НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кваліфікаційна наукова праця
на правах рукопису

**Борецький Тарас Романович**

УДК 004.272:004.315.7:004.42

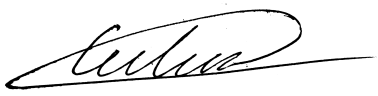
**Д И С Е Р Т А Ц І Я**

# РОЗРОБКА ТА РЕАЛІЗАЦІЯ МЕТОДІВ ОБЧИСЛЕННЯ ЕЛЕМЕНТАРНИХ ФУНКЦІЙ НА ОСНОВІ ПРОГРАМНИХ ТА АПАРАТНИХ ЗАСОБІВ

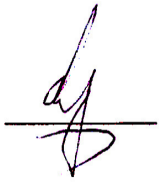
05.13.05 – Комп’ютерні системи та компоненти
05 – Технічні науки

Подається на здобуття ступеня кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело


/Т. Р. Борецький/

Науковий керівник:
Мороз Леонід Васильович
доктор технічних наук, доцент

Ідентичність всіх примірників дисертації ЗАСВІДЧУЮ:
Вчений секретар спеціалізованої вченої ради Д 35.052.08 
/Я.Т. Луцик/

Львів — 2019

# АНОТАЦІЯ

*Борецький Т. Р.* Розробка та реалізація методів обчислення елементарних функцій на основі програмних та апаратних засобів. - Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук (доктора філософії) за спеціальністю 05.13.05 «Комп’ютерні системи та компоненти» (122 - Комп’ютерні науки). - Національний університет “Львівська політехніка”, Львів, 2019.

Дисертаційна робота присвячена удосконаленню методів обчислення елементарних функцій, які на сьогоднішній день використовуються переважно у цифрових апаратних засобах напівпровідникової техніки. Основні завдання в дослідженні полягали у підвищенні швидкодії алгоритмів шляхом оптимізації використовуваних структур для досягнення нижчих значень латентності (і, відповідно, кількості тактів), а також покращення показників ресурсоємності при реалізації алгоритмів за допомогою апаратних засобів. Запропоновані методи були реалізовані як на програмному рівні для процесорних архітектур різного рівня складності, так і з допомогою програмованих логічних інтегральних схем, що, в свою чергу, дозволяє здійснити портування розроблених методів у інтегральну схему.

У вступі відображено актуальність проблеми досліджень, показано зв’язок обраного напрямку з науковими програмами, планами, темами, сформульовано мету та основні задачі досліджень, подано наукову новизну і практичне значення отриманих результатів, зазначено особистий внесок здобувача, наведено дані про апробацію, публікації за темою роботи та використання результатів дослідження.

У першому розділі розглянуто класичні способи обчислення поширених функцій - експоненціальних логарифмічних тригонометричних гіперболічних та показникових. Зазначено переваги та недоліки застосування традиційних підходів. Проаналізовані прийоми, які використовуються для наближення функцій, таких, як розклад у степеневий ряд, поліноміальна апроксимація, дробово-раціональна

апроксимація, наближення ланцюговим дробом, ітераційні методи. Подано огляд основних способів реалізації обчислень вищезгаданих функцій з використанням відомих на даний момент прийомів для їх удосконалення. Виділені сильні та слабкі сторони кожного із методів, враховуючи їхні діапазони обчислень, очікувану швидкодію та апаратні затрати. Також наведено зіставлення ефективності використання того чи іншого методу, виходячи з пріоритетів, які виникають в даного роду задачах, а також опису переваг та недоліків, що виникають при подальшому їх застосуванні. Здійснено огляд середовищ розробки, які застосовувались у дослідженні, а також опис програмних та апаратних засобів, використаних для реалізації пропонованих методів.

У другому розділі дисертації описані пропоновані методи, способи їх одержання та представлення у вигляді схем та графіків. Показано переваги та слабкі місця методів, що виникають чи можуть виникнути в тій чи іншій конфігурації кінцевого обладнання чи готового продукту. Подано способи, за допомогою яких стає можливим прискорити або спростити існуючі методи обчислень, підвищивши їх швидкодію чи зменшивши латентність та ресурсоємність. Описано процес виведення залежностей та формул. Наведено блок-схеми алгоритмів, гістограми розподілу результатів. Запропоновані нові підходи та шляхи обчислення функцій, наведені їх логічні схеми та здійснено зіставлення точності обчислень залежно від складності реалізації вибраного методу. Значну увагу приділено методу CORDIC, за допомогою якого можна обчислювати такі функції, як: синус, косинус, тангенс, експоненту, квадратний корінь, гіперболічні та обернені тригонометричні функції. Враховуючи, що найбільший інтерес із представлених функцій становить обчислення синуса та косинуса, під час практичної реалізації методів залежно від контексту поставленої задачі для демонстрації роботи використовуються саме ці функції. При використанні класичних методів точність обчислень залежить від розрядності операндів та кількості здійснених ітерацій. При невеликих кількостях ітерацій всі вхідні та вихідні значення можна задавати у вигляді таблиці, розміщеної в пам’яті.

Враховуючи, що у переважній більшості пристроїв доступним є деякий, хоч і незначний обсяг пам’яті, який можна використати для прискорення роботи більшості методів. Причому ефект буде відчутним і у випадку, якщо обсяг виділеної пам’яті становить сотні чи навіть десятки байт (оперативної чи флеш пам’яті у випадку використання мікроконтролера). Автором запропоновано нові методи знакозмінного та беззнакового перекодування вхідного аргументу, які дають змогу гнучко змінювати обсяг пам’яті, число ітерацій і, відповідно, апаратні затрати.

У третьому розділі здійснюється апаратна реалізація розглянутих методів на платформах ПЛІС. Аналізується доступна на сьогодні апаратна база для їх імплементації від виробників Intel (Altera) та Xilinx. Розглянуті особливості вибраних ПЛІС, специфіка апаратної реалізації алгоритмів, зокрема, вплив архітектури та інтегрованих блоків ПЛІС на функціональність реалізованих методів. Наведені схеми пропонованих алгоритмів на рівні регістрових передач та вентильного рівня на етапі їх розміщення (фітінгу) в кристалі. Розглянуто способи оптимізації алгоритмів в умовах використання конкретної платформи та залежно від версії та налаштувань середовищ розробки. Наведені результати імплементації методів безпосередньо в ПЛІС з оцінкою їх вихідних характеристик, таких, як максимальна тактова частота, ресурсоємність та енергозатратність. Верифікація коректності функціонування алгоритмів здійснюється як за допомогою імітаційного моделювання, так і після заміру фізичних показників та даних, отриманих в процесі тестування запрограмованої ПЛІС. Значну увагу приділено модифікованому методу знакозмінного перекодування - беззнакове перекодування, в якому всі аргументи приймають лише додатні величини, що дозволяє спростити реалізацію алгоритму та необхідні для цього апаратні ресурси зі збереженням переваг підходу перекодування. Основним параметром, що демонструє переваги пропонованого алгоритму, є латентність, яка оберненопропорційна тактовій частоті та залежить від кількості тактів, необхідних для обчислення функції. Кількість тактів, у свою чергу, може змінюватись завдяки змінам розміру

виділеної пам’яті. Порівняння результатів методу перекодування для платформи Xilinx здійснено із пропонованою виробником імплементацією IP бібліотеки CORIDC. Для платформи Intel порівняння здійснювалось із вбудованою мегафункцією.

Четвертий розділ присвячений програмній реалізації алгоритмів як для платформ із обмеженими ресурсами, так і для універсальних процесорних архітектур з складною системою команд. Програмна реалізація класичних та пропонованих алгоритмів здійснюється мовами C та асемблера (Intel, AT&T) для платформ x86-64 та мікроконтролерів Atmel AVR. Значна увага приділяється визначенню ефективності функціонування алгоритмів, а також точності їхніх обчислень. Досліджена специфіка функціонування методів залежно від арсеналу команд та можливостей цільової архітектури, за допомогою яких вдалось покращити ряд характеристик як самих алгоритмів, так і їх апаратних імплементацій. Отримані експериментальні результати зіставлені з класичними способами реалізації наведених функцій. Здійснена інтеграція розроблених алгоритмів у вигляді окремого пристрою. Розроблені алгоритми оптимізовано для досягнення максимальної швидкодії та найбільш повного використання ресурсів мікроконтролера. Організовано комунікацію між МК та ПК, використовуючи IEEE 1284 та режим “Byte Mode” для двосторонньої передачі даних, що дає змогу використовувати МК як додатковий периферійний пристрій комп’ютера та перекласти на нього частину задач. Отриманий у підсумку генератор ПВП функціонує на частотах, що відповідають стандарту IEC 60908, і, при потребі, може бути підключений до ЦАП зі шиною у 32 розряди. Таким чином, реалізовані алгоритми обчислення більшості елементарних функцій як для платформи з обмеженими ресурсами (AVR), так і для x86-64 мікропроцесорів із використанням цілочисельної логіки та доступного широкого набору команд. Даний факт дає змогу стверджувати, що запропоновані методи будуть працювати на більшості сучасних платформ.

Ключові слова: ітераційні методи, елементарні функції, тригонометричні функції, мікроконтролер, програмовані логічні інтегральні схеми, цифровий обчислювач повороту системи координат.

## ANNOTATION

*Boretskyy T. R.* Development and implementation of methods for elementary functions calculation on the basis of software and hardware. - Qualification scientific work on the rights of the manuscript.

A thesis submitted in fulfilment of the candidate of sciences (Ph.D.) degree in technical sciences on specialty 05.13.05 «Information technologies» (122 – Computer Sciences). - Lviv Polytechnic National University, Lviv, Ukraine, 2019.

The dissertation is devoted to the improvement of methods for calculating elementary functions, which today are used mainly in semiconductor equipment. The main tasks in the study were to increase the speed of algorithms by optimizing the structures used to achieve lower latency (and, consequently, the number of cycles), as well as improving the resource-intensity indices when implementing algorithms using hardware. The proposed methods were implemented at the software level for processor architectures of different complexity levels, and with the help of programmable logic integrated circuits, which allows the porting of the developed methods to the system on a chip.

The introduction reflects the relevance of the research problem, shows the connection of the chosen direction with the scientific programs, plans, themes, formulates the purpose and main objectives of the research, presents the scientific novelty and the practical value of the results obtained, indicates the personal contribution of the applicant, provides data on approbation, publications on the topic work and use of research results.

In the first section, we consider classical methods for calculating common functions — exponential, logarithmic, trigonometric and hyperbolic functions. The

advantages and disadvantages of applying traditional approaches are highlighted. The methods are used to approximate functions such as the expansion in a power series, polynomial approximation, fractional-rational approximation, approximation by a chain fraction, iterative methods. An overview of the main methods for calculating the above-mentioned functions with the use of known ways for their improvement. The strengths and weaknesses of each method are distinguished, taking into account their ranges of computing, expected performance and hardware costs. Also, the comparison of the effectiveness of using one method or another based on the priorities that arise in this kind of tasks, as well as a description of the advantages and disadvantages arising from their subsequent application. An overview of the development frameworks that were used in the investigations, as well as description of software and hardware, which will be used to further implement the proposed methods.

The second section of the dissertation presents the proposed methods, methods of obtaining and presenting them in the schematic and graphic view. The advantages and weaknesses of the methods that appear or may arise in one or another configuration of the final equipment or in the final product are shown. The ways in which it is possible to accelerate or simplify existing computational methods, increase their speed, or reduce latency and resource intensity is represented. The process of deducing dependencies and formulas is described. The block diagrams of algorithms, results distribution gistographs are presented. New approaches and ways of calculation of functions are offered, their logic diagrams are represented and the comparison of the accuracy of calculations is carried out depending on the complexity of the implementation of the chosen method. Considerable attention is paid to the CORDIC method by which one it is possible to compute functions such as sinus, cosine, tangent, exponent, square root, hyperbolic and inverse trigonometric functions. Taking into account that the great interest in the presented functions is the calculation of sinus and cosine in the practical implementation of methods depending on the context of the problem, the functions used to demonstrate the work are done. When we using classical methods, the accuracy of calculations depends on the number of operands and the number of iterations

performed. In small quantities of iterations, all input and output values can be set in the form of a table placed in memory. Given that, for the vast majority of devices, some memory, though insignificant, is available, can be used to accelerate the CORDIC method. Moreover, the effect will be noticeable if the volume of allocated memory is hundreds or even dozens of bytes (operating or flash memory in the case of microcontroller). The author proposes a new method for the conversion of the input argument into an alternate, which allows flexible change of the memory for the look-up table and the number of iterations.

In the third section the hardware implementation of the considered methods on the FPGA platforms is carried out. Analyzed the available hardware base for their implementation from Intel (Altera) and Xilinx manufacturers. The features of selected FPGAs, the specifics of the hardware implementation of algorithms, in particular, the influence of architecture and integrated FPGA blocks on the functionality of the implemented methods are considered. The schemes of the offered algorithms on the level of register gears and the low level at the stage of their placement (fitting) in the crystal are given. The methods of optimization of algorithms in terms of use a specific platform and depending on the version and settings of the development environment are considered. The results of impetence of methods directly in the FPGA with the estimation of their initial characteristics, such as the maximum clock frequency, resource intensity and energy consumption are given. The verification of correctness of algorithms functioning is carried out both by means of simulation modeling, and after measurement of physical indicators and data obtained during the testing of the programmed FPGA. Considerable attention is paid to the modified method of alternating transcoding - unsigned transcoding, in which all arguments take only positive values, allow simplification of the implementation of the algorithm, and necessary hardware resources for this, while preserving the advantages of the transcoding approach. The main parameter demonstrating the advantages of the proposed algorithm is the latency, which is inversely proportional to the clock frequency, and depends on the number of steps required to calculate the function. The

number of cycles, in turn, may change due to the change in the size of the allocated memory. Comparison of the results of the undocumented conversion method for the Xilinx platform was carried out with the proposed implementation by the CORIDC IP library. For the Intel prototype, a comparison was made with the built-in megafunction.

The fourth section is devoted to programmatic algorithms for both low-cost platforms and for more complex CISC processor architecture. The program implements classical and proposed algorithms in C languages and assembler (Intel, AT&T) for x86-64 platforms and Atmel AVR microcontrollers. Much attention is paid to determining the effectiveness of algorithms, as well as the accuracy of the abstracts and the errors they make. The specificity of the methods functioning depending on the possibilities of the target architecture, which helped to improve the range of characteristics of both the algorithms themselves and their hardware implementations. The obtained experimental results are compared with the classical methods of realization of the reduced functions. The integration of developed algorithms in the form of a separate device is carried out. The developed algorithms are optimized to achieve maximum performance and the most complete use of microcontroller resources. The communication between the microrontroller and PC using IEEE 1284 and the "Byte Mode" for two-way communication, allows to use the microcontroller as an additional peripheral device for PC and transfer some tasks to it. The resulting pseudorandomness generator operates at frequencies corresponding to IEC 60908 and, can be connected to a DAC with a 32-bit bus. Thus algorithms for calculating most elementary functions for wide range of platforms. From the limited capability platforms (like AVR) and to x86-64 microprocessors using integer logic and an accessible wide set of commands are implemented. This fact suggests that the proposed methods will work on most modern platforms.

Keywords: iteration methods, elementary functions, trigonometric functions, microcontrollers, field programmable gate array, coordinate rotation digital computer.

## Список публікацій здобувача

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Борецький Т., Мороз Л. Реалізація класичного та адаптивного CORDIC-методу на платформі IA32 : збірник наукових праць Української академії друкарства. Сер. Комп'ютерні технології друкарства. Львів, 2012. № 28. С. 130–140.

2. Борецький Т. Модифікований CORDIC-метод обчислення синуса-косинуса / Л. Мороз, Т. Борецький, Т. Луковський, С. Войтусік : збірник наукових праць Української академії друкарства. Сер. Комп'ютерні технології друкарства. Львів, 2015. № 33. С. 56–63.

3. Борецький Т. Швидкодіючий гібридний CORDIC-обчислювач тригонометрич-них функцій / Л. В. Мороз, Я. І. Грабовський, Т. М. Микитів, Т. Р. Борецький, Ю. М. Костів, С. С. Войтусік. *Науковий вісник НЛТУ України* : зб. наук.-техн. пр. Львів, 2014. Вип. 24.8. С. 352–358.

4. Борецький Т. Удосконалення методу CORDIC для обчислення тригонометричних функцій засобами програмованої логічної інтегральної схеми / Л. В. Мороз, Т. Р. Борецький, М. М. Сколоздра. *Науковий вісник НЛТУ України* : зб. наук.-техн. пр. Львів, 2015. Вип. 25.5. С. 292–301.

5. Борецький Т. Синус-косинусний FPGA-обчислювач на основі CORDIC-методу з перекодуванням кута / Л. В. Мороз, Т. Р. Борецький, Ю. М. Костів Науковий вісник НЛТУ України : зб.наук.-техн. пр. Львів, 2015. Вип. 25.6. С. 288–297.

6. Boretskyy T. Simple hybrid scaling-free CORDIC solution for FPGAs / L. Moroz, S. Nagayama, T. Mykytiv, I. Kirenko, T. Boretskyy // International Journal of Reconfigurable Computing. - 2014. Vol. - 2014. - pp. 1–4.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

7. Борецький Т., Мороз Л. Генерація випадкових чисел засобами ПЛІС.

*Захист інформації і безпека інформаційних систем* : матеріали V Міжнародної науково-технічної конференції. Львів, 2016.

8. Борецький Т., Микитів Т., Мороз Л. Покращення характеристик генератора псевдовипадкових послідовностей на основі тригонометричних функцій. *Інформаційна безпека в сучасному суспільстві* : матеріали I Міжнародної науково-технічної конференції. Львів, 2014. С.17-19.

9. Борецький Т., Мороз Л. Альтернативні методи обчислення тригонометричних функцій : матеріали 69-тої студентської науково-технічнрої конференції секції кафедр «Захист інформації» та «Безпека інформаційних технологій». Львів, 2011. С. 179-180.

# З М І С Т

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ADC — Analog Digital Codec

AHDL — Altera HDL

ARM — Advanced RISC Machine

ASCII — American Standard Code Information Interchange

ASIC — Application Specific Integrated Circuit

AVR — Alf Vegard RISC

BGA — Ball Grid Array

BIOS — Basic Input Output System

CISC — Complex instruction set computer

CORDIC — Coordinate Rotation Digital Computer

CPU — Central Processor Unit

DAC — Digital Analog Codec

DMA — Direct Memory Access

DSP — Digital Signal Processor

ECP — Extended Capability Port

EEPROM — Electrically Erasable Programmable ROM

EPP — Enhanced Parallel Port

FIFO — First In First Out

FILO — First In Last Out

FBGA — Fine Ball Grid Array

FPGA — Field-Programmable Gate Array

FIR — Finite Impulse Response

GPL — General Public Lisence

HDL — Hardware Description Language

IEC — International Electrotechnical Commission

IEEE — Institute Electrical Electronics Engineers

IP — Internet Protocol

IP Core — Intellectual Property Core

IRQ — Interrupt Request

ISE — Integrated Synthesis Environment

LUT — Look Up Table

MBR — Main Boot Record

OS — Operating System

PLL — Phase Locked Loop

RAM — Random Access Memory

RISC — Reduced Instruction Set Computer

ROM — Read Only Memory

RTL — Register Transfer Level

TSC — Time Stamp Counter

UART — Universal Asynchronous Receiver Transmitter

VHDL — VHSIC Hardware Description Language

VPN — Virtual Private Network

USB — Universal Serial Bus

АЦП — Аналого-цифровий перетворювач

БЖ — Блок живлення

НРН — Найкраще рівномірне наближення

ПЗ — Програмне забезпечення

ПК — Персональний комп’ютер

МК — Мікроконтролер

МКВ — Мадісетті-Квентус-Вілсон

ОС — Операційна система

ПВП — Псевдо випадкова послідовність

ПЗ — Програмне забезпечення

ПЛІС — Програмована логічна інтегральна схема

ППКІ — Паралельно-поліноміальна квадратична інтерполяція

ТПВ — Таблиця попередньої вибірки

ЦАП — Цифро-аналоговий перетворювач

ЦП — Центральний процесор

# ВСТУП

**Актуальність теми.** Дисертаційна робота присвячена удосконаленню методів обчислення елементарних функцій та їх подальшому впровадженню у складі високотехнологічних програмно-апаратних засобів. Незважаючи на суттєві напрацювання стосовно проектування та реалізації методів обчислення елементарних функцій, в даному напрямі залишається багато невирішених питань — таких, як пошук оптимальних алгоритмів обчислення функцій та нових підходів до їх проектування, оптимізації структур існуючих моделей та покращення їх характеристик. Високий рівень зацікавлення до галузі підвищення ефективності обчислень функцій та послідовностей привів до появи різноманітних підходів до проектування та синтезу алгоритмів і схем, що постійно вдосконалюються.

При проектуванні та побудові більшості цифрових пристроїв широкого а також спеціального призначення для використання в різних галузях науки і техніки часто необхідно досягти оптимального співвідношення показників ефективності та ресурсозатратності. Частковим випадком даної проблеми служить задача максимально можливого покращення певної характеристики методу без обмежень на підвищення складності реалізації схеми в цілому. Причому тенденція щодо покращення методів обчислень прослідковується як на програмному так і на апаратному рівнях, що дозволяє говорити про неоптимальність існуючих підходів для розв’язання даної проблеми.

Необхідним завданням на сьогоднішній день є проведення аналізу переваг та недоліків особливостей функціонування сучасних засобів обчислень та їх компонентів, вивчення особливостей архітектури та структури а також пропонованих методів їх удосконалення в той час, як розвиток потужностей обчислювальної техніки надає додаткові можливості для застосування доволі складних та ресурсовитратних методів. Це призводить до збільшення складності

обчислювальної системи і відповідно зростання енергозатрат, тоді як в загальному випадку застосування удосконалених методів обчислень забезпечує високу продуктивність функціонування системи в цілому. Тому актуальним завданням є розроблення і розвиток нових методів, які забезпечували б якісні результати на шляху підвищення ефективності роботи сучасної техніки, що є одним із перспективних напрямків її подальшого розвитку. Таким чином, науково-прикладне завдання підвищення продуктивності та ефективності функціонування програмних та апаратних комплексів є актуальним, що зумовлює необхідність розробки комплексу нових методів та засобів обчислень елементарних функцій. Крім того, важливо дослідити та розробити варіанти імплементації цих методів на різних обчислювальних платформах, забезпечити високу швидкодію їх функціонування, а також можливість вбудовування розроблених обчислювальних компонентів у широкий спектр електронних засобів.

**Зв’язок роботи з науковими програмами, планами, темами.** Тема дисертаційної роботи відповідає одному з наукових напрямів кафедри безпеки інформаційних технологій Національного університету «Львівська політехніка» “Розробка та реалізація методів обчислення елементарних функцій на основі програмних та апаратних засобів”. Результати досліджень, відображені у дисертаційній роботі, отримані у межах виконання науково-дослідних робіт кафедри безпеки інформаційних технологій Національного університету «Львівська політехніка» за темою «Розробка та вдосконалення ітераційних методів обчислення елементарних функцій для систем захисту інформації» № державної реєстрації 0110U004687 та “Використання ітераційного методу CORDIC у системах розпізнавання відбитків пальців” № державної реєстрації 0114U001234.

**Мета і завдання дослідження.** Метою дисертаційної роботи є підвищення ефективності функціонування цифрових пристроїв як шляхом оптимізації вже існуючої програмної та апаратної бази, так і розробкою нових способів

підвищення якості обчислювального процесу. Задля досягнення поставленої мети завдання дослідження були наступні:

- Проаналізувати сучасний стан та шляхи удосконалення відомих методів та засобів обчислень, аналіз їх особливостей, недоліків та переваг у порівнянні з існуючими аналогами.
- Здійснити комп’ютерне імітаційне моделювання функціонування розроблених методів обчислення елементарних, тригонометричних та трансцендентних функцій.
- Розробити структурно-функціональну модель представлених методів.
- Здійснити програмну реалізацію розроблених і вдосконалених методів. Визначити особливості її функціонування на платформах із різною архітектурою та системою команд.
- Дослідити способи ефективного синтезу та імплементації розроблених моделей та алгоритмів. Вивчити можливість розпаралелення обчислень та її вплив на функціонування схеми в цілому.
- Розробити апаратну реалізацію пропонованих пристроїв для платформи ПЛІС з різною функціональністю та архітектурою залежно від виробника.
- Порівняти з точки зору ефективності запропоновані рішення та відомі підходи щодо реалізації пропонованих функцій.

**Об’єкт дослідження** — дискретні процеси у цифрових схемах та пристроях.

**Предмет дослідження** — способи та методи удосконалення обчислень математичних функцій у засобах обробки інформації.

**Методи дослідження** — методи і апаратно-програмні засоби комп’ютерних систем, методи наближення функцій, теорія похибок і непевності результатів

вимірювань, теорія ймовірності, математичної статистики, лінійної алгебри, методи цифрового опрацювання сигналів, методи імітаційного моделювання, чисельні методи, методи обробки даних.

**Наукова новизна отриманих результатів.** За результатами проведених теоретичних та експериментальних досліджень а також їх практичної реалізації розв’язано ряд важливих науково-технічних проблем, щодо підвищення ефективності функціонування програмних та апаратних засобів, а також компонентів комп’ютерних систем. При цьому отримано такі наукові та практичні результати:

вперше:

- запропоновано метод інверсного повороту (вектора), який функціонує паралельно з алгоритмом одностороннього повороту і за рахунок повороту вектора лише в сторону зменшення кута дає змогу вдвічі скоротити кількість ітерацій методу при будь-якому вхідному значенні аргумента.
- запропоновано спосіб паралельної поліноміальної інтерполяції, який за рахунок оптимізації вибору коефіцієнтів та паралельного використання помножувачів дає змогу скоротити кількість ітерацій та час обчислення широкого спектру функцій.

удосконалено:

- метод перекодування кута, який дає змогу скоротити кількість необхідних для обчислень ресурсів шляхом зміни вхідного значення аргументу з наступним представленням та оперуванням лише додатними величинами, що забезпечує коректну роботу алгоритму для беззнакової логіки та спрощує проектування та апаратні затрати при описі схем мовами вищого рівня.
- метод кусково-нелінійної апроксимації, в якому збільшення розрядності та відповідно точності обчислень відбувається за рахунок використання

квадраторів, де розрахунок вищих порядків залишкового кута відбувається без використання додаткових коефіцієнтів та зміни розрядності аргументів у діапазоні обчислень.

- метод паралельної поліноміальної інтерполяції із використанням таблиці попередньої вибірки (ТПВ), що дає змогу зменшувати кількість поліномів та множень за рахунок використання ПЗП без обмежень на мінімальний об’єм таблиці, а при невеликих розрядностях аргументів використовувати лише поліноми першого порядку.

**Практичне значення одержаних результатів.**

- вперше на базі x86 платформи на мовах С та асемблера реалізовано метод перекодування кута із спрощеним конвеєром обчислень, що дає змогу реалізації метода CORDIC із меншою кількістю апаратних ресурсів. Виявлено сильні та слабкі сторони методу.
- адаптовано та реалізовано метод перекодування кута — беззнакове перекодування для платформи ПЛІС “Cyclone” мовою System Verilog, в якому всі аргументи приймають лише додатні величини, чим спрощують реалізацію алгоритму та необхідні апаратні ресурси із збереженням переваг підходу перекодування.
- за допомогою способу представлення розрядів при програмній реалізації методу одностороннього та інверсного повороту використана можливість наперед визначити майбутній хід ітераційного процесу, чим зробити залежності в середині алгоритму менш критичними та більш прогнозованими, що дозволило покращити швидкодію.
- вперше, використовуючи широкосмугові помножувачі ПЛІС реалізовано метод квадратичної апроксимації, який дає можливість з мінімальною латентністю обчислювати тригонометричні функції та спрощувати ітерації методу CORDIC.

- на базі мікроконтролера AVR мовою асемблера розроблено генератор псевдовипадкових чисел у вигляді зовнішнього інтерфейсу для ПК, із можливістю перекладення частини ресурсів процесора на даний периферійний пристрій.
- у НВПП “Спаринг-Віст Центр” розроблено швидкодіючі програмно-апаратні засоби для підвищення точності та розширення діапазону вимірювання детекторів поверхневої густини потоку та інтенсивності випромінювань.
- отримані результати по енергоспоживанню та ресурсоємності пропонованих методів. Практично перевірено коректність їхнього функціонування та визначено перспективи їхнього подальшого впровадження.
- окремі положення дисертаційного дослідження використовувались під час виконання науково-дослідних робіт кафедри безпеки інформаційних технологій Національного університету «Львівська політехніка», а також впроваджені у навчальний процес кафедри у таких дисциплінах як “Комп’ютерні методи високорівневого проектування систем захисту” та “Комп’ютерні методи аналізу та проектування електронних засобів”.

**Особистий внесок здобувача.** Усі наукові результати дисертаційної роботи отримані автором самостійно. У працях, опублікованих у співавторстві, автору належать: розроблення та опрацювання теоретичних основ, отримання математичних співвідношень [1,2,5]; формулювання ідей удосконалення алгоритму та визначення оптимальних шляхів їх реалізації [1,2,4]; організація методів досліджень, розрахунків та проведення експериментів [1,3,4,5,6]; дослідження характеристик пропонованих алгоритмів [1,4,5]; розроблення структури, проектування блок-схем та алгоритмів функціонування програмно-апаратних засобів [1,5,6]; написання програмного забезпечення, та його оптимізація [1-6]; замір швидкодії, точності обчислень, [1,3-6], а також латентності та споживаної потужності [4-6]; запропоновано метод інверсного

повороту та перекодування кута [4,5]; вдосконалення методу одностороннього повороту [2,6]; удосконалення класичного методу CORDIC із покращеною латентністю обчислень [1,4,6]; аналіз впливу різної розрядності імплементованих функцій на продуктивність роботи ПЛІС [4,5]; аналіз роботи гібридного методу із застосуванням запропонованого адаптивного алгоритму [3,5]; групування, аналіз та представлення результатів дослідження [1,3,4,5].

**Апробація результатів дисертації.** Про основні результати наукових досліджень автор доповідав на таких конференціях: V Міжнародна науково-технічна конференція “Захист інформації і безпека інформаційних систем” (Львів, 2-3 червня 2016р.) I Міжнародна науково-технічна конференція “Інформаційна безпека в сучасному суспільстві” (Львів, 21-22 листопада 2014р.) 69-та студентська науково-технічна конференція секції кафедр «Захист інформації» та «Безпека інформаційних технологій» (Львів, 17-18 жовтня 2011р.) Матеріали дисертації неодноразово були оприлюднені та обговорені на наукових семінарах кафедри безпеки інформаційних технологій та кафедри захисту інформації Національного університету «Львівська політехніка».

**Достовірність отриманих результатів.** Для перевірки достовірності результатів представлених у дисертації було проведено ряд експериментальних досліджень під час яких одержано:

- показники швидкодії пропонованих алгоритмів при використанні декількох видів апаратних платформ різних виробників з відмінною розрядністю, системою команд та функціональністю.
- дані щодо точності обчислення реалізованих методів представлених функцій у всьому діапазоні вхідних значень та показники відхилень від очікуваних величин. Результати випробувань свідчать про коректність як програмної реалізації розроблених методів, так і їхніх апаратних аналогів. Результати моделювання у достатній мірі корелюють і з результатами теоретичних досліджень, що відображено у дисертаційній роботі.

**Публікації.** Основні положення та результати дисертаційного дослідження викладено в шести друкованих працях, серед них одна стаття у науковому періодичному виданні іноземної держави, який включено до міжнародної наукометричної бази Scopus; п’ять статей у наукових фахових виданнях України з технічних наук та три публікації за матеріалами наукових конференцій.

**Структура та обсяг дисертації.** Дисертаційна робота складається із переліку умовних позначень, вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи складає 232 сторінки, з яких 213 сторінок основного тексту, що містять 36 рисунків та 42 таблиці. Список використаних джерел налічує 136 найменувань.

# РОЗДІЛ 1
# Загальний огляд та аналіз способів обчислення функцій

## 1.1 Способи обчислення елементарних функцій

У даному розділі розглядаються способи обчислення значень елементарних функцій. До елементарних зазвичай відносять функції піднесення до степеня (в тому числі до дробового степеня — квадратний, кубічний корінь), многочлени, логарифмічну, показову (експоненційну), тригонометричну, гіперболічну і зворотні до них функції. Знаходження значень тих або інших елементарних функцій часто потрібно при здійсненні значного спектру обчислень. При розрахунках на ПК зазвичай використовуються стандартні підпрограми обчислення основних функцій. У всіх розповсюджених алгоритмічних мовах і відповідних трансляторах передбачена можливість звертання до таких підпрограм чи функцій. Більш детально дані способи описані в [8]. Зазвичай, для практичних обчислень достатньо можливостей вбудованих підпрограм. Однак у деяких спеціальних застосуваннях стандартні бібліотечні функції можуть виявитися невідповідними для вирішення поставленої задачі. Часто трапляється так, що елементарна функція обчислюється із надто високою точністю, і це приводить до невиправданих витрат машинного часу. Чи навпаки, при розробці мікропрограм іноді доводиться здійснювати розрахунки значень елементарних функцій безпосередньо машинною мовою за допомогою асемблера, щоб підвищити їх швидкодію. Таким чином, знання способів обчислення часто використовуваних функцій допомагає не тільки усвідомлено застосовувати необхідні підпрограми, але й ефективно удосконалювати вже існуючі алгоритми, та знаходити нові підходи для їх реалізації [109].

## 1.2 Обчислення значень полінома. Схема Горнера

Однією з поширених функцій, що зустрічаються при розв'язуванні

різноманітних задач, є поліном (многочлен):

$$P(x)=C_0+C_1x+C_2x^2+\ldots+C_{n-1}x^{n-1}+C_nx^n \qquad (1.1)$$

де $n$ – степінь полінома. Поліноми використовуються, зокрема, при наближенні функцій, зображенні кривих і поверхонь, для компактного зберігання табличних даних і т. д.

У всіх зазначених випадках потрібно знаходити значення полінома $P(x)$ при заданому значенні аргументу $x$. Розглянемо цю задачу більш детально. При одноразовому обчисленні значення полінома невисокого порядку, послідовність виконання операцій не відіграє суттєвої ролі. Однак часто доводиться обчислювати значення полінома високого порядку у багатьох точках $x$, що належать деякому інтервалу (наприклад, це необхідно при виведенні графіка полінома). У такому випадку обчислення $P(x)$ повинно бути ефективно організоване, тому що воно буде складати основну частину витрат машинного часу.

Розглянемо можливі способи обчислення значення полінома. Нехай величина $x$ задана. Найбільш вдалий на перший погляд спосіб – це виконання операції безпосередньо так, як вони записані у формулі (1.1). Такий метод обчислення буде реалізовуватися, якщо представити поліном (1.1) в «стандартному» записі на деякій алгоритмічній мові з використанням операцій множення й додавання, наприклад:

$$P=C_0+C_1\cdot x+C_2\cdot x\cdot x+C_3\cdot x\cdot x\cdot x+\ldots \qquad (1.2)$$

У цьому випадку для обчислення члена $C_1x$ потрібно одне множення, для члена $C_2x^2 = C_2\cdot x\cdot x$ – два множення, для члена $C_3x^3 = C3\cdot x\cdot x\cdot x$ – три множення і т.д., для члена $C_nx^n$ – $n$ множень. Загальне число множень може бути знайдене відповідно до формули арифметичної прогресії:

$$N_{множ.}=1+2+3+\ldots+n=\frac{n(n+1)}{2} \qquad (1.3)$$

Число додавань у формулі (1.1) дорівнює $N_{дод.}=n$. Загальне число операцій (множень і додавань) при обчисленні полінома таким способом:

$$N = N_{множ.} + N_{дод.} = \frac{n(n+3)}{2} \qquad (1.4)$$

Однак слід мати на увазі, що операція множення є складнішою і виконується довше, ніж операція додавання.

Число операцій множення можна скоротити, якщо попередньо обчислити всі потрібні степені аргументу $x^2, x^3, \ldots, x^n$, при цьому кожний наступний степінь обчислюється із попереднього відповідно до співвідношень $x^k = x^{k-1} \cdot x$. Очевидно, для обчислення степенів до $x^n$ включно буде потрібно $n-1$ множень. Крім того, для одержання членів $C_k x^k$ ($k = \overline{1,n}$) потрібно ще $n$ множень на коефіцієнти $C_k$. Таким чином, загальне число множень дорівнює $N_{множ.} = 2n-1$. Число додавань як і раніше становить $N_{дод.} = n$, і, виходить, загальне число операцій $N = 3n - 1$.

Однак розглянутий спосіб не є найкращим, а значення полінома можна знайти, якщо записати його в такий спосіб:

$$P_n(x) = C_0 + x(C_1 + x(C_2 + \ldots + x(C_{n-2} + x(C_{n-1} + x \cdot C_n))\ldots)) \qquad (1.5)$$

Неважко переконатися в еквівалентності представлень полінома (1.1) і (1.5), для цього потрібно в (1.5) розкрити всі дужки.

Відповідно до формули (1.5), обчислення значення полінома $P_n(x)$ можна звести до наступного ряду операцій:

$$\begin{gathered} P = C_n \\ P = C_{n-1} + x \cdot P \\ P = C_{n-2} + x \cdot P \\ \ldots\ldots\ldots \\ P = C_1 + x \cdot P \\ P = C_0 + x \cdot P \end{gathered} \qquad (1.6)$$

тут величина $P$ оновлюється на кожному кроці, а її кінцеве значення дорівнює значенню полінома.

Спосіб знаходження значення полінома за формулою (1.5) називається схемою Горнера. З розгляду на ряд операцій (1.6) випливає, що тут використовується $n$ множень і $n$ додавань, тобто всього $2n$ арифметичних операцій. Доведено, що в загальному випадку не існує способу обчислення алгебраїчного багаточлена n-*ної* степені менш, ніж за $2n$ арифметичних дій, тобто

схема Горнера є найефективнішою.

Порівняємо три розглянуті способи на прикладі обчислення значення полінома десятого степеня ($n$=10). Число необхідних операцій множення, додавання, та загальне число операцій наведені в таблиці 1.1 (тут спосіб 1 – обчислення полінома безпосередньо по формулі (1.1), спосіб 2 – попереднє обчислення степенів аргументу *x)*.

*Таблиця 1.1*

Число операцій при обчисленні значення полінома

| Спосіб обчислення | Спосіб 1 | Спосіб 2 | Схема Горнера |
|---|---|---|---|
| Число множень $N_{множ.}$ | 55 | 19 | 10 |
| Число додавань $N_{дод.}$ | 10 | 10 | 10 |
| Загальне число операцій $N$ | 65 | 29 | 20 |

Як видно, при $n$=10 застосування схеми Горнера дозволяє в 5,5 раз зменшити число множень і більше, ніж в 3 рази скоротити загальне число операцій у порівнянні з «очевидним» способом обчислень за формулою (1.1). При багаторазовому обчисленні значень полінома цей виграш стає достатньо суттєвим. При збільшенні степеня полінома переваги схеми Горнера будуть ще більш значними.

Схема Горнера є достатньо зручною для реалізації на ПК завдяки циклічності обчислень і через свою ощадливість до використовуваної пам'яті – крім коефіцієнтів многочлена й значення аргументу, тут потрібно зберігати тільки одну проміжну величину ($P$). Алгоритмічною мовою здійснити обчислення значення полінома за схемою Горнера можна за допомогою єдиного циклу.

Ще однією перевагою схеми Горнера – менші похибки результату в порівнянні з безпосереднім обчисленням полінома згідно з формулою (1.1). Даний факт є наслідком меншого числа операцій.

Слід зазначити додаткове джерело помилок при записі полінома в «прийнятній» для алгоритмічної мови формі. У багатьох алгоритмічних мовах є операція піднесення числа в степінь (цілу або дробову). На перший погляд варто скористатися цією операцією для обчислення значень полінома. Однак зазвичай

операція піднесення в степінь реалізується з використанням логарифмічної і показової функцій. Нехай необхідно обчислити $s=x^a$. Тоді $\ln s = a \ln x = y$ і $s$ визначається як $s = e^y = e^{a \ln x}$.

Логарифм і експонента обчислюються за допомогою рядів, при обрахунку яких може мати місце похибка, що буде зростати при вищих степенях $a$. Тому знаходження значення полінома з використанням операції піднесення до степеня (тобто застосування «стандартного» запису полінома в даній алгоритмічній мові) призводить до додаткових похибок. При використанні схеми Горнера такі помилки відсутні.

## 1.3 Етапи обчислення елементарних функцій та способи їх реалізації.

Далі ми розглянемо обчислення таких розповсюджених елементарних функцій, як показова, логарифмічна, тригонометрична функція і добування квадратного кореня із числа.

Існують два основні способи реалізації обчислення елементарних функцій в ПК у вигляді програмної та апаратної реалізації.

Прикладом програмної реалізації служать стандартні підпрограми обчислення елементарних функцій, написані на одній з алгоритмічних мов програмування. Перевагою програмної реалізації є простота використання та універсальність, недоліком – низька швидкодія.

Під апаратною реалізацією елементарних функцій мається на увазі, що відповідні підпрограми «зашиті» в архітектурі чи мікрокоді процесора ПК. Апаратна реалізація використовується в програмувальних калькуляторах чи калькуляторах для інженерних розрахунків, спеціалізованих керуючих компонентах та ін. Серед основних переваг апаратної реалізації є висока швидкодія обчислень, ощадливе використання пам'яті, енергетичних ресурсів, можливість врахування специфіки задачі (наприклад, можливо обчислювати елементарну функцію саме з тією точністю, яка необхідна). Серед недоліків такого

підходу є мала універсальність та складність реалізації нових типів елементарних функцій.

У загальному випадку обчислення елементарних функцій включає такі етапи:

1) приведення аргументу до основного інтервалу, виділення ділянок зміни аргументу з лінійною залежністю функції, інтервалів переповнення й заборонених значень;

2) обчислення функції в основному інтервалі.

3) корекція результату у випадку зміни інтервалу обчислень.

Нижче ці етапи розглядаються на прикладі розповсюджених елементарних функцій.

## 1.4 Приведення аргументу до основного інтервалу

### 1.4.1 Область лінійного представлення функції

Зменшення діапазону зміни аргументу (приведення аргументу) у багатьох випадках сприяє поліпшенню збіжності функції й скороченню загального часу обчислення. Крім того, таке приведення дозволяє уникнути причин, що унеможливлюють обчислення функції в ПК (наприклад, переповнення розрядної сітки).

Необхідно нагадати, що діапазон чисел в ПК перебуває в межах від $-M_\infty$ до $M_\infty$, де $M_\infty$ - машинна нескінченність. При обчисленні елементарної функції весь діапазон значень аргументу $I=[-M_\infty,M_\infty]$ можна розбити на наступні області:

1) приведення аргументу $I_{пр}$;

2) лінійного представлення функції (лінійної апроксимації) $I_{лін}$;

3) переповнення $I_{пп}$;

4) заборонених значень аргументу $I_{заборон}$.

Область приведення аргументу $I_{пр}$ – це основний інтервал обчислення

елементарної функції $f(x)$. При цьому обчислення $f(x)$ у довільній точці $x \in I$ заміняють обчисленням цієї ж функції в точці $x_1 \in I_{\text{пр}}$. Прикладом можуть служити тригонометричні функції, для знаходження значень яких при будь-яких $x$ достатньо вміти обчислювати їх в інтервалі $x \in [0, \frac{\pi}{2}]$. Прийоми скорочення діапазону зміни аргументу достатньо різні й залежать від самої функції, деякі із цих прийомів будуть розглянуті нижче.

Для багатьох елементарних функцій доцільно виділити таку область зміни аргументу $I_{\text{лін}}$, у якій поведінка $f(x)$ описується найбільш простою лінійною залежністю. Фактично в інтервалі $I_{\text{лін}}$ ми описуємо $f(x)$ за допомогою розкладання в ряд Тейлора в околі деякої точки $x_0 \in I_{\text{лін}}$, обмежуючись першими двома членами:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) \qquad (1.7)$$

Залежність (1.7) є лінійною щодо аргументу $x$. Як приклад, можна навести відоме співвідношення $\sin x \approx x$, вірне при невеликих значеннях аргументу.

Виділення інтервалу $I_{\text{лін}}$ дозволяє знаходити значення $f(x)$ у цьому інтервалі за допомогою дуже простого лінійного співвідношення, це суттєво прискорює обчислення. Зазвичай за область $I_{\text{лін}}$ вибирають такий інтервал, у межах якого відхилення $f(x)$ від лінійної залежності (1.7) не перевищує останнього десяткового розряду машинної сітки, тобто досягається необхідна точність лінійного представлення функції.

При певних значеннях аргумента $x \in I_{\text{пп}}$ величина функції $f(x)$ така, що відбувається переповнення розрядної сітки ПК. У цьому випадку обчислення не проводяться й встановлюється прапорець переповнення. Величину аргументу, при якому відбувається переповнення розрядної сітки, можна визначити з умови:

$$|f(x)| > M_{\infty} \qquad (1.8)$$

У область заборонених значень входять значення аргументу $x$, при яких функція $f(x)$ не визначена, тобто її неможливо обчислити в принципі. Прикладами таких областей можуть служити всі негативні числа при добуванні квадратного

кореня, логарифма і т. д. При потраплянні аргументу в область $I_{\text{заборон.}}$ встановлюється прапорець забороненої операції. Розгляд зазначених ділянок зміни аргументу здійснимо на прикладі розповсюджених елементарних функцій.

### 1.4.2 Експонента

Функція $e^x$ визначена на всій числовій осі, тому аргумент може змінюватися в межах від $-M_\infty$ до $M_\infty$ (область $I_{\text{заборон.}}$ не існує).

Тому що функція $e^x$ монотонно зростає $e^x > x$, і при достатньо великих $x$ може відбутися переповнення розрядної сітки ПК. Умова переповнення має вигляд $e^x > M_\infty$ або $x > \ln M_\infty$. Обмежившись діапазоном $x \in [-M_\infty, \ln M_\infty]$, можна уникнути переповнення.

Якщо аргумент $x$ є негативним і достатньо великим по модулю, то функція $e^x$ прямує до нуля. Тому для експоненти можна виділити ділянку зміни аргумента $I_0$ з нульовим значенням функції. Ця ділянка випливає з умови $e^x < M_0$ або $x < \ln M_0$, де $M_0$ – машинний нуль. В області $I_0 = [-M_\infty, \ln M_0]$ значення $e^x$ дорівнює нулю в межах розрядної сітки.

Таким чином, діапазон зміни аргумента може бути визначений інтервалом $[\ln M_0, \ln M_\infty]$. Негативні аргументи можуть бути зведені до позитивних за допомогою співвідношення $e^{-x} = \frac{1}{e^x}$.

Тепер припустимо, що величина $x$ є позитивною, і представимо її у вигляді:

$$x = k \ln 10 + z, \qquad (1.9)$$

де $k$ – ціле число; $z$ – остача; $0 \le z < \ln 10$.

У результаті діапазон зміни аргумента виявляється зведеним до основного інтервалу $I_{\text{пр}} = [0, \ln 10)$, тому що $e^x = e^{z+k \ln 10} = e^z \cdot 10^k$.

Для знаходження лінійного представлення функції $e^x$ розглянемо її розкладання у ряд Тейлора в околі точки $x_0 = 0$. Обмежившись першими двома членами, одержимо:

$$e^x = 1 + x. \tag{1.10}$$

Можна показати, що при $x \le 10^{-m/2}$ лінійна апроксимація (1.10) є точною в межах машинної сітки з $m$ десятковими розрядами, тобто діапазон лінійного представлення експоненти $I_{\text{лін}} = [0;\ 10^{-m/2}]$.

На рис.1.1 показані знайдені інтервали зміни аргументу для функції $e^x$.

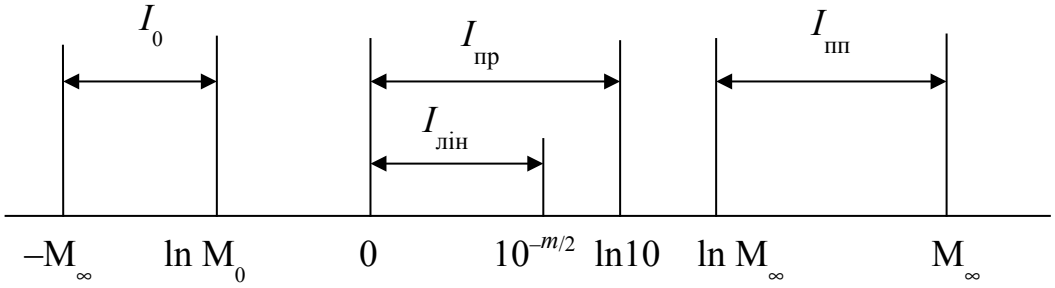



Рис. 1.1 Область приведеного аргументу функції $e^x$

### 1.4.3 Логарифм

Функція ln $x$ визначена тільки для позитивних значень аргументу $x$ (тобто $I_{\text{забор.}} = [-M_\infty,\ 0]$) і є монотонно зростаючою. Однак переповнення розрядної сітки відсутнє, тому що $\ln x < x$ (інтервал $I_{\text{пп}}$ не існує).

Для скорочення інтервалу $(0,\ M_\infty]$ представимо аргумент у вигляді:

$$x = X \cdot 10^k \tag{1.11}$$

де $1 \le X < 10$ і $k$ – ціле число.

Тоді $\ln x = \ln X + k \ln 10$, і діапазон зміни аргументу виявився зведеним до основного інтервалу $I_{\text{пр}} = [1;\ 10)$.

Ще один спосіб звуження інтервалу полягає в обчисленні числа $U = X / \sqrt{10}$. Тоді:

$$\ln x = \begin{cases} \ln x, \text{ if } 1 \le x \le \sqrt{10} \\ \ln x + 0.5 \cdot \ln 10, \text{ if } \sqrt{10} < x < 10 \end{cases} \tag{1.12}$$

У цьому випадку діапазон зведених значень задається інтервалом $I_{\text{пр}} = [1;\ \sqrt{10})$.

Лінійна апроксимація логарифма $\ln x \approx x - 1$ правильна у вузькому інтервалі

$|x–1| \le 2 \cdot 10^{–m}$, тому її використання недоцільне, на відміну, наприклад, від [11].

### 1.4.4 Тригонометричні функції синуса та косинуса

Тригонометричні функції sin *x* та cos *x* визначені на всій числовій осі й мають період, рівний 2π. Функція sin *x* непарна, а cos *x* – парна, що дозволяє замінити негативні аргументи позитивними:

$$\begin{aligned} \sin(-x) &= -\sin x \\ \cos(-x) &= \cos x \end{aligned} \tag{1.13}$$

Нехай тепер аргумент *x* позитивний. Віднімаючи від аргументу число, кратне 2π, і використовуючи відомі в тригонометрії формули приведення, можна звузити діапазон зміни кута до першого квадранта. Представимо аргумент *x* у вигляді:

$$x = \varphi + \frac{\pi}{2} l + 2\pi k, \tag{1.14}$$

де *k* і *l* – цілі числа; $0 \le l \le 3$; $0 \le \varphi < \frac{\pi}{2}$. Тоді тригонометричні функції можуть бути обчислені відповідно до таблиці 1.2, де кут φ змінюється в першому квадранті (тобто $I_{\text{пр}} = [0; \frac{\pi}{2})$).

*Таблиця 1.2*

Приведення аргументу тригонометричних функцій

| *l* | sin *x* | cos *x* |
|---|---|---|
| 0 | sin φ | cos φ |
| 1 | cos φ | –sin φ |
| 2 | –sin φ | –cos φ |
| 3 | –cos φ | sin φ |

Зазначене приведення досягається шляхом віднімання від аргументу *x* константи $\frac{\pi}{2}$, помноженої на ціле число $q = l + 4k$. Очевидно, що точність

приведення буде втрачена, якщо величина $\frac{\pi}{2}q$ вийде за межі розрядної сітки ПК. Такий випадок має місце при значеннях аргументу $x > 10^m$, де $m$ – число десяткових розрядів мантиси. Тому слід обмежити область зміни вихідного аргументу інтервалом $x \in [-10^m; 10^m]$, а області $x<-10^m$ і $x>10^m$ є забороненими. Зауважимо, що основну область обчислення тригонометричних функцій можна додатково звузити до інтервалу $I_{\text{пр}} = [0; \frac{\pi}{4}]$, якщо використовувати відомі співвідношення:

$$\begin{aligned} \sin x &= 2\sin\frac{x}{2}\cos\frac{x}{2} = \cos\left(\frac{\pi}{2} - x\right) \\ \cos x &= 2\cos^2\frac{x}{2} - 1 = \sin\left(\frac{\pi}{2} - x\right) \end{aligned} \qquad (1.15)$$

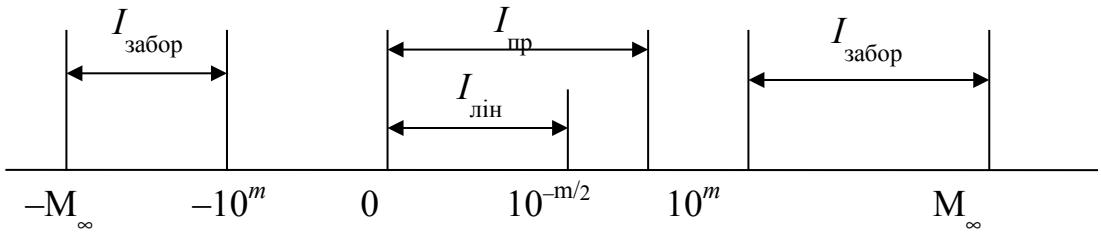



Рис. 1.2 Область приведеного аргументу тригонометричної функції

Наприклад, якщо $\frac{\pi}{4} < x \le \frac{\pi}{2}$, замість обчислення sin $x$ можна обчислити cos φ, де φ $= \frac{\pi}{2} - x$, $0 \le \varphi < \frac{\pi}{4}$.

З розкладання функцій sin $x$ і cos $x$ у ряд Тейлора в околі нуля одержимо лінійні представлення цих функцій:

$$\begin{aligned} \sin x &\approx x \\ \cos x &\approx 1 \end{aligned} \qquad (1.16)$$

В інтервалі $|x| < 10^{-m/2}$ дані представлення точні в межах m - *розрядної* сітки. Області наведених і заборонених значень аргументу функцій sin $x$ і cos $x$ показано на рис.1.2.

### 1.4.5 Квадратний корінь

Функція $f(x)=\sqrt{x}$ визначена тільки для позитивних значень аргументу, тому $x$ лежить в інтервалі [0; M$_\infty$], область $x\in[-M_\infty; 0)$ є забороненою. При $x\geq 1$ правильне співвідношення $\sqrt{x}\leq x$, отже, переповнення розрядної сітки відсутнє (область $I_{пп}$ не існує).

Представимо аргумент $x$ у вигляді:

$$\begin{matrix} x=X\cdot 10^{P} \\ 1\leq X<10 \end{matrix} \qquad (1.17)$$

Тоді величину $\sqrt{x}$ можна записати в наступному вигляді:

$$\sqrt{x}=\begin{cases} \sqrt{x}\cdot 10^{\frac{P}{2}}, if\ P\ even \\ \sqrt{10\cdot X}\cdot 10^{\frac{P-1}{2}}, if\ P\ odd \end{cases} \qquad (1.18)$$

Відповідно до (1.18), замість обчислення $\sqrt{x}$ в інтервалі $x\in[0;\ M_\infty]$ достатньо знайти $\sqrt{x}$ в діапазоні $x\in[1;\ 10)$ при парному $P$ і в діапазоні $x\in[10;100)$ при непарному $P$, тобто наведений інтервал аргументу рівний $I_{пр}=[1;\ 100)$.

На рис.1.3 показані області заборонених значень і наведеного аргументу квадратного кореня.

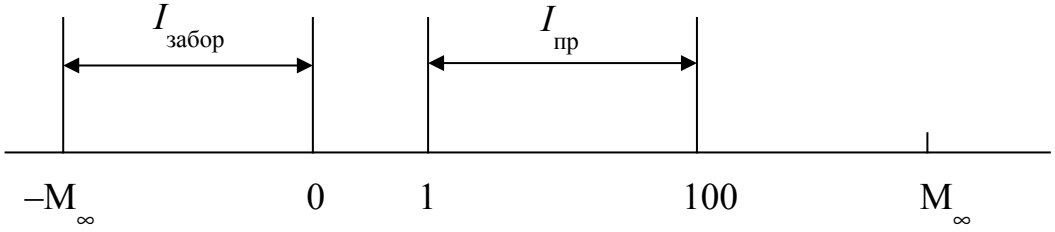



Рис. 1.3 Область приведеного аргументу функції

## 1.5 Обчислення значень елементарних функцій

### 1.5.1 Способи обчислення

Після приведення аргументу до основного інтервалу необхідно обчислити значення функції в цьому інтервалі. Найчастіше використовуються наступні

способи обчислення елементарних функцій:

1) розклад в степеневий ряд;
2) поліноміальна апроксимація;
3) дробово-раціональна апроксимація;
4) наближення ланцюговим дробом;
5) ітераційні методи.

Розглянемо ці способи.

### 1.5.2 Розклад в степеневий ряд

Представлений метод подає функцію $f(x)$ у вигляді часткової суми елементів ряду Тейлора:

$$f(x) \approx \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i, \quad i = \overline{1,n}, \tag{1.19}$$

де $n$ – степінь останнього члена, що залишається в ряді; $x_0$ – точка, щодо якої відбувається розклад функції f(x) у ряд $f^{(i)}(x_0)$ – похідна f(x) і-того порядку в точці $x_0$.

При фіксованій величині $x_0$ відрізок ряду (1.19) фактично являє собою поліном $P_n(x)$ аргументу $x$ степені $n$:

$$P_n(x) = C_0 + C_1 x + ... + C_n x^n \tag{1.20}$$

де коефіцієнти $C_i$ залежать від величини $x_0$ і $f^{(i)}(x_0)$. Поліном (1.20), коефіцієнти якого отримані шляхом розкладання f(x) у ряд Тейлора, називається поліномом (многочленом) Тейлора цієї функції. Представлення f(x) з допомогою полінома Тейлора (1.20) є частковим випадком поліноміальної апроксимації. Порядок полінома n (число членів відрізка ряду) визначається необхідною точністю наближення f(x).

Як приклад розглянемо представлення функції синсуса у вигляді відрізка степеневого ряду. Як зазначалося в розділі 1.4.4 достатньо вміти обчислювати

синус на основному інтервалі $x \in [0;\ \frac{\pi}{2}]$. Однак у випадку обчислень при розкладанні функції в ряд, більш зручним способом є приведення аргументу до інтервалу $x \in [0;\ 1]$. Таке приведення легко виконати шляхом заміни змінної $x$ на $\frac{\pi x}{2}$. У цьому випадку замість розкладання функції синуса на проміжку $x \in [-\frac{\pi}{2};\ \frac{\pi}{2}]$ розглядається розкладання функції $\sin\frac{\pi x}{2}$ на інтервалі $x \in [-1;\ 1]$ в околі $x = 0$. Обмежившись, наприклад, п'ятьма першими членами ряду Тейлора, одержимо поліном Тейлора 9-го порядку по непарних степенях $x$:

$$\sin\frac{\pi x}{2} \approx \frac{\pi x}{2} - \frac{1}{3!}\left(\frac{\pi x}{2}\right)^3 + \frac{1}{5!}\left(\frac{\pi x}{2}\right)^5 - \frac{1}{7!}\left(\frac{\pi x}{2}\right)^7 + \frac{1}{9!}\left(\frac{\pi x}{2}\right)^9 = \\ = C_1 x + C_3 x^3 + C_5 x^5 + C_7 x^7 + C_9 x^9. \qquad (1.21)$$

Чисельні значення коефіцієнтів полінома наведені в таблиці 1.3.

*Таблиця 1.3*

Коефіцієнти поліномів Тейлора найкращого рівномірного наближення для функції синуса

| Коефіцієнти | Поліном Тейлора | Поліном НРН |
|---|---|---|
| $C_1$ | 1,5707963 | 1,5707963 |
| $C_3$ | –0,6459641 | –0,64596336 |
| $C_5$ | $0{,}79692626 \cdot 10^{-1}$ | $0{,}79688475 \cdot 10^{-1}$ |
| $C_7$ | $-0{,}46817541 \cdot 10^{-2}$ | $-0{,}46722203 \cdot 10^{-2}$ |
| $C_9$ | $0{,}16044118 \cdot 10^{-3}$ | $0{,}15081716 \cdot 10^{-3}$ |

Похибка обчислення функції синуса за допомогою полінома Тейлора складається із двох частин – похибки округлення і похибки обмеження (методичної похибки). Остання похибка виникає через обчислення лише обмеженої кількості членів ряду. Для полінома (1.21) у зв'язку з його невисоким порядком переважною буде похибка обмеження.

Графіки функції $\sin\frac{\pi x}{2}$ і полінома Тейлора (1.21) наведені на рис.1.4. На

рис.1.5 показаний розподіл абсолютної похибки ε(x) функції $\sin\frac{\pi x}{2}$ поліномом на інтервалі $x \in [0; 1]$. Як видно, похибка є незначною в околі точки розкладу $x_0 = 0$ і зростає при віддаленні від цієї точки. Максимальна величина абсолютної похибки має місце при $x = 1$ і рівна $\varepsilon_{max}=3{,}54\cdot10^{-6}$. Подібний розподіл похибки в околі точки розкладу $x_0$ характерний при представленні будь-яких функцій за допомогою полінома Тейлора.

При розрахунках на ПК важлива не тільки точність представлення елементарної функції, але й швидкість її обчислення, яка визначається числом необхідних операцій [102]. При обчисленні функції $\sin\frac{\pi x}{2}$ за формулою (1.21) доцільно записати поліном за схемою Горнера:

$$\sin\frac{\pi x}{2}\approx + x\left(C_1 + x^2\left(C_3 + x^2\left(C_5 + x^2\left(C_7 + x^2 C_9\right)\right)\right)\right) \qquad (1.22)$$

Неважко переконатися, що в цьому випадку для обчислення синуса буде необхідно здійснити шість множень і чотири додавання. Коефіцієнти $C_i$ в (1.21), (1.22) обчислюються один раз і зберігаються в пам'яті ПК.

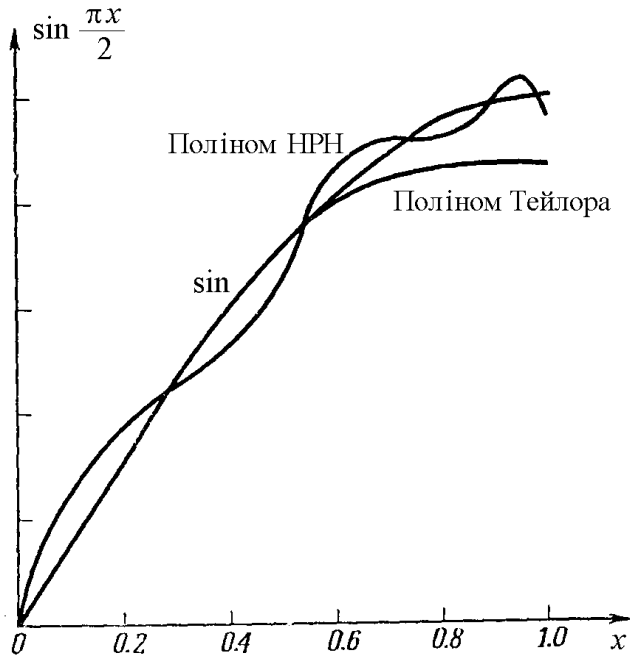



Рис. 1.4 Наближення функції синуса за допомогою полінома Тейлора і полінома НРН (масштаб похибок значно збільшений)

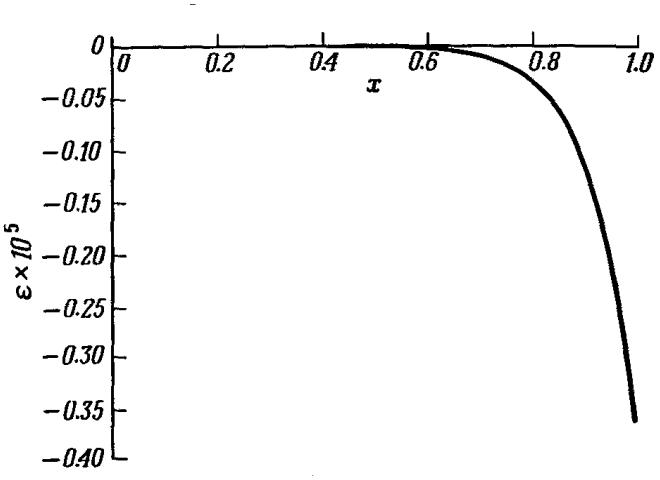



Рис. 1.5 Розподіл похибки ε($x$) на інтервалі $x\in$ [0, 1] при представлені функції синуса поліномом Тейлора

### 1.5.3 Поліноміальна апроксимація

З погляду точності представлення функції $f(x)$ апроксимація її поліномом Тейлора не є найкращою. При цьому ж порядку полінома $P_n(x)$ можливо так підібрати його коефіцієнти, що похибка наближення $f(x)$ на заданому інтервалі буде суттєво нижчою.

Найменшу величину похибки дають так звані поліноми найкращого рівномірного наближення (НРН). Їх отримують у результаті розв'язку задачі найкращої рівномірної (чебишевської) апроксимації функції $f(x)$ поліномом $P_n(x)$ на інтервалі $x\in[a; b]$:

$$\Delta = \max_{x\in[a;b]} |f(x) - P_n(x)| = \min \tag{1.23}$$

Відзначимо, що алгоритм знаходження коефіцієнтів таких поліномів досить складний і вимагає значно більших витрат машинного часу, ніж обчислення коефіцієнтів многочлена Тейлора. Слід враховувати, що коефіцієнти полінома НРН для заданої елементарної функції достатньо обчислити один раз і потім зберігати їх у пам'яті.

Чисельні значення коефіцієнтів непарного полінома НРН 9-ї степені для

функції $\sin\frac{\pi x}{2}$ представлені в таблиці 1.4. Графік полінома НРН показаний на рис.1.4, а розподіл абсолютної похибки представленої функції $\sin\frac{\pi x}{2}$ на інтервалі $x \in [0; 1]$ – на рис.1.6. Видно, що, на відміну від полінома Тейлора, похибки для полінома НРН рівномірно розподілені по всьому інтервалу наближення і приймають поперемінно рівні по модулю і різні за знаком значення. Максимальна величина абсолютної похибки на інтервалі $x \in [0; 1]$ рівна приблизно $5 \cdot 10^{-9}$. Таким чином, при тому ж порядку $n = 9$ (тобто при тій же складності обчислень) похибка обчислення функції синуса поліном НРН в сімсот разів менша в порівнянні з поліномом Тейлора.

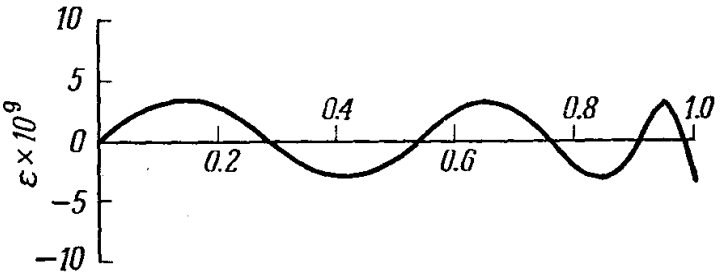



Рис. 1.6 Розподіл похибки $\varepsilon(x)$ на інтервалі $x \in [0, 1]$ при представленні функції синуса поліномом НРН

Варто звернути увагу на різницю в масштабі похибок в порівнянні з рис 1.5. Зауважимо, що похибка $5 \cdot 10^{-9}$ не перевищує одиниці останнього десяткового розряду для чисел одинарної точності в процесорах персональних комп'ютерів.

### 1.5.4 Дробово-раціональна апроксимація

Не всі функції можна з достатньою точністю наблизити поліномами невисоких степенів. Для деяких функцій кращі результати по точності й швидкості обчислення дає застосування дробово-раціональної апроксимації, тобто апроксимації відношенням двох поліномів:

$$f(x) \approx \frac{a_0 + a_1 x + a_2 x^2 + \ldots + a_s x^s}{b_0 + b_1 x + b_2 x^2 + \ldots + b_t x^t} \tag{1.24}$$

Коефіцієнти дробово-раціональної функції (1.24) можна знайти на основі розкладу функції $f(x)$ у ряд Тейлора. Припустимо, наприклад, що функція $f(x)$ повинна бути представлена у вигляді відношення двох поліномів другого порядку:

$$f(x) \approx \frac{a_0 + a_1 x + a_2 x^2}{1 + b_1 x + b_2 x^2} \tag{1.25}$$

Приймемо значення $b_0$ рівне одиниці, тому що будь-яку дробово-раціональну функцію можна привести до цього виду, скоротивши чисельник і знаменник на відповідну константу.

Коефіцієнти $a_i$ і $b_i$ є невідомими величинами й повинні бути визначені. Запишемо розкладання $f(x)$ у ряд Тейлора:

$$f(x) = C_0 + C_1 x + C_2 x^2 + \ldots \tag{1.26}$$

де коефіцієнти $C_i$ відомі. Прирівнявши праві частини (1.25) і (1.26), одержуємо:

$$a_0 + a_1 x + a_2 x^2 = (1 + b_1 x + b_2 x^2)(C_0 + C_{1x} + C_2 x^2 + \ldots) \tag{1.27}$$

Тепер розкриємо дужки і прирівняєм коефіцієнти при однакових степенях $x$. У результаті одержимо п'ять лінійних рівнянь для п'яти невідомих $a_0$, $a_1$, $a_2$, $b_1$, $b_2$:

$$\begin{gathered} a_0 = C_0 \\ a_1 = C_1 + C_0 b_1 \\ a_2 = C_3 + C_2 b_1 + C_1 b_2 \\ 0 = C_4 + C_3 b_1 + C_2 b_2 \\ 0 = C_5 + C_4 b_1 + C_3 b_2 \end{gathered} \tag{1.28}$$

Останні два рівняння в (1.28) демонструють той факт, що в чисельнику дробово-раціональної функції коефіцієнти при степенях $x$ вище двох, дорівнюють нулю. Розв'язати систему лінійних рівнянь (1.28) можна, наприклад, методом Гаусса.

За допомогою розглянутого способу знайдемо просте дробово-раціональне представлення функції $\sin\frac{\pi x}{2}$ :

$$\sin\frac{\pi x}{2} \approx \frac{a_1 x + a_3 x^3}{1 + b_2 x^2} \tag{1.29}$$

Враховуючи розклад (1.29) для функції $\sin\frac{\pi x}{2}$, запишемо:

$$a_1 x + a_3 x^3 = (1 + b_2 x^2)[\frac{\pi}{2} x - \left(\frac{\pi}{2}\right)^3 \frac{1}{3!} x^3 + \left(\frac{\pi}{2}\right)^5 \frac{1}{5!} x^5 - \ldots] \tag{1.30}$$

Звідси одержимо систему рівнянь:

$$\begin{aligned} a_1 &= \frac{\pi}{2} && \text{(коефіцієнти при } x\text{)} \\ a_3 &= b_2 \cdot \frac{\pi}{2} - \left(\frac{\pi}{2}\right)^3 \frac{1}{3!} && \text{(коефіцієнти при } x^3\text{)} \\ 0 &= -b_2 \left(\frac{\pi}{2}\right)^3 \frac{1}{3!} + \left(\frac{\pi}{2}\right)^5 \frac{1}{5!} && \text{(коефіцієнти при } x^5\text{)} \end{aligned} \tag{1.31}$$

Вирішуючи цю систему, знайдемо значення коефіцієнтів дробово-раціональної функції (1.29):

$$\begin{aligned} a_1 &= \frac{\pi}{2} = 1{,}57079633 \\ a_3 &= -\frac{7}{60}\left(\frac{\pi}{2}\right)^3 = -0{,}452174868 \\ b_2 &= \frac{1}{20}\left(\frac{\pi}{2}\right)^2 = 0{,}123370055 \end{aligned} \tag{1.32}$$

Використовуючи дані обчисленнями можна переконатися, що максимальна абсолютна похибка представлення (1.29) на інтервалі $x \in [0; 1]$ дорівнює приблизно $3{,}8 \cdot 10^{-3}$. Приблизно таку ж похибку дає ряд Тейлора із трьома членами:

$$\sin\frac{\pi x}{2} = \frac{\pi x}{2} - \frac{1}{3!}\left(\frac{\pi x}{2}\right)^3 + \frac{1}{5!}\left(\frac{\pi x}{2}\right)^5 = C_1 x + C_3 x^3 + C_5 x^5 \tag{1.33}$$

Прирівняємо по кількості операцій обчислення $\sin\frac{\pi x}{2}$ за допомогою полінома Тейлора (1.33) і дробово-раціональної функції (1.29). Якщо поліном (1.33) представити за схемою Горнера як $P_5(x) = x\,(C_1 + x^2\,(C_3 + x^2\,C_5))$, то для його

обчислення буде потрібно чотири множення і два додавання. Якщо ж при обчисленні виразу (1.29), використовувати систему Горнера для чисельника $a_1x + a_3x^3 = x\,(a_1 + a_3x^2)$, то необхідно чотири множення, два додавання і одне ділення.

Таким чином, у цьому випадку ніякої економії часу при обчисленні дробово-раціонального представлення (1.29) у порівнянні з поліномом Тейлора (1.33) не виходить. Слід, однак, мати на увазі, що вираз (1.29) отриманий на основі розкладу вихідної функції $f(x)$ у ряд Тейлора, не є оптимальним способом наближення $f(x)$ (див. розділ 1.5.2). Тому й отримані коефіцієнти дробово-раціональної функції (1.29) не є найкращими для заданих порядків чисельника й знаменника. Найменшу похибку можна одержати, якщо розв'язати задачу найкращої рівномірної апроксимації функції $f(x)$ дробово-раціональною функцією на інтервалі $x \in [a;\ b]$ при фіксованих порядках чисельника ($s$) і знаменника ($t$): знайти поліноми $A_s(x)$ і $B_t(x)$ такі, що:

$$\Delta = \max_{x\in[a;b]} \left| f(x) - \frac{A_s(x)}{B_t(x)} \right| = \min \qquad (1.34)$$

Задача (1.34) зазвичай вирішується за допомогою спеціальних ітераційних алгоритмів. У багатьох випадках похибка представлення $f(x)$ за допомогою знайденою в такий спосіб «найкращої» дробово-раціональної функції менша, ніж за допомогою полінома НРН (1.23) еквівалентного порядку (тобто степеня, що призведе до такого ж числа операцій). Завдяки цьому дробово-раціональне представлення достатньо часто використовується при обчисленні елементарних функцій.

### 1.5.5 Наближення ланцюговим дробом

У ряді випадків число операцій можна скоротити, якщо представити елементарну функцію у формі ланцюгового дробу. Ланцюговим (безперервним) дробом називається вираз виду:

$$D = d_0 + \cfrac{e_1}{d_1 + \cfrac{e_2}{d_2 + \cfrac{e_3}{d_3 + e_4}}} \tag{1.35}$$

Будь-яку дробово-раціональну функцію можна записати у вигляді еквівалентного ланцюгового дробу.

Як приклад перейдемо від дробово-раціонального представлення (1.29) функції $\sin\frac{\pi x}{2}$ до представлення у формі ланцюгового дробу. Спочатку перепишемо (1.29) до наступного виду:

$$D(x) = \frac{a_1 x + a_3 x^3}{1 + b_2 x^2} = \frac{a_3}{b_2} \frac{x^3 + \frac{a_1}{a_3} x}{x^2 + \frac{1}{b_2}} = k_0 \frac{x^3 + \frac{a_1}{a_3} x}{x^2 + \frac{1}{b_2}}, \tag{1.36}$$

$$\text{де } k_0 = \frac{a_3}{b_2}.$$

Розділимо тепер в останньому дробі чисельник на знаменник, та одержимо при цьому частку $x$ і остачу $\left(\frac{a_1}{a_3} - \frac{1}{b_2}\right)x$:

$$D(x) = k_0 \left[ x + \frac{\left(\frac{a_1}{a_3} - \frac{1}{b_2}\right)x}{x^2 + \frac{1}{b_2}} \right] = k_0 \left[ x + \frac{k_1 x}{x^2 + \frac{1}{b_2}} \right] \tag{1.37}$$

$$\text{де } k_1 = \frac{a_1}{a_3} - \frac{1}{b_2}.$$

Претворивши другий доданок у квадратних дужках, знайдемо:

$$D(x) = k_0 \left[ x + \frac{k_1}{x + \frac{1}{b_2 x}} \right] = k_0 \left[ x + \frac{k_1}{x + \frac{k_2}{x}} \right] \tag{1.38}$$

де $k_2 = \frac{1}{b_2}$.

В результаті одержуємо представлення функції $\sin\frac{\pi x}{2}$ у формі ланцюгового дробу:

$$\sin\frac{\pi x}{2} \approx k_0\left[x + \frac{k_1}{x + \frac{k_2}{x}}\right], \qquad (1.39)$$

де значення коефіцієнтів:

$$k_0 = \frac{a_3}{b_2} = -3{,}66519143;$$

$$k_1 = \frac{a_1}{a_3} - \frac{1}{b_2} = -11{,}4722143;$$

$$k_2 = \frac{1}{b_2} = 8{,}03055026.$$

Зрозуміло, що при знайдених коефіцієнтах $k_0$, $k_1$ і $k_2$ точність представлення функції $\sin\frac{\pi x}{2}$ за допомогою ланцюгового дробу (1.39) така ж, як і з використанням вихідної дробово-раціональної функції (1.29). Прирівняємо тепер число операцій. Очевидно, обчислення ланцюгового дробу (1.39) вимагає двох ділень, двох додавань і одного множення. Це значно менше, ніж необхідно для обчислення раціональної функції (1.29). У деяких випадках витрати машинного часу на операцію ділення ненабагато перевищують відповідні витрати на операцію множення, а у деяких процесорних архітектур тривалість цих операцій ідентична. Тому використання ланцюгового дробу (1.39) може виявитися ефективнішим, ніж обчислення $\sin\frac{\pi x}{2}$ за допомогою поліному Тейлора (1.33).

### 1.5.6 Ітераційні методи обчислення елементарних функцій

Нехай потрібно обчислити значення елементарної функції y= $f(x)$ при заданому аргументі $x = a$. Шукане значення функції позначимо як y = $f(a)$. Ітераційні методи допускають визначення послідовних наближень $y_1, y_2, \ldots, y_k, \ldots$ до шуканої величини $y$. При цьому наступне наближення $y_{k+1}$ знаходять по поточному наближенню $y_k$ за допомогою ітераційної формули виду:

$$y_{k+1} = F(y_k, a),\ k = 0, 1, \ldots \qquad (1.40)$$

де $F$ – деяка функція. Для реалізації ітераційного процесу (1.40) необхідно задати початкове наближення $y_0$ до шуканої величини – наприклад, грубу оцінку значення $y$. Процес припиняється, якщо виконується умова $| y_{k+1} - y_k | \le \varepsilon_y$, де величина $\varepsilon_y$ визначає необхідну абсолютну точність обчислення $y$.

Як приклад ітераційного методу розглянемо обчислення елементарної функції добування квадратного кореня з $x$. При $x = a$ ітераційний процес можна виразити формулою:

$$y_{k+1} = \frac{1}{2}\left( y_k + \frac{a}{y_k} \right),\ k = 0, 1, \ldots \qquad (1.41)$$

Формула (1.41) називається формулою Герона. У розділі 1.4.5 показано, що при добуванні квадратного кореня діапазон зміни аргументу може бути зведений до основного інтервалу $x \in [1;100]$. Тому в якості початкового наближення $y_0$ можна вибрати ціле число, квадрат якого найбільш близький до $a$.

Нехай, наприклад, необхідно обчислити $\sqrt{2}$, тобто $a = 2$. Точне значення $\sqrt{2}$ (з точністю 8 десяткових цифр) рівне $y$ = 1,4142136. Для використання формули Герона виберемо початкове наближення $y_0 = 1$ (величина $y^2 = 1$ найбільш близька до $a = 2$).

У таблиці 1.4 наведені значення послідовних наближень $y_k$, отримані за допомогою формули (1.41).

*Таблиця 1.4*

Послідовні наближення до величини $\sqrt{2}$

| Номер ітерації $k$ | Наближення $y_k$ |
|---|---|
| 0 | 1 |
| 1 | 1,000000 |
| 2 | 1,4166667 |
| 3 | 1,4142157 |
| 4 | 1,4142136 |

Як видно, послідовність наближень $y_k$ швидко сходиться до $y$. Для обчислення $\sqrt{2}$ з точністю більш, ніж 8 значущих цифр, потрібно лише чотири ітерації. Для обчислення цілого ряду складніших тригонометричних функцій, таких як синус чи косинус, можна використати ітераційний метод, відомий у вітчизняній літературі як “цифра за цифрою”, алгоритм Волдера або CORDIC у зарубіжних виданнях [1,10,40].

### 1.5.7 Принцип функціонування класичного методу CORDIC

Класичний метод CORDIC [131] ґрунтується на послідовному виконанні рекурентних рівнянь неявної матриці псевдоповороту вектора наступного виду:

$$\begin{bmatrix} x_n+1 \\ y_n+1 \end{bmatrix} = \underbrace{\begin{bmatrix} C & -S \\ S & C \end{bmatrix}}_{A_{ideal}} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \tag{1.42}$$

де значення S та C є наближеними значеннями синуса та косинуса деякого кута θ. У явному вигляді матрицю можна зобразити:

$$\begin{bmatrix} x_n+1 \\ y_n+1 \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{A_{ideal}} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \tag{1.43}$$

подібно до неї буде виглядати і матриця A для обрахунку гіперболічних синуса та косинуса:

$$A = \begin{bmatrix} \cosh & \sinh \\ \sinh & \cosh \end{bmatrix} \tag{1.44}$$

відрізняється вона лише додатніми значеннями величин, і у явному вигляді її можна записати як:

$$A = \rho \begin{bmatrix} \cosh(\theta) & \sinh(\theta) \\ \sinh(\theta) & \cosh(\theta) \end{bmatrix} \tag{1.45}$$

де

$$\rho = \sqrt{C^2 - S^2} \qquad \theta = \arctan h\left(\frac{S}{C}\right)$$

Оскільки при кожній операції повороту, довжина вектора ρ має тенденцію змінюватись у більшу сторону, то даний факт необхідно враховувати і задавати ще до першої ітерації алгоритму. Величина деформації вектора на кожній ітерації буде визначатись як:

$$\Delta_{\mathrm{mod}ul} = \rho_i - 1 \tag{1.46}$$

Подібним чином буде змінюватись і вхідний кут θ. Оскільки величина повороту на кожній ітерації не є круглою двійковою величиною, а лише арктангенсом кута повороту, то деформація внесена фактичним значенням кута ротації буде визначатись як:

$$\Delta_{angle} = \theta_i - 2^{-i} \tag{1.47}$$

де і — номер ітерації.

Таким чином, при використанні класичного Cordic методу, потрібно заздалегідь забезпечити коректне значення модуля вхідного вектора ρ, скорегованого на величину деформації згідно з вибраною кількістю необхідних ітерацій для досягнення бажаної точності. Цей факт не дає змогу виконувати меншу кількість ітерацій за ту, яка була наперед закладена модулем деформації, навіть у випадку, якщо поточний кут обертання на деякій ітерації точно відповідає шуканому куту. Аналогічна ситуація відбувається і з кутом мікроротації, що не дає можливість наперед визначити значення залишкового кута, яке утвориться після деякої кількості поворотів вектора. Ці дві умови не дозволяють зменшувати кількість ітерацій. Причому даний ефект спостерігається до того часу, поки значення деформації вектора чи кута мікроротації більші за молодший двійковий розряд обчислюваного аргументу. У випадку зменшення його до вказаних величин подальші ітерації алгоритму можна спростити, відійшовши від класичних ітераційних формул, на чому і ґрунтуються методи удосконалення алгоритму CORDIC.

Загалом класичний метод можна представити як набір рекурентних рівнянь наступного виду:

$$\begin{aligned} x_{i+1} &= x_i - \sigma_i \varepsilon_i y_i \\ y_{i+1} &= y_i + \sigma_i \varepsilon_i x_i \\ z_{i+1} &= z_i - \sigma_i \theta_i \end{aligned} \tag{1.48}$$

де - $i = 0,1,2,3,...,m$ , $\sigma_i = sign(z_i)$ , $\theta_i = arctg(\varepsilon_i)$ .

для розрахунку деформації, яка вноситься в результаті здійснення ітерацій згідно з поданою вище системою рекурентних рівнянь:

$$\begin{aligned} x_n &= \left(\sqrt{1+\varepsilon^2}\right)^n \cos(n\theta) \\ y_n &= \left(\sqrt{1+\varepsilon^2}\right)^n \sin(n\theta) \\ \theta &= arctg\varepsilon \end{aligned} \tag{1.49}$$

$$\begin{aligned} x_{m+1} &= K_m \left[ \; x_0 \cos\left(\sum_{i=0}^{m} \sigma_i \theta_i\right) - y_0 \sin\left(\sum_{i=0}^{m} \sigma_i \theta_i\right) \; \right], \\ y_{m+1} &= K_m \left[ \; y_0 \cos\left(\sum_{i=0}^{m} \sigma_i \theta_i\right) + x_0 \sin\left(\sum_{i=0}^{m} \sigma_i \theta_i\right) \; \right], \end{aligned} \tag{1.50}$$

$$K_m = \prod_{i=0}^{m} \rho_i = \prod_{i=0}^{m} \sqrt{1+\varepsilon_i^2} \tag{1.51}$$

Одержаний результат $K_m$ є коефіцієнтом деформації вектора, для сталої кількості ітерацій, і є незмінною величиною. При початковому значенні модуля вектора рівним:

$$x_0 = 1, \; y_0 = 0 \tag{1.52}$$

після здійснення усіх ітерацій, отримаємо наступний результат:

$$\begin{aligned} x_m &= K_m \cos\varphi \\ y_m &= K_m \sin\varphi \\ z_m &= \varphi_m \approx 0 \\ z_0 &= \varphi \end{aligned} \tag{1.53}$$

Щоб уникнути спотворення результату обчислень значень функції, доцільно при початку ітерацій замість значення $x_0$ взяти обернену до коефіцієнта деформації величину $\frac{1}{K_m}$. Тоді після завершення ітерацій отримаємо:

$$\begin{aligned} x_{m+1} &\approx \cos\varphi \\ y_{m+1} &\approx \sin\varphi \end{aligned} \tag{1.54}$$

Описаний алгоритм є класичним представленням алгоритму CORDIC, та в такому вигляді використовується як інженерами, так і програмістами для розроблення ПЗ, куди входить обчислення синуса та косинуса.

### 1.5.8 Способи обчислення гіперболічних функцій

При необхідності обчислити гіперболічний синус та косинус рекурентні рівняння набудуть наступного вигляду:

$$\begin{aligned} x_n &= \left(\sqrt{1-\varepsilon^2}\right)^n \cosh(n\theta) \\ y_n &= \left(\sqrt{1-\varepsilon^2}\right)^n \sinh(n\theta) \\ \theta &= \arctan h(\varepsilon) \end{aligned} \tag{1.55}$$

розв’язавши їх, одержимо:

$$\begin{aligned} x_{i+1} &= x_i + \sigma_i y_i 2^{-i} \\ y_{i+1} &= y_i + \sigma_i x_i 2^{-i} \\ z_{i+1} &= z_i - \sigma_i \theta_i \\ \sigma_i &= \begin{cases} -1\, if\ z_i < 0 \\ +1\, if\ z_i \geq 0 \end{cases} \\ i &= 1,2,3,4,4,5..12,13,13,14..m \\ \theta_i &= \arctan h\left(2^{-i}\right) \\ P'_{m1} &= \prod_{i=1}^{m} \cosh(\alpha_i) = \prod_{i=1}^{m} \frac{1}{\sqrt{1-2^{-2i}}} \end{aligned} \tag{1.56}$$

Як видно, рівняння є достатньо подібними до класичного методу CORDIC і відрізняються лише інверсією знаку повороту для ітерацій обчислення косинуса. Також початкове значення параметра “i” рівне одиниці, щоправда, це не зменшує складності обчислень, оскільки для забезпечення збіжності методу частину ітерацій доводиться повторювати. Коефіцієнт деформації також розраховується, виходячи з даного факту як:

$$P'_{m2} = \frac{1}{\sqrt{1-2^{-8}}}\frac{1}{\sqrt{1-2^{-26}}}\frac{1}{\sqrt{1-2^{-80}}}$$

$$P' = P'_{m1} \cdot P'_{m2}$$

$$K'_m = 1/P' \qquad (1.57)$$

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

До цього моменту були розглянуті випадки, коли вхідним невідомим аргументом виступав деякий заданий кут. Під час виконання ітерацій параметр z, який відіграє роль залишкового кута, зменшується та прямує до нуля, забезпечуючи необхідну точність обчислень. Даний режим з обчислення невідомого кута називається режимом обертання. Крім цього, режиму доступний також векторний режим, який полягає у здійсненні аналогічних ітерацій метода CORDIC, а роль шуканої величини відіграє значення $y_m$. В такому випадку при відомих вхідних значеннях величин $x_0$ та $z_0$, можна розраховувати обернені тригонометричні функції. Рівняння в цьому випадку будуть мати вигляд:

$$\sigma_i = \begin{cases} -1 & if\ y_i < 0 \\ +1 & if\ y_i \geq 0 \end{cases}$$

$$i = 0,1,2,...,m \qquad (1.58)$$

$$\theta_i = \arctan\left(2^{-i}\right)$$

а початкові умови аргументів визначаються, виходячи з формул:

$$x_{m+1} \approx K_m \sqrt{x_0^2 + y_0^2}$$
$$z_{m+1} \approx \arctan\left(\frac{y_0}{x_0}\right) \qquad (1.59)$$

За таким самим принципом можна отримати формули для обчислень гіперболічних функцій:

$$E = mc^2$$

$$\begin{aligned} x_{i+1} &= x_i - \sigma_i y_i 2^{-i} \\ y_{i+1} &= y_i - \sigma_i x_i 2^{-i} \\ z_{i+1} &= z_i + \sigma_i \theta_i \end{aligned}$$

$$\sigma_i = \begin{cases} -1 & if\ y_i < 0 \\ +1 & if\ y_i \geq 0 \end{cases} \tag{1.60}$$

$$i = 1,2,3,4,4,5..12,13,13,14..m$$

$$\theta_i = \arctan h\left(2^{-i}\right)$$

а початкові умови визначаються, виходячи з формул:

$$\begin{aligned} x_{m+1} &\approx K_m \sqrt{x_0^2 - y_0^2} \\ z_{m+1} &\approx \arctan h\left(\frac{y_0}{x_0}\right) \end{aligned} \tag{1.61}$$

## 1.6 Середовища розробки

### 1.6.1 Огляд середовища Quartus фірми Intel

Сьогодні актуальним середовищем для розробки програмного забезпечення для ПЛІС фірми Інтел є середовище Quartus. Даний продукт поширюється у двох варіантах (збірках): Web Edition та Subscription Edition. Перший варіант вільний для розповсюдження та використання і підтримує обмежене коло бюджетних ПЛІС. Версія Subscription Edition в свою чергу є платною та використовує усі можливості середовища Quartus. Також доступним є 30-денний пробний період використання даного ПЗ без підтримки компіляції файлу прошивки ПЛІС. Дане обмеження часто використовується на ринку ПЛІС, що дозволяє розробникам програмного забезпечення вивчити всі переваги пропонованої версії безпосередньо на існуючих проектах простим перенесенням їх наприклад з версії Web Edition. Оскільки під час роботи над дисертацією виникала необхідність перевірити коректність результатів компіляції, у деяких випадках крім вільної у доступі версії Web Edition використовувалась також Subscription Edition в режимі “evalution”. Такий підхід був викликаний еверестичними механізмами, які

використовуються компілятором, та результати роботи яких можуть сильно вплинути на основні характеристики проекту, такі як максимальна тактова частота, розмір проекту чи енергоспоживання у вихідному файлі. Оскільки, як було сказано, у пробній версії середовища неможливо згенерувати файл прошивки ПЛІС, це наклало певні обмеження на діапазон отримуваних результатів, таких як тепловиділення ПЛІС, яке визначається з використанням даного файлу. Причина даного обмеження є не однозначною, оскільки на практиці цей параметр успішно визначається у середовищі від конкуруючого виробника ПЛІС - Xilinx, із аналогічними обмеженнями по генерації прошивки. Ще один аргумент для використання платного середовища зумовлений його новішою версією, яка, у порівнянні з безкоштовною версією, для досліджуваної ПЛІС містить як удосконалені методи оптимізації проектів, так і ширший арсенал налаштувань, якими можна вплинути на якість фінальної реалізації алгоритмів. Також при використанні обидвох версій середовища можливе зіставлення результатів ідентичного програмного коду, що часто дає можливість зіставити оптимальність відкомпільованого проекту, та на скільки однозначною є реалізація написаного коду.

Основні версії середовища Quartus виходять двічі на рік з інтервалом у пів року. В цих версіях виробник включає підтримку нового обладнання а також забирає підтримку ПЛІС, які на його думку вже не актуальні. Основна ПЛІС, яка використовувалась для практичної перевірки алгоритмів у межах дисертаційного дослідження - CycloneIII, термін підтримки якого виробником зупинився на версії 13.1, та для якого було випущено чотири оновлення. Тому всі файли прошивок ПЛІС та практична перевірка алгоритмів призначених для CycloneIII були згенеровані версією Quartus 13.1.4 Web Edition. Під час написання дисертації були випущені версії середовища 14.0 - 17.1 Subscription Edition, які також використовувались у дослідженнях, але вже у якості допоміжного ПЗ.

Середовища, починаючи з 15.0.1, перш за все відрізняються від 13.1.4 рідною 64-бітною архітектурою, яка до цього була лише опцією. Даний факт

накладав певні обмеження під час інсталяції основного 32-розрядного пакета на 64-розрядній платформі. Для коректної інсталяції пакетів до версії 15.0 необхідно встановити у системі декілька 32-розрядних бібліотек та задовільнити ряд залежностей між пакунками. Версії середовища, починаючи з 15.0, можливо встановити лише на 64 розрядній платформі. Можливість запуску у 32-розрядному режимі - відсутня.

За допомогою поточних версій середовища Quartus здійснюється лише генерація програмного забезпечення ПЛІС без можливості верифікації його функціонування. Дана можливість була вбудована та доступна в попередніх версіях квартуса, та після виходу 10 версії продукту частину його функціоналу було усунуто. Серед суттєвих змін даної версії, крім відсутності симулятора можна також назвати припинення підтримки Інтелом (Альтерою) мови AHDL, що забирає у розробника можливість розробки схем на низькому рівні. Цей факт хоч і спрощує роботу програміста, перекладаючи значну частину роботи на компілятор, але й відбирає в нього можливість працювати з елементами структур схеми на рівні їхнього опису. Даний факт часто не давав можливості об’єктивно розібратись з причинами не характерної для компілятора поведінки, та проблематичність у заданні компілятору директив для роботи в тому руслі, що необхідно розробнику. У даний момент проекти для середовища Quartus можуть розроблятись двома мовами: VHDL, та Verilog, який в новішій версії стандарту класифікується як SystemVerilog.

Для здійснення верифікації проектів, незалежно від виробника та типу ПЛІС в більшості випадків використовується середовище ModelSim. Інсталяція даного пакету здійснюється автоматично разом із інсталяцією середовища Quartus, і доступна у двох пакетах: основного — ModelSim та спрощеного — ModelSim — IntelFPGA Edition, який на відміну від класичної версії, є безкоштовним, вільно поширюється і встановлюється по замовчуванню. Також можна відзначити, що існує версія ModelSim для ПЛІС фірми Xilinx - XilinxEdition. Суттєвих відмінностей у функціональності між повною та вільною версією продукту для

невеликих проектів, які будуть описуватись в межах даної дисертаційної роботи — немає. Проте враховуючи що продукт у даний час безперервно розвивається, однією із причин оновлення середовища Quartus було використання новіших та стабільніших версій пакету ModelSim. До одного з таких недоліків можна віднести критичну помилку при запуску програми через новішу версію бібліотеки libfreetype версії 2.5.2, через що доводилось понижувати версії бібліотек (64 та 32 розрядної) до 2.4.8. Даний недолік був виправлений лише у версії ModelSim, 10.3d, яка поширюється з версією Quartus15.0. Також нехарактерною для продукту такого рівня є неможливість запуску симулятора через наявність українських букв (кирилиці) у шляху до директорії з активним проектом.

Здійснювати верифікацію проектів у симуляторі можна як на логічному, так і на фізичному рівні. Для здійснення останнього потрібна підтримка середовищем компонентів у вигляді бібліотек з описом характеристик та часових затримок ПЛІС. Поточні версії симулятора містять необхідні для цього компоненти, але оскільки кінцева версія проекту проходить перевірку функціональності безпосередньо в кристалі, то на практиці здійснювалась лише логічна симуляція функціонування схеми. Причому лише після успішного тестування у кристалі, алгоритм признавався задовільним, а логічна симуляція здійснювалась через її простішу реалізацію в симуляторі, в. т. ч. за допомогою спеціально розроблених для цього скриптів, а також для більш наочної локалізації та усунення можливих помилок у роботі схеми. До переваг використання симулятора ModelSim можна віднести широку підтримку мов програмування; можливість здійснювати компіляцію проектів безпосередньо з джерельного коду (source code) значно швидше та ефективніше, через що більшість алгоритмів розроблялась саме у редакторі симулятора; а також можливість написання тестбенчів для алгоритмів у процесі їх розробки тією ж мовою — в даному випадку SystemVerilog, що й основний код програми. Єдиним суттєвим недоліком була відсутність підтримки директив компілятора Quartus, якими виробник ПЛІС (Intel) пропонує розширити функціонал мов програмування.

### 1.6.2 Середовище Vivado фірми Xilinx

Для роботи з ПЛІС фірми Xilinx виробник пропонує середовище Vivado. Його попередником було достатньо відоме середовище Xilinx ISE, остання версія якого 14.7 видана в жовтні 2013 року, і яке виробник припинив підтримувати, та помістив у заморожений стан (sustaining phase of its product life cycle). В подальшому не планується випуск його нових релізів. Достатньо проблематично отримати доступ до продукту Vivado, розміщеного на офіційному сайті з території України, незважаючи на його shareware ліцензію. Обмеження ґрунтується на законодавстві Сполучених Штатів про експорт — US Export Administration Regulations, де серед країн Європи, Україна разом з Росією та Білорусією поміщена в групу країн D, таких, які можуть нести загрозу національній безпеці США. Зв'язано це з тим, що фірма Xilinx постачає ПЛІС для силових структур, американської армії, НАСА, і.т.д. Слід мати на увазі, що у випадку разової фіксації фірмою Xilinx входу на сайт з країн із списку D або E [66] чи вказанні в профілі користувача будь-якої з цих країн доступ до усіх продуктів Xilinx буде заблокований згідно із законодавством Сполучених Штатів.

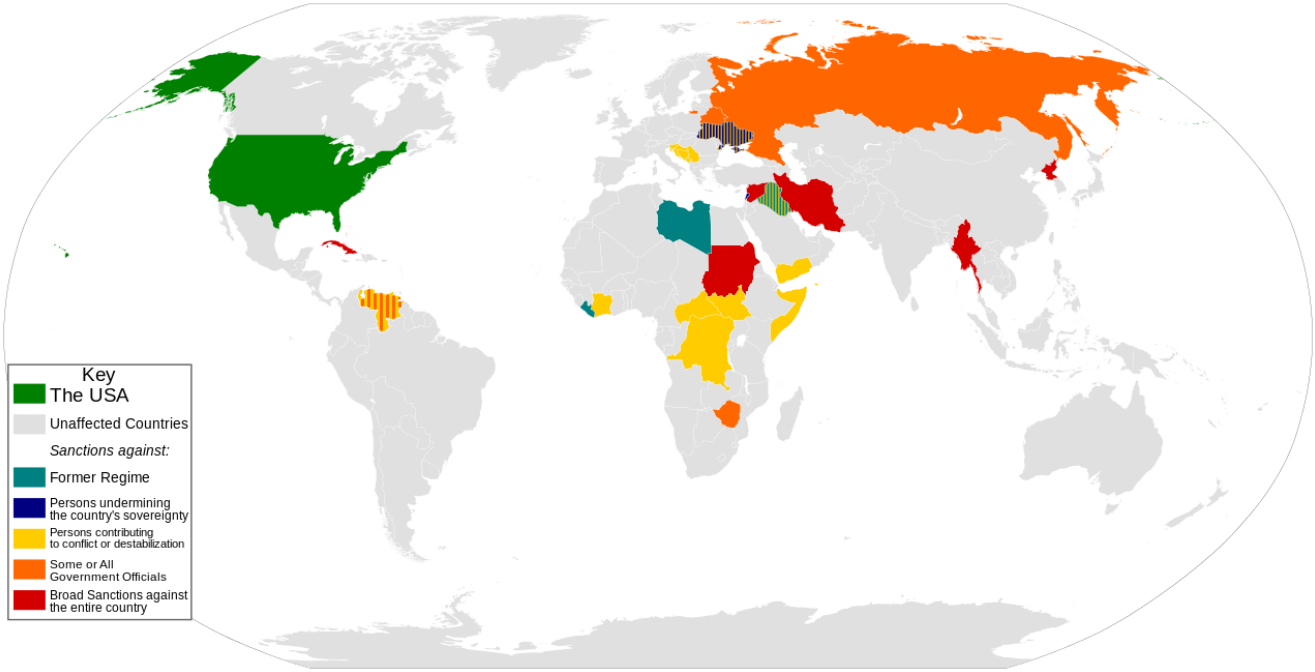



Рис 1.7 Країни, щодо яких США ввело ембарго, і, знаходячись у яких, ускладнено отримання програмних та апаратних продуктів фірми Xilinx

Фірма Xilinx здійснює маркування версій Vivado на основі поточного року, в якому відбувається їхній реліз. Як правило, випускаються чотири версії в рік, тоді як між щорічними версіями витримується пауза у півроку, після чого наступні три версії виходять в середньому що два місяці. При розробці програм, описаних у даній дисертації, використовувались версії Vivado з 2013.4 — 2017.4. При зіставленні результатів видно, що новіша версія здатна видати дещо кращі результати в порівнянні із попередніми версіями, хоча були зафіксовані випадки і зворотнього характеру. Незважаючи на те, що результат синтезу схем покращився, навряд чи можна назвати такий стан речей перевагою, оскільки немає можливості об’єктивно оцінювати характеристики алгоритмів без огляду на версію середовища. Крім того, немає впевненості щодо підвищення якості синтезованої схеми, а отримані кращі результати компіляції можуть бути пояснені зміною алгоритмів розрахунку характеристик схеми. Самі ж результати у всіх випадках виявились достатньо близькими, в тому числі і при так званих “провалах”, та не мали суттєвого впливу на загальну оцінку алгоритмів. Програми, розроблені у минулих, версіях були перекомпільовані для поточної версії для досягнення максимальної точності при зіставлені характеристик методів.

До переваг середовища Vivado можна віднести продуману, інтуїтивно зрозумілу ергономіку, порівняно невелику кількість додаткових налаштувань, вбудований симулятор, та більш функціональний набір вбудованих бібліотек - IP Core. Серед недоліків: невисока надійність середовища та його низька швидкодія, яка зумовлена написанням коду середовища мовою Tcl, з елементами Java, і яка орієнтована, насамперед, на крос-платформеність, через що дуже програє в показниках швидкодії. Наприклад, час компіляції та синтезу проекту може тривати протягом години, а в деяких випадках і довше. Ще одним недоліком став механізм визначення максимальної тактової частоти. Виходячи з принципів проектування в даному середовищі, наперед необхідно вказати бажану тактову частоту для проекту, і лише після синтезу можна зрозуміти, чи зумів компілятор вкласти проект у задані допустимі рамки. У результаті для визначення максимальної

тактової частоти одного проекту, необхідно здійснити більш, як десять компіляцій, через що складним є отримання оцінки швидкодії більш, як одного — двох проектів протягом робочого дня. Також в налаштуваннях компіляції та синтезу є доступними наперед визначені профілі з орієнтацією на тип задачі, для якої необхідно зробити акцент при компіляції. Враховуючи, що частина параметрів передається компілятору у вигляді параметрів командного рядка, який можна доповнити, зникає можливість точного визначення усіх параметрів, які були передані компілятору для досягнення максимального ефекту згідно із задачею компіляції. Даний факт буде наочно продемонстровано, коли кращий із двох пропонованих алгоритмів буде визначатися лише зміною однієї з опцій компілятора.

Використання усіх можливостей середовища Vivado, так само як для середовища Quartus, можливе протягом 30-денного періоду без опції по генерації файлу конфігурації. Отримати таку можливість для бюджетного сімейства ПЛІС доступно шляхом реєстрації на сайті Xilinx та отримання 30-денної ліцензії з прив’язкою до конкретного комп’ютера.

### 1.6.3 Середовища розробки для програм для ПК та AVR

У написанні програм для інших платформ, таких, як ПК, AVR, використовувались середовища Visual Studo, та AVR Studio у Windows. Також використовувався пакет Masm, якщо частину коду потрібно було написати у 64-розрядному асемблері, тоді як середовище Visal Studio підтримує компіляцію лише 32-розрядних програм з асемблерними вставками. Дане обмеження відсутнє для компілятора gcc, який є стандартним компілятором для Linux. А відлагодження 64-розрядних проектів мовою асемблер можна здійснити в будь-якому доступному середовищі, серед яких були NetBeans, Eclipse, Code::Blocks. Причому для проектів з асемблерними вставками по якості роботи з дебагером зручнішим був NetBeans, тоді як для проектів, написаних виключно мовою С,

використовувався Code::Blocks. Щодо модулів, написаних на асемблері, то їх компіляція здійснювалася вбудованим GNU-assembler. Особливістю асемблера платформи Unix є його синтаксис AT&T, який суттєво відрізняється від синтаксису Intel, що використовується в більшості сучасних асемблерів. Зумовлено це глибшою історією транслятора від AT&T, який зберігся на даній платформі до наших днів. Ще одним компілятором, який використовувався для дослідження швидкодії алгоритмів, був TASM. Програми, написані для операційних систем реального режиму та лише з прямою взаємодією із апаратурою комп'ютера задля досягнення максимально стабільних результатів. Особливістю таких програм, згенерованих в .com форматі міг бути їхній запуск як за допомогою засобів ОС, так і без них. Виконувався даний прийом шляхом запису бінарних даних у завантажувальний чи MBR сектор диску, або флеш-накопичувача. Зрозуміло, що працювати такі програми в багатозадачному режимі сучасних ОС не будуть, а при використанні емулятора віртуальної машини, втрачається їх основне призначення — досягнення максимальної швидкодії та можливості точного його заміру.

## Висновки першого розділу

1. Вибрано ітераційний метод псевдоповороту вектора CORDIC як один із перспективних для проведення подальших досліджень метод. При цьому в ньому присутній значний потенціал для удосконалення використовуючи математичні прийоми та програмні і апаратні засоби.
2. Базуючись на принципах роботи ітераційних методів можна констатувати, що основним їхнім недоліком є висока латентність, яка може бути зменшена, у більшості функцій, для яких застосовуються такі методи, як CORDIC.
3. Не всі методи наближень можна ефективно використати на сучасній цифровій техніці. Частина методів може бути відкинута на етапі розробки

через відсутність у цільової платформи програмних чи апаратних операцій, які вимагає алгоритм. Також неефективними можуть виявитись методи з операціями множення, ділення, та іншими трудомісткими діями.

4. Наближаючи деякі з функцій поліномами високої розрядності можна здійснювати їх факторизацію таким чином, щоб оптимізувалось розміщення ключових блоків у схемі та скорочувався час для обчислення функції в цілому.

# РОЗДІЛ 2
# Розвиток ітераційного методу CORDIC та паралельних поліноміальних обчислень

## 2.1 Вступні зауваження

Для здійснення обчислень елементарних функцій, таких як синус, косинус, тангенс, квадратний корінь чи експонента на апаратному та частково на програмному рівні значна частина виробників напівпровідникової техніки здебільшого використовує ітераційні методи. В основі обчислень лежить класичний метод CORDIC, який, незважаючи на лінійну збіжність, є оптимальним серед інших методів при його апаратній реалізації. Кожна стадія класичного метода CORDIC дає на виході один правильний розряд обчислювальної функції. Таким чином, кількість тактів, необхідних для обчислень класичним методом, завжди перевищує кількість біт розрядної сітки вихідних регістрів апаратного забезпечення. Виконання додаткових ітерацій зумовлено як наявністю похибок заокруглення ірраціональних чисел, які виходять за межі розрядної сітки внутрішніх регістрів, так і для здійснення масштабування вхідних та вихідних даних, для приведення їх до робочого діапазону метода. Так, для обчислень значень синуса та косинуса на будь-якому персональному комп’ютері знадобиться приблизно 100 тактів. Точне значення кількості тактів можна отримати безпосередньо з документації на процесор. Слід мати на увазі, що взяті безпосередньо з офіційних документів значення можуть виявитись неточними. Адже виробник здатен “відредагувати” структуру процесора в межах одного сімейства, а в деяких випадках і при зміні степінгу процесора, через що буде змінена швидкодія застосовуваних методів. Незважаючи на це, результат навряд чи буде суттєво відрізнятись від офіційно заявленого. Проте наявність даної невідповідності має місце, і до неї також потрібно бути готовим при зіставленні отримуваних даних у випадку практичних досліджень. Так, для процесорів сімейства K7, відомих як Athlon XP фірми AMD, кількість тактів для обчислень

синуса та косинуса буде становити 120 тактів. Для 64-розрядних процесорів K8, відомих як Athlon64, 104 такти. Розрядність блоку для операцій з рухомою комою у копроцесорах [9] архітектури x87 становить 80 біт [50]. Відповідно кількість тактів та розрядність співвідносяться як 106 до 80, що дає 32,5% надлишковості. До цього слід додати, що спосіб обчислення даних функцій (синуса, косинуса, тангенса) не змінювався ще з часів появи першого x87 сумісного копроцесора у 1978 році, так само як і його системи команд [77]. Загалом, усі команди, підтримувані блоком операцій з рухомою комою (не враховуючи технічних команд, що з'явились у процесорі у зв'язку з переходом у багатозадачний "захищений" режим роботи) які є у сучасному ПК і були присутні в копроцесорі 40-річної давності. Хоча деякі елементи представлень даних і були доповнені, такі як розширення діапазону вхідних аргументів чи відсутність необхідності у часткових результатах для вихідних даних, всі вони вже були присутні у копроцесорі x387 1985р. З виходом наступної серії 486 процесорів, блок для обчислень з рухомою комою є інтегрований у ЦП та не зазнає видимих змін вже протягом 30 років. Операції ж з обчислення синуса-косинуса-тангенса виконуються найдовше не тільки для блоку з рухомою комою, але й серед усіх команд арифметичних обчислень сучасного процесора.

Подібна ситуація спостерігається серед ПЛІС, де два провідні виробники на ринку SoC Xilinx та IntelFPGA пропонують для обчислень елементарних функцій використання класичного CODICa. Цей метод входить як у вбудовані бібліотеки IP Core середовищ Vivado та Xilinx ISE, так і до складу Quartus'a у вигляді мегафункції. В обох випадках при обчисленнях 32-розрядного синуса-косинуса знадобиться 36 тактів, оперуючи в середині алгоритму 48-розрядними цілочисельними регістрами. За цими параметрами обидві платформи достатньо близькі, хоча IP Core фірми Xilinx має значно кращу функціональність та швидкодію у порівнянні з мегафункцією від IntelFPGA. Незважаючи на це, для обчислень в обох випадках використовується класичний алгоритм Cordic із надлишковістю кількості тактів до розрядності отримуваних даних, як 36 до 32,

що становить 12.5%. Даний показник, звичайно, кращий за результат x86 копроцесора і пояснюється нижчою похибкою, викликаною меншою кількістю ітерацій. Крім того існують підстави до припущення того факту, що розрядність внутрішніх регістрів у копроцесорі практично не перевищує розрядності отримуваних даних, як і необхідність зводити вхідні результати для методу у значно ширших межах.

Якщо говорити про платформи з обмеженими апаратними можливостями, то у мікроконтролерах без підтримки апаратного множення, метод CORDIC не має конкурентів. А у випадку використання апаратного множення, при збільшенні розрядності обчислень стають очевидними переваги методу CORDIC над апроксимаційними методами на основі поліномів високих порядків.

Виходячи з вищенаведеного, можна констатувати, що провідні фірми, виробники апаратного забезпечення, такі як AMD, Atmel, Intel, Xilinx станом на сьогодні, використовують метод CORDIC, як основу для обчислення цілого ряду функцій, незважаючи на лінійну збіжність ітераційних процесів в ньому. Тому основою подальших досліджень стане розробка та практична реалізація способів обчислення елементарних функцій за допомогою модифікації методу CORDIC. При цьому наголос буде зроблений на зменшенні кількості ітерацій, необхідних для обчислень функцій за рахунок використання додаткових функціональних блоків та асинхронних режимів роботи алгоритму.

## 2.2 Метод перекодування кута

Далі пропонується новий метод перекодування вхідного кута, який дає змогу гнучко змінювати об’єм пам’яті таблиці та число ітерацій CORDICa. Так, для класичного методу CORDIC, основним недоліком є низька швидкодія через лінійну збіжність методу (не більше одного опрацьованого розряду вхідного аргументу за одну ітерацію) та відносна апаратна складність, пов’язана з необхідністю реалізації одночасно трьох ітераційних рівнянь (для $x_i$, $y_i$, $z_i$) у

випадку застосування конвеєрної структури обчислювача (1.56).

З метою спрощення апаратної реалізації обчислювача пропонується використання методу CORDIC з перекодуванням вхідного кута, що дає змогу звести систему (1.48) лише до двох ітераційних рівнянь для $x_i$, та $y_i$. Одночасно з цим для підвищення швидкодії методу, шляхом зменшення числа ітерацій, розроблено гібридні структури, що використовують послідовно три методи: табличний метод, CORDIC та залишкове множення. Найпростішим з точки зору апаратного втілення є CORDIC із класичним перекодуванням вхідного кута [98]. Однак він має суттєвий недолік – великий обсяг пам'яті переглядових таблиць, і при великих значеннях m - необхідна таблиця розміром, не меншим, ніж $2^{m/3}\cdot m$ розрядів. Крім того, вихідні помножувачі в класичному варіанті реалізовані у базисі {-1,1}, що ускладнює використання помножувачів, які присутні у блоках DSP сучасних ПЛІС.

Опишемо новий метод знакозмінного перекодування вхідного кута, який дає змогу гнучко змінювати об'єм пам'яті таблиці та число ітерацій CORDICa.

Нехай маємо двійкове m - розрядне представлення кута φ:

$$\varphi = \sum_{i=1}^{m} a_i 2^{-i}, \tag{2.1}$$

причому $a_i = \{0,1\}$, $\varphi \in [0, \pi/4]$.

Необхідно знайти функції sin(φ) та cos(φ), тобто повернути одиничний вектор на кут φ. Подамо φ у вигляді:

$$\varphi = \varphi_{var} + \Delta \tag{2.2}$$

Тут $\varphi_{var}$ - кут повороту, отриманий у результаті перекодування кута φ, а Δ-кут відставання, який з'являється у зв'язку з перекодування. Перекодування кута φ можна описати такими рівняннями:

$$\varphi_{var} = \varphi_r + \varphi_{const}, \tag{2.3}$$

де

$$\varphi_r = \sum_{i=2}^{m+1} b_i \cdot \arctan\left(2^{-i}\right), b_i = 2\, a_{i-1} \tag{2.4}$$

Після очевидних перетворень отримаємо, що:

$$\varphi_{var}=\sum_{i=2}^{m+1} a_{i-1}\cdot\arctan\left(2^{-i}\right)=\sum_{i=1}^{m} 2\,a_i\cdot\arctan\left(2^{-i-1}\right) \tag{2.5}$$

$$\varphi_{const}=\sum_{i=2}^{m+1}\arctan\left(2^{-i}\right)=\sum_{i=1}^{m} acrtan\left(2^{-i-1}\right) \tag{2.6}$$

Для повороту одиничного вектора на кут $\varphi_{var}$ застосуємо метод CORDIC з такими ітераційними рівняннями:

$$\begin{gathered} x_i=x_{i-1}-b_i\,y_{i-1}\cdot 2^{-i-1};\\ y_i=y_{i-1}-b_i x_{i-1}\cdot 2^{-i-1};\\ b_i=2\,a_i-1\\ i=\overline{1,m}\\ x_0=P_c\cos\left(\varphi_{const}\right)\\ y_0=P_c\sin\left(\varphi_{const}\right)\\ P_c=\frac{1}{\prod_{i=1}^{m}\sqrt{1+2^{-2i-2}}} \end{gathered} \tag{2.7}$$

У результаті отримаємо синус - $y_m$ та косинус - $x_m$ кута $\varphi_{var}$. Кут відставання $\Delta$ знаходиться за такими формулами:

$$\begin{gathered} \Delta=\varphi-\varphi_{var}=\sum_{i=1}^{m} a_i d_i\\ d_i=2^{-i}-2\arctan\left(2^{-i-1}\right) \end{gathered} \tag{2.8}$$

Особливістю такого представлення кута відставання є те, що він завжди додатній або рівний нулю. Зауважимо, що $\Delta$ - це кут, на який потрібно додатково повернути вектор $x_{m+1}$, $y_{m+1}$, отриманий після завершення ітерацій за методом CORDIC. Одержані в результаті такого додаткового обертання значення “x” та “y” і будуть шуканими функціями косинуса і синуса кута $\varphi$:

$$\begin{gathered} x=x_m\cos(\Delta)-y_m\sin(\Delta),\\ y=y_m\cos(\Delta)+x_m\sin(\Delta) \end{gathered} \tag{2.9}$$

Оцінимо число ітерацій, в межах яких слід брати до уваги значення $\Delta$. Для цього поставимо умову:

$$\begin{gathered} d_i\le 2^{-m}\text{, тоді}\\ d_i=2^{-i}-2\arctan\left(2^{-i-1}\right)\approx 2^{-i}-2\left(2^{-i-1}-2^{-3i-3}/3\right)=2^{-m} \end{gathered} \tag{2.10}$$

звідки знайдемо, що:

$$i_a=\lceil\frac{m-2-\log_2 3}{3}\rceil \tag{2.11}$$

Саме при такому $i=i_a$ найбільша складова кута відставання завжди буде меншою від $2^{-m}$, тобто $d_i<2^{-m}$.

Нижче наведена таблиця значень $i_a$ для різних m.

*Таблиця 2.1*

Мінімальне значення кількості біт кута відставання для різних розрядностей

| | |
|---|---|
| m=16 | $i_a=5$ |
| m=24 | $i_a=7$ |
| m=32 | $i_a=10$ |
| m=40 | $i_a=13$ |
| m=48 | $i_a=15$ |
| m=54 | $i_a=17$ |
| m=64 | $i_a=21$ |

Звідси випливає, що Δ слід обчислювати лише у межах значень “i”:

$$\Delta=\sum_{i=1}^{i_{a-1}} a_i d_i \tag{2.12}$$

Якщо ж в обчислювачі використовуються вихідні помножувачі для кусково - лінійної апроксимації, то необхідно, щоб найбільша складова $d_i$ кута відставання Δ не перевищувала значення:

$$d_i\le 2^{\frac{-m}{2}}, \text{або}$$
$$2^{-i}-2\arctan\left(2^{-i-1}\right)\le 2^{\frac{-m}{2}} \tag{2.13}$$

Тоді неважко знайти, що:

$$i_b=\lceil\frac{m-4-2\log_2 3}{6}\rceil \tag{2.14}$$

У таблиці 2.2 наведені значення $i_b$ для різних m.

*Таблиця 2.2*

Номери вхідних розрядів, для яких похибка при обчисленні залишковим множенням не вийде за межі розрядної сітки

| | |
|---|---|
| m=16 | $i_b$=2 |
| m=24 | $i_b$=3 |
| m=32 | $i_b$=5 |
| m=40 | $i_b$=6 |
| m=48 | $i_b$=7 |
| m=54 | $i_b$=8 |
| m=64 | $i_b$=10 |

Отже, біти вхідного кута будуть оброблятися таким чином: старші з номерами $i1=\overline{1,i_{LUT}}$ де $i_{LUT}=\overline{i_b-1,\ldots,i_a-1}$ - через використання пам'яті, під CORDIC відводяться біти з номерами $i2=\overline{i_{LUT}+1,m/2}$ а для методу залишкового множення – молодші біти з номерами $i3=\overline{\frac{m}{2}+1,m}$

## 2.3 Метод беззнакового перекодування кута.

Далі у роботі викладено новий метод для обчислення синуса-косинуса, придатний для реалізації на апаратних платформах з обмеженими обчислювальними ресурсами – мікроконтролерах, ПЛІС і т. п. Поданий математичний опис алгоритму, який можна класифікувати як гібридний метод CORDIC із застосуванням принципу перекодування кута.

Відомо, що у загальному випадку обертання по колу деякого вектора, заданого координатами $x_{i-1}$,$y_{i-1}$, на кут мікрообертання $a_i$ можна подати як:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} cos\alpha_i & -sin\alpha_i \\ sin\alpha_i & cos\alpha_i \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} = cos\alpha_i \begin{bmatrix} 1 & -\tan(\sigma_i a_i) \\ \tan(\sigma_i a_i) & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \quad (2.15)$$

де $\sigma_i=\{-1,+1\}$ – вказує напрямок обертання вектора (у даному випадку – двостороннє обертання за або проти годинникової стрілки).

Існують два популярні методи апаратної реалізації такого обертання:

-метод Волдера (відомий як CORDIC- метод) [131,132];

-метод Мадісетті-Квентуса-Вілсона (назвемо його МКВ-метод)[98,123].

У CORDIC - методі кут мікрообертання вибирається з умови що $a_i = \arctan(2^{-i})$, звідси $\tan(\sigma_i a_i) = \sigma_i 2^{-i}$. Тоді рівняння (2.15) матиме вигляд:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \cos(\arctan(2^{-i})) \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \tag{2.16}$$

Як бачимо, тут множення на $\tan(\sigma_i a_i)$ замінюється простою операцією зсуву на 'i' розрядів вправо, а $\sigma_i$ визначає тип операції – віднімання чи додавання. Основне вузьке місце традиційного CORDIC – методу, яке не дозволяє виключити з ітераційних рівнянь дві змінні $z_i$ та $\sigma_i$ і таким чином спростити архітектуру пристрою, є послідовне обчислення $\sigma_i$, яке не піддається попередньому передбаченню.

Тепер розглянемо МКВ-метод. Тут кут мікрообертання складає $\alpha_i = 2^{-i}$, а $\tan(\sigma_i \alpha_i) = \tan(\sigma_i 2^{-i})$. Крім того, у даному методі вхідний кут φ перекодовується у кут $\varphi_r$:

$$\begin{gathered} \varphi_r = \varphi_{const} + \varphi \ , де \\ \varphi_{const} = 2^{-1} - 2^{-m-1}, \varphi_r = \sum_{i=2}^{m+1} b_i 2^{-i}; b_i = 2a_{i-1} - 1 \end{gathered} \tag{2.17}$$

У цьому випадку $\sigma_i = b_i = (+1, -1)$, а ітераційні рівняння мають вигляд:

$$\begin{gathered} x_i = x_{i-1} - b_i y_{i-1} \cdot \tan(2^{-i}) \\ y_i = y_{i-1} - b_i x_{i-1} \cdot \tan(2^{-i}) \\ i = 2,\ldots, m+1 \end{gathered} \tag{2.18}$$

Початкові умови є такими:

$$\begin{gathered} x_1 = P_m \cdot \cos(\varphi_{const}) \\ y_1 = P_m \cdot \sin(\varphi_{const}) \\ P_m = \prod_{i=2}^{m+1} \cos(2^{-i}) \end{gathered} \tag{2.19}$$

$P_m$ - масштабуючий коефіцієнт МКВ-методу.

Тут враховано, що $\tan(\sigma_i 2^{-i}) = \sigma_i \tan(2^{-i})$.

Як бачимо, з ітераційного процесу (2.18) вилучено обчислення $z_i$ та $\sigma_i$ (у порівнянні з формулами (2.15) традиційного CORDICa) – і у цьому основна перевага МКВ – методу. Як і раніше $\sigma_i = b_i$ визначає тип операції – додавання чи віднімання. Однак, є серйозна проблема - множення на $\tan(2^{-i})$, яке дуже складно реалізувати апаратно.

Для того, щоб усунути цей недолік на практиці застосовують два прийоми – використання переглядових таблиць (LUT – look up table) на ранніх стадіях ітераційного процесу (2.18), коли $\tan(2^{-i}) \neq 2^{-i}$ [98,123], і об'єднання двох методів – CORDIC та МКВ в один з наступним розкладом кутів мікрообертань $2^{-i}$ на суми арктангенсів [67,84,85,88]. Розглянемо окремо кожен з цих прийомів.

МКВ – метод з використанням LUT.

У роботах [98,123] доведено, що для $i \geq \lfloor \frac{m}{3} \rfloor, \tan(2^{-i}) \approx 2^{-i}$, тобто у цьому випадку справджується умова – $\left|\tan(2^{-i}) - 2^{-i}\right| < 2^{-m}$. А для $i < \lfloor \frac{m}{3} \rfloor$ результати обчислень на кожній з ітерацій об'єднуються і замінюються на значення синусів та косинусів, які зчитуються з LUT. Таким чином, ітераційні рівняння (2.18) наберуть наступного вигляду:

$$\begin{gathered} x_i = x_{i-1} - b_i y_{i-1} 2^{-i} \\ y_i = y_{i-1} + b_i x_{i-1} 2^{-i} \\ i = \lfloor \frac{m}{3} \rfloor + 1, \ldots m+1; b_i = 2a_{i-1} - 1 \end{gathered} \tag{2.20}$$

а початкові умови будуть такими:

$$\begin{gathered} x\left[\lfloor \frac{m}{3} \rfloor\right] = P_m \cdot \cos(\varphi_{const} + \varphi_1) \quad y\left[\lfloor \frac{m}{3} \rfloor\right] = P_m \cdot \sin(\varphi_{const} + \varphi_1) \\ \varphi_{const} = 2^{-\lfloor \frac{m}{3} \rfloor - 1} - 2^{-m-1} \quad \varphi_1 = \sum_{i=1}^{\lfloor \frac{m}{3} \rfloor} a_i 2^{-i} \end{gathered} \tag{2.21}$$

де $P_m$:

$$P_m = \prod_{i=\lfloor \frac{m}{3} \rfloor + 1}^{m+1} \cos(2^{-i}) \tag{2.22}$$

Основним недоліком такого підходу є великі обсяги пам'яті таблиць типу LUT при великих значеннях m. Щодо переваг, то, як випливає з раніше викладеного матеріалу, основною перевагою МКВ-методу є наперед визначене значення $\sigma_i$ ітераційного процесу (2.18), що досягається з допомогою перекодування (2.17) вхідного кута φ. Якщо використати таке ж перекодування в методі CORDIC, то отримаємо наступну систему рекурентних рівнянь:

$$\begin{gathered} x_i = x_{i-1} - b_i\, y_{i-1} 2^{-i} \\ y_i = y_{i-1} + b_i\, x_{i-1} 2^{-i} \\ i = 2,\ldots,m+1;\; b_i = 2a_{i-1} - 1 \end{gathered} \tag{2.23}$$

при початкових умовах:

$$\begin{gathered} x_1 = P_c \cdot \cos(\varphi_{const}) \\ y_1 = P_c \cdot \sin(\varphi_{const}) \\ P_c = \prod_{i=2}^{m+1} \frac{1}{\sqrt{1+2^{-2i}}}. \end{gathered} \tag{2.24}$$

Очевидно, що на кожній ітерації кут $\alpha_i$ повороту (мікрообертання) вектора з координатами $\{x_{i-1}; y_{i-1}\}$ буде складати $\alpha_i = \arctan(2^{-i})$, у той час як згідно з (2.18) він повинен був би дорівнювати $2^{-i}$. Таким чином, на кожній ітерації буде відставання кута повороту на величину:

$$d_i = b_i\left[2^{-i} - \arctan(2^{-i})\right] \tag{2.25}$$

У результаті проведення усіх m - ітерацій (i=2…m +1) сумарний кут відставання становитиме:

$$\Delta = \sum_{i=2}^{m+1} d_i = \sum_{i=2}^{m+1} b_i\left[2^{-i} - \arctan(2^{-i})\right] \tag{2.26}$$

Отже, якщо провести ітерації за формулами (2.23) з початковими умовами (2.24), то після завершення цих ітерацій необхідно додатково повернути вектор $\{x_{i+1}; y_{i+1}\}$ на кут Δ для того, щоб мати повний поворот початкового вектора на кут φ і таким чином отримати шукані значення sin(φ) та cos(φ). Тут можуть бути використані два підходи для додаткових обертань:

1. розклад $d_i$ на суму $\arctan(2^{-i})$ і проведення додаткових ітерацій (без використання пам'яті типу LUT). Саме на такому принципі працюють

алгоритми [84]. Однак додаткові ітерації створюють додаткові затримки і вимагають додаткових апаратних затрат. Наприклад, для m = 64 потрібно додатково провести 39 ітерацій.

2. спроби об'єднати деякі мікрообертання [85] хоч і підвищують швидкодію, однак, не дають суттєвого покращення результатів. Дещо інші підходи можна знайти у [67,88], але вони кардинально не змінюють суті справи.

У пропонованому методі перекодування кута пропонується зменшення обсягів переглядових таблиць при обчисленні функцій синуса та косинуса. Це досягається за рахунок застосування гібридного CORDIC–методу обчислень, описаного у [25,26,104], із використанням принципу перекодування кута, який детально розглянутий вище. Вхідний кут $\varphi$ розбивається на три кути :

$$\begin{aligned} \varphi &= \varphi_1 + \varphi_2 + \varphi_3 \\ \varphi_1 &= \sum_{i=1}^{m_{LUT}} a_i 2^{-i} \\ \varphi_2 &= \sum_{i=m_{LUT}+1}^{m_{CORDIC}} a_i 2^{-i} \\ \varphi_3 &= \sum_{i=m_{CORDIC}+1}^{m} a_i 2^{-i} \end{aligned} \tag{2.27}$$

Тут $m_{LUT}$ – число старших бітів кута φ, що подаються на переглядову таблицю LUT, яка виконує функцію:

$$\begin{aligned} x_{m_{LUT}+1} &= P_c \cdot \cos\left(\varphi_1 + \varphi_{2const}\right) \\ y_{m_{LUT}+1} &= P_c \cdot \sin\left(\varphi_1 + \varphi_{2const}\right) \\ \varphi_{2const} &= 2^{-(m_{LUT}+1)} - 2^{-(m_{CORDIC}+1)} \end{aligned} \tag{2.28}$$

$m_{CORDIC}$ – число середніх бітів $\left(i_{CORDIC} = m_{LUT}+2 \ldots m_{CORDIC}+1\right)$, які обробляються за методом CORDIC (біти кута $\varphi_2$).Значення $m_{CORDIC}$ вибирається з умови $m_{CORDIC} = \lceil \frac{m}{2} \rceil$ [25,63,125]. Причому саме в CORDICу кут $\varphi_2$ перекодовується в кут $\varphi_{2r}$:

$$\varphi_{2r} = \sum_{i=m_{LUT}+2}^{m_{CORDIC}} b_i 2^{-i} = \sum_{i=m_{LUT}+2}^{m_{CORDIC}} \left(2a_{i-1} - 1\right) \cdot 2^{-i} \tag{2.29}$$

де $\varphi_{2const}$ визначається за формулою (2.28), так що $\varphi_2 = \varphi_{2r} - \varphi_{2const}$. Після завершення ітерацій за формулами:

$$\begin{gathered} x_i = x_{i-1} - b_i y_{i-1} 2^{-i} \\ y_i = y_{i-1} + b_i x_{i-1} 2^{-i} \\ i = m_{LUT}+2, \ldots, m_{CORDIC}+1 \end{gathered} \tag{2.30}$$

отримаємо вектор з координатами $\{x_{m_{CORDIC}+1}; y_{m_{CORDIC}+1}\}$. Цей вектор здійснює додатковий поворот на залишковий кут $\varphi_{зал}$, який дорівнює скоригованому куту $\varphi_3$ на кут $\Delta$:

$$\varphi_{зал} = \varphi_3 + \Delta \tag{2.31}$$

де $\Delta$ визначається за формулою, подібною до формули (2.26):

$$\Delta = \sum_{m_{LUT}+2}^{m_3} d_i = \sum_{m_{LUT}+2}^{m_3} b_i \left[2^{-i} - \arctan\left(2^{-i}\right)\right] \tag{2.32}$$

тут $m_3 = \lfloor \frac{m}{3} \rfloor - 1$

Далі виконуються дві дії, подібно до алгоритму, описаному в [104] – множення на кут $\varphi_{зал}$:

$$x_{m_{CORDIC}+2} = x_{m_{CORDIC}+1} - \varphi_{зал} \cdot y_{m_{CORDIC}+1} \quad y_{m_{CORDIC}+2} = y_{m_{CORDIC}+1} + \varphi_{зал} \cdot y_{m_{CORDIC}+1}. \tag{2.33}$$

Однак існує обмеження на значення кута відставання $\Delta$ для того, щоб мати можливість виконати поворот (2.33) з відповідною точністю обчислень.

Необхідною умовою виконання (2.33) є згідно з (2.15) набуття значень $\cos\left(\varphi_{зал}\right) \approx 1$, $\sin\left(\varphi_{зал}\right) \approx \varphi_{зал}$ з точністю до m бітів. Кут $\varphi_3$ відповідає цій умові, оскільки:

$$\cos\varphi_3 \approx 1 - \frac{\varphi_3^2}{2} \quad \sin\varphi_3 \approx \varphi_3 - \frac{\varphi_3^3}{6} \tag{2.34}$$

Для $\varphi_3 \le 2^{\frac{-m}{2}}$ складова $\frac{\varphi_3^2}{2}$ у найгіршому випадку становитиме $2^{m-1}$, що буде вдвічі меншим за прийняту похибку обчислень в “m” двійкових розрядів. Таким чином, необхідно обмежити кут відставання $\Delta$ до цього ж значення. Розкладемо функцію $\arctan(2^{-i})$ в ряд Тейлора з виразу (2.25):

$$\begin{gathered} d_i = b_i \left[2^{-i} - \arctan\left(2^{-i}\right)\right] \\ \arctan\left(2^{-i}\right) \approx 2^{-i} - \frac{2^{-3i}}{3} \end{gathered} \tag{2.35}$$

і висунемо умову:

$$\left|2^{-i}-\left(2^{-i}-\frac{2^{-3i}}{3}\right)\right|\le 2^{\frac{-m}{2}} \tag{2.36}$$

звідси знайдемо:

$$i=\lceil\frac{m-2\log_2(3)}{6}\rceil \tag{2.37}$$

очевидно, що знайдене значення "i" визначає мінімальне значення $m_{LUTmin}$:

$$m_{LUTmin}=i-1, \tag{2.38}$$

яке і обмежить найбільше значення кута Δ. В таблиці 2.3 наведені діапазони значень $m_{LUT}$ для різних значень m.

Таблиця 2.3

Число старших бітів mLUT вхідного кута φ, поданих на адресний вхід переглядової таблиці LUT

| m біт | $m_{LUT}$ - пропоноване | $m_{LUT}$ - класичне |
|---|---|---|
| 16 | 2...4 | 4 |
| 24 | 3...7 | 7 |
| 32 | 4...10 | 10 |
| 48 | 5...15 | 15 |
| 54 | 7...17 | 17 |
| 64 | 9...20 | 20 |

Як випливає з поданої таблиці 2.3 запропонований метод значно розширює межі значень $m_{LUT}$ у порівнянні з відомими результатами робіт [98,123]. Причому, якщо для "класичного перекодування" кількість розрядів таблиці є жорстко фіксованою, то вказані в таблиці 2.3 діапазони можна додатково зменшити за рахунок помножувачів. Із формули (2.25) випливає, що сума значень $d_i$ не повинна перевищувати кількості розрядів, відведених для кусково-лінійної апроксимації. Але дане застереження не є строгим, і при необхідності та архітектурній можливості доцільним є подальше скорочення таблиці LUT, що і було прийнято на практиці. Так для ПЛІС, де блок помножувача здатний функціонувати в режимі 18x18, було скорочено розмір таблиці LUT з $2^4$ до $2^3$ при m рівним 32-м розрядам за рахунок розширення діапазону корекції залишкового множення, яке за замовчуванням становить 17 розрядів.

## 2.4 Обчислення гіперболічних та експоненціальних функцій.

Метод перекодування кута підходить також і для обчислення гіперболічних функцій та експоненти. Враховуючи, що реалізація гіперболічного синуса та косинуса у порівнянні із класичним CORDIC методом замінює лише значення σ та початковий коефіцієнт деформації, то для модифікації методу слід дещо змінити формули їх розрахунку. Коефіцієнт деформації P тепер буде складатись з двох частин, $P_1$ та $P_2$, оскільки при обчисленнях для забезпечення збіжності ітераційного процесу деякі ітерації необхідно буде повторити.

$$P_1 = \prod_{m_{LUT}+1}^{\frac{m}{2}} \frac{1}{\sqrt{1-2^{-2i-2}}} \tag{2.39}$$

$P_2$ залежить від того, скільки ітерацій включено в таблицю попередньої вибірки LUT, відповідно до її розміру:

$$P_2 = \begin{cases} \frac{1}{\sqrt{1-2^{-80}}} if\ m_{LUT} > 13 \\ \frac{1}{\sqrt{(1-2^{-26})(1-2^{-80})}} if\ 14 > m_{LUT} > 4 \\ \frac{1}{\sqrt{(1-2^{-8})(1-2^{-26})(1-2^{-80})}} if\ m_{LUT} < 5 \end{cases} \tag{2.40}$$

Також корисним буде врахувати загальну розрядність обчислень. В таких випадках, якщо значення змінних виходять за межі розрядної сітки, то їх можна не враховувати при обчисленнях коефіцієнтів. Так, для m розрядних змінних, $P_2$ матиме наступний вигляд:

$$P_2 = \begin{cases} \frac{1}{\sqrt{1-2^{-8}}} if\ m \le 25 \\ \frac{1}{\sqrt{(1-2^{-8})(1-2^{-26})}} if\ 25 < m < 80 \\ \frac{1}{\sqrt{(1-2^{-8})(1-2^{-26})(1-2^{-80})}} if\ m_{LUT} \ge 80 \end{cases} \tag{2.41}$$

Відповідно, комбінуючи значення $m_{LUT}$ та розрядності m згідно з формулами будемо одержувати різні комбінації коефіцієнта $P_2$. Так, наприклад, для розрядностей m<40 та таблиці $m_{LUT}$>13 коефіцієнт $P_2$ доцільно взяти рівним

одиниці, оскільки старші два коефіцієнти будуть обчислені табличним методом згідно формулою (2.28), а молодший коефіцієнт формули (2.41) вийде за межі розрядної сітки. Сумарне значення $P_c$ буде становити:

$$P_c = P_1 \cdot P_2 \tag{2.42}$$

Повторення частини ітерацій дозволяє також розширити діапазон вхідних значень функції. Так, для обчислення експоненти $e^x$ значення порядку "x" залежить від кількості ітерацій методу та лежить в межах $x \in [-1{,}118173..1{,}118173]$ і обчислюється згідно з формулою:

$$x = \sum_{m=1}^{\infty} arc\tan\left(2^{-m}\right) \tag{2.43}$$

В результаті одержимо діапазон значень, в межах яких може лежати результат обчислень експоненти $e^x \in [0{,}32687...3{,}05926]$. Дана інформація знадобиться при виборі формату для змінних при апаратній та програмній реалізації функції.

Для обчислення кута корекції, який залежить в першу чергу від кількості розрядів перекодування, використовуємо вже відому формулу (2.33). Якщо крім методу залишкового множення використати також метод квадратичної апроксимації, то кількість розрядів для перекодування зменшиться, а молодші розряди $m_{cordic}$ будуть становити вже не половину (2.29), а старшу третину загальної кількості розрядів. Тому, розрахувавши кількість біт відведених на перекодування $m_{cordic}$, та одержавши кут $\varphi_{const}$, значення для таблиці попередньої вибірки, отримуємо згідно з формулою:

$$\begin{aligned} \exp_{m_{LUT}+1} &= P_c \cdot \exp\left(\varphi_1 + \varphi_{2\,const}\right) \\ \varphi_{2\,const} &= 2^{-(m_{LUT}+1)} - 2^{-(m_{CORDIC}+1)} \end{aligned} \tag{2.44}$$

Так само, як і для функцій синуса-косинуса, при повороті модуля вектора для функції експоненти за допомогою методу перекодування з кожною ітерацією буде накопичуватись сумарне відхилення Δ, яке буде становити:

$$\Delta = \sum_{i=m_{LUT}+1}^{m_{CORDIC}} b_i \cdot \left(2^{-i} - arctanh\left(2^{-i}\right)\right) \tag{2.45}$$

де $b_i$ — перекодований у базис {-1,+1} поточний (i -й) розряд вхідного

аргументу. На кожній ітерації обчислення експоненти обчислюється лише один вираз:

$$q_{i+1}=q_i+b_{i-1}\cdot q_i\cdot 2^{-i} \tag{2.46}$$

а для апаратної реалізації якого потрібно лише один конвеєр. Звернути увагу варто на значення змінної “i”, номер якої відрізняється для перекодованого розряду, $b_i$, та для змінної $q_i$. Такий підхід дозволяє скоротити кількість ітерацій та здійснювати першу ітерацію, починаючи з розряду $m_{LUT+2}$ згідно формули (2.29), хоча знак ітерації буде залежати від $m_{LUT+1}$ (2.28).

Метод залишкового множення також здійснюється за допомогою лише одного виразу і має вигляд:

$$q_{m_{CORDIC}+2}=q_{m_{CORDIC}+1}+\varphi_{зал}\cdot q_{m_{CORDIC}+1} \tag{2.47}$$

де значення $\varphi_{зал}$ в (2.47) отримується із формули наведеної вище (2.31).

Обчислення значень гіперболічного синуса та косинуса виконується за допомогою двох аналогічних до експоненти ітераційних рівнянь і відрізняються лише початковими значеннями. Діапазон обчислень функції відповідає діапазону обчислення експоненти, адже поворот вектора здійснюється на основі аналогічної таблиці гіперболічних арктангенсів (2.48). На першому етапі обчислень за допомогою табличного методу значення гіперболічного синуса та косинуса становить:

$$\begin{aligned} x_{m_{LUT}+1}&=P_c\cdot\cosh\left(\varphi_1+\varphi_{2\,const}\right)\\ y_{m_{LUT}+1}&=P_c\cdot\sinh\left(\varphi_1+\varphi_{2\,const}\right)\end{aligned} \tag{2.48}$$

Значення $P_c$ та $\varphi_{2const}$ аналогічні до попереднього методу. Таким самим чином відбувається перекодування вхідного значення (2.32), та процес накопичення сумарних $\Delta$ у першій половині розрядів. Самі ж ітераційні формули мають вигляд:

$$\begin{aligned} x_{i+1}&=x_i+b_{i-1}\cdot y_i\cdot 2^{-i}\\ y_{i+1}&=y_i+b_{i-1}\cdot x_i\cdot 2^{-i}\end{aligned} \tag{2.49}$$

Як видно, основна відмінність у порівнянні із обчисленням експоненти полягає лише у табличному методі, а також у попарно - паралельному способі реалізації функції. Після завершення ітерацій або паралельно з ними

обчислюється також кут корекції (2.31) та застосовується залишкове множення для молодшої половини розрядів:

$$x_{m_{CORDIC}+2} = x_{m_{CORDIC}+1} + \varphi_{зал} \cdot y_{m_{CORDIC}+1}$$
$$y_{m_{CORDIC}+2} = y_{m_{CORDIC}+1} + \varphi_{зал} \cdot x_{m_{CORDIC}+1} \quad (2.50)$$

Загалом, враховуючи подібність процесів обчислення гіперболічних синуса, косинуса та експоненти, а також ідентичні значення більшості змінних, можна здійснювати обчислення експоненти за допомогою ідентичного обладнання, що й для гіперболічних функції. Для цього знадобиться лише заміна табличного методу (2.48) на (2.44). Враховуючи, що для обчислення гіперболічних рівнянь необхідно використати вдвічі більше апаратних засобів через потребу реалізації двох паралельних конвеєрів, існує спосіб обчислювати два значення експоненти на тому самому обладнанні, що і гіперболічні функції. Для цього потрібно поміняти місцями дії додавання у формулах (2.49) та (2.50). У апаратній реалізації даний прийом вимагатиме лише зміни конфігурації монтажних з'єднань між елементами і не потребує використання додаткових логічних елементів. Зрозуміти причину такої спорідненості даних функцій можна з формули:

$$e^x = \sinh(x) + \cosh(x) \quad (2.51)$$

з якої бачимо, що на кожній ітерації методу обчислення гіперболічних функцій виконується рівність $e_i = x_i + y_i$. Таким чином, наближена величина експоненти завжди "розподілена" між двома змінними що зберігають значення гіперболічних функцій.

Загалом, підсумовуючи можливості методу перекодування для усього спектру обчислюваного ним функцій варто зазначити можливість прогнозування наступних ітерацій, яке відсутнє для класичного методу та його модифікацій. Завдяки цьому частину операцій, які необхідно буде здійснити, можна визначити відразу після одержання вхідного аргументу, що при апаратній реалізації дозволяє краще розмістити логічні елементи у схемі. Таким чином, після завершення однієї ітерації, логіка наступної ітерації вже готова до здійснення, а всі необхідні операції, викликані зміною напряму вектора повороту, вже застосовані. В

результаті цього напрям вектора повороту є наперед відомий, і не виникає потреби у здійсненні зміни “напряму” подальшої роботи алгоритму, що позитивно позначається на процесі обчислень.

Практична реалізація методу виявила і слабкі місця такого підходу, щоправда, не у всіх випадках вони виявляються вартими уваги. Так, множення чисел із знаком є більш складним і не сумісним з беззнаковим множенням, ніж наприклад, додавання та віднімання знакових та беззнакових чисел, які, по суті, представляють одну і ту ж операцію. І хоча у більшості випадків апаратна підтримка охоплює як знакові так і беззнакові операції множення, коректна їх реалізація може виявитись проблематичною, викликати перетворення типів змінних чи складнішу програмну реалізацію операції множення у мовах опису апаратних засобів. Зіткнувшись із ситуацією, коли підтримка множення знакових та беззнакових чисел була реалізована апаратно, але програмні методи у вигляді набору стандартів не дозволяли здійснювати такі операції без обов’язкового попереднього приведення змінних до одного формату, метод перекодування кута був модифікований та розвинений. Також варто відзначити що сам факт узгодження типів даних не є негативним, але в даному конкретному випадку такий підхід збільшує розрядність обчислень, і при реалізації алгоритмів виникла ситуація, коли необхідна кількість блоків помножувачів зросла у два рази через необхідність виконання вимог стандартів.

Зазвичай, при реалізації класичного методу CORDIC та його модифікацій, для роботи алгоритму діапазон вхідних та вихідних змінних звужують до першої чверті декартових координат. Однією з переваг такого підходу є можливість подальшої роботи лише із беззнаковими величинами. У методі ж перекодування кута можлива поява від’ємних чисел і у цьому діапазоні. Відбувається це у процесі повороту вектора у напрямку осі абсцис, коли сумарне значення модуля змінної $\Delta$ більше за $\varphi_3$, в результаті чого згідно з (2.31) $\varphi_{зал}$ набуває негативного значення.

Модифікований метод беззнакового перекодування кута базується на

уникненні виконання операцій, таких як (2.35) при негативному значенні $b_i$, що дозволяє отримувати тільки додатні значення Δ, а відповідно і $\varphi_{зал}$(2.45), яке використовується для операції множення у кусково-лінійній апроксимації. Щоб компенсувати значення від'ємних Δ утворених різницею арктангенсів:

$$\Delta = 2^{-i} - \arctan\left(2^{-i}\right) \tag{2.52}$$

або для гіперболічних функцій та експоненти:

$$\Delta = 2^{-i} - arctanh\left(2^{-i}\right) \tag{2.53}$$

їхнє значення враховується при побудові таблиць попередньої вибірки. Загальний вигляд формул для обчислення значень таблиці залишається незмінний:

$$x_{m_{LUT}+1} = P \cdot \cos\left(\varphi_1 + \varphi_{2const}\right) \quad y_{m_{LUT}+1} = P \cdot \sin\left(\varphi_1 + \varphi_{2const}\right) \tag{2.54}$$

а для гіперболічних функцій:

$$f_{m_{LUT}+1} = P_c \cdot \mathrm{func}\left(\varphi_1 + \varphi_{2const}\right) \tag{2.55}$$

де під змінною func розуміється sinh, cosh, чи $e^x$. Значення ж $\varphi_{const}$ змінюється у випадку застосування методу додатної дельти:

$$\phi_{const} = \sum_{m_{LUT}+2}^{\frac{m}{2}+1} \arctan\left(h\right)\left(2^{-i}\right) \tag{2.56}$$

значення арктангенсів для гіперболічних функцій також повинно бути гіперболічним. Використання вищезгаданого значення $\varphi_{const}$ дозволяє уникнути операції зменшення дельти, що в свою чергу змінює принцип її розрахунку для повороту вектора в сторону збільшення до осі ординат:

$$\Delta = \sum_{i=m_{LUT}}^{m/2} 2^{-i-1} - 2 \cdot \arctan(\mathrm{h}) 2^{-i-2} \tag{2.57}$$

а для формули знаку дельти значення $b_i$ зміниться на $a_i$, що при нульових значеннях відповідного розряду дозволяє уникнути зменшення дельти. Всі інші ітерації можуть здійснюватись без модифікації. Єдине, що необхідно також врахувати, це збільшення вдвічі сумарного значення дельти, і, відповідно, з'являється необхідність у збереженні істинності виразу $\Delta \leq \varphi_3$, в іншому випадку значення $\varphi_{зал}$ потрібно збільшити на відповідну кількість розрядів, виходячи з формули (2.45).

Загалом, метод не змінює складності та точності обчислень, хоча при послідовній програмній реалізації можливе незначне зменшення латентності через усунення операції додавання негативної дельти. У паралельній розгорнутій апаратній реалізації даний ефект буде спостерігатись у відсутності суматора для від'ємних дельта, хоча, здебільшого, апаратно реалізується тільки один суматор, а операція віднімання представляє собою лише інверсію розрядів аргументу. До недоліків беззнакового перекодування кута можна віднести хіба що менш наочне представлення операцій методу, а також вищезгадану несиметричність. Використання ж знакового перекодування можна рекомендувати у випадках, коли використання множення знакових чисел з якихось причин виявиться більш перспективним, наприклад, у плані швидкодії, чи якщо команди або блоки беззнакового множення чисел виявились відсутніми.

## 2.5 Способи удосконалення алгоритму CORDIC.

Прискорення роботи алгоритму можна досягнути шляхом використання методу залишкового множення. У результаті повороту модуля вектора з кожною наступною ітерацією коефіцієнт його деформації зменшується згідно з (2.26). З формули випливає, що після здійснення половини ітерацій коефіцієнт деформації виходить за межі розрядної сітки, через що кожний наступний крок алгоритму може бути представлений як операція зсуву та додавання, що представляє собою еквівалент операції множення. Виходячи з цих міркувань, відпадає потреба у здійсненні аналізу молодшої половини розрядів вхідного операнду, а точність методу підвищується вдвічі завдяки відсутності похибки округлення, яка вноситься кожною ітерацією.

У випадку якщо вибрана платформа не здатна забезпечити апаратне виконання дії множення, її можна реалізувати за допомогою зсувів та додавань, причому їх ефективність можна збільшити завдяки розпаралелюванню процесу. Так, для множення m-розрядних чисел необхідно здійснити $\log_2 m$ ітерацій

додавань. Як показала практична перевірка методу, оптимальним значенням для здійснення залишкового множення з точки зору точності обчислень становить $\frac{m}{2}$ розрядів операнда. У випадку збільшення кількості ітерацій класичного CORDICa похибка обчислень починає лінійно накопичуватись і при значеннях кількості ітерацій близьких до максимальних близька чи навіть рівна похибці класичного методу. Виконання залишкового множення раніше, ніж за $\frac{m}{2}$ ітерацій неможливе через коефіцієнт деформації, який ще присутній в операндах. Похибка, яка виникає таким чином, значно перевищує похибку класичного методу, в тому числі і у випадку, якщо кількість класичних ітерацій зменшилась лише на один розряд. Таким чином, вхідний кут φ розбивається на два діапазони:

$$\begin{gathered}\phi_{INPUT}=\phi_{CORDIC}+\phi_{MULTIPLY}\\ \varphi_{CORDIC}=\sum_{i=1}^{\frac{m}{2}} a_i 2^{-i}\\ \varphi_{MULTIPLY}=\sum_{i=\frac{m}{2}+1}^{m} a_i 2^{-i}\end{gathered} \tag{2.58}$$

для старшої половини розрядів $\varphi_{CORDIC}$ виконуємо класичні формули згідно з (1.48), тоді як молодші розряди обчислюємо наступним чином:

$$\begin{gathered}x_m=x_{\frac{m}{2}}-\varphi_{зал}\cdot y_{\frac{m}{2}}\ ;\ x_m=\cos(\varphi_{INPUT})\\ y_m=y_{\frac{m}{2}}+\varphi_{зал}\cdot x_{\frac{m}{2}}\ ;\ y_m=\sin(\varphi_{INPUT})\end{gathered} \tag{2.59}$$

де $\varphi_{зал}$ — кут в радіанах, який залишився після обчислення кута $z_{m/2}$ класичним методом. Комбінацію класичного методу та залишкового множення можна представити у вигляді блок-схеми на рис 2.1. Якщо ж операція множення на платформі відсутня і її необхідно синтезувати, як правило, методом додавань та зсувів. Приклад такої реалізацій для 16 розрядів поданий на рис. 2.2.

При повороті вектора класичним методом кількість кутів, а відповідно і точність обчислень, залежить від кількості здійснених ітерацій. Як було сказано в главі 1.5.7, при невеликих кількостях ітерацій всі вхідні та вихідні значення можна

задавати у вигляді таблиці, розміщеної в пам’яті. Враховуючи, що в переважній більшості пристроїв доступним є деякий об’єм пам’яті, хоч і незначний, його можна використати для прискорення методу CORDIC. Тоді ефект буде відчутний і у випадку, якщо обсяг виділеної пам’яті становить сотні чи навіть десятки байт (оперативної чи флеш-пам’яті у випадку МК). Також пам’яттю може виступати набір монтажних з’єднань із сигналом нуля та одиниці при апаратній реалізації функцій в архітектурах, де відсутні виділені блоки пам’яті. При застосуванні такої переглядової таблиці можна відзначити два підходи.

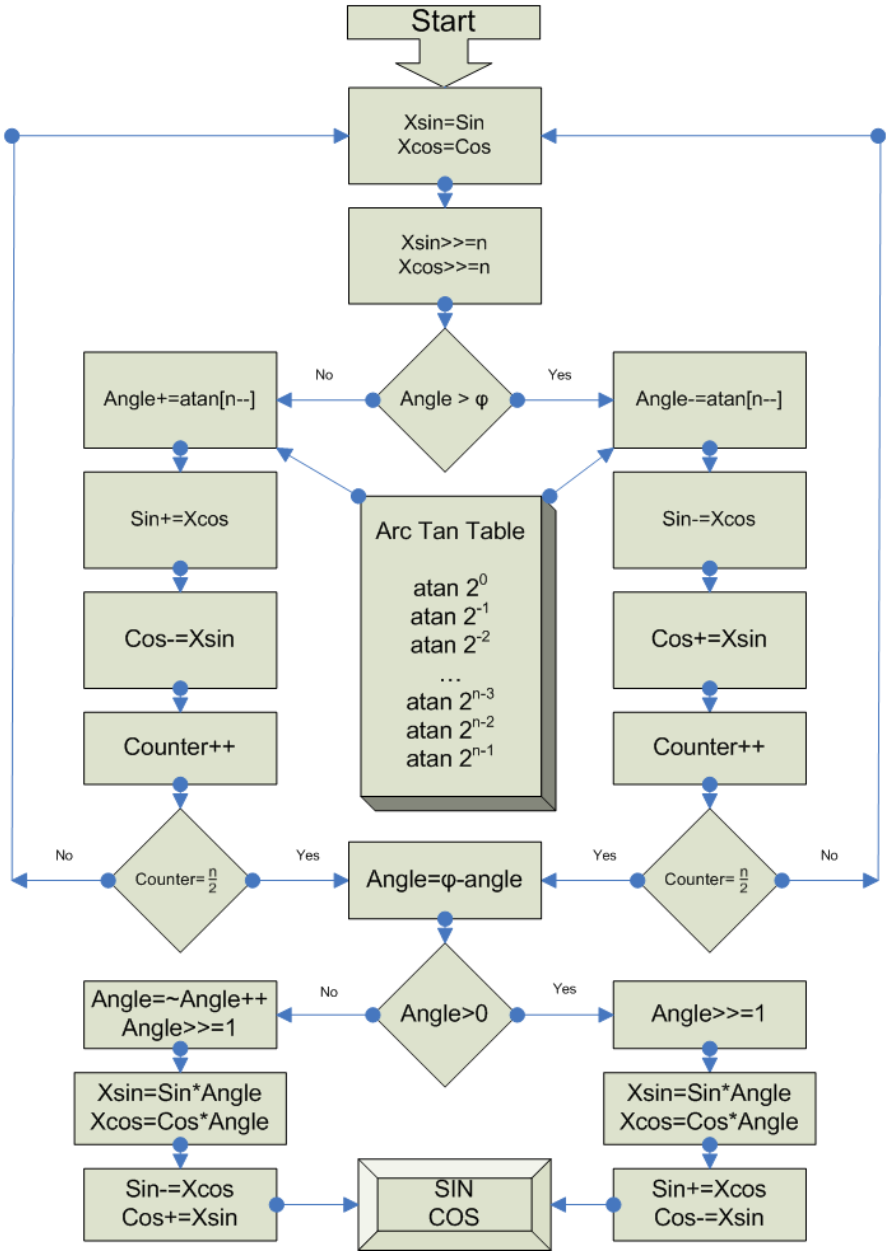



Рис. 2.1 Схема реалізації класичного CORDIC з методом залишкового множення

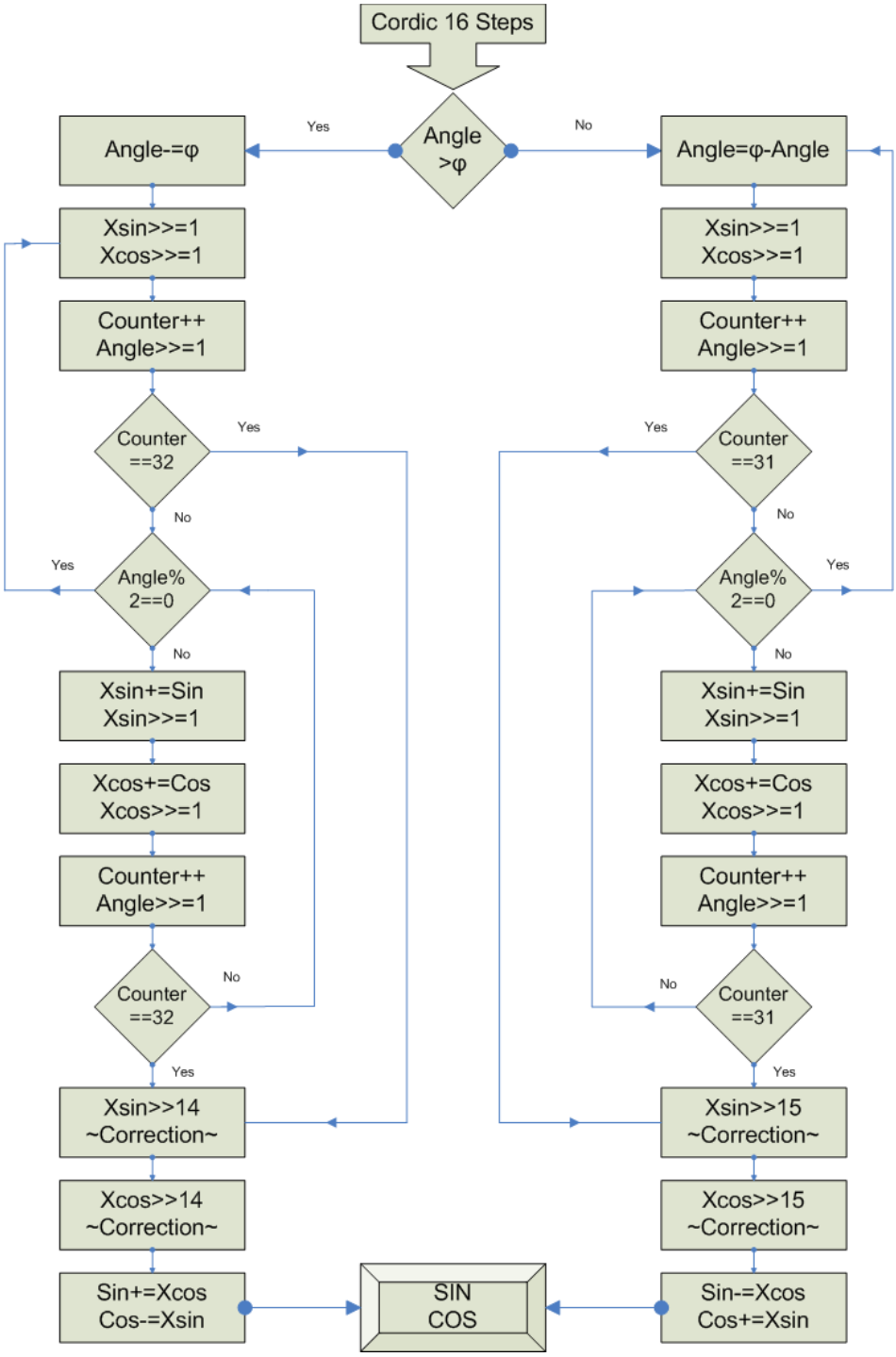



Рис. 2.2 Схема здійснення програмного множення на заключних ітераціях для обчислень методом кусково-лінійної апроксимації

При повороті вектора класичним методом кількість кутів, а відповідно і точність обчислень, залежить від кількості здійснених ітерацій. Як було сказано в главі 1.5.7, при невеликих кількостях ітерацій всі вхідні та вихідні значення можна задавати у вигляді таблиці, розміщеної в пам’яті. Враховуючи, що в переважній

більшості пристроїв доступним є деякий об’єм пам’яті, хоч і незначний, його можна використати для прискорення методу CORDIC. Тоді ефект буде відчутний і у випадку, якщо обсяг виділеної пам’яті становить сотні чи навіть десятки байт (оперативної чи флеш-пам’яті у випадку МК). Також пам’яттю може виступати набір монтажних з’єднань із сигналом нуля та одиниці при апаратній реалізації функцій в архітектурах, де відсутні виділені блоки пам’яті. При застосуванні такої переглядової таблиці можна відзначити два підходи.

Перший спосіб передбачає вибір кута після виконання поворотів вектора згідно з класичним методом без розрахунку параметрів $x_i$ та $y_i$, після чого відбувається вибір даних параметрів з пам’яті. Такий спосіб зменшує складність апаратної реалізації та кількість логічних елементів, оскільки частково усуває два конвеєра для обчислення значень $x_i$ та $y_i$, і замінює їх блоками пам’яті. В програмній реалізації такий підхід означатиме меншу кількість команд для кожної ітерації алгоритму, а кількість ітерацій, здійснених табличним методом, може бути довільною. Недоліком такого підходу є неможливість усунення початкових ітерацій алгоритму, тоді як по суті відбувається лише заміна арифметичних операцій — вибіркою з пам’яті. Для восьмикрокової таблиці схема алгоритму набуде наступного вигляду рис. 2.3.

Другий спосіб є більш ефективний і передбачає вибірку операндів з пам’яті за їх адресою, яка, в свою чергу, визначається комбінацією вибраної заздалегідь кількості старших розрядів вхідного кута. Складність реалізації цього підходу виникає через те, що поворот вектора здійснюється не на круглу двійкову величину, а на тангенс степені двійки. Обійти це обмеження можна у випадку, коли величина кута повороту φ буде приблизно дорівнювати степені двійки, а похибка вийде за межі розрядної сітки. Такий ефект спостерігається після виконання третини ітерацій, тобто розмір таблиці попередньої вибірки повинен включити як корекцію кута, так і, принаймні, $\frac{m}{3}$ розрядів таблиці. Зрозуміло, що такий підхід неприйнятний при обчисленні функцій з високою розрядністю та

невеликим обсягом пам’яті. Щоб уникнути цього недоліку, розроблено ряд методів, таких як: односторонній поворот, інверсний поворот, метод перекодування кута, та перекодування з додатною дельтою.

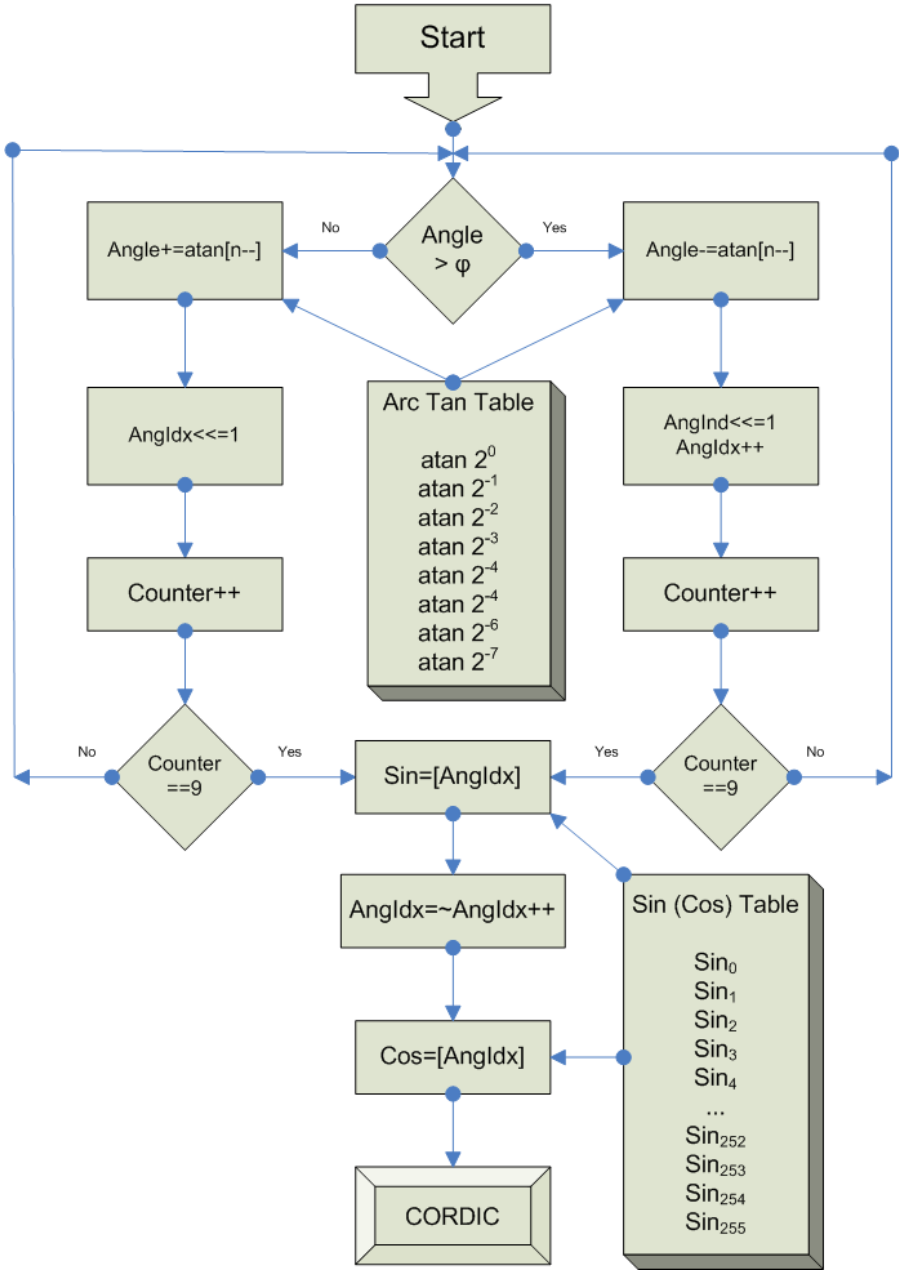



Рис. 2.3 Схема здійснення ітерацій алгоритму для наступної вибірки з таблиці

## 2.6 Односторонній поворот вектора

Метод одностороннього повороту базується на використанні модифікованого метода Рунге-Кутта третього порядку. Оригінальний метод Рунге-

Кутта має вигляд:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1-\frac{\varepsilon^2}{2} & \frac{\varepsilon^3}{6}-\varepsilon \\ \varepsilon-\frac{\varepsilon^3}{6} & 1-\frac{\varepsilon^2}{2} \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \tag{2.60}$$

де для спрощення операції ділення на шість використовується бінарний зсув операнда на три розряди в сторону молодших бітів. Таким чином, матриця псевдоповороту для модифікованого методу Рунге-Кутта набуває вигляду:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1-\frac{\varepsilon^2}{2} & \frac{\varepsilon^3}{8}-\varepsilon \\ \varepsilon-\frac{\varepsilon^3}{8} & 1-\frac{\varepsilon^2}{2} \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \tag{2.61}$$

Отримане таким чином спрощення апаратної реалізації має незначний вплив на точність результатів, і, як показала практика, збільшення відхилення все одно лежить в тих самих межах розрядної сітки, для якої необхідно забезпечити точний результат після округлення. Використовуючи цей метод ротація вектора здійснюється тільки в одну сторону — в сторону збільшення кута, аналізуючи в процесі обчислень лише одиничні розряди вхідного кута. Це означає, що таблиця попередньої вибірки повинна містити значення синусів та косинусів без урахування коефіцієнта деформації, а їхнє значення буде коректуватись в процесі повороту вектора шляхом здійснення додаткових операцій згідно з матрицею перетворень (2.61). У кінцевому варіанті формули для обчислень методом одностороннього повороту мають вигляд:

$$\begin{gathered} x_{i+1} = x_i \cdot \left(1-\frac{\varepsilon^2}{2}\right) - y_i \cdot \left(\frac{\varepsilon-\varepsilon^3}{8}\right) \\ y_{i+1} = y_i \cdot \left(1-\frac{\varepsilon^2}{2}\right) - x_i \cdot \left(\frac{\varepsilon-\varepsilon^3}{8}\right) \\ \varepsilon = 2^{-i} \end{gathered} \tag{2.62}$$

де i - номер ітерації.

Як видно, необхідно здійснити додаткову дію додавання-віднімання для значення $x_i$ та $y_i$, причому застосовувати її для всіх кутів не обов’язково, а лише для:

$$\frac{\varepsilon^3}{8} > 2^{-n} \tag{2.63}$$

де n-розрядність операндів, $\varepsilon = 2^{-i}$, i - номер ітерації,

тобто коли $2^{-3(i-1)} < 2^{-n}$, розрядність операції виходить за межі обчислень, і дану дію можна упустити. В результаті отримаємо спрощену формулу для обчислень:

$$\begin{aligned} x_{i+1} &= x_i \cdot \left(1 - \frac{\varepsilon^2}{2}\right) - y_i \cdot \varepsilon \\ y_{i+1} &= y_i \cdot \left(1 - \frac{\varepsilon^2}{2}\right) - x_i \cdot \varepsilon \end{aligned} \tag{2.64}$$

Незважаючи на вищу складність формул у порівнянні з класичним методом, основний виграш в швидкодії забезпечується за рахунок меншої кількості ітерацій, яка досягається приблизно у 85% випадків (2.65). Розглянемо даний факт на прикладі.

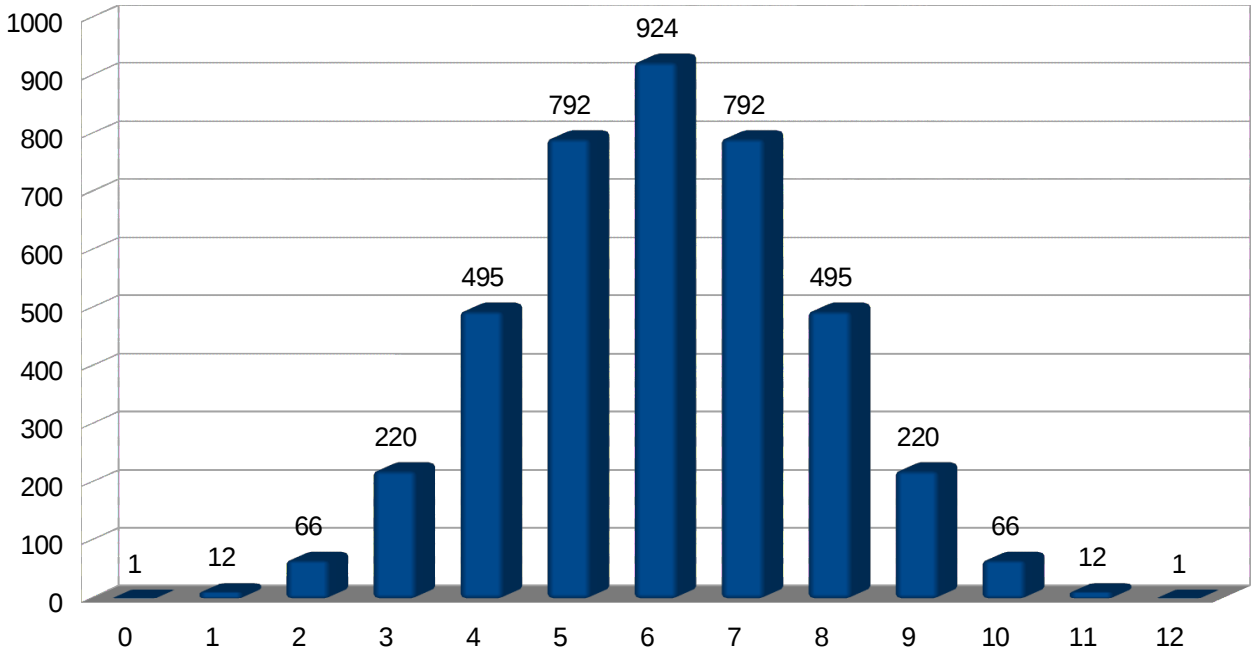



Рис. 2.4 Розподіл однотипних розрядів можливих вхідних комбінацій аргументу для методу одностороннього та інверсного повороту

При реалізації формул одностороннього повороту з табличним методом та залишковим множенням для 32 розрядної платформи можна відвести під зберігання значень таблиці лише 4 розряди, для чого знадобиться 128 байт.

Враховуючи, що молодші шістнадцять розрядів обчислюються методом кусково-лінійної апроксимації, через що обчислення методом одностороннього чи інверсного повороту відбувається для 32 - 4 - 16 = 12-ти розрядів. При цьому кількість випадків, при яких m - біт повороту дорівнює нулю (одиниці) подана на рис. 2.4.

Якщо весь діапазон значень однотипних розрядів поділити на три частини то 3,5 тис. кутів з 4 тис. припадають на центральну третину, тобто в середньому сім з восьми кутів. В результаті ймовірність попадання кута в проміжок 4 - 8 кроків зі всього діапазону 0 - 12 кроків буде становити:

$$\frac{495\cdot 2+792\cdot 2+924}{2^{12}}\cdot 100\,\%=\frac{3498}{4096}\cdot 100\,\%=85{,}4\,\% \qquad (2.65)$$

Для більш наочного розподілу ймовірностей, представимо дані у таблиці:

*Таблиця 2.4*

| Кількість однотипних розрядів | Кількість можливих комбінацій | Середня ймовірність одержання |
|---|---|---|
| 0 | 1 | 0,02% |
| 1 | 12 | 0,29% |
| 2 | 66 | 1,61% |
| 3 | 220 | 5,37% |
| 4 | 495 | 12,08% |
| 5 | 792 | 19,34% |
| 6 | 924 | 22,56% |
| 7 | 792 | 19,34% |
| 8 | 495 | 12,08% |
| 9 | 220 | 5,37% |
| 10 | 66 | 1,61% |
| 11 | 12 | 0,29% |
| 12 | 1 | 0,02% |

Формула для розрахунку ймовірності має наступний вигляд:

$$P_{prob}=\frac{n!}{m!\cdot(n-m)!\cdot 2^n}\cdot 100\,\% \qquad (2.66)$$

де n — загальна кількість розрядів, m — кількість однотипних розрядів

## 2.7 Метод інверсного повороту.

Доволі часто основними параметрами алгоритмів при їх наступній реалізації стають часові обмеження, задані в технічному завданні на проект. Тому важливішим, а іноді і критичним є наперед заданий проміжок часу, протягом якого гарантовано можна буде отримати кінцевий результат. І хоча метод одностороннього повороту виграє у класичного метода в швидкодії при аналогічних апаратних затратах, існує деякий діапазон вхідних значень, при яких швидкодія одностороннього повороту нижча за класичний метод. Кількість таких значень із нижчою швидкодією може становити менше 1%, але і перевага може бути надана алгоритму, у якого показник максимальної латентності нижчий, незважаючи на його вищу загальну сумарну затримку.

Для покращення показника латентності алгоритму запропонований метод інверсного повороту, який базується на методі одностороннього повороту і функціонує сумісно з ним. Основна відмінність полягає у напрямі повороту вектора при наближенні до заданого кута. На сам перед необхідно замінити таблицю попередньої вибірки метода одностороннього повороту, де старшим m розрядам кута φ відповідає синус та косинус кута:

$$\begin{gathered}\varphi_{input}=\varphi_{lut}+\varphi_{rot}\\ \varphi_{lut}=\sum_{i=1}^{m(lut)} a_i 2^{-i}\\ \varphi_{rot}=\sum_{i=m(lut+1)}^{m(lut+rot)} a_i 2^{-i}\\ \sin_{table}=\sin\left(\varphi_{lut}\right)\\ \cos_{table}=\cos\left(\varphi_{lut}\right)\end{gathered} \tag{2.67}$$

як видно, в таблицю записується кут, в якого всі розряди $\varphi_{rot}$ дорівнюють нулю, а кут $\varphi_{input} = \varphi_{lut}$. Для інверсного повороту, навпаки, всі значення кута $\varphi_{rot}$ повинні братись рівними одиниці, тобто:

$$\begin{gathered}\sin_{table}=\sin\left(\varphi_{lut}+a_i 2^{-m(lut)}-a_i 2^{-m(lut+rot)}\right)\\ \cos_{table}=\cos\left(\varphi_{lut}+a_i 2^{-m(lut)}-a_i 2^{-m(lut+rot)}\right)\end{gathered} \tag{2.68}$$

Також при виконанні алгоритму інверсного повороту, модифікації підлягають значення напряму повороту, що визначаються знаком операції і які

потрібно замінити на протилежні. Тому при повороті величина косинуса буде збільшуватись, а синуса зменшуватись:

$$\begin{gathered} x_{i+1} = x_i \cdot \left(1 - \frac{\varepsilon^2}{2}\right) + y_i \cdot \left(\frac{\varepsilon - \varepsilon^3}{8}\right) \\ y_{i+1} = y_i \cdot \left(1 - \frac{\varepsilon^2}{2}\right) - x_i \cdot \left(\frac{\varepsilon - \varepsilon^3}{8}\right) \\ \varepsilon = 2^{-i} \end{gathered} \tag{2.69}$$

де і - номер ітерації.

Операції в цьому методі здійснюються лише у випадку, якщо значення розрядів вхідного кута дорівнюють нулю, а у випадку одиниці — пропускаються. Звичайно, що використовувати потрібно тільки один із методів: односторонній поворот або інверсний поворот. Вибір методу, яким буде виконуватись поворот, пропонується здійснювати паралельно, або зразу після приведення кута в перший квадрант у випадку послідовної програмної реалізації. При апаратній реалізації метод легко модифікується для можливості використання в обох варіантах. Так, напрям повороту (заміна знаку при обчисленнях) реалізується за допомогою інверторів на входах та виходах суматорів. Цей прийом добре відомий і полягає в тому, що суматори здатні однаково ефективно здійснювати як операцію додавання, так і знаходження різниці. Звичайно, поява інверторів та мультиплексорів для ввімкнення та вимкнення інверторів у схемі здатна дещо знизити швидкодію, проте загальний результат все ж покращується через сумарне зменшення числа ітерацій, а при необхідності досягнення максимальної швидкодії можлива реалізація двох паралельних конвеєрів для обох методів. Подібна ситуація складається і з таблицею попередньої вибірки. Можна помітити, що значення кутів $\varphi_{LUT}$, для яких шукається значення функції, практично не відрізняється від:

$$\phi_{lut} \approx \phi_{lut-1} + a_i 2^{-m(lut)} - a_i 2^{-m(lut+rot)} \tag{2.70}$$

а різниця між сусідніми значеннями буде становити лише $2^{-i(lut+rot)}$. Таким чином, за рахунок незначної втрати точності можливо використовувати одну таблицю для обох методів. Значення кута для методу інверсного повороту буде відповідати наступному значенню в таблиці попередньої вибірки відносно того, яке б використовувалось для одностороннього повороту. А досягнення

максимальної точності методу можливе шляхом використання двох таблиць, що є співрозмірним по затратах до здійснення однієї ітерації методом одностороннього повороту. Наприклад, при обсязі спільної таблиці в 416 байт, необхідно здійснити 11 ітерацій одностороннього або інверсного повороту перед виконанням залишкового множення для 32 розрядних операндів. Хоча за допомогою цього ж обсягу можна розмістити дві точніші таблиці для кожного методу зокрема, щоправда, в такому випадку доведеться здійснити не 11 а 12 ітерацій методу. Блок схема алгоритму вибору значень з таблиці в п’ять розрядів при 32-х розрядній реалізації подана на рис. 2.5:

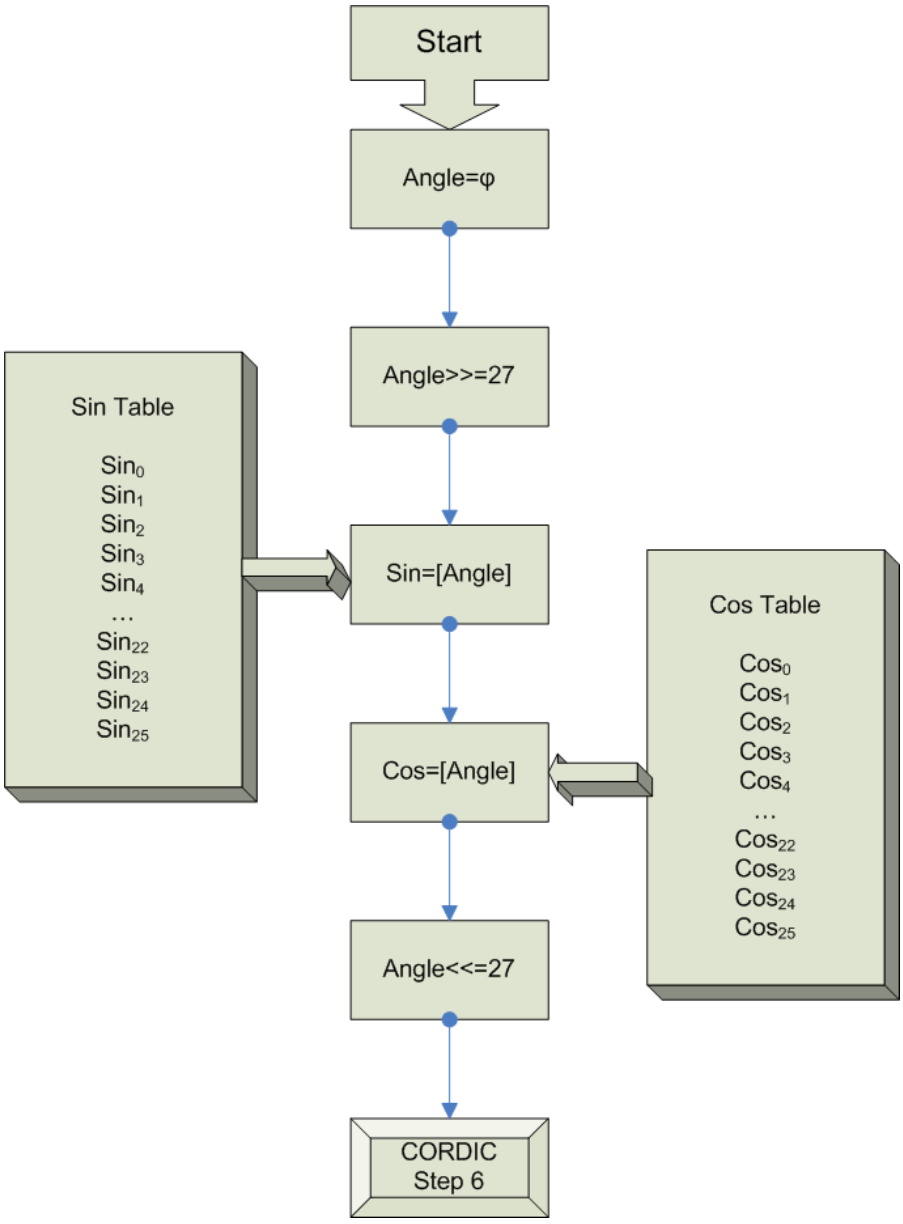



Рис. 2.5 Схема таблиці попередньої вибірки для гібридного методу

Блок-схема базової частини алгоритму, подана на рис. 2.6. Пунктиром позначена частина операцій, виконання яких можна пропустити при виході здійснюваних обчислень за межі розрядної сітки.

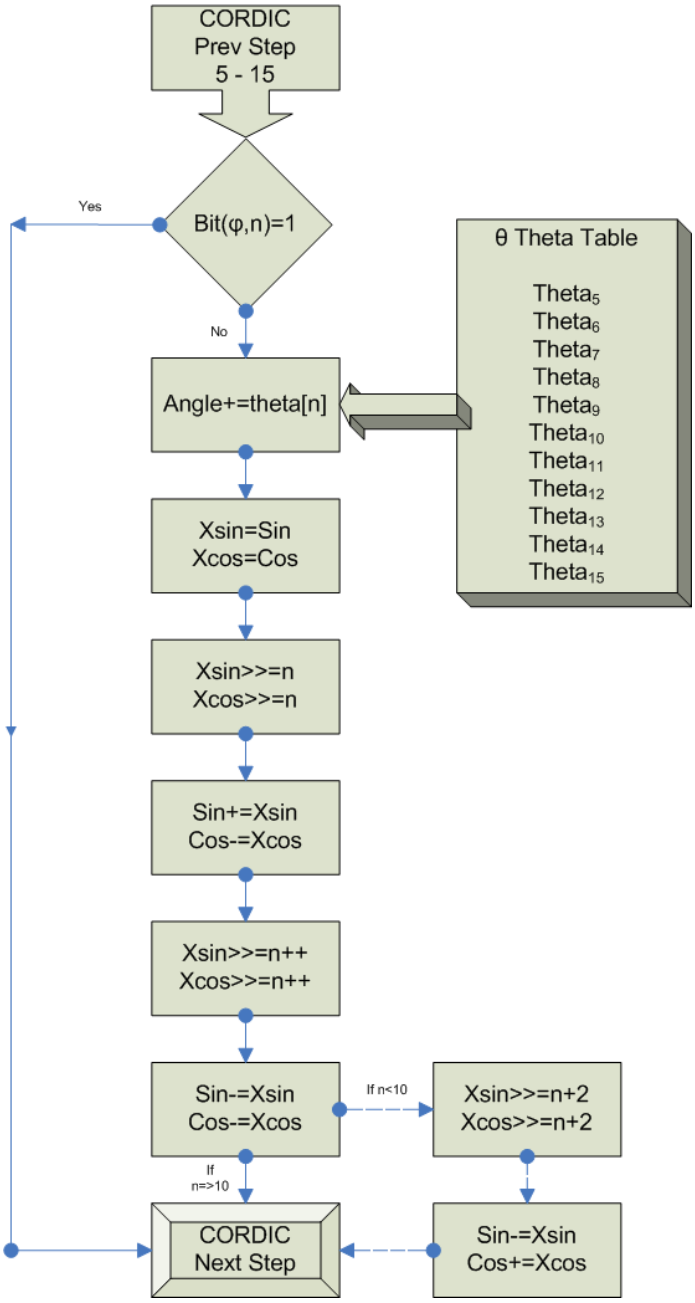



Рис. 2.6 Схема повороту вектора для методу інверсного повороту

У підсумку одержуємо метод, який здатний обчислювати функції з мінімальним числом ітерацій. Так, за допомогою 208 байт пам’яті та двох 16-розрядних помножувачів значення функцій синуса та косинуса одержується в найгіршому випадку за 9 тактів, з яких один такт іде на вибірку з 4- розрядної

таблиці, а два — на залишкове множення. Решта 12 розрядів алгоритму будуть виконані протягом 6-ти тактів, залежно від того, який метод буде ефективніший з огляду на варіант комбінації вхідних біт у ньому.

## 2.8 Удосконалений метод кусково-нелінійної апроксимації

Збільшити швидкість та точність обчислення функцій можна за допомогою використання додаткових дій множення. Зразу слід зауважити, що дане удосконалення відрізняється від наближення за допомогою поліномів, та в ньому не використовуються наперед обчислені коефіцієнти. Операції методом квадратичної інтерполяції здійснюються на основі залишкового кута $\varphi_{зал}$, який утворюється згідно з формулою (2.34) незалежно від методу, використаного для одержання цього значення. Це може бути як класичний, так і табличний метод з перекодуванням чи односторонній поворот. В методі квадратичної інтерполяції одночасно із дією додавання, яка утворилась при залишковому множенні, відбувається повторне множення результату на кут $\varphi_{зал}$, та зсув результату на один розряд згідно формулами:

$$\begin{aligned} x_m &= x_{\frac{m}{3}} \cdot \left(1 - \frac{\phi_{зал}^2}{2}\right) - y_{\frac{m}{3}} \cdot \phi_{зал} \\ y_m &= y_{\frac{m}{3}} \cdot \left(1 - \frac{\phi_{зал}^2}{2}\right) + x_{\frac{m}{3}} \cdot \phi_{зал} \end{aligned} \tag{2.71}$$

розрядність в цьому випадку зростає лише на третину у порівнянні з залишковим множенням, яке дає можливість подвоїти кількість правильних розрядів. Також можна підкреслити, що розрядність обчислень вдасться збільшувати і далі, додавши до формули ще один множник у вигляді залишкового кута, тепер уже третього порядку:

$$\begin{aligned} x_m &= x_{\frac{m}{4}} \cdot \left(1 - \frac{\phi_{зал}^2}{2}\right) - y_{\frac{m}{4}} \cdot \left(\phi_{зал} + \frac{\phi_{зал}^3}{6}\right) \\ y_m &= y_{\frac{m}{4}} \cdot \left(1 - \frac{\phi_{зал}^2}{2}\right) + x_{\frac{m}{4}} \cdot \left(\phi_{зал} + \frac{\phi_{зал}^3}{6}\right) \end{aligned} \tag{2.72}$$

Реалізувати дану формулу як програмно так і апаратно є досить

ресурсозатратно, оскільки ділення на шість не зводиться до операції зсуву. Якщо ж все-таки округлити дане значення, то збільшення розрядності обчислень відбудеться лише на одну четверту від вхідної розрядності. Це свідчить, що ефективність використання операцій множення з кожним новим аргументом рівняння — спадає. Враховуючи значну складність реалізації формул при збільшенні порядку чисел, оптимальним рішенням буде зупинитись на схемі квадратичної інтерполяції за формулою (2.71), та використовувати її в комбінації із залишковим множенням, коли виграш точності у третину розрядів та швидкодії буде обґрунтованим. RTL схема реалізації алгоритму подана на рис. 2.7.

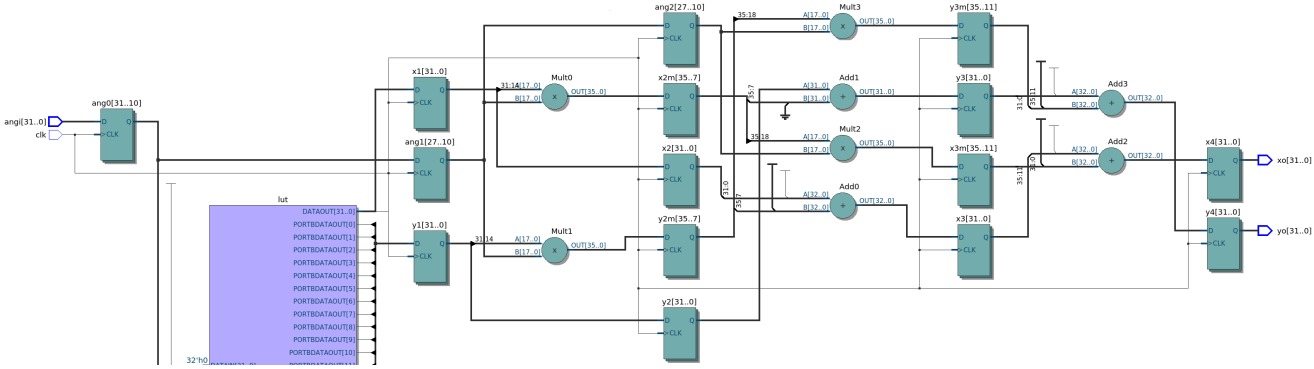



Рис. 2.7 32-бітна схема реалізації методу квадратичної апроксимації з одинадцятирозрядною таблицею

## 2.9 Паралельна поліноміально - квадратична інтерполяція

Для обчислення більшості елементарних функцій у заданих межах можливе використання поліноміальних наближень (розділ 1.5.3). Точність апроксимації в цьому випадку буде залежати від степеня полінома, та кількості і якості вибраних коефіцієнтів. Зрозуміло що формули типу (1.1) можна обчислювати за схемою Горнера (1.5), що однак не дає можливості здійснювати операції множення для вищих порядків, до моменту обчислення всіх попередніх степенів та коефіцієнтів. Спроба розпаралелювання обчислення функцій поліномами була здійснена у роботах [70], де поліноми розбивались на добутки виду $k(x-x^2)+1$, що також зменшувало кількість необхідних коефіцієнтів. Але в цьому випадку кількість послідовних операцій множення все одно не оптимальна, і після пошуку спільного

значення $x\text{-}x^2$, необхідно здійснювати множення на заданий коефіцієнт, тільки після цього стає можливим обчислення добутку усіх підфункцій. Тут також слід звернути увагу на те, що коефіцієнти підфункцій $k_n$ не є константами залежно від вибраної функції, а визнаються відносно до діапазону, у якому лежить вхідне значення функції.

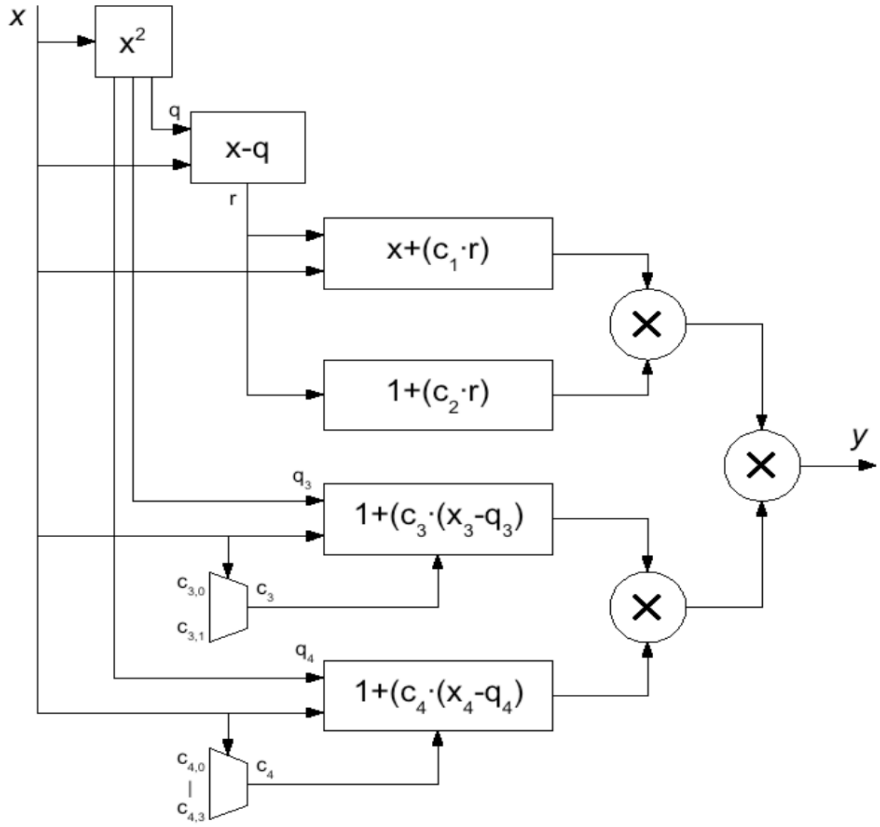



Рис. 2.8 Схема паралельного множення поліномів запропонована в [70]

При апаратній реалізації алгоритмів, в тому числі на платформі ПЛІС, доступна можливість задіювання ресурсів значної кількості помножувачів (DSP блоків), для досягнення мінімальних затримок шляхом розпаралелювання здійснюваних операцій. Для досягнення кращої ефективності методу пропонується обчислювати добутки поліномів за допомогою підфункцій виду $x^2+k_1x+k_2$, що дасть можливість шукати добуток kx одночасно із операцією піднесенням до степеня, а операція додавання після операції множення є вбудована у DSP блоки сучасних ПЛІС (множення з накопиченням). Незважаючи на появу коефіцієнта $k_2$ загальна кількість коефіцієнтів є меншою, оскільки всі вони є сталими для всього діапазону обчислень.

Пропонований метод паралельної поліноміально-квадратичної інтерполяції (ППКІ) може легко бути як звужений, так і розширений залежно від необхідної точності обчислень, шляхом зміни кількості добутків підфункцій, кількість яких прямо пропорційна порядку полінома. Оптимальна ж кількість помножувачів для знаходження добутку всіх підфункцій є степенем двійки, адже в цьому випадку можна здійснювати множення за допомогою деревоподібної структури. Серед функцій, які були реалізовані методом ППКІ, є синус, косинус, тангенс, арктангенс, гіперболічні функції, а також експонента, квадратний корінь, ділення, та інверсний корінь. Загальна схема ППКІ тринадцятого порядку подана на рис.2.9.

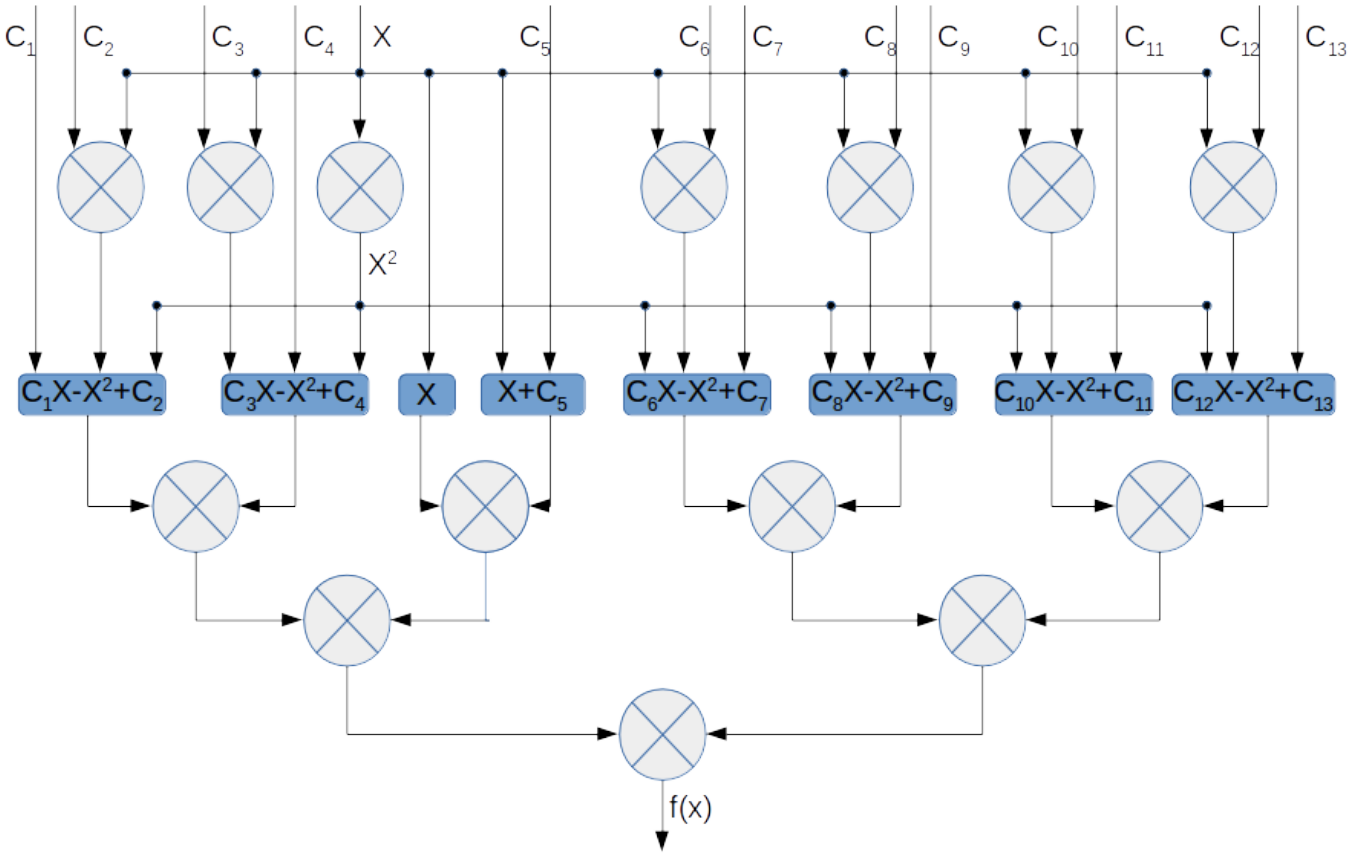



Рис. 2.9 Пропонована схема паралельної поліноміально - квадратичної інтерполяції із зменшеною кількістю етапів обчислень.

Для побудови оптимальної схеми методом ППКІ бажано наперед визначити необхідну точність обчислень, яку потрібно досягнути. Далі базуючись на даних

таблиць 2.5 — 2.8 вибирається поліном відповідної степені, виходячи з кількості точних бінарних розрядів, які він здатний забезпечити. Якщо ж задача стоїть у забезпеченні максимальної точності на деякій платформі, то основним показником для оптимізації швидкодії стає кількість помножувачів, які можуть працювати паралельно. В цьому випадку кількість помножувачів буде залежати від кількості коефіцієнтів і, як правило, становить:

$$multipliers = 2 \cdot (p-1) \qquad (2.73)$$

де p — степінь полінома. Точнішу кількість можна отримати із полів таблиці X, kX та $X^2$. Оптимальним значенням схеми, як вже було сказано, є поліном, що дорівнює деякій степені двійки. Далі наведені дані точності та ресурсомісткості обчислення функцій синуса, косинуса, тангенса для першої чверті, адже точність обчислень суттєво залежить від величини вибраного діапазону.

*Таблиця 2.5*

Кількість поліномів і точність обчислень для функції синуса на проміжку 0..π/4

| Степінь полінома | X | kX | $X^2$ | ±ΔX | Число точних розрядів |
|---|---|---|---|---|---|
| 3 | 1 | 2 | 0 | $4{,}7 \times 10^{-5}$ | 13,3 |
| 4 | 1 | 3 | 0 | $4{,}4 \times 10^{-6}$ | 16,7 |
| 5 | 1 | 2 | 2 | $6{,}0 \times 10^{-8}$ | 23,0 |
| 6 | 1 | 3 | 2 | $4{,}1 \times 10^{-9}$ | 26,8 |
| 7 | 1 | 4 | 2 | $4{,}2 \times 10^{-11}$ | 33,5 |
| 8 | 1 | 5 | 2 | $2{,}2 \times 10^{-12}$ | 37,7 |

*Таблиця 2.6*

Кількість поліномів та точність обчислень функції косинуса на проміжку 0..π/4

| Степінь полінома | X | kX | $X^2$ | ±ΔX | Число точних розрядів |
|---|---|---|---|---|---|
| 3 | 0 | 3 | 0 | $1{,}1 \times 10^{-4}$ | 12,1 |
| 4 | 0 | 4 | 0 | $1{,}8 \times 10^{-6}$ | 18,0 |
| 5 | 0 | 5 | 0 | $1{,}4 \times 10^{-7}$ | 21,7 |
| 6 | 0 | 4 | 2 | $1{,}7 \times 10^{-9}$ | 28,1 |
| 7 | 0 | 5 | 2 | $1{,}0 \times 10^{-10}$ | 32,2 |
| 8 | 0 | 6 | 2 | $9{,}1 \times 10^{-13}$ | 39,0 |

*Таблиця 2.7*

Кількість поліномів і точність обчислень для функції тангенса у проміжку 0..π/4

| Степінь полінома | X | kX | $X^2$ | ±ΔX | Число точних розрядів |
|---|---|---|---|---|---|
| 3 | 1 | 1 | 1 | $1,5x10^{-3}$ | 8,3 |
| 4 | 1 | 2 | 1 | $2,8x10^{-4}$ | 10,7 |
| 5 | 1 | 2 | 2 | $4,6x10^{-5}$ | 13,4 |
| 6 | 1 | 3 | 2 | $8,2x10^{-6}$ | 15,9 |
| 7 | 1 | 3 | 3 | $1,4x10^{-6}$ | 18,5 |
| 8 | 1 | 4 | 3 | $2,4x10^{-7}$ | 21,0 |
| 9 | 1 | 4 | 4 | $4,0x10^{-8}$ | 23,5 |
| 10 | 1 | 5 | 4 | $7,0x10^{-9}$ | 26,0 |
| 11 | 1 | 5 | 5 | $1,2x10^{-9}$ | 28,6 |
| 12 | 1 | 6 | 5 | $2,0x10^{-10}$ | 31,1 |
| 13 | 1 | 6 | 6 | $3,5x10^{-11}$ | 33,7 |

Арктангенс, будучи оберненою функцією тангенса, для забезпечення масштабованості обчислень повинен бути визначений у діапазоні від нуля до одиниці, таблиця 2.8.

*Таблиця 2.8*

Кількість поліномів та точність обчислень функції арктангенса на проміжку 0..1

| Степінь полінома | X | kX | $X^2$ | ±ΔX | Число точних розрядів |
|---|---|---|---|---|---|
| 3 | 1 | 2 | 0 | $1,1x10^{-3}$ | 8,8 |
| 4 | 1 | 2 | 1 | $1,1x10^{-4}$ | 12,2 |
| 5 | 1 | 3 | 1 | $2,1x10^{-5}$ | 14,5 |
| 6 | 1 | 3 | 2 | $6,4x10^{-6}$ | 16,3 |
| 7 | 1 | 3 | 3 | $4,1x10^{-7}$ | 20,2 |
| 8 | 1 | 4 | 3 | $1,9x10^{-7}$ | 21,3 |
| 9 | 1 | 5 | 3 | $4,2x10^{-8}$ | 23,5 |
| 10 | 1 | 5 | 4 | $1,8x10^{-9}$ | 28,0 |
| 11 | 1 | 6 | 4 | $1,7x10^{-9}$ | 28,1 |
| 12 | 1 | 6 | 5 | $2,7x10^{-10}$ | 30,8 |
| 13 | 1 | 6 | 6 | $2,9x10^{-11}$ | 34,0 |

Дані щодо точності обчислень, що були подані у таблицях до чотирьох функцій, узагальнені на графіку рис. 2.10.

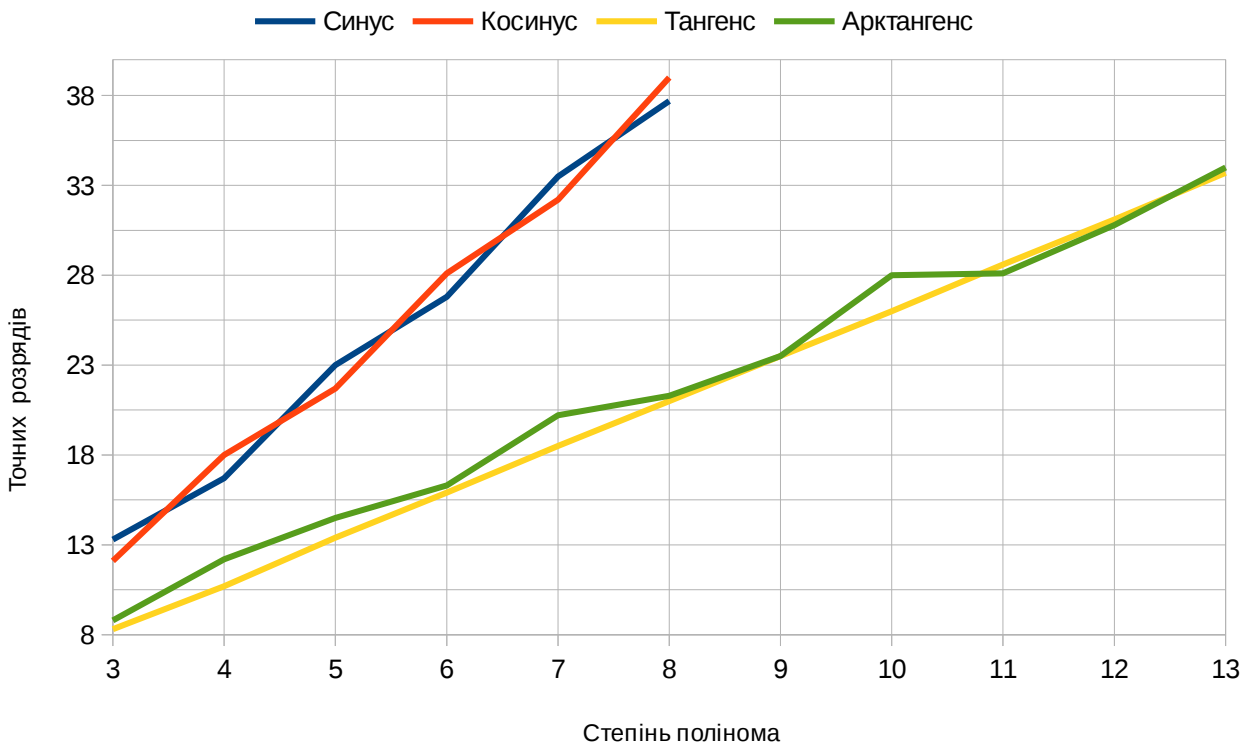



Рис. 2.10 Графік залежності порядку полінома та точності обчислення функції

При наближенні поліномами високих порядків високою точністю (найкращим наближенням) володіють функції синуса-косинуса, а також їх гіперболічні аналоги. Так, з допомогою полінома тринадцятого порядку можна обчислити будь-яку з представлених вище функцій з точністю понад 32 двійкові розряди, маючи при цьому достатній запас вірних біт. Так, функція косинуса буде мати тридцять дев’ять точних розрядів при використанні полінома восьмої степені.

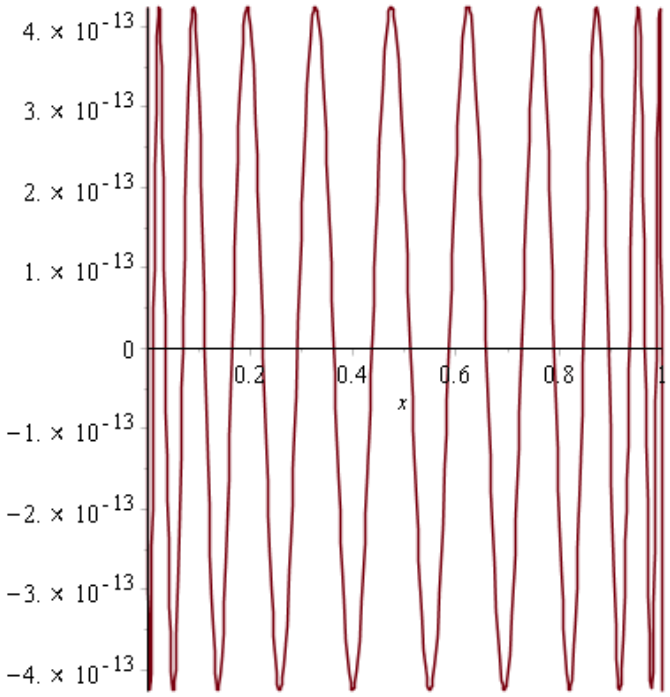



Рис. 2.11 Похибка обчислення функції косинуса поліномом восьмого порядку

Дещо меншу точність мають функції (у порядку спадання якості наближення) експоненти, логарифму, квадратного кореня, інверсного кореня та ділення. Функції тангенса та арктангенса є одні з найменш ефективних функцій, які ще можливо послідовно наближувати за допомогою степеневих поліномів. Тому на графіку 2.10 подано дві найбільш віддалені між собою групи функцій з точки їхньої зору точності, тоді як решта згаданих вище наближень будуть лежати в проміжку між синусом-косинусом і тангенсом. Крім того, існує ряд функцій, які не вдається наближати за допомогою поліномів. До них наприклад належать арксинус та арккосинус. Порядок полінома практично не впливає на точність їх обчислення, а після одержання полінома дев’ятого степеня не представляється можливим обчислити коефіцієнти поліномів вищих порядків. Максимальну точність, яку вдалось досягнути для даних функцій, мають 4,5 двійкові розряди, що становить ± 0.02195 в десятковому представленні, рис. 2.12.

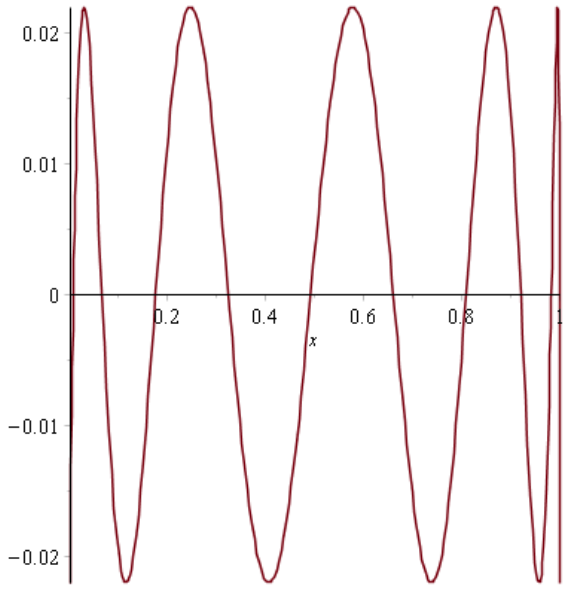



Рис. 2.12 Похибка обчислення функції арксинуса поліномом дев’ятого порядку

## Висновки до другого розділу

1. Вперше описано метод перекодування кута з широким діапазоном таблиці попередньої вибірки та мінімізацією кількості пам’яті з урахуванням можливостей вибраної платформи. Спрощення архітектури алгоритму шляхом усунення з нього конвеєра для повороту модуля вектора та таблиці коефіцієнтів кутів.
2. На основі методу перекодування кута вперше розроблено спосіб із представленням та оперуванням лише додатними величинами, що забезпечує коректну роботу алгоритму для беззнакової логіки та спрощує проектування та апаратні затрати при описі схем мовами вищого рівня.
3. Запропоновано метод інверсного повороту, який функціонує паралельно з алгоритмом одностороннього повороту та дає змогу вдвічі зменшити кількість ітерацій методу при будь-якому вхідному значенні аргументу.
4. Запропоновано спосіб паралельної поліноміально - квадратичної інтерполяції, який дає змогу скоротити кількість ітерацій та час обчислення широкого спектру функцій за рахунок паралельного використання

помножувачів. Спосіб ідеально підходить для систем, у яких доступно декілька широко-розрядних помножувачів.

5. Запропоновано гібридне поєднання паралельної поліноміально - квадратичної інтерполяції із використанням таблиці попередньої вибірки, що дає змогу зменшувати кількість поліномів та множень за рахунок ширшого використання пам’яті без обмежень на мінімальний об’єм таблиці, та при невеликих розрядностях використовувати поліноми лише першого порядку.

6. Отримав подальший розвиток метод залишкового множення (кусково-лінійної апроксимації), в якому збільшення розрядності обчислень в півтора рази здійснено через використання квадратичної апроксимації при обчисленнях вищих порядків залишкового кута без використання додаткових коефіцієнтів та зміни розрядності аргументів у діапазоні обчислень.

# РОЗДІЛ 3
# Апаратна реалізація пропонованих методів засобами ПЛІС

## 3.1 Програмне та апаратне забезпечення

Основна частина алгоритмів, представлених в роботі, розроблена на базі програмних засобів від Intel FPGA: Quartus, та ModelSim - Intel Edition. Апаратна реалізація була здійснена на платформі з кристалом Cyclone III EP3C16F484C6N, розміщеного на платі Terasic DE-0. Алгоритми та тестбенчі реалізовані мовою SystemVerilog, у яких частина коду, що відповідає за тестування та взаємодію між користувачем і ресурсами плати, винесена в окремий модуль, який не використовувався при синтезі та імплементації при одержанні кінцевих результатів.

## 3.2 Архітектура ПЛІС

Ринок ПЛІС поділений в основному між двома компаніями: Xillinx та Intel. Весь асортимент ПЛІС дані компанії ділять на три категорії: високошвидкісні ПЛІС, ПЛІС середнього рівня (mid-range) та бюджетні ПЛІС. Для Intel це: Stratix, Aria, Cyclone; а для Xilinx: Virtex, Kintex та Spartan, який в подальшому був замінений лінійкою Artix. Алгоритми, представлені у роботі, не вимагають значних обчислювальних ресурсів, а реалізувати їх можна на основі бюджетних моделей ПЛІС, не зіткнувшись з обмеженнями, викликаними характеристиками кристалу.

Апаратна реалізація алгоритмів здійснена на платформі CycloneIII, відлагоджувальної плати Terasic DE-0. Остання версія програмного забезпечення Quartus, яка підтримує даний кристал — 13.1.4. В наступних версіях програмного забезпечення відсутнє сімейство CycloneIII, і на його місце вибраний кристал CycloneIV з аналогічними характеристиками. Можна з високою долею ймовірності сказати, що згаданий вище CycloneIV містить ребрендингований

кристал CycloneIII, перейменований виробником у маркетингових цілях. Аналогом CycloneIII від фірми Xillinx вибрано модель серії Artinx-7. Даний кристал на сьогодні є найслабшим серед усіх представлених виробником моделей, чим подібний до вищезгаданого CycloneIV у поточній лінійці ПЛІС від Intel.

В кристалі ПЛІС можуть міститись різноманітні блоки, такі як трансивери, контролери пам’яті чи цілі ARM процесори. Звичайно, що в бюджетних моделях більшість таких блоків відсутні, тому зазначимо лише основні елементи, які використовуватись при дослідженнях і здатні вплинути на кінцеві результати.

### 3.2.1 Логічні елементи ПЛІС

Логічний елемент є основним архітектурним блоком ПЛІС. Кількість елементів на кристалі, як правило, становить від декількох тисяч до декількох мільйонів екземплярів. Протягом останніх років будова блоку логічного елемента була дещо змінена, і зараз на ринку присутні ПЛІС двох видів: із класичними 4-х входовими логічними елементами (4-cell) та перспективнішими 6-ти входовими (6-cell). Якщо заглянути в сам логічний елемент, то в ньому виділяються два основних конструктивних блоки: переглядова таблиця (ЛУТ) — апаратна реалізація таблиці істинності деякої функції та один чи декілька тригерів. Переглядову таблицю можна розглядати як блок динамічної пам’яті, комірки якої заповнюються на етапі конфігурування ПЛІС при її включенні. На сьогоднішній день існують також кристали FPGA, конфігурацію логічних елементів яких можна змінювати під час їх функціонування, відомі також, як реконфігуровані ПЛІС, але такі кристали не викликають інтересу в межах даного дослідження.

Зміну архітектури логічних елементів можна розглядати як наслідок дії закону Мура, коли через неперервне збільшення кількості логічних елементів на кристалі виникла необхідність у об’єднанні декількох логічних елементів. Так, шестивходовий елемент ЛУТ по суті являє собою таблицю істинності із шістдесяти чотирьох розрядів і дозволяє у деяких випадках замінити до шести

чотирьох входових елементів одним шести входовим ЛУТом.

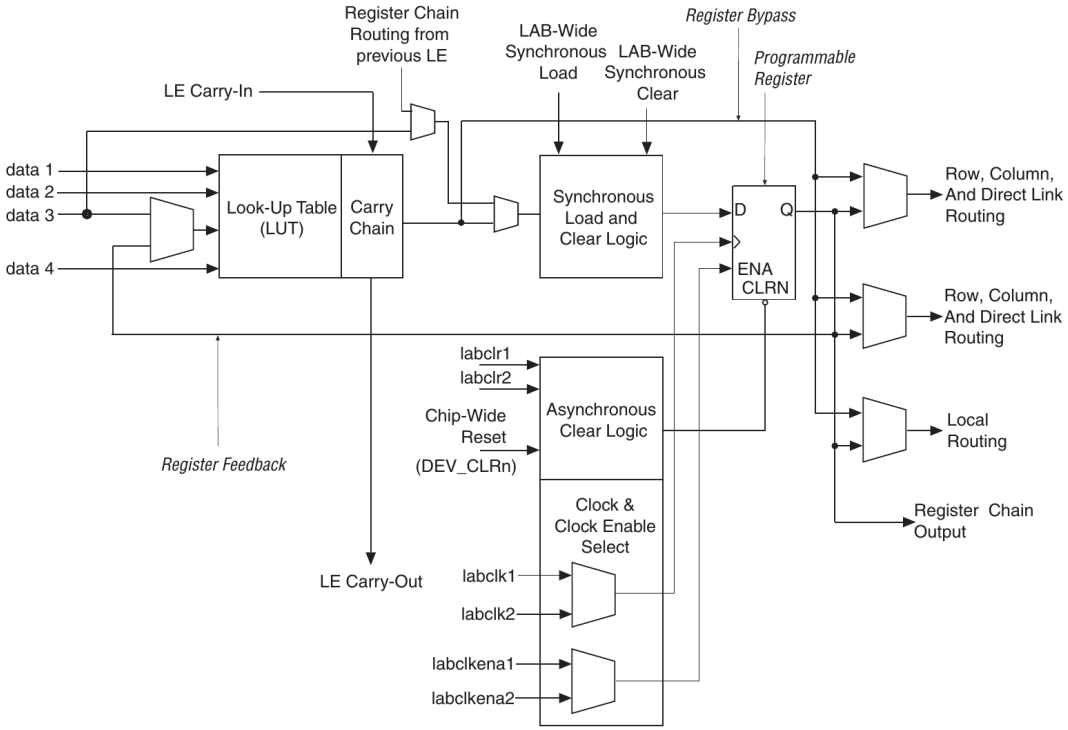



Рис. 3.1 Принципова схема 4-х входового логічного елемента ПЛІС від Інтел

Основне призначення тригерів полягає у збереженні результату функції логічного елемента, що є необхідною умовою для секвенційної логіки. Розміри проектів для ПЛІС постійно збільшуються, разом із необхідною кількістю логічних елементів для їх реалізації, що призводить. до збільшення числа сигнальних ліній між елементами. При цьому деякі сигнальні лінії повинні з’єднати віддалені між собою частини кристалу, тоді як центральна частина міжелементних зв’язків кристалу зайнята, і топологія шляху проходження сигналу відбувається не оптимальним чином. Даний ефект приводить до падіння тактової частоти ПЛІС, і, щоб утримати її у заданих межах, на шляху сигналу вставляються додаткові тригери, які за рахунок збільшення латентності дозволяють зберегти задану тактову частоту. В цьому випадку при оптимізації проекту значно збільшується загальна кількість використаних тригерів без зміни числа логічних елементів. Відповідно у сучасній архітектурі логічного елемента фірми Xillinx та Інтел оправданим є вмонтування двох або чотирьох тригерів, а не одного, як це було до недавнього часу.

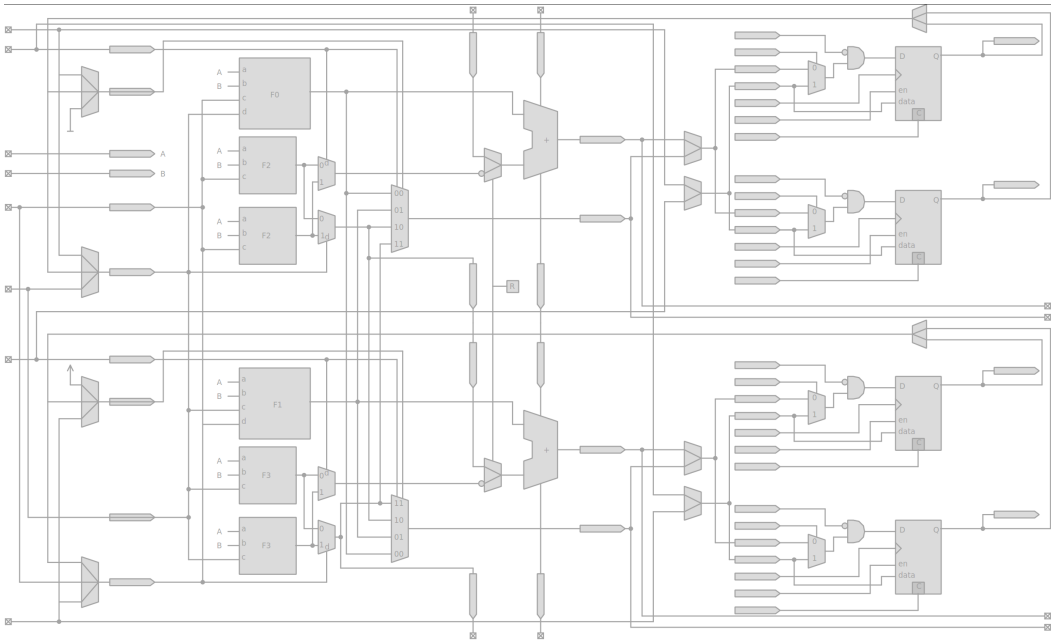


Рис. 3.2 Архітектура 6-ти входового логічного елементу

Також логічні елементи містять додаткові входи та виходи. З їх допомогою таблиці ЛУТ з’єднані із сусідніми комірками, а по лініях можуть передаватися розряди переносу у випадку функціонування ЛУТу в режимі суматора (arithmetic mode), чи розряди регістра зсуву. Також такими з’єднаннями є синхронні та асинхронні сигнали встановлення, скиду чи для визначення режиму роботи тригера. Завдяки цьому на основі апаратно реалізованого D-тригера у комбінації з таблицею ЛУТ можна синтезувати будь-який інший поширений тригер.

### 3.2.2 Архітектура апаратного помножувача

Блок помножувача у складі сучасної ПЛІС здатен оперувати як знаковими, так і беззнаковими цілочисельними величини. Кількість розрядів помножувача у більшості випадків кратна дев’яти. Також на входах та виходах блоків розміщують тригери, призначені для максимального зменшення шляху сигналу між тактовими імпульсами і забезпечення високої продуктивності функціонування помножувачів. Цей факт особливо важливий, якщо згенерована схема з логічних елементів має вищу частоту, ніж блок помножувача чи декількох помножувачів. У новіших версіях ПЛІС в складі ДСП блоків, крім помножувачів, використовується також

додаткова логіка, суматори та тригери, призначені для обчислення поширених функцій та фільтрів для засобів обробки мультимедійних даних. Такими, в основному, виступають множення з накопиченням, фільтр із скінченною імпульсною характеристикою та ін. Блоки помножувача мають нижчу швидкість, ніж блоки пам’яті та суматори такої ж розрядності, синтезовані з логічних елементів. Тому у невеликих проектах частота ДСП блоку часто обмежує загальну продуктивність ПЛІС. Серед алгоритмів, представлених у даній роботі, такий ефект часто спостерігався при використанні обчислень невеликої розрядності, в межах 20 розрядів. При подальшому збільшенні розрядності та складності схеми алгоритму швидкодія ДСП блоку перестає бути стримуючим фактором. Незважаючи на обмеження швидкодії, викликане використанням помножувачів, їхня реалізація у вигляді апаратних блоків у кілька разів ефективніша, ніж імплементація аналогічної функціональності за допомогою логічних елементів. Цей факт, а також універсальність дії множення служить причиною апаратної інтеграції в архітектуру практично усіх ПЛІС цієї операції у вигляді окремого блоку. Основною ж відмінністю між різними видами боків помножувачів є хіба що їхня розрядність. Так, для бюджетних ПЛІС від Інтела один такий блок може здійснювати множення двох чисел розрядністю 18 на 18 або працювати в режимі двох помножувачів 9 на 9 розрядів. У дорожчих моделях розрядність, як правило, збільшена до величин 27 на 27, і серед доступних конфігурацій з’являються такі режими, як 36 на 18, чи кілька помножувачів 9 на 9, або 18 на 18. Для фірми Xilinx стандартною є розрядність помножувача 27 на 18, хоча в обох фірмах існують деякі винятки, наприклад, помножувачі 25 на 18 чи 18 на 19. Тому розрядність помножувача для кожної моделі ПЛІС доводиться уточнювати. До цього слід додати, що у маркетингових цілях за величину кількості помножувачів часто видається не кількість блоків помножувачів, а загальна сума помножувачів у випадку їх функціонування при мінімальній розрядності, а один апаратний блок може реалізувати декілька таких помножувачів.

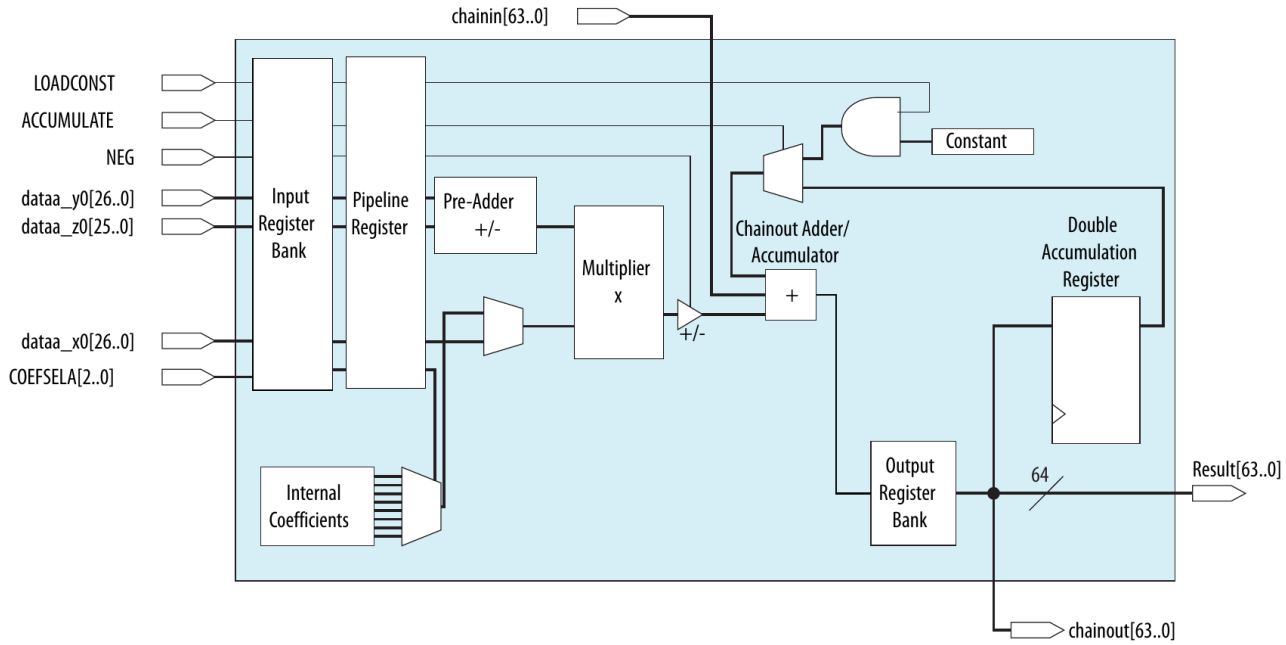



Рис. 3.3 Вигляд ДСП блоку сучасної ПЛІС від Інтел

Помножувачі у CycloneIII можуть використовуватись у режимах дев’ять на дев’ять чи вісімнадцять на вісімнадцять біт. Якщо розрядність операнду менша за розрядність помножувача, це не впливає на швидкість блоку помножувача, так само, як і використання знакових та беззнакових чисел. Невелика різниця спостерігається лише при переході між 9-ти та 18-ти розрядним режимом помножувача і становить менше 8%. Всього в CycloneIII розміщено 112 блоків з розрядністю дев’ять на дев’ять, а для обчислень з розрядністю вісімнадцять на вісімнадцять використовується пара таких блоків, тобто 56 елементів. В основному, використання сусідніх блоків не впливає на швидкодію, але якщо вхідний сигнал блоку має розгалуження також на інші входи, максимальна частота $f_{max}$ може дещо знизитись. Цей ефект спостерігається при підключенні помножувачів каскадним способом, що, безумовно, є необхідною умовою великої кількості блоків і обумовлюється довжиною комунікаційних ліній (interconnect). При подальшому збільшенні кількості розрядів компілятор згенерує схему “множенням в стовпчик”, що буде вимагати більшого числа помножувачів та деякої кількості логічних елементів для сумування результатів блоків. Падіння швидкодії при цьому спостерігається таке ж, як і при використанні програмних

блоків, тобто на третину від швидкості блоку помножувача для чисел в межах тридцяти шести розрядів. При збільшенні розрядності удвічі — до сімдесяти двох розрядів, частота зменшується приблизно в п’ять разів, до частоти в 72 МГц. Звичайно, що при такому підході можна розбити схему на частини і виконувати множення за декілька тактів. Проте в цьому випадку не слід забувати, що множення в стовпчик хоч і буде виконуватись на максимальній частоті блоку помножувача (оскільки операція знаходження суми, необхідна при множенні між логічними елементами, виконується швидше), проте не буде оптимальна по кількості використаних ресурсів. Звичайно, існують і кращі алгоритми множення, але їх реалізація є складнішою і виграш при застосуванні цих методів починається з декількох тисяч розрядів. Враховуючи, що для реалізації множення великої кількості розрядів ДСП елементів не вистачить і у високотехнологічних ПЛІС, підхід до задання алгоритму обчислення буде ітераційний, а модуль для макрооперацій буде синтезований одним із вищезгаданих методів. Наприклад, топові ПЛІС при використанні усіх ДСП блоків здатні помножити числа розрядністю:

Intel FPGA Virtex7 3600 DSP 27x27 = 1620 біт;

Xillix KintexUltraScale 5520 DSP ≈ 2000 біт.

У блоці помножувача можна задіювати тригери, розміщені на вході та на виході елемента. Латентність в такому випадку буде вдвічі вища за кількість блоків. Схема буде функціонувати на максимальній частоті ДСП блоків. Максимальна похибка при цьому лежить у межах 1MHz, більша за 326MHz (з 56-ти блоками) незалежно від розрядності — від десяти до вісімнадцяти розрядів;

Таблиця 3.1

Падіння швидкодії при збільшенні розрядності помножувача понад розмір апаратного блоку

| Розрядність | Частота при 0С | Частота при 85С | Блоків | Логіч. елем. | Триггерів |
|---|---|---|---|---|---|
| 19x19 | 201,69 | 177,75 | 5111 | 5312 | 4256 |
| 20x20 | 176,40 | 156,32 | 9903 | 8792 | 4480 |
| 21x21 | 137,31 | 122,90 | 14349 | 13832 | 4704 |

Як бачимо, частота при незначному збільшенні розрядності різко падає до швидкості програмної реалізації помножувача. При цьому використовується близько 93% ресурсів плати, а подальше збільшення розрядності на даному кристалі - неможливе.

Отже, при використанні апаратних помножувачів краще задіювати обидва (вхідний та вихідний) Д-тригери, частота в цьому випадку буде максимальна незалежно від кількості ДСП. При програмній реалізації двотактний помножувач не дає збільшення частоти, оскільки частота програмно згенерованих блоків є втричі нижчою за їх апаратний аналог. Щоправда, при задіянні всіх апаратних блоків кристалу максимальна частота може знизитись до 30 %. А при недостачі блоків функціональність блоку помножувача буде за замовчуванням емулюватись за допомогою вільних логічних елементів, що здійснюється компілятором автоматично. Як вже було показано, швидкість такого блоку буде втричі нижчою від апаратного помножувача, а частота практично не залежить від кількості синтезованих програмних блоків і лежить у межах 9% для CycloneIII. Так, для реалізації одного програмного помножувача потрібно 433 логічні елементи та 54 регістри. При синтезі більшої кількості помножувачів кількість зайнятих логічних елементів дещо знижується і становить 1573 регістри та 15302 логічні елементи (15305 комірок) із наявних 15408 на досліджуваній ПЛІС. Отже, степінь заповнення кристалу з 36-ма помножувачами становить (15305 ЛЕ / 15408 ЛЕ = 99.33%) і є максимально можливим значенням для вибраної платформи. В цьому випадку після оптимізації проекту для реалізації одного помножувача отримаємо середнє значення 15302 / 36 = 425 ЛУТів та 1573 / 36 = 44 регістри.

*Таблиця 3.2*

Тактові частоти при множенні знакових та беззнакових чисел

| Режим помножувача | 0С° мгц | 85С° мгц |
|---|---|---|
| 9x9 | 353,11 | 343,29 |
| 18x18 | 327,23 | 288,85 |
| 2 x 2 помнож. | 250,06 | 223,06 |
| 2 x 3 помнож. | 236,46 | 211,28 |

| | | |
|---|---|---|
| 2 x 5 помнож. | 233,37 | 208,72 |
| 2 x 10 помнож. | 235,57 | 210,35 |
| 2 x 16 помнож. | 222,72 | 199,96 |
| 2 x 56 помнож. | 211,69 | 188,25 |
| Програмний помнож. | 123,56 | 110,31 |
| 36 програмних помн. | 112,89 | 100,92 |

Якщо розрядність операндів є обмеженою, то для засобів мови Verilog множення знакових чисел на беззнакові є досить нетривіальною задачею [128]. Згідно зі стандартом мови Verilog при множенні знакових чисел результат також має знак, а при множенні беззнакових - немає. У випадку, якщо множиться знакове число на беззнакове, то за стандартом результатом число повинно бути беззнакове, тоді як для реалізації залишкового множення алгоритму CORDIC необхідно здійснити таку операцію і отримати число зі знаком. Якщо перетворити беззнакове число на знакове, то одержимо ще один старший розряд, значення якого завжди нуль. В цьому випадку компілятору не вдається використати один ДСП блок, і через цей один “зайвий” нульовий знаковий біт буде використано ще один помножувач. Доцільним в цій ситуації було б використання технологічних розрядів помножувача (SIGN A/B), які відповідають за представлення результату. На жаль, редагування даного біту в редакторі файлу параметрів помножувача не вдалось. Також таке редагування є неприйнятне при використанні великої кількості помножувачів. Другим варіантом виконання множення є використання мегафункції, однак стандартна мегафункція для ДСП блоку не має можливості для задання знаку кожного операнду окремо. Тому таким способом задаються лише операції, які можливо здійснити засобами мови Verilog, а саме: множити знакові та беззнакові числа, але не їх комбінації.

Ще одним способом є застосування складніших мегафункцій на основі множення з накопиченням. В цьому випадку стає можливим задання представлення кожного з чисел окремо, але в такому разі файл конфігурації помножувача виходить досить громіздким, з великою кількістю надлишкових директив. Також у цьому випадку зникає змога використання вбудованих в

помножувач блоків тригерів, режиму роботи за тактовим сигналом, а також зниження тактової частоти через довші міжелементні інтерконекти. Перенесення параметрів, що задають представлення чисел зі складнішої мегафункції “добуток з накопиченням” в класичну мегафункцію помножувача, результату не дало. Додані директиви не сприймаються компілятором, а підключення сигналів самостійно наштовхуються на зауваження компілятора про відсутність таких елементів в інтерфейсі модуля.

В загальному, щодо відмінностей між програмними ДСП блоками, можна зробити такі висновки:

- Частота програмних блоків втричі нижча за частоту апаратних ДСП блоків.
- Кількість програмних блоків практично не впливає на частоту і лежить в межах 9% незалежно від їх кількості.
- При каскадному підключенні блоків кількість необхідних логічних елементів зменшується.
- Швидкість програмних блоків значно нижча, а зайняті ресурси — суттєві.
- Програмно можна синтезувати лише до 70% з наявних апаратних блоків.
- Практичність використання програних блоків помножувачів навряд чи доцільна та знаходиться під сумнівом.

### 3.2.3 Вбудована пам’ять ПЛІС.

Так само, як і блоки помножувачів, блоки пам’яті дещо відрізняються один від одного залежно від моделі ПЛІС. Основним критерієм блоку є його обсяг, виміряний в бітах і який дорівнює деякій степені двійки, помноженій на дев’ять. Загалом, обсяг пам’яті у ПЛІС становить від десятків кілобіт до десятків мегабіт. Здебільшого необхідний обсяг пам’яті для проектів рідко перевищує один чи декілька мегабіт, а при потребі використання пам’яті великого обсягу задіюють зовнішні модулі, для використання яких частина моделей ПЛІС містять інтегрований контролер пам’яті.

Блоки пам’яті можуть працювати при різних розрядностях вхідних і вихідних даних, а також використовуватись як буфери (FIFO). Вони мають можливість одночасного зчитування та запису інформації протягом одного такту (true dual port). Зрозуміло, що адреси вхідних та вихідних даних в останньому випадку мають бути різними, інакше точність отриманих даних не гарантується, про що заздалегідь попереджає компілятор. Також слід враховувати, що при роботі пам’яті для розрядностей, які не є кратними доступним режимам роботи її блоку, частина пам’яті не буде задіюватись. Як показали досліди в цьому напрямі, втрата пам’яті може досягати 30% від її обсягу. З іншого боку, при використання всього обсягу пам’яті як одного масиву виникає втрата швидкодії, зумовлена збільшенням відстані проходження сигналів, викликаної нерівномірністю відстаней до різних її блоків. Також, серед методів ініціалізації пам’яті виробник пропонує директиви, які не стандартизовані та не сприймаються іншими середовищами та компіляторами.

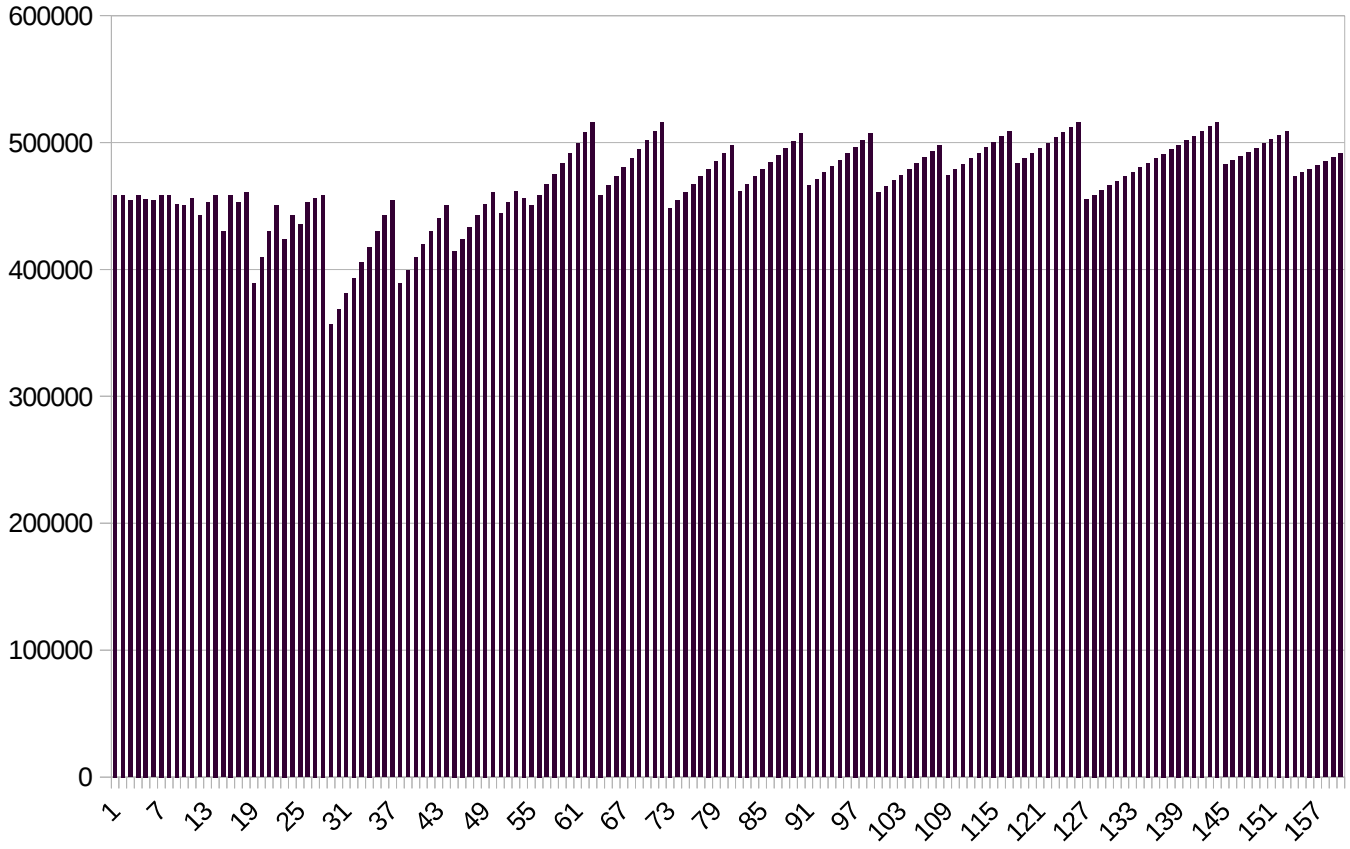



Рис. 3.4 Розподіл пам’яті залежно від розрядності шини

### 3.2.4 Блок фазового автопідстроювання частоти.

Блоки фазового автопідстроювання частоти (PLL - phase-locked loop) інтегровані безпосередньо в ПЛІС і дозволяють задавати тактові частоти для неї, маючи на вході лише один зовнішній тактовий генератор. Плата для дослідів містить один кварц частотою в 50МГц, який може бути використаний, як генератор тактового сигналу. Основним завданням блоків PLL та вбудованої мегафункції для їх налаштування є підбір значення множників та дільників основного тактового сигналу для забезпечення роботи ПЛІС на бажаних для розробника частотах. Мінімальна вхідна частота для блоку PLL становить 5МГц, а максимальна - 473.709МГц. Верхня межа є явно завищеною, оскільки коректна робота зовнішніх виводів даної ПЛІС на частотах вищих за 250МГц не передбачена.

Щодо множників та дільників частоти, то вони можуть бути лише цілочисельні, а максимальні їхні значення становлять 26 для множника та 43520 для дільника. Комбінуючи ці значення, можливо задавати робочі частоти для ПЛІС в широких межах із високою точністю. Так, в межах 1 гігагерца можливо встановлювати частоти з точністю до мегагерца. При нижчих частотах точність значно зростає. Максимальне значення частоти для плати DE-0 становить $50 \cdot 26 = 1300$МГц., мінімальне — 1148.9Гц.

Всього ПЛІС містить чотири PLL блоки, якими також можна регулювати скважність тактового сигналу (duty cycle), здійснювати синхронізацію сигналів та динамічно змінювати частоти безпосередньо під час роботи ПЛІС. Останній фактор зручно використовувати для регулювання споживаної потужності залежно від температури ПЛІС чи поточного навантаження на неї.

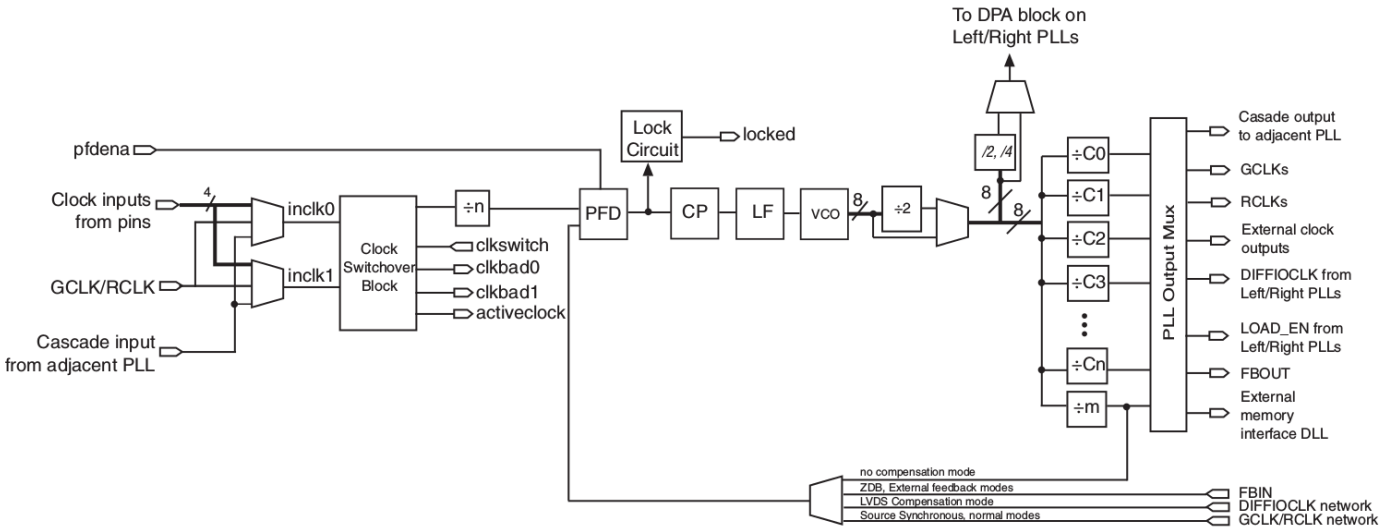



Рис. 3.5 Схема PLL блоку IntelFPGA

### 3.2.5 Конструкція вводів-виводів ПЛІС

ПЛІС містить значну кількість вводів-виводів, число яких переважно становить декілька сотень. Кількість таких виводів залежить від типу корпусу, в який поміщена ПЛІС і може відрізнятись для ідентичних за будовою чіпів. Також, залежно від корпусу, може змінюватись і максимально допустима температура ПЛІС. Слід виділити технічні та програмовані виводи. Перший клас виводів служить для подачі живлення, підключення програматорів та інших жорстко фіксованих цілей. Функції програмованих виводів визначаються на етапі конфігурації ПЛІС і можуть служити як входом, так і виходом сигналу. Доступний також режим двонаправленої передачі, але він не часто використовується — в основному, для підключення до загальних шин, чи пристроїв з обмеженою кількістю виводів. Напруга і сила струму на виходах визначається для кожного виводу зокрема та знаходиться в межах 1.2 - 3.3 вольти. Максимальна сила струму, рекомендована для стабільної роботи, становить 16мА, що співрозмірно з силою струму на виходах мікроконтролера. Всього ж конфігурації виводів підтримують кілька десятків стандартів для узгодження логіки різних архітектур з можливістю вибору сили струмів на різних рівнях. Слід зауважити, що буфери виводів ПЛІС можуть споживати значно більшу потужність, ніж сама ПЛІС, що призводить до

надлишкового тепловиділення і утруднює коректний розрахунок загальної споживаної потужності кристалу. Підключення зовнішніх елементів до виводів кристалу, в основному, визначається та задається виробником плати, на якій міститься ПЛІС, та подається у документації до неї.

### 3.2.6 Суматори та метод швидкого додавання

Реалізацію функцій додавання та віднімання в архітектурі ПЛІС виконують логічні елементи у спеціально передбаченому для такого випадку арифметичному режимі. У 4-х входових логічних елементах можна реалізувати повний суматор, а у 6-ти входових — два. Розрядність суматора в ПЛІС обмежена кількістю логічних елементів, розміщених в одному стовпці. Весь внутрішній простір ПЛІС поділений на блоки. По периметру в блоках розміщуються буфери вводу-виводу та PLL. У внутрішніх блоках може бути розміщено блок пам’яті, ДСП блок, або логічні елементи, кількість яких у нашому випадку дорівнює 15,4 тисяч. Можна помітити, що якщо кількість таких блоків по вертикалі дорівнює 28, то, знаючи кількість таких стовпців, можна визначити не тільки максимальні характеристики каскаду суматорів, але й кількість пам’яті та ДСП блоків, які також займають увесь стовпець. Для суматорів ця величина становить $16 \cdot 28 = 448$ розрядів. Подальше збільшення розрядності обмежено знаком переносу, який повинен пройти по вертикалі усю відстань між протилежними краями ПЛІС. Тому компілятор не імплементує дій додавання, віднімання змінних такої чи більшої розрядності. Часто при збільшенні розрядності змінних в алгоритмі виникає падіння швидкодії саме на каскадах суматорів. Спостерігати цей ефект можна при збільшенні розрядності обчислень класичним методом CORDIC, реалізація якого в ПЛІС базується в основному на суматорах та компараторах, які також є підвидом суматора. Щоб зрозуміти степінь оптимізованості проекту, швидкодія якого впирається в швидкість роботи деякого суматора, рекомендовано використовувати таблицю з максимальними частотами для різних розрядностей суматорів. Тоді,

зіставляючи дані таблиці та розрядність критичного по швидкості суматора, можна визначити рівень, до якого можна покращувати проект.

*Таблиця 3.3*

Швидкодія суматорів для аргументів різних розрядностей

| Bits | 9 | 18 | 36 | 72 | 114 | 128 | 256 | 448 |
|---|---|---|---|---|---|---|---|---|
| Ω 0° | 508,13 | 400 | 279,56 | 176,15 | 125,49 | 117,34 | 62,53 | 36,89 |
| Ω 85° | 575,04 | 455,58 | 320,1 | 202,35 | 144,3 | 135,74 | 72,43 | 42,7 |
| Ωmax | 817 | 702,74 | 487,33 | 303,67 | 216,4 | 201,2 | 106,9862 | 63,06 |

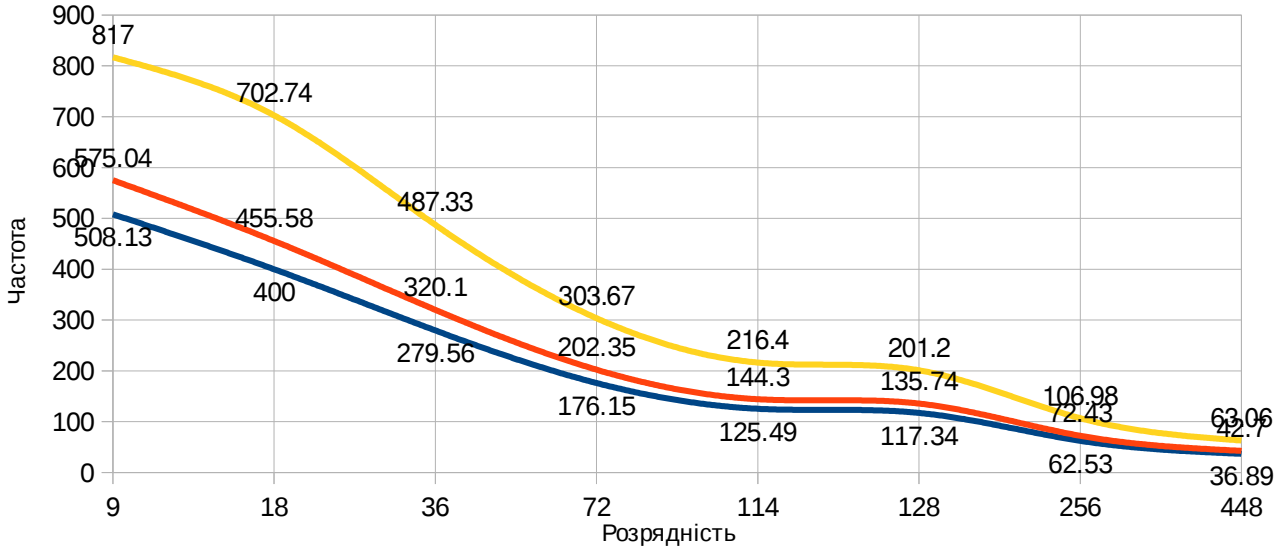



Рис 3.6 Максимальна частота суматора залежно від його розрядності

### 3.2.7 Співставлення апаратної бази ПЛІС: CycloneIII та Artix-7

Для практичної реалізації пропонованих методів обчислень засобами Vivado, аналогом кристалу CycloneIII від фірми Xillinx вибрано модель серії Artinx-7. Даний кристал є найслабшим серед усіх представлених виробником моделей, які підтримуються поточними версіями середовищ розробки, чим він подібний до вищезгаданого CycloneIV у поточній лінійці ПЛІС від Altera. Такий кристал є прикладом мінімальності характеристик серед FPGA чіпів, що випускаються на сьогоднішній день. Слід відзначити, що Artinx-7 містить новішу

архітектуру, подібно до CycloneV, в яких використовуються 6-ти вхідні ЛУТи та широкорозрядні помножувачі у складі ДСП блоків.

*Таблиця 3.4*

Порівняння використовуваних ПЛІС від виробників Altera та Xilinx

| | CycloneIII EP3C16F484C6N | Artinx-7 XC7A15TFGG484-3 |
|---|---|---|
| Logic Elements | 15408 | 10400 |
| Flip-flops | 15408 | 20800 |
| SRAM bit | 516096 | 921600 |
| Memory Blocks | 56 blocks 1024x9bit | 25 blocks 4096x9bit |
| DSP Blocks | 56 18x18 | 25 25x18 |
| User I/O | 345 | 250 |
| Package | FBGA 484 | FGG 484 |
| Speed Grade | 6 (6..8) | -3 (-1..-3) |
| Temperature rage | 0..85Cº | 0..85Cº (100) |
| Voltage | 1,2 (1,15-1,25) | 1,0 (0,95-1,05) |
| Price | $ 52,0 | $ 52,3 |

Основною відмінністю за технічними параметрами між кристалами в таблиці є обсяг пам’яті 900Кбіт у Artix7 проти 500Кбіт у Cyclone. Потрібно враховувати, що для реалізації більшості алгоритмів використовується не вся пам’ять ПЛІС, через що даний параметр не є критичним. Аналогічне твердження справедливе також для помножувачів, де за замовчуванням використовуються декілька з них у конфігурації 18x18. Корпуси в кристалів ідентичні, але з різною кількістю програмованих виходів. Для розглянутих алгоритмів кількість у 100 виходів є достатньою. Проте більша їх кількість ставить Artix-7 на один рівень з CycloneIII та допомагає у відлагодженні ряду проектів. Спід грейд вибрано в обох випадках максимальним для даної серії. Тут слід звернути увагу, що Artix-7 є новішою ПЛІС із меншим техпроцесом виробництва, що забезпечує їй кращі часові характеристики та енергоспоживання.

Щодо кількості логічних елементів та тригерів, то адекватне їх зіставлення потребує додаткового пояснення. Artix-7 випускається за новішою архітектурою, в якій розміщується один логічний елемент та два тригера. Це зв’язано з тенденцією збільшення розміру проектів та відповідно схем для їх реалізації. Критичною в цьому випадку постає довжина шляху між віддаленими одна від одної частинами схеми, що значно знижує тактову частоту. Для боротьби з цим явищем по шляху сигналу вставляються додаткові тригери, які за рахунок збільшення латентності дозволяють підняти частоту. Відповідно у великих проектах значно збільшується кількість тригерів без зміни кількості логічних елементів. Тому у сучасній архітектурі як фірма Xillinx, так і Altera використовують по два тригери, а не один, як це було до недавнього часу.

Новіша архітектура блоку Artix-7, з двома тригерами на один логічний елемент у даному випадку є надлишковою, оскільки невелика кількість логічних елементів не дозволяє повністю задіяти всі тригери, а кількість використаних логічних елементів приблизно відповідає кількості тригерів або навіть перевищує їх. Також архітектура логічних елементів, яка змінилась з 4- до 6-входової, не дозволяє однозначно зіставити дані ПЛІС. Наприклад, при використанні логічного елемента як інвертора, кожний логічний елемент із новішою архітектурою буде відповідати своєму старішому аналогу. В такому ідеалізованому випадку при зіставленні кристалів кількість логічних елементів у CycloneIII на 50% вища в порівнянні із Artix-7. Проте у випадку, коли вдається задіяти усі ресурси логічного елемента, для синтезу однієї 6-входової логічної функції знадобиться п’ять-шість 4-входових ЛУТів. У цьому разі кількість логічних елементів CycloneIII буде становити лише 30% від можливостей Artix-7. Відповідно, зіставити обидва варіанти архітектури можна тільки після синтезу кожного конкретного проекту під відповідну архітектуру. У середньостатистичному випадку виробник пропонує розглядати дану ПЛІС з 10 тис. логічних елементів шестирозрядної архітектури як ПЛІС з 15 тис. логічних елементів 4-розрядної архітектури, що і відповідає характеристикам CycloneIII.

## 3.3 Вибір формату даних для реалізації алгоритмів.

Вхідні та вихідні значення краще представлені в форматі з фіксованою комою [27]. Застосування рухомої коми при реалізації тригонометричних функцій в даному випадку є недоцільним, оскільки, як вхідні, так і вихідні значення функцій лежать у визначених межах. Так, для синуса і косинуса вхідне значення кута може лежати в діапазоні [0...π/2), а вихідні значення в межах [0...1).

Переваги формату з фіксованою комою:

- Формат з фіксованою комою легше реалізувати апаратно.
- Операції зсуву здійснюються монтажним чином, що дозволяє підвищити швидкодію та економити логічні елементи.
- У форматі відсутні надлишкові розряди знаку та порядку числа, які в цьому випадку не використовуються.
- Вихідні значення функцій можна без додаткових перетворень подавати на вхід ЦАП.
- При необхідності легше перетворити формат вихідних даних на число з рухомою комою, ніж здійснювати операції в цьому форматі всередині алгоритму. (Те саме стосується і вхідних даних).

Перевагами формату з рухомою комою можна назвати:

- Ширший діапазон значень операндів.
- Стандартизація формату.

Щодо діапазону значень, які приймають функції синуса та косинуса, то він є обмежений зверху значеннями [-1...1], відповідно значення порядку числа є константою. Збільшення точності обчислень (нижня межа) досягається збільшенням розрядності операндів.

Стандартизація формату представлення може бути викликана необхідністю узгодити вихідні формати різних функцій, діапазони яких суттєво відрізняються. Наприклад, діапазон значень гіперболічного синуса лежить в діапазоні (-∞...+∞) і також може бути обчислений алгоритмом CORDIC, щоправда в межах [-1.118 …

+1.118]. Тим не менше, першочерговою задачею при проведенні досліджень є досягнення оптимальних характеристик роботи алгоритму, а узгодження форматів може суттєво відрізнятись залежно від поставлених технічних умов. Виходячи з вищенаведених аргументів, формат з фіксованою комою краще підходить для поставленої задачі зіставлення алгоритмів, при якому виконання надлишкових операцій не може повною мірою розкрити всі якісні характеристики алгоритмів.

## 3.4 Обчислення тригонометричних функцій за допомогою вбудованих засобів середовищ проектування FPGA.

Із запропонованої виробником класичної реалізації методу CORDIC засобами вбудованої мегафункції для обчислення синуса чи косинуса необхідно виділити близько третини ресурсів кристалу (таблиця 3.5). Розрядність вхідних даних фіксована та становить тридцять два біти, а на виході алгоритму одержуємо одну із двох функцій — синус або косинус.

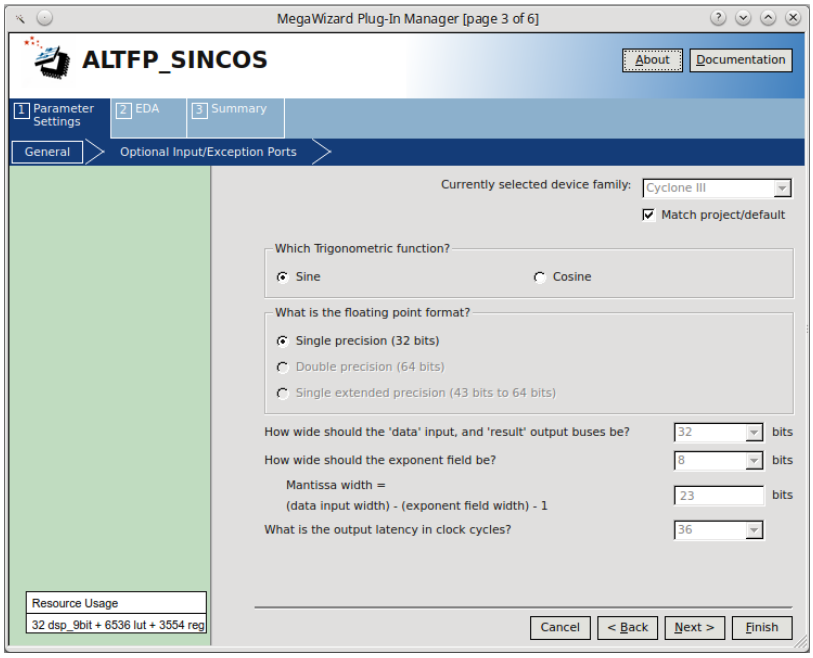



Рис. 3.7 Вікно з налаштування CORDIC Megafunction Wizard

Також мегафункція містить налаштування для збільшення розрядності методу, але у випадку використання CycloneIII єдиним прийнятним варіантом є 32-розрядна реалізація функції. Для більших кристалів можливий вибір між 43 -

64 розрядами вхідних та вихідних даних, а кількість необхідних елементів для реалізації буде збільшуватись як степенева функція при лінійному збільшенні розрядності обчислень. Саме через значне зростання необхідної кількості логічних елементів більші розрядності для досліджуваних ПЛІС не доступні, хоча мінімальна 32 - розрядна реалізація займає лише третину ресурсів CycloneIII.

*Таблиця 3.5*

Реалізація синуса та косинуса за допомогою інтегрованої мегафункції

| Bus bits | Clock | Bandwith Mbit/s | Latency ns | Blocks | Logic | Flip-flops | Mem bits | Freq. 85°C | Freq. 0°C | Freq. Max |
|---|---|---|---|---|---|---|---|---|---|---|
| 32 | 36 | 4372,16 | 263,49 | 5248 | 4996 | 2350 | 1362 | 136,63 | 152,91 | [243,7] |
| 32 | 36 | 4562,88 | 252,47 | 5056 | 4768 | 2247 | 320 | 142,59 | 158,15 | [246,6] |

Загальна RTL схема обчислення функції синуса зображена на рис. 3.6. Для спрощення відображення й так складної схеми алгоритму ключові елементи згруповані у блоки, які на схемі зображені у вигляді прямокутника.

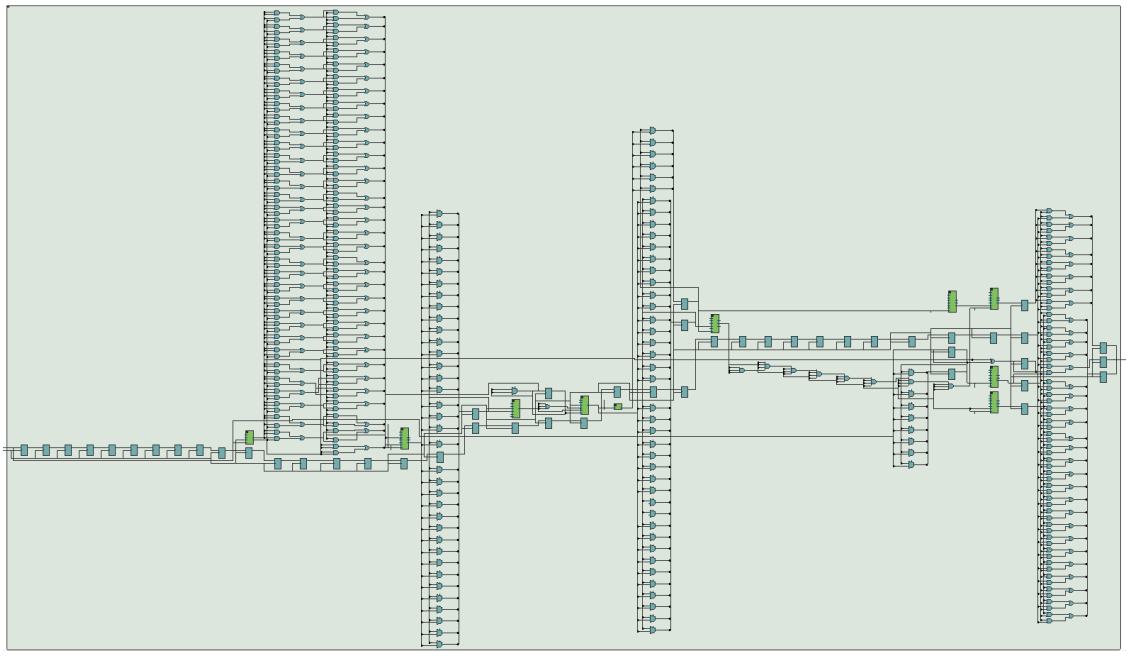


Рис. 3.8 Реалізація алгоритму CORDIC за допомогою вбудованої мегафункці

При здійсненні обчислень тригонометричних функцій за допомогою власної реалізації класичного алгоритму CORDIC доступна можливість оптимізувати його структуру та характеристики, виходячи зі специфіки поставленої задачі. Розрядність та кількість ітерацій алгоритму можна змінювати з кроком в один

розряд, забезпечуючи необхідну точність обчислень. Проте задля уніфікації результатів наводяться лише значення, кратні восьми бітам.

*Таблиця 3.6*

Реалізація класичного CORDICa високої розрядності

| Шина біт | Такти | Пропуск. здатність Мбіт/с | Латент-ність нс | Блоків | Логічних елементів | триг ерів | пам’ять біт | Частота при 85°C | Частота при 0°C | Макси-мальна частота |
|---|---|---|---|---|---|---|---|---|---|---|
| 32 | 24 | 7031 | 109,22 | 3306 | 3142 | 2132 | 0 | 219,73 | 248,39 | 376,8 |
| 32 | 24 | 7030 | 109,25 | 2285 | 2175 | 1491 | 558 | 219,68 | 246,06 | 369,9 |
| 32 | 32 | 6963 | 147,07 | 4326 | 4173 | 2900 | 0 | 217,58 | 246,79 | 378,2 |
| 32 | 32 | 6790 | 150,82 | 2889 | 2775 | 2011 | 608 | 212,18 | 237,08 | 364,3 |
| 40 | 32 | 7946 | 161,09 | 3889 | 3749 | 2525 | 1014 | 198,65 | 224,01 | 337,9 |
| 48 | 32 | 9006 | 170,56 | 4732 | 4664 | 2919 | 1316 | 187,62 | 212,99 | 314,0 |
| 48 | 48 | 8624 | 267,17 | 6577 | 6505 | 4471 | 2068 | 179,66 | 203,96 | 306,8 |
| 64 | 48 | 9912 | 309,94 | 9428 | 9360 | 5928 | 3474 | 154,87 | 174,89 | 268,0 |
| 64 | 64 | 9568 | 428,09 | 12158 | 12086 | 8245 | 3904 | 149,5 | 169,49 | 252,3 |

При порівнянні результатів слід врахувати, що вбудовані мегафункції розраховані, в першу чергу, на універсальність як з точки зору незалежності від платформи, так і даних, що представлені у форматі з рухомою комою. Відповідно, з усіх основних параметрів (частота, пропускна здатність, латентність) власна реалізація алгоритму програє лише по латентності при розрядності, більшій за 48 біт.

Саме висока латентність є слабким місцем класичного методу CORDIC. Кількість тактів, необхідних для обчислення функцій, переважно дорівнює кількості розрядів вхідного аргументу. Як результат, вихідні значення алгоритму отримуються із значною затримкою, а також з використанням великої кількості ресурсів кристалу.

При досягненні під час експериментів значення розрядності класичного CORDICa рівного 64, який займає 4/5 ресурсів кристалу, результати при подальшому збільшенні розрядності перестали бути об’єктивними. Оскільки згенерована компілятором схема розглядається на етапі імплементації як один великий взаємозв’язаний блок, то оптимальна форма для реалізації нагадує собою

круг з центром посередині ПЛІС (рис. 3.9). У випадку досягнення логічними елементами меж самої ПЛІС, компілятор змушений розміщати їх у неоптимальних для алгоритму місцях — часто такими виступають частини кристалу біля кутів. Через це суттєво погіршується характеристики алгоритму, хоча обмеженням в цьому випадку виступає будова самої ПЛІС.

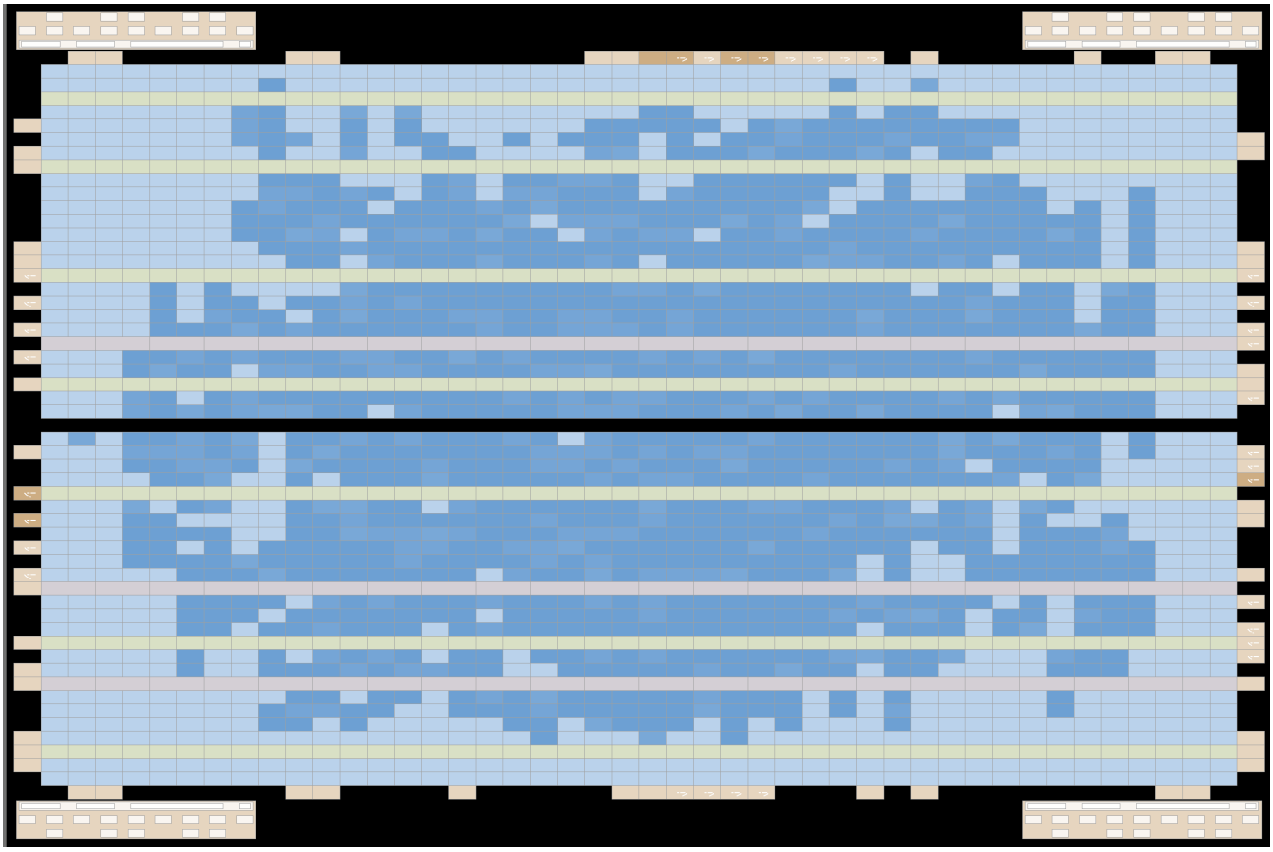


Рис. 3.9 Приклад розміщення логічних елементів ПЛІС при наближенні розміру проекту до неефективних для даної ПЛІС величин.

## 3.5 Заміна класичного підходу пам’яттю та помножувачами

При реалізації класичного методу у кристалі ПЛІС використовувались лише логічні елементи, тоді як для інших методів необхідним буде використання додаткових блоків пам’яті та помножувачів. Опис та характеристики цих блоків були розглянуті на початку глави. Спробуємо перевірити і зіставити класичну реалізацію алгоритму, основану виключно на логічних елементах, з обчисленням тих самих функцій за допомогою пам’яті та помножувачів. Для цього

використаємо табличний метод та залишкове множення. Уникнути ж використання логічних елементів в цілому не видається можливим, адже сигнали між блоками та операції додавання-віднімання після залишкового множення все одно будуть реалізовуватись на їх основі.

Табличний метод застосовується для старших розрядів операнду, кількість яких визначається доступною кількістю пам’яті. Розмір таблиці збільшується більш, як удвічі, з кожною наступною ітерацією. Для вибраного кристалу максимальна кількість ітерацій, які можна помістити в пам’ять, дорівнює дванадцяти, що дозволяє виконувати одну ітерацію зчитування з пам’яті замість дванадцяти ітерацій алгоритму CORDIC (таблиця 3.7).

У свою чергу, метод залишкового множення дозволяє замінити половину ітерацій методу CORDIC однією операцією множення та додавання для кожної з функцій синуса та косинуса. Даний метод застосується лише для молодшої половини розрядів вхідного кута, і при наявності в кристалі блоків з функцією множення стає можливим вдвічі зменшити кількість необхідних ітерацій, замінивши їх двома операціями множення та додавання. Швидкодія в цьому випадку буде обмежена частотою функціонування блоку помножувача та його розрядністю, яка в представленому випадку буде становити близько трьохсот мегагерц.

*Таблиця 3.7*

Табличний метод та залишкове множення (мінімальна латентність)

| Bus bits | Clock | Bandwith Mbit/s | Latency ns | Blocks | Logic | Flip-flops | Multipliers | Mem bits | Freq 85°C | Freq 0°C | Freq Max |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 4 | 2029 | 27,60 | 46 | 24 | 46 | 2 | 6144 | 144,91 | 160,98 | [269,8] |
| 14 | 56 x 4 | 110544 | 28,37 | 5317 | 3987 | 3620 | 112 | 334848 | 141,0 | 156,23 | [262,3] |
| 14 | 5 | 3126 | 22,39 | 62 | 26 | 68 | 2 | 6144 | 223,31 | 248,45 | [388,5] |
| 14 | 56 x 5 | 167164 | 23,45 | 5437 | 3735 | 3968 | 112 | 337920 | 213,22 | 236,46 | [370,0] |
| 24 | 4 | 3144 | 30,53 | 114 | 46 | 114 | 4 | 360448 | 131,01 | 146,13 | [241,0] |
| 24 | 5 | 4369 | 27,46 | 125 | 46 | 125 | 4 | 360448 | 182,05 | 203,62 | [314,4] |
| 24 | 6 | 6932 | 20,77 | 180 | 46 | 180 | 4 | 360448 | 288,85 | 327,12 | [493,1] |

У таблиці наведено результати імплементації для двох варіантів розрядності, які є критичними для вибраного кристалу. При розрядності 14 біт існує можливість задання усіх помножувачів кристалу в конфігурації 9x9 біт, чим досягнуто високі показники пропускної здатності та мінімальну латентність. Всього таким чином можливо створити 56 незалежних конвеєрів. Подальше збільшення розрядності призведе до зменшення кількості конвеєрів та використовуваних блоків помножувачів і при значенні 24 досягне свого максимуму з використанням лише одного помножувача з конфігурацією 18x18. В цьому випадку критичним значенням виступає обсяг вбудованої пам’яті, для доступу до якої на максимальній швидкості доводиться збільшувати кількість тактів.

При невеликих розрядностях аргументів для отримання результату обчислюваної функції достатнім є використання лише двох вищенаведених способів. Для кристалу CycloneIII верхня межа становить 24 розряди, з яких старші 12 розрядів обчислюються табличним методом, а молодші 12 — залишковим множенням. Теоретично, в цьому випадку максимальна частота дорівнює частоті блоку помножувача, а латентність становить чотири такти, два з яких припадає на зчитування адреси та вибірку з пам’яті, а наступні два — на множення та додавання. При практичній реалізації такого підходу відразу отримуємо ряд “підводних каменів”:

по-перше, блоки пам’яті при великій їх кількості розміщені в різних кінцях кристалу (рис. 3.10), і затримка сигналу між ними та будь-яким блоком помножувача буде суттєвою навіть на невеликих кристалах. У розглянутому випадку максимальна частота зменшилась більш, як удвічі, хоча кількість пам’яті в кристалі становить менше одного відсотка від кількості пам’яті в топових кристалах ПЛІС;

по-друге, у випадку використання одного чи декількох блоків пам’яті максимальна тактова частота може бути досягнута лише при потраплянні зчитаних даних з пам’яті безпосередньо у регістри помножувача, тоді як у всіх

сімействах ПЛІС блоки пам’яті та помножувачі розміщені на певній відстані один від одного, що не дозволяє здійснити множення відразу після операції зчитування з пам’яті;

по-третє, для підвищення тактової частоти необхідно розбити шлях від елементів пам’яті до блоків помножувачів на декілька розділених тригерами частин, при проходженні сигналу між якими відсутні будь-які логічні чи арифметичні операції. Тому в результаті отримаємо модель, у якій при зростанні латентності відсутні покращення у характеристиках алгоритму.

## 3.6 Метод знакового перекодування кута

Замість “холостих” ітерацій попереднього способу, з тим самим успіхом можна здійснювати ітерації методу CORDIC між блоками пам’яті та помножувачами, не змінюючи латентність та тактову частоту алгоритму. При цьому кожна ітерація CORDIC дозволить отримати на виході функції два додаткові бінарні розряди, враховуючи розряди, отримані за допомогою методу залишкового множення. При цьому недоліком є лише збільшення кількості необхідних логічних елементів для реалізації ітерацій алгоритму. Також не слід забувати, що при збільшенні розрядності між табличним методом та кусково-лінійною апроксимацією, необхідно буде проводити ітерації класичного CORDIC методу, з кожною ітерацією якого кількість логічних елементів буде зростати у арифметично-геометричній прогресії [119].

Зі всього спектру описаних елементарних функцій підвищений інтерес являють собою функції синуса та косинуса. При здійсненні обчислень за допомогою апаратних засобів отриманий згенерований гармонійний цифровий сигнал сигнал можна подавати на ЦАП, який перетворює отримані дані безпосередньо в коливання на заданій генератором частоті. При невеликих розрядностях ЦАПу здійснювати якісь додаткові перетворення вхідних кутів алгоритмічними методами не має змісту. Для розрядностей 4 - 8 біт можна

використовувати лише табличний метод. Перевагами в цьому випадку будуть простота реалізації, висока швидкодія та невелика ресурсомісткість. При подальшому збільшенні розрядності обсяг пам’яті буде зростати більш, як у два рази з кожним додатковим розрядом. Таким чином, вже при 12 розрядах, обсяг необхідної пам’яті досягне $10^5$ біт, що є великим для вбудовуваних систем та ПЛІС, і вже не можна буде обійтись без використання спеціалізованих блоків пам’яті.

При подальшому збільшенні обсягу для розрядностей в 32 біти обсяг пам’яті буде становити $2.75 \cdot 10^{11}$. Якщо розрахувати співвідношення швидкодії до кількості витрачених для реалізації функції ресурсів, то вже на межі 10 - 12 розрядів стає зрозумілим виграш саме алгоритмічних методів у порівнянні з табличними. Так, використавши метод залишкового множення у комбінації з табличним методом, можна значно збільшити розрядність даних, а кількість витрачених для цього ресурсів буде нижчою за ресурсоємність таблиці, яка б знадобилась для такої розрядності.

Тому для економії логічних елементів можна застосовувати запропонований метод перекодування кута, описаний в розділі 2.2, в якому частина логіки класичного методу CORDIC, що відповідає за визначення напряму повороту, не використовується. Такий спосіб дозволяє усунути конвеєр з каскаду суматорів та мультиплексорів класичного методу CORDIC. Крім цього, таблиця арктангенсів, яка часто реалізується компілятором у вигляді монтажних з’єднань логічних елементів, стає не потрібною.

*Таблиця 3.8*

Результати знакового методу перекодування кута

| Bus bits | Mem/Rot /Mult bits | Clock | Bandwith Mbit/s | Latency ns | Blocks | Logic | Flip-flops | Mem bits | Freq. 85°C | Freq. 0°C | Freq. Max |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 16 | 6/2/8 | 6 | 3868,5 | 28,95 | 228 | 138 | 217 | 832 | 241,78 | 266,31 | [435,73] |
| 16 | 56 x 6/2/8 | 56 x 6 | 208132,8 | 34,44 | 14952 | 7728 | 14560 | 46592 | 232,29 | 260,28 | [411,35] |
| 38 | 13/6/18 | 10 | 7069,9 | 53,75 | 1311 | 1073 | 793 | 488984 | 186,05 | 207,3 | [321,85] |

Запропонований метод перекодування кута реалізований у двох варіантах: 16 та 38 розрядному. В першому випадку задіюються всі помножувачі, при цьому використання блоків пам’яті становить менше, ніж 10% ресурсів кристалу. Варто також відзначити, що між операціями зчитування з пам’яті та множенням здійснюється метод перекодування кута, що дозволяє використати холості такти для арифметичних ітерацій алгоритму, користь яких найкраще проявляється при великих обсягах пам’яті. У випадку 16 розрядного CORDICa з 56 конвеєрами критичним параметром виступає кількість логічних елементів кристалу, кількість яких становить 15,4 тис. Подальше збільшення розрядності до 38 біт обмежується розрядністю блоку помножувача. Незважаючи на повну завантаженість блоків пам’яті, вони в даному випадку не є стримуючим фактором, а їхня кількість може бути зменшена за рахунок збільшення латентності.

При практичній реалізації запропонованого методу перекодування кута через спрощення основного алгоритму та зменшення необхідної кількості логічних елементів для його реалізації в процесі фітінгу вдається краще розмістити комбінаційні функції всередині кристалу, що дасть змогу збільшити тактову частоту алгоритму. Такий ефект також спостерігається при збільшенні розрядності класичного CORDICa, коли із збільшенням кількості логічних елементів відбувається падіння тактової частоти, хоча в теорії повинна змінюватись лише латентність методу. Таким чином, використовуючи перекодування кута, можна досягнути не лише економії логічних елементів, але й підвищення швидкодії та пропускної здатності.

Якщо в кристалі відсутні або зайняті блоки пам’яті чи помножувачі, вони можуть бути реалізовані за допомогою функцій комбінаційної логіки кристалу. Компілятор Quartus в даному випадку здатний самостійно згенерувати таку комбінаційну схему, хоча вона не завжди буде оптимальною. Також відбудеться падіння швидкодії, яку можна компенсувати за рахунок збільшення латентності.

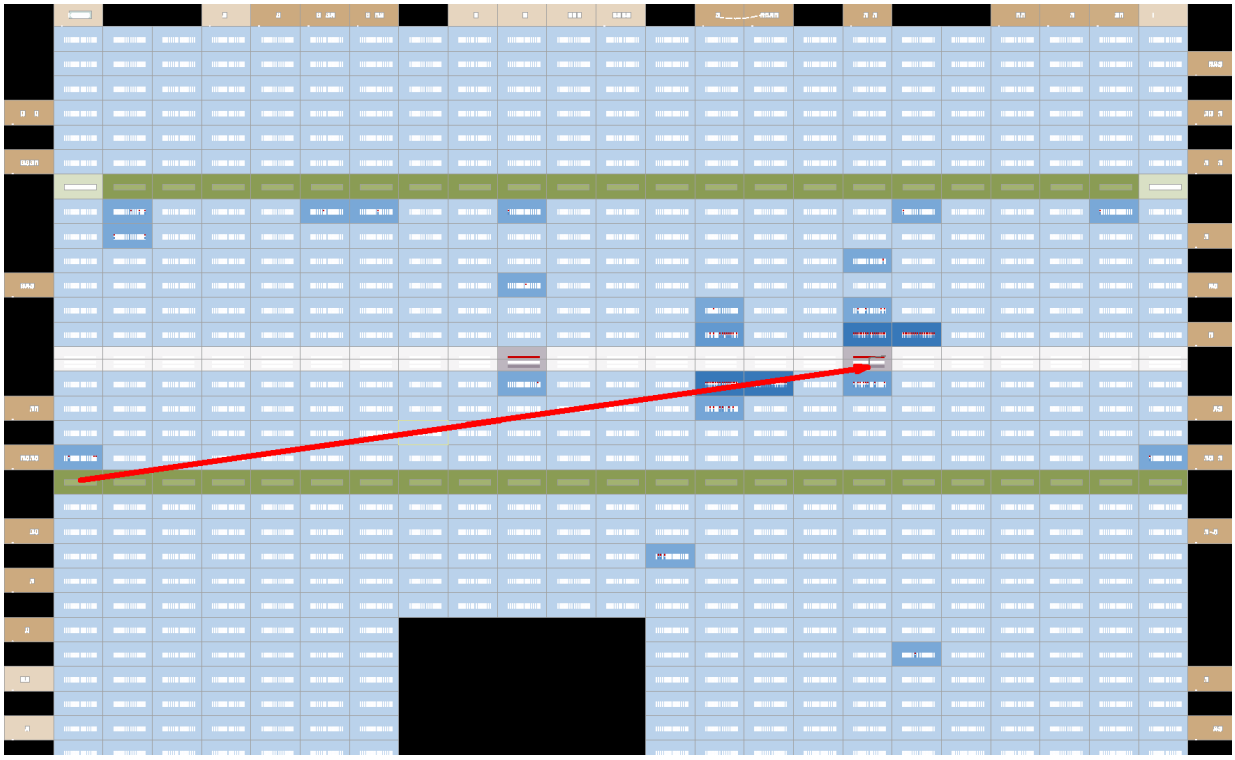


Рис. 3.10 Найдовший шлях сигналу між блоками пам’яті та помножувачем

Падіння частоти для апаратного та комбінаційного блоку помножувача залежно від розрядності, а також кількість необхідних елементів для їх реалізації відображено в таблиці 3.9. Саме частота функціонування блоку помножувача стає основним бар’єром на шляху підняття тактової частоти алгоритму в цілому, оскільки як блоки пам’яті, так і логічні елементи кристалу функціонують на вищій частоті, ніж помножувачі.

*Таблиця 3.9*

Швидкодія програмних та апаратних помножувачів

| Multipliers width | Multiply Blocks | Freq. 85°C | Freq. 0°C | Freq. Max |
|---|---|---|---|---|
| Hard Multipliers 9x9 | 1 Block | 353,11 | 343,29 | 559,6 |
| Hard Multipliers 18x18 | 2 Blocks | 327,23 | 288,85 | 510,7 |
| Soft Multipliers 9x9 | 115 LE | 190,26 | 170,10 | 301,1 |
| Soft Multipliers 18x18 | 424 LE | 123,56 | 110,31 | 192,8 |

У свою чергу, блоки пам’яті, так само як і помножувачі, можуть бути реалізовані за допомогою логічних елементів чи безпосередньо монтажних з’єднань, якщо дані є статичними. Структура таких блоків залежить від

розрядності та типу логічних елементів, використовуваних в кристалі, які можуть мати різну архітектуру, в тому числі в межах однієї серії. Як правило, компілятор самостійно здатний згенерувати елементи пам’яті, враховуючи архітектуру, наявні вільні ресурси та налаштування оптимізації.

Споживана потужність варіюється у вузьких межах та здебільшого залежить від кількості використовуваних виводів кристалу. Існує взаємозв’язок між кількістю задіяних логічних елементів та енергоспоживанням ядра алгоритму, але вона дуже незначна, і, переважно, не змінюється при різних конфігураціях проекту. Загалом статична споживана потужність ядра без урахування енергоспоживання ніжок ПЛІС для всіх апробованих алгоритмів лежить у межах 51,75 — 52,25 mw та через низьку інформативність не подана в таблиці. Також на споживану потужність впливає температура середовища, швидкість руху повітря, тип активного або пасивного охолодження та його параметри. Крім того, проекти не були оптимізовані для зменшення енергоспоживання і наведені вище параметри не мінімізовані.

Оптимізація всіх представлених алгоритмів проводилась задля досягнення максимальної швидкодії, за рахунок збільшення кількості зайнятих ресурсів та потужності. Щоправда, іноді через зменшення загальної площі проекту вдавалось покращити швидкодію, хоча компілятор не помічав такого варіанту компонування. Також у деяких випадках можна зменшити кількість логічних елементів за рахунок блоків пам’яті, що практично не впливало на швидкодію алгоритму. В таких випадках результати наводились для обох варіантів.

Розрахунок пропускної здатності та латентності здійснювався на основі найгірших експлуатаційних умов, при яких робота пристрою є гарантована виробником. Для пропускної здатності наведене мінімально можливе значення, а для латентності — максимально можливе. На практиці дані показники в 1,5-2 рази кращі — залежно від умов експлуатації.

Значна частина показників покращена за рахунок ширшого використання пам’яті, об’єм якої становить 489Кбіт. Дане значення може змінюватись залежно

від платформи і впливати на латентність. Враховуючи низьку ресурсоємність алгоритму, в представленому кристалі можна реалізувати до 56 паралельних конвеєрів алгоритму на основі методу перекодування кута, кількість яких обмежується кількістю помножувачів, а розрядність — кількістю логічних елементів. Внутрішня сумарна пропускна здатність такої конфігурації перевищує 200Гбіт/с, а всі ітерації містять логічне навантаження, що збільшує корисну роботу алгоритму. Тим не менше, алгоритм перекодування кута мав також ряд недоліків. Основними складностями в процесі реалізації алгоритмів даного методу були:

- реалізація арифметичних і логічних операцій із змінними та масивами різного типу;
- необхідна наявність блоків помножувача з підтримкою знакового представлення аргументів;
- неможливість коректно виконувати дію множення знакових та беззнакових величини засобами мови Verilog [128];
- необхідність зведення всіх представлених величин до одного формату;
- зменшення розрядності та точності обчислень через надлишковість формату.

## 3.7 Метод беззнакового перекодування кута

При проектуванні та налаштуванні алгоритму знакового перекодування кута виникали складнощі у зв’язку з вищенаведеними обмеженнями. Дані недоліки можна усунути, використавши метод беззнакового перекодування, описаний у розділі 2.3. Важливою перевагою пропонованого алгоритму є оперування лише додатними величинами, що досягається поворотом кута лише в одному напрямі після операції зчитування з ТПВ. Даний спосіб дозволяє отримувати виключно значення, більші від нуля, що знаходить своє відображення в тому, що усі змінні в методі трактуються як беззнакові величини, якими вони є за замовчуванням у мові Verilog. Також можна відзначити, хоч і незначне, збільшення швидкодії та точності

при застосуванні методу шляхом економії знакових розрядів після усунення надлишкових біт.

Удосконалений алгоритм беззнакового перекодування кута був реалізований як для платформи Altera CycloneIII, так і для Xilinx Artix-7 мовою програмування SystemVerilog. Характеристики реалізованого алгоритму були досліджені за допомогою програмного забезпечення Vivado від фірми Xillinx. Реалізація мегафункції обчислення синуса-косинуса від даного виробника є більш гнучкою та продуманою в порівнянні із версією від фірми Altera. Детальнішу інформацію про можливості Xilinx IP Core подано у [97].

Першим етапом пропонованого алгоритму є табличний метод, в якому значення результату вибирається із пам’яті, а вхідний аргумент функції відіграє роль адреси. Для оцінки апаратної ресурсоємності такого підходу придатним є кристал ПЛІС, класу CycloneIII, який використовувався для дослідів у даній роботі. Один логічний елемент кристалу здатний зберігати 16 двійкових розрядів, замінюючи собою невелику ТПВ. Для реалізації класичного CORDIC методу необхідно використати від двох до п’яти тисяч логічних елементів. Так, наприклад, при використанні Xillix IP Core 32 - розрядний CORDIC вимагає три тисячі логічних елементів, що відповідає обсягу пам’яті в $3000 \cdot 16 = 50$ Кбіт. При використанні такого обсягу пам’яті як ТПВ можна реалізувати функцію в межах дванадцяти розрядів, причому кожен наступний розряд буде збільшувати розмір таблиці більш, як удвічі, враховуючи збільшення розрядності даних, значення яких необхідно зберегти. Також при збільшенні розмірів таблиці з’являється необхідність у додатковій логіці, часто у вигляді мультиплексорів для вибору даних за заданою адресою, а кількість таких елементів може бути співрозмірна з обсягом самої таблиці.

Основним параметром, що демонструє переваги пропонованого алгоритму, є латентність, яка обернено пропорційна тактовій частоті та залежить від кількості тактів, необхідних для обчислення функції. Кількість тактів, у свою чергу, може змінюватись за рахунок зміни розміру таблиці попередньої вибірки. Чим більший

обсяг пам’яті кристалу, який можна виділити для подальшого використання, тим менша кількість тактів і логічних елементів, необхідних для функціонування алгоритму. Другим параметром, який змінювався при реалізації алгоритму, була розрядність помножувача. CycloneIII містить 56 блоків помножувачів, кожен з яких здатний здійснювати множення розрядністю 18x18 або два множення 9x9. Оптимальна розрядність помножувача становить:

$$MultiplerBusSize = \frac{m}{2} + 1 \quad (3.1)$$

де m — кількість розрядів вхідного аргументу.

Для розрядності алгоритму в 16 біт однозначним рішенням буде розрядність помножувача 9. Для розрядності у 32 біти вона становить 17, але при таблиці попередньої вибірки, що дорівнює трьом вхідним розрядам — розрядність помножувачів слід збільшити до 18. При розрядності у 24 біти питання режиму роботи помножувачів є неоднозначним. Звичайно, з точки зору кількості зайнятих ресурсів рекомендовано використання помножувачів у режимі 18x18, але при таблиці попередньої вибірки, меншій за 5 розрядів, одержимо надлишковість у старших розрядах, в результаті чого рекомендується використовувати помножувачі в режимі 12x12, і параметри частоти в цьому випадку будуть відрізнятись. Тому для розрядності у 24 розряди наведено три таблиці із конфігурацією помножувачів у режимі 12x12, 13x13, та 18x18. При експериментах з розрядністю помножувача було виявлено, що компілятор краще оптимізує проекти, в яких задіяні всі розряди помножувача, що практично обґрунтувало використання помножувачів 9x9 при 16 розрядах, у режимах, коли можна використати також режим 8x8. Те саме стосується помножувачів 18x18 для 32 розрядів, у випадках, коли можна було брати 17x17 чи навіть 16x16. Але для 24-х розрядів ефект виявився неоднозначний, хоча, наприклад, при ТПВ, що дорівнює п’ятьом, даний режим 18x18 продемонстрував кращі результати швидкодії, ніж у режимах помножувача 12x12 чи 13x13. Тому для об’єктивнішого зіставлення результатів у всіх випадках вибирався помножувач 18x18 або 9x9, залежно від розрядності. Щодо таблиць з

розрядністю 12x12 чи 13x13, то вони присутні для демонстрації залежності швидкодії ресурсів та латентності відповідно до зміни розрядності помножувачів.

*Таблиця 3.10*

Метод беззнакового перекодування кута з помножувачами в режимі 18x18

| Bit | 24 | 24 | 24 | 24 | 24 | 24 | 24 | 24 |
|---|---|---|---|---|---|---|---|---|
| Table | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Latency clk | 14 | 13 | 12 | 11 | 11 | 10 | 9 | 8 |
| Total latency ns | 49.78 | 41.65 | 43.01 | 36.88 | 39.34 | 35.17 | 30.53 | 28.61 |
| Blocks used | 1,116(7%) | 1,090(7%) | 1,007(7%) | 957(6%) | 732(5%) | 625(4%) | 529(3%) | 441(3%) |
| Logic elements | 894(6%) | 856(6%) | 764(5%) | 742(5%) | 523(3%) | 428(3%) | 348(2%) | 270(2%) |
| Flip-flops | 881(6%) | 879(6%) | 818(5%) | 748(5%) | 585(4%) | 512(3%) | 446(3%) | 381(2%) |
| Pins | 73(21%) | 73(21%) | 73(21%) | 73(21%) | 73(21%) | 73(21%) | 73(21%) | 73(21%) |
| Memory bits | 0(0%) | 0(0%) | 0(0%) | 0(0%) | 3,072(1%) | 6,144(1%) | 12,288(2%) | 24,576(5%) |
| Multipliers | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) |
| Frequency85° mhz | 252.91 | 276.4 | 248.57 | 262.61 | 246.97 | 250.19 | 262.26 | 247.65 |
| Frequency 0° mhz | 281.21 | 312.11 | 279.02 | 298.24 | 279.64 | 284.33 | 294.81 | 279.64 |
| Frequency max mhz | 434.03 | 471.03 | 437.45 | 446.03 | 415.28 | 428.45 | 446.03 | 415.11 |
| Total power mW | 70.86 | 70.86 | 70.85 | 70.85 | 70.84 | 70.84 | 70.83 | 70.83 |
| Core power mW | 51.78 | 51.78 | 51.77 | 51.78 | 51.77 | 51.76 | 51.76 | 51.75 |
| I/O power mW | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 |

*Таблиця 3.11*

Метод беззнакового перекодування кута з помножувачами в режимі 13x13

| Bit | 24 | 24 | 24 | 24 | 24 | 24 | 24 | 24 |
|---|---|---|---|---|---|---|---|---|
| Table | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Latency clk | 48.85 | 44.60 | 42.64 | 38.40 | 39.39 | 34.85 | 37.17 | 29.50 |
| Total latency ns | 14 | 13 | 12 | 11 | 11 | 10 | 11 | 8 |
| Blocks used | 1,115(7%) | 1,091(7%) | 990(6%) | 962(6%) | 733(5%) | 629(4%) | 532(3%) | 428(3%) |
| Logic elements | 894(6%) | 856(6%) | 764(5%) | 742(5%) | 523(3%) | 428(3%) | 348(2%) | 270(2%) |
| Flip-flops | 881(6%) | 879(6%) | 818(5%) | 748(5%) | 585(4%) | 512(3%) | 446(3%) | 381(2%) |
| Pins | 73(21%) | 73(21%) | 73(21%) | 73(21%) | 73(21%) | 73(21%) | 73(21%) | 73(21%) |

| Memory bits | 0(0%) | 0(0%) | 0(0%) | 0(0%) | 3,072(1%) | 6,144(1%) | 12,288(2%) | 24,576(5%) |
|---|---|---|---|---|---|---|---|---|
| Multipliers | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) |
| Frequency85° mhz | 253.23 | 260.21 | 248.08 | 251.76 | 246.18 | 254.97 | 262.4 | 240.85 |
| Frequency 0° mhz | 286.62 | 291.46 | 281.45 | 286.45 | 279.25 | 286.94 | 295.95 | 271.15 |
| Frequency max mhz | 437.25 | 456.62 | 428.45 | 426.26 | 422.30 | 436.87 | 452.90 | 410.51 |
| Total power mW | 70.86 | 70.86 | 70.85 | 70.85 | 70.84 | 70.84 | 70.83 | 70.83 |
| Core power mW | 51.78 | 51.78 | 51.78 | 51.77 | 51.77 | 51.76 | 51.76 | 51.76 |
| I/O power mW | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 |

*Таблиця 3.12*

Метод беззнакового перекодування кута з помножувачами в режимі 12x12

| Multiplier mode | Table=5 DSP17x17 | Table=5 DSP=12x12 | Table=6 DSP12 | Table=7 DSP12 | Table=8 DSP12 | Table=9 DSP12 |
|---|---|---|---|---|---|---|
| Table | 5 | 5 | 6 | 7 | 8 | 9 |
| Latency clk | 11 | 11 | 11 | 10 | 9 | 8 |
| Total latency ns | 38.48 | 38.30 | 38.18 | 34.28 | 31.68 | 28.53 |
| Blocks used | 958 ( 6 % ) | 958 ( 6 % ) | 735 ( 5 % ) | 627 ( 4 % ) | 526 ( 3 % ) | 432 ( 3 % ) |
| Logic elements | 742 ( 5 % ) | 742 ( 5 % ) | 523 ( 3 % ) | 428 ( 3 % ) | 348 ( 2 % ) | 270 ( 2 % ) |
| Flip-flops | 748 ( 5 % ) | 748 ( 5 % ) | 585 ( 4 % ) | 512 ( 3 % ) | 446 ( 3 % ) | 381 ( 2 % ) |
| Pins | 73 ( 21 % ) | 73 ( 21 % ) | 73 ( 21 % ) | 73 ( 21 % ) | 73 ( 21 % ) | 73 ( 21 % ) |
| Memory bits | 0 ( 0 % ) | 0 ( 0 % ) | 3,072 ( 1 % ) | 6,144 ( 1 % ) | 12,288 ( 2 % ) | 24,576 ( 5 % ) |
| Multipliers | 4 ( 4 % ) | 4 ( 4 % ) | 4 ( 4 % ) | 4 ( 4 % ) | 4 ( 4 % ) | 4 ( 4 % ) |
| Frequency85° mhz | 255.17 | 254.97 | 256.87 | 258.06 | 252.27 | 246.67 |
| Frequency 0° mhz | 285.88 | 287.19 | 288.1 | 291.72 | 284.09 | 280.43 |
| Frequency max mhz | 447.43 | 439.56 | 436.30 | 433.46 | 430.85 | 420.34 |
| Total power mW | 70.85 | 70.85 | 70.84 | 70.84 | 70.84 | 70.83 |
| Core power mW | 51.77 | 51.77 | 51.76 | 51.76 | 51.76 | 51.76 |
| I/O power mW | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 | 19.08 |

При реалізації функції синуса-косинуса інтегрованими засобами Altera CORDIC magafunction не дає можливості вибрати параметри компіляції. Для CycloneIII можливою є лише 32 розрядна конвеєрна функція синуса чи косинуса. Обчислення здійснюються у форматі з рухомою комою, де розмір мантиси

становить 24 розряди, а значення порядку дорівнює восьми розрядам. Пропонований алгоритм був реалізований у середовищі Quartus 13.1.4 з конфігурацією у 16, 24 та 32 розряди із різною розрядністю ТПВ. Для 32 розрядів кількість розрядів таблиці лежить у межах від 3 до 10 (Рис. 3.9).

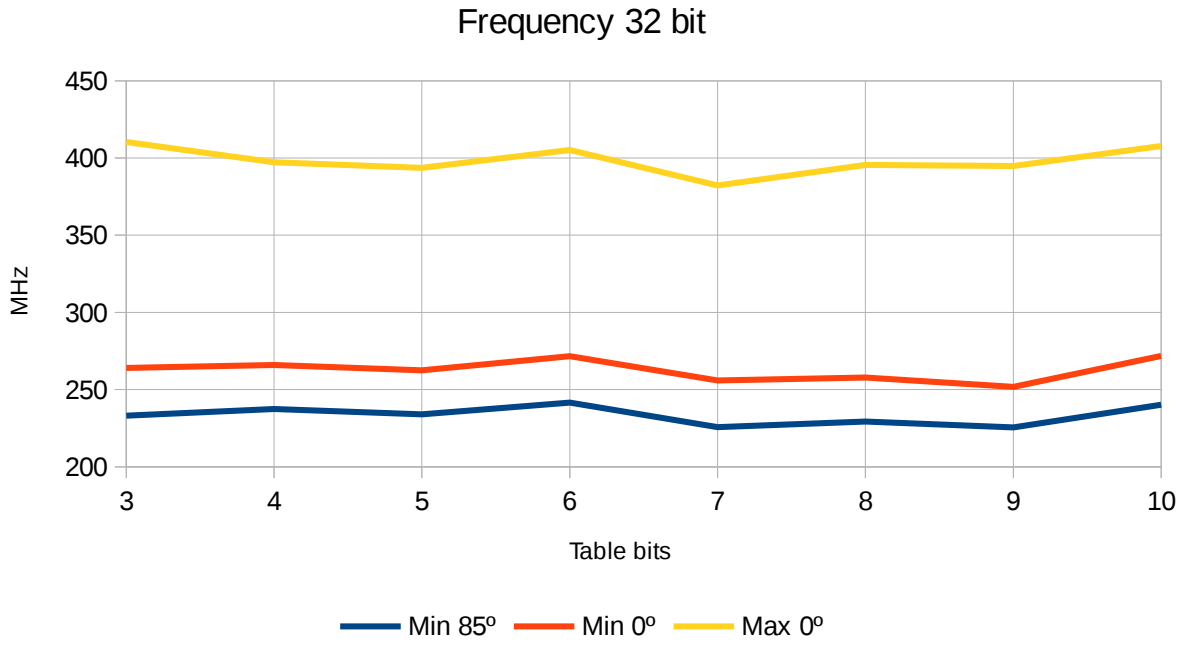



Рис. 3.11 Представлення частот у вигляді графіка для 32-х розрядів, залежно від розміру таблиці.

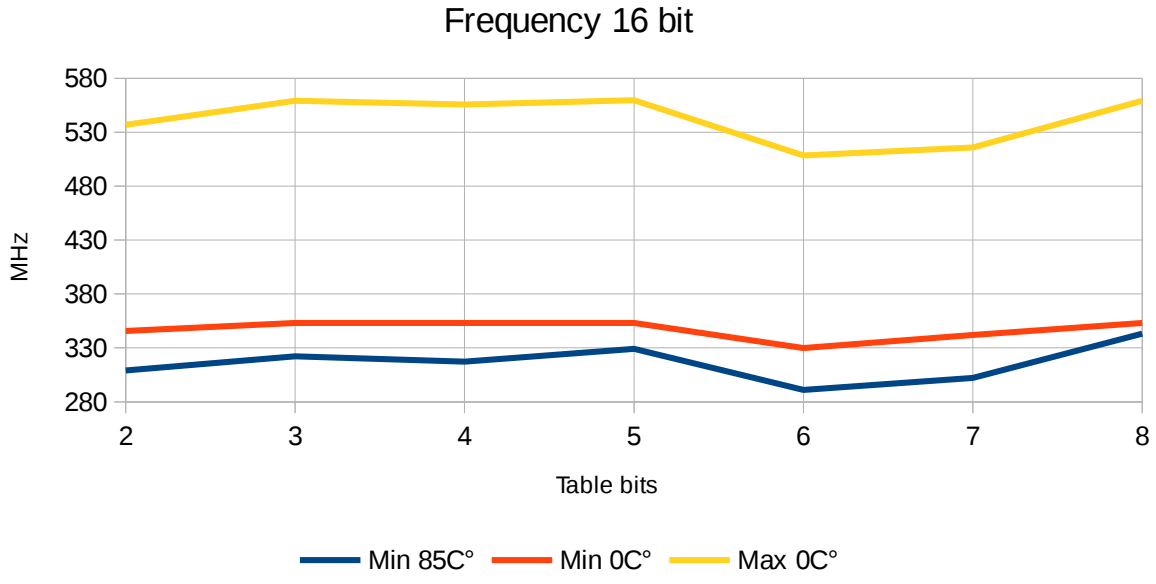



Рис. 3.12 Частота пропонованого алгоритму для 16 розрядів при ТПВ 2-8 біт.

Як видно з таблиці 2, для 32 розрядного пропонованого алгоритму латентність може становити від 40 до 64 нс залежно від розміру таблиці попередньої вибірки. При значеннях таблиці 3 чи 4 розряди внутрішня пам’ять не задіюється, а пам’ять синтезується за допомогою логіки кристалу. При досягненні розміру таблиці в 10 розрядів необхідно мати доступними 64 кбіт пам’яті, що дає виграш як у латентності, так і ресурсоємності алгоритму. Коливання частоти для різних конфігурацій становить менше, ніж ±4%. Для 16 розрядної реалізації алгоритму максимальна частота в більшості випадків обмежується швидкістю помножувача у режимі 9x9, хоча при значеннях таблиці попередньої вибірки 6, 7 з’являється відчутний провал швидкодії, зв’язаний, скоріше за все, з недосконалістю компілятора. Для прикладу, при зміні кристалу з CycloneIII на CycloneIV з аналогічними параметрами даний провал відсутній за умови використання середовища 13.1.4. Якщо ж узяти цей же кристал CycloneIV в новішому середовищі 15.0.1, провал швидкодії тільки посилиться. При чому будь-які спроби оптимізації тільки погіршать становище. Так, найкращих результатів для середовища 15.0.1 було досягнуто при конфігурації компілятора за замовчуванням, при чому дані результати були нижчі від результатів провалу компілятора 13.1.4.

*Таблиця 3.13*

Синтез методу беззнакового перекодування кута в різних версіях середовища

| IDE version | Quartus 15.0.1 | | | | Quartus 13.1.4 | | |
|---|---|---|---|---|---|---|---|
| FPGA model | CycloneIV | | | | CycloneIII | | CycloneIV |
| Optimizations | None (Default) | Speed+Advizer | Speed | Adviser | Optimal | None (Default) | Optimal |
| Table | 8 | 8 | 8 | 8 | 6 | 6 | 6 |
| Blocks used | 126(1%) | 134(1%) | 136(1%) | 134(1%) | 234(2%) | 221(1%) | 236(2%) |
| Logic elements | 39(1%) | 39(1%) | 32(1%) | 39(1%) | 130(1%) | 135(1%) | 130(1%) |
| Flip-flops | 122(1%) | 130(1%) | 136(1%) | 130(1%) | 213(1%) | 217(1%) | 213(1%) |
| Pins | 49(14%) | 49(14%) | 49(14%) | 49(14%) | 49(14%) | 49(14%) | 49(14%) |
| Memory bits | 8,234(2%) | 8,234(2%) | 8,192(2%) | 8,234(2%) | 2,048(1%) | 2,072(1%) | 2,048(1%) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Multipliers | 2(2%) | 2(2%) | 2(2%) | 2(2%) | 2(2%) | 2(2%) | 2(2%) |
| Frequency85° mhz | 258,13 | 234,69 | 223,71 | 233,21 | 291,12 | 301,11 | 316,56 |
| Frequency 0° mhz | 284,82 | 259,20 | 248,94 | 257,80 | 329,71 | 339,67 | 353,11 |
| Frequency max mhz | 488,76 | 449,64 | 388,80 | 446,03 | 508,39 | 524,93 | 546,15 |

Щодо 24 розрядного алгоритму (таблиці 3.9 - 3.11), то максимальну частоту необхідно визначати для кожного випадку окремо, виходячи з розрядності таблиці та розрядності блоку помножувача. При 32 розрядній реалізації пропонованого алгоритму частоти лежать у межах 247.65 МГц (ТПВ = 3) — 276.4 МГц (ТПВ = 9), в обох випадках при конфігурації помножувача 18x18, що становить ±5%.

Класична реалізація CORDIC засобами Vivado без масштабування кута вимагає здійснення m+2 тактів, де m - кількість розрядів вхідного кута. Розрядність алгоритму прямо пропорційна латентності та обернено пропорційна максимальній частоті, на якій здатний коректно працювати алгоритм. Так, для 16 розрядів класичного методу латентність становить 41,4нс, а максимальна частота — 435МГц. При досягненні максимальної розрядності в 48 розрядів, латентність збільшується до 177,5нс, а частота падає до 282МГц.

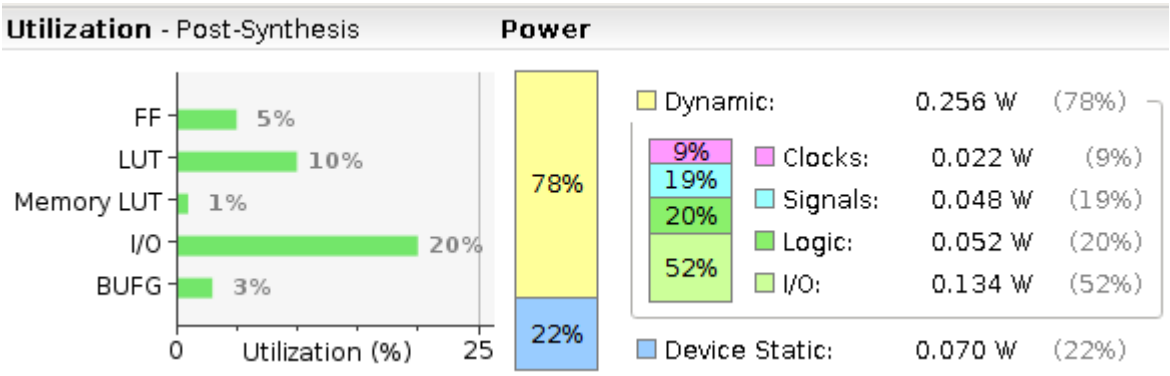



Рис. 3.13 Ресурси використані для реалізації 16-розрядного класичного методу CORDIC за допомогою бібліотеки IP Core Generator

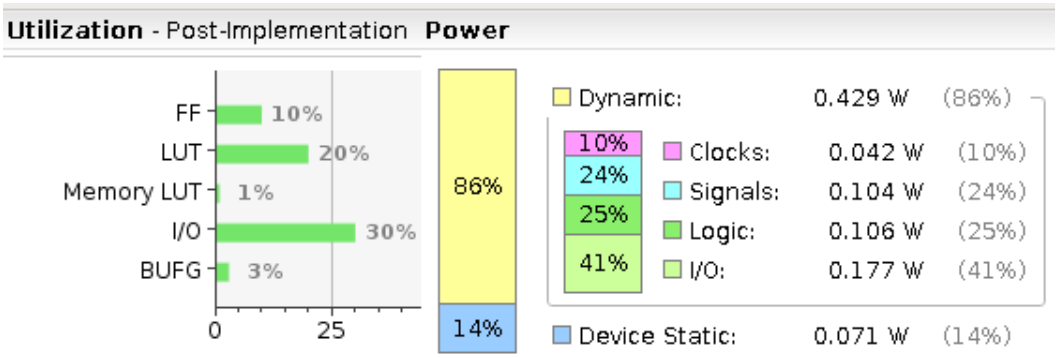
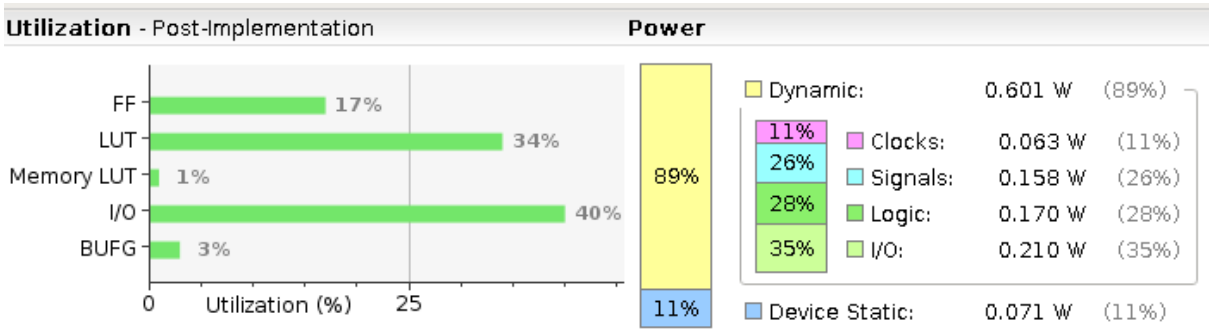


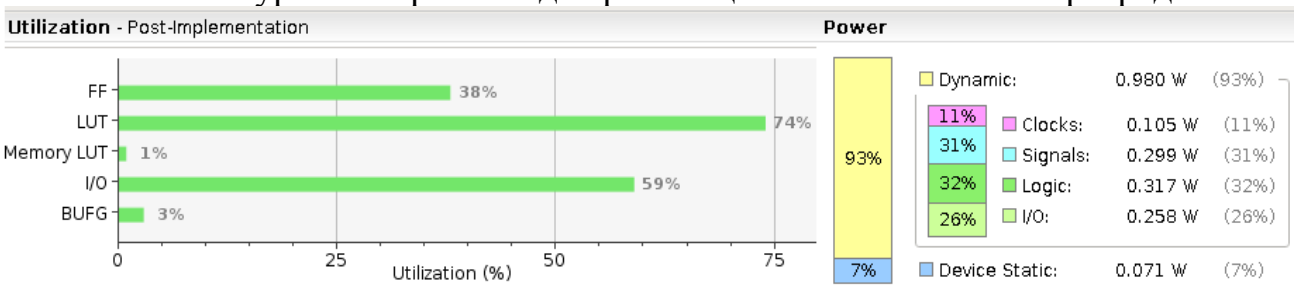

Рис. 3.14 Ресурси використані для реалізації 24-розрядного класичного методу CORDIC за допомогою бібліотеки IP Core Generator

Рис. 3.15 Ресурси використані для реалізації оптимального 32-розрядного




класичного методу CORDIC за допомогою бібліотеки IP Core Generator

Рис. 3.16 Ресурси використані для реалізації максимального 48-розрядного




класичного методу CORDIC за допомогою бібліотеки IP Core Generator

Бібліотека IP Core реалізована таким чином, що кількість вхідних та вихідних розрядів округлюється до величини, кратної вісім. Тому в нашому випадку доцільним буде зіставлення результатів для 24 та 32 розрядних версій алгоритму з пропонованим методом. Латентність даних розрядностей становить 67,6нс та 99,3нс, а частота 385МГц та 342МГц відповідно. При порівнянні класичного алгоритму CORDIC із пропонованим алгоритмом слід зауважити, що метод залишкового множення на платформі Artix-7 так само, як і на платформі

CycloneIII, буде обмежуватись частотою блоку помножувача, причому, враховуючи ефект збільшення частоти при зменшенні розрядності, на невеликих розрядностях частота алгоритму буде значно перевищувати частоту блоку помножувача. Так, максимальна частота помножувача для Artix-7 становить 350±1МГц, за умови використання вбудованих в блок помножувача тригерів. Оскільки структура пропонованого алгоритму не передбачає використання даних тригерів, частота буде меншою від максимальної на декілька відсотків. Так, при розрядності 24 частота пропонованого алгоритму буде становити 349,3МГц, що використовує практично всі ресурси помножувача, а це на 35МГц менше від частоти класичного алгоритму. Проте застосування методу залишкового множення дозволяє замінити 12 класичних ітерацій лише 2-а ітераціями з використанням ресурсів блоку помножувача, хоча і з меншою тактовою частотою.

При збільшенні розрядності до 32 тактова частота дещо падає - до величини 334 - 340МГц, залежно від величини таблиці попередньої вибірки. Максимальна частота і надалі залежить від характеристик помножувача, але через збільшення кількості логічних елементів пошук оптимального способу їх розташування стає важчим для компілятора. Так, при ТПВ у 4 — 6 розрядів, тактова частота залишається стабільною в межах 340МГц. При подальшому збільшенні таблиці до 8 розрядів тактова частота падає до 334,5МГц. Як уже було сказано вище, падіння частоти з надлишком компенсується покращенням параметрів латентності, і при значенні 99,3нс для класичного 32-розрядного методу її можна знизити до 44,1нс для пропонованого методу з ТПВ у 4, чи навіть 32,9нс при ТПВ, що дорівнює 8.

Прямо пропорційно до збільшення ТПВ зростає і обсяг необхідної для реалізації алгоритму пам’яті, але знижується кількість логічних елементів через зменшення кількості ітерацій методу. Так, кількість тригерів та логічних елементів для 24-розрядного пропонованого методу знизилась у 4,5 та 5,5 раз відповідно. Для 32 розрядного методу залежно від значення ТПВ кількість елементів знизилась від 3,5 до 6 разів. Виграшу було досягнуто за допомогою використання двох блоків помножувача та логічних елементів у режимі імітації пам’яті,

оскільки для досягнення максимальної швидкодії компілятор не використовував апаратні блоки пам’яті через її незначний обсяг.

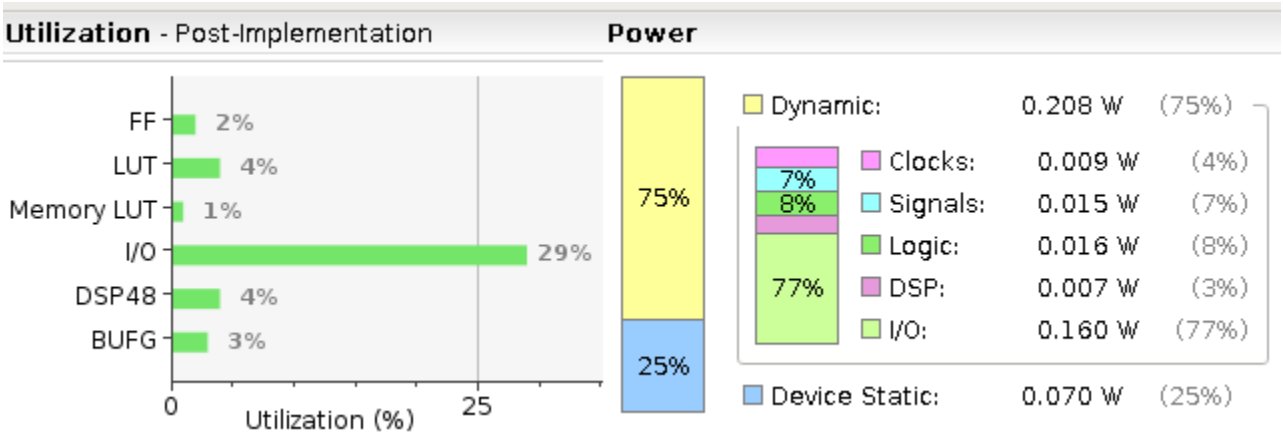



Рис. 3.17 Зайняті ресурси та споживана потужність для методу беззнакового перекодування кута з розрядністю 24 біти та ПТВ, що дорівнює п’ятьом

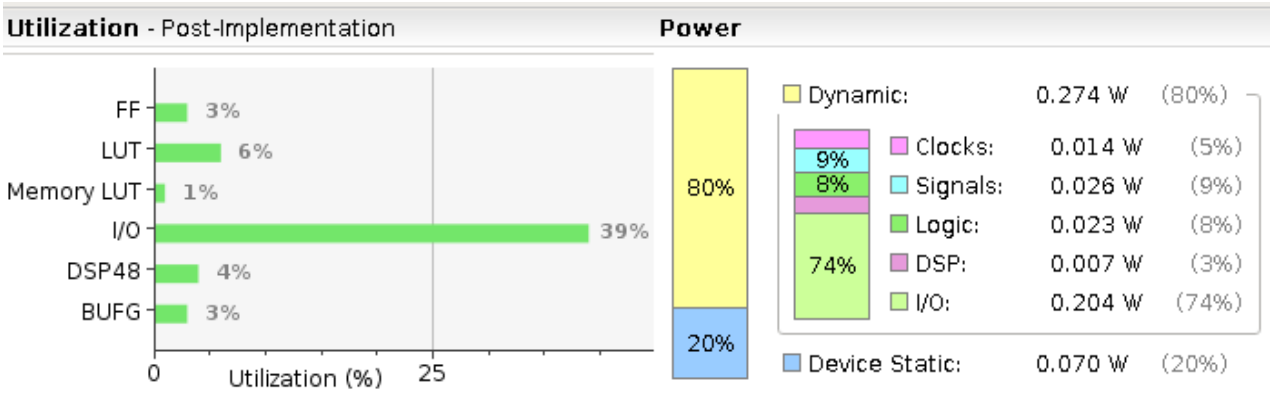



Рис. 3.18 Зайняті ресурси та споживана потужність для методу беззнакового перекодування кута з розрядністю 32 біти та ПТВ, що дорівнює чотирьом

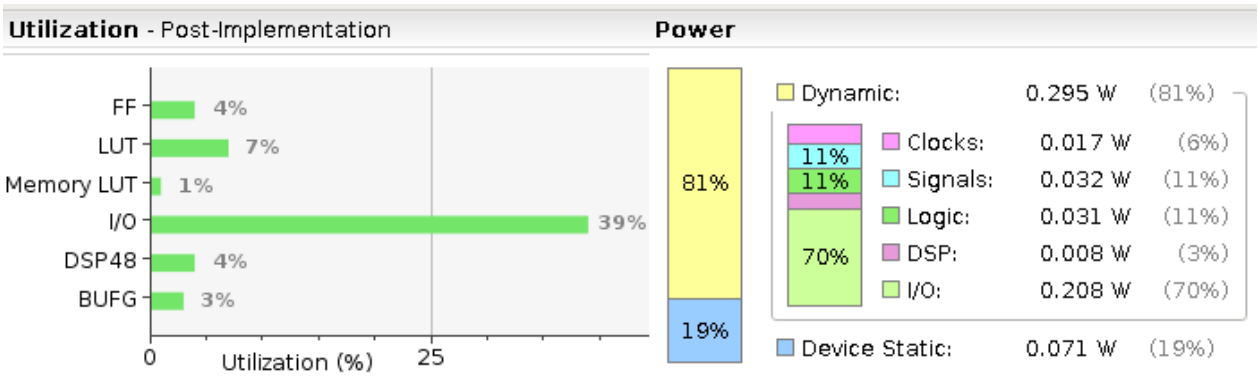



Рис. 3.19 Зайняті ресурси та споживана потужність для методу беззнакового перекодування кута з розрядністю 32 біти та ПТВ, що дорівнює шести

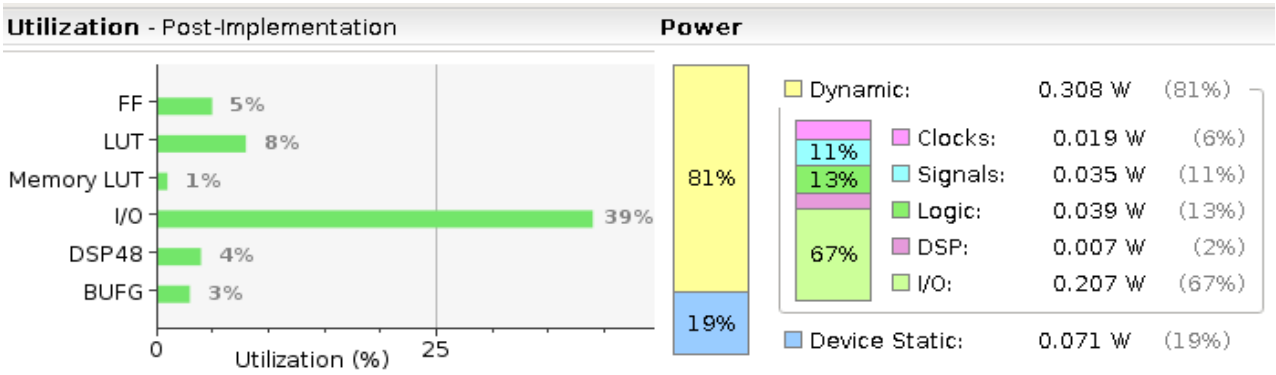



Рис. 3.20 Зайняті ресурси та споживана потужність для методу беззнакового перекодування кута з розрядністю 32 біти та ПТВ, що дорівнює восьми

Споживана потужність при використанні пропонованого методу знизилась майже удвічі у порівнянні з класичним варіантом алгоритму. Основний виграш у зменшенні потужності був отриманий у результаті збільшення частки динамічного споживання кристалу, яке прямо пропорційне кількості задіяних елементів логіки та кількості сигнальних ліній. Енергоспоживання блоків помножувача, які з'явились у пропонованому методі, виявилось незначним і становить лише 2-3% від загальної потужності кристалу.

*Таблиця 3.14*

Характеристики швидкодії та латентності алгоритму перекодування кута і вбудованого CORDIC IP CORE на плафорі Artix-7 середовища Vivado 2015.2

| Bus bit | Table bit | Latency clk | Total Time ns | Freq Best MHz | Freq Overflow MHz | Period Best ns | Period Overflow ns | Slack ps | Slack Owerflow ps | Total Slack ps |
|---|---|---|---|---|---|---|---|---|---|---|
| 32 | 4 | 15 | 44,100 | 340,14 | 340,25 | 2,940 | 2,939 | -16 | 107 | 202 |
| 32 | 6 | 13 | 38,181 | 340,48 | 340,60 | 2,937 | 2,936 | -6 | 39 | 88 |
| 32 | 8 | 11 | 32,879 | 334,56 | 334,67 | 2,989 | 2,988 | -4 | 5 | 5 |
| 24 | 5 | 10 | 28,630 | 349,28 | 349,41 | 2,863 | 2,862 | 26 | 23 | 0 |
| 16 | - | 18 | 41,364 | 435,16 | 435,35 | 2,298 | 2,297 | -21 | 96 | 486 |
| 24 | - | 26 | 67,600 | 384,62 | 384,76 | 2,600 | 2,599 | -36 | 2 | 2 |
| 32 | - | 34 | 99,280 | 342,47 | 342,58 | 2,920 | 2,919 | -9 | 88 | 1031 |
| 48 | - | 50 | 177,550 | 281,61 | 281,69 | 3,551 | 3,550 | -22 | 125 | 393 |

*Таблиця 3.15*

Характеристики ресурсоємності та енергоспоживання алгоритму перекодування кута і вбудованого CORDIC IP на плаформі Artix-7 середовища Vivado 2015.2

| Table bit | Flip-flops | LUT | Mem LUT | I/O | DSP | POWER mW | Temp. C | Thermal Margin C(W) |
|---|---|---|---|---|---|---|---|---|
| 4 | 955 | 827 | 14 | 97 | 2 | 378 | 26,1 | 73,9 (26,1) |
| 6 | 826 | 710 | 16 | 97 | 2 | 364 | 26,0 | 74,0 (26,2) |
| 8 | 698 | 584 | 18 | 97 | 2 | 345 | 26,0 | 74,0 (26,2) |
| 5 | 477 | 381 | 13 | 73 | 2 | 278 | 25,8 | 74,2 (26,2) |
| - | 976 | 947 | 2 | 51 | 0 | 327 | 25,9 | 74,1 (26,2) |
| - | 2161 | 2113 | 4 | 75 | 0 | 499 | 26,4 | 73,6 (26,0) |
| - | 3588 | 3529 | 2 | 99 | 0 | 673 | 26,9 | 73,1 (25,9) |
| - | 7834 | 7746 | 4 | 147 | 0 | 1052 | 28,0 | 72,0 (25,5) |

*Таблиця 3.16*

Характеристики алгоритму 32 - розрядного перекодування кута на плаформі CycloneIII середовища Quartus13.1.4 з різним розміром ТПВ

| Bit | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 |
|---|---|---|---|---|---|---|---|---|
| Table | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Latency clk | 17 | 16 | 15 | 15 | 14 | 13 | 12 | 11 |
| Total latency ns | 64.38 | 60.19 | 57.16 | 55.23 | 54.71 | 50.43 | 47.68 | 40.48 |
| Blocks used | 1,974(13%) | 1,852(12%) | 1,568(10%) | 1,441(9%) | 1,359(9%) | 1,214(8%) | 1,068(7%) | 945(6%) |
| Logic elements | 1,609(10%) | 1,494(10%) | 1,298(8%) | 1,165(8%) | 1,034(7%) | 905(6%) | 778(5%) | 670(4%) |
| Flip-flops | 1,520(10%) | 1,444(9%) | 1,224(8%) | 1,131(7%) | 1,039(7%) | 948(6%) | 851(6%) | 763(5%) |
| Pins | 97(28%) | 97(28%) | 97(28%) | 97(28%) | 97(28%) | 97(28%) | 97(28%) | 97(28%) |
| Memory bits | 0(0%) | 0(0%) | 2,048(1%) | 4,096(1%) | 8,192(2%) | 16,384(3%) | 32,768(6%) | 65,536(13%) |
| Multipliers | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) | 4(4%) |
| Frequency85°mhz | 233.05 | 237.47 | 233.92 | 241.6 | 225.58 | 229.25 | 225.43 | 240.21 |
| Frequency 0°mhz | 264.06 | 265.82 | 262.4 | 271.59 | 255.89 | 257.8 | 251.7 | 271.74 |
| Frequency max | 410.34 | 397.30 | 393.55 | 405.19 | 382.12 | 395.41 | 394.79 | 407.66 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Total power mW | 73.14 | 73.13 | 73.13 | 73.12 | 73.12 | 73.11 | 73.11 | 73.1 |
| Core power mW | 51.81 | 51.81 | 51.8 | 51.79 | 51.79 | 51.78 | 51.78 | 51.78 |
| I/O power mW | 21.33 | 21.33 | 21.33 | 21.33 | 21.33 | 21.33 | 21.33 | 21.33 |

*Таблиця 3.17*

Характеристики алгоритму 16 - розрядного перекодування кута на плаформі CycloneIII середовища Quartus13.1.4 з різним розміром ТПВ

| Bit | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
|---|---|---|---|---|---|---|---|
| Table | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Latency clk | 10 | 9 | 8 | 7 | 7 | 6 | 5 |
| Total latency ns | 28.92 | 25.49 | 22.66 | 19.82 | 21.23 | 17.55 | 14.16 |
| Blocks used | 493 ( 3 % ) | 464 ( 3 % ) | 399 ( 3 % ) | 385 ( 2 % ) | 234 ( 2 % ) | 180 ( 1 % ) | 128 ( 1 % ) |
| Logic elements | 357 ( 2 % ) | 327 ( 2 % ) | 266 ( 2 % ) | 265 ( 2 % ) | 130 ( 1 % ) | 80 ( 1 % ) | 32 ( 1 % ) |
| Flip-flops | 415 ( 3 % ) | 409 ( 3 % ) | 364 ( 2 % ) | 319 ( 2 % ) | 213 ( 1 % ) | 170 ( 1 % ) | 128 ( 1 % ) |
| Pins | 49 ( 14 % ) | 49 ( 14 % ) | 49 ( 14 % ) | 49 ( 14 % ) | 49 ( 14 % ) | 49 ( 14 % ) | 49 ( 14 % ) |
| Memory bits | 0 ( 0 % ) | 0 ( 0 % ) | 0 ( 0 % ) | 0 ( 0 % ) | 2,048 ( 1 % ) | 4,096 ( 1 % ) | 8,192 ( 2 % ) |
| Multipliers | 2 ( 2 % ) | 2 ( 2 % ) | 2 ( 2 % ) | 2 ( 2 % ) | 2 ( 2 % ) | 2 ( 2 % ) | 2 ( 2 % ) |
| Frequency85° mhz | 309.12 | 322.27 | 317.16 | 329.06 | 291.12 | 302.11 | 343.29 |
| Frequency 0° mhz | 345.78 | 353.11 | 353.11 | 353.11 | 329.71 | 341.88 | 353.11 |
| Frequency max mhz | 537.06 | 559.28 | 555.86 | 559.60 | 508.39 | 516.00 | 559.28 |
| Total power mW | 68.58 | 68.58 | 68.58 | 68.58 | 68.57 | 68.57 | 68.57 |
| Core power mW | 51.75 | 51.75 | 51.75 | 51.75 | 51.74 | 51.74 | 51.74 |
| I/O power mW | 16.83 | 16.83 | 16.83 | 16.83 | 16.83 | 16.83 | 16.83 |

## 3.8 Оптимізація алгоритмів та шляхи підвищення швидкодії

Оптимізація алгоритмів можлива лише за допомогою зміни налаштувань синтезу та імплементації, запропонованих виробником. Для синтезу схеми, окрім профілю за замовчуванням, пропонується ще три: оптимізації швидкодії, площі та часу компіляції. Кожен профіль містить набір із 13 параметрів, значення яких подано в таблиці 3.17. При пошуку оптимальної комбінації даних параметрів були

апробовані різні комбінації зазначених параметрів. За результатами дослідження можна сказати, що найкращої швидкодії можна досягти при значенні параметрів оптимізації flatten=rebuild та directive=runtime.

Таблиця попередньої вибірки у середовищі Vivado завжди формується монтажним чином. Відповідно швидкодія і кількість логічних елементів залежить від значень вибраних констант у таблиці. Взявши до уваги, що всі значення таблиці є округленим двійковим представленням ірраціональних чисел, стає зрозумілим, що залежно від налаштувань заокруглення та варіантів їхніх комбінацій можливі конфігурації таблиці можуть відрізнятись залежно від пріоритетів точності чи швидкодії обчислень. Експериментально було встановлено, що зміна способів представлення даних таблиці може відчутно впливати на характеристики вихідного проекту. Таким чином, всі основні параметри алгоритму, в тому числі спосіб розміщення його на кристалі, можуть залежати від значень параметрів ТПВ.

Слід також відзначити складність отримання даних максимальної частоти схеми у середовищі Vivado, яке лише намагається “вмістити” проект у задані часові рамки. Відповідно, доводиться щоразу задавати нову частоту і слідкувати за результатами компіляції, рухаючись спочатку від декількох наносекунд до їх тисячних долей — пікосекунд. Враховуючи, що процес компіляції може тривати до години, а іноді й більше, визначення максимальних частот однієї схеми може тривати протягом тривалого часу.Як видно з вищенаведених таблиць, значення часових затримок знайдені з точністю до однієї пікосекунди. Як виявила практика, десяті долі пікосекунди компілятор заокруглює до найближчого цілого. Прикладом знаходження максимальної частоти схеми може служити таблиця 3.18.

*Таблиця 3.18*

Приклад підбору частоти до 32-розрядного CORDIC IP Core середовища Vivado

| Freq ns | Freq. MHz | WNS ns | WNS MHz | TNS ns | Power W | Junction Temp. | Thermal Margin |
|---|---|---|---|---|---|---|---|
| 1,000 | 1000,0 | 1,968 | 336,9 | -4271.39 ns | 1.823 W | 30,1 ℃ | 69,9 ℃ (24,7 W) |
| 1,500 | 666,7 | 1,479 | 335,7 | -2530.157 ns | 1.237 W | 28,5 ℃ | 71,5 ℃ (25,3 W) |
| 2,000 | 500,0 | 0,979 | 335,7 | -934.824 ns | 0.946 W | 27,7 ℃ | 72,3 ℃ (25,6 W) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2,500 | 400,0 | 0,476 | 336,0 | -88.549 ns | 0.77 W | 27,2 °C | 72,8 °C (25,8 W) |
| 2,700 | 370,4 | 0,174 | 347,9 | -1.698 ns | 0.721 W | 27,0 °C | 73,0 °C (25,8 W) |
| 2,800 | 357,1 | 0,295 | 323,1 | -5.214 ns | 0.703 W | 27,0 °C | 73,0 °C (25,8 W) |
| 2,900 | 344,8 | 0,097 | 333,7 | -0.558 ns | 0.677 W | 26,9 °C | 73,1 °C (25,8 W) |
| 2,905 | 344,2 | 0,120 | 330,6 | -0,886 ns | 0,673 W | 26,9 °C | 73,1 °C (25,8 W) |
| 2,910 | 343,6 | 0,020 | 341,3 | -0.026 ns | 0.683 W | 26,9 °C | 73,1 °C (25,8 W) |
| 2,918 | 342,7 | 0,016 | 340,8 | -0.02 ns | 0.673 W | 26,9 °C | 73,1 °C (25,8 W) |
| 2,919 | 342,6 | 0,088 | 332,6 | -1.031 ns | 0.673 W | 26,9 °C | 73,1 °C (25,8 W) |
| 2,920 | 342,5 | -0,009 | 343,5 | - | 0.67 W | 26,9 °C | 73,1 °C (25,9 W) |
| 2,950 | 339,0 | -0,059 | 345,9 | - | 0.669 W | 26,9 °C | 73,1 °C (25,9 W) |
| 3,000 | 333,3 | -0,081 | 342,6 | - | 0.651 W | 26,8 °C | 73,2 °C (25,9 W) |
| 3,500 | 285,7 | -0,305 | 313,0 | - | 0.577 W | 26,6 °C | 73,4 °C (25,9 W) |

*Таблиця 3.19*

Порівняння результатів імплементації бібліотечних мегафункції,класичного підходу та методу перекодування

| | шина (біт) | к-ть тактів | частота (мгц) | латентність (нс) | кількість триггерів | к-ть логічних елементів |
|---|---|---|---|---|---|---|
| 1 перекод. T6 | 16 | 6 | 435,73 | 28,95 | 138 | 217 |
| 2 перекод. T5 | 24 | 10 | 349,28 | 28,83 | 477 | 381 |
| 3 перекод. T4 | 32 | 4 | 340,14 | 44,1 | 955 | 827 |
| 4 перекод. T8 | 32 | 8 | 344,56 | 32,88 | 698 | 584 |
| 5 перекод. T13 | 38 | 10 | 321,85 | 53,75 | 1073 | 793 |
| 6 cordic ip core | 24 | 26 | 384,62 | 67,6 | 2161 | 2113 |
| 7 cordic ip core | 32 | 34 | 342,47 | 99,28 | 3588 | 3529 |
| 8 класич. повн. | 16 | 18 | 478 | 37,66 | 969 | 1131 |
| 9 класич. звуж. | 16 | 13 | 368 | 35,33 | 625 | 719 |
| 10 класич. повн. | 24 | 28 | 439,9 | 63,65 | 1996 | 2144 |
| 11 класич. звуж. | 24 | 20 | 273,4 | 73,15 | 1401 | 1446 |
| 12 класич. повн. | 32 | 37 | 403,5 | 91,70 | 3495 | 3591 |
| 13 класич. звуж. | 32 | 24 | 256,8 | 93,46 | 2160 | 2234 |
| 14 класич. повн. | 40 | 45 | 375 | 120,00 | 5070 | 5289 |
| 15 класич. звуж. | 40 | 37 | 252 | 146,83 | 3695 | 3768 |
| | | | | | | |
| порівняння 2-6 | 24 | 10-26 | -9,19% | 57,35% | 77,93% | 81,97% |
| порівняння 3-7 | 32 | 4-34 | -0,68% | 55,58% | 73,38% | 76,57% |
| порівняння 4-7 | 32 | 8-34 | 0,61% | 66,88% | 80,55% | 83,45% |
| порівняння 2-10 | 24 | 10-28 | -20,60% | 54,71% | 76,10% | 82,23% |
| порівняння 2-11 | 24 | 10-20 | 27,75% | 60,59% | 65,95% | 73,65% |
| порівняння 3-12 | 32 | 4-37 | -14,61% | 48,09% | 27,32% | 23,03% |
| порівняння 3-13 | 32 | 4-24 | 32,45% | 52,81% | 55,79% | 62,98% |
| порівняння 4-12 | 32 | 8-37 | -14,61% | 64,14% | 80,03% | 83,74% |
| порівняння 4-13 | 32 | 8-24 | 34,17% | 64,82% | 67,69% | 73,86% |
| порівняння 5-14 | 38-40 | 10-45 | -14,17% | 55,21% | 78,84% | 85,01% |
| порівняння 5-15 | 38-40 | 10-37 | 27,72% | 63,39% | 70,96% | 78,95% |

*Таблиця* 3.20

Зіставлення методів перекодування та власної реалізації класичного алгоритму

| | шина (біт) | к-ть такти | латент. (нс) | к-ть блоків | к-ть лог. елемент. | к-ть триггерів | частота 85°C мгц | частота 0°C мгц | частота макс.мгц | пропуск. здат.мб/с |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 перекод. 3T | 32 | 17 | 64,38 | 1974 | 1609 | 1520 | 233,05 | 264,06 | 410,34 | 7457,6 |
| 2 перекод.10T | 32 | 11 | 40,48 | 945 | 670 | 763 | 240,21 | 271,74 | 407,66 | 7686,72 |
| 3 перекод. 13T | 38 | 10 | 53,75 | 1311 | 1073 | 793 | 186,05 | 207,3 | 321,85 | 7069,9 |
| 4 класич.власн. | 32 | 32 | 150,82 | 2889 | 2775 | 2011 | 212,18 | 237,08 | 364,3 | 6790 |
| 5 класич.уточн. | 40 | 32 | 161,09 | 3889 | 3749 | 2525 | 198,65 | 224,01 | 337,9 | 7946 |
| | | | | | | | | | | |
| порівняння 1-4 | 17 - 32 | | 57,31% | 31,67% | 42,02% | 24,42% | 9,84% | 11,38% | 12,64% | 9,83% |
| порівняння 2-4 | 11 - 32 | | 73,16% | 67,29% | 75,86% | 62,06% | 13,21% | 14,62% | 11,90% | 13,21% |
| порівняння 3-5 | 10 - 32 | | 66,63% | 66,29% | 71,38% | 68,59% | -6,34% | -7,46% | -4,75% | 11,03% |

Пропоновані алгоритми зі знаковим та беззнаковим перекодуванням дозволяють спростити реалізацію класичного методу у його програмній та апаратній реалізації, а також отримати виграш у порівнянні із вбудованими бібліотечними функціями, пропонованими розробниками середовищ. У випадку використання платформи Xilinx виграш у латентності пропонованих алгоритмів збільшується більш, як у 2 рази, при втраті тактової частоти всього у декілька мегагерц. Шляхом ширшого використання пам’яті та помножувачів вдається отримати економію як ресурсів кристалу, так і споживаної потужності від 2 до 7 разів. Приблизно таку ж економію ресурсів можна отримати і на платформі від Altera, причому як для класичної реалізації алгоритму CORDIC, так і для вбудованої мегафункції.

## Висновки до третього розділу

1. Вперше реалізовано метод перекодування кута зі спрощеним конвеєром обчислень, що дає змогу реалізації метода CORDIC з меншою кількістю апаратних ресурсів. Виявлено сильні та слабкі сторони методу.
2. Модифіковано метод перекодування кута — беззнакове перекодування, в якому всі аргументи приймають лише додатні величини, чим спрощують

реалізацію алгоритму та необхідні для цього апаратні ресурси зі збереженням переваг підходу перекодування.

3. За допомогою способу представлення розрядів з’являється можливість наперед визначити майбутній хід ітераційного процесу, чим зробити залежності в середині алгоритму менш критичними та більш прогнозованими, що позитивно позначається на швидкодії.
4. Вперше реалізовано метод квадратичної апроксимації, який дає можливість з мінімальною латентністю обчислювати тригонометричні функції та спрощувати ітерації методу CORDIC.
5. Досліджено вплив параметрів компіляції на швидкодію та знайдено оптимальні параметри синтезу та імплементації для ПЛІС фірми Xilinx та Intel. Запропоновано алгоритм одержання ефективних показників швидкодії.
6. Отримані результати щодо енергоспоживання та ресурсоємності пропонованих методів. Практично перевірено коректність їх функціонування та визначено перспективи їх подальшого впровадження.

# РОЗДІЛ 4
# Програмна реалізація алгоритмів на базі x86 комп’ютера та мікроконтролера

## 4.1 Вступні зауваження

У даній частині дисертації буде здійснено вибір процесорної архітектури, яка найбільш повно відповідає потребам, що виникають при дослідженні програмних версій пропонованих алгоритмів, та за допомогою якої можна здійснювати тестування та відлагодження різних модифікацій методів. До таких тестів належать: замір швидкодії, точності, впливів величини пам’яті на швидкодію чи набору інструкцій, які краще використовувати в тому чи іншому випадку. При цьому будуть продемонстровані різні варіанти реалізації алгоритмів, починаючи з класичних версій з використанням відомих методів їх удосконалення, закінчуючи пропонованими розробками у даному напрямі.

Для порівняння різних реалізацій алгоритмів та визначення їх характеристик, насамперед, необхідна підтримка широкого спектру команд та засобів для фіксації, заміру похибок і швидкодії. Для таких цілей найкращим вибором буде сучасна x86 процесорна архітектура, яка має наступні переваги:

- 32 та 64 розрядні регістри
- Висока швидкодія
- Широкий набір команд
- Великий обсяг оперативної пам’яті
- Підтримка операцій із фіксованою та рухомою комою
- Засоби моніторингу продуктивності
- Широкий спектр застосування

Завдяки регістрам високої розрядності з’являється можливість працювати зі змінними як з «одним цілим», що, безумовно, є перевагою, оскільки знімає обмеження для виконання аналогічних операцій над кожним блоком змінних та

засобів з узгодження їх між собою. І, навпаки, при меншій розрядності виникає потреба у виконанні переносів, аналізу результатів та написанні додаткового коду алгоритму, що, в свою чергу, збільшує час виконання основних операцій, нівелює різницю швидкодії між алгоритмами та не дає змогу об’єктивно оцінити ефективність кожного з представлених методів.

Завдяки високій швидкодії архітектури у декілька мільярдів операцій в секунду з’являється можливість в реальному часі протягом декількох хвилин отримати результати виконання кожного блоку алгоритму з перебором усіх можливих комбінацій вхідних даних та вихідних результатів. Те саме стосується заміру швидкодії кінцевої версії алгоритму, оскільки тестування усіх можливих комбінацій в один прохід здійснюється протягом 15 - 30 хвилин. А для визначення точної кількості тактів, які необхідні для роботи алгоритму при довільних значеннях у потоковому режимі, знадобиться від 4 до 15 годин, залежно від частоти та типу процесора.

Оперативна пам’ять використовується для зберігання різного роду даних, наприклад, таблиці арктангенсів кутів у класичному варіантах алгоритму CORDIC, кількість яких є невелика і дорівнює кількості ітерацій чи таблиці початкоих значень для вибраної кількості ітерацій. Тому можливість дослідження взаємозв’язку між швидкодією та розміром таблиці, обсяг якої зростає в геометричній прогресії, доступна практично для будь-яких значень і обмежується лише технічними характеристиками платформи, для якої призначається алгоритм. Враховуючи, що розміри таблиці подвоюються з кожною ітерацію, існує неоднозначна залежність між використанням прийомів прискорення обчислень шляхом збільшення розміру таблиці та розгортанням циклів, яке також дає покращення швидкодії завдяки збільшенню пам’яті. Таким чином, використання x86-тої платформи дає можливість підібрати оптимальний розмір необхідної пам’яті для кожної з можливих реалізацій алгоритму.

Платформа підтримує широкий арсенал команд (понад триста команд загального призначення), який включає в себе різноманітні команди для роботи з

бітами операндів, масштабуванням адрес та мультиплікативних операцій, що дає змогу максимально точно передати дії алгоритму, який проектуємо у вигляді машинних інструкцій, та здійснити порівняння між апаратною та програмною версією алгоритму у аспектах швидкодії, точності та правильності функціонування створеного програмного коду.

Особливістю даної платформи є можливість програмного моніторингу швидкодії, засоби для здійснення якого з’явились ще у моделях мікропроцесорів Pentium Pro та підтримуються усіма моделями сучасних сумісних мікропроцесорів. За допомогою широковживаної інструкції зчитування лічильника тактової частоти можна визначити кількість тактів, здійснених процесором при виконанні довільного блоку програмного коду, який був поміщений між двома сусідніми командами зчитування системного регістру лічильника тактів TSC (Time Stamp Counter).

До складу сучасних мікропроцесорів входить блок операцій із рухомою комою і, хоча ці операції не використовуються у фінальній версії алгоритмів з фіксованою комою, їх наявність є вкрай необхідна під час проектування та відлагодження алгоритмів. Оскільки ці алгоритми трактують цілочисельні значення операндів, як дробове число з фіксованим незмінним значенням порядку, то можливість паралельного обчислення та порівняння значень регістрів з рухомою комою та цілочисельних регістрів дає можливість оцінити точність результату та правильність виконання програмного коду. Крім того, у сучасних моделях процесорів підтримується апаратне обчислення тригонометричних функцій із точністю в 64 бінарні розряди, що дало змогу провести програмне порівняння і аналіз результатів роботи створених алгоритмів, яке недоступне для більшості платформ. Таким чином, стало можливим давати оцінку кожній з кінцевих та альтернативних підверсій алгоритму і відповідного їм програмного коду швидко та точно.

Платформа x86 є однією з найбільш поширених платформ у світі. Завдяки цьому написані алгоритми можуть бути легко вивчені, перевірені чи удосконалені.

Документація по даній платформі є відкритою та загальнодоступною, що дає можливість провести аналіз алгоритмів фахівцям у даній сфері. Широка розповсюдженість платформи дозволяє практично продемонструвати результати роботи створених програм на більшості x86-х сумісних ПК. Щоб не збільшувати обсяг друкованого варіанту дисертації та покращити його читабельність, у лістингах буде наводитись лише критична частина коду програм, яка реалізує той чи інший алгоритм.

### 4.1.1 Вибір оптимального формату представлення даних

Переважна більшість процесорів та мікроконтролерів, для яких призначаються розроблені алгоритми, є цілочисельними. Таку тенденцію можна пояснити кращою швидкодією цілочисельної логіки, її простішою апаратною реалізацією та більшою універсальністю. Числа в такому форматі в операндах пристрою представляються як цілі значення (знакові або беззнакові), які лежать в діапазоні від нуля до $2^n - 1$, де n – розрядність мікропроцесорної архітектури, яка також, як правило, є степенем двійки.

У класичному варіанті методу CORDIC програма повинна визначити значення синуса та косинуса заданого кута φ. Значення кута задається в радіанах та повинно лежати в діапазоні від нуля до π/2. Максимальне значення функцій при даному діапазоні кутів змінюється в інтервалі від нуля до одиниці. Виходячи з цих даних та розмірності операндів, з якими маємо змогу працювати на даній процесорній архітектурі, можна вибрати оптимальний та найменш збитковий формат даних, якими буде оперувати програма.

Отже, отримуємо наступні інтервали для змінних:

$$\begin{gathered} 0 \le \sin\phi < 1 \\ 0 \le \cos\phi < 1 \\ 0 \le \phi < \frac{\phi}{2} \end{gathered} \tag{4.1}$$

Взявши до уваги те, що значення синуса та косинуса - це дробове значення, де ціла частина завжди дорівнює нулю, а мантиса такого числа може бути подана

у двійковій формі наступним чином:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 . | $x_0$ | $x_1$ | $x_2$ | ……. | $x_{60}$ | $x_{61}$ | $x_{62}$ | $x_{63}$ |

Рис. 4.1 Вигляд цілочисельної мантиси для представлення кута

Таке представлення числа забезпечить високу точність та відсутність надлишковості у його представленні.

Якщо розглядати вищеподану схему (рис 4.1) як ціле двійкове число без знаку (N), то для переведення такого числа у представлений вище формат дробового числа (Z) потрібно виконати:

$$Z = N \cdot 2^{-n} \tag{4.2}$$

де n – кількість розрядів мантиси (в нашому випадку n рівне 64). Для зворотнього перетворення обчислюємо:

$$N = Z \cdot 2^{n} \tag{4.3}$$

Виходячи з логіки роботи алгоритму CORDIC, де кожна наступна ітерація обчислення синуса та косинуса описується рекурентним рівнянням, яке визначається поточним значенням косинуса та синуса відповідно:

$$\begin{aligned} \sin_{n+1} &= f(\sin_n, \cos_n) \\ \cos_{n+1} &= f(\sin_n, \cos_n) \end{aligned} \tag{4.4}$$

Через взаємно пропорційну залежність синуса та косинуса значення обох функцій не може одночасно бути меншим за $\sqrt{2} = 0.7071...$ в кожний конкретний момент часу у межах першого квадранту, через що немає необхідності вводити у представлення числа величину порядку, виходячи з наступних міркувань:

- якщо значення синуса (чи косинуса) стане меншим за величину 0.5 (старший біт мантиси рівний нулю), то для обрахунку наступної ітерації необхідно обчислити функцію від косинуса (чи синуса), значення якого в цей момент є більшим від 0.5, що, в свою чергу, призведе до відсікання молодших розрядів меншої величини, оскільки кінцевий результат все одно буде відсікатись за розрядністю більшої мантиси.
- Виділення біт для зберігання величини порядку числа, які використовуються для мантиси в нашому випадку, зменшують точність

обчислень. А, враховуючи сказане у першому пункті, це призведе до втрати значущих біт незалежно від кількості розрядів, відведених під величину порядку числа.

Для зберігання значення кута φ, яке може лежати в межах від нуля до π/2, виділимо один розряд для зберігання цілого значення, а решту – для дробового. Таким чином, число буде мати наступний вигляд:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $x_0$ . | $x_1$ | $x_2$ | $x_3$ | …… | $x_{n-4}$ | $x_{n-3}$ | $x_{n-2}$ | $x_{n-1}$ |

Рис. 4.2 Цілочисельна мантиса для зберігання вхідних значень функцій

Надлишковість в такому випадку буде становити:

$$\frac{2-\frac{\pi}{2}}{2}\cdot 100\% = 21.46\% \qquad (4.5)$$

Для переведення чисел з двійкового цілочисельного формату у формат, показаний на рис 4.2, слід використовувати формулу 4.2. Для зворотного перетворення – 4.3. Значення n при обчисленнях в цьому випадку повинно дорівнювати 63, оскільки старший розряд операнду буде містити цілу частину, на відміну від вихідних значень обчислюваних функцій.

### 4.1.2 Класична реалізація CORDIC методу

Реалізація класичного методу вимагає попереднього обчислення значень арктангенсів таких кутів (1.58), де n дорівнює кількості ітерацій, і в нашому випадку становить 64. Розрядність мантиси результату відповідає кількості ітерацій. Оскільки таблиця арктангенсів відрізняється для різних версій алгоритму, їх значення може бути обчислено з необхідною точністю за допомогою програми, поданої у лістингу 4.1.

Класичний варіант реалізації алгоритму CORDIC є найбільш компактним з усіх запропонованих варіантів, але водночас він має найвищу обчислювальну складність через велику кількість ітерацій. Під час написання програми виникло ряд альтернативних варіантів, від яких залежить точність та швидкість обчислень,

а саме:

- Доцільність здійснення корекції аргументів після кожної ітерації, в тому числі при реалізації програмного множення
- Використання кількох простих (RISC) чи складних (CISC) команд для виконання логічних операцій
- Способи реалізації округлення вхідних значень алгоритму та таблиці арктангенсів:

  - відсікання

  - доповнення

  - до найближчого цілого

Вибрати оптимальний варіант реалізації алгоритму вдалось тільки після здійснення тестування значень відхилення кутів шляхом їх повного перебору і порівняння з більш точними значеннями. Дане тестування буде описано в розділі .

Оскільки кожен з запропонованих варіантів реалізації алгоритму впливає на швидкість його роботи, проводити тестування швидкодії доцільно тільки після вибору оптимального варіанту у відношенні: швидкодія / точність.

Усі варіанти алгоритмів для даної мікропроцесорної архітектури, насамперед, оптимізовувались для досягнення найвищої швидкодії. Якщо існують компромісні варіанти вибору між швидкодією і точністю, то даний факт буде зазначений.

Лістинг 4.1 Фрагмент коду реалізації методу класичного CORDICa на мові асемблера у вигляді вставки в програму на «С»

```
        ….
        mov     ebx,6487ed51h
        mov     eax,9b74eda8h
        mov     edx,eax
        mov     ecx,1fh
agn:    xor     esi,esi
        xor     edi,edi
        shld    esi,eax,cl
        shld    edi,edx,cl
        cmp     ebx,[agl]
        jc      v2
        sub     ebx,[agl+ecx*4]
        add     eax,edi
```

```
        sub     edx,esi
        dec     cl
        jnz     agn
        jmp     fin
v2:     add     ebx,[agl+ecx*4]
        sub     eax,edi
        add     edx,esi
        dec     cl
        jnz     agn
fin:    mov     x,eax
        mov     y,edx
```

### 4.1.3 Застосування методу кусково-лінійної апроксимації

Метод кусково-лінійної апроксимації полягає у прискоренні роботи алгоритму CORDIC шляхом зменшення кількості виконуваних ітерацій. Дана оптимізація стає можливою завдяки збіжності функції обчислення коефіцієнта деформації, коли наступний крок його обчислення виходить за межі розрядної сітки. Оскільки наступні ітерації алгоритму проводяться вже зі сталим коефіцієнтом деформації, то стає можливим застосування операції множення для оптимізації роботи алгоритму. Крім того, слід взяти до уваги те, що для реалізації оптимізованого варіанту метода «залишкового множення» потрібно використовувати машинні інструкції для аналізу біт операнду, яких може не виявитись у простіших моделях мікроконтролерів, що зробить неможливим правильне визначення кращого алгоритму подальших обчислень шляхом аналізу вхідних даних. Щодо методу залишкового множення, то слід відзначити його меншу чутливість в плані швидкодії до даних, які подаються на вхід алгоритму. Результати обчислень усіх варіантів вхідних даних дають менший діапазон розбіжностей у показнику швидкодії алгоритму, що також відіграє значну роль у випадку, якщо критичним показником виступає максимальна затримка, яка виникає при обчисленні деякого випадкового кута.

Суть методу залишкового множення полягає у знаходженні різниці між вхідним та доступним в даний момент значенням кута, яким оперує алгоритм, з наступним добутком цього кута на поточне значення синуса та косинуса. Останнім етапом є знаходження суми та різниці між значенням синуса, косинуса і

знайденим добутком. Операція, яку слід здійснювати в цьому випадку (додавання чи віднімання) визначається знаком кута, який був обчислений при знаходженні різниці кутів.

Даний спосіб «залишкового кута» вимагає здійснення двох операцій множення і був реалізований двома способами:

- з використанням апаратного множення
- з допомогою використання методу Add-Shift

Основною причиною використання апаратного множення стала можливість порівняння цього способу з класичним методом обчислень CORDICa, а також аналіз результатів швидкодії між ним та альтернативним методом Add-Shift. Можливість використання апаратного множення також служило практичним засобом перевірки коректності обчислень щодо другого способу.

Метод Add-Shift розроблявся як альтернативна версія даної частини алгоритму для мікроконтролерів, які не підтримують операцій апаратного множення. Як виявилось в процесі проектування даного методу, після закінчення обчислень, враховуючи особливості реалізації даного способу розрахунків, є можливість досить легко здійснити корекцію вихідного результату щодо молодших біт, які не потрапили у виділену результуючу мантису.

Також було перевірено на практиці значення оптимальної кількості ітерацій класичного методу, після якого слід приступати до методу залишкового множення. З одного боку, чим швидше приступити до даного методу, тим більшою буде швидкодія алгоритму в цілому, але в цьому випадку зменшується точність обчислень, і навпаки. Після тестування алгоритмів із різними кількостями ітерацій вдалось визначити їх оптимальну величину, яка дорівнює тридцяти одному. При меншій кількості ітерацій виникає значний програш у точності результатів, а при їхньому збільшенні - більш полога крива відхилень отриманого результату від реального значення, яка, тим не менше, не перетинає межі максимальної похибки обчислень.

Лістинг 4.2 Уривок алгоритму CORDIC з п’ятнадцятьма ітераціями та апаратним множенням

```
        mov   ebx,6487ed51h
        mov   eax,9b74eda8h
        mov   edx,eax
        mov   ecx,1fh
agn:    xor   esi,esi
        xor   edi,edi
        shld  esi,eax,cl
        shld  edi,edx,cl
        cmp   [agl],ebx
        jnc   v2
        sub   ebx,[agl+ecx*4+4]
        add   eax,edi
        sub   edx,esi
        dec   cl
        cmp   cl,15
        jnz   agn
        jmp   v3
v2:     add   ebx,[agl+ecx*4+4]
        sub   eax,edi
        add   edx,esi
        dec   cl
        cmp   cl,15
        jnz   agn
v3:     mov   esi,edx
        sub   ebx,agl
        jnc   v4
        neg   ebx
        shl   ebx,1
        mul   ebx
        add   edx,esi
        jmp   fin
v4:     shl   ebx,1
        mul   ebx
        sub   esi,edx
        mov   edx,esi
fin:
```

Лістинг 4.3: Фрагменту коду що здійснює множення за допомогою методу Add-Shift після п’ятнадцятьох ітерацій класичного CORDICa.

```
        xor   eax,eax
        xor   edx,edx
        sub   ebx,agl[0]
        jnc   n2
        neg   ebx
        inc   cl
        shr   ebx,1
        jc    m3
m1:
        shr   eax,1
        shr   edx,1
        dec   cl
        jz    m4
m2:
        shr   ebx,1
        jnc   m1
m3:
        add   eax,esi
        rcr   eax,1
        add   edx,edi
        rcr   edx,1
        dec   cl
        jnz   m2
m4:
        shr   eax,14
        adc   eax,0
        shr   edx,14
        adc   edx,0
        add   esi,edx
        sub   edi,eax
        jmp   fin
n1:
        shr   eax,1
        shr   edx,1
        dec   cl
        jz    n3
n2:
        shr   ebx,1
        jnc   n1
        add   eax,esi
        rcr   eax,1
        add   edx,edi
        rcr   edx,1
        dec cl
        jnz n2
n3:
        shr   eax,15
        adc   eax,0
        shr   edx,15
        adc   edx,0
        sub   esi,edx
        add   edi,eax
fin:
```

### 4.1.4 Використання табличних методів для прискорення роботи алгоритмів

Алгоритм CORDIC можна прискорити шляхом створення таблиці з результатами наперед обчисленої деякої кількості ітерацій алгоритму. Тоді на першій стадії виконання програми обчислюється лише індекс розташування кута у таблиці. Даний етап подібний до класичного повороту вектора, але без операцій обчислення синуса та косинуса. Після деякої фіксованої кількості ітерацій отримуємо в таблиці значення індексу, розрядність якого дорівнює кількості пройдених ітерацій. Розмір таблиці можна визначити за формулою:

$$S=\frac{2^n \cdot m}{8} \tag{4.6}$$

де S – розмір таблиці в байтах; n – кількість ітерацій; m – розмір мантиси (в бітах)

Перевагою табличного методу для класичного алгоритму є використання лише однієї таблиці для одержання значень синуса та косинуса. Недоліком можна вважати невисоку універсальність таблиці, оскільки складена таблиця є актуальною лише для певної фіксованої початкової кількості ітерацій із заданою розрядністю. Також така таблиця дійсна лише при певній сталій кількості кроків алгоритму через відмінність коефіцієнтів деформації, які можуть змінюватись при різній кількості ітерацій алгоритму.

Індексний покажчик представляє собою двійкове число розрядністю n біт, розряди якого встановлюються у випадку збільшення поточного обчислюваного алгоритмом кута щодо значення заданого на вході програми, а також скидаються в нуль у разі його зменшення. Слід відзначити, що використання наперед обчисленої таблиці для класичного методу не усуває потребу у зберіганні значень арктангенсів. В результаті початкові ітерації все одно здійснюються, використовуючи значення даної таблиці, зчитування якої повинно виконуватись на кожній ітерації алгоритму обчислення індексного показника, що сповільнює роботу цього метода.

Удосконалений метод CORDICa з одностороннім поворотом вектора має ряд

переваг над класичним способом. За допомогою цього методу з’являється можливість індексувати кути у таблиці, використовуючи старші n бітів вхідного кута, що, в першу чергу, знімає потребу у використанні таблиці арктангенсів для знаходження індексу. По друге, відпадає необхідність у здійсненні циклічних повторень коду для розрахунку індексу: значення індексу можна одержати за допомогою декількох машинних інструкцій. У нашому випадку одержання стартових значень для довільної кількості ітерацій здійснюється за чотири машинні інструкції (по дві на пошук та завантаження). Таблиця також може бути динамічно розширена чи звужена, доповнюючи чи усуваючи вже існуючі значення.

Недоліком цього методу в порівнянні із класичним способом є необхідність використання двох окремих таблиць для значень синуса та косинуса, що вдвічі збільшує кількість необхідної пам’яті. Незважаючи на це, даний недолік компенсується більш швидким одержанням початкових значень кутів, а також відсутністю таблиці арктангенсів, розмір якої є співрозмірний з розміром доданої таблиці при невеликій кількості початкових ітерацій. Крім того, при вдвічі більшій таблиці даний підхід не буде програвати класичному методу і не відіб’ється суттєво на кінцевій версії швидкодії алгоритму в цілому.

Загалом, метод одностороннього повороту дає однозначний виграш у підвищенні швидкодії алгоритму залежно від кількості початкових ітерацій, величина яких визначається технічними характеристиками платформи та кількості вільної пам’яті, яку можна виділити під зберігання вищезгаданих таблиць.

Лістинг 4.4 Вибір початкових значень алгоритму класичним методом

```
        mov   ebx,6487ed51h
        mov   ecx,31
        xor   eax,eax
agn:    cmp   ebx,agl[0]
        jc    vl
        shl   eax,1
        sub   ebx,[ecx*4+4+agl]
        dec   cl
        cmp   cl,23
        jnz   agn
        jmp   rtb
```

```
vl:         rcl     eax,1
            add     ebx,[ecx*4+agl+4]
            dec     cl
            cmp     cl,23
            jnz     agn
rtb:        mov     esi,[tbl+eax*4]
            not     al
            mov     edi,[tbl+eax*4]
```

Лістинг 4.5 Одержання початкових значень покращеною версією алгоритму:

```
            mov     ebx,agl[0]
            shr     ebx,27
            mov     esi,[sct+ebx*8]
            mov     edi,[sct+ebx*8+4]
            ror     ebx,5
```

### 4.1.5 Гібридний (комбінований) метод обчислень

Гібридний метод обчислення являє собою удосконалений метод, в основі якого лежать ідеї класичного алгоритму CORDICa та способи його вдосконалення. Зміна коефіцієнта деформації, варіації якого створюють перешкоди для спрощення обчислень у класичному методі, компенсуються за допомогою коефіцієнтів θ, значення яких є наперед обчислені і кількість яких дорівнює кількості можливих ітерацій методу.

Зростання швидкодії у методі досягається за рахунок усунення частини ітерацій, або, іншими словами, наближення до заданого кута здійснюється тільки з однієї сторони. Таку поведінку алгоритму можна забезпечити, аналізуючи одиничні розряди в двійковому представленні кута. Розряди аргументу, значення яких дорівнює нулю, алгоритм пропускає, що дає приріст, в середньому, близько п’ятдесяти відсотків. При виявленні одиничного розряду збільшується сумарне значення змінної тета θ, чим компенсується відсутність повороту для нульових розрядів. Відмінністю даного методу від класичного залишкового множення є використання сумарного значення коефіцієнтів θ, що забезпечує нормування результату обчислень, які були спотворені через ігнорування коефіцієнта деформації. Саме такий підхід дозволив вилучати з алгоритму зайві ітерації та обчислення. Основним інструментом множення величин для платформ, на яких

зазвичай функціонує алгоритм, виступає метод Add-Shift, що, як буде продемонстровано, дає стабільний виграш у швидкодії. Під час аналізу граничних величин сумарного коефіцієнту θ для тридцяти двох розрядів операнду при усіх можливих комбінаціях вхідних кутів був визначений діапазон значень у межах:

$$-3119 \le \theta \le 65535 \tag{4.7}$$

Для таких величин можна здійснювати 16 ітерацій методу Add-Shift для додатних значень тета, оскільки число можна записати у шіснадцятирозрядному двійковому представленні. А також за 12 ітерацій для від'ємних значень, оскільки число 3119 лежить у межах до $2^{12}$ яке можна представити у вигляді дванадцяти двійкових розрядів.

Для порівняння, граничні значення, які може набувати значення залишкового кута при тридцяти двох розрядних операндах, лежать у межах:

$$-65535 \le \phi_{зал} \le 65536 \tag{4.8}$$

для роботи з яким методу Add-Shift знадобиться 16 та 17 ітерацій для від'ємного та додатнього значення відповідно

Під час проектування гібридного (комбінованого) методу для операндів у тридцять два розряди п'ять ітерацій було виділено під стартову таблицю, що вимагає 25 · 32 ÷ 8 = 128 байт. Оскільки необхідно мати два окремих примірники таблиць для синуса та косинуса, то 128 · 2 = 256 байт пам'яті для зберігання таблиць. Даним методом достатньо виконати лише половину, а саме 16 із 32 ітерацій. Для знаходження кінцевого значення замість виконання наступних шістнадцяти ітерацій використовуємо вище розглянутий модифікований метод множення кута із коефіцієнтом тета. Оскільки перших п'ять ітерацій виконуються на основі таблиці, то є необхідність у здійсненні лише одинадцяти ітерацій алгоритму.

Застосування комбінованого (гібридного) алгоритму проходить у два етапи. Другий етап можна розглядати, як частковий випадок першого етапу, коли розрядність операндів вищих порядків стають меншими за розрядність обчислювальної мантиси, що дозволяє опустити частину ітерацій і прискорити

виконання алгоритму, не вплинувши на точність обчислень. Перший етап гібридного методу буде тривати п’ять ітерацій, другий – шість. Оскільки вхідними даними алгоритму виступає двійкове представлення кута, значення розрядів якого буде поетапно проаналізоване, то можна зообразити етапи виконуваних операцій на прикладі розрядів вхідного кута:

| табличний метод | | | | | перший етап | | | | | другий етап | | | | | | залишкове множення | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a_0$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | $a_9$ | $a_{10}$ | $a_{11}$ | $a_{12}$ | $a_{13}$ | $a_{14}$ | $a_{15}$ | $a_{16}$ | $a_{17}$ | $a_{18}$ | ... | $a_{30}$ | $a_{31}$ |

Рис. 4.3 Схема поділу розрядів вхідого кута між застосовуваними методами

Лістинг 4.6 Фрагмент коду, який здійснює обчислення однієї з ітерацій першого етапу алгоритму:

```
        …
        bt      agl[0],31 - n
        jnc     skp
        add     ebx,teta[n*4]
        mov     eax,esi
        mov     edx,edi
        shr     eax,n++
        shr     edx,n++
        sub     edi,eax
        add     esi,edx
        shr     eax,n++
        shr     edx,n++
        sub     esi,eax
        sub     edi,edx
        shr     eax,n--
        shr     edx,n--
        add     edi,eax
        sub     esi,edx
skp:
```

Лістинг 4.7 Код для другого етапу обчислень має наступний вигляд:

```
        bt      agl[0],31 - n
        jnc     sn
        add     ebx,teta[n*4]
        mov     eax,esi
        mov     edx,edi
        shr     eax,n++
        shr     edx,n++
        sub     edi,eax
        add     esi,edx
        shr     eax,n
        shr     edx,n
        sub     esi,eax
        sub     edi,edx
sn:
```

Представлений лістинг реалізації метода add-shift багато в чому збігається із способом, використовуваним у комбінованому методі, за винятком того, що враховується значення $\theta_{сум}$ для знаходження різниці із кутом $\varphi$, яке і буде використовуватись для обчислення добутку операндів.

## 4.2 Дослідження похибок алгоритмів

Визначення відхилень відіграє ключову роль при оцінюванні алгоритму, адже показники точності та швидкодії є основними показниками, за якими можна судити про ефективність та практичну придатність алгоритму. А оскільки на етапі створення та проектування нових алгоритмів перевірку на швидкодію слід здійснювати вже після відлагодження та вибору одного із запропонованих підверсій алгоритму, то основним показником у цьому плані будуть виступати значення похибок, які були виявлені під час його тестування.

Основним завданням при аналізі деякого блоку алгоритму було швидке та точне оцінювання значень максимальних похибок, які виникають при використанні алгоритму. Враховуємо, що перші версії програм, які були написані на основі класичних алгоритмів, були створені мовою «С» і не могли забезпечити необхідну швидкість, а іноді й точність обчислень, незважаючи на те, що мова «С» застосовує хорошу оптимізацію до генеруючого коду. Перебір усіх можливих вхідних даних за допомогою програм цією мовою тривав протягом кількох годин, незважаючи на значну частину коду, написану за допомогою асемблерних вставок.

На перших етапах тестування відхилень здійснювалось заданням деякого кута та порівнянням отриманого значення в кінці алгоритму із отриманим за допомогою засобів мови «С». Різниця між ними також виводилась, як число з рухомою комою, що не було достатньо інформативним та не дозволяло більш об’єктивно оцінити величини похибок при різних значеннях відхилень. Значною проблемою стало те, що при відлагодженні алгоритмів вибирались такі версії

коду, які забезпечували більш точне значення тільки при проаналізованих вхідних даних. Проте, як виявилось пізніше, після тестування усіх можливих вхідних значень такі способи давали більші похибки у деяких інших комбінаціях вхідних даних, які не потрапили у діапазон тестування. Тому результати, отримані першим способом, були поставлені під сумнів, і їх довелось обчислювати за допомогою більш прогресивних методів. Крім того, тестуючи алгоритми першим способом, було б неможливо оперативно провести дослідження. Тому був спроектований спеціальний блок коду мовою асемблера, який поміщався відразу після досліджуваного алгоритму та проводив перевірку коректності здійснених алгоритмом обрахунків, перебираючи всі можливі варіанти вхідних даних та заповнюючи таблиці відхилень між отриманими та фактичними значеннями. Даний алгоритм написаний у вигляді асемблерної вставки і має наступні переваги:

- 80 - бітна точність результатів обрахунків
- мінімальний розмір коду та його максимальна швидкодія
- використання команд із рухомою комою, які можуть виконуватись одночасно з цілочисельними командами для розпаралелювання обчислень

Враховуючи те, що для заміру швидкодії процесор повинен працювати в однозадачному режимі із відключеними перериваннями, то при розробці алгоритмів одночасно використовувались три комп’ютери: для проектування, для заміру величини похибок та для визначення швидкодії.

Лістинг 4.8 Фрагмент програми, який здійснює перевірку отриманих значень

```
unsigned int delta[32];
unsigned int ts=(sizeof(delta)>>2)--;
….
fild qword ptr agl
fscale
fsincos
fxch st(1)
fxch st(2)
fxch st(1)
fscale
fistp qword ptr z
fxch st(1)
sub esi,z
jnc     l1
neg esi
```

```
l1:
        cmp     esi,ts
        jc      l2
        mov     esi,agl[0]
        mov     t,esi
        mov     esi,ts
l2:
        inc dword ptr [delta+esi*4]
        inc     agl
        cmp     agl,0xc9000000
        jne tst
```

Таблиця відхилень delta[x] являє собою масив подвійних слів, кількість яких можна задати на початку програми і які будуть відображати кількісну величину кутів, похибка яких дорівнює:

$$delta[x]=n\cdot 2^{-32} \tag{4.9}$$

Під час аналізу вже відлагоджених алгоритмів для роботи було достатньо величини у 40 – 50 комірок, хоча велика кількість комірок суттєво допомагала здійснювати аналіз відхилень та їхню тенденцію при великих значеннях похибок під час тестування бета-версій алгоритмів.

При аналізі отриманих таблиць та їх результатів можна виділити два різновиди зміни значень похибок для альтернативних модифікацій алгоритму:

- Збільшення чи зменшення максимальної виявленої похибки алгоритму
- Зміна кривизни параболи відхилень при сталому піковому значенні відхилення

Зміна відхилень першого роду є більш значущою, тому що характеризує точність результатів алгоритму в цілому. Тому при виборі варіанту реалізації алгоритму перевага надавалась алгоритму з мінімальним значенням пікового відхилення.

Відхилення іншого характеру проявлялися досить часто, хоча вони є більш специфічні. Зазвичай, така поведінка спостерігалась при незначних модифікаціях алгоритму, під час перерозподілу результатів чи реорганізації структур, методів чи таблиць, коли не відбувалась модифікація коду, що відповідає за розрахунки. Таким чином, із запропонованих варіантів вибирався той, який забезпечував

більшу швидкодію або меншу кількість обчислень, що в більшості випадків є взаємно пропорційною величиною. Якщо ж швидкодія не збільшувалась чи змінювалась несуттєво в межах одного середньостатистичного такту процесора, то вибирався алгоритм з більшою крутизною параболи відхилень, що давало вищу ймовірність отримувати точніше значення при обчисленнях, оскільки кількість кінцевих значень, обчислених з нижчою похибкою, була більшою.

До зміни величини похибок, зазвичай, призводить:

- Зміна структури алгоритму
- Зміна (корекція) дій алгоритму
- Зміна способів заокруглень чисел
- Корекція результату після арифметичних дій з дробовими числами

Реалізуючи деякий алгоритм, можна дати досить однозначну відповідь про його точність при різних альтернативних версіях із подібною структурою. Це стосується і дій алгоритму, оскільки їх вплив на кінцевий результат є більш явним і чітко вираженим у порівнянні з іншими методами, які впливають на точність остаточного результату. Крім того, структуру та дії алгоритму можна представити у математичному вигляді, чого не зробиш стосовно заокруглень та методів корекції. Тому в першому випадку вибрати правильний варіант реалізації було досить просто, враховуючи також невелику кількість альтернативних варіантів реалізації, тоді як для другого випадку вибір кращого варіанту був обгрунтований тільки після здійснення повного перебору усіх можливих комбінацій та зіставлення кінцевих результатів.

Для класичного варіанту алгоритму CORDICa була поставлена задача визначити оптимальний спосіб округлення дробових значень таблиці арктангенсів для забезпечення максимально точних вихідних результатів алгоритму. Округлення можна здійснювати методом відсікання, доповнення, чи округленням до найближчого цілого. Тому було складено три версії цієї таблиці з різними варіантами округлення. Найкращий результат після перебору усіх варіантів

показав третій спосіб – округлення до найближчого цілого.

Значення коефіцієнту деформації під час тестування методом підстановки показував кращі результати при округленні до більшого значення. Але оскільки цей спосіб не враховував всіх можливих вхідних даних під час тестування, методом повного перебору було показано, що при відсіканні молодших біт максимальна похибка, на яку може відхилитись отримане значення, є менша, чому й був вибраний саме цей спосіб.

Для початкових значень синуса та косинуса ситуація є подібною, але кількість можливих початкових комбінацій дорівнює чотирьом. Для даних змінних необхідно представити числа у двійковому цілочисельному форматі при такому методі заокруглення, коли максимально можлива похибка буде мінімальною. Оптимальним значенням для ініціалізації коефіцієнта деформації при обчисленнях в 32-розрядному режимі є 9b74eda8h для синуса та косинуса.

Що стосується структури алгоритму при ітераціях, коли усі двійкові розряди заданого кута дорівнюють поточному, вибір гілки алгоритму, по якій він повинен далі йти, є довільним; оскільки всі операції з кутами є симетричними, то і похибка обчислень буде сталою для обох випадків. Це твердження, яке було теоретично прогнозовано, підтвердилось практичним тестуванням. Крім того, в цьому випадку відсутня можливість перервати хід алгоритму, тому що коефіцієнт деформації на початку коду був заданий для тридцяти двох ітерацій, через що значення в такому разі буде неточне. Щоправда, і ймовірність таких збігів є достатньо низькою:

$$\begin{gathered} \textit{ітерація}\,31-50\,\% \\ \textit{ітерація}\,30-25\,\% \\ \textit{ітерація}\,29-12.5\,\% \\ \textit{ітерація}\,28-6.25\,\% \\ \textit{ітерація}\,27-3.125\,\% \\ \textit{ітерація}\,26-1.5625\,\% \\ \ldots \\ \textit{ітерація}\,1-4.6566\cdot 10^{-12}\,\% \end{gathered} \qquad (4.10)$$

Враховуючи те, що процесор також повинен витрачати час на аналіз

подібних збігів, ймовірність появи яких є досить низькою, а для отримання точних результатів необхідне ще й зберігання в пам’яті таблиці коефіцієнтів деформації, то практична цінність алгоритму у такій його реалізації різко зменшується.

Після відлагодження класичного алгоритму CORDICa для одержання найвищої точності отримуємо наступні таблиці відхилень для синуса та косинуса:

*Таблиця 4.1*

Розподіл відхилень вихідних значень функцій синуса — косинуса для класичного методу CORDIC в одиницях молодшого розряду

| $\|\Delta n\|$ | $\Sigma \sin(\varphi_n)$ | $\Sigma \cos(\varphi_n)$ |
|---|---|---|
| 0 | 348262335 | 345989658 |
| 1 | 668701693 | 663974630 |
| 2 | 591546456 | 586612563 |
| 3 | 481862292 | 477364994 |
| 4 | 360937838 | 358064030 |
| 5 | 248276282 | 247932751 |
| 6 | 156786113 | 158862985 |
| 7 | 90916432 | 94503306 |
| 8 | 48404133 | 52396977 |
| 9 | 23642344 | 27213624 |
| 10 | 10581938 | 13296955 |
| 11 | 4341496 | 6126487 |
| 12 | 1630107 | 2653988 |
| 13 | 556146 | 1076361 |
| 14 | 170942 | 405174 |
| 15 | 46080 | 141039 |
| 16 | 10951 | 44348 |
| 17 | 2134 | 12575 |
| 18 | 345 | 2952 |
| 19 | 37 | 604 |
| 20 | 2 | 88 |
| 21 | 0 | 7 |

При застосуванні методу залишкового кута зменшується кількість ітерацій класичного методу до шістнадцяти, після чого здійснюємо апаратне або програмне множення значень залишкового кута та поточних значень синуса і косинуса. Впливати на точність в цьому випадку можемо, здійснюючи корекцію результату в останній фазі алгоритму.

*Таблиця 4.2*

Розподіл відхилень вихідних значень функцій синуса — косинуса із застосуванням методу кусково-лінійної апроксимації

| $\|\Delta n\|$ | $\Sigma \sin(\varphi_n)$ | $\Sigma \cos(\varphi_n)$ |
|---|---|---|
| 0 | 468810087 | 469470145 |
| 1 | 872120658 | 874918637 |
| 2 | 706086237 | 709235061 |
| 3 | 493051809 | 493513578 |
| 4 | 286912558 | 285872934 |
| 5 | 136237335 | 134518654 |
| 6 | 52398202 | 50327696 |
| 7 | 16179228 | 14915924 |
| 8 | 4054859 | 3352051 |
| 9 | 754292 | 507599 |
| 10 | 70792 | 43817 |
| 11 | 39 | 0 |

Подані результати відхилень для методу кусково-лінійної апроксимації майже вдвічі кращі в порівнянні з класичним методом. Наступним кроком є реалізація методу одностороннього повороту за допомогою програмного множення add-shift, який включає в себе сімнадцять чи шістнадцять ітерацій залежно від знаку отриманого залишкового кута. Даний метод є цінний тим, що при його використанні витрачається менше процесорного часу, і він проводить корекцію отриманого результату відносно молодших біт, які відсікаються при здійсненні обчислень.

*Таблиця 4.3*

Розподіл відхилень вихідних значень функцій синуса — косинуса при здійсненні програмного множення методом одностороннього повороту

| $\lvert \Delta n \rvert$ | $\Sigma \sin(\varphi_n)$ |
|---|---|
| 0 | 476881962 |
| 1 | 886614949 |
| 2 | 711122255 |
| 3 | 492763118 |
| 4 | 278362244 |
| 5 | 127427206 |
| 6 | 46309076 |
| 7 | 13296618 |
| 8 | 3447890 |
| 9 | 427924 |
| 10 | 22854 |

| $\lvert \Delta n \rvert$ | $\Sigma \cos(\varphi_n)$ |
|---|---|
| 0 | 478950496 |
| 1 | 887806538 |
| 2 | 714884303 |
| 3 | 493080431 |
| 4 | 276759124 |
| 5 | 126046667 |
| 6 | 44189214 |
| 7 | 12029541 |
| 8 | 2614302 |
| 9 | 310628 |
| 10 | 4852 |

Як бачимо, обчислення даним методом здатне зменшити максимальну похибку, а також досягнути більшої кучності отриманих результатів. Забігаючи наперед, можна сказати, що корекція забирає два такти процесора, що дещо зменшує швидкодію та обсяг коду, але дозволяє збільшити точність обчислень. У випадках, коли основним показником повинна виступати висока швидкодія, можливим є збільшення швидкодії обчислень за рахунок їх точності.

Табличний метод, який призначений для прискорення роботи алгоритму, сам по собі не здійснює обчислень, а тільки вибирає з таблиці готовий результат, що дорівнює значенню змінних, при якому слід розпочинати обчислення. Таким чином, даний метод не впливає на точність обчислень, що дозволяє змінювати параметри табличного методу, не боячись вплинути на кінцеві результати.

Комбінований метод інверсного повороту дещо відрізняється від розглянутих вище способів. Відповідно вплив здійснюваних ним операцій на кінцевий результат роботи алгоритму не однозначний, через меншу кількість

виконуваних ітерацій, а його оптимізація не остаточна. Тому результати роботи алгоритму гіпотетично можуть містити способи для покращення кінцевих результатів обчислень. При використанні корекції результату на останньому етапі роботи комбінованого алгоритму становить:

*Таблиця 4.4*

Значення похибок для комбінованого методу обчислень функцій синуса-косинуса

| $\lvert \Delta n \rvert$ | $\Sigma \cos(\varphi_n)$ | $\Sigma \sin(\varphi_n)$ |
|---|---|---|
| 0 | 382037038 | 224917494 |
| 1 | 754037524 | 536165351 |
| 2 | 709182754 | 668969219 |
| 3 | 597069812 | 693470464 |
| 4 | 429769649 | 544306255 |
| 5 | 260581422 | 349997434 |
| 6 | 137371305 | 193587272 |
| 7 | 61816699 | 94670419 |
| 8 | 25416401 | 41334412 |
| 9 | 9038761 | 17633147 |
| 10 | 3650721 | 4594179 |
| 11 | 1324902 | 1916988 |
| 12 | 596354 | 488646 |
| 13 | 209747 | 169136 |
| 14 | 66698 | 0 |
| 15 | 16362 | 0 |
| 16 | 1498 | 0 |

Максимально абсолютне відхилення значення синуса для комбінованого алгоритму становить $13 \cdot 2^{-32}$ що є хорошим, хоча і не найкращим результатом. Поведінка функції косинуса є досить неординарною. Вона забезпечує найкращу кучність результатів для невеликих значень відхилень. Для прикладу, значення відхилення п’ятдесяти п’яти відсотків кутів не перевищує $2 \cdot 2^{-32}$. Такої кучності не зміг забезпечити жоден із представлених алгоритмів. Під кінець таблиці значення відхилень почали згладжуватись, і в результаті для максимального значення відхилення отримано похибку в $16 \cdot 2^{-32}$ одиниць, проте дане відхилення не найбільше з усіх представлених алгоритмів. Серед способів для покращення

алгоритму можна відзначити повний розрахунок коефіцієнтів корекції θ. У першому випадку коефіцієнти відсікались, а в другому доповнювались. На жаль, досягнути покращення точності таким чином не вдалось. У перший варіант округлення під час тестування дало ідентичний результат у порівнянні з округленням до найближчого цілого, а в другому випадку величина похибки зросла.

Тому, як показав аналіз отриманих результатів, дані коефіцієнти θ слабо впливають на кінцевий результат, а точність алгоритму забезпечується завдяки добре підібраним способам обчислень в алгоритмі.

Для прикладу, якщо провести ітерації алгоритму над довільним кутом з діапазону [0;π/2) без використання коефіцієнтів корекції θ та не використовуючи методів залишкового множення, отримаємо максимально можливу абсолютну похибку у межах:

$$\begin{aligned} \Delta n_{max} \cdot \sin\phi &= 6236 \cdot 2^{-32} \\ \Delta n_{max} \cdot \cos\phi &= 6242 \cdot 2^{-32} \end{aligned} \qquad (4.11)$$

З такою величиною відхилення можна забезпечити 19 – 20 точних бінарних розрядів обчислювальної функції, чого може бути достатньо для обчислень, в яких не ставиться за мету висока точність результатів. У такому вигляді запропонований алгоритм можна реалізувати виключно на основі операцій додавання і зсуву, без використання будь-яких додаткових затрат пам’яті на зберігання констант та таблиць, а при правильній реалізації сам алгоритм можна розмістити у декілька десятків байт машинного коду, чого на сьогоднішній день не можуть забезпечити існуючі алгоритми обчислення значень функцій синуса і косинуса. Крім того, алгоритм в скороченому варіанті матиме високу швидкодію завдяки невеликій кількості простих логічних ітерацій, частину з яких можна вилучити.

### 4.2.1 Порівняння швидкодії алгоритмів

Питання визначення швидкодії є достатньо складним, оскільки на цей параметр впливає багато факторів при складних архітектурах процесорів.

Платформа x86, для якої розроблялись алгоритми, включає багато факторів, які, з одного боку, призначені для покращення швидкодії системи, але, з іншого, дуже неоднозначно впливають на швидкодію, залежно від елементів конфігурації системи та внутрішнього середовища процесора.

Необхідно зауважити, що у кожного процесора, в тому числі в межах одного сімейства, часто відрізняється архітектура ядра, що призводить до різних результатів замірів. І навіть на процесорах із однаковим ядром можлива зміна степінгу ядра та організація команд мікрокоду, яка зазвичай постійно вдосконалюється і може бути замінена вже на етапі експлуатації процесора. Тому проводити заміри усіх досліджуваних алгоритмів слід лише на одній, вибраній наперед платформі, без модифікації компонентів, що входять до неї.

Точне визначення кількості тактів при виконанні деякого коду є досить складним завданням. Незважаючи на даний факт, існує ряд напрацювань із цього питання, які дозволяють дати однозначну оцінку коду з точністю до одного машинного такту. Якщо алгоритм складається з лінійно виконуваних команд чи переходів, які виконуються безумовно (або умовно при постійно однаковому значенні конкретної умови), то величина тактів буде являти собою ціле число – конкретне значення кількості тактів для фрагменту коду.

При аналізі алгоритмів, у яких відбуваються умовні переходи, їх значення постійно змінюються і визначити їхню поведінку наперед не виявляється можливим, тож потрібно використовувати дещо інший підхід. У найпростішому випадку дана неоднозначність пов’язана з конвеєрним виконанням команд, а оскільки процесори x86 архітектури використовують досить складні методи конвеєризації, неможливо однозначно визначити, по якому відгалуженню програми піде конвеєр, зіткнувшись з умовним переходом. Враховуючи, що всі розроблені та аналізовані у роботі алгоритми потрапляють до цієї категорії, то і заміри потрібно виконувати наступним способом. На вхід алгоритму подаються усі можливі комбінації вхідних даних, які використовуються алгоритмом. У нашому випадку - це один тридцятидворозрядний регістр, значення якого після

кожної ітерації нашого алгоритму інкрементується та аналізується рівність його нулю. Обов’язковою умовою є незмінність даного регістра аналізованим кодом. Враховуючи, що такий циклічний перехід через велику кількість можливих значень лічильника виконується практично безумовно, можемо бути впевнені, що конвеєризація піде саме по цьому переході. Таким чином, зчитавши молодші 32 розряди лічильника кількості тактів, достатньо зберегти його в одному із невикористовуваних регістрів. Одразу після цього приступаємо до виконання основного циклу, задаючи усі можливі вхідні комбінації для нашого алгоритму. Коли лічильник збився у нуль, цикл переривається і відбувається повторне зчитування лічильника тактової частоти та шукається різниця між отриманим значенням та попереднім. Таким чином, якщо початкове значення лічильника основного циклу дорівнювало нулю, то повинно пройти $2^{32}$ ітерацій основного циклу, перш ніж відбудеться повторне зчитування лічильника тактової частоти. Тому, після запуску на виконання такого “пустого” циклу з нульовим початковим значенням лічильника, одержані два тридцятидвобітні слова будуть містити кількість тактів, які зайняло виконання алгоритму, та кількість тактів, які витратились на зчитування лічильника тактової частоти, його збереження, а також збій конвеєра у момент, коли відбувався умовний перехід, організований основним циклом. Орієнтовні значення на представленій платформі становлять : два такти для організації циклу і 50 – 60 тактів для лінійних команд та перезавантаження конвеєру. Для інших платформ це значення, скоріш за все, буде інакшим, хоча й подібним.

При дослідженні об’ємніших фрагментів коду значення цих лічильників будуть значно більшими, що дозволяє прирівняти отримане під час експерименту відхилення до нуля для спрощення обрахунків. Тим не менше, віднявши дане значення від отриманих результатів, можна точно сказати, за скільки тактів був виконаний досліджуваний код, поміщений в основному циклі.

Слід зробити наступне зауваження – як було визначено практично, для точного результату слід виконати в середньому п’ять повних проходів основного

циклу, перш ніж буде отримано точне значення кількості тактів.

Це викликано такими особливостями платформи :

- Завантаженням програми з оперативної пам’яті в кеш пам’ять усіх рівнів
- Заповнення лічильників для команд умовних переходів з такою тенденцією, яка найчастіше використовується в алгоритмі

Крім того, на код, що виконується в основному циклі, накладається ряд обмежень:

- Заборона використання даних в оперативній пам’яті, користуватись потрібно виключно регістрами чи безпосередніми операндами
- Заборона використання стеку, оскільки він розміщується в оперативній пам’яті
- Кількість команд умовних переходів повинна бути мінімальною
- Вся програма та її дані повинна вміщуватись у кеші першого рівня, інакше отримаємо «розмитий» кінцевий результат через постійні підвантажування програми з кешів нижчих рівнів
- Заборона використання програмних, системних, апаратних переривань та роботи з пристроями
- Використання регістрів, що містять поточний лічильник основного циклу та молодші розряди кількості тактів процесора

Також для коректної роботи програми потрібно: провести настроювання використовуваних програмою ресурсів (відеорежим, відеобуфер, сегменти даних та коду) до початку виконання основного циклу із заміру швидкодії.

Позитивні моменти полягають у тому, що завдяки відсутності використання стеку можна вільно, на власний розсуд, оперувати двома 32 розрядними регістрами, які призначались виключно для використання стеком. Саме в них і розміщені лічильники, які не повинні використовуватись досліджуваним кодом. Таким чином, можна взяти сегмент коду, який був написаний у багатозадачному режимі, при використанні якого ці регістри не могли

використовуватись, і вставити його в основний цикл для дослідження, не боячись, що не знайдеться вільних регістрів для зберігання даних лічильників.

Як було сказано вище, через наявність умовних переходів у досліджуваних алгоритмах є практично неможливим визначення точної цілої кількості тактів для сегменту коду через те, що на його виконання при різних вхідних даних та різному середовищі процесора буде затрачено різну кількість тактів. Тим не менше, можна дати точну, хоча й дробову, характеристику алгоритму при його довгостроковому виконанні та довільних однаково ймовірних вхідних значеннях.

Таким чином, отримане значення – це середнє арифметичне, до якого входять набір значень кількості тактів, за які виконується досліджуваний код при всіх можливих вхідних комбінаціях.

Отримуємо дані у вигляді двох тридцятидворозрядних чисел $counter_{high}$ і $counter_{low}$, перше з яких це є ціла частина кількості затрачених тактів, а друга – дробова частина. Тому для переведення її в десятковий дробовий формат потрібно обчислити:

$$\xi = counter_{high} + counter_{low} \cdot 2^{-32} \qquad (4.12)$$

Значення отриманих даних (значення лічильників) після проходження кількох “прогрівальних” циклів стають сталими, що дозволяє однозначно визначити кількість процесорного часу, затраченого на виконання досліджуваного фрагменту коду, та аналізувати тенденцію зміни швидкодії при внесенні незначних поправок в роботу алгоритму.

### 4.2.2 Аналіз швидкодії розглянутих методів

Для заміру швидкодії використовується програма, яка розрахована на виконання в однозадачному режимі роботи процесора «Real Mode». Програма працює із вимкнутими перериваннями та не має змоги працювати з функціями операційної системи та біосу, а роботу з обладнанням та відеобуфером слід здійснювати лише напряму. Виконання програми можна поділити на такі етапи:

- Налаштування сегментів даних та коду;
- Вибір відеорежиму;
- Налаштування ресурсів для роботи з відеобуфером;
- Очистка екрану;
- Заборона переривань;
- Налаштування регістрів та лічильників для виконання алгоритму;
- Зняття показників лічильників тактової частоти;
- Виконання основного циклу з усіма можливими вхідними значеннями для коду, що аналізується;
- Повторне зняття показників лічильників тактової частоти та знаходження різниці з попереднім значенням;
- Переведення отриманого значення в десятковий формат ASCII;
- Вивід результату роботи у відеобуфер;
- Безумовний перехід на етап ініціалізації регістрів та лічильників.

Лістинг 4.9 Код програми для заміру швидкодії довільного алгоритму

```
.startup
        cld
        lea     si,agl
        xor     di,di
        mov     cx,52
        rep     movsd
        cli
        mov     ax,0b800h
        mov     es,ax
        mov     eax,7000700h
        xor     di,di
        mov     cx,1000
        rep     stosd
beg:    xor     ecx,ecx
        xor     ebp,ebp
        db      0fh,31h
        mov     esp,eax
```

```
sta:    ;Main Cyc
nxt:    ;Code End
        inc     ebp
        jnz     sta
        db      0fh,31h
        sub     eax,esp
        xor     ebx,ebx
        mov     esp,edx
        mov     bx,ofs
        mov     cl,10
        add     bx,10
        wr:
        xor     edx,edx
        div     ecx
        or      dl,30h
        mov     es:[ebx*2],dl
        dec     bx
        cmp     bx,ofs
        jnz     wr
        mov     eax,esp
        dec     bx
        sub     ofs,6
        wrt:
        xor     edx,edx
        div     ecx
        or      dl,30h
        mov     es:[ebx*2],dl
        dec     bx
        cmp     bx,ofs
        jnz     wrt
        add     ofs,56h
        jmp     beg
end
```

Щоб проаналізувати алгоритм, слід помістити його код між коментарями, відміченими як «Main Cyc» та «Code End». Після того, як досліджуваний алгоритм поміщений в основний цикл, проводиться компіляція програми, і у випадку успішного завершення запускається отриманий машинний код на виконання. Перебір усіх можливих вхідних даних алгоритму на платформі для тестування тривав 15 – 25 хвилин, після чого результат виводився на екран та здійснювався повторний прохід алгоритму. Враховуючи, що для нормалізації

результатів, зв’язаних із заповненням кеш пам’яті вищих рівнів та нормуванням лічильників переходів, які прогнозуються, необхідно приблизно п’ять проходів основного циклу, після виконання яких значення кількості тактів, які витрачаються на алгоритм, є сталим числом і практично не змінюється. Тому приблизно через дві години роботи програми показники кількості тактів стабілізуються, завдяки чому через чотири - п’ять годин можна однозначно сказати, скільки тактів потребує виконання досліджуваного коду. Кількість результатів, які виводить програма, обмежена розміром відеобуфера, і при стандартному значенні в 25 рядків, заповнення яких настає через 10 – 15 годин виконання алгоритму, програма перестає бути інформативною. Тим не менше, дана величина є надлишковою, і ще до заповнення відеобуферу можна дати оцінку алгоритмові, що аналізується. Оскільки тестування усіх розроблених версій алгоритмів, кількість яких під час проведення досліджень вимірювалася десятками, потребує значної кількості часу, точні значення швидкодії наведені лише для основних методів. Так, класичний метод CORDIC потребує: 419 – 2415617222 такти, що при переведенні у десяткове дробове представлення згідно з (4.12) становить: 419.56243 такти. Дане значення одержане на тестовій платформі з процесором сімейства AMD K7. Для прикладу, можна навести дані виконання цього ж алгоритму на наступних платформах:

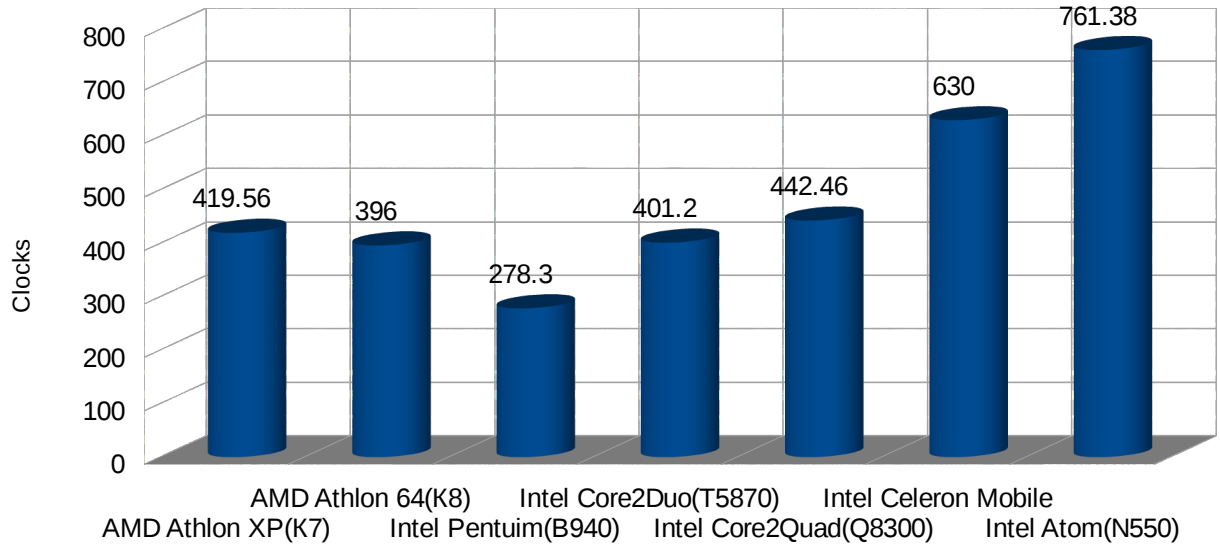



Рис 4.4 Кількість тактів алгоритму CORDIC залежно від вибраної платформи

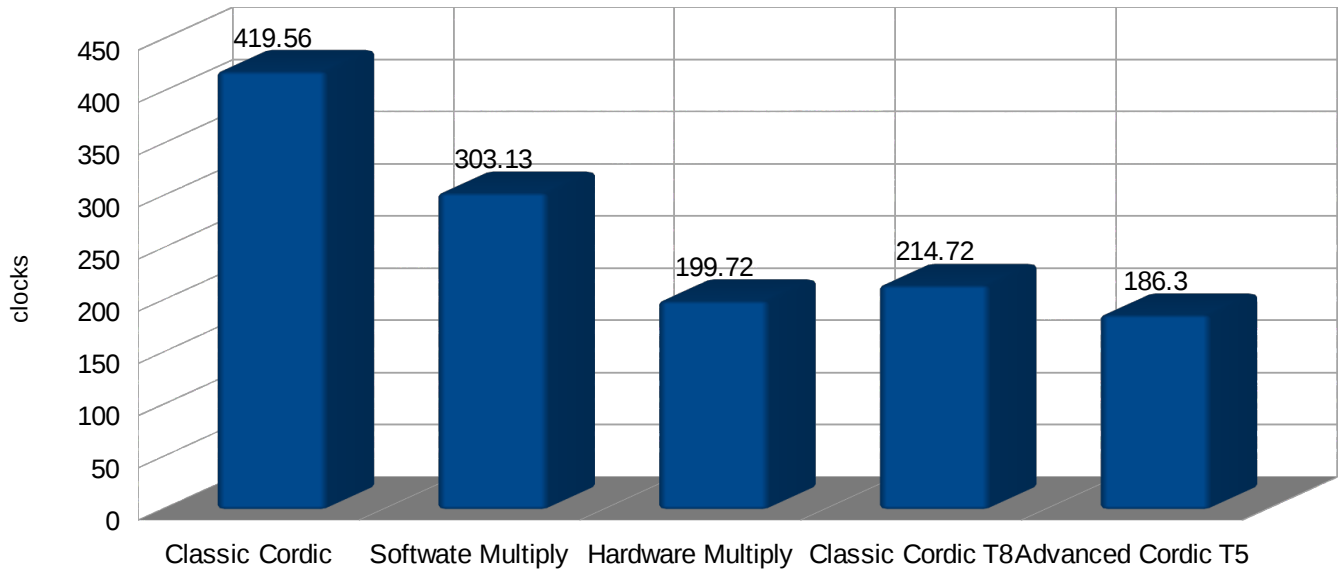



Рис 4.5 Переконвертована середня кількість тактів для досліджених методів

Як бачимо, кількість тактів суттєво відрізняється залежно від платформи, що, в першу чергу, можна пояснити досконалістю процесорної архітектури. Тому для порівняння була вибрана платформа AthloXP, при тестуванні на якій спостерігалась найкраща кучність отримуваних результатів.

Проаналізувавши отримані результати, можна зробити висновок, що класична реалізація алгоритму CORDIC має найнижчі показники швидкодії та точності, що свідчить про низьку ефективність його практичного застосування.

Використання апаратного множення здатне вдвічі прискорити виконання алгоритму, але існує обмежена кількість платформ, для яких доцільним виявиться застосування даного способу. Тому більш перспективним є використання програмних засобів множення.

Множення методом add-shift здатне забезпечити більш точні результаті при виграші швидкодії в порівнянні з класичним алгоритмом в районі 25%. Корекція результату у даному методі віднімає приблизно два такти процесора, тому можна не використовувати, пожертувавши більш точними результатами.

Табличний метод здатний суттєво збільшити швидкодію вже існуючого алгоритму до тридцяти відсотків при використанні восьмикрокової таблиці, для

зберігання якої знадобиться один кілобайт вільної пам’яті. Тому на етапі оптимізації алгоритмів, у яких планується застосування табличного методу, рекомендується підбирати параметри таблиці, виходячи з кількості вільної пам’яті, яка залишилась після реалізації основної частини алгоритму.

## 4.3 Реалізація пропонованих методів для платформи AVR

### 4.3.1 Специфіка розробки програмного забезпечення для мікроконтролера

При реалізації пропонованих алгоритмів для мікроконтролера (МК) першим кроком було написання мовою C++ макету майбутніх програм, в яких використано асемблерні вставки та спільні діапазони пам’яті для передачі аргументів. Використання мови асемблера, крім підвищення швидкодії алгоритму для покращення результатів аналізу, пояснюється ще й бажанням повністю контролювати процес розрахунків. Необхідно зазначити, що не всі компілятори підтримують конструкції для роботи з 80-розрядним “рідним” для копроцесора форматом, і при визначенні “long double” продовжуть трактувати числа як 64-розрядні. Також це відноситься і до методу округлення чисел, яких в копроцесорі передбачено чотири види. Вбудовані засоби мов високого рівня на відміну від асемблера не дозволяють контролювати та при потребі динамічно змінювати методи заокруглення чисел, які необхідно використовувати. До цих недоліків також можна додати збільшення часу виконання програми при виклику бібліотечних функцій та неефективність чи пропрієтарність методів їхної роботи. Тоді як з точки зору використання асемблера все відбувається прозоро для програміста і повністю контролюється ним. Недоліком такого підходу є підвищена складність виконання роботи та мінімізація переносимості коду програми. Ефективність використання асемблера можна продемонструвати на прикладі програми, яка здатна генерувати довільну кількість псевдовипадкових чисел згідно [31] (до 64-розрядів кожне) із заданням вхідних параметрів алгоритму та

виводу їх на екран у вигляді десяткових та шістнадцяткових значень, чи, при бажанні, у вигляді бінарного файлу, що є зручнішим для подальшої автоматизованої обробки, чи для аналізу великих вихідних обсягів інформації, наприклад, для знайдення в них закономірностей.

### 4.3.2 Характеристики мікроконтролера ATtiny 2313

Для реалізації запропонованих алгоритмів був вибраний мікроконтроллер ATtiny 2313 фірми Atmel. Мікроконтроллер має наступні характеристики:

- Архітектура – RISC
- Частота 0 – 20MHz
- 32 регістри загального призначення по 8 біт
- 128 байт оперативної пам’яті
- 128 байт пам’яті EEPROM
- 2Кбайт Flash пам’яті
- 18 ліній вводу/виводу
- ціна: $1 (станом на березень 2018р.)

З вісімнадцяти ліній вводу-виводу МК для оперування фактичними даними при роботі алгоритму практично доступними є лише п’ятнадцять ліній. Інші три виводи використовуються кварцом для задання частоти МК та подачі сигналу скиду. Арсенал команд МК містить лише логічні інструкції та команду додавання. При цьому команда віднімання реалізується через команду додавання шляхом заміни знаку аргументу через представлення його в доповняльному коді. Виконання множення та ділення даний МК не підтримує. Обмеження по командах: лише 16 з 32-х регістрів підтримують роботу з безпосередніми операндами, заданими в команді, і тільки частина команд здатна до роботи з ними. Наприклад, відсутнє додавання з числом, заданим в команді, (як логічне додавання, так і арифметичне), відсутнє порівняння з операндом із врахуванням

переносу. Логічні зсуви здійснюються лише на один розряд, з них циклічні лише через прапорець переносу. Щодо переваг МК, то до них слід віднести можливість виконання більшості операцій за один такт, роботу на частоті 20MHz а також низьку ціну МК.

### 4.3.3 Комунікація з мікроконтролером

Для підключення мікроконтролера потрібно вибрати порт із можливістю паралельної передачі значної кількості розрядів за один такт. Такий спосіб є оптимальним з точки зору можливостей мікроконтролера, оскільки дозволяє передавати чи приймати декілька біт інформації за одним пакетом, використовуючи семи чи восьмирозрядні порти мікроконтролера. Також доцільною буде можливість роботи порта у двонаправленому режимі (bi-directional). Виходячи з цих міркувань, оптимальним вибором буде порт IEEE 1284, який присутній на більшості материнських плат ПК. Тут слід зазначити, що інтегрована в кристал мікроконтролера апаратна підтримка стандарту UART не підходить з декількох причин, зокрема:

- Значно нижча швидкість передачі даних, яка, скоріш за все, буде недостатньою при роботі кристалу на високих частотах
- Необхідність програмного налаштування порта та здійснення контролю передачі даних в т. ч. з використанням переривань зі сторони контролера, що суттєво вплине на швидкодію
- Необхідність використовувати осцилятори лише з фіксованою, сумісною для послідовного порта частотою, що суттєво обмежить як діапазон частот, на котрих може працювати алгоритм, так і можливість роботи від внутрішнього генератора

Всіх вищевказаних недоліків позбавлений порт IEEE 1284, швидкість передачі даних в якому є достатньою для поставленої задачі. А наявність різноманітних режимів роботи порта дозволить вибрати оптимальний спосіб

комунікації, виходячи з потреб алгоритму.

В стандарті, представленої для порта IEEE 1284 [75], можна виділити такі ключові моменти:

- Порт має двадцять п’ять виходів, з яких доступними до використання є лише перші сімнадцять. Решта ліній замкнуті на землю
- Для передачі даних використовуються 8 ліній
- Для передачі службової інформації – 9 ліній, з яких чотири є вихідними, а п’ять - вхідними
- Підтримується п’ять режимів роботи порта як з апаратним, так і з програмним контролем передачі даних (hardware / software flow control)
- Швидкість і напрямок передачі визначається режимом роботи і лежить в межах від 100Кбайт/с до 24Мбайт/с
- Підтримка переривань (IRQ) та прямого доступу до пам’яті (DMA). Останнє тільки для режиму (ECP)

### 4.3.4 Режими роботи IEEE 1284

Як було сказано вище, існує п’ять режимів, в яких може працювати порт:

- SPP (Standard Parallel Port) – стандартний паралельний порт
- Nibble Mode – напівбайтовий режим (Nibble *англ.* - 4 біта)
- Byte Mode – байтовий режим
- EPP (Enhanced Parallel Port) - розширений паралельний порт
- ECP (Extended Capabilities Port) – порт широких можливостей

Дані режими можна поділити на дві підгрупи: з апаратним та програмним контролем передачі даних. Програмний контроль передбачений в перших трьох режимах (SPP / Nibble / Byte Mode), а апаратний - в наступних двох (EPP / ECP mode). Вибір конкретного режиму здійснюється на основі переговорів (negotiation) між ПК та підключеним пристроєм, за результатами якого вибирається

підтримуваний обома сторонами переговорів режим. Опишемо коротко кожний режим та його можливості.

Standard Parallel Port mode – режим стандартного паралельного порта, який підтримується усіма без виключення контролерами паралельних портів і робота у якому проводиться у випадку неможливості роботи підключеного пристрою в будь-якому зі складніших режимів чи у випадку ігнорування будь-якою із сторін ініційованих переговорів. У цьому режимі вісім ліній (2 - 9) табл. 4.5 використовуються лише для передачі даних, а решта (1, 10 – 17) - для передачі службової інформації як до так і від пристрою, згідно з призначеннями та напрямком передачі кожного з каналів. Режим SPP передбачає можливість кожного пристрою функціонувати хоча б на швидкості 100кбайт/с (100кHz), проте більшість портів нормально функціонує в заданому режимі при швидкостях 600-1000 кбайт/с, залежно від реалізації контролера порта, вмонтованого в чіпсет материнської плати.

Nibble Mode – режим, який був розроблений для усунення основного недоліку стандартного режиму (SPP), а саме - неможливості передачі даних від підключеного пристрою до ПК (крім жорстко фіксованих сигналів). Тому в цьому режимі чотири з п’яти ліній, які використовувались в SPP для прийому сигналів керування, використовуються для передачі даних, що дозволяє передати чотири біти за один такт.

Перевагою даного режиму є його повна сумісність з SPP, і для його використання не потрібно вносити зміни в архітектуру порта. Необхідною умовою виступає лише підтримка пристроєм даного режиму та програмного забезпечення ПК, що здатне коректно працювати з даним протоколом. Недоліком режиму є його асиметричність, через що не всі лінії, які задіяні в SPP, а саме роз’єми (6…9) не використовуються. Також складність у реалізації виникає через необхідність ділення кожного переданого байту на дві частини, а стороні, яка приймає, – виконувати їх складання, що збільшує складність алгоритму роботи за цим стандартом та завантажує процесор та шину вводу/виводу ПК.

Режим byte mode був розроблений, у першу чергу, для усунення недоліків режиму Nibble mode, таких, як передача байту частинами. У цьому режимі передача та приймання даних здійснюється через лінії даних (2..9), напрямок яких визначається бітом “5”, в регістрі контролю порта (Port Control Register). Передбачається, що наявність цього біта, а також можливість програмної зміни його значення, свідчить про можливість порта зі сторони ПК функціонувати в даному режимі. Швидкості приймання та передавання інформації лежать в тих самих межах, що і для стандартного режиму SPP.

EPP mode – режим з апаратною реалізацією контролю передачі даних, є удосконаленим режимом byte mode, в якому апаратне забезпечення здатне саме генерувати усі необхідні службові сигнали для передачі та прийому даних. Цей режим дозволяє однією командою зі сторони ПК передати чи прийняти 32 розряди інформації, тоді як всі сигнали керування, які в попередніх режимах генерувалися програмно, будуть згенеровані апаратно. З точки зору алгоритму, що розробляється, використання режиму EPP не є бажаним, оскільки передбачає генерацію додаткових контрольних сигналів мікроконтролером під час роботи, що зменшить швидкодію та збільшить код алгоритму, а можливість передачі 32 біт за такт здатна покращити лише програмну частину та швидкість роботи зі сторони ПК, але шляхом ускладнення її на підключеному до порта пристрої. Остаточним аргументом щодо неприйнятності використання даного режиму становить тайм-аут, який генерується апаратно через 10 мікросекунд при неодержанні даних від пристрою при їхньому запиті. В цей проміжок часу програмна частина алгоритму МК може не вкластись, що призведе до переключення порта назад в стандартний SPP режим. Якщо ж і вдасться успішно забезпечити виконня усіх часових обмежень пристроєм, використання даного протоколу змусить кристал завжди працювати лише на високих частотах, без можливості покрокового відлагодження коду чи забезпечення наглядної роботи алгоритму.

Режим ECP був остаточно затверджений у 2000р. і підтримує різноманітні функції порта, такі, як робота одночасно з кількома пристроями, автоматичне

визначення параметрів кабелю, його характеристик та максимальної пропускної здатності (до 24 Мбайт/с), а також передачі та прийому даних апаратно, безпосередньо у пам’ять ПК. Даний режим є зручний для програми, функціонуючої на комп’ютері, але він достатньо складний у реалізації зі сторони периферії. Враховуючи необхідність забезпечення можливості переговорів, параметрів, що передаються, приймаються та обробляються в цей час, і для коректної реалізації даного режиму потрібно використати ще один МК з аналогічними характеристиками, що не входить у поставлену задачу. Достатньо часто виробники периферійних пристроїв відмовляються від реалізації даного режиму, якщо не мають завдання досягнути максимальної пропускної здатності порта, враховуючи надлишковість такого підходу - як фінансового, так і апаратного - і звертають свою увагу в бік простішого режиму, наприклад, EPP.

Взявши до уваги подану у стандарті IEEE 1284 інформацію, а також визначивши сильні та слабкі сторони кожного режиму роботи порта, можна однозначно сказати, що режимом, який найкраще задовільнить потреби щодо реалізації способу комунікації мікроконтролера Attiny2313 з ПК, є режим byte Mode. Даний режим у подальшому став основою організації з’єднання кристалу та ПК, а також згідно з ним був розроблений протокол комунікації між ними.

### 4.3.5 Пріоритетний режим byte mode

Першим кроком при реалізації алгоритму комунікації є перевірка доступності режиму byte mode на цільовому сегменті платформ. Як виявилось під час дослідів, існує частина контролерів, які не здатні працювати в заданому режимі, незважаючи на те, що всі інші режими, передбачені в стандарті IEEE 1284, функціонують коректно. Звичайно, існують старіші платформи, які були випущені до остаточного затвердження стандарту IEEE 1284, проте якраз на даних платформах режим byte mode функціонує коректно, вони повністю його підтримують. Також можна сказати, що серед ПК, випущених за останні десять

років, всі режими, передбачені стандартом 1284, реалізовані.

Інформація, подана в порт, може бути записана, а біт, що відповідає за ввімкнення даного режиму в контрольному регістрі порта, скинутий в нуль. Зарезервовані біти, які не використовуються, і, відповідно, не підтримуються апаратно, за замовчуванням виставлені в одиницю як за стандартом, так і за прийнятою практикою. Незважаючи на нестандартне нульове значення прапорця підтримки режиму byte mode, цей параметр можна було змінити програмно, що, знов-таки за стандартом IEEE 1284, вказує на підтримку портом даного режиму. Для перевірки можливості порта щодо роботи в режимі byte mode був розроблений код для програмного тестування його можливостей та виводу звіту за результатами тестування.Апробація щодо коректності роботи ПЗ була підтверджена декількома десятками протестованих платформ. Хоча, звичайно, не правильно було б виключати ймовірності існування платформ із деякою нестандартною конфігурацією, у яких результати програмного тестування можуть виявитись некоректними.

Тому остаточно підтвердити (чи спростувати) підтримку портом режиму byte mode можна тільки при проведенні апаратного тесту порта, який полягає в замиканні одного з виводів (по замовчуванню першого) із будь-якою з ліній даних (2..9 виводи). Під час такого експерименту використовується спеціально розроблена програма низькорівневого тестування. І, у випадку одержання позитивних результатів, можна однозначно стверджувати, що режим byte mode підтримується портом. Під час проведення даного досліду слід бути обережним, оскільки існує хоч і незначна ймовірність пошкодити порт у випадку відсутності чи некоректного функціонування вищезгаданого режиму.

Відсутність підтримки різними платформами режиму byte mode, як виявилось, була викликана розбіжностями технічних параметрів південного моста материнської плати. Так, чіпсети фірми Via для процесорів AMD K7, не здатні використовувати режим byte mode, тоді як чіпсети nFoece (nVidia) для 64 розрядних процесорів AMD K8,K10 на усіх протестованих платформах його

підтримували. Щодо чіпсетів фірми Intel – то вони зазвичай підтримують режим byte mode, хоча не завжди можуть працювати у всіх інших режимах.

### 4.3.6 Протокол комунікації ATtiny 2313 (IEEE 1284)

Щоб приступити до реалізації протоколу комунікації, слід насамперед визначити, які саме дані планується передавати між МК та ПК. За задумом, кристал повинен самостійно проводити усі розрахунки алгоритму, а в момент одержання результатів викликати за допомогою перериванням ПК, завдяки чому досягається розвантаженість процесора комп’ютера організацією комунікації, а дані зчитуються лише в момент виникнення переривання, ініційованого контролером. Виходячи з цих міркувань, можна запропонувати такі етапи організації зв’язку зі сторони ПК:

- Одержання програмою вхідних параметрів
- Ініціація рестарту кристалу з визначенням довжини імпульсу зі сторони ПК
- Передача одержаних параметрів алгоритму у кристал
- Переключення ліній виводу на ввід для ПК і з вводу на вивід для МК
- Завершення роботи основної частини програми із залишенням резидентної частини в пам’яті для обробки переривань від контролера
- Одержання та запис у файл отриманих від контролера даних при виникненні переривання від нього
- При досягненні алгоритмом необхідної кількості згенерованих даних обробник переривання посилає мікроконтролеру сигнал про зупинку генерації послідовності

Подібно виглядають і етапи обробки переривання зі сторони мікроконтролера:

- Після одержання сигналу скидання МК переходить в режим очікування на передачу вхідних даних алгоритму
- При одержанні параметрів переключає лінії вводу на вивід

- Виконує першу частину алгоритму (перші шістнадцять кроків), після чого опускає лінію переривання в нуль
- Виконавши другу частину алгоритму, аналізує біт зайнятості ПК і при його одиничному значенні виставляє отримане значення на восьмибітну шину обміну між МК та ПК
- Генерує переривання, після чого проводить необхідні маніпуляції зі значенням вхідного кута згідно з формулою розрахунку та безумовно переходить на початок алгоритму генерації
- Даний процес відбувається до моменту одержання МК сигналу reset від ПК

### 4.3.7 Організація фізичного з’єднання компонентів

Перш за все слід визначити сигнал reset. Одиничне його значення свідчить про відсутність сигналу, а нульове – про активність. В МК цей сигнал підключений до першого роз’єму (рис. 4.6).

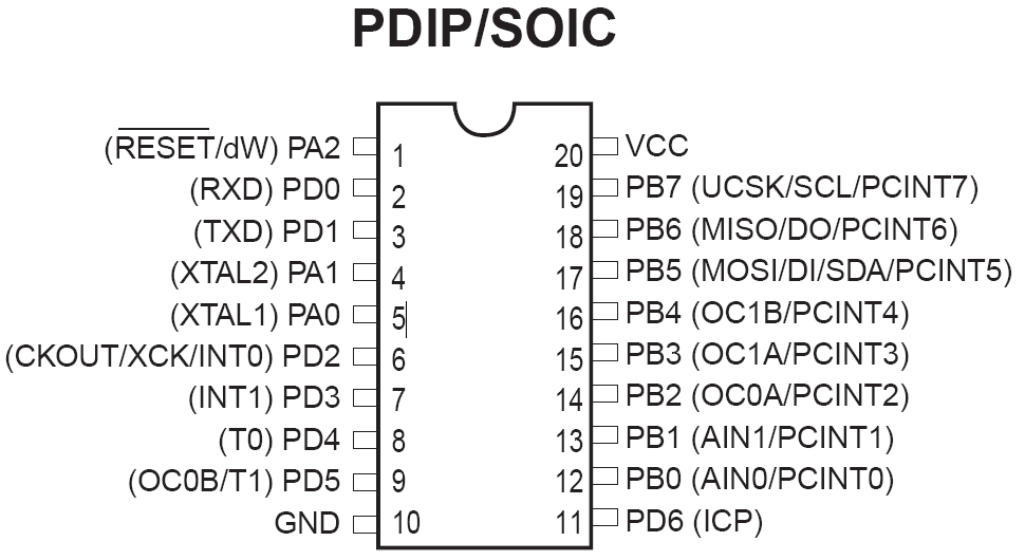



Рис. 4.6 Виводи МК ATtiny 2313

Програмно цей вивід доступний через біт порта PA2. Для збільшення наочності як апаратної, так і програмної частини даний вивід підключений до першого роз’єму порта (рис 4.7).

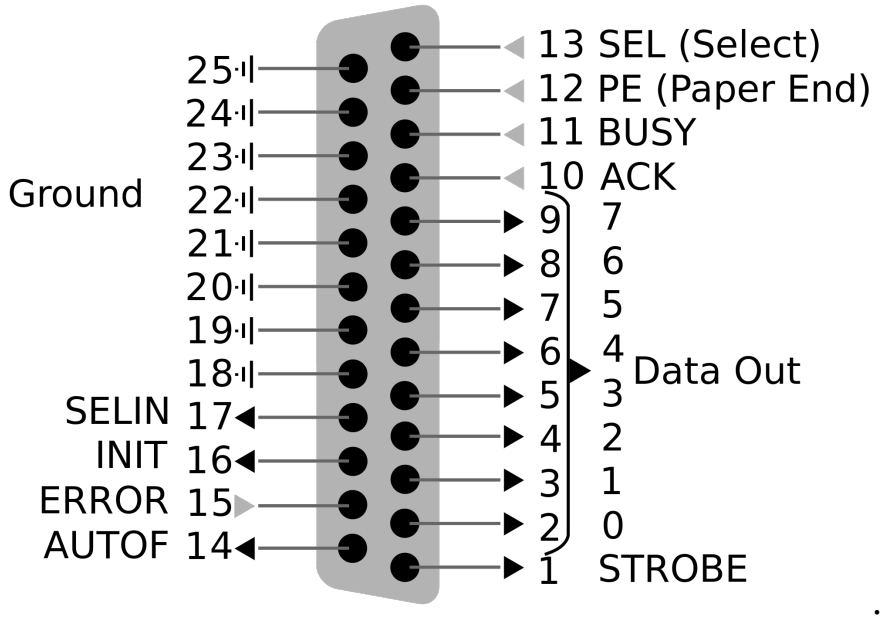



Рис. 4.7 Виводи порта IEEE 1284

Даний вивід управляється нульовим бітом регістру контролю порта IEEE 1284 і є інвертованим, одиничне значення якого свідчить про нульовий сигнал на лінії та активність цього сигналу для МК.

Наступним кроком є організація передачі даних. В порті 1284 за це відповідають лінії 2..9, які за допомогою режиму byte mode можна переключити з вивода на ввід. В МК є лише один восьмибітний порт – PORTB. Фізично він виведений на 12..19 виводи, де молодший номер виводу відповідає молодшому номеру розряду. Відповідно, наступні вісім ліній порта 1284 2..9 замкнені на 12..19 виводи МК. За організацію переривань у порті 1284 відповідає вхід ACK #10. Зростаючий сигнал свідчить про наявність переривання на цій лінії. Дана лінія, як і лінії передачі даних, при їхньому переведенні в режим прийому є підключеними до підтягуючих резисторів, що знімає необхідність робити це фізично чи за допомогою функціональності МК. Схема організації порта у МК показана на рис 4.8.

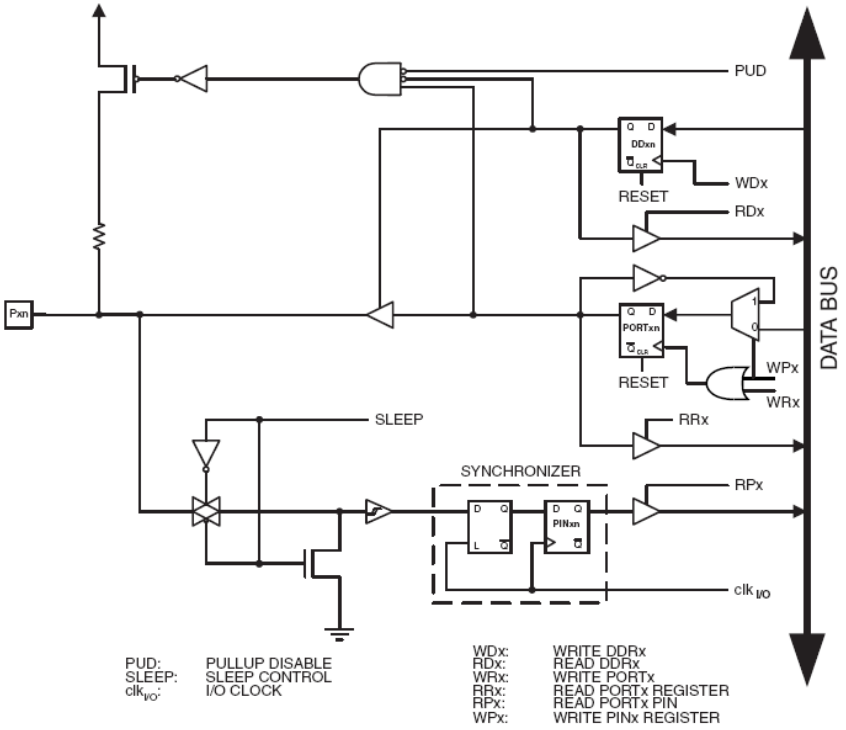



Рис. 4.8 Електрична схема організації порта вводу-виводу у МК

Подібно виглядає і організація вводу-виводу у порті IEEE 1284, Для простоти наведено лише умовну його частину (рис 4.9).

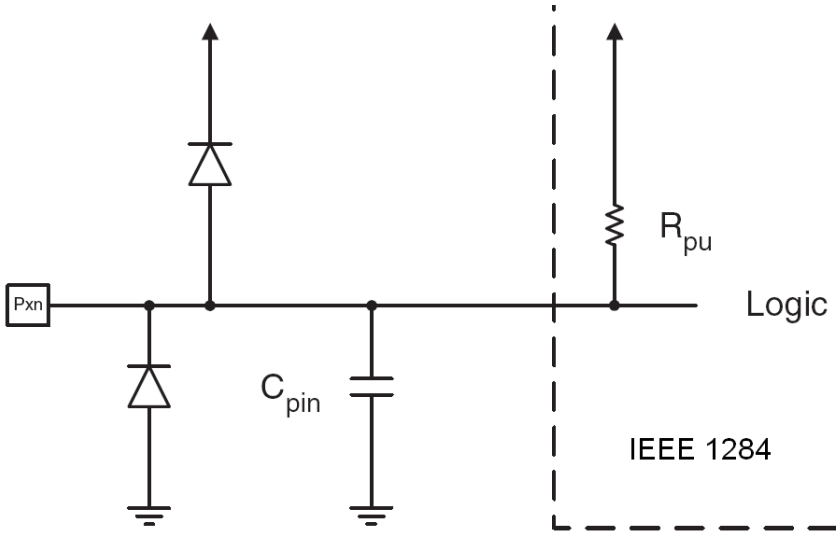



Рис. 4.9 Схема порта 1284 з підтягуючим резистором

На рис 4.9 $R_{pu}$ - підтягуючий резистор номіналом 220 ом, що вмикається при переході порта 1284 в режим прийому даних. $C_{pin}$ – ємність проводу з’єднання логіки двох пристроїв для врахування факту наявності гістерезису при формуванні часових характеристик сигналу. Забігаючи наперед, можна сказати, що під час

тестування та відлагодження алгоритму виявилась необхідність підключити між МК та 1284 ще одну сигнальну лінію: між чотирнадцятим виходом порта і першим виходом мікроконтролера. Причиною такого кроку є потреба в призупиненні виконання алгоритму МК у відлагоджувальних цілях.

Також в електричній схемі слід зазначити наявність кварцу на частоті 20MHz. Підкючається він до фіксованих контактів МК, а саме 4 і 5, за які відповідають програмні порти PA0 і PA1. Наявність кварцу дозволяє забезпечити роботу МК на частотах до 20MHz, тоді як при використанні вбудованого генератора частота осциляції обмежується вісьмома мегагерцами. До недоліків використання кварцу можна віднести хіба що необхідність у наявності програматора з кварцом із аналогічними характеристиками, оскільки для поширеного FBPRG програматора даний кристал стає недоступний. Програмно за підключений кварц відповідають старші вісім ф’юзів МК, які якраз і визначають його наявність та характеристики. Електрична схема під’єднання кварцу до МК подана на рис 4.10.

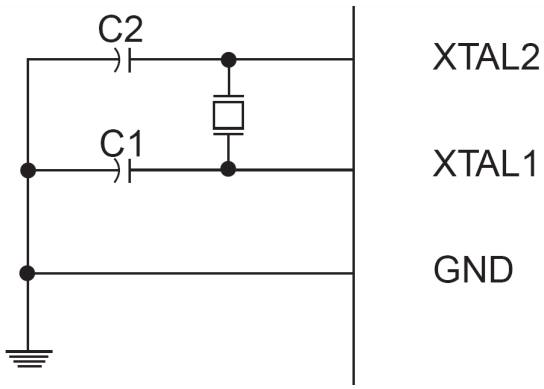



Рис. 4.10 Підключення кварцу до МК

Підключення конденсаторів є рекомендоване виробником МК, і їх наявність також є бажаною при функціонуванні МК на частотах від 0,9MHz. Ємність може коливатись від 12 до 22 пікофарад. У випадку, що розглядається, взято два конденсатори ємністю по 18пФ.

Живлення до МК подається на 10 та 20 виходи. Номінальне живлення може становити в межах від 1,8 до 5,5 В. Критичними параметрами є напруга 1,1В, нижче якої відбувається примусовий скид МК, доки напруга не зросте до

величини 1,2В. Верхня критична межа становить 6В, і доцільність у наближенні до неї виникає хіба що при роботі МК на частотах, вищих за встановлені виробником. У даному випадку для функціонування МК на частотах 8MHz і вище слід подавати живлення не нижче 4.5В, що відразу відкидає можливість живлення МК від порта 1284, яке є цілком прийнятним для нижчих частот. Подати живлення безпосередньо від БЖ комп’ютера є небезпечним, оскільки будь-який пристрій, підключений до цієї ж лінії живлення, може створити небажаний імпульс, який здатний значно перевищити граничну для МК величину в 6В. Тому живлення вирішено подавати від USB порта, яке становить 5,0В, сила струму від USB порта хоч і значно нижча від 5-ти вольтової лінії блока живлення, все ж на порядки вища, ніж та, що споживається мікроконтролером, і яка, згідно з графіком, наданим виробником, становить біля 11 міліампер (рис 4.11), тоді як порти USB першої та другої ревізії видають струм у межах 500 – 1000мА.

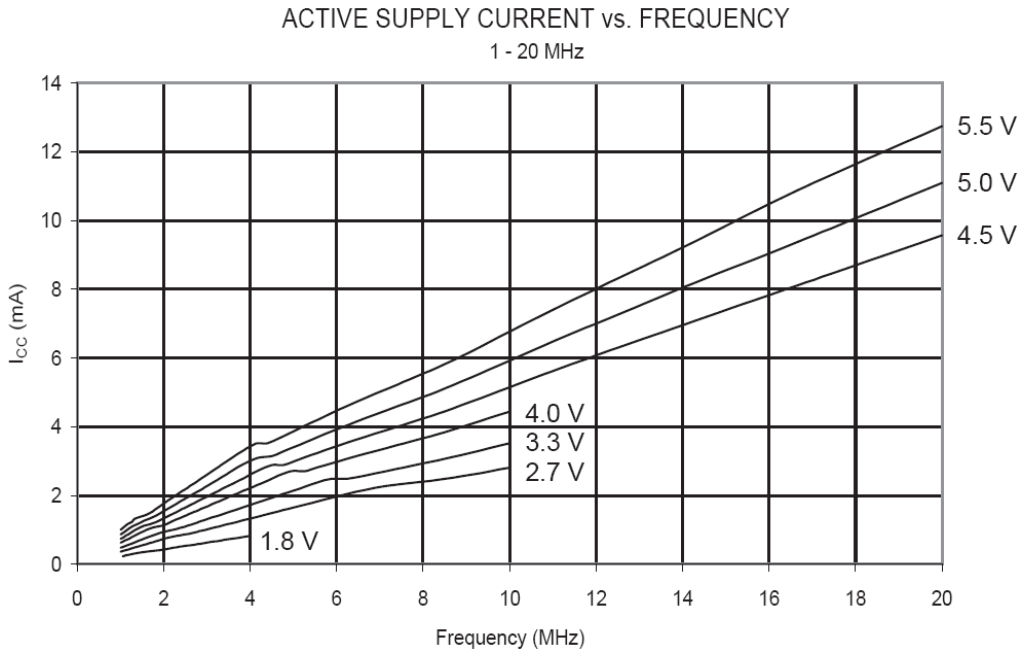



Рис. 4.11 Енергоспоживання МК при різних частотах та струмах

Задля підвищення безпеки та надійності функціонування МК до нього паралельно живленню було підключено два конденсатори ємністю 100 нанофарад, та 1000 мікрофарад, слюдяний та електролітичний відповідно. Перший здатний гасити короткі імпульси, створені МК, наприклад, при виконуванні ресурсоємної

операції, під час якої струм різко падає, а потім зростає. Електролітичний також гасить коливання, але вже не короткочасного, а довгострокового характеру. Наприклад, МК почав споживати більше струму, коли один з його виходів був замкнений на землю. Без використання конденсатора в такому випадку можливе короткочасне падіння напруги, що призведе до скидання МК чи подальшої некоректної його роботи. Нижче в таблиці 4.5 подана розпіновка контактів МК:

*Таблиця 4.5*

Розпіновка кабелю між IEEE 1284 та ATtiny 2313

| ATtiny 2313 pin # | Destination | Function |
|---|---|---|
| PA2 #1 | 1284 #1 | Reset |
| PD0 #2 | 1284 #14 | Debug wire |
| PA1 #4 | Quartz | External Oscillator |
| PA0 #5 | Quartz | External Oscillator |
| GND #10 | USB Gnd | Ground |
| PD6 #11 | 1284 #11 | ACK / IRQ call |
| PB0 #12 | 1284 #2 | Data line 0 |
| PB1 #13 | 1284 #3 | Data line 1 |
| PB2 #14 | 1284 #4 | Data line 2 |
| PB3 #15 | 1284 #5 | Data line 3 |
| PB4 #16 | 1284 #6 | Data line 4 |
| PB5 #17 | 1284 #7 | Data line 5 |
| PB6 #18 | 1284 #8 | Data line 6 |
| PB7 #19 | 1284 #9 | Data line 7 |
| Vcc #20 | USB +5V | Power supply |

### 4.3.8 Управління мікроконтролером

Для забезпечення керування мікроконтролером була розроблена програма, яка функціонує на ПК і виконує наступні дії:

- Приймає параметр, який був заданий при запуску, та створює файл із заданим іменем. При виникненні помилки – виводить повідомлення, використовуючи функцію 0x9 переривання 21h, та завершує роботу
- Зчитує пакет даних з восьми байт, уведених із клавіатури, використовуючи функцію 0x1 переривання 21h. Оскільки значення параметрів алгоритму 32-

бітні, то перші чотири байти задають величину Z алгоритму, а наступні чотири - величину ΔX. Під час виконання цього циклу програму можна перервати комбінацією клавіш Ctrl+Break

- Рестарт МК з паузою в 0,5сек. Дану величину при бажанні можна змінити і перевести порт в режим передачі — byte mode згідно зі стандартом IEEE-1284
- Послідовно передаємо мікроконтролеру пакет даних із восьми символів та чотирьох додаткових байт, що визначають число π, яке використовується для корекції кута. Даний нюанс зумовлений особливістю команд МК, серед яких немає команди порівняння числа з безпосереднім операндом з урахуванням переносу, через що порівняння чисел є ускладненим у випадку, якщо вони не вміщаються у вісім розрядів. А завантажувати значення для порівняння в регісти МК найкраще під час процесу ініціалізації.
- Переключення ліній даних на вхід, використовуючи специфіку режиму byte mode, змінивши п’ятий біт в контрольному регістрі порта. Програмування нового значення маски переривань для контролера переривань 8259, шляхом перезапису (запису, зчитування та повторного запису) нової маски в порт 21h. Переключення сегментів та модифікація таблиці переривань із заміною вектора 0Fh для 1284 порта та вектора 28h для організації запису в файл
- Завершення роботи програми та передача управління ОС із збереженням в пам’яті резидентних програм та двох сегментів пам’яті по 4Кб для запису в файл
- При виникненні переривання 0Fh від 1284 порта обробник даного переривання зчитує інформацію з порта та заносить її в активний буфер і у випадку, якщо він заповнений, переключається на резервний буфер
- При виникненні переривання 28h (не документоване переривання), суть якого дати можливість резидентним програмам використати функції ОС, де програма обробки переривання порівнює значення активного буфера із тим,

який вже був записаний на диск. Якщо ці значення ідентичні, то керування передається підпрограмі, яка була за цим номером по ланцюгу (hooking)

- Якщо активний буфер не збігається з останнім записаним, то здійснюються виклики наступних функцій переривання 21h: 3Dh, 42h, 40h, 3Eh. Дані функції відкривають файл, заданий при запуску програми, настроюють позицію запису на кінець файлу, записують буфер в файл та закривають його. При виникненні будь-якої помилки під час кожної з них програма обробки переривання відновлює значення регістрів і передає управління наступній підпрограмі по ланцюгу
- При успішному запису в файл обчислюється розмір файлу. У випадку, якщо згенерована кількість перевищує задану при запуску програми величину, яка, за замовчуванням, становить один гігабайт, генерація припиняється і шляхом генерації сигналу Reset, нульового біта контрольного регістру порта і порт IEEE 1284 переключається назад у режим роботи за замовчуванням - SPP mode

При написанні та відлагодженні даної програми були виявлені деякі особливості її функціонування. Знаходячись у фоновому режимі, програма не має ніяких документованих шляхів звернутись до функцій ОС, а такі спроби будуть блокуватись самою ОС, наприклад, при спробі дописати файл, який був створений і відкритий тією ж програмою, але до переходу в фоновий режим. Це не залежить від типу переривання, яке викликається, і були перебрані усі документовані шляхи для її вирішення: від перенаправлення функції 0x1 переривання 21h з консолі у файл до прямого запису на диск з допомогою функції переривання BIOS 13h, яке не є регенеративним. Тому чи не єдиним способом зробити запис на диск доступним буде використання недокументованої можливості - переривання 28h. Незважаючи на це, багато програм перехоплюють та використовують дане переривання і не завжди передають керування далі по ланцюгу, за рахунок чого при запуску інших прикладних програм запис на диск може зупинитись, хоча генерація даних по перериванню номер сім контролера переривань продовжує

надходити, а дані зчитуються з МК і вносяться в буфер. Також слід мати на увазі, що переривання порта 1284, на відміну від інших переривань, здатне виникнути без наявних на це причин. Така поведінка контролера переривань є вивчена та описана, наприклад, в [78], але тим не менше виробником не виправлена. Можливо, саме через це більшість операційних систем, такі як DOS, Windows та Unix-подібні ОС не використовують дане переривання, а корпорація Microsoft не рекомендує використання переривань цього порта у програмному забезпеченні.

### 4.3.9 Особливості програмування і розпределення ресурсів МК

Алгоритм CORDIC, який лежить в основі обчислень, реалізованих у МК, був випробуваний у двох різних комбінаціях - із п’ятикроковою та шестикроковою ТПВ. Усі попередні тести для даного алгоритму, в тому числі для x86 платформи, здійснювались при п’ятикроковій таблиці, виходячи з міркувань, що вбудованим системам з обмеженими ресурсами доступно принаймні $2^8$ байт постійної пам’яті. У випадку використання мікроконтолера після написання та відлагодження коду алгоритму виявилось можливим використати шестикрокову таблицю, тобто запас вільної статичної пам’яті становив $2^9$ байта. Відповідно, за основу для порівнянь взятий шестикроковий варіант коду із зазначенням його відмінностей від п’ятикрокового варіанту.

Весь код, написаний для МК, можна поділити на три основні частини:

- Ініціалізація, яка включає в себе настройку портів МК, частоти роботи, зчитування та перенаправлення ліній даних МК
- Безпосереднє обчислення заданої функцій, яке представляє основний цикл програми
- Статичні таблиці функцій, розміром 256 та 512 байт для 5-ти і 6-ти крокового способу відповідно, які містяться у верхніх адресах флеш пам’яті

Звичайно, що найбільш ресурсоємним виявився етап обчислень значень функції. Цей етап, у свою чергу, можна поділити на три підетапи, кожен з яких

також має в собі дві модифікації. Структурно етап генерації на прикладі функці синуса та косинуса можна зобразити наступним чином:

- Табличний метод
    - переключення на сторінку з таблицю синусів та зчитування значення відповідно до кута
    - переключення на сторінку з таблицю косинусів та зчитування значення відповідно до кута
- Гібридний метод
    - розрахунок кута для розрядів 6...9 за повною формулою
    - розрахунок кута для розрядів 10...15 за спрощеною формулою
- Метод Add-Shift
    - розрахунок для розрядів 16...23 з урахуванням 24 біт кута деформації
    - розрахунок для розрядів 24...31 з урахуванням 16 біт кута деформації
- Вивід результату та розрахунок наступних параметрів для обчислень.

Основним показником, за яким буде визначатись якість алгоритму, є кількість тактів, необхідних для обчислення тригонометричної функції. В дослідженні з реалізації алгоритмів на x86 платформі у розділі  було здійснено аналіз точності обчислень кожного з відомих методів на основі CORDIC, та вибір оптимального способу за показниками швидкодії. Тому основний акцент при реалізації алгоритмів на платформі AVR – дослідження функціонування методів при обмежених обчислюваних можливостях апаратних засобів, визначення основних характеристик алгоритму в цих умовах та перспектив його подальшого практичного впровадження.

Особливістю платформ сімейства восьмирозрядних AVR є максимально спрощений конвеєр команд, що дозволяє ще на етапі проектування алгоритму точно розрахувати кількість тактів, необхідних для виконання деякого фрагменту коду. Конвеєр МК включає лише два етапи – вибірку та виконання. Кожна з них виконується протягом одного такту, що на виході з конвеєра дає виконання однієї команди за такт за умови, що не відбулась команда умовного чи безумовного

переходу. Переважна більшість команд виконується за один такт, тоді як команди переходу чи комади з умовним пропуском операції виконуються за два такти.

Розрахунок кількості тактів, протягом яких виконується досліджувана функція, можна проводити як для самого алгоритму обчислень, без урахування додаткових команд (здебільшого для організації основного циклу та вводу-виводу даних), так і для загальної кількості тактів, виконаних протягом основного циклу, а саме, між командами генерації переривання про готовність результату. Можна сказати, що перший показник – це свого роду недосяжна теоретична величина швидкодії досліджуваного алгоритму, але яка найбільш точно відображає якість запропонованого методу. Тоді як другий показник – це практичний результат швидкодії методу і в його сукупності із першим показником дозволяє судити про довершеність додаткового програмного коду, “що обслуговує” алгоритм у запропонованій версії його реалізації.

Розрахунки швидкодії основної частини алгоритму можна розділити на декілька етапів. Перша частина - табличний метод. Виконується завжди за однакову кількість ітерацій. Для п’ятикрокової таблиці кількість тактів становить 26, а для шестикрокової- 28. Різниця спричинена необхідністю переключення сторінки, в той час, як в п’ятикроковому методі таблиці синуса та косинуса вкладалися в межі однієї сторінки. Зрозуміло, що якщо використовується шестикроковий алгоритм, то необхідність у п’ятому кроці, поданому в таблиці 4.6, відпадає.

*Таблиця 4.6*

Час виконання алгоритму залежно від значення біт вхідного кута

| Номер біту кута | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16...23 | 24...31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Нульове значення | 74 | 62 | 46 | 41 | 49 | 45 | 49 | 45 | 51 | 41 | 31 | 10 | 9 |
| Одиничне значення | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 7 | 6 |

Виходячи з даних таблиці, добре видно, який приріст швидкодії дасть, наприклад, збільшення розміру табличного методу, чи перевагу add-shift операцій

над гібридним CORDIC-ом, при однаковій ймовірності одержання будь-якого з варіантів вхідних кутів. Слід зазначити, що в таблиці не враховуються ряд нюансів та способів використання оптимізації для деяких видів обчислень. Наприклад, обчислення величини корекції залишкового кута можна здійснювати після шістнадцятої ітерації, тобто перед використанням формул залишкового множення. Такий підхід буде ефективнішим, якщо ймовірність старших нульових розрядів є високою, а корекцію кусково-лінійною апроксимацією можна здійснити в один прохід. При одиничному значенні шістнадцятого розряду вхідного кута виконання ітерації триває лише три такти. Вимір швидкодії за допомогою таблиці ускладнений ще й тим, що вона включає ряд нелінійних елементів, таких, як інверсія знаку залишкового кута, якщо він менший за величину тетта, чи поправку щодо наступного кута, якщо вхідний кут Z в сукупності з Δx вийде за межі обчислень розрядної сітки. Також алгоритм містить команди ініціалізації регістрів, роботи з прапорцями та організацію вводу-виводу, яка може змінюватися залежно від потреб конкретної апаратної реалізації. Наприклад, при необхідності виводити значення всіх розрядів операнду, для чого знадобиться не менше трьох ітерацій. Чи, навпаки, без реалізації переривань можливим є зменшення часу виконання коду на чотири такти, а без виконання зупинки МК код скоротиться ще на два такти. Даний варіант також може бути використаний, наприклад, у випадку, якщо потрібно неперервно генерувати цифровий синусоїдальний сигнал, а підключене зовнішнє апаратне забезпечення здатне саме вибрати момент для отримання числа з потоку генерованих значень.

Підсумовуючи наведене вище, можна сказати, що одним із практичних способів для однозначного визначення середнього часу обчислення різних комбінацій вхідних даних є подання усіх допустимих аргументів функції на вхід МК, а отриману в результаті цього кількість тактів слід поділити на кількість обчислених вихідних значень реалізованої функції. На жаль, апаратні можливості МК не дозволяють провести такі заміри оперативно, а постановка такого експерименту для вибраної розрядності буде вимірюватись місяцями. Крім того, у

МК відсутні засоби моніторингу, наприклад, для визначення точної кількості виконаних тактів, на відміну від x86 платформи, на якій досліджувались характери різних методів. Це саме стосується і програмної симуляції роботи МК, яка у кілька разів повільніша за апаратну реалізацію. Тому на практиці отримуємо доволі парадоксальну ситуацію. На платформі з RISC ядром можна теоретично з достатньо високою точністю обробляти параметри алгоритму, але без можливості її практичної перевірки. На CISC платформі (x86) практична перевірка як швидкодії так і похибок не становить проблеми, а теоретичний розрахунок параметрів алгоритму практично неможливий через високу складність конвеєра, доповненого багаторівневими алгоритмами кешування та мультизадачністю. Тому будемо аналізувати швидкодію, відштовхуючись від мінімальних та максимальних показників алгоритму, обчислити та перевірити які цілком реально за допомогою симуляції.

### 4.3.10 Результати практичної реалізації алгоритму

Основний параметр, який оптимізовувався в першу чергу при написанні алгоритмів, – досягнення максимальної швидкодії. Через що алгоритм для МК написаний у вигляді “розгорнутого” коду, де всі можливі цикли та процедури замінені свого роду макрокомандами. Такий підхід вимагає наявності більшої кількості пам’яті для коду програми, але, без сумніву, має вищу швидкодію. Крім того, кожна така макроопераціяможе бути оптимізована для обчислення тих статичних параметрів функції, для яких вона буде виконуватись. В цьому випадку кожна така макроопераціяістить унікальну частину коду і є оптимізованою за часом виконання. Також альтернативним способом збільшення швидкодії та зменшення довжини коду є використання ТПВ більшого розміру. Але не слід забувати, що розмір таблиць подвоюється з кожним кроком, і їх максимальний розмір не може перевищувати половини пам’яті МК (одного кілобайту з двох доступних), що в представленому випадку дозволить упустити лише один крок алгоритму, для біту кута номер шість. Цей крок у представленому методі

виконується за 62 такти, тоді як при використанні процедур, яких повинно бути для усіх розрядів методу не менше десяти, лише їх виклики, без урахування передачі параметрів та налаштування регістрів будуть виконуватися протягом 70 тактів. З них три такти команда виклику rcall, та чотири такти команда повернення з процедури ret. Відповідно, якщо припустити, що код у розгорнутому вигляді є достатньо оптимізованим, то навряд чи існує спосіб покращення часових параметрів алгоритму шляхом модифікації його коду, а швидкодія реалізованого алгоритму є максимальною для вибраного МК.

Особливістю реалізованого алгоритму є й те, що він не використовує оперативної пам’яті МК та стеку, який розміщується в ОП, та EEPROM пам’яті (використання якої не рекомендоване виробником на максимальних частотах). Задіювати вищезгадані елементи МК не обов’язково, якщо всі необхідні дані можна розмістити в регістрах МК, тоді як звертання як до комірок пам’яті, так і до стеку буде втратою машинного часу. Також важливим є правильно розмістити параметри в регістрах таким чином, щоб мати до них доступ за допомогою доступних команд, які передбачені для роботи з відповідними регістрами. Адже, як було сказано при опису МК, не всі дані, які розміщені в регістрах, будуть доступні для частини команд. При цьому всі регістри є зайняті динамічними параметрами програми під час виконання основного циклу, що дає 100% використання регістрового файлу МК.

Тридцять два регістри МК розподілені наступним чином:

R0…R3 – значення числа π в двійковому представленні, для корекції кута обчислень при виході його за встановлені межі.

R4…R7 – Значення кута φ, для якого здійснюється обрахунок функції і початкове значення якого може бути задано МК ззовні.

R8…R11 – Значення параметру Δx, що відповідає значенню кута, на який необхідно здійснити поворот вектора кожної наступної ітерації.

R12…R19 – Пара значень синуса та косинуса, які в даний момент обчислюються основним циклом (глобальні значення).

R20…R27 – Пара тимчасових значень синуса та косинуса, які використовуються на кожному кроці алгоритму як тимчасові (локальні) значення.

R28…R29 — Значення залишкового кута, яке на початку алгоритму дорівнює молодшим шістнадцяти бітам вхідного кута φ і модифікується методом одностороннього чи інверсного повороту для використання даного значення при виконанні операції add-shift.

R30 – Адреса в таблиці синусів/косинусів старших біт кута для φ. При виконанні одностороннього повороту та add-shift методу у цей регістр записується нульове значення в службових цілях.

R31 – Старший байт адреси сторінка флеш пам’яті, з якої здійснюється вибірка початкового значення синуса та косинуса, і для п’ятикрокового методу завжди дорівнює семи.

Значення функцій розміщуються у регістрах парами, так що при необхідності їх можна перенести в тимчасові регістри за один такт командою копіювання пари сусідніх регістрів. Тоді як при послідовному розміщенні виникне надлишковість у використанні такої операції, оскільки не завжди потрібно оперувати усіма розрядами.

### 4.3.11 Швидкодія алгоритму

Як було зазначено вище, точно визначити швидкодію алгоритму при усіх можливих варіантах вхідних значень за допомогою засобів МК є завданням, яке важко реалізувати на практиці. Тому основою для показів результатів швидкодії стали значення середньої кількості тактів виконання алгоритму. Одержане таким чином значення означає, що результат є правильним у випадку рівномірного розподілення часу обрахунку всіх допустимих вхідних значень у цих межах. Звичайно, рівномірність навряд чи є досяжною, враховуючи несиметричність часу виконання різних гілок алгоритму. Але в цілому (і це підтверджують результати тестів на інших платформах) час виконання алгоритму лише незначною мірою

відхиляється від середнього значення. Також, як вже було сказано, наводяться результати швидкодії для методу CORDIC та для алгоритму генерації синусоїдального сигналу в цілому. Дані, отримані при дослідженні, показані на діаграмі рис 4.10.

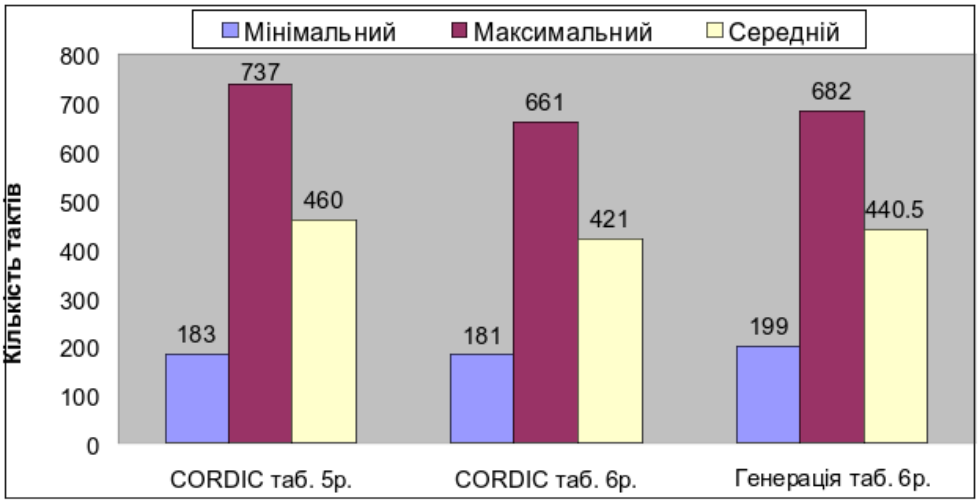



Рис. 4.12 Порівняння швидкодії модифікацій алгоритму за кількістю тактів

Маючи дані у вигляді середньої кількості тактів роботи алгоритму, можна визначити характеристики, які з цього випливають. Наприклад, знаючи тактову частоту функціонування МК, можна сказати, за скільки часу, в середньому, буде обрахований один кут. Виходячи з цих даних, можна визначити швидкість обчислення тригонометричних функцій за одиницю часу. Результати подані у вигляді діаграми на рис. 4.13.

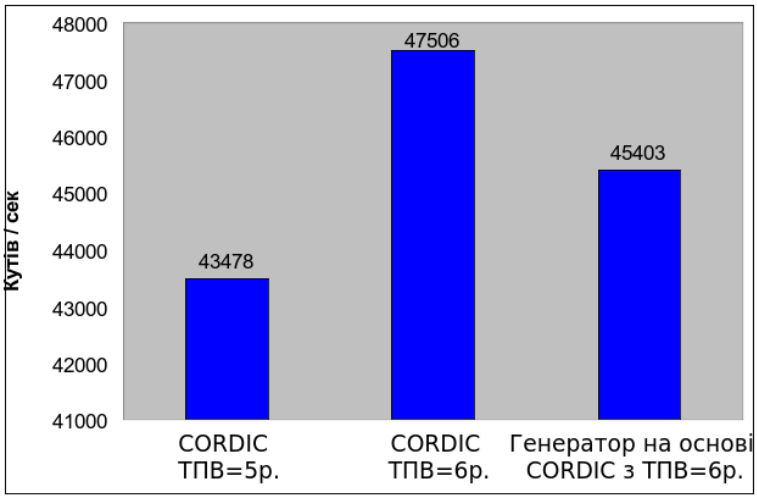



Рис. 4.13 Середня кількість аргументів, обчислених за секунду, та швидкість генерації ПВП на основі синусоїдальної функції

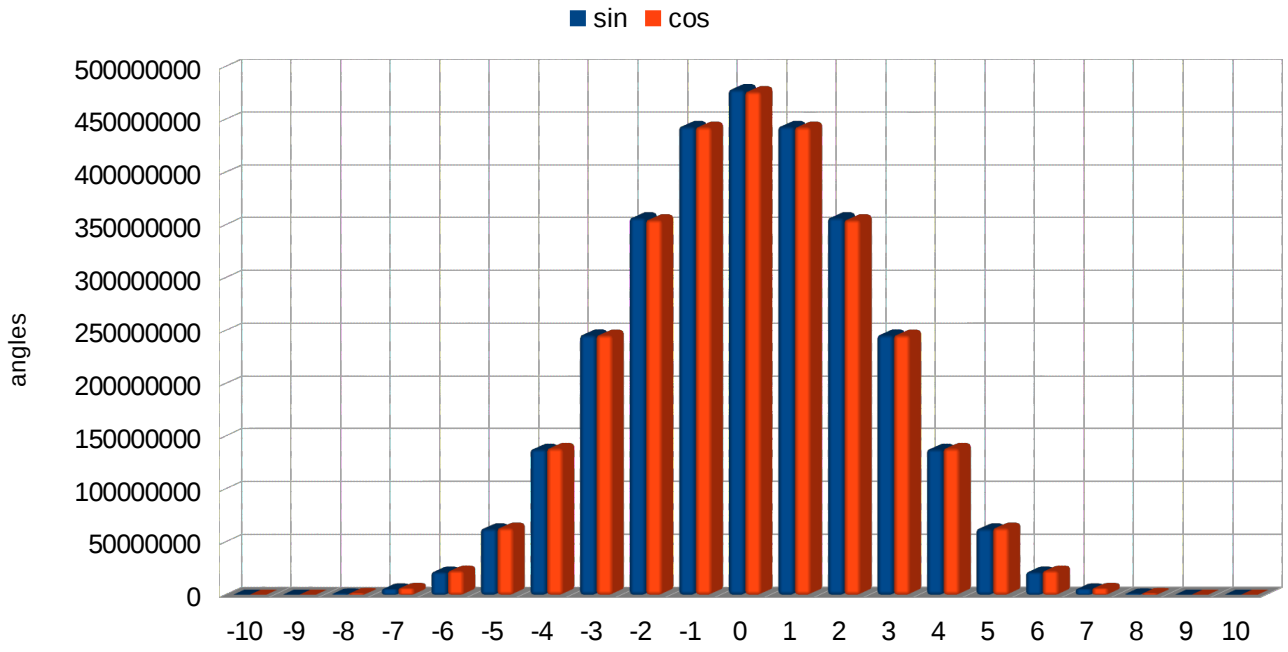



Рис. 4.14 Розподіл відхилення функцій синуса та косинуса від еталонного значення в одиницях молодшого розряду

## Висновки до четвертого розділу

1. Пропоновані методи реалізовано на платформі x86, використовуючи її цілочисельну логіку та широкий набір команд. Використовуючи високу швидкодію платформи одержано показники продуктивності методів для різних режимів їхнього функціонування.
2. За допомогою ресурсів копроцесора отримано розподіл вихідних похибок обчислень для всього спектру вхідних значень аргументів та запропоновано способи підвищення точності розрахунків.
3. Реалізовано обчислення тригонометричної функції синуса та косинуса засобами платформи з обмеженою функціональністю (AVR). Написаний код оптимізовано для досягнення високої швидкодії та найбільш повного використання ресурсів мікроконтролера.
4. На основі розроблених алгоритмів було реалізовано цифровий синусоїдальний генератор, який дозволяє здійснювати генерацію ПВП на

частотах, що відповідають стандарту IEC 60908, з можливістю підключення його до ЦАП із шириною шини до 32 розрядів.

5. Організовано комунікацію між МК та ПК, використовуючи протокол IEEE 1284 та режим byte mode для двосторонньої передачі даних, що дозволяє використовувати МК, як додатковий периферійний пристрій комп’ютера з перекладенням на нього частини обчислювальних задач.

# ВИСНОВКИ

У дисертаційній роботі на основі здійснених теоретичних та експериментальних досліджень розв’язано актуальне наукове завдання в області створення нових ефективних способів обчислення елементарних функцій. Основні теоретичні та експериментальні дослідження, представлені в роботі, можна узагальнити такими висновками:

1. Розроблено нові методи, технології та апаратно-програмні засоби, які в ході експериментальних досліджень підтвердили коректність постановки задач і математичних методів, які були використані при їх розв’язанні. Так, для платформ з великою кількістю помножувачів найкращим способом використання ресурсів буде метод ППКІ. Якщо ж їх кількість обмежена чи емулюється програмно, можна використовувати метод квадратичної апроксимації чи односторонній/інверсний поворот. Для економії ресурсів кристалу також корисним буде метод перекодування кута.
2. Здійснено комп’ютерне імітаційне моделювання функціонування розроблених методів обчислення елементарних функцій. Результати моделювання показали, що запропоновані методи дають змогу знизити латентність, ресурсоємкість та збільшити швидкодію. Також вдалось покращити точність обчислень, знизивши похибки методу CORDIC з 20 до 10-11 (залежно від наявності корекції) одиниць молодшого розряду.
3. Розроблено пристрій генерації псевдовипадкових чисел на основі тригонометричних функцій в кристалах із обмеженою функціональністю. Практично продемонстровано можливість використання такого підходу для бюджетних платформ без втрати якості в процесі їх експлуатації. Робочий прототип пристрою здатний генерувати до 2.9 мбіт/с корисної інформації при частоті у 20МГц.
4. Здійснено програмну реалізацію пропонованих методів як на процесорних архітектурах з високою розрядністю та широким набором команд, так і для

бюджетних мікроконтролерів, із урахуванням специфіки функціонування даних алгоритмів на кожному із них. Враховуючи, що апаратне забезпечення відіграє суттєву роль у функціонуванні програмної частини, найкраща за стабільністю платформа AthlonXP продемонструвала приріст швидкодії у 2.25 рази в порівнянні з класичним методом обчислення функцій.

5. Проведене оцінювання апаратних затрат при використанні різних сімейств ПЛІС для реалізації пропонованих методів. Також одержані параметри енергоефективності використання того чи іншого методу обчислень залежно від сфери його подальшого впровадження, основним параметром значення якого була латентність, де виграш становив до 65% - 55% від початкового значення (залежно від розрядності). В цей же час падіння тактової частоти не перевищувало 10% - 20%.
6. У результаті проведених досліджень, присвячених розв'язанню важливої науково-технічної проблеми в області створення нових ефективних алгоритмів обчислення тригонометричних, гіперболічних, експоненціальних та степеневих функцій, запропоновано удосконалення методу CORDIC як ефективного способу прискорення обчислень чи зменшення необхідних для цього апаратних затрат.
7. Розроблені методи і алгоритми успішно впроваджені у вигляді програмних та апаратних засобів, здатних обчислювати широкий спектр функцій із високою ефективністю та/або нижчим рівнем енергоспоживання. Впровадження пропонованих методів можливе як у високотехнологічних інтегральних схемах та процесорах, так і в бюджетних сімействах мікроконтролерів чи програмних продуктів.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Азаров О. Д., Дудник О. В. Методи та засоби високоточного слідкувального аналого-цифрового перетворення з ваговою надлишковістю. Вінниця : ВНТУ, 2014. 120 с.

2. Борецький Т., Мороз Л. Реалізація класичного та адаптивного CORDIC-методу на платформі IA32 : збірник наукових праць Української академії друкарства. Сер. Комп’ютерні технології друкарства. Львів, 2012. № 28. С. 123-32.

3. Боюн В. П. Динамическая теория информации: Основы и приложения. К. : Ин-т кибернетики им. В.М.Глушкова НАН Украины, 2001. 326 с.

4. Горпенюк А. Я. Дудикевич В.Б., ЛагунА. Е. Широкодіапазонний синусно-косинусний функціональний перетворювач. Вісник НУ “Львівська політехніка”. Сер. Радіоелектроніка та телекомунікації. Львів, 2000. №387. С.420-424.

5. Захаров А. В., Хачумов В. М. CORDIC-алгоритмы: Современное состояние и перспективы. Алгоритмы CORDIC. Переславль-Залесский, 2004. С. 353 - 372.

6. Зотов В. Ю. Проектирование цифровых устройств на основе ПЛИС фирмы XILINX в САПР WebPACK ISE. М. : Горячая линия - Телеком, 2003. 624 с.

7. Максимович В. М. Число-імпульсні функціональні перетворювачі з імпульсними зворотними зв’язками : автореф. дис. д-ра техн. наук. Львів, 2007. 33 с.

8. Мак-Кракен Д., Дорн У. Численные методы и программирование на фортране. Москва: Мир, 1977. 583 с.

9. Мейнарович Є. Кратко М. Англійсько-український словник з математики та інформатики. Київ, 2010.

10. Мельник А. О. Архітектура комп’ютера : підручник. Луцьк: Луцьке обласне видавництво, 2008. 470с.

11. Мельник А. А. Конвейерное устройство для вычисления логарифмической и экспоненциальной функции. Авторское свидетельство СССР № 662937. 1983.

12. Микитів Т. М., Мороз Л.В. Використання методу CORDIC у біометричних

системах доступу на основі аналізу відбитків пальців. *Захист інформації і безпека інформаційних систем* : матеріали 1-ої міжнародної науково-технічної конференції 31 травня – 01 червня, 2012р. Львів, С. 90-91.

13. Микитів Т. М., Мороз Л. В. Мікроконтролерний синусно-косинусний обчислювач підвищеної швидкодії. Вісник НУ “Львівська політехніка”. Сер. Автоматика, вимірювання та керування. Львів, 2011. № 695. С. 69-74.

14. Мороз Л., Голінко Ю. Апроксимація арктангенса : збірник наукових праць Української академії друкарства. Сер. Комп’ютерні технології друкарства. Львів, 2009. № 22. С. 36-45.

15. Мороз Л. В., Стахів М. Ю. Похибки двійкового число-імпульсного помножувача. Вісник НУ “Львівська політехніка”. Сер. Автоматика, вимірювання та керування. Львів, 2005. №530. С. 13-22.

16. Мороз Л. В. Адаптивний CORDIC–метод обчислення деяких функцій : збірник наукових праць Української академії друкарства. Сер. Комп’ютерні технології друкарства. Львів, 2010. № 24. С. 101-106.

17. Мороз Л.В., Стрілецький З.М., Стахів М.Ю. Аналіз шляхів покращення метрологічних характеристик число-імпульсних помножувачів : збірник наукових праць Української академії друкарства. Сер. Комп’ютерні технології друкарства. Львів, 2002. № 7. С. 175-180.

18. Мороз Л. В., Микитів Т. М. Використання методу CORDIC у біометричних системах доступу на основі аналізу відбитків пальців. Сучасний захист інформації: науково-технічних журнал, 2012. № 4. С.25-30.

19. Мороз Л. В. Микитів Т. М. Дослідження модифікацій синусно-косинусних CORDIC алгоритмів на мікроконтролері AVR: матеріали IV Міжнародної конференції молодих вчених CSE-2010. Львів, 2010. С. 218 – 219.

20. Мороз Л. В. Теорія та швидкодіючі апаратно-програмні засоби ітераційних методів обчислення функцій. автореф. дис. д.т.н. Львів, 2013.

21. Мороз Л. В. Ітераційні формули для CORDIC-методу: збірник наукових праць

Української академії друкарства. Сер. Комп’ютерні технології друкарства. Львів, 2012. № 28. С. 111-120.

22. Мороз Л. В., Борецький Т. Р., Сколоздра М. М. Удосконалення методу CORDIC для обчислення тригонометричних функцій засобами програмованої логічної інтегральної схеми. Науковий вісник НЛТУ України. Львів, 2015. №5. С. 292-301.

23. Мороз Л. В. Перекодування кута CORDIC-методу: збірник наукових праць Української академії друкарства. Сер. Комп’ютерні технології друкарства. Львів, 2015. № 33. С. 51-55.

24. Мороз Л. В., Борецький Т.Р., Луковський Т.Л. Модифікований CORDIC-методу обчислення синуса-косинуса : збірник наукових праць Української академії друкарства. Сер. Комп’ютерні технології друкарства. Львів, 2015. № 33. С. 56-63.

25. Мороз Л. В., Грабовський Я.І., Микитів Т.М., Борецький Т.Р., Костів Ю.М., Войтусік С.С. Швидкодіючий гібридний CORDIC-обчислювач тригонометричних функцій. Науковий вісник НЛТУ України. 2014. Вип. 24.8. С. 352-357.

26. Мороз Л. В. Теорія та швидкодіючі апаратно-програмні засоби ітераційних методів обчислення функцій : автореф. дис. д.т.н. Львів, 2013.

27. Мэтьюз Джон Г., Куртис Д. Финк. Numerical Methods: Using MATLAB", Third Edition. Москва: Вильямс, 2001. 720 с.

28. Осипов Jl. А. Обработка сигналов на цифровых процессорах. Линейно-аппроксимирующий метод. М.: Горячая линия - Телеком, 2001. 112 с.

29. Палагин А. В. Реконфигурируемые вычислительные системы: Основы и приложения. К.: Просвіта, 2006. 280 с.

30. Руфицкий М. В., Слик А., Филиппов А. К. Адаптивное дискретное косинусное преобразование: *Перспективные технологии в средствах передачи информации* : материалы пятой международной научно-технической конференции Владимир: Связьоценка, 2003. С. 214-216.

31. Сизов В. П. Алгоритм защиты информации на основе тригонометрических функцій. Автореферат десетрації – Уфа 2012

32. Соловьев В. В. Проектирование цифровых систем на основе программируемых логических интегральных схем. М.: Горячая линия — Телеком, 2001. 636 с.

33. Стешенко В. Б. ПЛИС фирмы «Altera»: элементная база, система проектирования и языки описания аппаратуры. М.: Издательский дом «Додэка-XX1», 2002. 576 с.

34. Филиппов А. К. Базовые принципы построения комплементарных вычислительных устройств *Электроника, информатика и управление:* Сборник научных трудов преподавателей, сотрудников и аспирантов. Вып. 5. Владимир: ВлГУ, 2004. С. 4-8.

35. Филиппов А. К. Выбор метода вычисления функций для аппаратной реализации в ЭВС. Проектирование и технология электронных средств. Владимир: ВлГУ, 2004. №1. С. 31-35.

36. Филиппов А. К. Исследование возможности применения способа непосредственной проверки сходимости для расчета значений прямых тригонометрических функций. Проектирование и технология электронных средств. Владимир: ВлГУ, 2004. №2. С. 50-55.

37. Филиппов А. К. Построение поведенческой модели устройства вычисления функций синус и косинус. Электроника, информатика и управление: сборник научных трудов преподавателей, сотрудников и аспирантов. Вып. 3 Владимир: ВлГУ, 2002. С. 70-74.

38. Филиппов А. К. Уменьшение интервала изменения аргумента для прямых тригонометрических функций. Электроника, информатика и управление: сборник научных трудов преподавателей, сотрудников, аспирантов. Вып. 3. Владимир: ВлГУ, 2003. С. 94-100.

39. Чиркунова Ж. В. Пространственная обработка сигналов в цифровых антенных решетках: диссертация на соискание степени кандидата технических наук. Ж. В. Чиркунова. Москва: МИЭТ, 2009.

40. Ярушевський О. О. Швидкі алгоритми обчислювальних криптопримітивів. Master's Thesis. Київ, 2018.

41. Aggarwal, Supriya, Pramod K. Meher, and Kavita Khare. "Concept, design, and implementation of reconfigurable CORDIC." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 24.4 (2016): 1588-1592.

42. Aggarwal S. Area-Time Efficient Scaling-Free CORDIC Using Generalized Micro-Rotation Selection. / Supriya Aggarwal, Pramod K. Meher, K. Khare // IEEE transactions on very large scale integration (vlsi) sistems. – vol. 20, – №.8, – August 2012, – pp. 1542– 1546.

43. Aggarwal S. Design techniques targeting low-area-power-delay product in hyperbolic CORDIC algorithm. / S. Aggarwal, K. Khare // The Computer Journal. – vol. 55. - № 5, - 2012, - pp. 616 – 628.

44. Aggarwal S. FPGA implementation of area–time efficient CORDIC processor dedicated to compute exponentials in neural networks. / S. Aggarwal, K. Khare // Int. J. Signal and Imaging Systems Engineering. – vol. 3, – № 4, – 2010, – pp. 255 – 260.

45. Aggarwal S. Hardware efficient architecture for generating sine/cosine waves. / S. Aggarwal, K. Khare // 25th International Conference on VLSI Design.-2012,-pp.57- 61.

46. Aggarwal S. Leading one detection hyperbolic CORDIC with enhanced range of convergence. / S. Aggarwal, K. Khare // J Sign Process Syst. – vol. 70, – №1, – 2013, pp. 49 – 57.

47. Aggarwal S. Redesigned-Scale-Free CORDIC Algorithm Based FPGA Implementation of Window Functions to Minimize Area and Latency. / S. Aggarwal, K. Khare // International Journal of Reconfigurable Computing Volume 2012, Article ID 185784. – 2012, – pp. 1–8.

48. Aggarwal S. Scale-free hyperbolic CORDIC processor and its application to waveform generation. / S. Aggarwal, P. K. Meher, S. Member, K. Khare // IEEE Transactions on circuits and systems – vol. 60, – № 2, February 2013, – pp. 314 – 326.

49. Antelo E. High-radix CORDIC rotation based on selection by rounding. / E. Antelo,

T. Lang, J. D. Bruguera // Journal of VLSI Signal Processing Systems, – v. 25, – №2, – 2000, – pp. 141-153.

50. AMD 64 Architecture Programmer’s Manual Volume 5: 64-Bit Media and x87 Floating-Point Instructions, - May 2018.

51. Antelo E. Low latency pipelined circular CORDIC. / E. Antelo, J. Villalba // IEEE Symposium on Computer Arithmetic (ARITH-17). – June, 2005, – pp. 280 – 287.

52. Antelo E. Very-High Radix Circular CORDIC Vectoring and Unified Rotation/Vectoring. / E. Antelo, T. Lang, J. D. Bruguera // IEEE Transactions on Computers, – v. 49, – № 7, – 2000, – pp. 727-739.

53. Aoki T. Radix-2-4-8 CORDIC for Fast Vector Rotation. / T. Aoki, I. Kitaori, T. Higuchi // IEEE Trans. Fundamentals, – v. E83-A, – №6, – 2000, – pp. 1106-1114.

54. Archana, C. B., and C. Nagaraju. "Efficient window architecture design using completely scaling free cordic pipeline" International Journal For Technological Research In Engineering, - May 2014.

55. Arndt J. Matters computational.Ideas,algorithms,source code./J.Arndt.-2010,-p. 966.

56. Butler J. T. Numeric Function Generators./ J. T. Butler, T. Sasao, S. Nagayama//. Report OMB No. 0704-0188. Naval Postgraduate School, Department of Electrical and Computer Engineering, Monterey, CA, 93943, July 2009.

57. Chen C.-Y. High-resolution architecture for CORDIC algorithm realization. / C.-Y. Chen, C.-Y. Lin // IEEE Xplore. Restrictions apply. – 2006, – pp. 579 – 582.

58. Cho J. U. A Motion-Control Chip to Generate Velocity Profiles of Desired Characteristics. / J. U. Cho, J. W. Jeon. – ETRI journal, – vol. 27, – № 5, October, – 2005, – pp. 563-568.

59. Cornea, M. Scientific Computing on Itaniumbased Systems. / Marius Cornea, John Harrison, and Ping Tak Peter Tang. // Intel Press, 2002.

60. Desai, Bandenamaj H., and Mahesh B. Neelagar. "Implementation of Fast CORDIC Algorithm for Embedded Application." (2017).

61. Detrey, J. Floating-point trigonometric functions for FPGAs. In International Conference on Field Programmable Logic and Applications/ Jérémie Detrey and Florent de Dinechin // Amsterdam, Netherlands, - Aug 2007. IEEE. doi: 10.1109/FPL.2007. 4380621.pp 29-34.

62. Dhume N. Parameterizable CORDIC-based floating-point library operations. / N. Dhume and R. Srinivasakannan // Application Note: Spartan-6, Virtex-6, 7 Series, and Zynq-7000 Devices. XAPP552. – vol. 1, – June 1, 2012, – pp. 1 – 18.

63. Dinechin F. Istoan M. Sergent G. “Fixed-point trigonometric functions on FPGAs,” in Highly-Efficient Accelerators and Reconfigurable Technologies, - Mar. 2013.

64. Ercegovac, M.D. Digital Arithmetic / M. D. Ercegovac and T. Lang // Kluwer Academic Publishers, 2004.

65. Ercegovac, M.D. Improving goldschimdt division, square root and square root reciprocal / Ercegovac, M.D, L. Imbert, D. Matula, J. M. Muller, and G.Wei // IEEE Trans.Computers, 49(7):759–763, 2000.

66. Export Administration Regulations Supplement No. 1 to Part 740 - Country Groups Aug 2018 [Electronic resource]. - Mode of access https://www.bis.doc.gov/index.php/

documents/regulation-docs/2255-supplement-no-1-to-part-740-country-groups-1/file

67. Francisco Aguirre-Ramos, Alicia Morales-Reyes, Rene Cumplido, Claudia Feregrino-Uribe. An Area Efficient Composed CORDIC Architecture. Advances in Electrical and Computer Engineering,Volume 14, Number 2, 2014, pp. 113–116.

68. Gustafsson O. Addition Aware Quantization for Low Complexity and High Precision Constant Multiplication. / O. Gustafsson, F. Qureshi // IEEE Signal Processing. – vol. 17, – № 2, – 2010, – pp. 173–176.

69. Han DS, Juan RO, Jung MW, Cha HW, Kim HS. Development of a Novel Fast Rotation Angle Detection Algorithm using a Quasi-Rotation Invariant Feature Based on Sobel Edge. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), - Jun 2017.

70. Hertz, Erik, Nilsson. "A methodology for parabolic synthesis of unary functions for

hardware implementation." 2008 2nd International Conference on Signals, Circuits and Systems. IEEE, 2008.

71. Holimath, V. Division and Square root for Mobile and Scientific computing markets. PhD thesis / V. Holimath // PhD thesis. October 2007.-128p.

72. Hauck S. Reconfigurable Computing. The Theory and Practice. / S. Hauck and A. DeHon // Elsevier Inc. – 2008, – p. 908.

73. Haviland G.L. A Cordic arithmetic processor chip. / G. L. Haviland, A. A. Tuszynski // IEEE Journal of Solid-state Circuits, – vol.SC-15, – N 1, – 1980, – p.4-15.

74. Higham N. J. Accuracy and Stability of Numerical Algorithms./ Higham N. J.// SIAM, Philadelphia, PA, 2nd edition, 2002. ISBN 0-89871-521-0.

75. IEEE 1284: Parallel Ports // Lava Computer MFG Inc. - May 2002

76. Intel itanium architecture software developer’s manual // 1:191–203, October 2002.

77. Intel Corporation 8087 Support Library Reference Manual, - Aug. 1981.

78. Intel Programmable Interrupt Controller 8259A, 1988.

79. ISO/IEC 9899: 1999 Standard Programming Language C, – December 1999.

80. Itanium 2 processor reference manual for software development and optimization, - May 2004.

81. Jaime F. J. Enhanced Scaling-Free CORDIC. / F. J. Jaime, M. A. Sánchez, J. Hormigo, J. Villalba, E. L. Zapata // IEEE Transactions on circuits and systems – vol. 57, – № 7, – July, 2010, – pp.1654 – 1662.

82. Jonathan Y. Stein. Function evaluation algorithms. / Jonathan Y. Stein // Digital Signal Processing: A Computer Science Perspective. – 2000, – pp. 605 – 618.

83. Juang Y. -S. Optimization and Implementation of Scaling-Free CORDIC-Based Direct Digital Frequency Synthesizer for Body Care Area Network Systems. / Y.–S. Juang, L.– T. Ko, J.-E. Chen, T.-Y. Sung, and H.-C. Hsin // Hindawi Publishing Corporation Computational and Mathematical Methods in Medicine. – 2012, – pp. 1–9.

84. Juang T.-B. Hsiao S.-F., and M.-Y. Tsai, “Para-CORDIC: Parallel CORDIC Rotation

Algorithm,” IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 51, no. 8, pp. 1515–1524, Aug. 2004.

85. Juang T, “Low Latency Angle Recoding Methods for the Higher Bit-Width Parallel CORDIC Rotator Implementations,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 55, no. 11, pp. 1139–1143, Nov. 2008.

86. Koren I. Computer arithmetic algorithms. Second edition. / I. Koren. – C.: A K Peters, – 2002, – p. 281.

87. Kornerup P. Choosing starting values for certain newton-raphson interations. / P. Kornerup and J.-M.Muller//Theoretical Computer Science, 351:101-110,February 2006.

88. Kuhlmann M. and K. K. Parhi, “P-CORDIC : A Precomputation Based Rotation,” EURASIP Journal on Applied Signal Processing, vol. 2002, no. 1, pp. 936–943, 2002. [Electronic resource]. - Mode of access http://dx.doi.org/10.1155/S1110865702205028.

89. Lachowicz S. Fast evaluation of nonlinear functions using FPGAs. / S. Lachowicz, H.-J. Pfleiderer // Advances in Radio Science 6. – 2008, – pp. 233–237.

90. Lachowicz S. Fast evaluation of nonlinear functions using FPGAs. / S. Lachowicz, H.-J. Pfleiderer // Advances in Radio Science 6. – 2008, – pp. 233 – 237.

91. Lee, In-Hee. An Implemetation of a 32 bit fixed-poin CORDIC Transcendental Function Calculator Using FPGA/ In-Hee Lee, Min-Ho Kim, Hyun-Phil Kim, Ha-Young Jeong, Yong-Surk Lee // Processor Laboratory, Yonsei University, Seoul, Korea, 2006.

92. Lee J. - S. Implementing and optimizing a direct digital frequency synthesizer (DDFS) on FPGA. / J. - S. Lee, X. Y. J. - S. Lee // May 10, 2006, – pp. 1010 – 1015.

93. Liddicoat A. A. High-Performance Arithmetic for Division and the Elementary Functions. / A. A. Liddicoat // A dissertation for the degree of Doctor of Philosophy. Stanford University, – 2002. – p. 141.

94. Liedel M. Secure distributed computation of the square root and applications. / M. Liedel // Electronic Design, Test and Applications, Proceedings jf IEEE 5th International Symposium on Electronic Design, Test and Applications, Ho Chi Minh

City, Vietnam, 13 January 2010, pp. 42-52.

95. Liu J. DCD algorithm: architectures, FPGA. Implementations and applications. / J. Liu. – 2008, – p. 159.

96. Llamocca-Obregón, Daniel R., and Carla P. Agurto-Ríos. "A fixed-point implementation of the expanded hyperbolic CORDIC algorithm." Latin American applied research 37.1 (2007): 83-91.

97. LogiCore IP CORDIC v6.0, Dec 2017. [Electronic resource]. – Mode of access http://www.xilinx.com/support/documentation/ip_documentation/cordic/v6_0/pg105-cordic.pdf

98. Madisetti A, A.Y.Kwentus, A.N.Willson. A 100 MHz, 16-b, direct digital frequency synthesier with 100-dBc supurious-free dynamic range. IEEE Journal of Solid-State Circuits. – vol.34, – № 8, – 1999, – pp. 1034 – 1043.

99. Maharatna K. Modified virtually scaling-free adaptive CORDIC rotator algorithm and architecture. / K. Maharatna, S. Banerjee, E. Grass, M. Krstic, A. Troya // IEEE Trans. Circuits Syst. Video Technol. – vol. 11, – November, 2005, – pp. 1463 – 1474.

100. Maharatna K. Virtually scaling free adaptive CORDIC rotator. / K. Maharatna., A. Troya, S. Banerjee, E. Grass // IEEE Proc. Comp. Dig. Tech. – vol. 151, – 2004, – pp. 448 – 456.

101. Marino M. Implementazione di funzioni trigonometriche su microcontrollori a 8 bit mediante CORDIC. / M. Marino - 2009.

102. Markstein, P. IA-64 and Elementary Functions: Speed and Precision. Hewlett-Packard Professional Books/ Markstein, P.// Prentice Hall, 2000.

103. Meher P. K. 50 years of CORDIC: algorithms, architectures, and applications. / P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, K. Maharatna // IEEE Transactions on Circuits and Systems—I: Regular Papers. – vol. 56, – № 9, – September, 2009, – pp. 1893 – 1907.

104. Moroz Leonid, Shinobu Nagayama, Taras Mykytiv, Ihor Kirenko, Taras Boretskyy / “Simple Hybrid Scaling-Free CORDIC Solution for FPGAs ” //

International Journal of Reconfigurable Computing, vol. 2014, Article ID 615472, 4 pages, 2014. [Electronic resource]. - Mode of access http://dx.doi.org/10.1155/2014/615472.

105. Moroz L. Study of Modifications of Sine-Cosine CORDIC Algorithms on AVR Microcontroller / Leonid Moroz, Taras Mykytiv // Proceedings of the IV International Conference of Young Scientists CSE-2010, Computer Science and Engineering – Lviv, November 25-27, 2010 – pp. 218-219.

106. Moroz L. Improved Scaling-Free CORDIC algorithm / Leonid Moroz, Taras Mykytiv, Martyn Herasym. // Proceding of IEEE East-West Design & Test Symposiym (EWDTS'2012) – Kharkov, September 14-17, 2012 – pp. 470-474.

107. Moroz, L. Fast Arctangent CORDIC method / Leonid Moroz, Ihor Kirenko, Volodymyr Maksymovych // Proceedings of the 4-th International Conference ACSN-2009, Advanced Computer System and Networks: Design and Application Conference – Lviv, November 9-11, 2009 – p. 42.

108. Moroz L. Improved scaling-free CORDIC algorithm. / L. Moroz, T. Mykytiv, M. Herasym // Proceding of IEEE East-West Design & Test Symposiym (EWDTS'2012).

109. Muller J.-M. Elementary functions. Algorithms and implementation. Second edition. / J.-M. Muller. – USA: Birkhäuser, – 2006, – p. 265.

110. Muller J.-M. Handbook of floating-point arithmetic. / J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod,V. Lef`evre, G. Melquiond, N. Revol, D. Stehlr̈e, S. Torres. - USA: Birkhäuser, – 2010, – p. 272.

111. Munuoz, D. M., et al. FPGA based floating-point library for CORDIC algorithms. // In: Programmable Logic Conference (SPL), 2010 VI Southern. IEEE, 2010, - pp. 55-60.

112. Munoz D. M. Tradeoff of FPGA design of a floating-point library for arithmetic operators. / D. M. Munoz, D. F. Sanchez1, C. H. Llanos1 and M. Ayala-Rincyn // Journal Integrated Circuits and Systems. – 2010, – pp. 42 – 52.

113. Obregon D.R.L. A fixed-point implementation of the expanded hyperbolic

CORDIC algorithm. / D.R.L. Obregon, C.P. Agurto-Ríos // Latin Am. Appl. Res. – vol. 37, – 2007, – pp. 83 – 92.

114. Paska B. High-performance floating-point computing on reconfigurable circuits / Paska B.// PhD thesis. Septembre 2011.-180p.

115. Pierick H. T. Generating a non-linear waveform. / H. T. Pierick, C. J. Van Valburg. – US Patent, – № 6 489 735, – Dec. 2002.

116. Pottathuparambil R. Implementation of a CORDIC based double-precision exponential core on an FPGA. / R. Pottathuparambil and R. Sass // Proceedings of RSSI. – 2008, – pp. 1 – 4.

117. Pouyan, Peyman, Hertz, Nilsson. "A VLSI implementation of logarithmic and exponential functions using a novel parabolic synthesis methodology compared to the CORDIC algorithm." 2011 20th European Conference on Circuit Theory and Design (ECCTD). IEEE, 2011.

118. Proceedings of 9th International Symposium on Integrated Circuits, Devices and Systems, Singapore. – 3-5, – September, – 2001, – pp. 480-483.

119. Ramis Jean-Pierre Mathématiques Tout-en-un pour la Licence 1 - 3e éd. /, André Warusfel, Xavier Buff, Josselin Garnier, Emmanuel Halberstadt, Thomas Lachand-Robert, François Moulin, Jacques Sauloy 1024p. - 2018. - pp. 532.

120. Renardy, Antonius P., Nur Ahmadi, Ashbir A. Fadila, Naufal Shidqi, and Trio Adiono. "FPGA implementation of CORDIC algorithms for sine and cosine generator." In 2015 International Conference on Electrical Engineering and Informatics (ICEEI), pp. 1-6. IEEE, 2015.

121. Rudagi J. M. Performance Analysis of Radix 4 CORDIC Processor in Rotation mode with Parallel Scale factor Computation. / J. M. Rudagi, Srikant, Basavaraj B. Patil, Dr. S. Subbaraman // International Journal of Emerging Technology and Advanced Engineering. – vol. 2, – № 7, – July 2012, – pp. 507 – 510.

122. Sanchez D. Parameterizable floating-point library for arithmetic operations in FPGAs in in Press Proc. / D. Sarnchez, D. Munoz, C. Llanos, and M. Ayala-Rincorn //

ACM Inter. Symp. on Int. Circuits and Systems Design. – 2009.

123. Shoab, A. K. Digital design of signal processing systems: A practical approach (1st ed). New York, NY: John Wiley, 2011.

124. Sumanasena M. G. B. A scale factor correction scheme for the CORDIC algorithm. / M. G. B. Sumanasena // IEEE Trans. Comput. – vol. 57, – 2008, – pp. 1148 – 1152.

125. Timmermann D. Hahn H. Hostika B. Modified CORDIC algorithm with reduced iterations.Electronics Letters, 25(15):950–951, 1989.

126. Titare, P. V., Talmale, G. R. Design of An Adaptive Neuro-Fuzzy Inference System for Floating Point Function Generation using CORDIC Algorithm // International Journal of Science and Research (IJSR), - May 2015.

127. Thu Nguyen Thi Hong. Research on COordinate Rotation DIgital Computer Hardware Architectures and Applications, 2018.

128. Tumbush G. Dr. Signed Arithmetic in Verilog 2001 – Opportunities and Hazards // Starkey Labs, Colorado Springs CO.

129. Turner C. S. Recursive discrete-time sinusoidal oscillators. / C. S. Turner // IEEE Signal Processing Magazine. – vol. 20, – № 3, – May, 2003, – pp. 103 – 111.

130. Vazquez A. Computation of decimal transcendental functions using the CORDIC algorithm. / A. Vazquez, J. Villalba and E. Antelo // 19th IEEE Symposium on Computer Arithmetic Portland (USA). – June 8 – 10, 2009, – pp. 205 – 211.

131. Volder J. E. “The CORDIC Trigonometric Computing Technique,” IEEE Transactions on Electronic Computers, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

132. Walther J. S. "A unified algorithm for elementary functions", in Proc. AFIPS Conf., vol. 38, 1971, pp. 385-389.

133. Wang X. Variable precision floating-point divide and square root for efficient FPGA implementation of image and signal processing algorithms. / X. Wang // Ph.D. dissertation, Northeastern University. – 2007.

134. Yang B. Complex division and square-root using CORDIC. / B. Yang, D. Wang, L.

Liu // National Natural Science Foundation of China (NSFC). – № 61106022, – 2012, – pp. 2464 – 2468.

135. Yates, Randy. "Fixed-point arithmetic: An introduction." *Digital Signal Labs* 81.83:198 - 2009.

136. Zhang R. Low power CORDIC IP core implementation. / R. Zhang, J.-H. Han, A. T. Erdogan, T. Arslan // IEEE Edinburgh, EH9 3JL. – 2006, – pp. 956 – 959.

# ДОДАТКИ

Список праць опублікованих за темою дисертаціїї

*Наукові праці, в яких опубліковано основні результати дисертації*

1. Борецький Т., Мороз Л. Реалізація класичного та адаптивного CORDIC-методу на платформі IA32 : збірник наукових праць Української академії друкарства. Сер. Комп'ютерні технології друкарства. Львів, 2012. № 28. С. 130–140.

2. Борецький Т. Модифікований CORDIC-метод обчислення синуса-косинуса / Л. Мороз, Т. Борецький, Т. Луковський, С. Войтусік : збірник наукових праць Української академії друкарства. Сер. Комп'ютерні технології друкарства. Львів, 2015. № 33. С. 56–63.

3. Борецький Т. Швидкодіючий гібридний CORDIC-обчислювач тригонометрич-них функцій / Л. В. Мороз, Я. І. Грабовський, Т. М. Микитів, Т. Р. Борецький, Ю. М. Костів, С. С. Войтусік. *Науковий вісник НЛТУ України* : зб. наук.-техн. пр. Львів, 2014. Вип. 24.8. С. 352–358.

4. Борецький Т. Удосконалення методу CORDIC для обчислення тригонометричних функцій засобами програмованої логічної інтегральної схеми / Л. В. Мороз, Т. Р. Борецький, М. М. Сколоздра. *Науковий вісник НЛТУ України* : зб. наук.-техн. пр. Львів, 2015. Вип. 25.5. С. 292–301.

5. Борецький Т. Синус-косинусний FPGA-обчислювач на основі CORDIC-методу з перекодуванням кута / Л. В. Мороз, Т. Р. Борецький, Ю. М. Костів Науковий вісник НЛТУ України : зб.наук.-техн. пр. Львів, 2015. Вип. 25.6. С. 288–297.

6. Boretskyy T. Simple hybrid scaling-free CORDIC solution for FPGAs / L. Moroz, S. Nagayama, T. Mykytiv, I. Kirenko, T. Boretskyy // International Journal of Reconfigurable Computing. - 2014. Vol. - 2014. - pp. 1–4.

*Наукові праці, які засвідчують апробацію матеріалів дисертації:*

7. Борецький Т., Мороз Л. Генерація випадкових чисел засобами ПЛІС. *Захист інформації і безпека інформаційних систем* : матеріали V Міжнародної науково-технічної конференції. Львів, 2016.

8. Борецький Т., Микитів Т., Мороз Л. Покращення характеристик генератора псевдовипадкових послідовностей на основі тригонометричних функцій. *Інформаційна безпека в сучасному суспільстві* : матеріали I Міжнародної науково-технічної конференції. Львів, 2014. С.17-19.

9. Борецький Т., Мороз Л. Альтернативні методи обчислення тригонометричних функцій : матеріали 69-тої студентської науково-технічнрої конференції секції кафедр «Захист інформації» та «Безпека інформаційних технологій». Львів, 2011. С. 179-180.

ЕКОТЕСТ®
Засоби радіаційного контролю

07.02.2019 р.

АКТ

про використання результатів дисертаційної роботи
Борецького Тараса Романовича
«*Розробка та реалізація методів обчислення елементарних функцій на основі програмних та апаратних засобів*», представленої на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.13.05 – комп'ютерні системи та компоненти.

Приватне підприємство «НВПП «Спаринг-Віст Центр» (м. Львів) у відповідності до угоди про співпрацю між підприємством та кафедрою безпеки інформаційних технологій Національного університету «Львівська політехніка» у 2014-2016 роках отримало документацію і прикладні програми, автором яких є асистент кафедри безпеки інформаційних технологій Борецький Тарас Романович. Розробка стосується вирішення важливих для підприємства задач – створення швидкодіючих апаратно-програмних засобів опрацювання імпульсних і цифрових сигналів, що можуть використовуватись, зокрема, при розробці пристроїв для вимірювання параметрів іонізуючого випромінювання і засобів, що використовуються при налагодженні та перевірці даних пристроїв. Теоретичні розробки були використані при створені імітаційних моделей вхідних сигналів дозиметричних детекторів на основі генераторів псевдо-випадкових чисел і пуассонівських імітаційних послідовностей. Це дозволило забезпечити задані метрологічні характеристики дозиметричних пристроїв в широкому динамічному діапазоні вимірюваних величин. Основними перевагами пристроїв, що реалізовані з використанням наукових досліджень Борецького Тараса Романовича, є підвищення точності і розширення діапазону вимірювання поверхневої густини потоку та інтенсивності іонізуючих випромінень з опрацюванням вихідних сигналів дозиметричних детекторів в реальному часі.

Директор-генеральний конструктор
ПП «НВПП «Спаринг-Віст Центр»
лауреат державної премії України
в галузі науки і техніки, к.т.н. Ю. Б. Сторонський

ПП „НВПП „Спаринг-Віст Центр"
Україна, 79026, м. Львів, вул. Володимира Великого, 33
тел.: (032) 2421515, факс: (032) 2422015
e-mail: market@ecotest.ua, www.ecotest.ua

**ЗАТВЕРДЖУЮ**
Проректор з науково-педагогічної роботи
Національного університету
«Львівська політехніка»
_____Давидчак О.Р.
«____»_____2018 р.

**А К Т**
про впровадження результатів дисертаційної роботи
асистента кафедри безпеки інформаційних технологій
Борецького Тараса Романовича
«Розробка та реалізація методів обчислення елементарних функцій
на основі програмних та апаратних засобів»,
представленої на здобуття наукового ступеня *кандидата технічних наук*
за спеціальністю 05.13.05 – комп'ютерні системи та компоненти.

Комісія Національного університету «Львівська політехніка» у складі:
голова комісії – директор ІКТА, д.т.н., проф. Микийчук М.М.,
члени комісії: зав. кафедри безпеки інформаційних технологій, д.т.н., проф. Максимович В.М., зав. кафедри захисту інформації, д.т.н., проф. Дудикевич В.Б.

Даним підтверджуємо, що дослідження, проведені Борецьким Т.Р., виконувалися на кафедрі безпеки інформаційних технологій у рамках пріоритетних напрямів розвитку науки і техніки України, зокрема, при виконанні науково - дослідних робіт у рамках наукової теми «Розробка та вдосконалення ітераційних методів обчислення елементарних функцій для систем захисту інформації (номер державної реєстрації №0110U004687).

Отримані автором наукові результати, зокрема, використовуються:
- при проектуванні генераторів псевдовипадкових чисел;
- при розробленні криптографічних протоколів;
- при проведені науково-дослідних робіт та у навчальному процесі
- при вивченні курсу «Комп'ютерні методи високорівневого проектування пристроїв захисту» на кафедрі безпеки інформаційних технологій Національного університету «Львівська політехніка».

Голова комісії,
директор ІКТА, д.т.н., проф. _____Микийчук М.М.

Члени комісії:
зав. кафедри БІТ, д.т.н., проф. _____Максимович В.М.

зав. кафедри ЗІ, д.т.н., проф. _____Дудикевич В.Б.

**ЗАТВЕРДЖУЮ**
Проректор з наукової роботи
Національного університету
«Львівська політехніка»
проф. _____ Чухрай Н.І.
«__» _____ 2018 р.

**А К Т**
про використання результатів дисертаційної роботи
Борецького Тараса Романовича
«Розробка та реалізація методів обчислення елементарних функцій на основі програмних та апаратних засобів», представленої на здобуття наукового ступеня кандидата технічних наук, які виконувались в інституті ІКТА Національного університету "Львівська політехніка"

Комісія у складі голови — начальника науково-дослідної частини, к.т.н., доц. Жук Л. В., та членів: завідувача кафедри безпеки інформаційних технологій, д.т.н., проф. Максимовича В. М., завідувача відділу науково-організаційного супроводу наукових досліджень, к.т.н., Лазько Г. В., та заступника начальника планово-фінансового відділу, Чулой Т. М. підтверджують, що результати дисертаційної роботи Борецького Тараса Романовича на тему «Розробка та реалізація методів обчислення елементарних функцій на основі програмних та апаратних засобів» використовувались у науково-дослідній роботі кафедри безпеки інформаційних технологій на тему "Розробка та вдосконалення ітераційних методів обчислення елементарних функцій для систем захисту інформації" (номер державної реєстрації №0110U004687). Зокрема, Борецьким Т.Р. розроблено генератор псевдовипадкових чисел на основі тригонометричних функцій у вигляді периферійного комп'ютерного пристрою з використанням мікроконтролера AVR та апаратних обчислювачів на основі ПЛІС.

Голова комісії:

Начальник НДЧ, к.т.н., доц. _____ Л. В. Жук

Члени комісії:

Зав. кафедри БІТ, д.т.н., проф. _____ В. М. Максимович

Зав. відділу НОСНД, к.т.н.. _____ Г. В. Лазько

Заст. нач. відділу ПФВ _____ Т. М. Чулой