УДК 004.312.26:519.725

Я. М. Крайник, *к.т.н., докторант, ст. викл. кафедри комп'ютерної інженерії*,
e-mail: codebreaker7@ukr.net, yaroslav.krainyk@chmnu.edu.ua
В. О. Перов, *аспірант кафедри комп'ютерної інженерії*,
e-mail: perov.vlad92@gmail.com
М. П. Мусієнко, *д.т.н., професор, професор кафедри комп'ютерної інженерії*
e-mail: musienko2001@ukr.net
Чорноморський національний університет імені Петра Могили
вул. 68 Десантників, 10, м. Миколаїв, Україна

МОДЕЛЮВАННЯ ПРОЦЕСУ ДЕКОДУВАННЯ TURBO-PRODUCT-КОДІВ ЗАСОБАМИ СЕРЕДОВИЩА MODELSIM

У статті представлено результати моделювання роботи розробленого декодера Turbo-Product-кодів (TP-кодів) у середовищі ModelSim. Описано методику проведення моделювання та особливості розробки тестових оточень для таких пристроїв. Декодер, для якого проводиться моделювання, характеризується тим, що він може змінювати налаштування відносно коду, яким закодоване повідомлення. Відповідно, він здатен працювати не з одним кодом, а з кількома та проводити їх декодування. Зміна налаштувань відносно оброблюваного коду може відбуватися після завершення обробки поточного повідомлення. Це дає значну перевагу, оскільки такий декодер може використовувати різні коди залежно від зміни умов передачі даних. Це дозволяє підвищити ефективність роботи системи передачі інформації та може мати суттєвий виграш відносно багатьох параметрів роботи декодера (коригуюча здатність, швидкодія, вартість розробки декодера для нових кодів та ін.). Розглянуто усі етапи роботи декодера з наведенням діаграм сигналів для кожного окремого етапу. Середовище ModelSim дає можливість наочно представити роботу кожного блока і, що важливо, взаємодію з іншими блоками, які входять до складу декодера. Це дало змогу оцінити як коректність роботи відносно стадій декодування, так і загальну картину роботи декодера. Тестове оточення, як і сам декодер, реалізоване мовою схемотехнічного опису VHDL. При роботі тестового оточення інтенсивно використовується робота з файлами (вхідні, вихідні та проміжні дані). Результат роботи тестового оточення порівнюється з результатом роботи моделі, реалізованої програмними засобами, для того щоб підтвердити правильність реалізації. Порівняння проводиться побайтово і показує однакові результати як для ModelSim, так і для програмної моделі. У подальшому реалізовані засоби можуть бути розширені для застосування при моделюванні декодерів для TP-кодів з іншою структурою або для інших завадостійких кодів.

***Ключові слова**: моделювання, тестове оточення, декодер, Turbo-Product-коди.*

Стадія моделювання роботи схемотехнічного опису пристрою є однією з найважливіших з точки зору перевірки коректності роботи схеми загалом. Незважаючи на те, що ця стадія проходить без безпосереднього застосування апаратних засобів, саме на основі її результатів приймається рішення про перехід до подальшої стадії тестування – лабораторного тестування. За умови коректної організації процесу моделювання у більшості випадків, апаратне тестування має підтверджувати результати, показані на стадії моделювання.

Відповідно, важливим є саме коректне налаштування тестового оточення (англ. testbench) [4] розробленого модуля. Тестове оточення зазвичай організовується за принципом, який характерний для організації модульних тестів, тобто:

1. Подача вхідних стимулів на вхід тестового модуля (англ. Design-Under-Test – DUT).
2. Запуск роботи модуля.
3. Перевірка збігу отриманих результатів з очікуваними.

Кількість операцій, які виконує обчислювальний модуль, також є важливою. Складні тестові модулі характеризуються великою кількістю операцій, які виконуються одночасно, а також великою кількістю проміжних етапів обчислень. Це, у свою чергу, означає підвищену складність тестового оточення для

такого модуля. Саме до модулів високого типу складності відноситься декодер Turbo-Product-кодів (ТР-кодів) [1] на базі програмовних логічних інтегральних схем (ПЛІС). Загалом, ці компоненти є складовими систем передачі даних, які розміщені на стороні отримувача. Проведення моделювання роботи таких компонентів вимагає подачі на вхід великої кількості даних, які фактично являють собою потік даних, що відповідає потоку даних у реальній системі передачі даних.

Загальна практика щодо побудови тестових оточень для таких модулів полягає в тому, що вхідні дані зберігаються у файлах, а результат обробки також записується у файл. Після цього проводиться перевірка на збіг отриманих результатів та результатів, які показала реалізована математична модель (зазвичай окремо реалізована програма, на вхід якої подаються дані з того самого файлу, що і для моделювання цифрового модуля). Для декодерів, які можуть оброблювати різні типи кодів, процедура доповнюється тим, що проводиться зміна налаштувань оброблюваних сигналів без перепрограмування пристрою. Саме таким є основний варіант використання цього декодера

У роботі представлено результати проведення моделювання роботи декодера ТР-кодів на базі ПЛІС. Описано методику організації тестового оточення для процесу моделювання, а також усі необхідні підготовчі кроки, які необхідні для повноцінного тестування системи. Також окремо виділено моменти, які характерні для моделювання декодера ТР-кодів з можливістю зміни оброблюваного коду без перепрограмування пристрою. Саме на зміні коду зі збереженням подальшої обробки даних робиться акцент при описі процесу загалом.

Огляд джерел. Середовище ModelSim [2] від компанії Mentor Graphics стало стандартом де-факто для процесу розробки цифрових схем на базі ПЛІС або спеціалізованих мікросхем. Однією з його переваг є те, що доступна безкоштовна версія програмного забезпечення, яке з певними обмеженнями можна використовувати для моделювання. Основні виробники ПЛІС (Xilinx та Altera) у власному програмному забезпеченні надають можливість обрати як середовище моделювання ModelSim, окрім власного програмного забезпечення, яке постачається з середовищами розробки. Моделювання саме у середовищі ModelSim дозволяє проводити тестування схемотехнічних описів, які мають бути незалежними від цільової мікросхеми, що також є великою перевагою. Однак підтримується також моделювання за наявності компонентів, які є специфічними для певної платформи. Поміж інших середовищ моделювання ModelSim виділяється підтримкою надзвичайно великої кількості функцій, пов’язаних з моделюванням (мови SystemC, SPL, SystemVerilog), підтримкою визначення покриття, а також перевірок різного типу. ModelSim широко використовується для моделювання систем різних типів [5, 6].

Методика створення тестових оточень передбачає активне використання операцій з файлами. У мові VHDL [3] наявні засоби для роботи з файлами, які покривають необхідний мінімум для запису та читання значень з відповідним форматуванням [7]. З цим пов’язані складнощі, які виникають при необхідності реалізувати складну послідовність запису/зчитування зі специфічним форматуванням. В основному використовується функціонал, який надається двома пакетами (std.textio та ieee.std_logic_textio). Основним моментом, який необхідно враховувати при реалізації запису/читання на базі файлів у тестових оточеннях, є те, що велика кількість процесів у схемотехнічному описі виконуються паралельно, тому необхідно коректно організувати виклик цих операцій для того, щоб отримати адекватні дані.

Ця робота є продовженням робіт [8-10] та базується на них. Метою роботи є опис методики проведення тестування, опис розробленого тестового оточення та представлення результатів моделювання, проведеного у середовищі ModelSim.

Основна частина. Процес зміни кодів, відповідно до яких проводиться обробка у декодері, характеризується зупинкою прийому нових вхідних даних та завантаженням у пам’ять нових параметрів налаштувань. Сигналом для цього може бути спеціальне повідомлення від передавача або припинення передачі зі зміною налаштувань оператором системи. У будь-якому випадку цей процес має бути узгоджений обома сторонами взаємодії. Для декодера це означає зміну налаштувань кодів рядків та стовпців. Цей параметр залежить від загальної кількості кодів, реалізованих у декодері. Шлях, який проходять дані між елементами комбінаційної логіки декодера, змінюється саме відповідно до даного налаштування. У більшості випадків

переключення означатиме подачу нового сигналу на вхід вибору мультиплексора з подальшою подачею даних на необхідний логічний компонент з набору. У такому наборі наявні декілька таких компонентів, які відповідають за обробку для кодів різної довжини. Подібні мультиплексори є на всьому шляху даних у схемотехнічній реалізації декодера.

Для запису даних та видачі результатів у декодер використовується шина даних, яка відповідає стандартній шині Avalon, зокрема, специфікація Avalon Memory-Mapped Interface. Таким чином, інтерфейс декодера є уніфікованим, і до нього можна проводити підключення зовнішніх пристроїв, які підтримують передачу даних за зазначеним стандартом. Для більшої універсальності опису мовою VHDL використовувались не готові Intellectual Property-ядра (IP-ядра), а інтерфейс був повністю описаний у вихідному коді проекту. Таким чином, реалізація цієї частини проекту не буде залежною від кінцевої мікросхеми. Моделювання запису даних зображено на діаграмі (рис. 1).

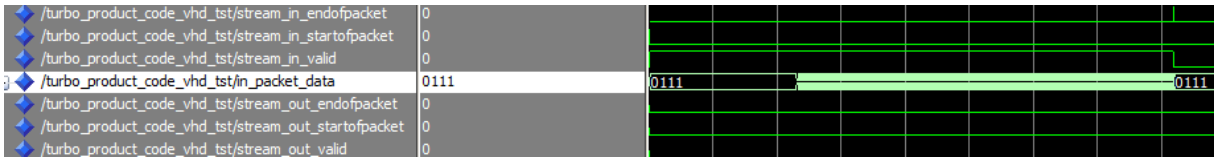

Рис. 1. Діаграма станів сигналів шини Avalon у процесі запису даних

На діаграмі видно, що певний проміжок часу шина даних є незмінною. Це відповідає запису даних у пам'ять для коду з довжинами складових, які не дорівнюють значенню степеня двійки, що обрані за базову довжину для груп кодів. У такому випадку необхідно заповнити пам'ять значеннями, які не впливатимуть на процес декодування. Після завершення декодування відбувається видача кінцевого масиву даних. Процес видачі декодованих даних у вигляді діаграми станів сигналів зображено на рис. 2.

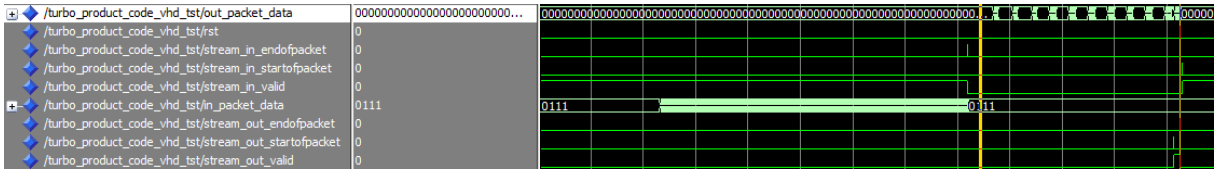

Рис. 2. Діаграма станів сигналів у процесі видачі даних

На діаграмі при порівнянні тривалості імпульсів видачі і запису видно, що видача відбувається набагато швидше, що пояснюється тим, що видача відбувається у паралельному форматі і для жорстких даних, тобто, для видачі повного пакета даних витрачається набагато менше часу, ніж на запис. Це, у свою чергу, гарантує те, що буфер буде звільнений до початку нової ітерації декодування з новим пакетом даних. Фактично видача даних при оцінюванні часових затрат у такому варіанті потребує навіть менше часу, ніж проведення однієї пів-ітерації, що однозначно дозволяє забезпечити вимогу щодо запису вхідних даних до закінчення видачі результату декодування. Варто також відзначити, що ширина шини даних при видачі відповідає довжині коду з максимальною довжиною кодового слова. Відповідно, на стороні, яка отримує оброблені дані, необхідно організувати відсічення частини даних, яка не відповідає довжинам складових кодів.

Основними моментами, які слід виділити у процесі безпосередньо декодування, є:

1. Пошук мінімальних позицій.
2. Генерація тестових векторів.
3. Отримання результатів жорсткого декодування.
4. Корекція м'яких значень.

Пошук мінімальних позицій у векторі є доволі складною операцією, яка потребує виконання великої кількості складних операцій. Логічні операції, які виконуються на цій стадії, також організовані у декілька стадій, через що наявна затримка між подачею вхідного вектора і отриманням результатів у вигляді трьох мінімальних позицій та значень у цих позиціях. Показовою є діаграма станів сигналів, представлена на рис. 3.

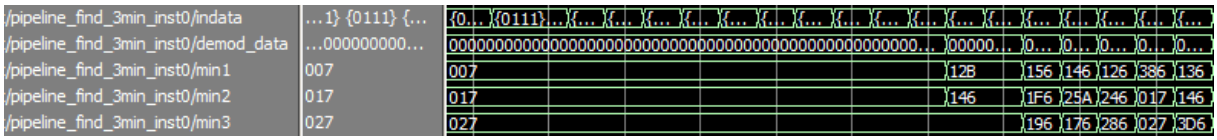

Рис. 3. Скріншот діаграми станів процесу пошуку позицій з мінімальними значеннями

З цієї діаграми добре видно, що проходження значень по внутрішній логічній мережі блока займає 12 тактів тактової частоти, на якій працює пристрій. Лише дані вказаної кількості тактів можуть бути передані до стадії генерації тестових векторів.

Процес генерації тестових векторів базується на зміні жорстких значень у знайдених позиціях у базовому векторі. Відбувається підстановка усіх можливих варіантів значень у ці позиції, тому один із векторів збігається з базовим вектором. На рис. 4 показано, як саме при моделюванні виглядають діаграми станів цих векторів.

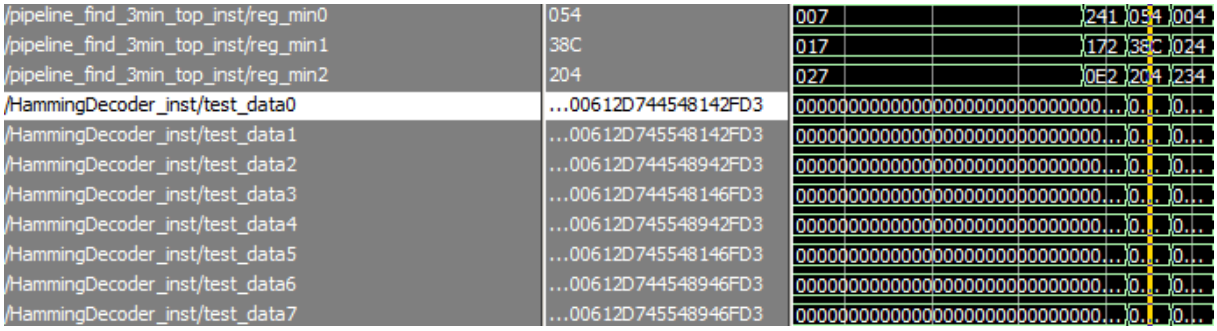

Рис. 4. Скріншот діаграми сигналів для генерації тестових векторів

Між подачею сигналів позицій та отриманням тестових векторів зі зміненими значеннями проходить лише один такт, тому ця стадія є більш швидкою порівняно з пошуком мінімумів. Як видно з рисунка, позиціями, в яких відбувається зміна для першого вектора, мають бути 0x24, 0x17, 0x0E (шістнадцяткове представлення). Курсор встановлений на наступному значенні виходу цього блока, тому легко переконатися, що відбулася зміна значень саме у необхідних позиціях (з метою зменшення місця, яке займає діаграма, тестові вектори так само наводяться у вигляді шістнадцяткових чисел).

Для кожного з тестових векторів обчислюється синдром. Ця операція обчислюється з використанням функціональності жорсткого декодера. Діаграму станів для цього процесу зображено на рис. 5.

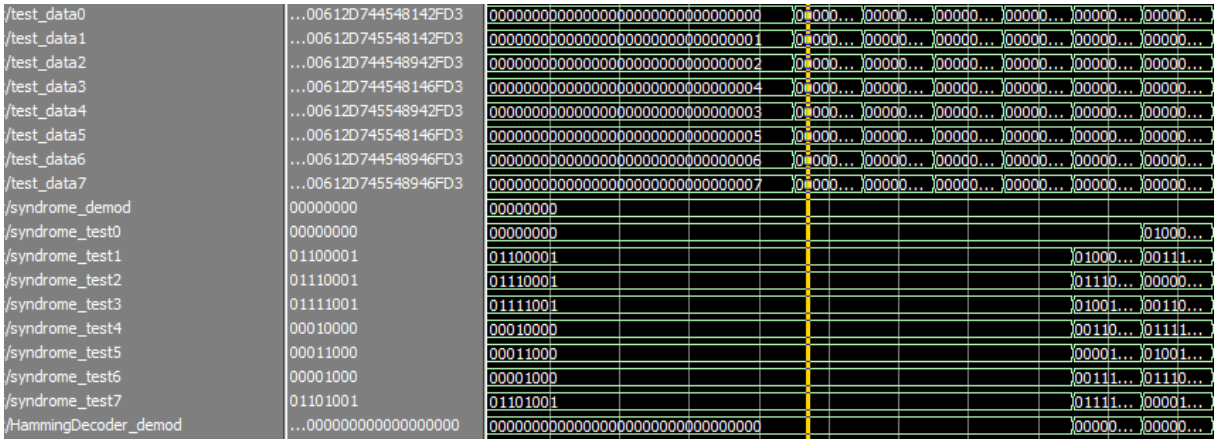

Рис. 5. Скріншот діаграми сигналів для стадії обчислення синдрому

Так само, як і попередньому випадку, затримка між видачею результату цього блока займає лише один такт. Це пояснюється тим, що на цій стадії використовуються прості бітові операції типу ВИКЛЮЧНЕ АБО, які застосовуються до всіх бітів вхідного повідомлення відповідно до обраного коду. Проте необхідною є організація затримки видачі для синхронізації цього блока з наступними та паралельними. Саме з цією метою кінцевий результат проходить через ланцюг регістрів. Вони забезпечують необхідну затримку у видачі результату.

Після цього етапу проходить корекція жорсткого вектора відповідно до значення синдрому. На цій стадії може бути інвертований один із бітів у тестових векторах або у вхідному векторі (рис. 6).

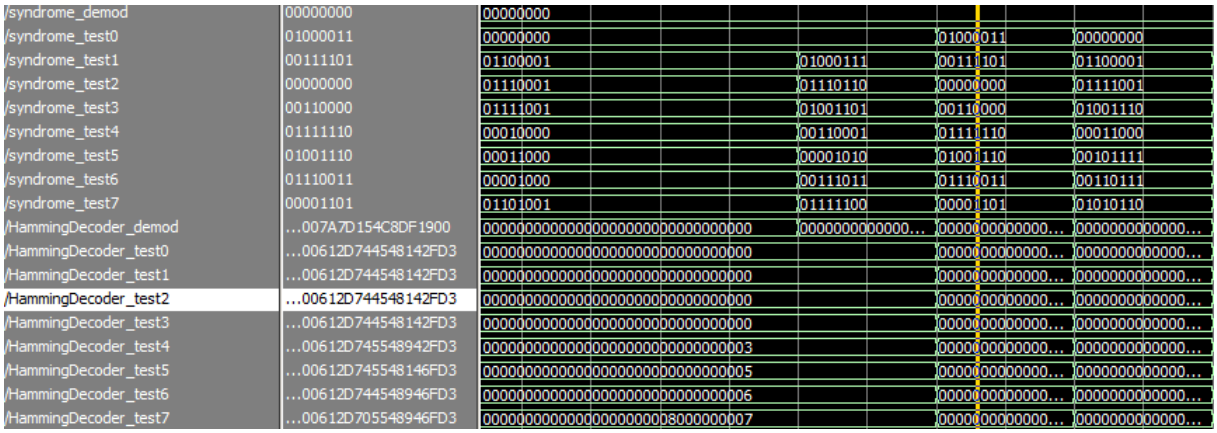

Рис. 6. Скріншот діаграми сигналів для процесу жорсткого декодування

Саме ці жорсткі вектори використовуються у подальшому для обчислення м'яких значень залежно від того, які значення синдромів були отримані.

На стадії корекції м'яких значень проводиться обробка відповідно до усіх даних, які може використовувати декодер (м'які вхідні значення, тестові вектори та їх синдроми та ін.). У результаті цієї стадії отримуються нові м'які значення, які є входом до наступної пів-ітерації декодера. Корекція відбувається шляхом:

1. Зміни знака м'якого значення.
2. Зміни амплітуди сигналу.

Саме ці дві операції характеризують цю стадію декодування у запропонованому декодері.

Діаграма станів основних сигналів при записі результуючих даних показана на рис. 7. За запис даних у пам'ять відповідальний окремий модуль, який, окрім безпосередньо самого запису, виконує також зсув даних з метою забезпечення подальшого коректного зчитування на наступній пів-ітерації.

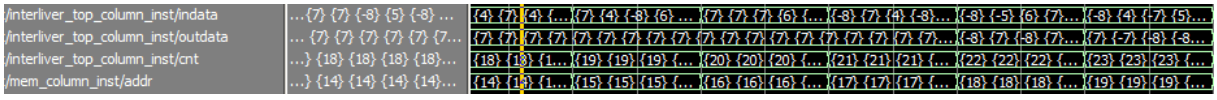

Рис. 7. Скріншот діаграми сигналів для процесу запису

У цьому випадку на діаграмі представлені масиви чисел, тому повністю продемонструвати значення сигналів не є можливим. Проте наявні сигнали cnt і addr, які вказують на те, як необхідно провести зсув даних і в які адреси пам'яті провести запис результуючих даних. Відповідно до наведеного рисунка затримка між отриманням даних на вході і записом їх у пам'ять становить 4 такти.

Також важливою характеристикою декодера є те, скільки тактів займає подача вхідних значень та початок видачі результату. Проходження даних через усі компоненти комбінаційної логіки займає суттєву кількість тактів. На рис. 8 зображено діаграму станів, яка демонструє вихід та вхід блока декодування.

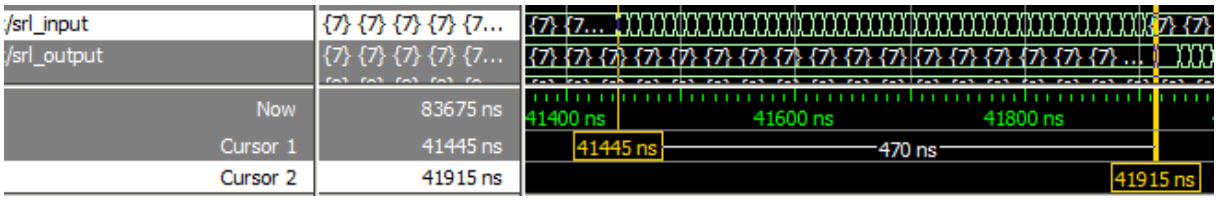

Рис. 8. Скріншот діаграми сигналів для входу і виходу даних

З цієї діаграми зрозуміло, що кількість тактів, які проходять після подачі першого вектора даних, становить 47 тактів (період роботи тактової частоти при моделюванні у цьому випадку заданий значенням 10 нс). Тобто, варто відзначити доволі суттєву затримку у видачі результату декодування вектора. З цим пов'язане і обмеження у максимальній швидкодії декодера, оскільки стадія проходження через конвеєр займає доволі велику кількість часу.

З приводу загальної перевірки роботи декодера і представлення його роботи під час моделювання декодування розділене на декілька стадій (обробка рядків і обробка стовпців), які чергуються між собою, таким чином формуючи ітерацію. На верхньому рівні керування декодером здійснюється скінченим автоматом, який подає необхідні керуючі імпульси на входи відповідних блоків та зупиняє операції відповідно до розмірності коду. Діаграму станів основних сигналів, якими керує автомат, наведено на рис. 9.

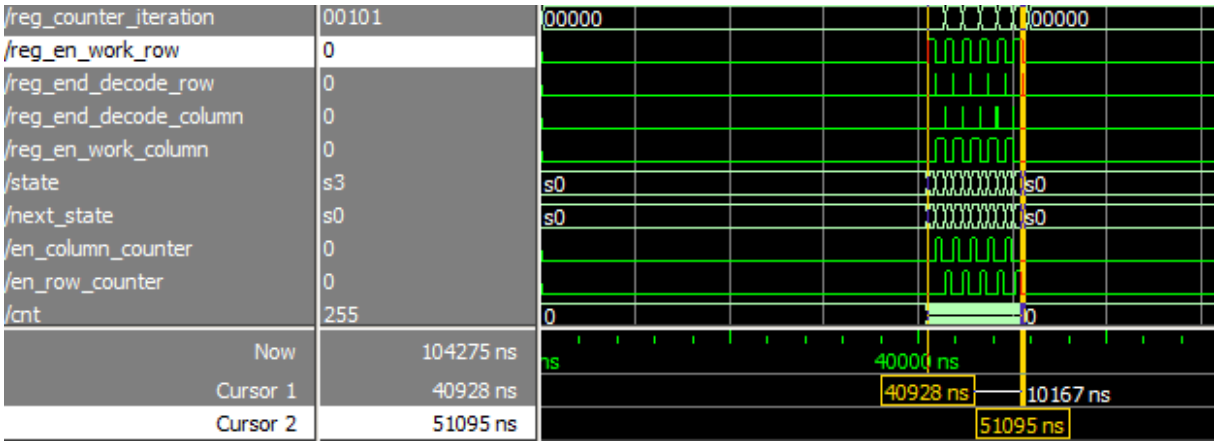

Рис. 9. Скріншот діаграми станів керуючих сигналів автомату

Як видно з рисунка, повний процес декодування займає значно більше часу, ніж очікування запису на попередньому. Це пов'язано з тим, що охоплюються всі операції декодування. Керуючий скінчений автомат працює на основі лічильника і відповідно до його значення визначається тривалість керуючих імпульсів. Лічильник постійно інкрементується з переведенням до нульового значення при досягненні кінцевого значення, яке залежить від довжини коду. Після обнулення лічильник рахує спочатку до заданого значення.

Висновки. У роботі представлено методику проведення моделювання розробленого схемотехнічного опису мовою VHDL декодера ТР-кодів. При моделюванні використовувались засоби середовища Mentor Graphics ModelSim. Розроблено спеціалізоване тестове оточення, яке дозволяє подавати дані на вхід декодера та отримувати результат з записом отриманих даних у файл. Особливістю розробленого тестового оточення є те, що воно дозволяє проводити декодування кількох кодів у рамках одного тестового запуску, забезпечуючи процес переключення між різними кодами, з якими може працювати декодер. Представлено діаграми сигналів основних процесів, які відбуваються під час декодування. Також проведено порівняння отриманих результуючих значень декодування з очікуваними, яке показало повний збіг розробленої математичної моделі та отриманих даних.

Список літератури

1. Berrou C., Glavieux A. Thitimajshima P. Near Shannon limit error-correcting coding and decoding: Turbo-codes. *IEEE ICC'93*. 1993. P. 1064–1071.
2. Model Sim Intel FPGA Edition Software. URL: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/model-sim.html (дата звернення: 20.10.2018). Назва з екрану.
3. Mealy B., Tappero F. Free range VHDL. Free range factory, 2011. 151 p.
4. VHDL testbench tutorual. URL: https://moodle.epfl.ch/pluginfile.php/1772382/mod_resource/content/3/vhdl_testbench_tutorial.pdf (дата звернення: 19.10.2018). Назва з екрану.
5. Li Y., Huo J., Li X., Wen J. et al. An open-loop Sin microstepping driver based on FPGA and the co-simulation of Modelsim and Simulink. *International Conference on Computer, Mechatronics, Control and Electronic Engineering, 24-26 Aug. 2010*. Changchun, China, 2010. P. 223–227. DOI: 10.1109/CMCE.2010.5609859
6. Kung Y.-S., Quynh N. V., Nguyen H. T. et al. Simulink/Modelsim co-simulation and FPGA realization of speed control IC for PMSM drive. *Procedia Engineering*. 2011. Vol. 23. P. 718–727. DOI: 10.1016/j.proeng.2011.11.2571
7. Savas E. VHDL Basic I/O. URL: http://people.sabanciuniv.edu/erkays/el310/io_10.pdf (дата звернення: 20.10.2018). Назва з екрану.
8. Krainyk Y., Perov V., Musiyenko M. Low-complexity high-speed soft-hard decoding for turbo-product codes. *IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO)*. Kyiv, 2017. P. 471–474.
9. Крайник Я. М., Перов В. О. Метод комбінованого декодування Turbo-Product-кодів для реалізації на базі FPGA. *Наукові праці:* наук. журн. Чорномор. нац. ун-ту ім. Петра Могили. Миколаїв, 2017. Т. 308. Вип. 296. С. 100–104.
10. Krainyk Y., Perov V., Musiyenko M., Davydenko Ye. Hardware-oriented Turbo-Product codes decoder architecture. *Conference Proceedings of IEEE Intelligent Data Acquisition and Advanced Computer Systems-2017*. Bucharest, 2017. P. 151–154.

References

1. Berrou, C., Glavieux, A., Thitimajshima, P. (1993) Near Shannon limit error-correcting coding and decoding: Turbo-codes. *IEEE ICC'93*, pp. 1064–1071.
2. Intel.com (2018) Model Sim Intel FPGA Edition Software. URL: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/model-sim.html [Accessed 20.10.2018].
3. Mealy, B., Tappero, F. (2011). Free range VHDL. Free range factory, 151 p.
4. Moodle.epfl.ch (2017) VHDL testbench tutorual. URL: https://moodle.epfl.ch/ pluginfile.php/1772382/mod_resource/ content/3/vhdl_testbench_tutorial.pdf [Accessed 19.10.2018].
5. Li, Y., Huo, J., Li, X., Wen, J. et al. (2010) An open-loop Sin microstepping driver based on FPGA and the co-simulation of Modelsim and Simulink. In the: *International Conference on Computer, Mechatronics, Control and Electronic Engineering*, 24-26 Aug., Changchun, China, pp. 223–227.
6. Kung, Y.-S., Quynh, N. V., Nguyen, H. T et al. (2011) Simulink/Modelsim co-simulation and FPGA realization of speed control IC for PMSM drive. *Procedia Engineering*, Vol. 23, pp. 718–727. DOI: 10.1016/j.proeng.2011.11.2571
7. Savas, E. VHDL Basic I/O URL: http://people.sabanciuniv.edu/erkays/el310/io_10.pdf [Accessed 20.10.2018].
8. Krainyk, Y., Perov, V., Musiyenko, M. (2017) Low-complexity high-speed soft-hard decoding for turbo-product codes. In the: *IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO)*. Kyiv, pp. 471–474.
9. Krainyk, Ya. M., Perov, V. O. (2017) Method of combined decoding of Turbo-Product-codes for the implementation in FPGA. *Naukovi pratsi:* sci. journal of Petro Mohyla Black Sea national un-ty, Vol. 308, iss. 296, pp. 100–104 [in Ukrainian].
10. Krainyk, Y., Perov, V., Musiyenko, M., Davydenko, Ye. (2017) Hardware-oriented Turbo-Product codes decoder architecture. In the: *Conference Proceedings of IEEE Intelligent Data Acquisition and Advanced Computer Systems-2017*. Bucharest, pp. 151–154.

Ya. M. Krainyk, *Ph.D., doctoral student, senior lecturer,*
e-mail: codebreaker7@ukr.net, yaroslav.krainyk@chmnu.edu.ua
V. O. Perov, *post-graduate,*
e-mail: perov.vlad92@gmail.com
M. P. Musiyenko, *D.Tech.S., professor*
e-mail: musienko2001@ukr.net
Petro Mohyla Black Sea National University
68 Desantnykiv str., 10, Mykolaiv, Ukraine

MODELING OF TURBO-PRODUCT-CODES DECODING PROCESS USING MODELSIM ENVIRONMENT

In the article the results of modeling of developed Turbo-Product-codes (TP-codes) decoder in ModelSim environment have been presented. The method of modeling and implementation of test-benches for this type of devices are described. The decoder under modeling has a feature of changing the code to process the message immediately during the work state. Correspondingly, it is able to work not just with one code but also with a set of codes. Configuration change can be performed after the end of decoding current encoded message. It provides a great advantage as such decoder can utilize different codes according to external conditions concerned with signal transmission. It allows to improve the efficiency of information transmission system and can eventually make a notable gain for many decoder's working parameters (correction ability, throughput, cost of development, etc.). All the stages of decoder work are thoroughly considered and presented with wave diagrams for every single stage. ModelSim environment ensures the possibility of demonstration of each separate block, as well as the whole system and intercommunication among blocks. It makes possible to assess the correctness of decoding stages and general decoding results. Testbench and decoder itself have been implemented in VHDL description language. During the work of testbench file operations are used intensively (input, output, and intermediate data). The result of testbench work is compared with the result of software model to prove the correctness of implementation. The comparison is performed byte-to-byte and demonstrates the same results both for ModelSim and software model. In future, implemented means can be extended for using in decoders modeling for TP-codes with different structure and for other types of error-correcting codes.

Keywords: *modeling, testbench, decoder, Turbo-Product-codes.*