=>rst,res=>out_shift
ers(N));
    end generate SH1;

    MS : for M in 23 downto 0 generate
          with in1(M) select
               out_mux(M)<=out_shifters(M) when '1',
                                   (others=>'0') when others;
    end generate MS;

     syncrho:process (clk, rst)
     begin
          if rst='0' then
               r <= (others => '0');
               done1<='0';
               done<='0';
          else
               if (rising_edge(clk)) then
                    done1<='1';
                    done<=done1;
                    if (in1="10000000000000000000000000" or
in2="10000000000000000000000000")then
                      r<=(others => '0');
                    else
                      r<=std_logic_vector(r7+r8);
                    end if;
               end if;
          end if;
     end process;

end arch_mul_int_1;
```

add.sub.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity add_sub is
  port (

  add_sub_in_a:in std_logic_vector(7 downto 0 );
  add_sub_in_b:in std_logic_vector(7 downto 0 );
  add_sub_out:out std_logic_vector(7 downto 0 );
  add_sub_sel:in std_logic

);

end add_sub ;
---// when the sel is ONE = Addition when select zero =
subtraction \\---
architecture behaviour of add_sub is
begin
  process (add_sub_in_a,add_sub_in_b,add_sub_sel)
    begin
      if (add_sub_sel='1')then
      add_sub_out<=add_sub_in_a+add_sub_in_b;
    else
    add_sub_out<=add_sub_in_a-add_sub_in_b;
  end if ;
end process;
end  behaviour;
```

control.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity control is
     port (clk,rst,en: in std_logic;
               done_m: in std_logic;
               done:out std_logic;
               en_mul: out std_logic_vector(1 downto 0)
               );
end control;

architecture arch_control of control is

  type states is (idle,load,multiply,deploy);
  signal State, NextState : states;
```

```
  begin

      FSM:process(State) begin
    NextState<=idle;
      done<='0';
      en_mul<="00";

    case State is
      when idle =>
            done<='0';
            en_mul<="00";
        if(en='1') then
          NextState<=load;
        else
          NextState<=idle;
        end if;

          when load =>
        NextState<=multiply;
            done<='0';
            en_mul<="01";

          when multiply =>
            done<='0';
            en_mul<="11";
        if(done_m='1') then
          NextState<=deploy;
        else
          NextState<=multiply;
        end if;

          when deploy =>
            done<='1';
            en_mul<="11";
          NextState<=idle;
    end case;
  end process;


  transitions:process (clk, rst) begin
    if rst='0'then
      State <= idle;
    elsif (clk'event and clk='1')then
      State <= NextState;
    end if;
  end process;

end arch_control;
```

datapath.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity datapath is
     port (a,b: in std_logic_vector(31 downto 0);
               clk,rst: in std_logic;
               en: in std_logic_vector(1 downto 0);
               c: out std_logic_vector(31 downto 0);
               done,m47: out std_logic
               );
end datapath;

architecture arch_datapath_1 of datapath is
component reg is
     port (clk,en,rst: in std_logic;
                    a:in std_logic_vector((31) downto 0);
                    r: out std_logic_vector((31) downto 0)
                    );
end component;

component mul_int1 is
     port (in1: in std_logic_vector(23 downto 0);
           in2: in std_logic_vector(23 downto 0);
                    clk,rst: in std_logic;
                    done:out std_logic;
                    res: out std_logic_vector(47 downto
0):=(others=>'0')
                    );
end component;

component extractor is
  port (

  ext_in:in std_logic_vector(47 downto 0 );
  ext_out:out std_logic_vector(22 downto 0 )

);
end component;

signal rra,rrb: std_logic_vector(31 downto 0);
alias signa: std_logic is rra(31);
alias signb: std_logic is rrb(31);
alias expa: std_logic_vector(7 downto 0) is rra(30 downto 23);
alias expb: std_logic_vector(7 downto 0) is rrb(30 downto 23);
alias manta: std_logic_vector(22 downto 0) is rra(22 downto 0);
alias mantb: std_logic_vector(22 downto 0) is rrb(22 downto 0);
signal mana,manb: std_logic_vector(23 downto 0);
```

```
signal done_mul: std_logic;
signal mul_out: std_logic_vector(47 downto 0);
signal ext_out: std_logic_vector(22 downto 0);
signal signf: std_logic;
signal expf:std_logic_vector(7 downto 0):=(others=>'0');
signal expg:std_logic_vector(7 downto 0):=(others=>'0');
signal exph:std_logic_vector(7 downto 0):=(others=>'0');

signal ric: std_logic_vector(31 downto 0);

     begin
   mana<='1' & manta;
   manb<='1' & mantb;
   rega: reg port map(clk=>clk,en=>en(0),rst=>rst,a=>a,r=>rra);
     regb: reg port map(clk=>clk,en=>en(0),rst=>rst,a=>b,r=>rrb);

     mult: mul_int1 port
map(in1=>mana,in2=>manb,clk=>clk,rst=>rst,done=>done_mul,res=>mu
l_out);

     ext: extractor port map(ext_in=>mul_out,ext_out=>ext_out);

     signf<=signa xor signb;

     expg<=std_logic_vector(expa+expb);
     exph<=std_logic_vector(expg)-"01111111";
     expf<=exph+("0000000" & mul_out(47));

     ric<=signf & expf & ext_out;
     regc: reg port map(clk=>clk,en=>en(1),rst=>rst,a=>ric,r=>c);

     done<=done_mul;
     m47<=mul_out(47);

end arch_datapath_1;
```

device.vhd

```
library ieee;
use ieee.std_logic_1164.all;


entity device is
     port (a,b: in std_logic_vector(31 downto 0);
                    clk,en,rst: in std_logic;
                    c: out std_logic_vector(31 downto 0);
                    done: out std_logic
                    );
end device;
```

```
architecture arch_device_1 of device is

component datapath is
     port (a,b: in std_logic_vector(31 downto 0);
               clk,rst: in std_logic;
               en: in std_logic_vector(1 downto 0);
               c: out std_logic_vector(31 downto 0);
               done,m47: out std_logic
               );
end component;
component exceptions
     port (in1: in std_logic_vector(31 downto 0);
           in2: in std_logic_vector(31 downto 0);
               int_mul: in std_logic_vector(31 downto 0);
               enable: in std_logic;
               m_in47:in std_logic;
               exp_out: out std_logic_vector(31 downto 0)

               );
end component;
component control
     port (clk,rst,en: in std_logic;
               done_m: in std_logic;
               done:out std_logic;
               en_mul: out std_logic_vector(1 downto 0)
               );
end component;

signal done_s : std_logic ;
signal res: std_logic_vector(31 downto 0);
signal m47: std_logic;
signal en_m:std_logic_vector(1 downto 0);
signal d: std_logic;
begin
  done<=d;
  dp: datapath port
map(a=>a,b=>b,clk=>clk,rst=>rst,en=>en_m,c=>res,done=>done_s,m47
=>m47);
  ex:exceptions port
map(in1=>a,in2=>b,int_mul=>res,enable=>d,m_in47=>m47,exp_out=>c)
;
  ctrl: control port
map(clk=>clk,rst=>rst,en=>en,done_m=>done_s,done=>d,en_mul=>en_m
);

end arch_device_1;
```

toplevel.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity multiplier is
  port (clk   : in  std_logic;
        reset : in  std_logic;
        addrA : in  std_logic_vector(2  downto 0);
        addrB : in  std_logic_vector(2  downto 0);
        showAB: in  std_logic;
        start: in  std_logic;
        result: out std_logic_vector(31 downto 0);
        outAB : out std_logic_vector(31 downto 0);
        ready : out std_logic_vector(7  downto 0)
  );
end multiplier;

architecture synt of multiplier is

  signal romAOut : std_logic_vector (31 downto 0);
  signal romBOut : std_logic_vector (31 downto 0);

  component graph_driver
     port (in_graph : in  std_logic_vector(31  downto 0);
               out_graph  : out  std_logic_vector(31  downto 0)
     );
     end component;
     signal outAB_graph_s: std_logic_vector(31 downto 0);
     signal result_graph_s: std_logic_vector(31 downto 0);

  component romMemOpA
    port(
      addr    : in  std_logic_vector (2  downto 0);
      dataOut : out std_logic_vector (31 downto 0)
    );
  end component;

  component romMemOpB
    port(
      addr    : in std_logic_vector  (2  downto 0);
      dataOut : out std_logic_vector (31 downto 0)
    );
  end component;

  component device
     port (a,b: in std_logic_vector(31 downto 0);
                   clk,en,rst: in std_logic;
                   c: out std_logic_vector(31 downto 0);
                   done: out std_logic
```

```
                    );
  end component;

  signal en_mul:std_logic;
  signal rst_mul:std_logic;
  signal result_graph_ss: std_logic_vector(31 downto 0);
  signal done_mult: std_logic;
  signal graph: std_logic_vector(31 downto 0);
  type state is (initial, multiply, view_result);
  signal CurrentState, NextState : state;

begin
  process(CurrentState,start) begin
    NextState<=initial;
      result_graph_s<=(others=>'0');

    case CurrentState is
      when initial =>
        en_mul<='0';
        rst_mul<='0';
            -----result<="11110000010000000010010011110001";
        if(start='0') then
          NextState<=multiply;
        else
          NextState<=initial;
        end if;

      when multiply =>
        en_mul<='1';
        rst_mul<='1';
        if(done_mult='1') then
          NextState<=view_result;
        else
          NextState<=multiply;
        end if;

      when view_result =>

        if(start='0') then
          NextState<=initial;
        else
          NextState<=view_result;
        end if;


    end case;
  end process;

  transitions:process (clk, reset) begin
    if reset='0'then
      CurrentState <= initial;
    elsif (clk'event and clk='1')then
```

```
      CurrentState <= NextState;
    end if;
  end process;

  uOpA: romMemOpA
    port map (
      addr    => addrA,
      dataOut => romAOut
    );

  uOpB: romMemOpB
    port map (
      addr    => addrB,
      dataOut => romBOut
    );

  uMult: device
    port map (
      a    => romAOut,
      b    => romBOut,
      clk=>clk ,
      en=> en_mul,
      rst=> rst_mul,
      c =>  result_graph_ss,
      done=>done_mult
    );

      outtAB_graph: graph_driver port
map(in_graph=>outAB_graph_s,out_graph=>outAB);
      result_graph: graph_driver port
map(in_graph=>result_graph_ss,out_graph=>result);



  -- When the button assoiciated to outAB is pushed romAOut is
displayed
   outAB_graph_s<= romAOut when showAB = '0'
                   else romBOut;


  ready(7 downto 0) <= (others => done_mult);

end synt;
```