УДК 004.25, 004.383

І. І. Лопіт
Національний університет “Львівська політехніка”,
кафедра безпеки інформаційних технологій

ОПТИМІЗАЦІЯ ВІДОБРАЖЕННЯ ПРОГРАМНИХ МОДЕЛЕЙ ПОСТІЙНОЇ ПАМ’ЯТІ В АРХІТЕКТУРУ ПЛІС МЕТОДОМ СЛОВНИКА

Розглянуто питання відображення програмних моделей пристроїв постійної пам’яті, що є компонентами спеціалізованих процесорів, в архітектуру програмованих логічних інтегральних схем. Проаналізовано існуючі підходи до компресії даних у пристроях постійної пам’яті і, як альтернативу до них, запропоновано новий підхід, який дає змогу досягти високого рівня компресії і раціональніше використовувати ресурси ПЛІС.

Ключові слова: пам’ять з довільним доступом, ПЛІС, спеціалізовані процесори.

The problem of mapping the read-only memory program models, being the components of the application-specific processors, into the programmable logical integral circuit architecture were considered. The existing approaches to the data compression in the read-only memory devices are analyzed and, alternatively, a new approach is suggested allowing a high degree of compression to be achieved and the FPGA resources to be used more rationally.

Key words: FPGA, application-specific processors program models, read-only memory, data compression.

Вступ

За сучасного стану комп’ютерної елементної бази продуктивність комп’ютерних систем підвищують переважно екстенсивним методом – збільшенням кількості ядер універсальних процесорів та підвищенням частоти їх роботи. До того ж підхід застосування універсальних процесорів для досягнення високих показників продуктивності має принципові недоліки, головними з яких є висока споживана потужність і низька ефективність використання обладнання. Для уникнення цих недоліків створюють комп’ютерні системи із спеціалізованими процесорами, однак вони є ефективними лише для вузьких класів алгоритмів, а їх побудова потребує значних зусиль і ресурсів. Один з варіантів вирішення цієї проблеми полягає в генеруванні програмних моделей спеціалізованих процесорів (ПМСП) з високорівневого подання алгоритму їх роботи та реалізації в програмовних логічних інтегральних схемах (ПЛІС). У результаті отримують спеціалізовані процесори, які використовують як прискорювачі комп’ютерних систем, функції яких можна змінювати реконфігуруванням ПЛІС і створенням у ній іншого спеціалізованого процесора (СП).

Одним з часто застосовуваних компонентів ПМСП є постійний запам’ятовувальний пристрій (ПЗП, англ. ROM – *Read-Only Memory*). Його використовують для збереження константних величин, команд, реалізації табличних і таблично-алгоритмічних обчислювальних пристроїв тощо. Одним із проблемних питань реалізації ПЗП в ПЛІС є нераціональне використання її ресурсів, що зумовлено надлишковістю репрезентації даних. Як варіант, вирішення цієї проблеми застосовують за допомогою компресії даних, що містить ПЗП.

Компресія даних – це набір правил для подання цих даних в іншому вигляді, в якому вони займають менший об’єм. Фактично компресію даних здійснюють їх перекодуванням. Алгоритми компресії даних без втрати розділяють на два основні класи:

- одування даних з використанням словника;
- алгоритми ентропійного кодування даних.

Найпоширенішими алгоритмами у класі кодування із словником є алгоритм Лемпеля-Зіва-Велча (LZW) [1] та його варіації. У класі ентропійного кодування – це алгоритми на основі коду Хаффмана [2].

Аналіз досліджень та публікацій

Алгоритми Лемпеля-Зіва-Велча описані в [1], Хаффмана – в [2]. У [3] продемонстрована реалізація в ПЛІС функції декомпресії даних у вбудованих блоках пам'яті за алгоритмом LZW. Апаратний декодер Хаффмана, реалізований у ПЛІС, описаний у [4].

Постановка проблеми

Сучасні алгоритми компресії, описані в [1, 2], демонструють високий коефіцієнт компресії, проте складність і низька швидкодія їх апаратної реалізації обмежує їх широке застосування для компресії даних в ПЗП. Аналізуючи варіанти апаратної реалізації декодерів [3, 4], було виявлені недоліки в їх характеристиках, а саме:

- невисока тактова частота роботи;
- істотна затримка доступу до наступної комірки пам'яті, для [3] – 6 тактів, для [4] – 4 такти синхросигналу.

Вказані недоліки істотно впливають на ефективність ПМСП, в яких ПЗП є компонентом. Пошук рішень, позбавлених цих недоліків, є предметом наукового дослідження, висвітленого у цій роботі.

Під час генерування ПМСП з мови програмування високого рівня вміст ПЗП є відомий наперед, а отже, система генерування може здійснити усі необхідні перетворення над вмістом у момент синтезу програмної моделі. Одним з найпростіших перетворень над даними в апаратній реалізації є їх перестановка. Вона не потребує додаткових апаратних затрат і може бути реалізована зміною порядку під'єднання портів. Використовуючи ці особливості, автор запропонував алгоритм компресії груп даних з використанням методу словника. Цей алгоритм умовно можна розділити на дві частини: компресію простими для апаратної реалізації методами і пошук оптимальних груп даних, над якими виконуються ці методи.

1. Компресія даних методом словника

Метод компресії даних з використанням словника передбачає кодування набору значень фіксованої довжини у закодований набір значень і таблицю ключів для їх відновлення. Схематично апаратну реалізацію цього методу показано на рис. 1.

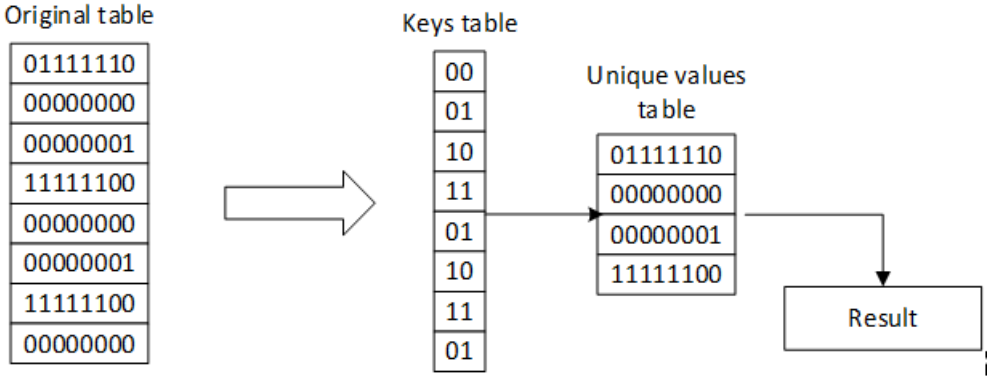

Рис. 1. Варіант апаратної реалізації компресії даних методом словника

Принцип роботи пристрою такий: спочатку відбувається доступ до ключа, потім за його допомогою здійснюється доступ до унікального значення. Час очікування на нове значення становить два такти синхросигналу.

Коефіцієнт компресії для цього методу можна обчислити за такою формулою:

$$h = \frac{[\log_2 u]\cdot h + u\cdot w}{h\cdot w}, \quad (1)$$

де u – кількість унікальних значень у первинній таблиці; w – розрядність слова даних у цій таблиці; h – її висота.

Цей метод характеризується порівняно низьким рівнем компресії, проте його апаратна реалізація проста і забезпечує високу швидкодію. Дослідження показали, що у випадку, коли ПЗП реалізований за допомогою вбудованих блоків пам'яті ПЛІС, тактова частота декомпресора для ПЛІС сімейства *Altera Cyclone V* становить близько 240 МГц.

2. Алгоритм групування даних

Для підвищення ефективності компресії методом словника автор запропонував алгоритм групування даних за допомогою виконання пошуку оптимальних груп з використанням перестановок. Одиницею перестановки є стовпець у блоці пам'яті.

Вхідні дані алгоритму: вміст блока пам'яті для компресії; w – розрядність оригінального блока пам'яті; h – кількість комірок у блоці пам'яті, wc – розрядність підблока після компресії, max_c – максимальне значення кількості можливих комбінацій на кроці; min_c – мінімальне значення кількості можливих комбінацій на кроці.

Критерії алгоритму: "Швидкість" – перевага надається швидкості виконання компресії, "Компресія" – перевага надається максимальному коефіцієнту стиснення.

Вихідні дані алгоритму: набір підблоків пам'яті після компресії.

Алгоритм полягає у виконанні таких кроків:

1. Якщо w/wc не є цілим числом, доповнити оригінальний блок пам'яті для компресії останнім стовпцем.
2. Створити множину стовпців R на основі оригінального блока пам'яті.
3. Згенерувати множину підкроків S. Якщо кількість $C_w^{wc} > max\_c$, то wc розкласти на прості множники, з яких сформувати множину S, в іншому випадку w стає єдиним елементом множини S.
4. Вибрати перший елемент s з множини S
5. Над елементом s виконати такі кроки:

(a) Якщо кількість $C_R^S < min\_c$, вибрати наступний простий множник і помножити на нього поточне значення. Перейти до кроку 5.

(b) Використовуючи C_R^S, сформувати усі можливі комбінації розміщення груп – P.

(c) На основі комбінацій P вибрати з R стовпці з відповідними індексами і сформувати множину можливих груп G.

(d) Для кожної групи g в G виконати алгоритм компресії.

(e) Виконати пошук оптимальної групи Go з множини G, отриманої на кроці 5(d). Якщо критерієм оптимізації є "швидкість", то використати жадібний алгоритм пошуку (крок 5(f)). Якщо критерієм оптимізації є "компресія", то використати розширений жадібний алгоритм пошуку (крок 5(g)).

(f) Жадібний алгоритм пошуку. Поки кількість груп не дорівнюватиме R/wc, шукати найменшу групу, для якої справедлива рівність $g \cap Go = \varnothing$. Знайдену групу додають до множини Go.

(g) Розширений жадібний алгоритм. Для кожної групи g у множині G виконати жадібний пошук (крок 5(f)) з винятком, коли замість першої ітерації найменшим елементом буде прийнятий поточний елемент g.

(h) Сформувати нову множину елементів-стовпців R, взявши за основу множину Go, отриману з кроку 5(f) або 5(g).

(i) Якщо S містить елементи, вибрати наступний s і перейти до кроку 5, в іншому випадку перейти до кроку 6.

6. Результатом роботи алгоритму є вміст останньої групи Go.
7. Кінець.

Цей алгоритм передбачає здійснення ітераційного пошуку оптимального розміщення груп з метою їх максимальної компресії. Кожну ітерацію можна розділити на дві основні частини: генерування можливих комбінацій розташувань елементів та пошук оптимального розташування у множині можливих комбінацій розташувань.

Генерування можливих комбінацій розташувань – це задача з області комбінаторики, яка має факторіальну складність. Кількість можливих розташувань можна обчислити за такою формулою:

$$C_n^k = \frac{n!}{k!(n-k)}. \qquad (2)$$

Наприклад, якщо пам'ять має n=100 стовпців, а розмір блока – k=2, то кількість можливих розміщень дорівнює 4950. Якщо k=4, то кількість можливих розміщень дорівнює 3921225. Якщо ж k=6, то кількість можливих розміщень дорівнює 1192052400. З цього можна зробити висновок, що цю задачу не можна розв'язувати методом повного перебору. Тому для досягнення поставлених цілей автор прийняв рішення використати алгоритми наближення.

Пошук у множині можливих комбінацій розташувань оптимального є частковим випадком завдання комівояжера. Ця задача належить до класу NP повних і має факторіальну складність. Вона може бути розв'язана методом гілок і меж, проте автор відмовився від використання цих методів, надавши перевагу жадібним алгоритмам, з огляду на їхню простоту і швидкодію.

Керування точністю/швидкістю алгоритму здійснюється за допомогою величин *max_c* і *min_c*. Вимоги до них такі: *min_c* повинна бути не більшою за *max_c*, але й не меншою за C_R^2. В іншому випадку алгоритм не матиме збіжності, а отже, триватиме вічно. Якщо встановлювати на кожному кроці *max_c* і *min_c* такими, що дорівнюють кількості комбінацій C_R^2, то отримаємо найшвидший варіант цього алгоритму.

3. Оцінка складності алгоритму групування даних

Однією з найголовніших характеристик будь-якого алгоритму є його складність. Розглянемо складність алгоритму групування даних у разі, коли значення *max_c* і *min_c* дорівнюють C_R^2.

Якщо на кожному кроці *max_c* і *min_c* дорівнюють C_R^2., то кількість комбінацій на кожному кроці дорівнює $\approx R^2/2$ з формули (2).

Складність жадібного алгоритму можна оцінити на основі його складових так:

пошук найменшої групи *g*. Складність пошуку лінійна, в гіршому випадку необхідно $R/2$ операцій, щоб знайти мінімальну групу;

забезпечити виконання умови $g \cap Go = \varnothing$: здійснити маркування груп *g*, які містять використані елементи в множині *Go*. Для цього необхідно додатково виконати $R/2$ операцій, щоб встановити відповідний прапорець після того, як буде знайдений мінімальний елемент.

У разі, коли значення *max_c* і *min_c* дорівнюють C_R^2, значення s дорівнюватиме 2, а отже, нам необхідно знайти $R/2$ мінімальних груп.

Отже, загальну кількість операцій для жадібного пошуку за найгіршого сценарію можна записати як:

$$\frac{R}{2}\left(\frac{R^2}{2}+\frac{R^2}{2}\right)=\frac{R^3}{2}. \qquad (3)$$

Для розширеного жадібного алгоритму цю кількість операцій необхідно домножити на кількість елементів $R^2/2$:

$$\frac{R^2}{2}\left(\frac{R^3}{2}\right)=\frac{R^5}{2}. \qquad (4)$$

Позначимо кількість операцій на ітерації як *n*. Оскільки на кожному наступному кроці значення *R* буде меншим вдвічі, n теж зменшуватиметься в 2 рази. Це дає змогу подати цю послідовність кроків як скінченну спадну геометричну прогресію, кількість елементів якої дорівнює $\log_2 wc$, у якої перший елемент $b_1 = n$, а визначник – $q = 1/2$:

$$n+\frac{n}{2}+\ldots+\frac{n}{2^{\log_2 wc-1}}. \qquad (4)$$

Підставивши ці дані у формулу суми геометричної прогресії

$$S = b_1 /(1-q), \tag{5}$$

визначимо кількість операцій за таким виразом:

$$S = \frac{n}{(1-1/2)} = 2n. \tag{6}$$

Підставимо заміть n кількість команд на кожній ітерації, врахувавши, що на кожному кроці нам додатково необхідно обчислити $R^2/2$ операцій компресії зі складністю C для кожної з них.

Отже, складність алгоритму групування даних з використанням жадібного алгоритму пошуку можна описати виразом

$$O\left(2\left(\frac{R^3}{2}+\frac{R^2}{2}C\right)\right) = O(R^3 + R^2C). \tag{7}$$

Складність алгоритму групування даних з використанням розширеного жадібного алгоритму пошуку можна описати виразом

$$O\left(2\left(\frac{R^5}{2}+\frac{R^2}{2}C\right)\right) = O(R^5 + R^2C). \tag{8}$$

Для інших варіацій алгоритму групування даних, коли значення *max_c* і *min_c* дорівнюють іншим величинам, складність обчислюється подібно.

4. Результати експериментальних досліджень

З метою підтвердження ефективності запропонованого алгоритму автор провів експериментальні дослідження. Суть експерименту така: отримані в результаті генерування засобами високорівневого проектування Chameleon [5] моделі ПЗП опрацьовуються, а оптимізовані моделі ПЗП синтезуються у цільовій ПЛІС. У дослідженнях використано засоби логічного синтезу Quartus II 13.1 і ПЛІС Cyclone V 5CGXFC9E7F35C8 від компанії Altera.

Результати експериментальних досліджень для ПЗП розрядністю 143 біти і висотою 51200 комірок з використанням компресії даних методом словника наведено у табл. 1. Результати апаратної реалізації запропонованого алгоритму та алгоритмів LZW [3] і Huffman [4] наведено у табл. 2. Результати синтезу оптимізованої моделі наведено у табл. 3.

Таблиця 1

Результати компресії даних методом словника

Розмір підблока	Коефіцієнт компресії без групування, %	Коефіцієнт компресії з використанням жадібного алгоритму, %	Коефіцієнт компресії з використанням розширеного жадібного алгоритму, %
16	57.06	53.61	48.38
32	44.41	49.70	37.69
36	40.01	44.17	32.79

Таблиця 2

Порівняння результатів апаратної реалізації алгоритмів компресії

Метод компресії	Словника	LZW	Huffman
Частота, МГц	165	300.661	100
Кількість логічних комірок	98	287 + 283	13824
Розрядність слова, біт	144	8	16
Вбудована RAM пам’ять, КВ	2365.6	252	34.56
Необхідно пристроїв, шт	1	18	9
Додаткове сховище для стиснених даних	Ні	Так	Так
Час доступу до комірки пам’яті, тактів	3	нестабільний	4
Коефіцієнт компресії, %	32.79	19.39	39.18
Довільний доступ	Так	Ні	НІ
Максимальна частота пристрою, МГц	550	700	491

Таблиця 3

Порівняння характеристик початкової та оптимізованої моделей ПЗП

Критерій порівняння	Ємність, Біт	Споживання енергії (без статичного споживання), Ват	Частота, МГц
Початкова	9,082,880	0.713 (0.173)	150
Оптимізована	2,422,384	0.694 (0.160)	165

5. Перевірка коректності результатів алгоритму

Запропонований алгоритм здійснює наближений пошук оптимального розміщення, а отже, він не гарантує оптимального результату. Проте коректність цього алгоритму можна підтвердити статистичним аналізом.

Для оцінки коректності використаємо такі статистичні величини: коефіцієнт осциляції *Vr* – це відношення розмаху варіації до середнього значення у відсотках, яке показує відносний розмах варіації до середнього, та коефіцієнта варіації *V* – це відношення середньоквадратичного відхилення до середньоарифметичного у відсотках.

Коефіцієнт варіації дає змогу оцінити однорідність сукупності даних, якщо $V < 17$ % – послідовність абсолютно однорідна, 17–33 % – достатньо однорідна, 35–40 % – недостатньо однорідна, 40–60 % – неоднорідна. У випадку, коли множина результатів є однорідною, можна стверджувати, що початкове перемішування даних не впливає на результат роботи алгоритму, а отже, результати виконання алгоритму залежать тільки від даних, тобто пошук є ефективним.

Як вхідні дані для оцінки було згенеровано 500 наборів даних, отриманих з початкового файла перестановкою у випадковий спосіб. Відсортований за зростанням розподіл результатів показано на рис 2.

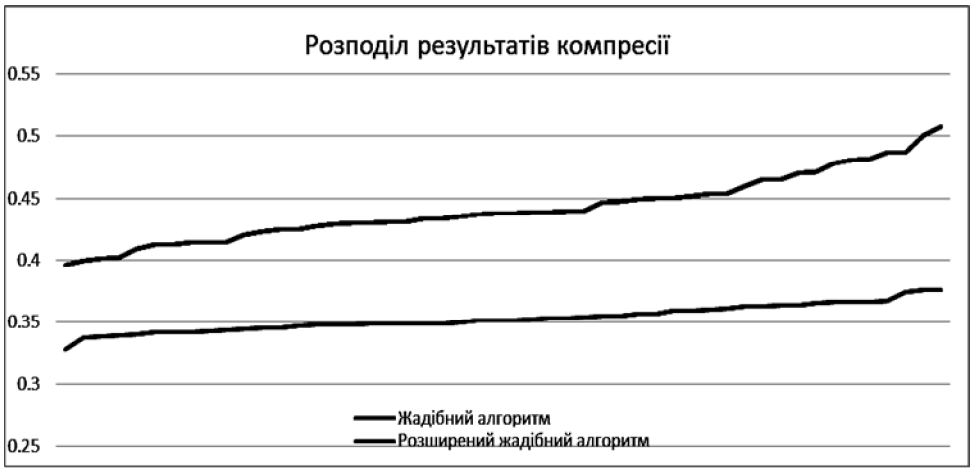

Рис. 2. Розподіл результатів компресії залежно від алгоритму пошуку

Графічне зображення результатів компресії, показаних на рис. 2, демонструє стабільність запропонованого методу групування, більша частина графіків містить плавний повільний розподіл.

Для варіації алгоритму компресії з використанням жадібного алгоритму були отримані такі значення: *V* та *Vr* 6.01 і 25.27 %, для варіації алгоритму компресії з розширеним жадібним алгоритмом – 2.98 і 13.57 %, відповідно. Значення коефіцієнтів варіації та осциляції вказують що отримані результати є однорідними, а отже, запропонований алгоритм компресії показує стабільні результати і є коректним.

Висновок

Розглянуто проблеми ефективного відображення ПЗП, що є компонентами ПМСП, в архітектуру ПЛІС. Запропонований алгоритм передбачає здійснення оптимізації відображення ПЗП в архітектуру ПЛІС шляхом компресії груп даних з використанням методу словника.

Як показали результати синтезу в ПЛІС Cyclone V 5CGXFC9E7F35C8, оптимізована модель ПЗП використовує 2,422,384 біти вбудованої пам’яті, 98 логічних елементів і працює на частоті 165 МГц. Коефіцієнт компресії до синтезу становить 32.79 %, після синтезу – 33.09 %. Зменшення коефіцієнта викликане необхідністю розміщувати вміст пам’яті в блоки фіксованого розміру (вбудовані блоки пам’яті).

Визначальний вплив на продуктивність синтезованих в ПЛІС пристроїв мають тактова частота вбудованих блоків пам’яті і їх організація. Максимальна тактова вбудованих блоків для використаної нами ПЛІС становить 240 МГц. Максимальна висота блока – 8192 комірки. Це приводить до того, що для об’єднання блоків пам’яті необхідно використовувати додаткову логіку. У разі, коли оригінальний блок пам’яті можна цілком розмістити у вбудованому блоці пам’яті, швидкодія буде найвищою, а частота оптимізованої моделі дорівнює частоті вбудованих блоків пам’яті.

1. Welch T. A. A technique for high-performance data compression // Computer. – Vol. 6, no. 17. P. 8–19, 1984. 2. Huffman D. A. A Method for the Construction of Minimum-Redundancy Codes // Proceedings of the I.R.E., September, 1952. – P. 1098–1102. 3. Xin Zhou, Yasuaki Ito, and Koji Nakano. An Efficient Implementation of LZW Decompression Using Block RAMs in the FPGA (Preliminary Version) // Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140, Volume 5, Number 1. – P. 12–19, January 2016. 4. Laurentiu Acasandrei, Marius Neag. Fast Parallel Huffman Decoder For FPGA Implementation // Acta Technica Napocensis: Electronics and Telecommunications, Vol. 49, Number 1, 2008. 5. Chameleon – the System-Level Design Solution. [Електронний ресурс] – Режим доступу: http://intron-innovations.com/?p=sld_chame. 6. Статистика. Ч. 1 // А. В. Юдина, ред.: Л. И. Александрова [Електронний ресурс]. – Режим доступу: http://abc.vvsu.ru/books/ pr_stat1/page0010.asp.