УДК 004.312.26:519.725

Я. М. Крайник, к. т. н., докторант; В. О. Перов

ОСОБЛИВОСТІ ПОБУДОВИ ОКРЕМИХ БЛОКІВ ТРС-ДЕКОДЕРУ НА БАЗІ ПЛІС

У статті розглянута проблема побудови окремих блоків для реконфігуровного декодеру Turbo-Product-кодів. Запропоновано принципи організації блоків пошуку позицій мінімальних значень у вхідному векторі і універсального блоку зсуву, які дозволяють забезпечити повторне використання логічних ресурсів мікросхем програмованої логіки для обробки кодів різної структури. Принципи для організації блоків придатні для використання як для спеціалізованих мікросхем, так і для програмованих логічних інтегральних схем (ПЛІС), проте акцент робиться саме на останніх через їх універсальність та можливість багаторазової зміни конфігураційної структури. Розроблений блок пошуку використовує у своїй структурі два типи простих елементів і на їх основі виділяє значення, які є мінімальними. При цьому, запропонована структура, яка дозволяє виконувати пошук заданої кількості мінімальних значень. У свою чергу, універсальний блок зсуву забезпечує необхідні переміщення у вхідному або вихідному векторі значень таким чином, що вони можуть бути записані у модуль пам'яті або передані до модулю обробки. За рахунок використання цього блоку забезпечується в тому числі паралельний доступ до значень у пам'яті, що значно підвищує швидкодію процесу декодування. Крім цього, блок орієнтований на роботу з кодами різної довжини і виконує зсув значень за мінімальну кількість тактів робочої частоти. Відповідно, незалежно від коду, який використовується під час декодування, зсув значень буде виконано за одну й ту саму одиницю часу. Відповідно до наведених у роботі положень розроблені відповідні блоки мовою схемотехнічного опису VHDL. Проведено моделювання роботи розроблених блоків, яке показало коректність їх роботи. Ці блоки у складі декодеру Turbo-Product-кодів дозволяють підвищити універсальність декодеру з забезпеченням високої пропускної здатності.

***Ключові слова**: декодер, блок, Turbo-Product-коди, ПЛІС.*

Вступ

Алгоритми декодування завадостійких кодів і, зокрема, Turbo-Product-кодів (ТР-кодів) отримали широке практичне застосування та були досліджені у великій кількості наукових робіт [1 – 8]. Вони використовуються у системах передачі інформації різного цільового призначення та з використанням різних фізичних середовищ. Тим не менш, не усі алгоритми можуть бути переведені відповідно до математичної моделі у фізичну реалізацію на базі сучасних мікросхем. Це пояснюється обмеженнями щодо ресурсів, які наявні у таких мікросхемах.

Реконфігурований декодер на базі програмованих логічних інтегральних схем (ПЛІС) може проводити декодування для різних кодів. В той час, як самі по собі системи на базі ПЛІС є реконфігурованими системами, реконфігурація декодеру означає можливість проведення зміни налаштувань, що призведе до того, що декодер буде здатний оброблювати інформацію, сформовану на основі нового коду. Таким чином, можна виділити одразу два рівня реконфігурації:

перший – апаратний – забезпечується можливостями налаштування самої ПЛІС;

другий – логічний – забезпечується реалізацією на рівні мови опису схемотехніки з подальшим перетворення у бітовий файл прошивки мікросхеми.

Саме другому аспектові присвячена ця робота, зокрема, в частині побудови універсальних блоків, які можуть виконувати окремі стадії декодування.

В залежності від заданих налаштувань реконфігурованим декодером може проводитись обробка кодових слів різної довжини. Актуальним, при цьому, є питання повторного використання ресурсів реалізованих блоків для обробки різних кодів. Саме така концепція

надає можливість не застосовувати окремий блок на кожен оброблюваний код, а зменшити використання ресурсів за рахунок їх повторного використання у складі одного універсального блоку, який виконує задану функцію для усіх кодів. Чим більшою є кількість таких універсальних блоків, тим більше ресурсів можна зекономити. У цьому полягає можливий економічний ефект від застосування цієї роботи.

Аналіз наукових джерел

Декодування ТР-кодів з використанням алгоритму Чейза [2] є доволі поширеним завдяки своїй простоті та підвищеній здатності до виправлення помилок, ніж у жорсткому декодері. Він використовується, в тому числі, як базовий для інших алгоритмів декодування [3]. Основна ідея алгоритму ґрунтується на припущенні, що помилки (за умови їх наявності у кодовому слові), які виникли під час передачі у бітах мають найменше абсолютне значення. Проводиться генерація тестових векторів шляхом перебору усіх можливих двійкових значень у заданих позиціях. Для кожного кодового слова проводиться обчислення метрики, яке дозволяє оцінити близькість до коректного кодового слова. На основі значення метрики обирається базове рішення, яке використовується при подальшій обробці. Алгоритм Чейза відноситься до переліку спискових алгоритмів, які намагаються знайти коректне рішення на основі генерації списку рішень, серед елементів якого обираються такі, які задовольняють заданим критеріям.

У роботі [4] представлено блок декодування, який дозволяє опрацьовувати дані для складника коду довжиною 32 символи. Такий блок знаходить 3 мінімальні значення. Крім того, для коректної його роботи на нього необхідно подавати лише 32 значення. Саме тому для реалізації реконфігурованого декодеру обов'язковою є розробка блоку пошуку мінімальних значень, який здатен опрацьовувати дані для кодів різної довжини.

У великій кількості робіт [5 – 7] акцент робить на загальну архітектуру декодеру, в той час як самі окремі блоки, з яких складається декодер, потребують більш детального опису. Особливо, якщо запропонований алгоритм орієнтований на реалізацію у апаратній платформі ПЛІС.

У роботах [8, 9] представлені метод декодування та загальна архітектура декодеру для реалізації на базі ПЛІС. Ця робота є логічним продовженням згадуваних робіт і у ній робиться акцент на реалізацію окремих модулів у складі декодеру.

Метою роботи є підвищення універсальності декодеру для можливості обробки повідомлень на основі різних кодів.

Для досягнення поставленої мети необхідно провести дослідження щодо організації блоків пошуку позицій мінімальних значень і універсального блоку зсуву для реконфігурованого декодеру ТР-кодів. За рахунок цього стане можливими проведення декодування на основі різних кодів з можливістю зміни налаштувань для горизонтального та вертикального складника коду. Цільовою платформою для реалізації такого декодеру обрані мікросхеми ПЛІС, тому мають бути враховані особливості цього типу мікросхем. Результатом проведеного дослідження мають бути методи побудови вказаних блоків для реконфігурованого декодеру на базі ПЛІС, що дозволить проводити обробку кодів різної довжини, відповідно, зробити цей елемент систем передачі інформації більш універсальним.

Основна частина

Цикл декодування одного закодованого слова починається з подачі вектору значень на вхід декодуючого блоку. У цьому векторі необхідно знайти вказану кількість мінімальних значень. Для того, щоб визначити одну позицію, абсолютне значення якої в кодовому слові не перевищує будь-яке інше значення у векторі, достатньо використовувати блок, схематична структура якого представлена на рис. 1. Він з'єднується з іншими подібними

блоками у послідовний каскад. Такі блоки використовуються у розглянутих роботах [4, 5].

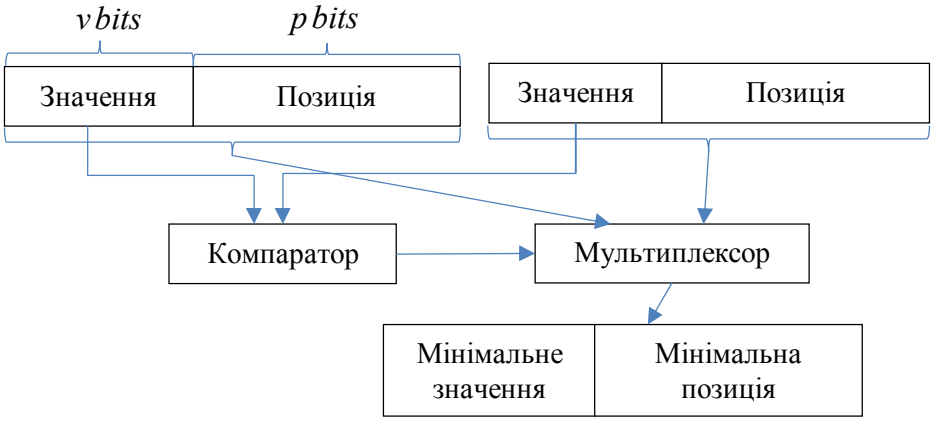

Рис. 1. Блок порівняння двох значень

Цей блок дозволяє знайти одне значення, проте для генерації списків тестових векторів необхідно визначити декілька позицій з мінімумом абсолютних значень. При цьому, значення, які не обираються в якості мінімального на певному рівні відкидаються такими блоками, тому неможливо коректно визначити позиції мінімумів з використанням таких блоків.

Особливістю каскаду блоків порівняння є те, що значення, яке виявилось меншим з двох не відкидається одразу, а подається на вхід паралельного каскаду. При цьому обидва вхідні значення, подані на вхід такого елементу, у результаті можуть бути обрані в якості кінцевих мінімумів. Якби одне зі значень відкидалось одразу, то, за умови порівняння двох мінімумів, один з них був би втрачений і позиція мінімуму була б обчислена некоректно. Організація паралельного каскаду надає можливість відкинутому значенню пройти по паралельній гілці та потрапити до числа мінімумів. Значення, які подаються на вхід блоку є комбінацією з самого значення та його індексу. Під значення відводиться фіксована довжина, яка дорівнює v бітам. Під значення індексу відводиться довжина p біт. Для роботи такого каскаду необхідним є інший тип блоків, наведений на рис. 2. Блоки цього типу використовуються у роботах [4, 5] для організації декодування.

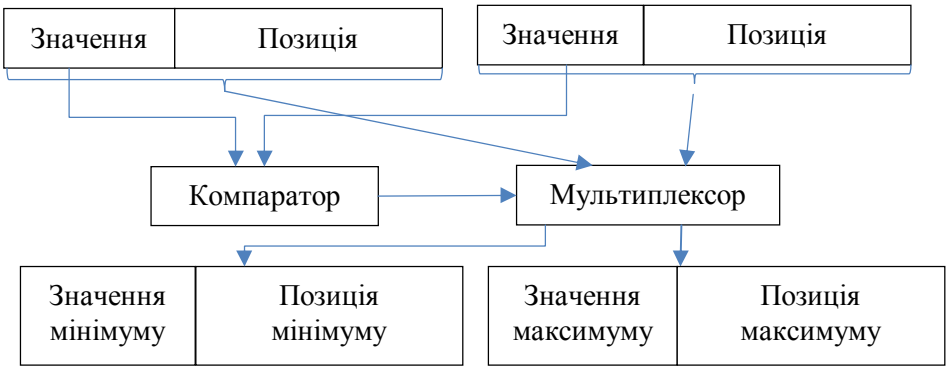

Рис. 2. Блок порівняння двох значень для організації паралельного каскаду

На відміну від попереднього блоку у цьому випадку мультиплексор має два виходи. Один з них призначений для того, щоб обрати мінімальне значення. У випадку рівності значень перевага надається значенню з меншим індексом. На інший вихід подається значення, яке не

було обране в якості мінімального.

На рис. 1, 2 блоки представлені у вигляді регістрів. Також їх можна представити як блоки з відповідною кількістю входів та виходів, оскільки регістри можна розглядати як окремий компоненти, які використовуються для зв'язування блоків комбінаційної логіки.

Іншим важливим моментом у роботі блоку пошуку мінімумів є можливість обробки вхідних даних з різною довжиною кодового слова. Для цього на вхідні регістри, індекси яких перевищують довжину кодового слова, подається значення, яке відображає максимальне абсолютне значення у системі. Також необхідно забезпечити, щоб усі результуючі значення позицій мінімумів входили до діапазону, що визначається довжиною коду – $[0, n-1]$.

У випадку, коли значення поза діапазоном не були відсіяні (це можливо, наприклад, коли усі вхідні значення однакові або для деяких варіантів кодових слів), вони мають бути відсіяні безпосередньо на останньому етапі для того, щоб коректно сформувати тестові вектори.

Перейдемо безпосередньо до організації каскаду пошуку мінімумів.

Послідовне з'єднання вищенаведених блоків у каскад з кількома рівнями дозволить однозначно визначити мінімальний елемент з найменшим індексом (за умови рівності значень). Проте, для роботи алгоритму на основі генерації списку тестових векторів одного значення недостатньо, тому такі рівні можуть повторюватись лише до певного моменту, коли буде виділена визначена кількість значень, серед яких і необхідно обрати кінцевий набір значень.

На першому рівні такого каскаду задіяні блоки з двома виходами, що дозволяє відсортувати значення. Їх кількість на першому рівні становить $n_{\max}/2$. У результаті отримуються $n_{\max}/2$ значень, які потрапляють у перший каскад, а інша половина потрапляє у паралельний. Дві половини оброблюються з використанням різних блоків. Перша частина потрапляє на блоки з подвійним виходом, в той час, як друга – на блоки з одиничним виходом. Таким чином, на другому рівні необхідно задіяти $n_{\max}/4$ блоків з подвійним виходом і таку саму кількість блоків з одиничним виходом. Особливість обробки значень, які потрапили до другої частини є те, що, фактично, тепер це є самостійний каскад, який у результаті має видати єдине значення. Щодо першої частини, яка потрапила на другому рівні на вхід блоків з подвійним виходом, то для результатів обробки цієї частини застосовується та сама процедура, що і для першого рівня. Як результат, половина з отриманих значень будуть подані на вхід каскаду, що містить $n_{\max}/8$ блоків з подвійним виходом (відповідно, це буде $n_{\max}/4$ від загальної кількості значень). Така процедура повторюється до того моменту, коли буде знайдено перше мінімальне значення, яке отримується на виході останнього блоку з подвійним виходом (вихід, що вказує на мінімальне значення). Загальна кількість рівнів, яка необхідна для організації першої частини такого каскаду становить

$$lev\_first\_stage = \log_2 n_{\max}. \quad (1)$$

На виході першої стадії каскаду буде отримана така кількість значень:

$$first\_stage\_outs = lev\_first\_stage + 1. \quad (2)$$

Одне з отриманих значень однозначно належатиме множині значень позицій мінімумів.

Відповідно, необхідно обробити $first\_stage\_outs - 1$ значень. $first\_stage\_outs - 2$ значень є виходами підкаскадів, сформованих з блоків з одиничним виходом, а ще одне значення обирається як таке, що програло при порівнянні на останньому рівні з використанням блоку з подвійним виходом.

Другий рівень каскаду пошуку множини мінімумів формується на основі припущення:

$$first\_stage\_outs > k. \quad (3)$$

Відповідно необхідно виділити зі знайдених значень необхідну кількість. Це пов'язано з тим, що генерація і подальша обробка великої кількості тестових векторів потребує великої кількості ресурсів та може не відповідати обмеженням щодо логічних ресурсів у обраній мікросхемі ПЛІС.

Обробка на другій стадії каскаду призначена для того, щоб виявити решту $k-1$ значень. Організація цієї стадії каскаду відрізняється від попередньої. Загальна структура блоку пошуку мінімумів представлена на рис. 3.

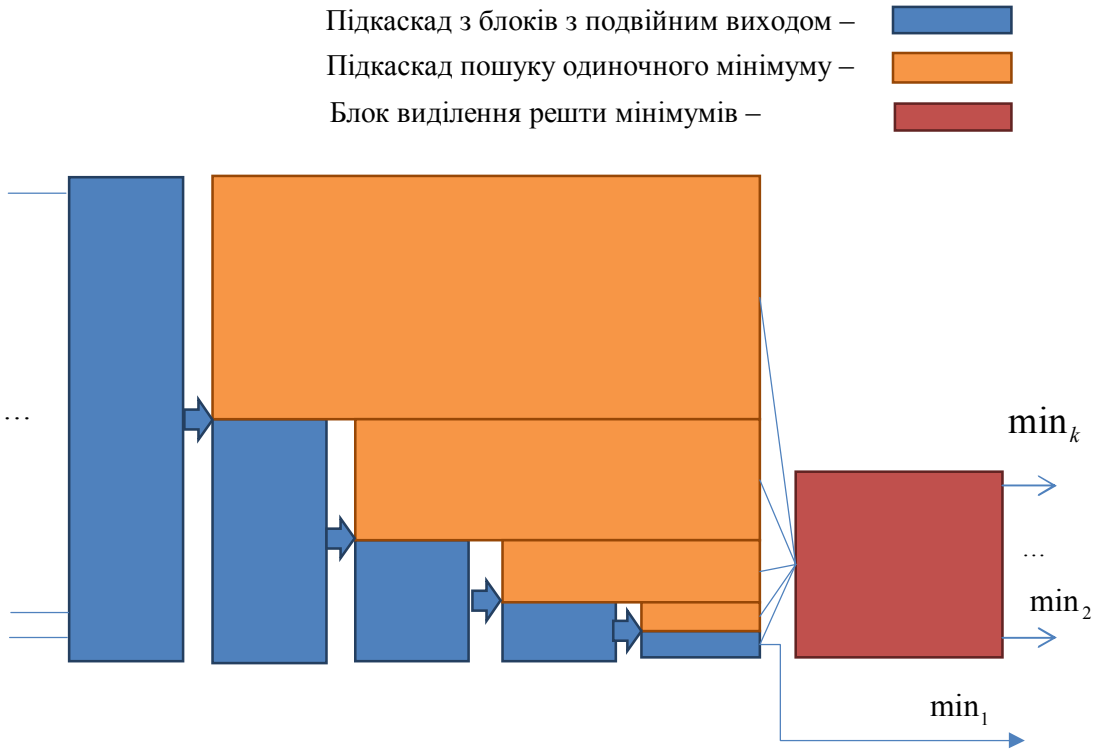

Рис. 3. Організація каскаду пошуку позицій з мінімальними значеннями

Окремої уваги заслуговує блок пошуку решти мінімумів, який призначений для того, щоб виділити серед отриманих значень необхідну кількість мінімальних значень на виході. Так само, як і для попередніх блоків, його складовими елементами є вищеописані типи блоків. Кількість рівнів компонентів, які утворюють цей блок буде визначена як:

$$layer\_count = first\_stage\_outs - 1. \quad (4)$$

Також можна відзначити, що кількість компонентів з одиночним виходом становитиме:

$$out1\_count = layer\_count - 1, \quad (5)$$

Тобто, з кожним рівнем, окрім першого, один компонент буде відкидатись.

Представлення матриці вхідних даних у апаратній структурі, зазвичай, організоване з використанням регістрової або блокової пам'яті. Така пам'ять передбачає, що з неї за один такт можна зчитати лише одне значення. Відповідно, у декодері наявні $n_{\max}$ модулів такої пам'яті. Вони дозволяють за один такт отримати необхідний вектор вхідних даних або для рядка, або для стовпця. Проблема полягає в тому, що при записі вхідних даних за тими самими адресами, оброблені дані для наступної пів-ітерації будуть розміщені в одному модулі пам'яті. Тому для зчитування вхідних даних необхідно буде витратити $n_{\max}$ тактів замість одного такту. Це означатиме, у свою чергу, значну втрату пропускної здатності декодеру.

Для того, щоб уникнути такої втрати у продуктивності декодеру, необхідною є така організація процесу запису і читання ресурсів у пам'яті ПЛІС, яка дозволить виконувати зчитування і рядків, і стовпців за один такт. Основою для такої організації є універсальний блок зсуву (англ. barrel shifter), який забезпечує зсув/обертання вхідного вектору на задану кількість позицій. Він підключається до блоку, що виконує зчитування/запис даних з пам'яті. У свою чергу, блок читання/запису підключається до $n_{\max}$ блоків пам'яті. Він передає дані з пам'яті на універсальний блок зсуву. Важливим моментом, який необхідно враховувати для реконфігуровного декодеру, є необхідність коректного виконання зсуву для слів різної довжини. Можливим є варіант реалізації обертання для набору кодів базових довжин. У такому випадку, обробка векторів, виконується відповідно до налаштувань базового коду, а невідповідність між фактичними довжинами має бути врахована на наступних етапах обробки, щоб кінцевий результат був коректним. В якості базових довжин кодів рекомендується обирати такі значення, які є результатом піднесення у степінь двійки (наприклад, 16, 32, 64, 128). Для кодів, довжина яких не є рівною значенню $n_{\max}$ виконується зсув лише частини від повного набору даних, яка і відповідає фактичній довжині коду. Це може бути записано як

$$shift\_vals\_denom = 2^{log_2 n_{max} - log_2 n_{cur}}, \quad (6)$$

де $shift\_vals\_denom$ – дільник для фактичної кількості значень, що має бути зсунута;

n_{cur} – фактична довжина оброблюваного коду.

У свою чергу, кількість значень, що буде зсунута, обчислюється як

$$shift\_count = \frac{n_{max}}{shift\_vals\_denom}. \quad (7)$$

З точки зору реалізації модуль універсального блоку зсуву є вимогливим щодо використання логічних ресурсів, оскільки повинен забезпечувати велику кількість з'єднань між різними логічними компонентами. Тому, цей чинник має бути врахований при виборі кінцевої мікросхеми ПЛІС, яка повинна містити достатню кількість логічних ресурсів для виконання зсуву слів з заданою максимальною довжиною. Загалом, такий модуль складається з $n_{\max}$ мультиплексорів, кожен з яких має по $n_{\max}$ входів. У результаті на вихід кожного мультиплексору отримується одне значення відповідно до значення лічильника. Загальна принципова схема цього модулю представлена на рис. 4.

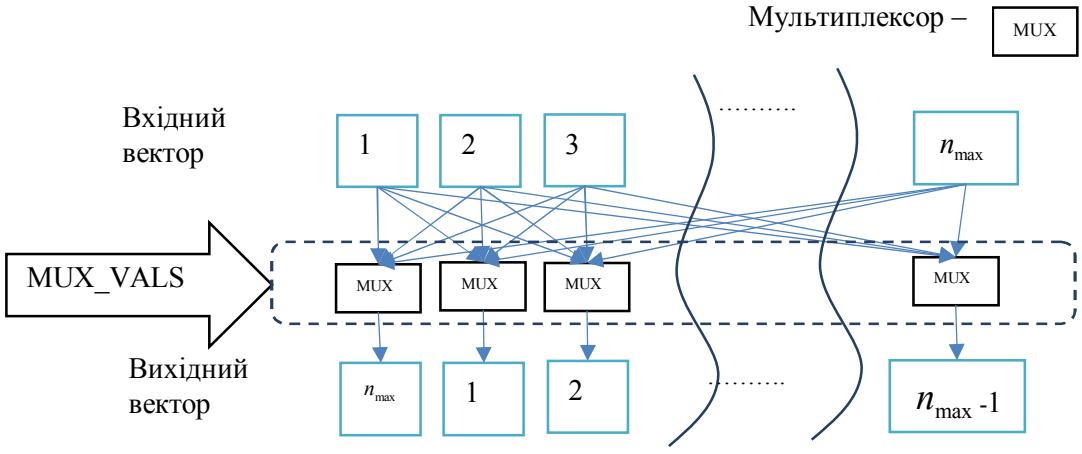

Рис. 4. З'єднання між вхідним та вихідним вектором у блоці зсуву

Для того, щоб коректно обрати вектор з пам'яті, значення адрес MUX_VALS мають бути

різними. Обчислення значення MUX_VALS забезпечується за рахунок використання лічильників, яким задається початкове значення, після чого вони починають рахувати. Після досягнення максимального значення по лічбі, яке визначається довжиною оброблюваного коду, відбувається перехід до значення 0 і продовження лічби. У випадку, коли значення лічильника декрементується, після досягнення значення 0 відбувається перехід до максимального значення з подальшим декрементом за необхідності.

На рис. 4 наведена спрощена структура модулю, яка не враховує того факту, що кількість значень, що підлягають зсуву не завжди відповідає кількості входів.

Для запропонованих блоків декодеру виконана реалізація мовою схемотехнічного опису VHDL. Проведено моделювання їх роботи у середовищі MentorGraphicsModelSim 10.1c. Результати моделювання блоку пошуку позицій мінімумів наведено на рис. 5 і 6.

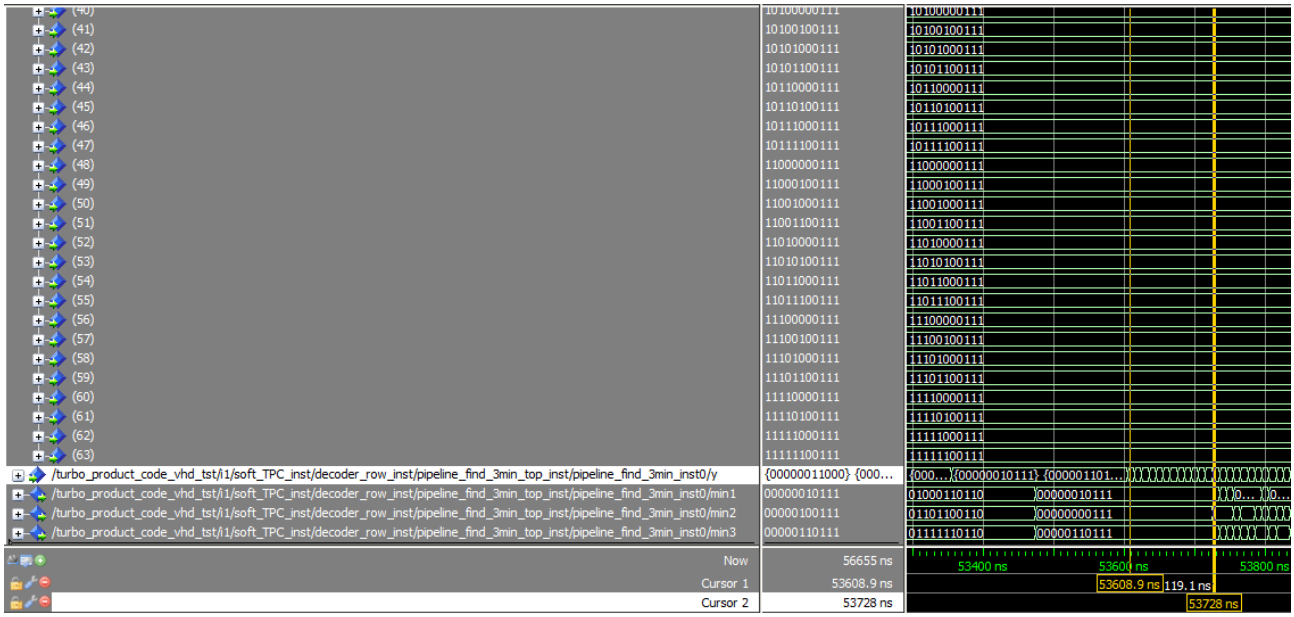

Рис. 5. Скріншот моделювання роботи блоку пошуку мінімумів

На рис. 5 показано, яким чином подаються дані на модуль і які дані отримуються як результат. Вектор, що подається на вхід розділений на дві частини. Кожне вхідне значення у прикладі містить 11 біт: сім використовуються для представлення позиції, а останні чотири – для самого значення. На вхід були подані значення однакові за модулем, тому у результаті отримані значення 7. Варто звернути увагу на індекси значень, які були обрані під час подачі вхідних даних: вони відповідають позиціям 0, 1 та 3.

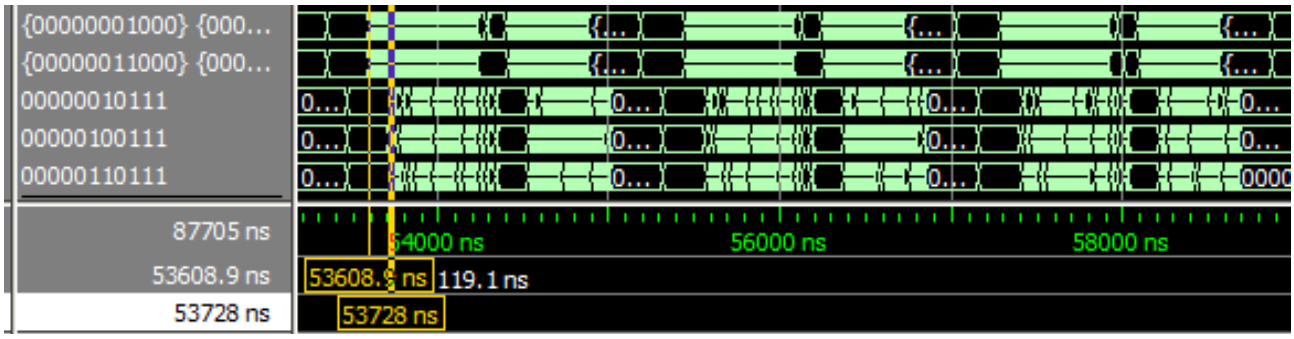

Рис. 6. Скріншот моделювання ітерацій блоку пошуку мінімумів

На рис. 6 представлено проведення моделювання відповідно до ітерацій (масштаб є меншим у порівнянні з рис. 5). На ньому видно, що блок працює не постійно, а з

інтервалами. Відповідно, мають бути наявні два такі блоки для обробки рядків і стовпців.

За допомогою засобів Dataflow у середовищі ModelSim отримане наступна схема блоку пошуку мінімумів на рівні регістрової логіки (рис. 7).

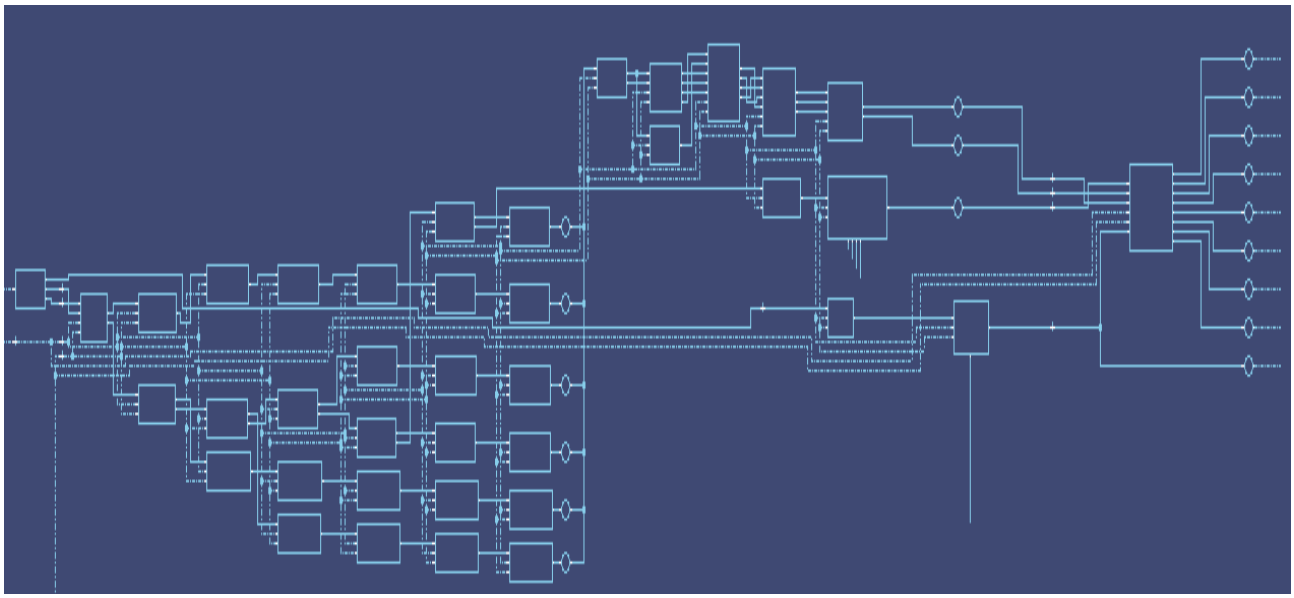

Рис. 7. Структурна організація блоку пошуку мінімумів

Автоматизовані засоби побудови подібних схем не завжди генерують найкращий варіант структурної схеми, проте цього достатньо для того, щоб ознайомитись з внутрішньою будовою блоку.

Так само виконана реалізація універсального блоку зсуву для кодів з базовою довжиною 32, 64 та 128 біт.

Висновки

У роботі запропоновані та описані принципи побудови окремих блоків для організації декодування ТР-кодів реконфігуровним декодером. Основну увагу присвячено блоку пошуку позицій мінімальних за амплітудою значень у кодовому слові, а також універсального блоку зсуву. Запропоновано методи організації вказаних блоків, які, на відміну, від відомих досліджень дозволяють проводити обробку кодових слів різної довжини. Запропонований блок пошуку мінімумів є універсальним під час обробки кодів з заданою максимальною довжиною. Повторне використання ресурсів у блоці для пошуку мінімальних значень дозволяє підвищити універсальність пристрою. За рахунок цього можливе проведення декодування не одного набору кодів, а одразу кількох різних кодів з можливістю зміни налаштувань як для горизонтальної, так і для вертикальної складової коду. Досліджено особливості їх реалізації для реконфігурованого декодеру. Визначено, що такі блоки можуть виконувати обробку для кодів з різною довжиною кодового слова і яких правил під час розробки цих блоків необхідно дотримуватись для того, щоб забезпечити таку можливість. Визначені основні положення, на основі яких відбувається побудова блоку пошуку мінімальних значень для формування тестових векторів. Також запропоновано структуру універсального блоку зсуву та те, яким чином він має виконувати свої функції для обробки кодових слів різної довжини. Завдяки використанню запропонованих у цій роботі блоків декодер отримує можливість коректно проводити обробку кодових слів для різних кодів. Проведено моделювання роботи реалізованого декодеру з використанням середовища ModelSim для кодів з базовою довжиною 32, 64, 128 біт та похідних від них кодів. Тестування роботи реалізованого декодеру проведено на платі AlteraDE0-Nano, яка надана

для досліджень фірмою «Віаком» (м. Київ).

СПИСОК ЛІТЕРАТУРИ

1. Berrou C. Near Shannon limit error-correcting coding and decoding : Turbo-codes / C. Berrou, A. Glavieux, P. Thitimajshima // IEEE ICC'93. – 1993. – P. 1064 – 1071.

2. Chase D. A class of algorithms for decoding block codes with channel measurement information / D. Chase // IEEE Trans. Inform. Theory. – Jan. 1972. – Vol. IT-18. – P. 170 – 182.

3. Pyndiah R. Near-Optimum Decoding of Product Codes: Block Turbo Codes / R. Pyndiah // IEEE Transactions on Communications. – August 1998. – Vol. 46, No. 8. – P. 1003 – 1010.

4. Turbo product code decoder without interleaving resource: From parallelism exploration to high efficiency architecture / C. Leroux, C. Jego, P. Adde [et al.] // Journal of Signal Processing Systems, Springer. – № 64 (1). – 2011. – P. 17 – 29.

5. Megha S. Iterative Decoding Algorithm for Turbo Product Codes / S. Megha // International Journal of Innovative Research and Advenced Engineering (IJIRAE). – April 2014. – Vol. 1, Issue 2. – P. 151 – 154.

6. Han J. High Speed Max-Log-MAP Turbo SISO Decoder Implementation Using Branch Metric Normalization / J. Han, E. Erdogan, T. Arslan // Proceedings of the IEEE Computer Society Annual Symposium on VLSI. – 2005. P. 173 – 178. DOI: 10.1109/ISVLSI.2005.37

7. Mathana J. FPGA Implementation of High Speed Architecture for Max Log Map Turbo SISO Decoder / J. Mathana, Dr. Rangarajan // International Journal of Recent Trends in Engineerign. – 2009. – Vol. 2, No. 6. – P. 142 – 146.

8. Krainyk Y. Low-complexity high-speed soft-hard decoding for turbo-product codes / Y. Krainyk, V. Perov, M. Musiyenko // 2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO). – 2017. – P. 471 – 474.

9. Hardware-Oriented Turbo-Product Codes Decoder Architecture / Y. Krainyk, V. Perov, M. Musiyenko [et al.] // Conference Proceedings of IEEE IDAACS-2017. – Bucharest, 2017. – P. 151 – 154.

Стаття надійшла до редакції 02.03.2018 р.
Стаття пройшла рецензування 16.03.2018 р.

Крайник Ярослав Михайлович – к. т. н., ст. викладач кафедри комп'ютерної інженерії, e-mail: codebreaker7@ukr.net, yaroslav.krainyk@chmnu.edu.ua.

Перов Владислав Олегович – аспірант кафедри комп'ютерної інженерії, e-mail: vlad.perov92@gmail.com.

Чорноморський національний університет імені Петра Могили.