УДК 004.71

Я. М. Крайник, *аспірант*
Чорноморський державний університет імені Петра Могили
вул. 68 Десантників, 10, м. Миколаїв, Україна
codebreaker7@mail.ru

ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ ПАМ'ЯТІ ЧАСТКОВО ПАРАЛЕЛЬНОГО LDPC-ДЕКОДЕРА

У статті описано розроблений метод реалізації LDPC-декодера з меншим використанням ресурсів пам'яті. З урахуванням особливостей алгоритму декодування мінімальної суми визначено можливості оптимізації та розпаралелювання обчислень для мікросхем програмованої логіки. Часткове виконання інструкцій для стадії обробки результатів вузлів перевірки на етапі отримання результатів відносно вузлів значень дозволяє використовувати лише один тип блока обробки. Результати, отримані на попередній ітерації, надають можливість швидкого обчислення необхідних значень без залучення додаткових ресурсів пам'яті. Реалізація на ПЛІС з використанням розробленого методу показала значне зменшення використання ресурсів блокової пам'яті.

***Ключові слова**: LDPC-декодер, декодування, алгоритм мінімальної суми, ПЛІС.*

Вступ. Коди з низькою щільністю перевірок на парність були винайдені у 60-х роках XX ст. Р. Галлахером [1]. Проте через складність апаратної реалізації декодування вони не знайшли широкого застосування на той час. Поява субоптимальних методів декодування, що здатні забезпечувати низьку кількість помилок у декодованій інформації, разом з підвищенням можливостей інтегральних мікросхем дозволили використовувати LDPC-коди в технологіях з високою пропускною здатністю каналу (більше 1 Гбіт/с). Субоптимальні методи декодування дозволяють використовувати менше апаратних ресурсів при незначному програші в якості декодування.

Аналіз останніх джерел досліджень і публікацій. Найбільш поширеними є реалізації LDPC-декодерів на основі спеціалізованих інтегральних схем [2] і програмованих логічних інтегральних схем (ПЛІС) [3].

З точки зору виконання операцій декодування, в роботах різних дослідників [4, 5] виділяють такі типи архітектур LDPC-декодерів:

1) послідовні (всі операції виконуються послідовно, мають просту внутрішню організацію, проте не можуть забезпечувати високу пропускну здатність, через що рідко використовуються);

2) повністю паралельні (усі вузли графа Таннера мають своє апаратне відображення, складну внутрішню структуру та потребують великої кількості з'єднань між елементами, проте забезпечують високу пропускну здатність) [4];

3) частково паралельні (виконують часткове відображення вузлів графа Таннера на апаратні блоки в ході ітерацій декодування; здатні забезпечувати необхідну пропускну здатність та мають значно простішу внутрішню організацію порівняно з повністю паралельними декодерами) [5].

Декодування інформації за допомогою кодів з малою щільністю перевірки парності відбувається на основі матриці, що має низьку щільність значимих елементів. Однією з форм представлення цієї матриці є граф Таннера, що містить два типи вузлів:

– вузли значень (v-вузли);
– вузли перевірок (c-вузли).

Найбільшого поширення набули алгоритми декодування на основі обміну повідомленнями (англ. Message Passing) між вузлами в ході ітерації декодування. Вони, в свою чергу, поділяються на методи з жорстким та м'яким рішеннями. Методи з м'яким рішенням забезпечують кращі показники корекції помилок при передачі та можуть використовуватися на вхідних даних, що мають велику кількість помилок. Найчастіше вхідними даними, а також формою представлення інформації в процесі декодування для алгоритмів з м'яким рішенням є показник логарифмічного відношення подібності (англ. Log-Likelihood Ratio, LLR) для кожного прийнятого біта. До

найбільш відомих алгоритмів обміну повідомленнями з м'яким рішенням відносяться:

1) алгоритм суми добутків (англ. Sum Product Algorithm, SPA);

2) алгоритм мінімальної суми (англ. Min Sum Algorithm, MSA) та його модифікації – нормалізований алгоритм мінімальної суми, алгоритм мінімальної суми з розділенням рядка, алгоритм мінімальної суми з розділенням рядка з граничним значенням та ін.

Варто зауважити, що реалізація LDPC-декодера потребує використання великих ресурсів пам'яті. Побудова декодерів, що працюють на основі великих матриць перевірки парності в такому випадку стає неможливою та вимагає використання мікросхем, що коштують значно дорожче. Тому проблема зменшення використання ресурсів пам'яті при реалізації LDPC-декодера є важливою.

Метою роботи є розробка методу побудови LDPC-декодера, який дозволить підвищити ефективність використання пам'яті при апаратній реалізації.

Основна частина. Декодування повідомлення за допомогою алгоритму мінімальної суми відноситься до алгоритмів з обміном повідомленнями між вузлами. Декодеру відома матриця H, розмірністю $M \times N$ (M – кількість рядків, N – кількість стовпців). Кожну ітерацію алгоритму можна розділити на декілька частин:

1) обмін інформацією від вузлів значень до вузлів перевірки (обробка рядка матриці – горизонтальний крок);

2) обмін інформацією від вузлів перевірки до вузлів значень (обробка стовпця матриці – вертикальний крок);

3) формування результуючого значення ітерації;

4) перевірка синдрому.

На етапі виконання горизонтального кроку для ітерації k виконується обробка значень, отриманих від v-вузлів:

$$\alpha_{ij}^{k} = \prod_{j' \in J \setminus j} sign(\beta_{ij'}^{k-1}) \times \min_{j' \in J \setminus j} (|\beta_{ij'}^{k-1}|), \qquad (1)$$

де α_{ij}^{k} – значення повідомлення вузла перевірки для поточної ітерації,

$\beta_{ij'}^{k-1}$ – значення повідомлення v-вузла на попередній ітерації.

Для кожного вузла значення визначається мінімальне значення та його знак, проте при розрахунках не використовуються дані від поточного вузла.

На другому етапі проводиться сумування повідомлень вузлів перевірки, з'єднаних з вузлом значення:

$$\beta_{ij}^{k} = \sum_{i' \in I \setminus i} \alpha_{i'j}^{k} + \lambda_j, \qquad (2)$$

де λ_j – початкове значення повідомлення v-вузла, що відповідає початковим даним для декодування.

Формування результуючого значення ітерації відбувається за рахунок сумування всіх повідомлень вузлів перевірки, під'єднаних до вузла значення:

$$Z_j^{k} = \sum_{i \in I} \alpha_{ij}^{k} + \lambda_j, \qquad (3)$$

де Z_j^{k} – результуюче значення для ітерації k.

Відповідно до (2) та (3) другий і третій етапи ітерації можна проводити паралельно, оскільки вони відрізняються лише одним доданком при проведенні сумування.

Отримавши результуюче значення ітерації, виконується пошук жорстокого рішення за таким співвідношенням:

$$V_j^{k} = \begin{cases} 0, & Z_j^{k} > 0 \\ 1, & Z_j^{k} \le 0 \end{cases}. \qquad (4)$$

На основі отриманого жорсткого рішення обчислюється синдром:

$$s = H \cdot V^{T}. \qquad (5)$$

У випадку, якщо

$$s = 0 \bmod 2, \qquad (6)$$

то кодове слово проходить перевірку на парність і декодування можна завершувати, видавши як результат значення вектора V. Декодування відбувається, поки не виконані всі ітерації або не виконана умова (6).

Перевагою цього методу при апаратній реалізації є те, що використовуються прості операції знаходження мінімуму, суми, виділення знака, що мають просте відображення в реалізації у вигляді блоків компараторів, суматорів та логічних елементів виключного АБО (XOR) відповідно.

Однак реалізація алгоритму мінімальної суми потребує також значних ресурсів блокової пам'яті для ПЛІС, які є обмеженими. Тому зменшення використання пам'яті за рахунок проведення додаткових простих обчислень дозволить спростити архітектуру декодера. Можливим рішенням цієї проблеми є обчис-

лення значень повідомлень від вузлів значень безпосередньо перед їх обробкою на основі поточного результуючого значення, а також значення повідомлення вузла перевірки. Виразимо на основі (2) та (3) значення повідомлення v-вузла:

$$\beta_{ij}^{k} = Z_{j}^{k} - \alpha_{ij}^{k}. \qquad (7)$$

Зазначена операція легко може бути реалізована апаратно за допомогою блока віднімання. При цьому, оскільки таких блоків може бути декілька, то всі необхідні операції для (7) можуть виконуватися паралельно.

Проте, оскільки відповідно до (7) пам'ять значень Z_j^k повинна залишатися незмінною до завершення поточної ітерації, то необхідно виділити додаткову пам'ять для збереження результатів проміжних обчислень ітерації. Розмір цієї пам'яті відповідає розміру пам'яті значень Z_j^k. Позначимо значення, що знаходяться в цій пам'яті, як $Ztemp_j^k$.

Після проведення обчислення повідомлення вузла перевірки для одного рядка є можливість виконати часткове обчислення результуючого значення ітерації

$$Ztemp_j^k = Ztemp_j^k + \alpha_{ij}^k. \qquad (8)$$

Після виконання обробки всіх рядків у пам'яті значень $Ztemp_j^k$ міститиметься значення результуючого значення ітерації. Оскільки в такому випадку відпадає необхідність проведення операцій вертикального кроку, то кількість операцій для однієї ітерації декодування зменшується до кількості рядків у матриці H – M.

Розглянемо метод реалізації декодера на основі зазначеного підходу. Визначається максимальна кількість одиниць у рядку (вага рядка) $w_{r\max}$ для заданої матриці H:

$$w_{r\max} = \max(w_{ri}),\ r \in R, \qquad (9)$$

де R – множина рядків матриці H;

w_{ri} – вага i-го рядка.

Значення $w_{r\max}$ характеризує те, на скільки блоків необхідно розділити пам'ять для збереження індексів та пам'ять значень α_{ij}^k. Розмірність кожного блока визначається кількістю рядків M для заданої матриці H. Розрядність значень залежить від формату представлення даних для декодера.

Збереження вихідних даних для декодування, а також проміжних результатів обчислень та кінцевих результатів для кожної ітерації відбувається в двох блоках пам'яті, що мають розмірність $1 \times N$. Розрядність пам'яті так само визначається форматом представлення даних.

Таким чином, архітектура декодера передбачає наявність чотирьох видів пам'яті:

1) пам'ять індексів одиниць у рядках матриці H, $Memory_{address}$;

2) пам'ять для значень α_{ij}^k, $Memory_{\alpha}$ (перед початком декодування ініціалізується значенням 0);

3) пам'ять збереження проміжних результатів ітерації, $Memory_{Ztemp}$ (ініціалізується значеннями вхідного набору даних для декодування);

4) пам'ять початкових даних та кінцевих результатів ітерації, $Memory_Z$.

Пам'ять $Memory_{address}$ ініціалізується індексами одиниць у рядках матриці H. У випадку нерегулярної матриці, такої що

$$w_{ri} \neq w_{r\max}, \qquad (10)$$

значення в рядках заповнюються адресою, яка відсутня у початковій матриці, наприклад, максимальним значенням для даної розрядності. У ході виконання кожної ітерації дані з такими адресами обробляються спеціальним чином, щоб вони не мали вплив на коректні дані.

На кожній ітерації декодування відбувається порядкове зчитування значень з пам'яті $Memory_{address}$ та пам'яті $Memory_{\alpha}$. За отриманими індексами відбувається зчитування значень з пам'яті $Memory_Z$. Відповідно до (7) поточні значення α_{ij}^k і Z_j^k подаються на блоки віднімання, у результаті чого отримуємо значення β_{ij}^k. Ці дані подаються на вхід блока компараторів та побітового виключного АБО для реалізації операцій, описаних в (1). Отримані α_{ij}^k перезаписуються до пам'яті $Memory_{\alpha}$ у поточний рядок. Також для цих даних виконується сумування з відповідними значеннями $Ztemp_j$ та збереження результату в пам'яті $Memory_{Ztemp}$.

Така послідовність організації ітерації з частковим виконанням обчислень вертикального кроку на етапі горизонтального кроку дозволяє отримати такий самий результат ітерації, як і при послідовному виконанні горизонтального та вертикальних кроків. В той же час, це дає можливість відмовитись від виділення окремих блоків пам'яті для збереження значень β_{ij}^{k} та відповідних індексів, а також додаткових логічних ресурсів ПЛІС для організації обробки вертикального кроку. Додатково необхідно лише виділити пам'ять для $Memory_{Ztemp}$. Для матриці з максимальною вагою стовпця

$$w_{c\max} = \max(w_{cj}),\ c \in C, \qquad (11)$$

де C – множина стовпців матриці H;

w_{cj} – вага j-го стовпця, виграш відносно необхідності збереження кількості значень у пам'яті становить

$$Count = 2 \cdot w_{c\max} \cdot N - N. \qquad (12)$$

На завершальному етапі ітерації виконується копіювання значень з пам'яті $Memory_{Ztemp}$ в пам'ять $Memory_Z$. Отримані значення використовуються для проведення перевірки синдрому, на основі чого визначається коректність результатів декодування.

У ході дослідження для перевірки реалізації LDPC-декодера використана нерегулярна матриця з низькою щільністю значимих елементів розмірністю (5000, 10000). Для цієї матриці $w_{r\max} = 9$, а $w_{r\min} = 5$. Візуальне представлення значущих елементів матриці зображено на рис. 1.

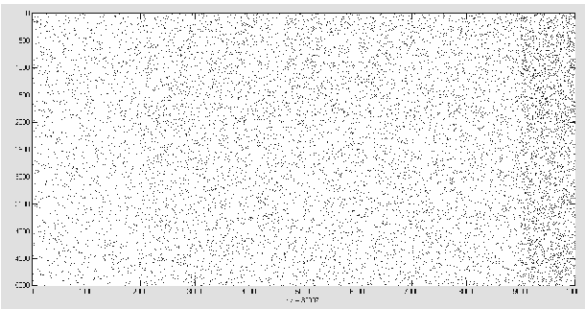

Рис. 1. Візуальне представлення матриці з низькою щільністю перевірки парності

Кількість одиниць у стовпцях матриці також не є однаковою. Характер розподілу кількості значущих елементів відповідно до стовпців зображено на рис. 2.

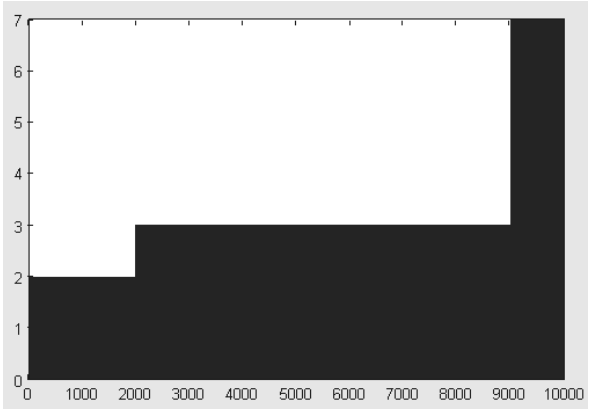

Рис. 2. Розподіл значущих елементів матриці за стовпцями

З точки зору розпаралелювання виконання обчислень, важливим є також розподіл індексів значущих елементів по позиціях у рядках. У тому випадку, коли діапазони значень індексів елементів не перетинаються, пам'ять можна організувати таким чином, щоб запис та зчитування даних виконувалися паралельно. Результати розподілу для обраної матриці наведені в табл. 1.

З табл. 1 видно, що діапазони індексів елементів за позиціями перетинаються, тому оптимізувати цей процес шляхом паралельного виконання не можна.

Наведені вище показники свідчать про те, що обрана матриця є складною для виконання розпаралелювання дій для проведення декодування з апаратної точки зору.

Таблиця 1

Максимальне та мінімальне значення індексів під'єднаних вузлів значень

№ вузла перевірки	Мінімальне значення індексу	Максимальне значення індексу
1	0	8234
2	49	9433
3	375	9592
4	812	9974
5	1207	9972
6	2669	9999
7	4748	9999
8	5355	9999
9	9651	9651

Для обраної матриці на основі запропонованого методу побудови реалізовано LDPC-декодер. Для реалізації використовувались мова схемотехнічного опису VHDL та середовище розробки Altera Quartus II 13.1

Web Edition. Відповідно до запропонованого методу використовується лише один тип блока обробки повідомлень вузлів. Максимальна тактова частота, на якій може працювати модуль, становить вище 200 МГц для мікросхеми Altera Stratix IV. Відповідно до (12) та з урахуванням того, що в організації пам'яті використовувалась блочна пам'ять M9K розмірністю 512x16 біт, зменшити використання цього ресурсу вдалося на 254 таких блоків пам'яті. Кількість обробок для кожної ітерації становить 5000.

Висновки. У роботі наведено результати розробленого методу побудови частково паралельного LDPC-декодера, який дозволяє значно зменшити використання ресурсів блокової пам'яті ПЛІС, а також спрощує виконання декодування у ході ітерації за рахунок використання лише одного типу блока обробки. Додаткове обчислення необхідних значень реалізовується безпосередньо перед початком обробки рядку матриці. Проведено реалізацію декодера на основі запропонованого підходу для нерегулярної матриці з низькою щільністю значущих елементів, яка показала значне зменшення використання ресурсів блокової пам'яті.

Список літератури / References

1. Gallager, R. G. (1962) Low-density parity check codes. *IRE Transaction Info.Theory, IT-8*, 21–28, January.
2. Mohsenin, T., Thuong, D. and Baas, B. (2009) Multi-split-row threshold decoding implementations for LDPC codes. *IEEE International Symposium on Circuits and Systems (ISCAS'09)*, May, pp. 2449–2452.
3. Aravind, N. and Praveen Kumar, P. (2013). Low hardware layered decoding architecture for LDPC code [Internet]. *International Journal of Science and Research*, 2 (3), March 2013. Available from: <http://www.ijsr.net/archive/v2i3/IJSRON2013490.pdf>
4. Mohsenin, T. (2010) Algorithms and architectures for efficient low density parity check (LDPC) decoder hardware: a thesis submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in electrical and computer engineering. University of California, Davis, 122 p.
5. Karkooti, Marjan (2004) Semi-parallel architectures for real-time LDPC coding: a thesis submitted in partial fulfillment of the requirements for the degree of Master of Science. Rice University, Houston, Texas, 87 p.

Y. M. Krainyk, *post-graduate*
Petro Mohyla Black Sea State University
68 Desantnykiv str., 10, Mykolaiv, Ukraine
codebreaker7@mail.ru

RISING OF MEMORY EFFICIENCY OF PARTIALLY PARALLEL LDPC-DECODER

In the article the method for LDPC-decoder implementation that allows to reduce memory consumption is described. Minimal sum algorithm peculiarities are considered in context of FPGA integrated circuit. According to the peculiarities, capabilities for optimization and parallel execution of algorithm are determined. Partial variable node processing at the stage of check node processing allows to use only one type of processing unit. The results from the previous iteration provide fast necessary values calculation without usage of additional memory resources. Partial variable node processing also establishes overall number of internal iteration reducing during decoding process. Overall number of internal iteration during decoding process is equal to number of rows in LDPC-matrix and column processing is executed partially at horizontal steps. Architecture implementation in FPGA results in significant reducing of block memory usage. New architecture is compared with straightforward approach for LDPC-decoder implementation and the comparison indicates large reduce in memory resource usage.

Keywords*: LDPC-decoder, decoding, minimal sum algorithm, FPGA.*

Стаття надійшла до редакції 01.11.2014.

Рецензенти: Мусієнко М. П., д.т.н., професор,
Коваленко І. І., д.т.н., професор.