DOI: 10.20535/2523-4455.mea.205929

УДК 004.31

# Апаратна реалізація потокового обчислювача логарифму для даних в форматі з фіксованою комою

Гордієнко[f] Я. О., ORCID 0000-0001-7127-7232
Варфоломєєв А. Ю., к.т.н, ORCID 0000-0002-6990-7140
Короткий[s] Є. В., к.т.н., ORCID 0000-0001-8302-4873
Кафедра конструювання електронно-обчислювальної апаратури keoa.kpi.ua
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського» kpi.ua
Київ, Україна

***Анотація*—В роботі розглянуто існуючі підходи до обчислення логарифму. Запропоновано параметризовану апаратну реалізацію потокового обчислювача логарифму за основою 2 для даних в форматі з фіксованою комою, що дозволяє визначати точність обчислень, а також розрядності цілої й дробової частин даних на вході та виході обчислювача. Створено високорівневу параметризовану модель запропонованого обчислювача в MATLAB® Simulink®, з застосуванням якої виконано оцінку точності розрахунків. З високорівневої моделі MATLAB® Simulink® синтезовано вихідний код реалізації запропонованого обчислювача на мові Verilog, а також тестовий стенд (тестбенч) для верифікації на рівні регістрових передач. В симуляторі ModelSim проведено верифікацію запропонованого обчислювача на рівні регістрових передач. Зі створеного вихідного коду на мові Verilog синтезовано апаратну реалізацію запропонованого обчислювача в базисі програмованої логіки з використанням Intel FPGA Quartus Prime. Виконано порівняння запропонованого обчислювача з існуючими аналогами за помилкою розрахунку результату та апаратурними витратами.**

***Ключові слова — обчислювач; логарифм; фіксована кома; Verilog; ПЛІС; FPGA; програмована логіка***

## I. Вступ

Операція логарифмування знаходить широке застосування в галузі телекомунікацій, зокрема використовується в радіо розвідці для перетворення значень амплітудного спектру в децибели з метою спостереження оператора за радіо ефіром, або для автоматизованого аналізу спектру спеціалізованими програмно-апаратними засобами, наприклад, з метою пеленгації джерел радіо випромінення [1-2].

Для виконання обчислень над потоком даних з радіо ефіру, що має високу частоту дискретизації та моніторингу радіо сигналів, що швидко змінюються (наприклад, сигналів з псевдо випадковим перестроюванням несучої частоти), з метою зменшення тривалості процесу обчислень, доцільне застосування потокової апаратної реалізації обчислювачів, наприклад на основі мікросхем програмованої логіки типу FPGA. Потокова апаратна реалізація дозволяє опрацьовувати вхідний потік даних у реальному часі без втрат відліків сигналу на вході та спрощує конвеєризацію обчислень, що створює передумови для підвищення тактової частоти роботи системи та збільшення продуктивності обчислень у кількості операцій на одиницю часу.

Компанії Intel FPGA та Xilinx, що займають провідні позиції у виробництві мікросхем програмованої логіки типу FPGA, пропонують потокові реалізації операції логарифму для даних з плаваючою комою у вигляді захищених блоків інтелектуальної власності (IP Cores). Блоки обчислення логарифму для даних з плаваючою комою і можливістю синтезу апаратної реалізації на мовах Verilog/VHDL також входять до складу компонентів системи наскрізного моделювання MATLAB® Simulink®.

З іншого боку, обчислювачі для даних в форматі з плаваючою комою обумовлюють значні апаратурні витрати, а параметризовані потокові апаратні реалізації операції логарифму для даних з фіксованою комою та відкритим вихідним кодом наразі відсутні. Як наслідок, актуальною стає задача розробки відкритої апаратної реалізації потокового обчислювача операції логарифму для даних в форматі з фіксованою комою для зменшення апаратурних витрат обчислювача, що є метою пропонованої роботи.

Робота структурована наступним чином. У другому розділі наведені відомі підходи до обчислення логарифму. У третьому розділі розглянуті теоретичні засади запропонованого апаратного обчислювача. У четвертому розділі описана реалізація запропонованого обчислювача у вигляді моделі MATLAB® Simulink® в форматі, що дозволяє виконати синтез апаратної реалізації на мовах Verilog/VHDL. В п'ятому розділі наведені оцінки апаратурних витрат, аналіз точності запропонованого обчислю-

DOI: 10.20535/2523-4455.mea.205929

вача та результати його верифікації шляхом симуляції на рівні регістрових передач в середовищі ModelSim, а також виконано порівняння запропонованого обчислювача з аналогами.

## II. Аналіз відомих напрацювань

Відповідно до огляду, що наведений в [3], для реалізації апаратних обчислювачів логарифму використовують табличний та ітеративний підходи, алгоритм повороту координат CORDIC та апроксимацію за допомого ряду Тейлора.

Табличний підхід до обчислення функції логарифму описаний в [4-6]. Подібний спосіб розрахунку не потребує значних апаратурних витрат, зокрема немає необхідності у застосуванні перемножувачів. З іншого боку, для досягнення прийнятної точності та збереження всіх можливих цифрових кодів результату у вигляді таблиці може знадобитися значний обсяг пам'яті. Вимоги до обсягу пам'яті можна зменшити у випадку обмеження діапазону значень на вході обчислювача, однак у такому разі знижується універсальність рішення.

Обчислювати логарифм можна з використанням ітеративного підходу, що описаний в [7-8]. Подібне рішення застосовується для значень на вході обчислювача в інтервалі від 1 до 2, що обмежує використання підходу. Для розрахунку логарифму значення на вході обчислювача зберігається у змінній $X$, а вміст змінної ітеративно підноситься до квадрату. На кожній ітерації визначається один біт результату в залежності від того, чи виконується умова $X^2 > 2$. Кількість ітерацій обумовлена розрядністю результату. Деталі реалізації наведені в [7-8]. Подібний підхід у випадку ітеративної реалізації потребує одного перемножувача (або одного блоку піднесення до квадрату) і не використовує блоки пам'яті. Однак у випадку ітеративної реалізації втрачається можливість потокової обробки даних. Конвеєризація рішення можлива, однак потребує значного збільшення кількості перемножувачів, що призводить до зростання апаратурних витрат.

Для обчислення логарифму може бути використаний широко відомий алгоритм повороту координат CORDIC [3], [9-10]. Для реалізації алгоритму CORDIC необхідні суматори, пристрої зсуву та блоки пам'яті. Однак створення обчислювача логарифму на основі CORDIC додатково потребує застосування перемножувачів, що призводить до збільшення апаратурних витрат.

Існує можливість апроксимувати дробову частину логарифму з використанням ряду Тейлора [3]. Подібний підхід потребує залучення значної кількості перемножувачів, що негативно впливає на апаратурні витрати і максимально можливу тактову частоту.

## III. Теоретичні засади

У пропонованій роботі розроблено апаратну реалізацію обчислювача логарифму за основою 2.

Маючи можливість розраховувати значення функції $\log_2(X)$ від аргументу $X$, для обчислення логарифму за будь якою іншою основою $b$ можна скористатися тотожністю $\log_b(X) = \frac{\log_2(X)}{\log_2(b)}$. На апаратному рівні для зміни основи логарифму результат обчислення $\log_2(X)$ необхідно домножити на константу $\frac{1}{\log_2(b)}$, де $b$ – нове значення основи логарифму, що потребує одного додаткового перемножувача на константу. Враховуючи, що перемноження на константу можна реалізувати з використанням shift-add алгоритму на кількох суматорах, зміна основи логарифму не приводить до значного зростання апаратурних витрат.

Розглянемо найпростіший випадок розрахунку цілочисельного логарифму за основою 2 для цілочисельного аргументу $X$.

Із визначення логарифма не складно довести, що:

$$\forall\, X \in \mathbb{N},\ \exists\, p \in \mathbb{N} : 2^{p-1} \le X < 2^p, \qquad (1)$$

де $p$ – кількість значущих бітів, необхідна для представлення $X$. Дійсно, якщо число $X$ представляють $p$ значущих бітів, значить:

- $\max(X) = 2^p - 1 < 2^p$;
- принаймні біт з позицією $p - 1$ приймає значення 1, отже $\min(X) = 2^{p-1}$.

З (1) отримуємо:

$$(p-1) \le \log_2(X) < p. \qquad (2)$$

Отже:

$$\begin{cases} \lfloor \log_2(X) \rfloor = p-1, \\ \lceil \log_2(X) \rceil = p \end{cases} \qquad (3)$$

де $\lfloor \cdot \rfloor$ та $\lceil \cdot \rceil$ позначає округлення до найближчого меншого та більшого цілого відповідно.

Підходи до оцінки кількості значущих бітів $p$ для числа $X$ будуть розглянуті далі в роботі.

Для більш точного розрахунку логарифму за основою 2 від цілочисельного аргументу $X \in \mathbb{N}$ запишемо:

$$X = 2^{p-1} \cdot X_m, \qquad (4)$$

де $p$ – кількість значущих бітів, необхідна для представлення $X$, а $X_m \in [1, 2) \wedge X_m \in \mathbf{R}$.

З виразу (4) можна отримати $X_m$:

$$X_m = \frac{X}{2^{p-1}} = X >> (p-1), \qquad (5)$$

де >> позначає операцію логічного зсуву вправо, що реалізується апаратно за допомогою схеми швидкого зсуву (barrel shifter). У разі представлення $X_m$ у беззнаковому форматі з фіксованою комою, ціла частина $X_m$ буде займати 1 біт і завжди прийматиме значення 1, а дробова частина $X_m$ буде займати $p - 1$ біт.

Виходячи з (4) можна обчислити логарифм за основою 2 від цілочисельного аргументу $X$:

DOI: 10.20535/2523-4455.mea.205929

$$\log_2(X) = \log_2(2^{p-1} \cdot X_m) = p - 1 + \log_2(X_m), \quad (6)$$

де $p - 1$ визначає цілу частину логарифму, а $\log_2(X_m)$ визначає дробову частину логарифму і приймає значення на проміжку [0, 1).

Враховуючи, що $X_m \in [1, 2)$, доцільно обчислювати $\log_2(X_m)$ табличним методом. Далі в роботі показано, що використання навіть невеликої таблиці на 32 комірки на області значень $X_m \in [1, 2)$ дозволяє досягти прийнятної точності.

Розглянемо підхід до розрахунку логарифму за основою 2 для дійсного аргументу $Y \in R \wedge Y > 0$.

У разі представлення аргументу $Y$ у форматі з фіксованою комою і дробовою частиною, що займає $k \in \mathbb{N}$ бітів, значення $Y$ можна записати, як:

$$Y = \frac{X}{2^k}, \quad (7)$$

де $X \in \mathbb{N}$.

З урахуванням (6) можна записати:

$$\log_2(Y) = \log_2\left(\frac{X}{2^k}\right) = \log_2(X) - k = \\ = p - k - 1 + \log_2(X_m) \quad (8)$$

Найбільш очевидним способом розрахунку кількості значущих бітів $p$, необхідних для представлення $X$ на апаратному рівні, є використання пріоритетного шифратора. Однак в роботі [11] показано, що реалізація пріоритетного шифратора для аргументів зі значною розрядністю (10 бітів і вище) призводить до високих апаратурних витрат і тому доцільна лише у випадку, коли необхідно отримати результат за один такт, як наприклад у контролерах переривань. Для задач потокової обробки з конвеєризацією обчислень для розрахунку $p$ доцільно використовувати підхід на основі операції $CLZ(\cdot)$ з меншими апаратурними витратами.

Функція $CLZ(\cdot)$ є скороченням від "Count Leading Zeroes" і використовується для підрахунку кількості провідних нульових бітів двійкового аргументу скінченної розрядності, що передують старшому ненульовому розряду аргументу. У випадку нульового значення аргументу, результатом $CLZ(\cdot)$ буде кількість розрядів аргументу. Іноді для операції $CLZ(\cdot)$ використовують позначення *NLZ* (Number of Leading Zeroes).

За допомогою операції $CLZ(\cdot)$ можна визначити кількість значущих бітів $p \in \mathbb{N}$ для аргументу $X$ розрядністю $m \in \mathbb{N}$ біт:

$$p = m - CLZ(X) \quad (9)$$

Враховуючи (9), формулу (8) можна переписати у вигляді:

$$\log_2(Y) = m - k - 1 - CLZ(Y) + \log_2(X_m). \quad (10)$$

В [12] описано алгоритм обчислення $CLZ(X)$ на основі двійкового пошуку, який реалізується у вигляді схеми з низькими апаратурними витратами. Алгоритм можна описати за допомогою псевдокоду:

```
data = X;
result = 0;
for(i = ceil(log2(m)) - 1; i >= 0; i = i - 1) {
  if (hi(data) != 0) {
    data = hi(data);
  } else {
    data = lo(data);
    setbit(result, i);
  }
}
result = result + !data;
```
(11)

У наведеному псевдокоді цілочисельна змінна $X$ містить аргумент операції $CLZ(X)$, змінна *result* містить результат операції $CLZ(X)$, константа $m$ визначає розрядність аргументу $X$, функція *hi*(*data*) повертає старшу половину бітів аргументу *data*, функція *lo*(*data*) повертає молодшу половину бітів аргументу *data*, функція *setbit*(*result*, *i*) встановлює *i*-й біт змінної *result*, що еквівалентно додаванню $2^i$ до *result*. Розрядність аргументу $X$ повинна дорівнювати ступіню двійки. Якщо ця умова не виконується, необхідно доповнити $X$ нулями у старших розрядах для дотримання умови.

Алгоритм, описаний в (11) має наступну логіку роботи. На кожній ітерації алгоритму оцінюється старша половина бітів змінної *data*. Якщо старша половина бітів змінної *data* містить ненульові біти, зазначена старша половина бітів змінної *data* записується у змінну *data* і передається для аналізу на наступну ітерацію алгоритму, а змінна *result* зберігає своє значення. Якщо старша половина бітів змінної *data* містить лише нулі, молодша половина бітів змінної *data* записується у змінну *data* і передається для аналізу на наступну ітерацію алгоритму, а до змінної *result* додається $2^i$, де $i$ – номер ітерації алгоритму, а $2^i$ дорівнює половині від розрядності змінної *data* на початку ітерації алгоритму. Відповідно, в кінці кожної ітерації алгоритму розрядність змінної *data* зменшується вдвічі.

## IV. Модель обчислювача

Для реалізації (10) та (11) розроблено параметризовану модель обчислювача логарифму за основою 2 в MATLAB® Simulink® з використанням компонентів сумісних з HDL Coder, що дозволяє використовуючи інструмент HDL Coder синтезувати з зазначеної високорівневої моделі реалізацію обчислювача на рівні регістрових передач, описану на мовах Verilog/VHDL.

Розроблена Simulink модель log2_fixed_point.slx та m-файл для визначення параметрів моделі set_log2_fixed_point_params.m знаходяться у відкритому доступі і можуть бути завантажені за посиланням [13].

DOI: 10.20535/2523-4455.mea.205929

Ліцензію з повною підтримкою компонентів MATLAB® Simulink® отримано від компанії MathWorks™ в рамках її університетської програми для участі в конкурсі студентських проектів Digilent Design Contest 2019.

Параметри розробленої моделі зберігаються у файлі set_log2_fixed_point_params.m в структурі *log2_fixed_point*. Параметр *in_word_width* визначає розрядність аргументу обчислювача, що представлений в форматі з фіксованою комою. Параметр *in_fraction_width* визначає розрядність дробової частини аргументу обчислювача. Максимальна розрядність аргументу обчислювача обумовлена параметром *in_word_width_max* і дорівнює 128 біт. Параметр *out_word_width* визначає розрядність виходу обчислювача (іншими словами, розрядність розрахованого значення логарифму), що представлений в форматі з фіксованою комою. Параметр *out_fraction_width* визначає розрядність дробової частини виходу обчислювача. Параметр *log2_table_size* визначає розмір таблиці, що зберігає значення логарифму для розрахунку $\log_2(X_m)$.

Створена Simulink модель обчислювача, що реалізує (10), наведена на рис.1. На вхід i_DATA подається аргумент операції логарифму. На вхід i_VALID подається строб коректності даних на вході i_DATA. У випадку неперервного потоку даних на вході i_DATA, сигнал на вході i_VALID постійно рівний логічній одиниці. На виході o_DATA з'являється результат операції $\log_2$(i_DATA), з затримкою 9 тактів відносно подачі відповідного значення i_DATA на вхід обчислювача. Отже латентність (latency) обчислювача складає 9 тактів сигналу синхронізації. Потік даних опрацьовується в режимі конвеєра.

Компонент $z^{-q}$ моделює лінію затримки даних на $q$ періодів тактового сигналу. В апаратній реалізації такий компонент представляється послідовним з'єднанням $q$ синхронних по фронту D тригерів для одно бітових сигналів або $q$ синхронних по фронту паралельних регістрів для багато бітових сигналів.

Блок CLZ є компонентом для обчислення операції $CLZ(\cdot)$. Результат операції $CLZ(\cdot)$ з'являється на виході o_CLZ блоку CLZ.

Оскільки латентність потокового обчислення операції CLZ складає 8 тактів, сигнали з входів i_VALID та i_DATA теж затримуються всередині блоку CLZ на 8 тактів і затримані значення з'являються на виходах o_VALID, o_DATA блоку CLZ.

Компоненти UintCast та ZeroExtend розширюють нулями у старших розрядах значення з входу i_DATA до 128 біт. Такий підхід не приводить до значного збільшення апаратурних витрат для низьких розрядностей i_DATA, оскільки під час синтезу схемотехнічної реалізації САПР оптимізує (вилучає) логічні елементи на входах яких сигнал постійно приймає нульове значення.

Константа Const0 зображена на рис. 1 дорівнює *in_word_width_max - 1 - in_fraction_width*. Над значенням Const0, виходом o_CLZ та значенням $\log_2(X_m)$ з виходу блоку Log2Table виконується арифметична операція відповідно до (10), результат якої з використанням блоку CastOutType приводиться до типу визначеного параметрами *out_word_width* та *out_fraction_width* і результат передається на вихід обчислювача.

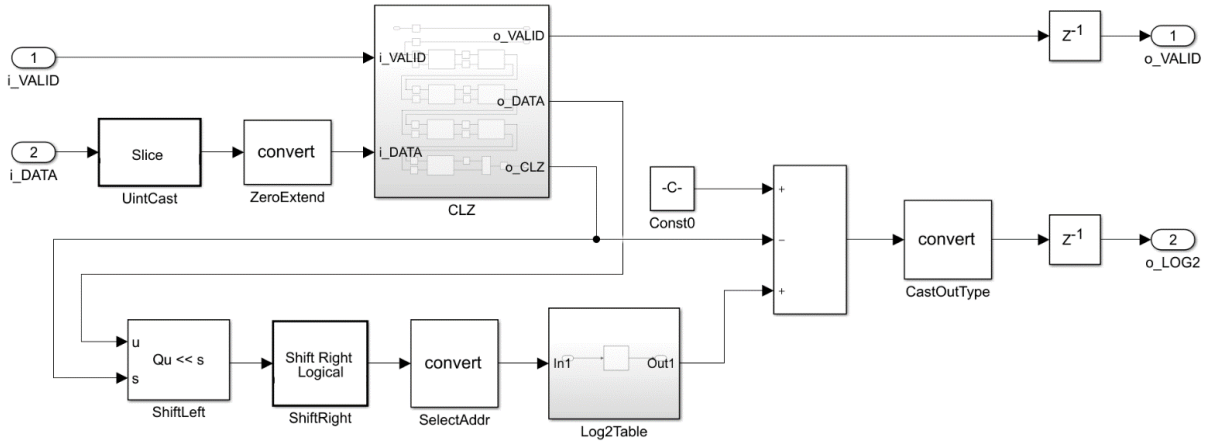


Рис. 1 Simulink® модель потокового обчислювача логарифму за основою 2

DOI: 10.20535/2523-4455.mea.205929

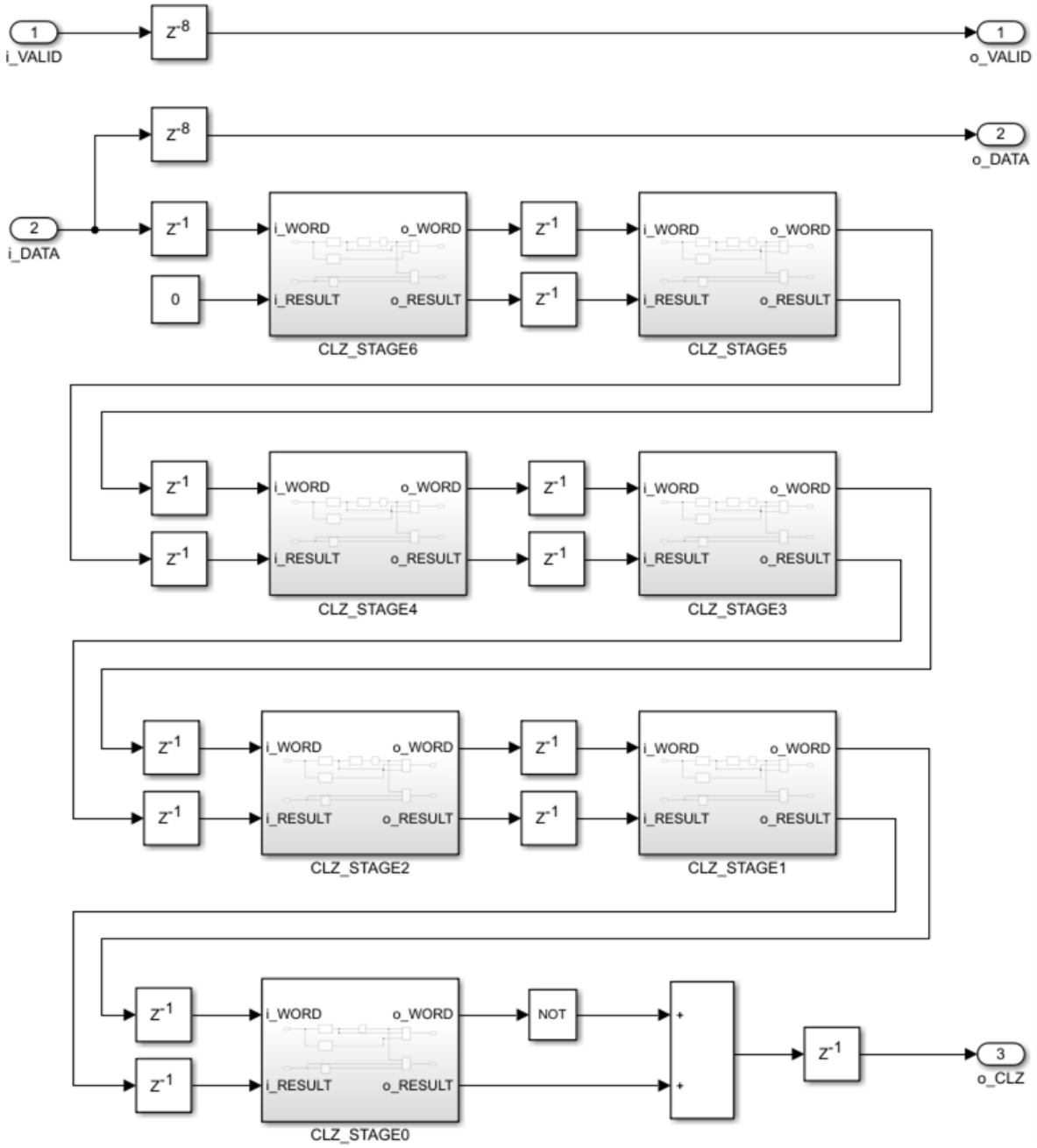


Рис. 2 Simulink® модель блоку CLZ

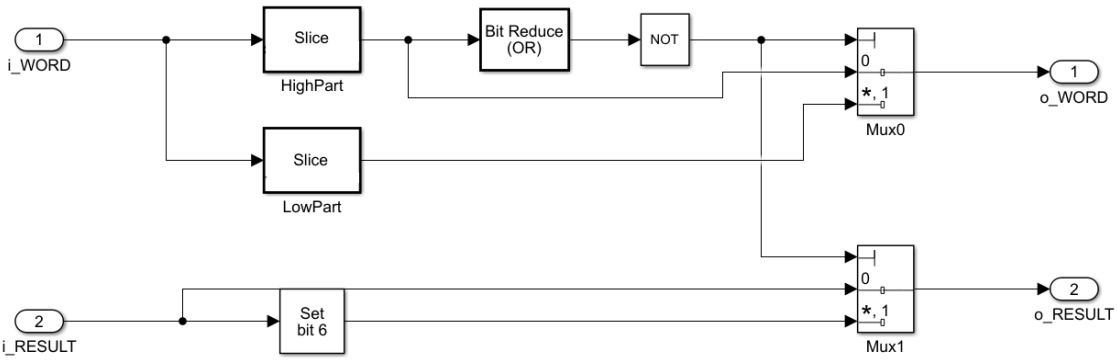


Рис. 3 Simulink® модель блоку CLZ_STAGE6

DOI: 10.20535/2523-4455.mea.205929

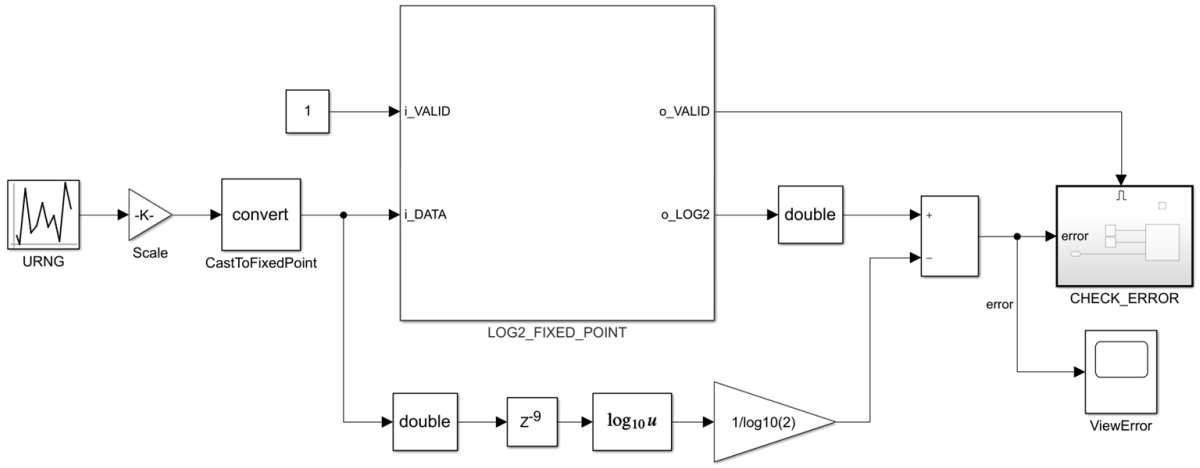


Рис. 4 Simulink® модель тестового стенду для верифікації потокового обчислювача логарифму за основою 2

Розрахунок $\log_2(X_m)$ виконується табличним методом в модулі Log2Table з використанням блоку пам'яті на основі компоненту 1-D Lookup Table. Розрядність входу адреси блоку пам'яті визначається виразом

$$log2\_Addr\_w = ceil(log2(log2\_table\_size)).$$

Вміст комірок пам'яті створюється за допомогою програмного коду в файлі set_log2_fixed_point_params.m [13]:

$$B = ceil(log2(log2\_table\_size)) + 1;$$

$$log2\_table = log2((2^{(B-1)} : (2^{(B)} - 1)) / 2^{(B-1)});$$

У залежності від кількості комірок компонент 1-D Lookup Table реалізується або на основі блочної пам'яті, або на основі логічних елементів ПЛІС.

Оскільки старший значущий біт аргументу на вході обчислювача завжди приймає значення 1, він не бере участі у формування адреси для блоку Log2Table. Компоненти ShiftLeft, ShiftRight та SelectAddr на основі значення виходу o_CLZ виділяють з аргументу обчислювача $log2\_Addr\_w$ бітів дробної частини $X_m$ (що йдуть одразу після старшого значущого біта аргументу обчислювача), формуючи значення адреси для блоку Log2Table.

На рис.2. наведена будова блоку CLZ, що реалізує алгоритм, описаний в (11). Схема блоку CLZ містить 7 блоків CLZ_STAGE з номерами від 6 до 0, що виконують обчислення на етапах алгоритму.

Будова блоку CLZ_STAGE6 наведена на рис.3. Будова інших блоків аналогічна. Компонент HighPart виділяє старшу половину бітів входу i_WORD. Компонент LowPart виділяє молодшу половину бітів входу i_WORD. Компонент Bit Reduce (OR) виконує логічну операцію АБО над бітами свого входу і разом з елементом інверсії використовується для перевірки виходу HighPart на нульове значення. Компоненти Mux0, Mux1 реалізують мультиплексори.

## V. Оцінка помилки обчислень та апаратурних витрат

Для оцінки точності запропонованого обчислювача LOG2_FIXED_POINT створено тестовий стенд у вигляді моделі Simulink®, будову якої наведено на рис. 4.

Вхідні дані формує блок генерації псевдовипадкових додатних чисел URNG з рівномірним законом розподілу. Цифрові коди згенеровані блоком URNG приводяться до формату на вході i_DATA обчислювача за допомогою блоків Scale та CastToFixedPoint. У якості еталонного значення функції $\log_2(\cdot)$ використаний блок Simulink для обчислення логарифму від аргументу в форматі з плаваючою комою подвійної точності (тип *double*).

Було досліджено 3 конфігурації запропонованого обчислювача: *рішення 1*, *рішення 2* та *рішення 3* з розміром блоку пам'яті Log2Table рівним 32, 128 та 1024 комірки відповідно.

Автори роботи виконали порівняння помилки обчислень та апаратурних витрат трьох зазначених рішень з обчислювачами логарифму на основі табличного підходу (LUT), ряду Тейлора (Taylor) та алгоритму повороту координат (CORDIC), що описані в [3]. Для досягнення однакових умов порівняння, значення параметру SQNR було обрано рівним 65 дБ, як і в роботі [3], що відповідає розрядності входу та виходу обчислювача 10 біт з дробовою частиною 9 біт.

Оцінка помилки обчислень виконувалась шляхом симуляції в Simulink® моделі наведеної на рис. 4 для одного мільйону випадковим чином згенерованих значень на вході обчислювача. Для верифікації запропонованого обчислювача на рівні регістрових передач, з моделі, наведеної на рис. 4, з використанням інструменту HDL Verifier було створено тестовий стенд (тестбенч) на мові Verilog (файл LOG2_FIXED_POINT_tb.v, джерело [13]) та шляхом симуляції в ModelSim ASE перевірено відсутність помилок. Ліцензію для ModelSim ASE отримано в рамках університетської програми Intel FPGA. Для

DOI: 10.20535/2523-4455.mea.205929

симуляції на рівні регістрових передач дані на вході обчислювача та очікувані дані на виході обчислювача взято із результатів моделювання в Simulink.

Для порівняння за апаратурними витратами при реалізації запропонованих рішень було використано мікросхему ПЛІС Stratix IV типу FPGA, що є найближчим еквівалентом Virtex-6, який застосовано в [3]. Обидві серії мікросхем ПЛІС створені за технологічним процесом 40 нм, однак для Virtex-6 розмір блоків вбудованої пам'яті складає 36 Кбіт на блок, а для Stratix IV розмір блоків вбудованої пам'яті складає 9 Кбіт та блок. Цю обставину варто брати до уваги при порівнянні за апаратурними ресурсами.

З метою отримання апаратних реалізацій запропонованих конфігурацій обчислювача, з розробленої Simulink моделі, з використанням інструменту HDL Coder для обраних параметрів, було синтезовано вихідний код рішень на мові Verilog. Синтез конфігураційного файлу для ПЛІС виконано в САПР Quartus Prime (ліцензію отримано в рамках університетської програми Intel FPGA).

Додатково, у вигляді Simulink моделі було реалізовано обчислювач логарифму за основою 2 для даних в форматі з плаваючою комою одинарної точності (файл log2_float_single.slx, джерело [13]) та створено його апаратну реалізацію з використанням інструменту HDL Coder.

Значення помилки обчислень та апаратурних витрат для описаних вище обчислювачів наведено в табл. 1. З таблиці видно, що запропоноване *рішення 3* забезпечує найменшу помилку обчислень для даних з фіксованою комою.

У порівнянні з існуючою CORDIC реалізацією усі запропоновані рішення забезпечують менші апаратурні витрати логічних елементів і вищу максимально можливу тактову частоту. *Рішення 2* і *3* забезпечують меншу помилку обчислень. Зокрема, вони не використовують перемножувачі, що є суттєвою перевагою, оскільки апаратні перемножувачі у вигляді DSP блоків є обмеженим ресурсом FPGA мікросхем, а реалізація перемножувачів на логічних елементах спричиняє стрімке зростання апаратурних витрат і зменшення максимально можливої тактової частоти. *Рішення 3* додатково використовує один блок вбудованої пам'яті обсягом 9 Кбіт. Такі незначні витрати вбудованої пам'яті можуть бути прийнятним компромісом враховуючи досягнуті переваги.

У порівнянні з існуючою Taylor реалізацією усі три запропоновані рішення забезпечують менші апаратурні витрати логічних елементів, вищу максимально можливу тактову частоту і не використовують DSP блоки. Реалізація Taylor споживає 7 DSP блоків, що є суттєвим значенням. При цьому запропоноване *рішення 3* забезпечує меншу помилку обчислень за рахунок збільшення апаратурних витрат на один блок вбудованої пам'яті обсягом 9 Кбіт, що є незначною величиною.

У порівнянні з існуючою LUT реалізацією запропоноване *рішення 3* забезпечує меншу помилку обчислень і потребує в 14,8 разів більше логічних елементів та у 52 рази менше вбудованої блочної пам'яті. Це робить доцільним використання запропонованого рішення в проектах, де необхідна низька помилка обчислень і доступний малий обсяг вбудованої пам'яті.

Загалом, перевагами запропонованого обчислювача є незначне споживання логічних елементів, відсутність перемножувачів і високе значення максимально можливої тактової частоти. Також перевагою є відсутність необхідності у вбудованій блочній пам'яті для малих значень параметру *log2_table_size*. У випадку доступності одного блоку вбудованої пам'яті обсягом 9 Кбіт, запропоноване рішення дозволяє досягти найменшої помилки обчислень для даних в форматі з фіксованою комою у порівнянні з аналогами. Конфігурованість дозволяє налаштувати запропонований обчислювач у залежності від допустимого значення помилки обчислень і наявних апаратурних ресурсів.

## Висновки

В роботі запропоновано математичну модель та структуру потокового обчислювача логарифму за основою 2 для даних з фіксованою комою, з використанням операції підрахунку провідних нульових бітів аргументу і табличного підходу.

Створено MATLAB® Simulink® модель запропонованого обчислювача. З використанням інструменту HDL Coder зі створеної моделі синтезовано апаратну реалізацію обчислювача на мові Verilog та тестовий стенд для верифікації на рівні регістрових передач. Розроблені моделі та вихідні коди доступні за посиланням [13].

Результати порівняння з аналогами свідчать, що запропоноване рішення в цілому, залежно від конфігурації, дозволяє досягти компромісних показників у відношенні меншої помилки обчислень та (або) нижчих апаратурних витрат.

Таблиця 1 Помилка обчислень та апаратурні витрати у порівнянні з аналогами

| **Обчислювач** | **Помилка** | **Апаратурні ресурси** | | | | |
|---|---|---|---|---|---|---|
| | | ***Лог. елем., ALM/Slice*** | ***DSP блоки*** | ***Вбудована RAM, Кбіт*** | ***Латентність (Latency), такти*** | ***Максимальна тактова частота, МГц*** |
| CORDIC | $10^{-2}$ | 1387 | 2 | 0 | 21 | 194 |
| Taylor | $3{,}5\cdot10^{-3}$ | 205 | 7 | 0 | 9 | 151 |
| LUT | $2\cdot10^{-3}$ | 6 | 0 | 468 | 2 | 324 |
| Float Single | $10^{-6}$ | 603 | 13 | 45 | 6 | 160 |
| Рішення 1 | $45\cdot10^{-3}$ | 91 | 0 | 0 | 9 | 270 |
| Рішення 2 | $9\cdot10^{-3}$ | 92 | 0 | 0 | 9 | 280 |
| Рішення 3 | $1\cdot10^{-3}$ | 89 | 0 | 9 | 9 | 240 |

DOI: 10.20535/2523-4455.mea.205929

ПЕРЕЛІК ПОСИЛАНЬ


[1] Demmel, F. (2009). Practical Aspects of Design and Application of Direction-Finding Systems. In Classical and Modern Direction-of-Arrival Estimation (pp. 53–92). Elsevier. **DOI:** 10.1016/B978-0-12-374524-8.00008-8

[2] C. Park, D. Kim, "The Fast Correlative Interferometer Direction Finder using I/Q Demodulator," 2006 Asia-Pacific Conference on Communications, Busan, 2006, pp. 1-5, **DOI:** 10.1109/APCC.2006.255915

[3] A. M. Mansour, A. M. El-Sawy, M. S. Aziz, A. T. Sayed, "A New Hardware Implementation of Base 2 Logarithm for FPGA," International Journal of Signal Processing Systems, vol. 3, no. 2, pp. 177-182, Dec. 2015. **DOI:** 10.12720/ijsps.3.2.177-182

[4] "Implement Fixed-Point Log2 Using Lookup Table," 2020. [Online]. Available: https://www.mathworks.com/help/fixedpoint/ug/implement-fixed-point-log2-using-lookup-table.html [Accessed 16, May 2020]

[5] H. Hassler, N. Takagi, "Function evaluation by table look-up and addition," in Proceedings of the 12th Symposium on Computer Arithmetic, Bath, UK, 1995, pp. 10-16, **DOI:** 10.1109/ARITH.1995.465382

[6] D.D. Sarma, D. W. Matula, "Measuring the accuracy of ROM reciprocal tables," in IEEE Transactions on Computers, vol. 43, no. 8, pp. 932-940, Aug. 1994, **DOI:** 10.1109/12.295855

[7] D. K. Kostopoulos, "An algorithm for the computation of binary logarithms," IEEE Transactions on Computers, vol. 40, no. 11, Nov. 1991. **DOI:** 10.1109/12.102831


**Надійшла до редакції 21 березня 2020 року**


[8] "A way to calculate logarithm with a base of two," 2019. [Online]. Available: https://habr.com/ru/post/469327/ [Accessed 16, May 2020]

[9] P. K. Meher, J. Valls, T. Juang, K. Sridharan, K. Maharatna, "50 Years of CORDIC: Algorithms, Architectures, and Applications," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 56, no. 9, pp. 1893-1907, Sept. 2009, **DOI:** 10.1109/TCSI.2009.2025803

[10] L. Bangqiang, H. Ling, Y. Xiao, "Base-N logarithm implementation on FPGA for the data with random decimal point positions," 2013 IEEE 9th International Colloquium on Signal Processing and its Applications, Kuala Lumpur, 2013, pp. 17-20, **DOI:** 10.1109/CSPA.2013.6530006

[11] R. D. Yershov, "A Scalable VHDL-Implementation Technique of the Priority Encoder Structure into FPGA," 2018 IEEE 38th International Conference on Electronics and Nanotechnology (ELNANO), Kiev, 2018, pp. 727-732, **DOI:** 10.1109/ELNANO.2018.8477465

[12] H. Warren, Hacker's Delight, Addison-Wesley, 2012. **ISBN:** 978-0-321-84268-8

[13] I. Korotkyi, "Source code for the proposed Log2 hardware accelerator," 2020. [Online]. Available: https://github.com/KorotkiyEugene/log_hdl_simulink/

UDC 004.31

DOI: 10.20535/2523-4455.mea.205929

# Hardware Implementation of Streaming Logarithm Computing Unit for Fixed-Point Data

Yu. O. Hordiienko[f], ORCID 0000-0001-7127-7232
A. Yu. Varfolomieiev, PhD Assoc.Prof., ORCID 0000-0002-6990-7140
Ie. V. Korotkyi[s], PhD Assoc.Prof., ORCID 0000-0001-8302-4873
Department of design of electronic digital equipment keoa.kpi.ua
National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute" kpi.ua
Kyiv, Ukraine

***Abstract*—The work aims to create hardware implementation of the streaming computing unit for logarithm calculation in fixed-point. Logarithms are widely used in telecommunications, particularly in radio intelligence to convert power spectrum values to decibels for further processing of spectrum data, e.g. for radio signals detection, range-finding, or direction-finding. For spectrum analysis of high sampling rate wideband signals, it is expedient to utilize hardware computing units for streaming logarithm calculation, implemented inside FPGA or ASIC chips. The market offers a large amount of IP cores for logarithm calculation in floating point. Floating-point calculation units offer a high dynamic range but also consume a large number of hardware resources that could diminish the maximum clock frequency of devices. In the proposed work, different approaches for logarithm calculating are considered, including CORDIC, Taylor series, and table-based methods. Authors proposed a mathematical model and architecture of streaming computing unit for base 2 logarithm calculation in fixed-point that can be easily adapted to any other base, simply multiplying the result by a constant. The proposed computing unit utilizes a table-based approach and counting leading zeroes in the argument. Based on the mathematical model, the high-level computational model in MATLAB® Simulink® was created. All the components of the mentioned model are compatible with HDL Coder. The proposed MATLAB® Simulink® model is parameterizable, one can set word and fraction width for input/output data, and memory size for table-based part. Using HDL Coder, Verilog HDL implementation for the proposed logarithm computing unit was synthesized. Utilizing HDL Verifier authors created the testbench in Verilog language for the verification of created computing unit on RTL level of abstraction based on reference data collected during simulation in Simulink. Running generated testbench in ModelSim simulator for one million clock cycles proved that there are no differences in operation between the Simulink Model and generated HDL design. The authors were synthesized HDL implementation of the created computing unit in Quartus Prime for the Stratix IV FPGA chip to evaluate the hardware cost of the proposed solution. The developed logarithm calculation unit was compared to the existing CORDIC, Taylor series, and table-based implementations in terms of calculation error and hardware costs. Additionally, for comparison purposes the authors created a hardware implementation for the base 2 logarithm calculation unit in a single-precision floating-point. During the evaluation of the calculation error, the double-precision floating-point logarithm computing unit from Simulink® was chosen as the source for reference results. The comparison showed that created computing unit provides less calculation error compared to the existing fixed-point solutions, requires fewer hardware resources for implementation and can operate on higher clock frequencies. All the created models and source codes are open for utilization and can be downloaded from GitHub.**

***Keywords — computing unit; logarithm; fixed-point; Verilog; FPGA; programmable logic***