УДК 004.383

СЕРГІЄНКО А.М.,
СЕРГІЄНКО П.А.

РЕАЛІЗАЦІЯ ФУНКЦІЇ КВАДРАТНОГО КОРЕНЯ У ПЛІС

В статье предложен модифицированный алгоритм извлечения квадратного корня основанный на том, что его первые итерации вычисляются табличным способом, за счет чего уменьшаются аппаратные затраты и уменьшается время вычислений. Алгоритм реализован в Web-приложении, которое генерирует VHDL-модели блоков с заданными параметрами, реализуемые в ПЛИС.

A modified square root algorithm is proposed. In the algorithm the first iterations are calculated by the table functions, that minimizes both hardware volume and speed of calculations. The algorithm is implemented in the parametrized Web application, which generates the VHDL entities for FPGA projects.

1. Вступ

Технологія розробки застосувань на програмованих логічних інтегральних схемах (ПЛІС) основана на описі алгоритму такою мовою, як VHDL та автоматичній трансляції цього опису в опис на рівні логічних таблиць (ЛТ), та інших функціональних компонентів ПЛІС. Але елементарні функції в ПЛІС, як правило, реалізуються як окремі проекти чи замовлені віртуальні модулі, в яких при настроюванні вказується розрядність даних та іноді – внутрішня будова [1].

Відміна ПЛІС останнього покоління полягає у зростанні вхідної розрядності ЛТ з чотирьох до шести. Крім того, вбудовані блоки пам'яті об'ємом в десятки кілобіт також стали застосовуватися як ЛТ [2]. Тому усі напрацювання по побудові віртуальних модулів для реалізації елементарних функцій мають бути переглянуті з метою кращого використання особливостей нових ПЛІС. В даній статті досліджено побудову віртуального модуля для реалізації функції квадратного кореня в ПЛІС.

2. Алгоритми добування квадратного кореня

При виборі алгоритму квадратного кореня слід мати на увазі особливості будови ПЛІС, яка має ресурси ЛТ, мультиплексорів, суматорів, блоків множення, але не має блоків ділення. Функція кореня повинна мати регульовані точність обчислень та розрядність вхідних даних. Для свого швидкого виконання вона має бути реалізована як паралельна структура, що допускає конвеєрну реалізацію.

При визначенні апаратної складності алгоритму Θ_S, яка далі розраховуватиметься у кількості необхідних суматорів, приймається до уваги, що складність блоку множення у ПЛІС для розрядності операндів у межах 16-32 оцінюється як апаратні витрати двадцяти суматорів такої самої розрядності. Часова складність Θ_T алгоритму обчислюється як сумарна затримка комбінаційної схеми блоку, яка приведена до затримки суматора [3].

Далі розглядатимуться алгоритми добування квадратного кореня з оцінкою їх ефективності для 24-розрядних вхідних даних та результатів з фіксованою комою, які можуть претендувати на реалізацію в ПЛІС.

2.1. Поліноміальна апроксимація

Традиційне рішення розрахунку елементарної функції – це обчислення полінома, який є, наприклад, рядом Тейлора, як наступний [4].

$$\sqrt{1+x} = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3 - \frac{5}{128}x^4 + \frac{7}{256}x^5 \ldots .$$

При цьому неможливо досягти похибки обчислення меншої за 0,2%, якщо $x \subset (0;1)$. Крім того, алгоритм потребує виконання багатьох множень. Тому він неприйнятний для реалізації у ПЛІС, хоча, він може згодитись при кусково-поліноміальній апроксимації.

2.2. Ітеративний алгоритм

Відомий наступний ітеративний алгоритм на основі формули Ньютона-Рафсона, який не потребує операцій ділення [4]:

$$y_{i+1} = 1{,}5\, y_i - 0{,}5\, x\, y_i^2 \quad (i = 0, 1, \ldots, n),$$

де $y_0 \approx 1/\sqrt{x}$ – наближене значення функції, $\sqrt{x} \approx x\, y_n$. Кожна наступна ітерація алгоритму приблизно подвоює кількість правильних розрядів результату. Тому для одержання коректного 24-розрядного результату необхідно виконати $n = 2$ ітерації алгоритму та одержати значення y_0 з

таблиці з семирозрядним входом адреси або одну ітерацію, якщо таблиця має 13-розрядний вхід. Часова Θ_T та апаратна Θ_S оцінки складності цього алгоритму приведені в табл.1. При розрахунку Θ_S вважалось, що згадані таблиці реалізовані у ПЛІС як постійна пам'ять, яка має приблизну складність у 2 та 60 суматорів, відповідно.

1.3. Алгоритм "цифра за цифрою"

Алгоритм, що належить до класу "цифра за цифрою" дає змогу одержувати одну точну цифру результату на кожному кроці. Відомий алгоритм Волдера для обчислення $\sqrt{x}$, але більш конструктивним є алгоритм [5,6]:

$y_0 = x;\ x_0 = x;\ m = 0; f = 0;$
for $(i = 1; i < n; i = i + 1)$ {
  $t = x_i + 2^{-m} x_i;$
  $u = t + 2^{-m} t$;
  if $(u \ge 1)$ {
    $f = 1;$
    $x_{i+1} = x_i;$
    $y_{i+1} = y_i;$
  }
  else {
    $x_{i+1} = u;$
    $y_{i+1} = y_i + 2^{-m} y_i;$
  }
  if $(f == 1)$ $m = m + 1;$
}

Під час виконання алгоритму знаходяться псевдоцифри m результату, причому спочатку, коли $m = 0$, виконується нормалізація операнду x_i з корекцією часткового результату y_i. Потім $m = 1, 2,\ldots, n$ і в процесі конвергенції x_i прямує до одиниці, а y_i – до $\sqrt{x}$.

Для реалізації алгоритму в ПЛІС необхідно блоки нормалізації x_i , зсуву y_i та $3n$ суматорів. Цей алгоритм варто реалізувати у блоці, який виконує кілька функцій за алгоритмом "цифра за цифрою", такому як [7].

2.4. Алгоритм на основі зсуву та віднімання

Метою алгоритму є одержання наближення двійкового числа $B_i = 0,b_1b_2...b_i0\ldots0$ до значення $\sqrt{x}$. Він оснований на перевірці знаку остачі $R_i = x - B_i^2$, у результаті чого вибирається чергова цифра b_i. Нехай вважається, що $b_i = 1$, тоді

$$R_i = x - (B_{i-1} + 2^{-i})^2 = R_{i-1} - 2^{-(i-1)}(0,b_1...b_{i-1}01),$$

а якщо виявиться, що $R_i \le 0$, то

$$R_i = R_{i-1}\ ;\ b_i = 0\ ;$$

і шукається наступна цифра b_{i+1} .

Алгоритм оснований на відніманні від остачі R_i зсунутого вправо наближення B_i . При його реалізації в ПЛІС він має n ступенів, кожен з яких має суматор в режимі віднімання та мультиплексор для вибору R_i, який керується знаковим розрядом суматора [8].

Даний алгоритм називають алгоритмом з відновленням остачі. Використовують також алгоритм без відновлення остачі, коли R_i є знакозмінним числом. Але він не має суттєвих переваг, оскільки в ньому використовується знакозмінний суматор, який за складністю такий самий, як суматор з мультиплексором [1].

Табл. 1. Витрати для обчислення $\sqrt{x}$

Алгоритм	Блоків множення	Θ_S	Θ_T
Ітеративний, 1 ітерація	2	102	7
Ітеративний, 2 ітерації	4	86	13
"Цифра за цифрою"	–	72	48
Зі зсувом та відніманням	–	24	24
Модифікований зі зсувом та відніманням	–	22	21

Отже, аналізуючи параметри розглянутих алгоритмів у табл.1, можна зробити висновок, що алгоритм на основі зсуву та віднімання реалізується у ПЛІС з мінімальними апаратними витратами. Якщо його виконати у конвейерному блоці, то такий блок матиме високу пропускну спроможність з періодом тактового інтервалу, що дорівнює затримці одного суматора та мультиплексора.

2.5. Модифікований алгоритм зсуву та віднімання

Одна i-та ітерація алгоритму зсуву та віднімання полягає у наступних обчисленнях:

$r_i = R_i - P_i$;
$b_i =$ **not** $\mathrm{sign}(r_i)$;
$B_{i+1} = (B_i, b_i);$
$P_{i+1} = 2^{-i+1}\,(B_{i+1}, 01);$
if $(b_i == 1)$
  $R_{i+1} = r_i;$
else
  $R_{i+1} = R_i;$

де sign(x) = 1 при $x < 0$, інакше – 0; $i = 1,\ldots,n$; (a,b) – операція конкатенації.

Недолік алгоритму полягає у великій кількості ітерацій, яка дорівнює числу розрядів результату. Помітно скоротити цю кількість можна, якщо перші k ітерацій не виконувати, а результати k – ї ітерації знаходити за допомогою табличних функцій.

Такими табличними функціями є

$$B_k = f_B(x_k) \approx \sqrt{x_k} \;; \qquad R_k^{'} = f_R(x_k),$$

де x_k – $2k$ старших розряди вхідного данного x, причому

$$R_k = R_k^{'} + x - x_k.$$

Наприклад, на рис.1 показана функціональна схема, яка виконує модифікований алгоритм для 12-розрядних вхідних даних, $n = 6$ – розрядного результату та $k = 3$. При цьому табличні функції f_B та f_R реалізовані у блоках постійної пам'яті ROMB та ROMR, відповідно.

Завдяки застосуванню блоків постійної пам'яті, у даному випадку вдалось зменшити кількість ітерацій з шести до чотирьох, перша з яких – це звертання до цих блоків. Відповідно, схема добування кореня стала меншою на три суматори та три мультиплексори.

Недоліком модифікованого алгоритму є те, що блоки постійної пам'яті повинні мати $2k$-розрядні адресні входи. Через це вони мають значний об'єм при великому k.

Проте, цей алгоритм доцільно використовувати в сучасних ПЛІС при $k \le 6$. Так, при $k = 3$ для n = 24-розрядних даних параметри блоку квадратного кореня наведені в табл.1., тобто з застосуванням цього алгоритму параметри блоку квадратного кореня покращуються.

3. Експериментальні результати

Модифікований алгоритм добування квадратного кореня було реалізовано у Web-додатку, призначеному для генерації віртуальних модулів, описаних мовою VHDL, який впроваджено на сайті www.kanyevsky.kpi.ua. У ньому організовано ввід таких параметрів, як розрядність вхідних та вихідних n даних, розрядність $2k$ адрес блоків постійної пам'яті, наявність вхідного та вихідного регістрів, побудова повністю конвеєризованої чи частково конвеєризованої схеми, тобто з n–k або з](n–k)/2[ступенями конвеєра. При цьому користувачу даються оцінки апаратних витрат модуля та максимальної тактової частоти. Наприклад, для 12-розрядних вхідних даних, 6-розрядного результату, $k = 3$ при відсутності конвеєризації генерується наступний поведінковий опис модуля квадратного кореня:

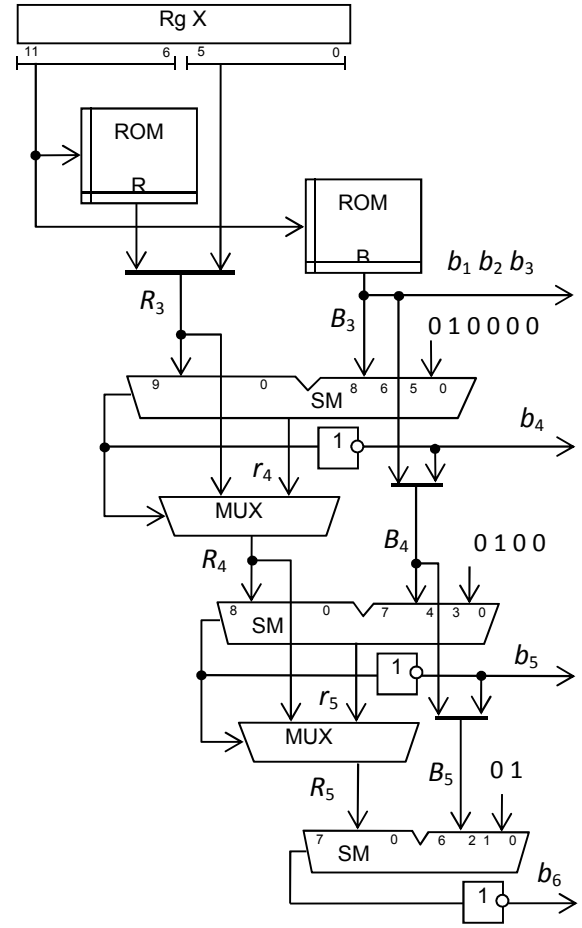

Рис.1. Схема блоку добування квадратного кореня

```
process(CLK) begin
 if rising_edge(CLK) then
   x<=DI;
   DO<=qhr(6)(12 downto 7);
 end if;
end process;

xh<=x(11 downto 6);
qr(3)<= xqh_6(conv_integer(xh))&x(5 downto 0)&"00000";
pr(3)<= "0"&qh_6(conv_integer(xh))&"01"&z(8 downto 0);
qhr(3)<= qh_6(conv_integer(xh))&z(9 downto 0);

Stages:for i in 3 to 5 generate
 rr(i)<=qr(i) - pr(i);
 biti(i) <= '1' when rr(i)(14)='0' else '0';
 qhr(i+1)<=qhr(i)(12 downto 13-i)&biti(i) & z(11-i downto 0);
 pr(i+1)<='0' & qhr(i+1)(12 downto 12-i) &"01"&z(10-i downto 0);
 qr(i+1)<=SHL(rr(i),"1") when rr(i)(14)='0' else SHL(qr(i),"1");
end generate;
```

У ньому масив xqh_6 описує постійну пам'ять для функції f_R, а масив qh_6 – для функції f_B. Цей опис повністю відповідає схемі на рис.1.

Після синтезу та розміщення модулів блоку квадратного кореня у ПЛІС Xilinx Spartan-6 було одержано ряд характеристик апаратних витрат (рис.2) у кількості ЛТ та максимальної тактової частоти, МГц (рис.3) для різних значень k при вхідній розрядності 24 та розрядності результату n = 12. Розглядались модулі у вигляді комбінаційної схеми, повністю конвеєризовані та частково конвеєризовані модулі.

Аналіз цих характеристик показує, що оптимальним за апаратними витратами та за швидкодією є блок з k = 3. Блок з k = 5 має помітно менші апаратні витрати за рахунок застосування окремого блоку пам'яті BlockRAM для реалізації таблиць. Використання значень $k > 6$ недоцільне через значне зростання необхідного об'єму пам'яті та зменшення тактової частоти.

На рис. 4, 5 показані характеристики блоків для k = 3 в залежності від вихідної розрядності n, яка дорівнює вхідній розрядності. Там же для порівняння приведені характеристики блоків, які пропонуються фірмою Xilinx.

Отже, синтезовані модулі для добування квадратного кореня мають помітно кращі показники за модулі, що пропонуються фірмою Xilinx.

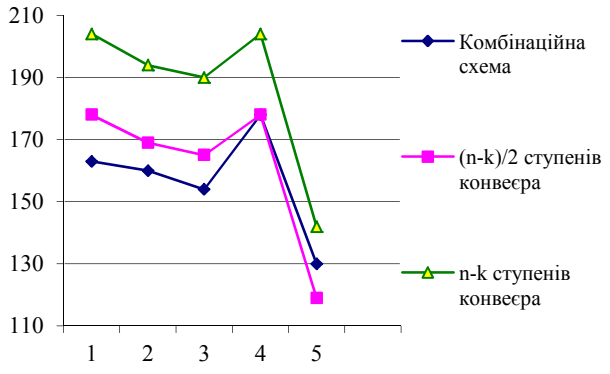

Рис.2. Апаратні витрати блоку $\sqrt{x}$ в залежності від k

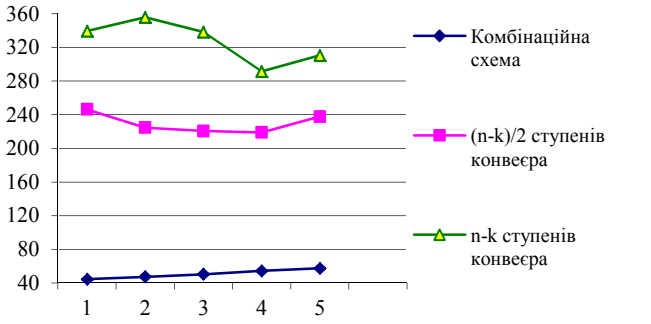

Рис.3. Максимальна тактова частота блоку $\sqrt{x}$ в залежності від k

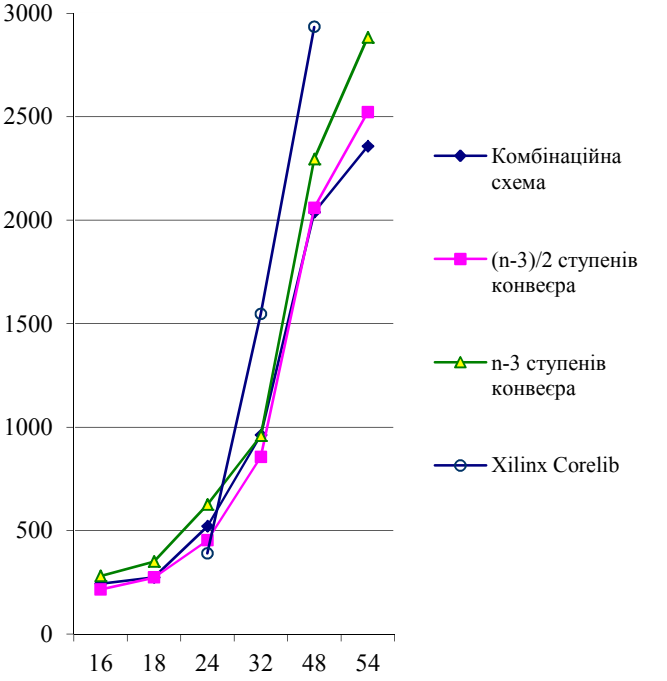

Рис.4. Апаратні витрати блоку $\sqrt{x}$ в залежності від n

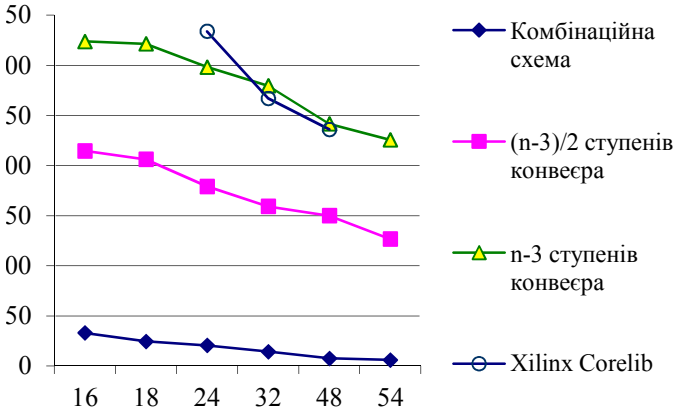

Рис.5. Максимальна тактова частота блоку $\sqrt{x}$ в залежності від n

Модулі обчислення $\sqrt{x}$ часто використовуються у блоках для обчислень з плаваючою комою, для яких критичною є латентна затримка між входом та виходом. Цей параметр дорівнює $T_Л =](n-k)/2[/f_C$ для модулів з частковою конвеєризацією та $T_Л = (n-k)/f_C$ – для модулів з повною конвеєризацією, де f_C – тактова частота. Так, для $n = 24$ і $k = 3$ ця затримка дорівнює $T_Л = 61{,}4$ нс і 70,4 нс, відповідно. Для порівняння, модуль фірми Xilinx має $T_Л = 71{,}9$ нс. Отже, з точки зору зменшення латентної затримки доцільно застосовувати модулі з частковою конвеєризацією, які до того ж мають менші апаратні витрати.

Висновки

Проаналізовано алгоритми обчислення квадратного кореня і вибрано за основу алгоритм зі зсувом та відніманням. Цей алгоритм було модифіковано таким чином, що його перші k ітерацій обчислюються табличним способом. За рахунок цього дещо зменшуються апаратні витрати та час виконання алгоритму.

Модифікований алгоритм добування квадратного кореня було реалізовано у Web-додатку, який генерує модулі, описані мовою VHDL стилем для синтезу. Одержані характеристики ряду згенерованих модулів свідчать про те, що ці модулі є кращими за аналогічні модулі, що пропонуються фірмою-виробником ПЛІС.

Встановлено, що оптимальними є модулі, для яких $k = 3$. Також встановлено, що задля зменшення латентної затримки та апаратних витрат доцільно застосовувати модулі з частковою конвеєризацією.

Список посилань

1. Deshamps J.-P., Bioul G.J.A., Sutter G.D. Synthesis of Arithmetic Circuits: FPGA, ASIC, and Embedded Systems. –John Wiley & Sons, Inc. –2006. – 556 p.
2. FPGA Architecture. White Paper WP-01003-1.0. – Altera Corp. – 2006. – 14 p. Available at http://www.altera.com/
3. Сергиенко А.М., Симоненко В.П. Отображение периодических алгоритмов в программируемые логические интегральные схемы // Электронное моделирование. –2007. –Т.29. –№2. –С. 49–61.
5. Люстерник Л.А., Червоненкис О.А., Янпольский А.Р. Математический анализ. Вычисление элементарных функций. –М.: ГИЗ физ.-мат. лит. – 1963. – 247 с.
6. Chen T.C. Automatic computations of exponentials, logarithms, ratios and square roots. IBM J. Res. and Develop. –1972. –№4. – P. 380-388.
7. Благовещенский Ю.В., Теслер Г.С. Вычисление элементарных функций на ЭВМ. – Киев: Техніка. –1977. – 208 с.
8. Каневский Ю.С., Куц Н.Е., Лозинский В.И., Сергиенко А.М. Устройство для вычисления элементарных функций. А.С. СССР N 1141399, Б.И. N 7, 1985.
9. Каневский Ю.С. Компьютерная арифметика. Конспект лекций. – Киев: Диасофт. – 1994. – 234 с.