УДК 519.7
© Т. Борецький[1], Л. Мороз[1], 2012

РЕАЛІЗАЦІЯ КЛАСИЧНОГО ТА АДАПТИВНОГО CORDIC-МЕТОДІВ НА ПЛАТФОРМІ IA32

Проведено аналіз швидкодії класичного та адаптивного методу CORDIC на платформі IA32 мовою асемблера, а також відомих методів оптимізації алгоритмів, таких як табличний метод і залишкове множення.

Performed speed analysis of classic and adaptive CORDIC-methods on the IA32 platform with Assembler programming language, and well-known methods of algorithm optimisation such as table method and residual multiplication.

1. ВСТУП

Ітераційний метод CORDIC – COordinate Rotation DIgital Computer є одним з найпопулярніших методів обчислення багатьох елементарних функцій. Незважаючи на те, що методу уже понад 50 років, він продовжує інтенсивно розвиватись [5,6,8] . Теоретичні основи методу CORDIC описані в роботах [1-4,7]. Найповніше узагальнення методу проведено у роботах [1,7,11]. Таке узагальнення важливе при побудові реконфігурованих обчислювачів, що працюють за методом CORDIC, тому що дозволяє побудувати універсальну структуру, яка реалізує будь-яку функцію з числа відтворюваних.

2. МЕТА РОБОТИ

В даній роботі проведено аналіз швидкодії класичного та адаптивного методу CORDIC на платформі IA32 мовою асемблера, а також відомих методів оптимізації алгоритмів, таких як табличний метод і залишкове множення. Вибір даної платформи пояснюється, у першу чергу, її 32 розрядною архітектурою, що дає змогу працювати з операндами як з одним цілим і зосередитись безпосередньо на діях алгоритму. По друге, висока швидкодія даної архітектури дозволяє отримати значення кількості тактів, які необхідні для усіх можливих вхідних значень алгоритму, і при цьому здійснювати це динамічно, під час виконання алгоритму. Розділивши отримане значення на 2^{32} і округливши його до найближчого цілого, одержимо середнє значення необхідної кількості тактів для довільного кута досліджуваного алгоритму. Всі заміри проводились на платформі Athlon 1600+ з ядром Thoroughbred.

[1] Національний університет «Львівська політехніка»

Слід відзначити, що отримане значення кількості тактів не залежить від тактової частоти процесора, а змінюється в залежності від його внутрішньої архітектури. Тому, як показала практика, отримані результати є порівнювальні лише в межах одного сімейства процесорів.

3. ОПИС ДОСЛІДЖЕНЬ

Порівняння методів відбувається у наступному порядку: реалізується класичний метод CORDIC, і після його оптимізації шляхом застосування методів залишкового множення і таблично-алгоритмічного методу, здійснюється його співставлення з адаптивним методом по показниках швидкодії та похибок обчислень.

У класичному методі CORDIC для обчислення деякого косинуса(x) та синуса(y) кута φ виконується:

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

де

$$\alpha_i = \arctan\left(2^{-i}\right)$$
$$\sigma_i = \begin{cases} -1 \; if \;\; z_i < 0 \\ +1 \; if \;\; z_i \geq 0 \end{cases}$$

$$x_0 = P = 1/K_m \;\; , \;\; y_0 = 0 \;\; , \;\; z_0 = \varphi \, .$$

На рис. 1а зображений класичний 32-ітераційний CORDIC. Робота даного алгоритму займає 420 тактів процесора. Покращити цей результат можна використовуючи метод, відомий як «залишкове множення» [12] (Рис. 1б.). Суть його полягає у множенні отриманих результатів синуса та косинуса на залишковий кут, який отримується після виконання половини ітерацій алгоритму. Даний спосіб стає можливий через те, що похибка результату множення коефіцієнта деформації на одержані значення функцій після половини ітерацій виходить за межі розрядної сітки у тридцять два розряди і не впливає на точність обчислень.

У випадку використання залишкового множення виконується 15 проходів класичного алгоритму (замість 31) і здійснюється дві операції множення, що покращує результат у два рази – до 200 тактів.

$$x_{17} = x_{16} - \phi_2 y_{16}, \quad x_{17} = \cos(\phi_{input}),$$

$$y_{17} = y_{16} - \phi_2 x_{16}; \quad y_{17} = \sin(\phi_{input});$$

Структура методу зображена на рис 2а.

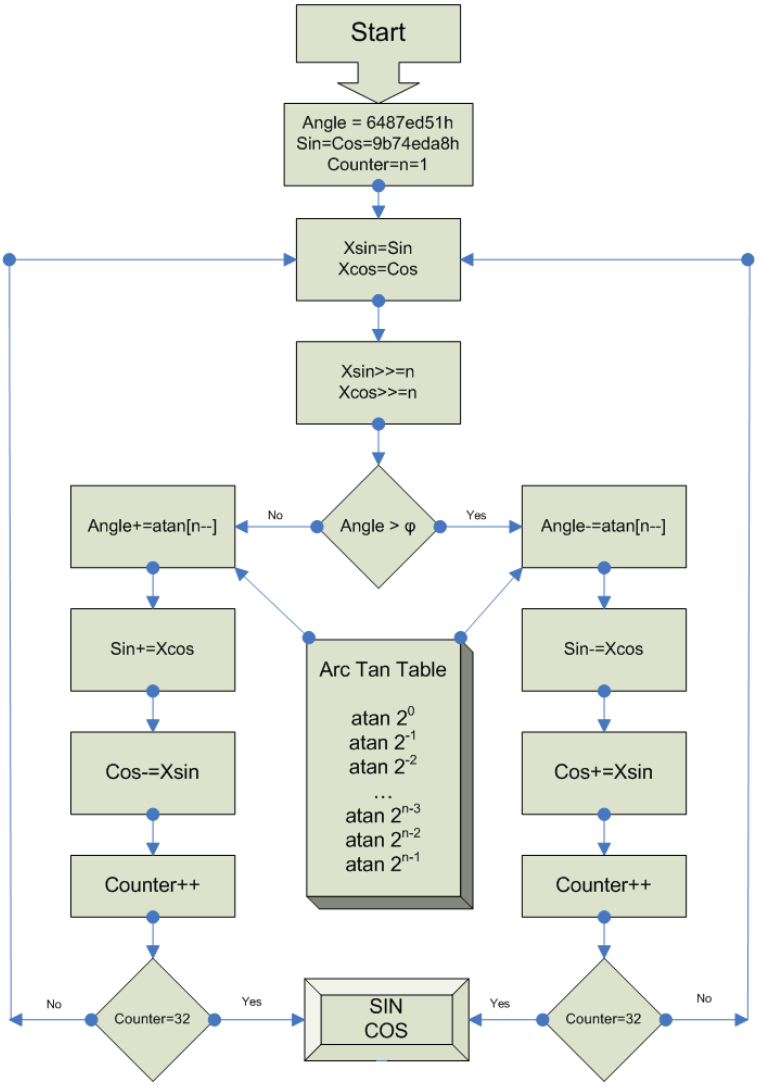

Рис. 1а Класичний CORDIC

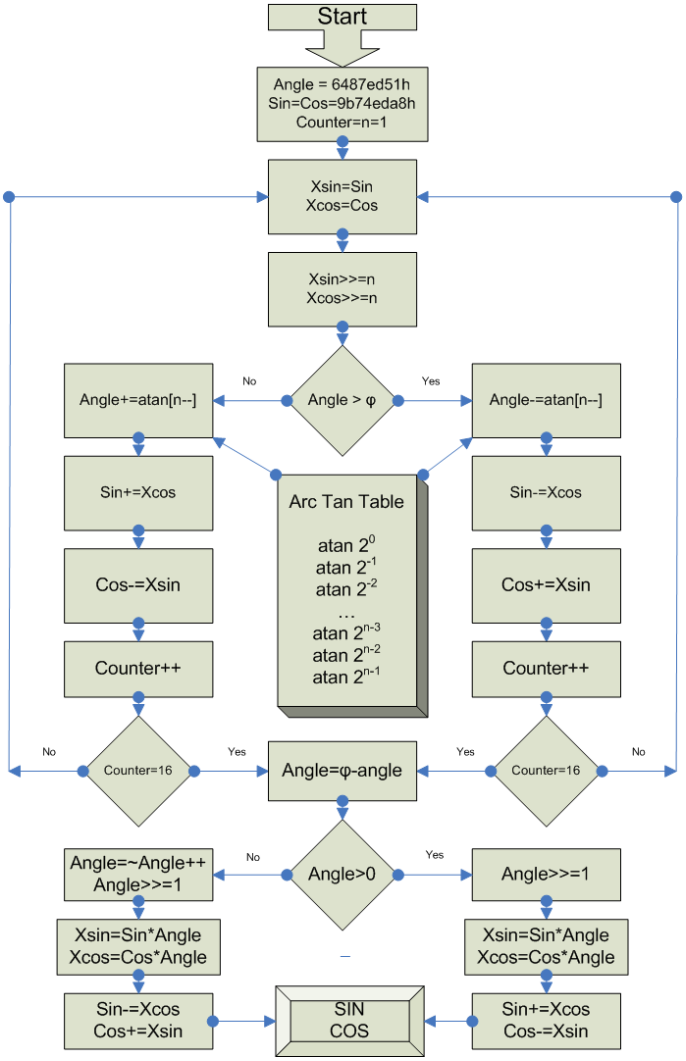

Рис. 1б. Залишкове множення

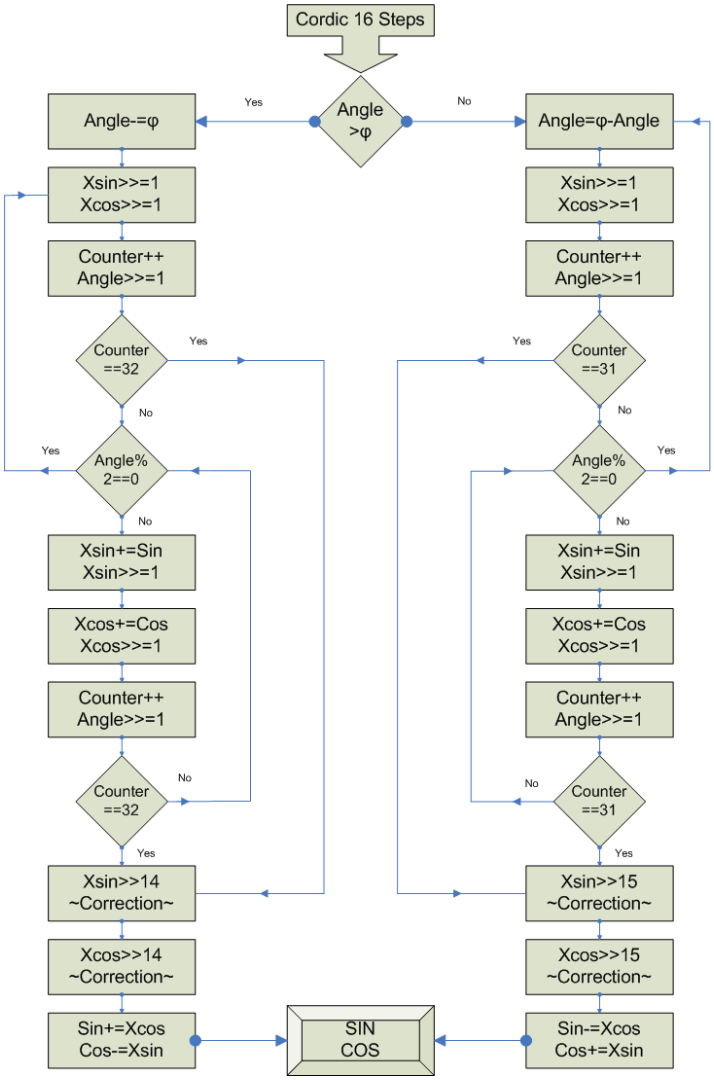

Рис. 2а Метод Add-shift

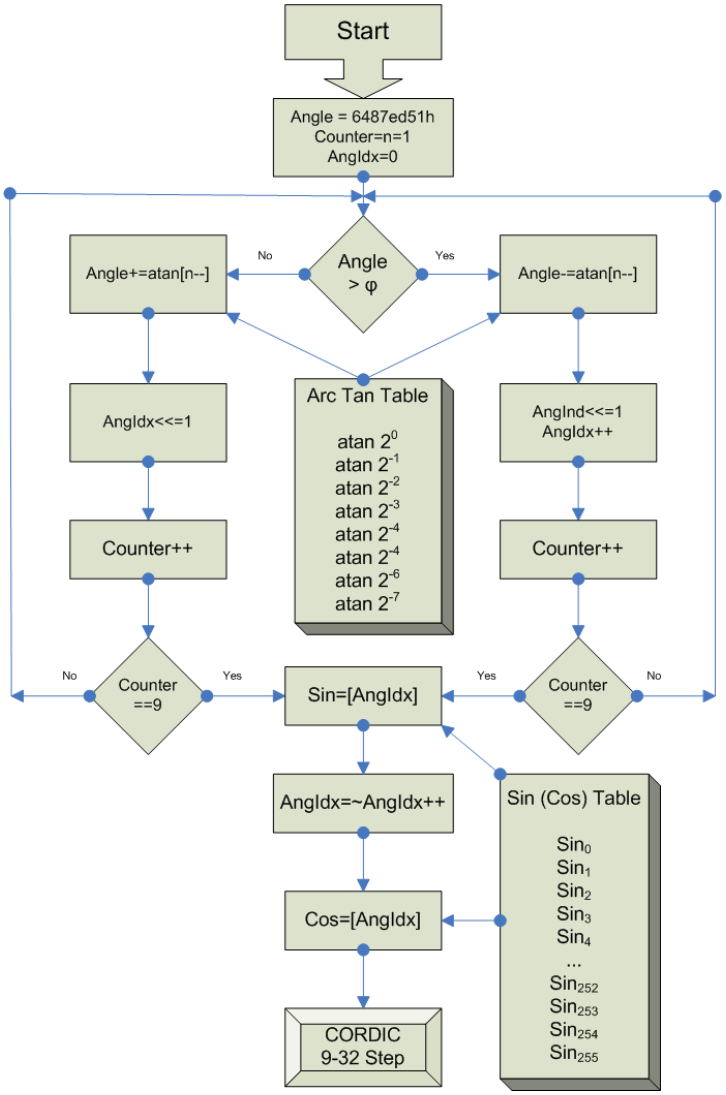

Рис. 2б Табличний метод

Слід зауважити, що даний метод вимагає здійснення операції множення, яка зазвичай є недоступна для платформ, на яких реалізується CORDIC. Тому таку операцію доцільно замінити методом add-shift:

$$x_{17}=x_{16}-y_{16}(a_{17}2^{-17}+ a_{18}2^{-18}+\ldots a_{31}2^{-31}),$$
$$y_{17}=y_{16}-x_{16}(a_{17}2^{-17}+ a_{18}2^{-18}+\ldots a_{31}2^{-31});$$

Виконання методу add-shift в алгоритмі CORDIC триває 303 такти.

Ще одним способом покращення результату є табличний метод. Він полягає в створенні наперед обчисленої таблиці значень синуса та косинуса для n-ої кількості ітерацій. Розмір таблиці для кожної наступної ітерації збільшується вдвічі, тому цей розмір є строго фіксований для кожної конкретної платформи і, зазвичай, невеликий. Для порівняння, була використана таблиця у 8 ітерацій (1Кб пам'яті) Рис. 2б. Швидкодія табличного методу у сукупності із Add-shift склала 215 тактів.

Розмір таблиці можна розрахувати за наступною формулою:

$$S = m \cdot 2^{n-1}$$

S- розмір таблиці (в байтах); n-кількість ітерацій; m-кількість таблиць.

Адаптивний метод

$$\phi_{input} = \phi_1 + \phi_2 + \phi_3 + \phi_4$$

$$\phi_1 = \sum_{i=0}^{4} a_i \times 2^{-i} \quad \phi_2 = \sum_{i=5}^{9} a_i \times 2^{-i} \quad \phi_3 = \sum_{i=10}^{15} a_i \times 2^{-i} \quad \phi_4 = \sum_{i=16}^{31} a_i \times 2^{-i}$$

$x_4=\cos(\phi_1)$, $y_4=\sin(\phi_1)$;

$x_{i+1}=x_i(1-\varepsilon_i^2/2)-y_i(\varepsilon_i-\varepsilon_i^3/8)$; $y_{i+1}=y_i(1-\varepsilon_i^2/2)+x_i(\varepsilon_i-\varepsilon_i^3/8)$;

$x_{i+1}=x_i(1-\varepsilon_i^2/2)-y_i\varepsilon_i$; $y_{i+1}=y_i(1-\varepsilon_i^2/2)+x_i\varepsilon_i$;

$x_{i+1}=x_i-\phi_4 y_i$, $y_{i+1}=y_i+\phi_4 x_i$;

$\varepsilon_i = 2^{-i}$.

Для реалізації табличного методу в адаптивному CORDICу необхідно створити дві окремі таблиці із значеннями синуса та косинуса. Для адаптивного методу була реалізована п'ятикрокова таблиця, об'єм якої згідно вищенаведеної формули становить 256 байт.

Наступні 11 кроків алгоритму (з 5 по 15) здійснюються безпосередньо адаптивним методом, після чого виконується операція залишкового множення add-shift методом. В результаті у всьому алгоритмі використовуються лише операції бінарного зсуву та додавання, а також 256 байт пам'яті для зберігання таблиць. Швидкодія алгоритму адаптивного CORDICа склала 186 тактів, що на 16% швидше, ніж аналогічний класичний метод з таблицею розміром в один кілобайт. Результати роботи алгоритмів представлені у вигляді гістограми рис. 4.

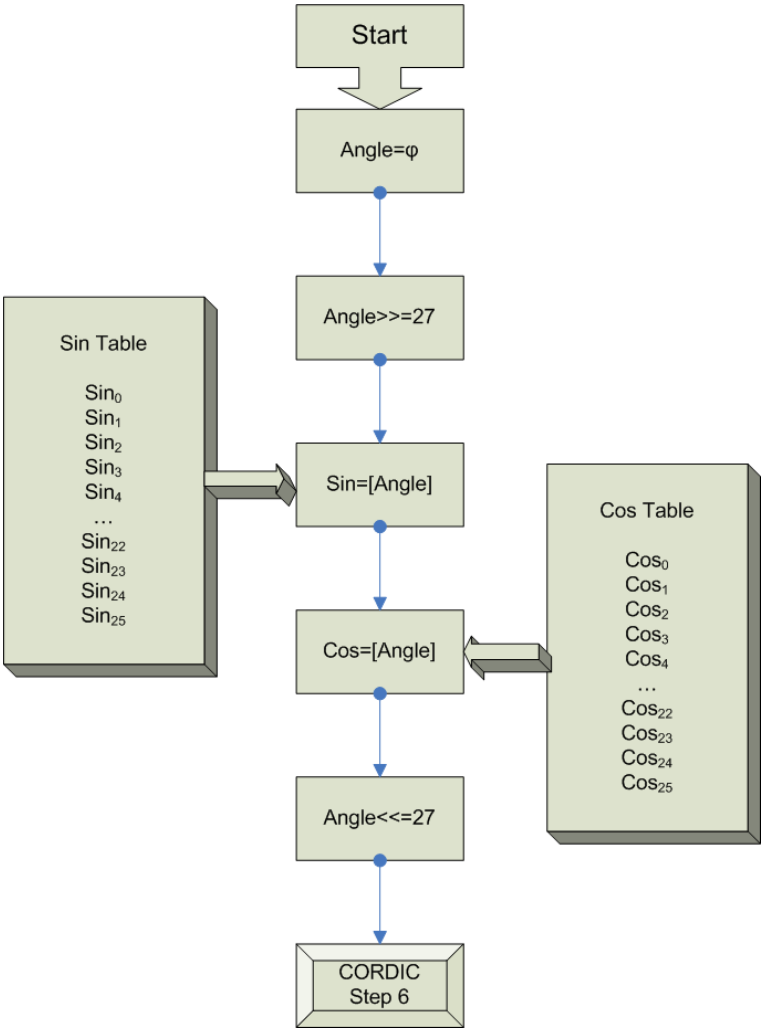

Рис. 3а. Табличний адаптивний

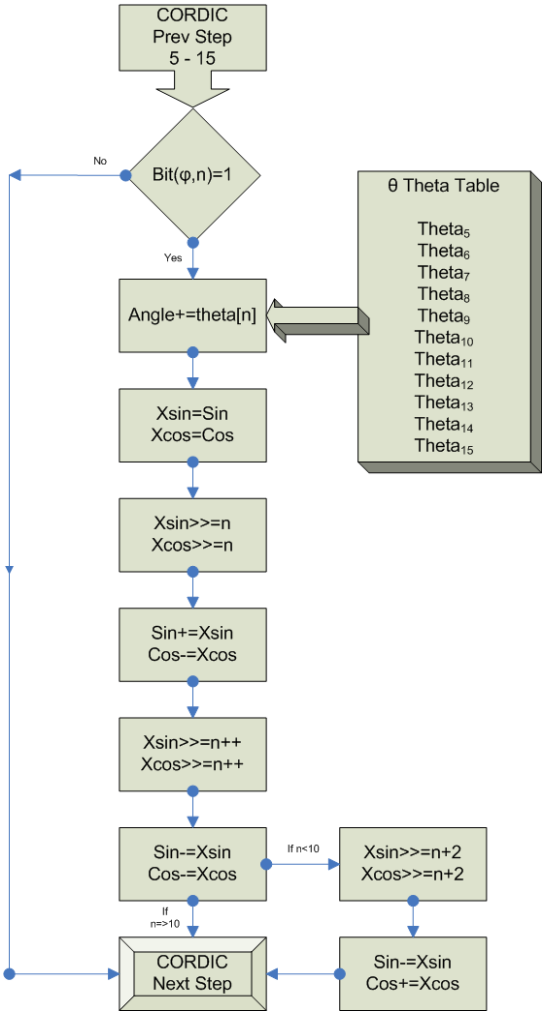

Рис. 3б. Адаптивний основний

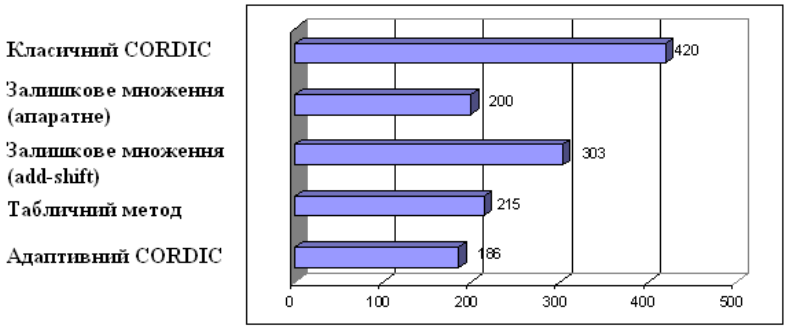

Рис. 4. Результати дослідження швидкодії (в тактах процесора)

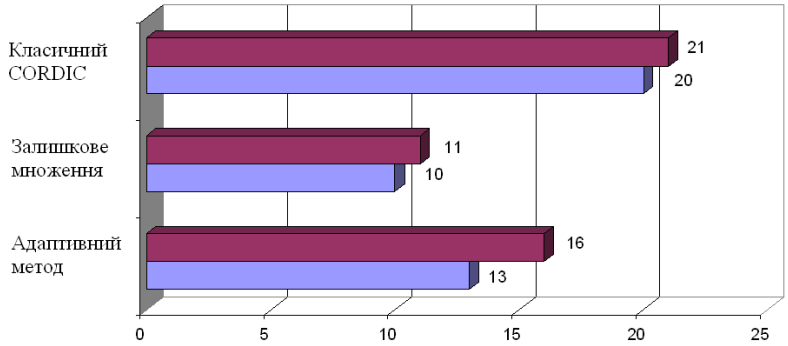

Рис. 5. Максимальні значення похибок

Оскільки алгоритм оперує цілочисельною тридцятидворозрядною мантисою, то похибку алгоритму можна виразити як модуль різниці абсолютного і одержаного значення. Для переведення відхилення у множину раціональних чисел необхідно домножити похибку на 2^{-32}, або здійснити бінарний зсув вправо на тридцять два розряди. Також можна визначити кількість точних двійкових розрядів, для цього необхідно від загальної кількості розрядів відняти похибку δ, яка в нашому випадку буде становити

$$32 - \log_2 \delta.$$

Як видно з рис. 5 найбільші значення похибок має класичний метод, який нівелює п’ять молодших розрядів. Метод залишкового множення (апаратний чи програмний), має найвищу точність результатів, в той час як адаптивний хоча і відстає від нього, але вписується в показник у чотири плаваючі розряди, що дає 28 точних двійкових розрядів. Таблично-алгоритмічний метод на рисунку не наведений, оскільки

його застосування не відображається на точності алгоритму, а впливає лише на його швидкодію.

4. ВИСНОВКИ

Отже, аналіз класичного та адаптивного алгоритму CORDIC із застосуваням до них відомих способів оптимізації, таких як залишкове множення і таблично-алгоритмічний метод, показує, що адаптивний CORDIC в порівнянні з класичним має вищу швидкодію і в практичній реалізації при 32-бітних обчисленнях дає виграш у 16%. При цьому до переваг адаптивного методу можна віднести вчетверо менший об'єм таблиць, що дозволяє реалізовувати його на платформах з нижчими технічними вимогами.

1.Walther, J.S. A unified algorithm for elementary functions. In Proceedings of AFIPS 1971 Spring Joint Computer Conference, vol. 38, AFIPS Press, Arlington, Va., 1971, pp. 379-385. 2. Volder J.E. The CORDIC trigonometric computing technique, IRE Trans. Electron. Comput., Vol. EC-8, NO.3, Sept. 1959,pp. 330-334.3.Оранский А.М. Аппаратные методы в цифровой вычислительной технике.- Минск: Издательство БГУ,1977.-208с. 4.Байков В. Д., Смолов В. Б. Специализированные процессоры: итерационные алгоритмы и структуры . – Москва: "Радио и связь", 1985.-288с. 5.Beuler M. CORDIC-Algorithmus zur Auswertung elementarer Funktionen in Hardware. FH-Report. Juni 2008. http://dok.bib.fhgiessen. *de/opus/volltexte/2009/4148/ 6. Pramod K. Meher, Javier Valls,Tso-Bing Juang,K. Sridharan,Koushik Maharatna. 50 Years of CORDIC: Algorithms, Architectures,and Applications. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS–I: REGULAR PAPERS, VOL. 56, NO. 9, SEPTEMBER 2009, pp. 1893- 1907. 7. Аристов В.В. Функциональные макрооперации. Основы итерационных алгоритмов. – К.: Наукова думка, 1992.-280с. 8. Lakshmi B. and Dhar A. S. CORDIC Architectures: A Survey. VLSI Design. Volume 2010, Article ID 794891, 19 pages, January 8, 2010. 9. Llamocca-Obregón D.R., Agurto-Ríos C.P. A Fixed-Point Implementation of the Natural Logarithm Based on a Expanded Hyperbolic CORDIC Algorithm. XIIWORKSHOP IBERCHIP, p.p. 322 -323, 2006. 10. X. Hu, R. Huber, S. Bass, "Expanding the Range of Convergence of the CORDIC Algorithm", IEEE Transactions on Computers. Vol. 40, N° 1, pp. 13-21,1991. 11.Beuler, M.; Bonath, W. Berechnung elementarer Funktionen für das Huber/Braun-Neuronenmodell in digitaler Architektur; MPCWorkshop Konstanz, Juli 2008. http://dok.bib.fh-giessen.de/opus/volltexte/2009/4149/.12. Микитів Т.М.,Мороз Л.В. Мікроконтроллерний синусно-косинусний обчислювач підвищеної швидкодії. Вісник НУ "Львівська політехніка" – "Автоматика, вимірювання та керування", №, 2011. – С. 69-74.*